



UNIVERSITÀ
DEGLI STUDI
DI PADOVA



DIPARTIMENTO
DI INGEGNERIA
DELL'INFORMAZIONE

UNIVERSITY OF PADUA

DEPARTMENT OF INFORMATION ENGINEERING
MASTER'S DEGREE IN COMPUTER ENGINEERING

Graph-Based Representation Learning for Assembly State Recognition Using Synthetic-Driven Part Detection

SUPERVISOR

Prof. Stefano Ghidoni
University of Padua

CANDIDATE

Mattia Toffolon
ID: 2104286

CO-SUPERVISOR

Prof. Matteo Terreran
University of Padua

ACADEMIC YEAR 2025/2026
GRADUATION DATE: 14/04/2026

*To my parents
and friends*

Contents

Abstract	xiii
Sommario	xv
1 Introduction	1
1.1 Assembly State Recognition	2
1.1.a Challenges	2
1.2 Contribution	3
1.3 Thesis outline	4
2 Related Works	7
2.1 Assembly State Recognition	7
2.2 Representation Learning	13
2.3 Graphs and GNNs in Computer Vision	15
3 Background	17
3.1 Representation Learning	17
3.2 Object Detectors	19
3.3 Graph Neural Networks	21
3.4 Vision Transformers	23
4 Representation Learning for Assembly State Recognition	27
4.1 IndustReal Dataset	27
4.2 Baseline Approach	32
4.2.a Image Feature Projection Model	33
4.2.b Intermediate-State Informed Loss	34
4.2.c Limitations	36
5 Assembled Parts Graph Embeddings for ASR	37
5.1 Assembled Parts Detection	38
5.2 Graph Generation	39
5.3 Graph Embedder Model	42
5.4 Multimodal Approach	45
5.4.a Naive Fusion	46
5.4.b Gated Fusion	47

6	Synthetic Dataset Generation Pipeline	51
6.1	Assets Setup	52
6.2	Image Rendering	53
6.2.a	Light Sources	54
6.2.b	Camera Poses	54
6.2.c	Camera Parameters	57
6.3	Dataset Organization	58
7	Sim-to-Real Adaptation	63
7.1	Closing the Sim-to-Real Gap	64
7.2	Detections Processing	66
8	Object Detectors Evaluation	69
8.1	Evaluation of Part Detector	71
8.2	Evaluation of Assembled Object Detector	75
8.2.a	Class Agnostic Assembled Object Detector	80
8.3	Assembled Object Detector for ASR	81
9	Assembled Parts Graph Embeddings Evaluation	85
9.1	Performance on IndustReal Test Set	88
9.2	Generalization to Unseen States	92
9.3	Generalization to Unseen Error States	95
10	Conclusions	99
10.1	Future Works	100
	Bibliography	103

List of Figures

1.1	Example of a simplified industrial assembly procedure. Starting from a set of parts, the final assembly is obtained by sequentially reaching a predefined series of valid assembly states.	2
2.1	Comparison between Assembly State Classification (ASC) and Assembly State Recognition (ASR). Top: ASC paradigm, where predictions are directly obtained by mapping the input sample to a predefined set of classes without the use of reference samples. Bottom: ASR paradigm, where predictions are inferred through similarity comparisons between the query sample and a set of labeled reference samples in a learned representation space. The figure also highlights the limitation of classification-based approaches in handling high intra-class variability, which is mitigated by representation learning.	9
2.2	Error-state generation framework adopted in [4]. The Assembly Graph of the dataset is constructed by connecting well-defined states and incorporating possible error states (a). A subgraph is reserved for evaluation (b). Error-state samples are automatically generated for each well-defined state (c).	10
2.3	SimCLR framework. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement.	13
2.4	Overview of the method proposed in [38]. Two kinds of the graphs are constructed based on the object detections, hand joints and gaze in the view. The two kinds of graphs are processed via two stream Graph Convolutional Network to recognize the label of the action.	16
3.1	Contrastive loss training using the triplet loss framework. Here, X denotes the anchor, X^+ the positive sample, X^- the negative sample and f represents the encoding function.	19

3.2	Example of object detector inference results. The model identifies relevant objects by generating bounding boxes with associated labels and each prediction is assigned an overall confidence score.	20
3.3	Comparison of node feature aggregation in Graph Convolutional Network (GCN) and Graph Attention Network (GAT) layers. In GCN, aggregation treats all neighbors uniformly, whereas in GAT, aggregation is weighted by attention coefficients computed from the features of both the target node and its neighbors, enabling more informative and adaptive feature aggregation.	23
3.4	Vision Transformer (ViT) model architecture with an example of image classification inference (left) and the corresponding Transformer Encoder architecture (right).	24
4.1	Overview of the construction-toy car's parts, components and one of the assembly states used in IndustReal. The maintenance task represents a component upgrade and consists of replacing the long braces with short braces on the rear chassis.	28
4.2	Example frame from the IndustReal dataset, showing the typical scene with both assembled components and loose parts resulting from the progression of the assembly procedure.	29
4.3	Samples from a clip in the IndustReal dataset, demonstrating the modalities and annotations for all three tasks. Gaze is indicated by the cross, detected hand joints by the dots. AR: action recognition, ASD: assembly state detection, PSR: procedure step recognition.	30
4.4	Error categorization with visual examples and the number of frames in IndustReal for each category. Depending on whether the error occurs at the component level or the part level, mistakes are grouped into four distinct categories.	31
4.5	Scheme depicting the overall Assembly State Recognition framework based on Representation Learning developed in [2] and adopted for this thesis work.	32
4.6	Baseline approach encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning.)	34
4.7	Illustration of influence of the proposed loss function that leverages images containing intermediate assembly states, i.e. non-defined, transitional states between pre-defined states.	35
5.1	Novel GNN-based approach encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning; snowflake icon and blue color indicate components trained separately.)	38

5.2	Example of graph generated from assembled part detections. Nodes represent individual parts, edges encode structural relationships and the accompanying numerical values illustrate examples of crafted node and edge features.	40
5.3	<i>Graph Embedder</i> model architecture. A custom GNN-based model for part detections graph feature extraction.	43
5.4	Example illustrating the intermediate–real state labeling issue in the dataset. Both frames depict the same real assembly state (13); however, the first is labeled as <i>intermediate</i> solely due to the presence of the operator’s hands in the frame.	45
5.5	Multimodal approach with <i>naive</i> input fusion encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning; snowflake icon and blue color indicate components trained separately.)	47
5.6	<i>Gated</i> input fusion module enabling learnable modulation of modality contributions to better exploit multimodal information.	48
5.7	Multimodal approach with <i>gated</i> input fusion approach encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning; snowflake icon and blue color indicate components trained separately.)	49
6.1	Synthetic dataset generation pipeline. Starting from a set of 3D models representing assembly states, the system automatically generates data by instantiating a scene with lighting and virtual cameras. For each camera, images and raw annotations are rendered and subsequently post-processed to obtain clean object- and part-level bounding boxes. The final dataset is produced by aggregating and splitting the generated samples according to user-defined parameters.	52
6.2	Visualization of the camera pose parameterization strategy. The camera position is defined in the xy plane through radius and azimuth, with height z , while orientation is determined by a look-at target point and an in-plane rotation (roll).	55
6.3	Beta probability density functions for the values of α and β adopted for camera pose generation.	56
6.4	Examples of images obtained through the implemented synthetic dataset generation pipeline. The adopted filename format for rendered images encodes the depicted assembly state (e.g., <code>state16_inter13</code>), the specific asset for that state and rendering chunk (e.g., <code>1b</code> , <code>1f</code> , <code>1k</code> , <code>2c</code>), and a unique image identifier.	59

6.5	Example of BlenderProc-produced annotations post-processing performed to obtain assembled object and parts bounding boxes.	61
7.1	Comparison of real and synthetic images showing assembly state 11 from a similar viewpoint. The main difference is the presence of loose parts in the real image.	65
7.2	Comparison of part detection on a real image before and after closing the synthetic-to-real gap. Before adaptation, many false positives are produced (e.g., detections on the operator's hands and duplicate predictions) along with false negatives (e.g., a missed <i>long brace</i>). After adaptation, these errors are largely mitigated, resulting in cleaner and more accurate detections.	66
7.3	Example of step-by-step part prediction filtering procedure on a real image.	67
8.1	Part detector mAP_{50-95} performance (per class and overall) on a fixed synthetic test set, shown for different training dataset sizes.	72
8.2	Part detector mAP_{50-95} performance (on all parts and on assembled ones only) on a fixed real test set, shown for different synthetic training dataset sizes.	74
8.3	Part detector mAP_{50-95} performance (on all parts and on assembled ones only) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.	75
8.4	Assembled object detector mAP_{50-95} performance (per class and overall) on a fixed synthetic test set, shown for different training dataset sizes.	77
8.5	Assembled object detector mAP_{50} performance (per class and overall) on a fixed real test set, shown for different training dataset sizes.	79
8.6	Assembled object detector mAP_{50} performance (per class and overall) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset	80
9.1	UMAP visualization of the embedding space learned by the four Assembly State Recognition models on the IndustReal training set. Intermediate states are depicted as gray crosses, and defined states as colored dots according to the legend.	90
9.2	UMAP visualization of the embedding space generated by the four pre-trained Assembly State Recognition models from synthetic images depicting unseen assembly states. Embeddings corresponding to the novel assembly states are represented as colored dots according to the legend.	94
9.3	Examples of synthetic assembly state reference images generated for the model robustness to unseen errors test.	96

List of Tables

2.1	Comparative overview of assembly-related datasets, highlighting acquisition setup, annotation types, error modeling and overall scale.	12
5.1	Official performance comparison of YOLO11 models on the COCO dataset.	39
6.1	Sampling ranges and distribution parameters for camera pose generation. Values for v_{min} and v_{max} are expressed in meters. . .	57
7.1	Overall mAP_{50-95} performance of object detectors trained exclusively on synthetic data, evaluated on synthetic and real test sets. The substantial performance drop on real data highlights the presence of the sim-to-real gap.	64
8.1	Part detector mAP_{50-95} per class and overall performance on a fixed synthetic test set for training dataset sizes 25K, 50K and 100K.	73
8.2	Part detector mAP_{50-95} per class and overall performance (all parts) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.	76
8.3	Part detector mAP_{50-95} per class and overall performance (assembled parts only) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset. . . .	76
8.4	Assembled object detector overall mAP_{50-95} results on a fixed synthetic test set for training dataset sizes from 5K to 100K. . . .	78
8.5	Assembled object detector overall mAP_{50} results on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.	80
8.6	Generic assembled object detectors Recall ₅₀ results on a fixed real test set.	81
8.7	Assembled object detector performance, in terms of precision, recall and F1-score, on the IndustReal test set when used as image classifier.	82

9.1	Model performance on Assembly State Recognition using the IndustReal dataset test set, compared across multiple evaluation metrics. The best results are shown in bold, and the second-best results are underlined.	89
9.2	Model performance on Assembly State Recognition evaluated on additional synthetic images depicting assembly states unseen during training, compared across multiple evaluation metrics. The best results are highlighted in bold, and the second-best results are underlined.	93
9.3	Model performance on Assembly State Recognition in the robustness-to-errors binary verification test on the IndustReal test set, focusing on erroneous assembly states unseen during training. Per-class mean Average Precision (mAP) is reported first, followed by aggregated metrics micro-averaged mAP , macro-averaged mAP and area under the ROC curve ($auROC$). Best results are shown in bold and second-best underlined. (Legend: Cat I = missing component errors; Cat II = incorrect component orientation; Cat III = incorrect component placement; Cat IV = part-level errors.)	96

Abstract

Assembly processes in industrial environments are prone to errors that can reduce efficiency and increase costs. Automatic recognition of the current assembly state from visual data can support operators by providing timely feedback and reducing mistakes. An effective Assembly State Recognition system should not only achieve high accuracy on known states, but also generalize to unseen states, remain scalable as parts change, and be robust to execution errors.

This thesis addresses Assembly State Recognition using the IndustReal dataset, which consists of egocentric videos of operators assembling and maintaining a toy vehicle. Initial experiments show that naive solutions, such as training a task-specific object detector, fail to meet the required generalization and scalability properties. The problem is therefore approached from a representation learning perspective, where video frames are mapped into an embedding space using contrastive learning.

While prior work relies on generic visual features extracted directly from video frames as input to the embedding model, such representations lack explicit information about the object’s structural composition, which limits their discriminative properties. The main contribution of this thesis is a structure-aware representation that explicitly captures part-level assembly information. Two custom object detection models are trained to identify individual parts and the fully assembled object, enabling the filtering of unassembled parts. To ensure scalability and avoid costly manual annotation, a synthetic data generation pipeline is introduced, which produces realistic and balanced training data directly from 3D models of the assembly states.

The resulting part detections are used to construct graphs that encode the assembly structure depicted in each frame. These graphs are processed by a custom Graph Neural Network-based model to produce graph embeddings, which act as a substitute for the image-based visual features. In addition, multi-modal approaches are investigated, exploring how graph-based and visual features can be combined to leverage the complementary strengths of both representations.

Experimental results demonstrate that the proposed graph-based representation encodes meaningful and discriminative information, leading to a consistent improvement in Assembly State Recognition performance. Notably, the approach yields a significant gain in generalization capabilities, while remaining robust to errors and naturally scalable to new assembly configurations.

Sommario

I processi di assemblaggio in ambienti industriali sono soggetti a errori che possono ridurre l'efficienza e aumentare i costi. Il riconoscimento automatico dello stato di assemblaggio (Assembly State Recognition) a partire da dati visivi può supportare gli operatori fornendo feedback tempestivo e riducendo gli errori. Un sistema efficace di Assembly State Recognition dovrebbe non solo ottenere elevate prestazioni sugli stati noti, ma anche generalizzare a stati di assemblaggio non visti, rimanere scalabile al variare dei componenti ed essere robusto a errori di esecuzione.

Questa tesi affronta il problema dell'Assembly State Recognition utilizzando il dataset IndustReal, composto da video egocentrici di operatori impegnati nell'assemblaggio e nella manutenzione di un veicolo giocattolo. Esperimenti preliminari mostrano che soluzioni semplici, come l'addestramento di un object detector specifico per il task, non soddisfano i requisiti di generalizzazione e scalabilità. Il problema viene quindi affrontato da una prospettiva di representation learning, in cui i frame video vengono mappati in uno embedding space tramite contrastive learning.

Mentre lavori precedenti si basano su feature visive estratte dai frame video come input per il modello di embedding, tali rappresentazioni mancano di informazioni esplicite sulla composizione strutturale degli oggetti, limitandone le proprietà discriminative. Il principale contributo di questa tesi è una rappresentazione structure-aware che cattura esplicitamente informazioni di assemblaggio a livello di componente. Vengono addestrati due object detection model per identificare le singole componenti e l'oggetto completamente assemblato, consentendo il filtraggio delle componenti non assemblate. Per garantire scalabilità ed evitare costose annotazioni manuali, viene introdotta una pipeline di generazione di dati sintetici, che produce dati di training realistici e bilanciati a partire dai modelli 3D degli stati di assemblaggio.

Le detection delle parti vengono utilizzate per costruire grafi che codificano la struttura dell'assemblaggio rappresentata in ciascun frame. Tali grafi sono elaborati da un modello basato su Graph Neural Networks per produrre graph embeddings, che fungono da sostituto delle feature visive image-based. Inoltre, vengono investigate strategie multimodali, esplorando come le feature graph-based e visive possano essere combinate per sfruttare i punti di forza di entrambe.

I risultati sperimentali dimostrano che la rappresentazione graph-based proposta codifica informazioni significative e discriminative, portando a un miglioramento consistente delle prestazioni nell'Assembly State Recognition. In particolare, l'approccio mostra un notevole incremento nelle capacità di generalizzazione, mantenendo robustezza agli errori e scalabilità verso nuove configurazioni di assemblaggio.

Introduction

Procedural activities consist of predefined sequences of actions that, when executed precisely and in the correct order, enable the accomplishment of complex goals. In industrial contexts, procedures typically correspond to a sequence of assembly states that an object must pass through in order to be correctly assembled from a given set of parts, as illustrated in Figure 1.1, which depicts a simplified example of such a process.

In such settings, errors during the execution of a procedure can have significant consequences. Missing a part or performing an incorrect operation at any step may compromise the correctness of the entire assembly process. Critically, such errors are not always immediately detectable and may only become evident in later stages, or even at the end of the procedure. In these cases, the assembly often needs to be partially or completely redone, resulting in substantial additional effort, increased material consumption and higher economic costs. Moreover, this scenario places additional pressure on operators, contributing to increased stress and reduced overall productivity.

To address these challenges, the method proposed in this thesis aims to provide continuous support during the assembly process. Specifically, it enables real-time monitoring, by identifying at each step the current configuration of the assembly, and feedback, by informing operators whether the configuration corresponds to a valid procedural state or if deviations have occurred.

This approach is designed to improve operational efficiency, reduce cognitive load on workers and minimize the likelihood of execution errors. By detecting mistakes early in the process, it helps prevent costly rework and material waste, ultimately lowering operational costs. Therefore, the proposed method is relevant not only from a technical perspective but also from a practical and economic standpoint.

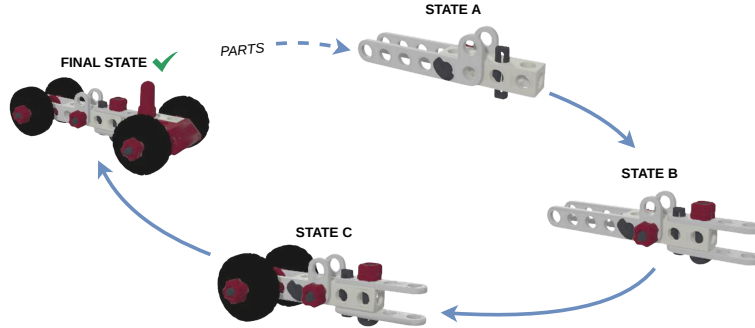


Figure 1.1: Example of a simplified industrial assembly procedure. Starting from a set of parts, the final assembly is obtained by sequentially reaching a predefined series of valid assembly states.

1.1 Assembly State Recognition

Assembly State Recognition (ASR) is the task of automatically identifying the current assembly state of an object during the execution of a procedure. As assembly procedures are typically defined as a sequence of discrete steps, each associated with a well-defined assembly state, the objective of ASR is to accurately classify the observed configuration, determining whether it precisely corresponds to one of the predefined states or if it does not belong to any of them.

1.1.a Challenges

Although conceptually straightforward, ASR is inherently challenging. Each assembly state is defined by a specific and often fine-grained composition of parts. Even minimal differences, such as a single misplaced part, can correspond to a different state. This makes the recognition task highly sensitive to subtle structural variations.

Moreover, strong performance on predefined assembly states alone is not sufficient for an optimal ASR solution. In industrial environments, assembly procedures are subject to change: states and parts may be updated, replaced or newly introduced. A purely classification-based approach, which assumes a fixed set of known labels, is therefore unsuitable. Any modification in the set of possible states would require costly data annotation and model retraining, severely limiting scalability. Instead, a robust ASR system should rely on approaches that support generalization to previously unseen assembly states.

Finally, robustness to execution errors is a fundamental requirement. The

system should be able to recognize when the current configuration does not correspond to a valid procedural step. However, enumerating all possible erroneous states is infeasible. Consequently, an effective ASR solution must detect errors without explicitly relying on error-state examples during training, ensuring the capability to identify previously unseen mistakes.

1.2 Contribution

In this thesis, the ASR problem was addressed with the objective of developing a solution capable of satisfying all the desired properties previously discussed: (i) accuracy on known states, (ii) generalization to unseen ones, (iii) robustness to errors and (iv) scalability.

To develop and evaluate the proposed approaches, the publicly available IndustReal dataset [1] was selected, as it provides all the characteristics required to thoroughly assess ASR solutions under these aspects. The dataset was created by recording operators through a mixed reality headset while assembling or performing maintenance on a toy vehicle according to a predefined procedure, resulting in egocentric video data.

IndustReal was chosen because it offers frame-level assembly state annotations, specifying whether a given frame corresponds to a well-defined assembly state and, if so, which one. This enables supervised training and evaluation on seen states. Furthermore, operators were allowed to make mistakes during the assembly process, and error states are explicitly labeled in the dataset. This makes it possible to assess model robustness to execution errors. Additionally, since the toy vehicle was specifically designed for the dataset, 3D models of all assembly states are provided. This allows the generation of alternative assembly configurations and synthetic images, supporting the evaluation of generalization and scalability.

Building upon recent research in ASR, this thesis first confirms that classification based approaches, such as fine-tuning an object detector to directly classify video frames, are inadequate, particularly with respect to generalization to unseen states (ii) and robustness to errors (iii). Consequently, the focus is shifted toward a Representation Learning framework. Through contrastive learning, such approaches aim to learn an embedding space in which assembly states can be inferred via similarity to reference representations rather than direct classification. This paradigm removes the intrinsic limitations of fixed-label classification and offers the potential for improved generalization and error detection.

While prior works have already adopted representation learning for ASR, they rely primarily on image-based feature extractors to generate embeddings.

Although effective to some extent, purely visual features may lack explicit information about the structural composition of the assembled object. As a result, their discriminative power for fine-grained assembly reasoning may be limited, achieving suboptimal accuracy on both known (i) and unseen (ii) configurations, and exhibiting unsatisfactory robustness to errors (iii).

To address this limitation and provide a solution that advances the state of the art with respect to the previously discussed critical aspects (i–iii), the main contribution of this thesis is the introduction of a novel structure-aware representation that explicitly captures part-level assembly information. To obtain this representation, a detection module was developed, composed of two object detectors, one for single parts and one for the assembled object, along with a dedicated filtering algorithm. This system enables the extraction, for each video frame, of the set of assembled parts composing the object, thereby providing raw structural information about the current assembly state.

Particular attention was devoted to ensuring scalability (iv). The detection models are trained primarily on synthetic data generated through a newly implemented synthetic dataset generation pipeline. This pipeline produces realistic and diverse training samples, automatically labeled at both part and assembled object level, directly from provided 3D assembly models, making it straightforward to generate new data whenever parts or states change.

The extracted part detections are then organized into a graph-based representation that encodes both the presence of parts and their spatial relationships, effectively modeling the assembly structure depicted in the frame. To process these graphs, an ad-hoc Graph Neural Network (GNN) based model architecture was designed to generate embeddings that can be mapped into the representation space, effectively replacing purely image-based features.

Finally, this work also investigates multimodal approaches, exploring how graph-based and image-based features can be combined to leverage the complementary strengths of structural and visual information.

1.3 Thesis outline

The remainder of this thesis is structured as follows.

Chapter 2 reviews related literature, focusing on computer vision approaches that address ASR or other tasks with similar objectives, highlighting similarities and differences with respect to the proposed framework. Chapter 3 briefly introduces the fundamental concepts required to properly understand the methodologies and design choices adopted in this work. Chapter 4 presents in detail the dataset used in this thesis and describes the current state of research in ASR

upon which this work builds. Chapter 5 introduces the proposed graph-based representation, details its construction process, presents the GNN-based model and explains the investigated multimodal approaches. Chapter 6 illustrates the design and implementation of the synthetic dataset generation pipeline. Chapter 7 describes the development of the assembled parts detection system. Chapters 8 and 9 report the experimental evaluations: the former focuses on the performance assessment of the detection system, while the latter evaluates the proposed ASR solutions across all relevant aspects, including accuracy, generalization and robustness to errors. Finally, Chapter 10 summarizes the main contributions of this thesis and outlines possible directions for future research.

Related Works

This thesis proposes a novel approach to the Assembly State Recognition task based on representation learning and a Graph Neural Network model that operates on custom constructed assembled parts graphs.

Although this approach is not yet explored in the literature, several related tasks and key components of the proposed method have been studied in prior works. Analyzing these contributions is important to understand their strengths and limitations and to better position the proposed approach with respect to existing solutions.

First, the Assembly State Recognition task and closely related, more extensively studied problems are introduced, together with a review of relevant works. Additionally, commonly used public datasets for their evaluation are presented. Then, key contributions in the field of representation learning are introduced. Finally, the use of graphs and Graph Neural Networks in computer vision tasks is reviewed, highlighting how structured representations are commonly designed and processed in similar domains.

2.1 Assembly State Recognition

Assembly State Recognition (ASR) is the task of identifying the current state of an assembly by leveraging auxiliary labeled reference samples, from which predictions are inferred through similarity measures computed between the query data and the reference set.

The primary contribution in the literature addressing this task is [2]. In this work, the proposed method is grounded in representation learning, with the objective of learning a feature space in which similarity between samples can be directly inferred from their corresponding representations. The developed

model is composed of an image encoder coupled with a multi-layer perceptron (MLP), which jointly extract visual features and project them into a latent embedding space. This approach, further detailed in Chapter 4.2, provides enhanced generalization capabilities and robustness to previously unseen assembly errors.

ASR remains a relatively underexplored problem in the literature, as no other works explicitly address the identification of assembly states from this perspective. Nevertheless, the broader problem of assembly state identification has been extensively studied, even if under slightly different formulations.

The most common formulation is Assembly State Classification (ASC), which consists of assigning an input sample to one of a predefined set of assembly state classes. In contrast to ASR, ASC relies solely on the model and the query input, without exploiting any auxiliary reference samples at inference time. Figure 2.1 illustrates the conceptual differences between the two paradigms, particularly in terms of their prediction mechanisms. It also highlights a key limitation of classification-based approaches, discussed in greater detail in subsequent chapters, namely, their difficulty in effectively modeling classes characterized by high intra-class variability. This limitation can be mitigated by adopting a representation learning framework, as done in ASR.

Another related formulation, which has received more attention than ASR, is Assembly State Detection (ASD). This task extends ASC by requiring not only the classification of the assembly state but also its spatial localization within the input, typically through bounding box prediction in the case of image data.

In the following, existing works addressing ASC and ASD are reviewed in order to identify common methodological approaches to assembly state identification, despite their differing underlying assumptions compared to ASR.

Starting from ASD, a natural and widely adopted solution consists in training an object detection model. Such models are explicitly designed to jointly localize and classify objects of interest within an image, making them well-suited for addressing the ASD task. This is the approach adopted in [1], where a YOLOv8 object detector is trained to detect assemblies and classify them into 22 well-defined real states, along with a generic error state. To improve performance, the use of synthetically generated data and different training strategies is investigated.

In [3], this approach is further extended by introducing an assembly state graph encoding the feasible transitions between states. By injecting the previous state into the neck of the object detector, the model is guided to learn valid state transitions, effectively reducing the prediction space and simplifying the inference process. This line of work is further developed in [4], where robustness to error states is explicitly addressed. Given the vast number of possible error

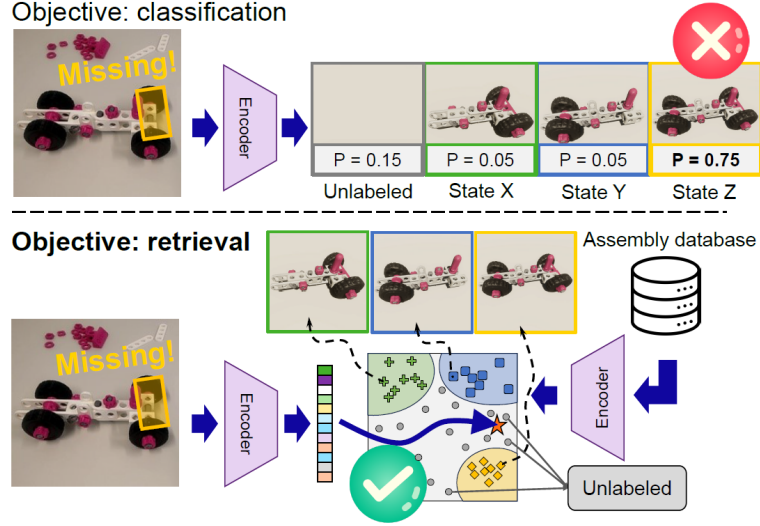


Figure 2.1: Comparison between Assembly State Classification (ASC) and Assembly State Recognition (ASR). Top: ASC paradigm, where predictions are directly obtained by mapping the input sample to a predefined set of classes without the use of reference samples. Bottom: ASR paradigm, where predictions are inferred through similarity comparisons between the query sample and a set of labeled reference samples in a learned representation space. The figure also highlights the limitation of classification-based approaches in handling high intra-class variability, which is mitigated by representation learning.

configurations, a pipeline is proposed to automatically generate 3D models of erroneous assemblies by perturbing correct configurations through small part displacements. These models are then used to render synthetic images for training the detector. Figure 2.2 illustrates this approach together with a few examples of the generated error states.

Despite their effectiveness, these approaches remain relatively limited in expressiveness at inference time, as they rely on object detection to identify and classify the assembled object as a whole, without explicitly modeling its internal compositional structure. Consequently, they fail to capture fine-grained relationships among individual parts. In contrast, several works shift the focus toward part-level reasoning, explicitly incorporating information about individual parts into the model input, as also proposed in this thesis. These methods typically address the ASC task and can be broadly categorized into two groups: (i) approaches based on manually designed part-level knowledge and (ii) approaches that rely on the automatic learning of part relevance.

For the former category, one approach is presented in [5], where the observed object is back-projected into 3D space, aligned to a canonical configuration and

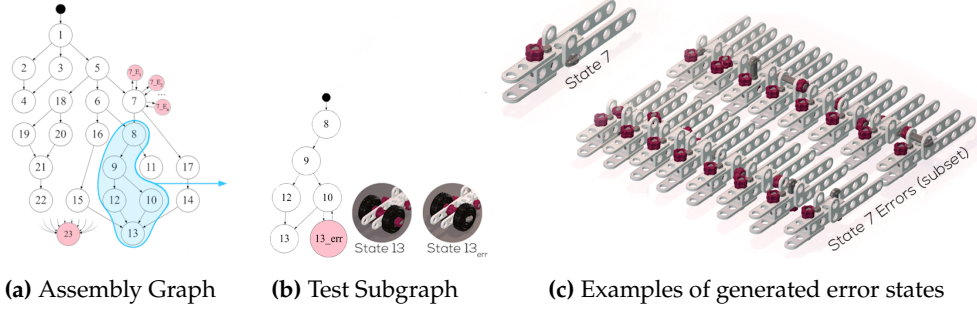


Figure 2.2: Error-state generation framework adopted in [4]. The Assembly Graph of the dataset is constructed by connecting well-defined states and incorporating possible error states (a). A subgraph is reserved for evaluation (b). Error-state samples are automatically generated for each well-defined state (c).

voxelized. By leveraging prior knowledge of part locations, the occupancy of corresponding voxels is used to infer the assembly state. A more refined strategy is based on 6D part pose estimation, as explored in [6, 7, 8]. In these works, part poses are estimated either by detecting keypoints in the image and solving a Perspective-n-Point (PnP) problem, or via direct template matching between observed and reference point clouds. Once the pose of each part is obtained, the assembly state is inferred by verifying geometric constraints on relative transformations, typically through thresholds on translations and rotations. A different perspective is proposed in [9], where the focus shifts from assembled parts to unassembled ones. Parts are recognized via template matching using synthetic images and when a part approaches the assembly, it is snapped to the closest valid mounting point in 3D space, producing the resulting configuration.

Focusing on the second category, which comprises more complex methods relying on the automatic learning of part relevance, Region of Interest (RoI) proposal mechanisms are integrated into the model to identify informative regions for assembly state classification. The design of the input features, as well as the strategy for exploiting the extracted regions, varies across different approaches. In [10, 11], multi-scale features are extracted from the image and either directly used or enriched via cross-correlation between feature channels before being fed into the RoI proposal module. The resulting RoI features are then used for state inference. In [12], RoIs are instead determined offline on a voxelized 3D model by identifying the most relevant regions through Grad-CAM analysis on rendered images. During inference, both image and point cloud data are used to extract features from these regions, which are then processed by an LSTM to predict the assembly state.

A different line of work, not directly targeting ASC or ASD but particularly

effective in capturing error states, is presented in [13]. In this approach, an input image is compared with a reference image depicting a correct assembly state. Cross-correlation between feature representations enables the generation of a binary segmentation mask highlighting discrepancies, effectively localizing errors. While this method is capable of identifying a wide range of errors, it does not explicitly provide a discrete classification of assembly states.

By analyzing common patterns and frequently adopted components in existing methods, several recurring design choices can be identified. Object detectors consistently serve as a reliable foundation for localizing and labeling relevant regions within images, which can then be exploited as an initial processing stage in downstream models. In the context of identifying assembly states, relevant regions may correspond to semantically meaningful parts of the assembly or to the individual components themselves. Furthermore, industrially manufactured objects are typically designed through the development of 3D CAD models, which can also be leveraged to generate synthetic data. This enables the creation of large-scale, automatically annotated datasets on which models can be effectively and reliably trained.

Regarding data usage, a notable trend is that many existing works rely on proprietary datasets, which limits reproducibility and hinders fair comparison across approaches, ultimately slowing research progress. Nevertheless, several publicly available datasets for industrial-like assembly processes have been introduced and can support the development and evaluation of ASR, ASC, and ASD methods. These datasets differ in acquisition setup, annotation granularity, and treatment of procedural correctness and errors. The most relevant ones are briefly reviewed in the following and a comparative summary of them is reported in Table 2.1.

MECCANO [14] focuses on the assembly of a toy motorbike using Meccano components in an industrial-like tabletop environment. Data is acquired from an egocentric perspective through a custom headset, and annotations include temporal action segments, object and hand bounding boxes, as well as egocentric human–object interactions. Although minor procedural mistakes may occur during recording, they are typically corrected immediately and are not explicitly annotated as persistent error states.

IKEA ASM [15] considers the assembly of real furniture across both laboratory and domestic environments. It provides multi-view third-person recordings enriched with detailed annotations, including atomic actions, object instance segmentation masks, and human poses in both 2D and 3D. However, the dataset focuses exclusively on correct task execution and does not include annotations for errors or incorrect assembly steps.

Dataset	Year	Views	Error Annotation	Scale
MECCANO [14]	2021/22	Egocentric	No (implicit)	~7h / 20 seq.
IKEA ASM [15]	2021	Multi-view (3rd)	No	~35h / 371 vids
Assembly101 [16]	2022	Multi-view + Ego	Yes	513h / 4321 vids
HA-VID [17]	2023	Multi-view (3rd)	Yes	86.9h / 3222 vids
IndustReal [1]	2023	Egocentric	Yes	5.8h / 84 seq.

Table 2.1: Comparative overview of assembly-related datasets, highlighting acquisition setup, annotation types, error modeling and overall scale.

Assembly101 [16] significantly increases both the scale and variability of available data, featuring unscripted assembly and disassembly of 101 toy vehicles. It combines egocentric and multi-view third-person recordings and provides both coarse- and fine-grained temporal annotations, along with 3D hand poses. Importantly, it explicitly captures and annotates procedural errors and their subsequent corrections, making it particularly suitable for studying error recovery.

HA-VID [17] introduces a controlled industrial-like environment based on a custom assembly box composed of standardized components. It provides rich annotations, including subject–action–object relationships, spatial bounding boxes and collaboration patterns. A key characteristic of this dataset is the explicit labeling of incorrect actions and pauses, enabling the analysis of both execution errors and cognitive aspects during task learning.

IndustReal [1], which is adopted in prior ASR [2] and ASD [1, 4] works and in this thesis, is specifically designed for industrial assembly scenarios, including both assembly and maintenance operations. Data is acquired from an egocentric perspective and includes annotations for action recognition, assembly state detection and procedural steps. Notably, it explicitly models both procedural and execution errors, making it particularly suitable for evaluating robustness to incorrect configurations.

In summary, ASR represents a promising yet underexplored paradigm for identifying assembly states, offering increased flexibility compared to traditional classification-based approaches. Existing methods based on ASC and ASD provide effective solutions within constrained settings, but inherently limit their predictions to a fixed set of predefined states, thus preventing generalization to unseen configurations. Moreover, they exhibit limited robustness to errors, particularly those not observed during training. These limitations motivate the adoption of representation learning and part-based reasoning, which constitute the foundation of the approach proposed in this thesis.

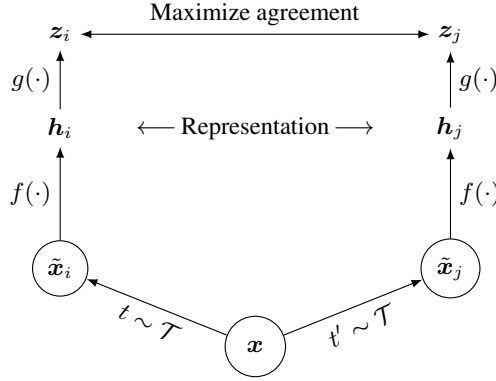


Figure 2.3: SimCLR framework. Two separate data augmentation operators are sampled from the same family of augmentations ($t \sim \mathcal{T}$ and $t' \sim \mathcal{T}$) and applied to each data example to obtain two correlated views. A base encoder network $f(\cdot)$ and a projection head $g(\cdot)$ are trained to maximize agreement.

2.2 Representation Learning

Representation learning aims to map raw data into a compact and semantically meaningful embedding space, where relevant relationships between samples are preserved. Following this paradigm, representations are learned directly from data, such that similar samples are mapped close together, while dissimilar ones are pushed apart.

A common approach to learn such embeddings is contrastive learning, which relies on comparing samples during training. By defining positive (related) and negative (unrelated) pairs, contrastive loss functions optimize the relative distances in the embedding space. This enables the learning of discriminative representations that generalize effectively.

In literature, similarity between representations has been defined through data augmentations: multiple transformed views of the same image, obtained for example through cropping, color jittering or rotation, are treated as positive pairs, while all other samples act as negatives. Frameworks such as SimCLR [18, 19] show that carefully designed augmentations, together with large batch sizes and projection heads, enable the learning of highly discriminative representations without requiring manual annotations. This can be interpreted as maximizing agreement between different views of the same instance while enforcing invariance to nuisance factors. The SimCLR framework is illustrated in Figure 2.3.

Another fundamental architecture for representation learning is the Siamese Neural Network (SNN), originally introduced for signature verification [20].

SNNs consist of multiple identical subnetworks with shared weights, each processing a different input. Their outputs are compared through a distance metric, such as Euclidean or cosine distance, to estimate similarity. This structure enforces consistent feature extraction and allows the model to learn a similarity function rather than explicit class boundaries. Subsequent works extended this paradigm to more complex tasks, including face verification [21] and one-shot recognition [22], where only a single example per class is available.

To effectively train such models, several loss functions have been proposed to structure the embedding space. A widely used formulation is the triplet loss [23], previously introduced in Section 3.1. However, this function in its basic form has a major issue: the number of possible triplets grows cubically with dataset size, making naive sampling inefficient, as many triplets already satisfy the constraint and do not contribute to learning. To mitigate this, hard mining strategies are adopted. In particular, the Batch Hard triplet loss [24] selects the most challenging positive and negative samples within each mini-batch, thus focusing the optimization on informative examples and accelerating convergence.

An alternative formulation is the InfoNCE loss [25], which casts the problem as a similarity-based classification task. Given an anchor, the model must identify its corresponding positive among a set of negatives. This approach scales efficiently with batch size and forms the basis of modern self-supervised methods such as SimCLR [18, 19]. The supervised contrastive (SupCon) loss [26] extends this idea by leveraging label information, allowing multiple positives per anchor (i.e., all samples belonging to the same class in the mini-batch), which results in improved intra-class compactness.

A key advantage of representation learning is its ability to generalize beyond the training distribution. Unlike traditional classification or detection models, which are optimized to separate a fixed set of classes, representation learning focuses on encoding similarity relationships between samples [21, 22]. Consequently, the learned embeddings capture transferable features that can generalize to new instances and even unseen classes, making them particularly suitable for one-shot and few-shot scenarios.

In the context of Assembly State Recognition, such generalization is exactly what is sought. Learning a robust embedding space, rather than rigid class boundaries, enables the model to handle previously unseen assembly configurations and to identify deviations from well-defined states, thereby improving robustness to errors. Representation learning therefore provides a principled and scalable framework for addressing the challenges of Assembly State Recognition.

2.3 Graphs and GNNs in Computer Vision

Graph structures provide a powerful abstraction for modeling entities and their relationships, while Graph Neural Networks (GNNs) enable reasoning over such structured representations. For this reason, these concepts, introduced here and discussed in greater detail in Section 3.3, have become a well-established paradigm in machine learning and computer vision.

In recent years, graphs and GNNs have been successfully applied to a variety of vision tasks, ranging from classical problems such as object detection and semantic segmentation to more complex ones like multi-label classification and action recognition. A common perspective across these works is to reinterpret visual data in terms of entities (nodes) and their relationships (edges), enabling relational reasoning beyond standard convolutional approaches.

In multi-label image classification, the goal is to predict multiple labels for a single image, often spanning different levels of semantic granularity. While traditional methods treat each label independently, several works exploit dependencies between labels through graph structures. In [27] and [28], labels are modeled as nodes in a graph, with edges encoding semantic similarity derived from WordNet embeddings. A GNN-like architecture, implemented via recurrent models, propagates information across labels and the resulting features are combined with global image features to improve prediction. Similarly, [29] integrates both label co-occurrence statistics and label embeddings, refining label-specific features through graph propagation before classification. These approaches demonstrate how structured label relationships can enhance classification performance.

Graph-based reasoning has also been explored in object detection, where the task consists of localizing and classifying objects within an image. Standard detectors process region proposals independently, ignoring contextual relationships. To address this limitation, [30] constructs a graph over detected regions, where nodes represent proposals and edges encode visual similarity. Through message passing, region features are refined using contextual information from related regions, improving classification accuracy. This idea is extended in [31] by incorporating an additional “transfer graph” that connects regions to external label spaces, enabling the integration of knowledge from other domains and facilitating transfer learning.

In semantic segmentation, where pixel-wise classification is required, graph structures are used to model spatial dependencies. In [32], images are represented as graphs with pixels as nodes and edges connecting spatially neighboring pixels, allowing feature refinement via Graph Convolutional Networks (GCNs).

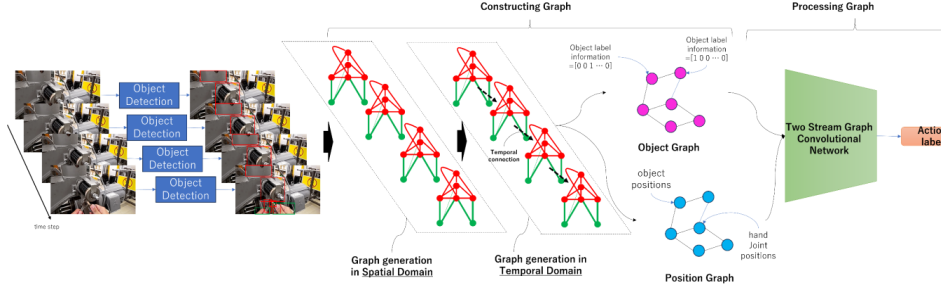


Figure 2.4: Overview of the method proposed in [38]. Two kinds of the graphs are constructed based on the object detections, hand joints and gaze in the view. The two kinds of graphs are processed via two stream Graph Convolutional Network to recognize the label of the action.

Building on this, [33] introduces label-specific graphs to enforce consistency among pixels assigned to the same class, further improving segmentation quality. Graph-based approaches have also been extended to 3D data, such as point clouds, where nodes represent points enriched with geometric and appearance features and edges capture local neighborhoods. GNN-based encoder-decoder architectures, as in [34], enable effective feature aggregation aligned with the underlying 3D structure and subsequent point classification leading to semantic segmentation of the point cloud.

Graphs are particularly effective in tasks requiring high-level reasoning over interactions, such as action recognition. In these scenarios, objects or agents are modeled as nodes and their spatial and temporal relationships are encoded as edges. Works such as [35] and [36] construct spatio-temporal graphs over detected entities across video frames, refining node features through GNNs before performing classification. More advanced formulations, such as [37], employ dynamic graphs where edge weights are adaptively learned, allowing the model to capture context-dependent interactions. Similarly, [38] combines multiple graph structures (e.g., object-centric and motion-based) to model complementary aspects of the scene, demonstrating the flexibility of graph-based representations. The latter approach is summarized by the scheme illustrated in Figure 2.4.

These works share a common principle: leveraging graph structures to model dependencies between entities and using GNNs to learn from these relationships for downstream prediction tasks. This paradigm is closely related to Assembly State Recognition, where the target label depends on the presence, identity and spatial configuration of multiple parts. As in the aforementioned tasks, modeling parts as nodes and their interactions as edges provides a natural and expressive representation. Therefore, the approach proposed in this thesis is inspired by these works.

Background

This chapter introduces the theoretical foundations underlying the proposed method and formalizes the key concepts discussed throughout the thesis. It provides a concise yet comprehensive overview of the essential topics required for a proper understanding of the developed approach.

The chapter begins with representation learning, which constitutes the basis for extracting meaningful features from data. It then reviews object detection models, which aim at localizing and classifying relevant objects in visual inputs. Subsequently, graph neural networks are presented as a framework for learning representations from graph-structured data, followed by an overview of transformer-based architectures and their extension to visual data.

3.1 Representation Learning

The concept of representation learning originates from the broader objective of machine learning to automatically extract meaningful patterns from raw data. Traditionally, feature engineering required domain expertise to design descriptors tailored to specific tasks. However, with the rise of data-driven approaches, it became evident that models could instead learn such representations directly from data, often outperforming handcrafted features. Early developments in this direction can be traced back to unsupervised learning methods such as autoencoders and manifold learning techniques, which aimed to capture the underlying structure of data distributions [39]. With the advent of deep learning, representation learning has become a central paradigm, enabling models to learn hierarchical and increasingly abstract features.

Formally, representation learning seeks to map input data $x \in \mathcal{X}$ into a latent

space $\mathcal{Z}$ through a function f_θ parameterized by θ :

$$z = f_\theta(x), \quad z \in \mathcal{Z} \quad (3.1)$$

The objective is to construct a representation z that preserves relevant information for downstream tasks while discarding irrelevant variability. Ideally, this latent representation should exhibit properties such as invariance to nuisance factors, discriminability between different classes or states, and compactness.

One of the most influential paradigms for learning such representations is contrastive learning. The key idea is to learn representations by comparing samples and enforcing similarity between related inputs while pushing apart unrelated ones. Typically, this is achieved by defining positive pairs (same category data points) and negative pairs (different categories data points). The learning objective encourages the model to minimize the distance between representations of positive pairs while maximizing the distance for negative pairs. A common formulation is based on similarity functions such as cosine similarity:

$$\text{sim}(z_i, z_j) = \frac{z_i^\top z_j}{\|z_i\| \|z_j\|} \quad (3.2)$$

This approach has proven highly effective, particularly in scenarios where labeled data is scarce, as it enables self-supervised learning of robust and transferable features [18].

A fundamental component of contrastive learning frameworks is the choice of loss function. Among the most widely used is the triplet loss, which operates on triplets of samples: an anchor x_a , a positive sample x_p and a negative sample x_n . The goal is to ensure that the distance between the anchor and positive representations is smaller than that between the anchor and negative representations by at least a margin α , as visualized by Figure 3.1. This can be expressed as:

$$\mathcal{L}_{\text{triplet}} = \max(0, d(f_\theta(x_a), f_\theta(x_p)) - d(f_\theta(x_a), f_\theta(x_n)) + \alpha) \quad (3.3)$$

where $d(\cdot, \cdot)$ is a distance metric, typically the Euclidean distance. The margin α enforces a minimum separation, preventing trivial solutions and improving generalization.

In summary, representation learning provides a principled framework for mapping raw data into structured and informative embeddings. Through approaches such as contrastive learning and loss functions like the triplet loss, models learn to capture meaningful relationships within the data. These embeddings form the basis for a wide range of downstream tasks, enabling improved performance and generalization.

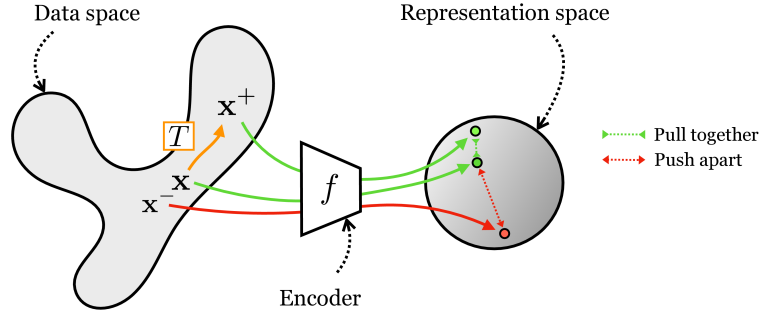


Figure 3.1: Contrastive loss training using the triplet loss framework. Here, X denotes the anchor, X^+ the positive sample, X^- the negative sample and f represents the encoding function.

3.2 Object Detectors

Object detection arises from the need to not only recognize the presence of entities within data but also to localize them. While early machine learning approaches focused on classification tasks, practical applications quickly highlighted the importance of identifying where relevant elements are located. Classical methods relied on handcrafted features combined with sliding window strategies, which were computationally expensive and limited in generalization. The introduction of deep learning significantly transformed this paradigm, enabling end-to-end trainable models that jointly learn feature extraction and localization [40].

Formally, object detection can be defined as the task of predicting a set of objects within an input x , typically represented as bounding boxes and associated class labels along with a prediction score, as shown in the example of Figure 3.2. Let $\mathcal{Y} = \{(b_i, c_i)\}_{i=1}^N$ denote the set of ground truth objects, where b_i represents the bounding box coordinates and c_i the corresponding class. A detector is a function f_θ that outputs a set of predictions:

$$\hat{\mathcal{Y}} = f_\theta(x) = \{(\hat{b}_j, \hat{c}_j, s_j)\}_{j=1}^M \quad (3.4)$$

where $\hat{b}_j$ are predicted bounding boxes, $\hat{c}_j$ class labels and s_j confidence scores. The objective is to minimize the discrepancy between $\mathcal{Y}$ and $\hat{\mathcal{Y}}$, accounting for both localization and classification accuracy.

Modern object detectors are generally structured around three main components: a backbone network for feature extraction, a neck for feature aggregation and a head for prediction. The backbone, often based on convolutional or transformer architectures, extracts hierarchical representations. The neck aggregates multi-scale features to improve detection of objects of varying sizes, while the



Figure 3.2: Example of object detector inference results. The model identifies relevant objects by generating bounding boxes with associated labels and each prediction is assigned an overall confidence score.

head produces bounding box coordinates and class probabilities.

Training such models involves optimizing a multi-task loss that combines classification and localization objectives. A common formulation is:

$$\mathcal{L} = \lambda_{\text{cls}} \mathcal{L}_{\text{cls}} + \lambda_{\text{loc}} \mathcal{L}_{\text{loc}} \quad (3.5)$$

where $\mathcal{L}_{\text{cls}}$ is typically a cross-entropy loss for classification and $\mathcal{L}_{\text{loc}}$ measures the discrepancy between predicted and ground truth bounding boxes. The latter is often based on Intersection over Union (IoU):

$$\text{IoU}(b, \hat{b}) = \frac{|b \cap \hat{b}|}{|b \cup \hat{b}|} \quad (3.6)$$

The combination of these losses allows the model to jointly learn accurate localization and robust classification.

Among modern object detection approaches, the YOLO (You Only Look Once) family represents one of the most widely adopted solutions, particularly in applications requiring real-time performance. Introduced by Redmon et al. [41], YOLO formulates object detection as a single-stage regression problem, directly predicting bounding boxes and class probabilities from the input image in a single forward pass. This design enables fast inference while maintaining competitive accuracy.

The core idea of YOLO is to process the entire image holistically, allowing the model to capture global contextual information while simultaneously localizing and classifying objects. Over successive iterations, the YOLO family has evolved to incorporate architectural improvements, including more efficient backbone

networks, multi-scale feature extraction, and refined training strategies. Recent versions, such as the YOLO11 family, further improve performance through enhanced feature representation, modular design and better generalization capabilities.

Due to their efficiency and scalability, YOLO models are particularly well-suited for industrial applications, where real-time processing and robustness are critical. Their ability to operate with high throughput while maintaining reliable detection performance makes them a practical choice for tasks involving continuous monitoring and automation.

In this work, multiple YOLO-based detectors are employed as core components of the proposed approach to perform part-level and assembled-object-level detection. Their balance between accuracy and computational efficiency makes them well-suited for integration within the pipeline, enabling reliable extraction of structured information from visual data.

3.3 Graph Neural Networks

The increasing need to model relational and structured data has motivated the development of learning paradigms beyond traditional grid-like representations. Many real-world problems are naturally expressed as graphs, where entities interact through complex and often irregular relationships. Classical machine learning approaches struggled to effectively capture such dependencies, as they typically assume independent and identically distributed (i.i.d.) data. The emergence of Graph Neural Networks (GNNs) addressed this limitation by enabling end-to-end learning directly on graph-structured data [42, 43].

Formally, a graph is defined as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ the set of edges. Each node $i \in \mathcal{V}$ is associated with a feature vector $h_i^{(0)} \in \mathbb{R}^d$. The goal of a GNN is to learn node (or graph-level) representations by iteratively aggregating information from neighboring nodes. At layer l , the representation of node i is updated as:

$$h_i^{(l+1)} = \sigma \left(\text{AGG} \left(\{h_j^{(l)} : j \in \mathcal{N}(i)\} \right) \right) \quad (3.7)$$

where $\mathcal{N}(i)$ denotes the neighborhood of node i , $\text{AGG}(\cdot)$ is a permutation-invariant aggregation function and σ is a non-linear activation. Through multiple layers, nodes progressively incorporate information from larger receptive fields, enabling the model to capture both local and global graph structure.

A prominent and widely adopted class of GNNs is Graph Convolutional Networks (GCNs). Introduced as a spectral approximation of graph convolutions,

GCNs simplify the aggregation process by applying a normalized linear transformation over neighboring nodes [43]. The layer-wise propagation rule can be written as:

$$H^{(l+1)} = \sigma \left(\tilde{D}^{-1/2} \tilde{A} \tilde{D}^{-1/2} H^{(l)} W^{(l)} \right) \quad (3.8)$$

where $\tilde{A} = A + I$ is the adjacency matrix with added self-loops, $\tilde{D}$ is the corresponding degree matrix, $H^{(l)}$ is the matrix of node representations at layer l and $W^{(l)}$ are learnable weights. The inclusion of self-loops ensures that each node retains its own features during aggregation. GCNs are computationally efficient and effective in capturing smoothness assumptions over the graph, meaning that connected nodes tend to have similar representations. However, their fixed aggregation scheme can limit expressiveness, especially in heterogeneous or highly dynamic graphs.

To address these limitations, Graph Attention Networks (GATs) introduce an attention mechanism that allows nodes to weigh the importance of their neighbors adaptively [44]. Instead of treating all neighbors equally, GATs compute attention coefficients that quantify the relevance of each neighbor during aggregation. This concept is highlighted in Figure 3.3, which compares GCN and GAT node feature aggregation schemes, and explained in detail by the following equation. For a pair of nodes i and j , the attention score is computed as:

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(a^\top [Wh_i \parallel Wh_j]))}{\sum_{k \in \mathcal{N}(i)} \exp(\text{LeakyReLU}(a^\top [Wh_i \parallel Wh_k]))} \quad (3.9)$$

where W is a learnable linear transformation, a is a weight vector and $\parallel$ denotes concatenation. The node update then becomes:

$$h_i^{(l+1)} = \sigma \left(\sum_{j \in \mathcal{N}(i)} \alpha_{ij} Wh_j^{(l)} \right) \quad (3.10)$$

Self-loops are typically included, allowing nodes to attend to their own features. Additionally, multi-head attention is often employed to stabilize training and improve representational capacity.

The key advantage of GATs lies in their flexibility: by learning attention weights, the model can focus on the most informative neighbors, effectively handling graphs with varying connectivity and noise. This makes them particularly suitable for scenarios where relationships between nodes are not equally informative or may change dynamically.

In summary, Graph Neural Networks provide a powerful framework for learning from structured data by leveraging relational inductive biases. While GCNs

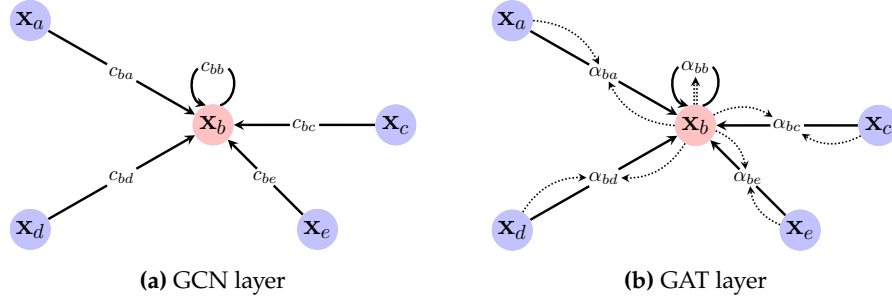


Figure 3.3: Comparison of node feature aggregation in Graph Convolutional Network (GCN) and Graph Attention Network (GAT) layers. In GCN, aggregation treats all neighbors uniformly, whereas in GAT, aggregation is weighted by attention coefficients computed from the features of both the target node and its neighbors, enabling more informative and adaptive feature aggregation.

offer a simple and efficient baseline through uniform aggregation, GATs extend this paradigm by introducing adaptive, attention-based weighting mechanisms, resulting in improved expressiveness and robustness.

3.4 Vision Transformers

The introduction of Transformers marked a significant shift in the design of deep learning architectures, particularly in sequence modeling tasks. Originally proposed for natural language processing, Transformers replaced recurrent and convolutional structures with attention mechanisms, enabling more efficient modeling of long-range dependencies [45]. The key innovation lies in the self-attention mechanism, which allows each element of an input sequence to dynamically attend to all other elements, capturing global contextual relationships without relying on sequential processing.

Formally, given an input sequence represented as a set of vectors $X = [x_1, x_2, \dots, x_n]$, the Transformer computes queries (Q), keys (K) and values (V) through learned linear projections. The self-attention operation is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V \quad (3.11)$$

where d_k is the dimensionality of the key vectors. This formulation enables the model to weigh the importance of each element in relation to others, producing context-aware representations. Multi-head attention extends this idea by allowing multiple attention mechanisms to operate in parallel, improving expressiveness.

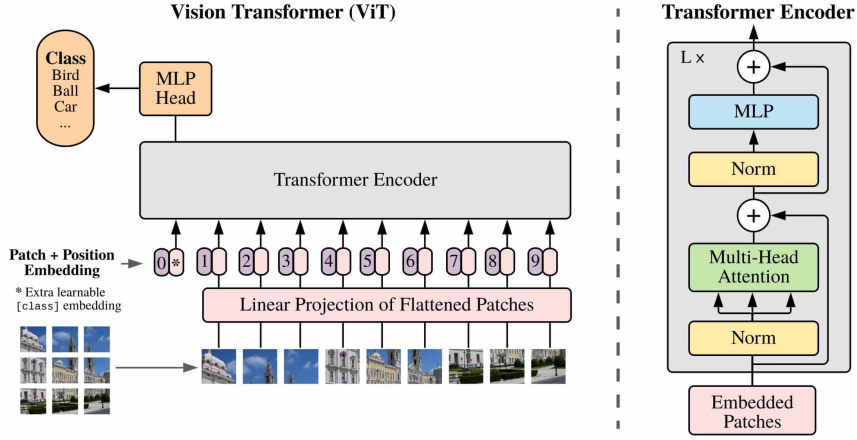


Figure 3.4: Vision Transformer (ViT) model architecture with an example of image classification inference (left) and the corresponding Transformer Encoder architecture (right).

While initially developed for sequential data, the flexibility of Transformers has led to their adaptation to other domains. A notable extension is the Vision Transformer (ViT), which applies the Transformer architecture to image data by reinterpreting images as sequences of patches [46]. As visualized by Figure 3.4, instead of processing an image as a grid of pixels, ViT divides it into fixed-size patches, each of which is flattened and linearly projected into an embedding space:

$$z_i = Ex_i + p_i \quad (3.12)$$

where x_i is the i -th image patch, E is a learnable embedding matrix and p_i is a positional encoding that retains spatial information. The sequence of embedded patches is then processed by a standard Transformer encoder.

This design allows the model to capture global relationships between different regions of the input from the very first layer, in contrast to convolutional architectures that build such understanding progressively. Additionally, a special classification token is often appended to the sequence, whose representation is used for downstream prediction tasks.

One of the main advantages of Vision Transformers is their ability to model long-range dependencies and global context effectively. This property is particularly beneficial in scenarios where relationships between distant regions are important. Moreover, ViTs exhibit strong scalability: their performance improves significantly with larger datasets and model sizes, often surpassing traditional architectures when sufficient data is available.

However, this flexibility comes with trade-offs. Unlike convolutional models, Transformers lack inherent inductive biases such as locality and translation invariance, which can make them less data-efficient. As a result, they often require large-scale pretraining or hybrid approaches that incorporate convolutional components to achieve optimal performance.

Beyond classification, Vision Transformers are increasingly used as general-purpose feature extractors. The learned representations can be transferred to a variety of downstream tasks, benefiting from the model's ability to encode rich and globally consistent information. This has led to their widespread adoption in modern deep learning pipelines.

In summary, Vision Transformers extend the Transformer paradigm to non-sequential data by reformulating inputs as token sequences. Through self-attention mechanisms and flexible architecture design, they provide a powerful alternative to traditional models, particularly in settings where capturing global context is essential.

Representation Learning for Assembly State Recognition

Specifically for the Assembly State Recognition task, previous work has already been presented in the literature following a Representation Learning approach.

As previously mentioned in Section 2.1, Tim J. Schoonbeek et al., in their work “Supervised Representation Learning Towards Generalizable Assembly State Recognition” [2], introduced such an approach for the task using the IndustReal dataset [1], which was also created and is maintained by the same authors. This work is taken as the starting point for the research presented in this thesis, with the objective of further improving the proposed solution.

In order to clearly understand both the baseline approach and the contributions introduced in this thesis, two aspects must first be described in detail. First, the dataset used in both the previous work and in this thesis must be introduced, in order to understand its composition and the semantics of its data, which accurately reflect the industrial scenario for which ASR solutions are developed. Second, the previously proposed approach must be explained together with its underlying motivations, so as to clarify how a representation learning based solution for ASR can be designed to effectively exploit the available data and its structure. Both aspects are addressed in the following sections.

4.1 IndustReal Dataset

The IndustReal dataset was specifically designed to reflect a realistic industrial scenario and to enable the development and evaluation of methods for monitoring assembly procedures. Data were collected using an augmented reality headset, the Microsoft HoloLens 2¹, which recorded 27 participants acting as

¹Microsoft HoloLens 2 documentation. (<https://learn.microsoft.com/en-us/hololens/>)

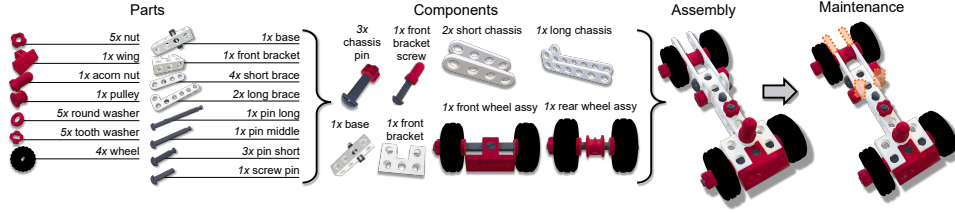


Figure 4.1: Overview of the construction-toy car’s parts, components and one of the assembly states used in IndustReal. The maintenance task represents a component upgrade and consists of replacing the long braces with short braces on the rear chassis.

operators in an industrial-like environment while they performed assembly procedures on a construction-toy vehicle based on model available on STEMFIE².

The toy vehicle was specifically designed for the creation of the dataset and carefully defined in terms of its structural elements. Individual pieces are defined as *parts*, while partial assemblies of parts form *components*. Larger compositions of parts and components define an *assembly state*. Figure 4.1 presents the full set of parts and their needed quantities, the possible components and an example of an assembly state resulting from a procedure. As can be observed, the model exhibits considerable structural complexity, including multiple types of washers, pins, and nuts.

Regarding the procedures, participants were provided with printed instructions describing how to build the vehicle from scratch or perform maintenance operations. These instructions were not strict step-by-step guidelines but rather described subgoals that operators had to achieve during the procedure. These subgoals consist of assembling or disassembling specific components, either partially or fully, enabling the operator to transition from one assembly state to the next according to the procedure. In total, 22 assembly states were formally specified as particular combinations of assembled parts, and for each one of them, a 3D model was made available to incentivize the generation and use of synthetic data. While following the procedure, participants were allowed to make mistakes, meaning that not all subgoals were necessarily reached correctly.

During data collection, all sensors of the headset were used, resulting in multimodal recordings including RGB video, stereo video (left and right streams), depth and visible-light measurements per pixel, gaze tracking, hand joint tracking and head-pose information. Recording conditions were kept consistent across sessions. Participants performed the procedures on a white desk placed against a white background, ensuring uniform lighting and minimal background variation. Such controlled conditions were considered representative of common

²STEMFIE model used for IndustReal creation. (<https://stemfie.org/sps-000001>)

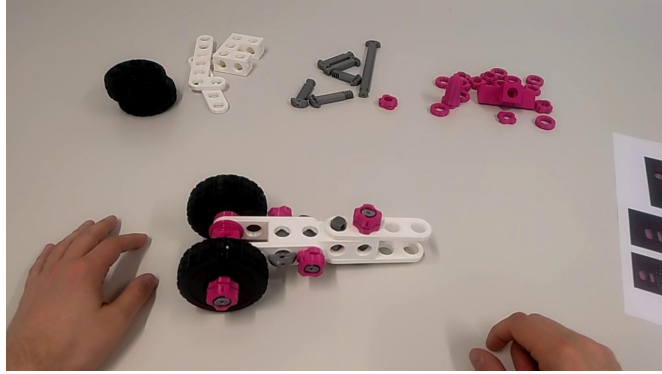


Figure 4.2: Example frame from the IndustReal dataset, showing the typical scene with both assembled components and loose parts resulting from the progression of the assembly procedure.

industrial environments. Before each recording, all necessary parts, along with some redundant pieces, were sorted by category into heaps randomly placed on the desk. As the procedure progresses, parts are gradually assembled, resulting in scenes that contain both assembled components and loose parts, as illustrated in Figure 4.2, which shows a representative frame from the dataset.

The collected data are annotated with multiple label types to support different research tasks. For Action Recognition (AR), frame-level annotations specify the action performed by the operator at each time step. For frames corresponding to segments in which the operator reached a subgoal, bounding box annotations localize the assembled object and assign it one of 22 predefined assembly states if correctly achieved or an *error state* label otherwise, enabling the study of the Assembly State Detection (ASD) task, previously discussed in Section 2.1. Additionally, the annotation scheme represents assembly states not only as discrete labels but also as sequences of integers, where each element encodes the status of a distinct component: correctly assembled (1), incorrectly assembled (-1) or not assembled (0), allowing for more fine-grained supervision and evaluation.

A central focus of the dataset is the Procedure Step Recognition (PSR) task, which aims to determine which procedural steps are completed at which points in time. For this purpose, ground-truth annotations provide the correct temporal sequence of assembly states with associated timestamps.

Overall, the dataset constitutes an egocentric multimodal benchmark comprising approximately 5.8 hours of recordings. The data are split by operator into training, validation, and test sets (12/5/10 operators), ensuring viewpoint variability between splits. Figure 4.3 summarizes the available data modalities and annotations through an example extracted from one of the recorded sequences.

For the ASR task, the dataset author for his work [2] prepared and released

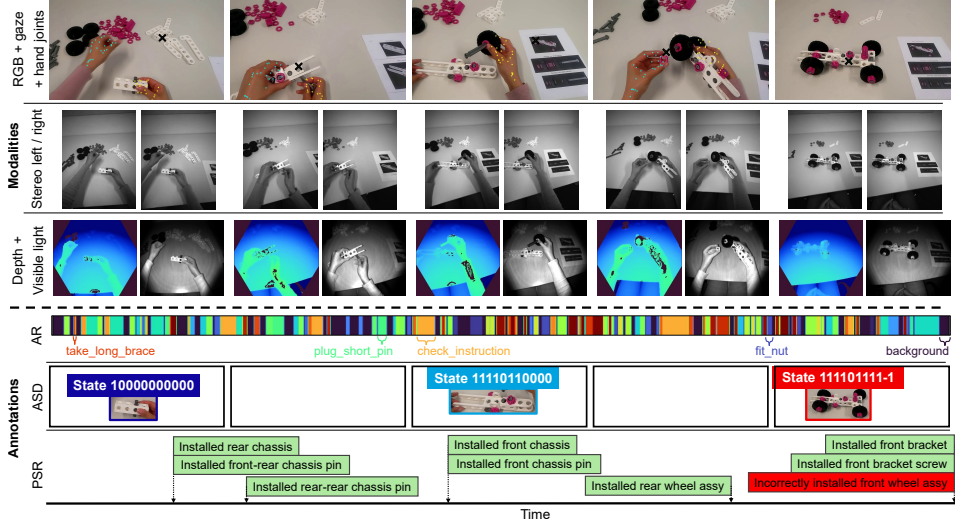


Figure 4.3: Samples from a clip in the IndustReal dataset, demonstrating the modalities and annotations for all three tasks. Gaze is indicated by the cross, detected hand joints by the dots. AR: action recognition, ASD: assembly state detection, PSR: procedure step recognition.

an extended version of the IndustReal dataset specifically designed to enable a more in-depth study of Assembly State Recognition.

Maintaining the original train/validation/test split, the IndustReal videos were decomposed into individual frames, and each frame was assigned a label corresponding to the assembly state depicted in the image. While partial information about assembly states was already available through the ASD annotations, those labels only covered video segments in which a procedural step had been completed. In such cases, the depicted state corresponded either to one of the 22 well-defined assembly states or to an error state. However, no annotations were available for the frames belonging to the transitions between two states.

The main contribution of this dataset extension is the introduction of labels for these previously unlabeled frames. To this end, an additional class called *intermediate state* was introduced. This class represents all transitional configurations that the assembled object goes through while the operator moves from one well-defined assembly state to another during a procedure step. This design choice is necessary because explicitly defining all possible transition states is infeasible in practice. The number of such states would grow exponentially with the number of parts and possible attachment configurations, making their explicit enumeration impractical. Consequently, all transitional configurations are grouped into a single *intermediate state* category.

As expected, this design results in a strong class imbalance. Since most frames

Error category	Description	Example reference – sample pair	# frames
I. Missing (component)	Component is missing from the assembly state		1144
II. Orientation (component)	Orientation of component is wrong		398
III. Placement (component)	Component is placed at a wrong position		85
IV. Part level	Parts within components are missing/oriented wrong		763

Figure 4.4: Error categorization with visual examples and the number of frames in IndustReal for each category. Depending on whether the error occurs at the component level or the part level, mistakes are grouped into four distinct categories.

depict operators actively manipulating the object during the procedure, the majority of frames correspond to transitional configurations. In fact, more than 90% of the dataset frames are labeled as intermediate states.

In addition, to support a detailed analysis of the robustness of ASR solutions to assembly errors, further annotations were introduced for frames labeled as error states. Since errors occur while attempting to reach a specific well-defined assembly state, it is possible to identify the intended target state that the operator was aiming to achieve. This information was added for each *error state* frame. Moreover, errors were categorized according to their nature, enabling evaluation across different error types. The considered categories are: missing component (Category I), incorrect component orientation (Category II), incorrect component placement (Category III) and part-level errors (Category IV). Figure 4.4 illustrates these categories with visual examples and shows their distribution among the available samples.

Finally, the dataset extension also provides a pool of synthetically generated images depicting different assembly states, which can be used for experimentation. However, as will be explained later in this thesis, the synthetic images used in the present work were generated through a different pipeline developed specifically for this research.

Since this version of the dataset, the one proposed in [2], is the one designed for the study of Assembly State Recognition, from this point onward the term “IndustReal dataset” will refer to this version, which is also the one adopted throughout this thesis.

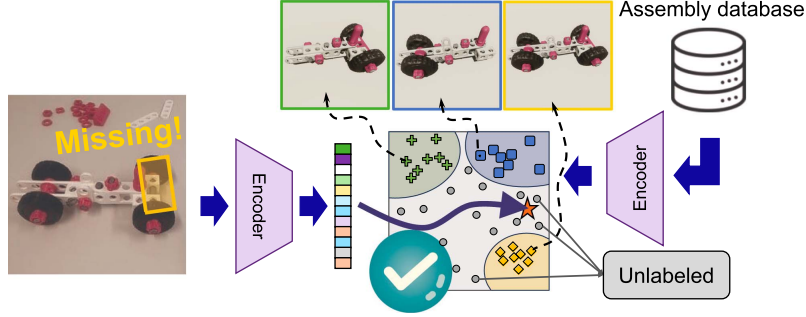


Figure 4.5: Scheme depicting the overall Assembly State Recognition framework based on Representation Learning developed in [2] and adopted for this thesis work.

4.2 Baseline Approach

In [2], the author investigated different configurations for ASR systems that aim to satisfy all the desired properties previously discussed: accuracy on known states, generalization to unseen ones, robustness to errors and scalability. The outcome of this study was the identification of a general representation learning based design for ASR systems, composed of three main components: (i) an encoder model, (ii) a loss function and (iii) a similarity function.

The first, possibly implemented as a full encoding pipeline including additional preprocessing steps, is required to generate a representation vector, also referred to as an embedding, from an input video frame. Secondly, an appropriate loss function must be selected to train the encoder so that the produced embeddings are meaningful and representative of the corresponding samples. The objective is to learn a representation space in which samples belonging to the same class are mapped to nearby points, while samples from different classes are mapped far apart from each other. Finally, the generated embedding alone is not sufficient to directly infer an assembly state label. Instead, prediction is performed by comparing the embedding of a query sample with a set of embeddings computed from labeled reference samples. The label of the most similar reference embedding is then assigned to the query sample. Therefore, a similarity function must also be chosen to measure distances in the learned representation space. Figure 4.5 illustrates the overall setup of the proposed framework.

In the original study, several alternatives for these components were explored. The best-performing configuration reported in the paper was adopted as the baseline for this thesis and is described in detail in the remainder of this chapter.

While the choice of similarity measure was relatively straightforward, cosine

similarity proved to perform better than alternatives such as the ℓ_2 distance, the design choices concerning the encoder and the loss function are more involved and require a more detailed explanation. These aspects are therefore discussed in the following dedicated sections.

4.2.a Image Feature Projection Model

As encoder model for generating representations of video frames, the authors opted for an image-based encoder. Using such a model, it is possible to directly extract visual features from the frames, which can then serve as candidate representations for the samples. In the best-performing configuration, the image encoder was implemented using a Vision Transformer (ViT) [46], previously introduced in Section 3.4, pre-trained on the ImageNet-1k dataset [47]. The encoder was obtained by removing the final fully connected layer of the ViT architecture and using the output of the penultimate layer as the image representation, resulting in a 384-dimensional feature vector.

Before being fed to the encoder, several preprocessing steps were applied to ensure efficient training and stable model behavior. Images were resized to 224×224 pixels ($W \times H$) and normalized using the mean and standard deviation computed over the entire dataset. During training only, data augmentation was applied to increase robustness and reduce overfitting. Specifically, random Gaussian blur (kernel size 5, $\sigma = 2$) was applied, together with brightness, saturation, and contrast jittering with maximum intensities of 0.1, 0.7, and 0.1, respectively.

Rather than directly using the features (f_i) extracted by the image encoder as the final representations, an additional three-layer multilayer perceptron (MLP) was attached to the encoder and used as a projection head, mapping the features into embeddings $e \in \mathbb{R}^D$. This design choice is motivated by previous work showing that introducing a projection head and defining the loss function in the projected space $\mathbb{R}^D$ leads to more effective representation learning [18].

During inference, however, the last two layers of the three-layer MLP projection head are removed to obtain the final embeddings. Retaining only an intermediate projection layer has been shown to provide better performance than discarding the projection head entirely [48].

Overall, the complete encoding pipeline consists of both the image encoder and the projection head. During training, the image encoder is fine-tuned, while the projection head is trained from scratch.

The encoding pipeline of the baseline approach is illustrated by Figure 4.6.

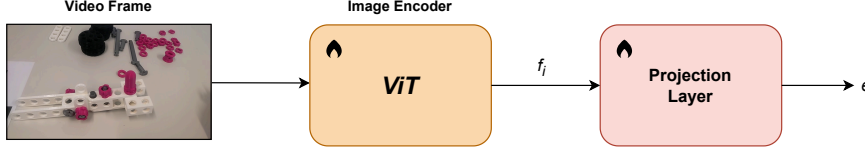


Figure 4.6: Baseline approach encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning.)

4.2.b Intermediate-State Informed Loss

As discussed in Section 3.1, representation learning commonly relies on contrastive loss functions that encourage samples belonging to the same class to have similar representations, while pushing apart representations of samples from different classes.

However, this principle is not entirely compatible with the semantics of the IndustReal dataset. In particular, the *intermediate state* category, as explained in Section 4.1, is defined as a generic label that groups together all transitional states. Consequently, frames labeled as intermediate state may correspond to arbitrary transitions between different assembly configurations and are therefore often unrelated to each other. For this reason, enforcing similarity among representations of all *intermediate state* samples would be inherently incorrect. Treating them as belonging to a single class would force the model to learn artificial relationships between visually and structurally different configurations, potentially limiting the expressive capacity of the learned embedding space.

At the same time, discarding *intermediate state* samples during training would also be suboptimal. The absence of a precise label does not imply the absence of meaningful visual information that could contribute to representation learning. Furthermore, the capability to distinguish frames depicting a well-defined assembly state from those corresponding to an intermediate state is highly desirable for robust deployment of ASR systems in real-world scenarios.

To address the limitations posed by intermediate states, the baseline work introduces an intuitive modification to the adopted supervised contrastive loss functions, referred to as *Intermediate-State Informed Loss* (ISIL). This loss formulation leverages intermediate assembly states by treating them exclusively as hard negative samples. In this way, intermediate states are encouraged to be dissimilar from all explicitly defined assembly states, while no penalty is applied for dissimilar embeddings between intermediate states themselves, avoiding the enforcement of artificial similarity among potentially uncorrelated training samples.

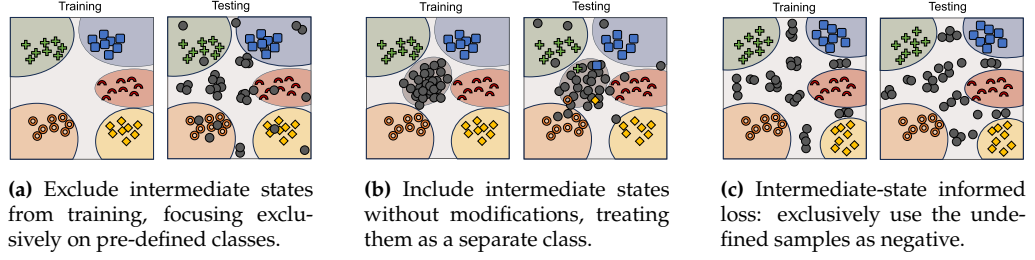


Figure 4.7: Illustration of influence of the proposed loss function that leverages images containing intermediate assembly states, i.e. non-defined, transitional states between pre-defined states.

The ISIL concept can be applied to a variety of representation learning loss functions. In the baseline study, the best-performing approach among the tested alternatives was the Supervised Contrastive (SupCon) loss [26]. With the ISIL modification, the SupCon loss is redefined as

$$\mathcal{L}_{\text{SupCon}} = \sum_{i \in I} \log \frac{-1}{|P(i)|} \sum_{p \in P(i)} \frac{\exp \left(\frac{g(f(x_i)) \cdot g(f(x_p))}{\tau} \right)}{\sum_{a \in A(i)} \exp \left(\frac{g(f(x_i)) \cdot g(f(x_a))}{\tau} \right)} \quad (4.1)$$

where $A(i) \equiv I \setminus \{i\}$ the set of all indices in mini-batch I besides i , the one of the considered sample, $P(i) \equiv \{p \in A(i) : (y_p = y_i) \wedge y_p \in C\}$ is the set of indices of all positives for defined classes, C is the set of pre-defined assembly classes, $f(x)$ is the embedding function, $g(x)$ is the projection head and τ denotes the temperature constant.

In this function, the ISIL modification is enforced by excluding the undefined states from the set of positives indices $P(i)$, while permitting them to act as negatives in $A(i)$.

Figure 4.7 illustrates how the ISIL strategy differs from other approaches in handling intermediate state embeddings during training and testing.

If intermediate states are completely excluded from training, their embeddings during testing can be arbitrary, potentially very close to real states (Figure 4.7a). Including them as positive examples to cluster intermediate states together can also be problematic: since not all possible intermediate states are seen in training, during testing some embeddings may differ from the others, potentially overlapping with real state ones. Also, attempting to correlate usually uncorrelated data can hinder model training, potentially causing some real state embeddings to shift toward intermediate ones and reducing overall performance (Figure 4.7b).

The ISIL loss, in contrast, treats intermediate states exclusively as negative samples during training. This encourages the model to focus on separating real

from intermediate states without enforcing any similarity among intermediate states themselves, resulting in appropriate embedding behavior during testing (Figure 4.7c).

4.2.c Limitations

Although the baseline approach is well designed from a training perspective, as it accurately reflects the properties of the data and clearly defines the learning objective, the adopted encoder model architecture consists solely of an image encoder and a projection head.

Assembly states are characterized by specific part configurations, and the visual differences between well-defined states can be extremely subtle. Even when fine-tuned on the target dataset, an image encoder operating as a generic feature extractor may struggle to capture such fine-grained structural variations. Consequently, adopting an alternative feature extraction strategy that explicitly models and preserves the structural composition of the assembly depicted in the video frames is highly desirable.

This thesis is motivated by the aforementioned limitations and proposes a novel method to enable part-level reasoning. Specifically, assembled components are first detected in each frame using object detection models, and the resulting detections are used to construct graph representations that encode the structural configuration of the assembly. These graphs serve as enriched inputs, capturing the relationships between components in the current state. An ad-hoc Graph Neural Network (GNN) based model is then employed as an encoder to learn meaningful representations from this structured data. The proposed method is presented in detail in Chapter 5. Subsequently, Chapter 6 introduces a synthetic data generation pipeline to efficiently train the object detectors, while Chapter 7 discusses strategies to enable their effective deployment on real-world data.

Assembled Parts Graph Embeddings for ASR

As previously introduced, the method proposed in this thesis addresses the ASR task within a representation learning framework by designing a novel encoder model capable of extracting meaningful, structure-aware features. In particular, the model focuses on capturing information about the assembled parts that define the current assembly state.

At the core of the approach, structural information is extracted from video frames by detecting assembled components using custom-trained object detection models. These detections provide the raw assembly-state information, which constitutes the foundation of the proposed method. However, such outputs cannot be directly exploited for representation learning, thus motivating the design of a suitable data representation able to effectively encode and leverage part-level information.

Building on this representation, a dedicated deep learning model is developed to generate assembly state embeddings, which represent the final output of the framework in accordance with the representation learning paradigm.

An overview of the complete pipeline is shown in Figure 5.1. This chapter details all its components: first, the design choices for assembled part detection are presented; then, the proposed data representation is introduced; next, the model architecture for processing this representation is described; finally, multimodal integration strategies with the baseline image-based approach are explored, with the aim of enhancing the quality of the learned assembly-state representations.

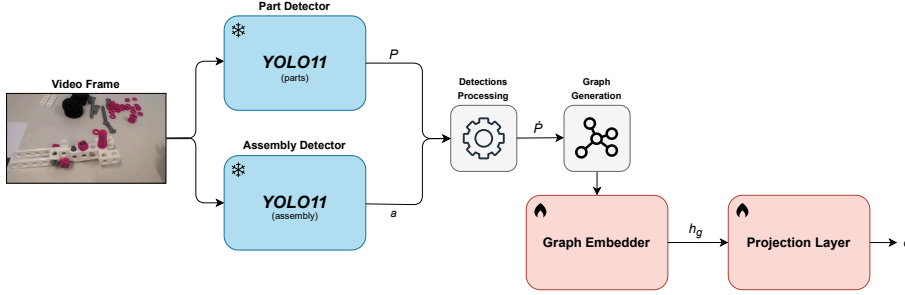


Figure 5.1: Novel GNN-based approach encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning; snowflake icon and blue color indicate components trained separately.)

5.1 Assembled Parts Detection

To detect assembled parts in video frames, object detection models, introduced in Section 3.2, represent a suitable and effective solution.

Accordingly, the first step in developing the proposed system consists in selecting an appropriate object detection model. As previously discussed, YOLO-based detectors represent a convenient choice because they provide an effective combination of state-of-the-art performance, high inference speed and a relatively lightweight architecture.

For this reason, YOLO11 [49] was selected as the object detection model since, at the time of development, it offered the best available performance within the YOLO family. YOLO11 is developed and maintained by Ultralytics¹, an open-source AI organization focused on state-of-the-art, efficient and production-ready computer vision models. In addition to the model itself, Ultralytics provides a comprehensive repository² including utility scripts that simplify training and evaluation, allow customization of data augmentation and training hyperparameters, and support training monitoring. The availability of these tools was an additional factor in selecting YOLO11, as they enabled extensive and efficient experimentation with model training and evaluation.

YOLO11 is available in multiple model variants, ranging from *nano* to *XL*, where larger models have more parameters and higher performance but slower inference. Table 5.1 compares these variants through official training and evaluation results on the COCO dataset [50] in terms of detection performance through the mAP_{50-95} metric, presented in detail later in Chapter 8, and inference times

¹Official Ultralytics organization website. (<https://www.ultralytics.com>)

²Official Ultralytics GitHub repository. (<https://github.com/ultralytics/ultralytics>)

Model	mAP_{50-95}	Speed CPU (ms)	Speed T4 (ms)	Params (M)
YOLO11n	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6
YOLO11s	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4
YOLO11m	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1
YOLO11l	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3
YOLO11x	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9

Table 5.1: Official performance comparison of YOLO11 models on the COCO dataset.

on CPU and GPU (T4) after model conversion to ONNX and TensorRT formats.

For this thesis, the medium-sized model YOLO11m was selected as it provides a balanced trade-off between performance and speed. As shown in Table 5.1, YOLO11m achieves only 3.2 points lower mAP_{50-95} than the largest variant, while requiring less than half the inference time on both ONNX and TensorRT. Reducing the model size further to YOLO11s would result in a more significant drop in performance ($-4.5 mAP_{50-95}$), making it less suitable for the task.

The results of the YOLO11 models training on the COCO dataset, namely the model weights, were made available by Ultralytics. These weights constitute an important starting point, as deeper components of the network, such as the backbone, are already trained to extract meaningful features for downstream object detection tasks. Accordingly, an effective learning strategy was adopted, where YOLO11m models were fine-tuned from weights pre-trained on the COCO dataset using IndustReal images. The complete training and evaluation protocol is detailed in Chapter 8. Specifically, the fine-tuning process leverages both real video frames from the dataset and synthetic images generated through a custom pipeline, described in detail in Chapter 6.

As suggested and visually supported by the encoding pipeline overview in Figure 5.1, assembled part detection is performed using a multi-detector system. In particular, two YOLO11m models are employed: one dedicated to part detection and the other to assembled object detection. A subsequent post-processing step combines their outputs to identify assembled parts.

The motivations behind these design choices and the combined use of real and synthetic data for model training are presented in Chapter 7.

5.2 Graph Generation

The data representation identified as most suitable for encoding the raw assembly structure information extracted from assembled part detections is graph-based. This choice is motivated by the fact that an assembly state can be effec-

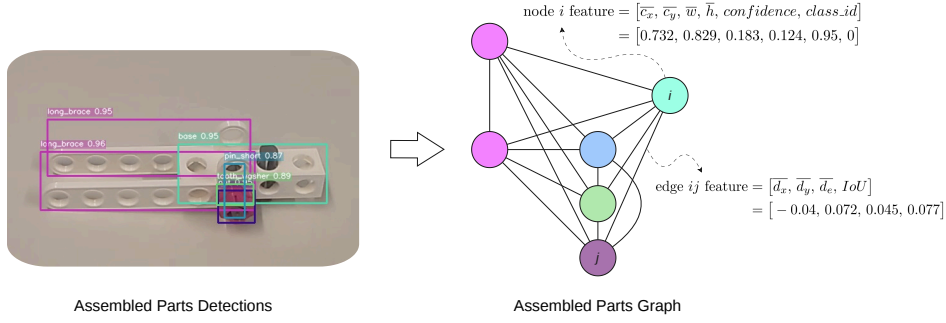


Figure 5.2: Example of graph generated from assembled part detections. Nodes represent individual parts, edges encode structural relationships and the accompanying numerical values illustrate examples of crafted node and edge features.

tively represented by modeling detected parts as nodes, whose features encode the specific properties of each detection, and by explicitly modeling the relationships between detections through edges.

Given the conceptual similarities between the context of this work and the computer vision field of action recognition (AR), discussed in Section 2.3, previous approaches that construct graphs from object detections were taken as inspiration to design a tailored graph formulation for this problem. Given a set of assembled parts detections $\hat{P}$, a graph $G = (V, E)$ is constructed, where V denotes the set of nodes and E the set of edges. The construction of nodes and edges is detailed in the following subsections.

As will be shown, both crafted node and edge features rely exclusively on detection-level information, namely bounding boxes, predicted classes and confidence scores. No additional visual features are incorporated. This differs from common AR approaches, where region-of-interest visual features extracted by a separate image encoder are often used, as in [35] and [37]. In the proposed approach, visual appearance is considered unnecessary for assembly state recognition, since the presence, type, and spatial arrangement of assembled parts already encode the relevant information. Consequently, the adopted features are primarily geometric, focusing on bounding box dimensions and spatial relationships. Feature engineering choices for node representations were inspired by [36], while edge feature design draws inspiration from [38].

The generation design for assembled part graphs is illustrated in Figure 5.2, showing an example of a generated graph with nodes and edges, along with some numerical examples of crafted node and edge features.

Nodes. As introduced earlier, in the graph formulation associated with a set of detections $\hat{P}$, one node is added to the node set for each detection. Each node

is enriched with a set of features designed to characterize the corresponding detected part. Given that the image dimensions (width and height) are known, and that each detection is defined by a bounding box, a predicted class and a confidence score, the following six features are computed.

A bounding box in COCO format is defined as (x, y, w, h) , where (x, y) denotes the top-left corner and w and h represent width and height, respectively. From this representation, the centroid coordinates are computed as

$$c_x = x + \frac{w}{2}, \quad c_y = y + \frac{h}{2}.$$

The node features are then defined as:

- normalized centroid x coordinate: $\overline{c_x} = c_x/W$;
- normalized centroid y coordinate: $\overline{c_y} = c_y/H$;
- normalized bounding box width: $\overline{w} = w/W$;
- normalized bounding box height: $\overline{h} = h/H$;
- prediction confidence: *confidence*;
- predicted class identifier: $class_id \in \{i \mid i \in [0, 14]\}$, accounting for the 14 part classes plus the additional *unknown* class.

All features are carefully designed to lie within the $[0, 1]$ interval in order to improve numerical stability, optimization efficiency and convergence speed during training. Input normalization is a standard practice in machine learning and is therefore applied here as well. The only exception is the class identifier, which, although bounded, takes discrete integer values also greater than one. This feature is not directly fed into the model; instead, it is transformed into an alternative representation, as will be detailed later, to improve class-wise expressivity and facilitate more effective training.

Edges. With the node set defined, the remaining step is the construction of the edge set to complete the graph. Since nodes correspond to detections, edges are used to encode the relationships between detections. The underlying idea is to exploit all available information by explicitly modeling how each detection relates to every other one. This design choice results in the creation of an edge for every ordered pair of distinct nodes, yielding a *complete digraph*, which corresponds to a directed graph in which every pair of distinct vertices is connected by two unique edges, one in each direction [51].

In related literature, edges in such graphs are often assigned a scalar weight to encode relational information. For instance, in [36], edge weights are defined as

the exponential inverse of the normalized *Euclidean* distance between bounding box centroids. While effective in certain contexts, this strategy restricts each edge feature to a single scalar value, which was considered limiting for the purposes of this work. Instead, all edges are assigned a uniform weight (set to 1.0), and relational information is encoded through a dedicated edge feature vector, similarly to the node representation.

Formally, for each ordered pair of detections (p_i, p_j) such that $p_i, p_j \in \hat{P}$ and $i \neq j$, a directed edge (p_i, p_j) is added to the edge set E . Each edge is described by a four-dimensional feature vector. Given bounding boxes in COCO format (x, y, w, h) , where (x, y) denotes the top-left corner and w and h represent width and height, and denoting the image dimensions by W and H , the edge features are defined as:

- normalized horizontal offset: $\overline{d_x} = (x_j - x_i)/W$;
- normalized vertical offset: $\overline{d_y} = (y_j - y_i)/H$;
- normalized Euclidean distance between bounding box centroids $\overline{d_e}$:

$$\frac{\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}}{\sqrt{W^2 + H^2}};$$

- intersection over union (IoU):

$$\frac{\text{area}(p_i \cap p_j)}{\text{area}(p_i \cup p_j)}.$$

All edges are explicitly directed. While an undirected formulation could have been considered, connecting every pair of nodes would imply treating (p_i, p_j) and (p_j, p_i) as equivalent. This is undesirable in this context because, although the IoU and normalized distance are symmetric, the horizontal and vertical offsets change sign when swapping the node order. Preserving this directional information is crucial to accurately encode the relative spatial arrangement of bounding boxes, and therefore motivates the use of a directed graph formulation.

5.3 Graph Embedder Model

To process the graphs constructed from the detections, an appropriate deep learning architecture needed to be identified with specific capabilities. The model must be able to take as input the formulated graphs, comprising feature vectors associated with both nodes and edges, and effectively exploit this information to learn a meaningful representation of the assembly state.

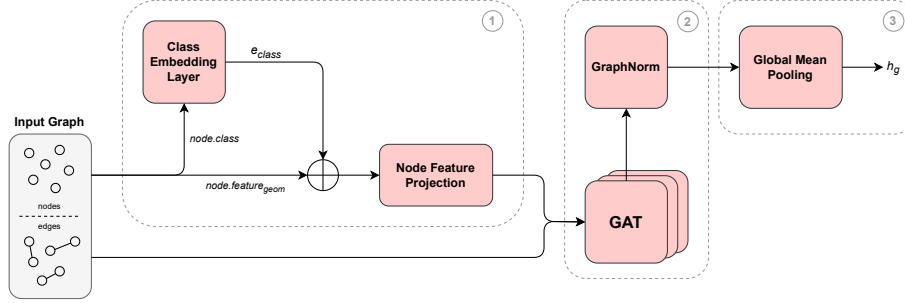


Figure 5.3: *Graph Embedder* model architecture. A custom GNN-based model for part detections graph feature extraction.

As described in Section 3.3, Graph Neural Networks (GNNs) provide exactly these capabilities and were therefore selected as the fundamental building block for the proposed model.

For the same reasons considered in the design of the baseline embedding model, the architecture was divided into two main components. The first component is responsible for extracting graph-level features from the input graph, similarly to how the image encoder operates in the baseline approach. The second component consists of a projection module that maps the extracted features into the embedding space representing assembly states.

The projection module was kept identical to the one used in the baseline, as it already proved effective. The graph feature extractor, instead, was implemented from scratch using PyTorch [52] together with the PyTorch Geometric (PyG) library [53]. Its architecture is described in detail in the remainder of this section, along with an explanation of how the input graphs are processed during inference.

The proposed GNN-based model, referred to as *Graph Embedder*, operates as follows. Given an input graph (or a batch of graphs), the model processes the data through three main stages: (i) node feature setup, (ii) node feature update via message passing, and (iii) graph-level feature extraction, matching the model architecture illustrated in Figure 5.3. This structured design highlights the role of each component and how information is progressively transformed from part-detection graphs into compact assembly state representations.

Node feature setup. The first stage consists of constructing expressive node features. As explained in earlier sections, node attributes are pre-computed and directly available, with the exception of the `class_id`. Rather than encoding class information as a scalar identifier, a richer representation is adopted. Inspired by word embedding techniques in Natural Language Processing, such as *word2vec*

[54], each part class is associated with a learnable embedding vector of dimension 16. Given the limited number of classes (15, including the *unknown* class), this dimensionality is sufficient to capture class-specific semantics. During inference, the `class_id` of each node is used to retrieve the corresponding class embedding, which is concatenated with the remaining ‘geometric’ node features.

After concatenation, node features are projected from their initial dimensionality (21) to a higher-dimensional space of 64 using a multi-layer perceptron composed of two linear layers with a ReLU activation in between. This projection step increases node-level expressivity and prepares the representations for effective message passing.

Node feature update. In the second stage, node representations are iteratively refined through message passing. This is implemented using a stack of Graph Attention Network (GAT) layers, introduced in Section 3.3. Specifically, three GAT layers are employed, each with four attention heads and a constant node feature dimensionality of 64. GATs are particularly well suited to this task, as their attention mechanism learns to weight neighboring nodes while explicitly incorporating edge features. This allows the model to fully exploit the adopted graph representation. Self-loops are also added to the graph, ensuring that a node’s own features are preserved and considered during aggregation. The number of layers and attention heads follows standard practices in the literature.

The output of the final GAT layer is then passed through a *GraphNorm* layer [55], which stabilizes training and accelerates convergence by normalizing node features at the graph level. After this step, each node is associated with a refined and stable representation that encodes both local and contextual information.

Graph-level feature extraction. The final, third, stage consists of extracting a single representation for the entire graph. This is achieved through a global mean pooling operation, which computes the graph embedding by averaging the features of all nodes. The resulting graph-level embedding compactly summarizes the assembly configuration depicted in the input frame. This is the feature that is subsequently passed to the projection layer, kept identical to the baseline and previously described in Section 4.2.a.

For training, the same approach adopted by the baseline, presented in Section 4.2.b, was retained. This strategy was considered appropriate for the task and remains fully compatible with the novel graph-based model.

For testing and general ASR inference, the procedure introduced in the baseline approach was also retained. Given a reference set of labeled samples, a query sample can be assigned a label by comparing its embedding with those of



(a) intermediate state (val/00014359.jpg)



(b) real state (val/00014370.jpg)

Figure 5.4: Example illustrating the intermediate–real state labeling issue in the dataset. Both frames depict the same real assembly state (13); however, the first is labeled as *intermediate* solely due to the presence of the operator’s hands in the frame.

the reference set and selecting the label of the closest reference embedding.

5.4 Multimodal Approach

The novel graph-based approach to representation learning for assembly state recognition, which leverages part detection and Graph Neural Networks, represents a significant advancement compared to the generic baseline approach. It has the potential to extract fine-grained information about the assembly state depicted in video frames, potentially improving task performance. However, one limitation remains.

As discussed in earlier sections, the IndustReal dataset labels video frames either with a real assembly state (1-22) or with the intermediate state label (0). According to the recording protocol described in the dataset paper [1], operators were instructed to remove their hands from the table when completing a step, thereby reaching a real assembly state. Consequently, only frames without visible hands can receive a real state label, while all others cannot.

This creates a challenge: in segments where an operator has reached a real assembly state but still has the object in hand or hands on the table, the novel graph-based model would predict a real assembly state based solely on the assembled parts. Yet, the ground truth is labeled as intermediate. This discrepancy can hinder training, as equivalent assembled part configurations may have different labels across frames. This limitation is imposed by the dataset itself and must be addressed to fully realize the potential of the novel model in terms of dataset-based metrics.

Figure 5.4 illustrates a clear example of this issue by showing two video frames from the dataset in which two identical assembly configurations are assigned different labels.

One possible solution would be to incorporate information about hand presence directly into the input representation, for example by detecting hands with an object detector and integrating those detections into the previously introduced graph formulation. Such a solution could potentially address the issue, but it would be highly dependent on the specific characteristics of the adopted dataset. In different industrial scenarios, the intermediate-state labeling might be influenced not only by the presence of hands but also by other foreign objects, such as assembly tools. Consequently, obtaining and maintaining detectors capable of recognizing all possible foreign objects that may appear in the scene would be impractical.

Instead, the proposed solution draws inspiration from the baseline approach. While an image encoder may be less suitable for fine-grained assembly state recognition, as discussed in earlier sections, it can still be effective in distinguishing intermediate states from well-defined assembly states. This is because the resulting embeddings are sensitive to the presence of large or salient objects in the scene, such as hands, tools, or other foreign elements. These visual cues strongly influence the assignment of the “intermediate” label and therefore can provide useful complementary information.

Therefore, an alternative, multimodal, approach that combines the strengths of the baseline and the novel approach was developed, aiming to combine their strengths and to complement their weaknesses. In particular, two models were implemented to evaluate different fusion strategies. First, a *naive fusion* approach is introduced, combining the modalities in a straightforward manner to assess the potential and limitations of a simple yet theoretically effective integration. Subsequently, a *gated fusion* approach is implemented to investigate whether a more specialized multimodal fusion mechanism can yield improved representation quality and downstream performance.

5.4.a Naive Fusion

To formulate a combined solution, it is essential to first examine the differences and similarities between approaches across modalities. The baseline method employs an image encoder that processes an input image to produce a visual feature embedding, initialized with pre-trained weights and fine-tuned in training. In contrast, the proposed approach leverages a GNN-based model operating on graphs derived from assembled part detections to generate graph-level embeddings, and is trained from scratch.

Despite these differences, both approaches share the same final component: a projection layer that maps the learned embedding to the assembly state representation space. The only distinction lies in the input dimensionality of the first

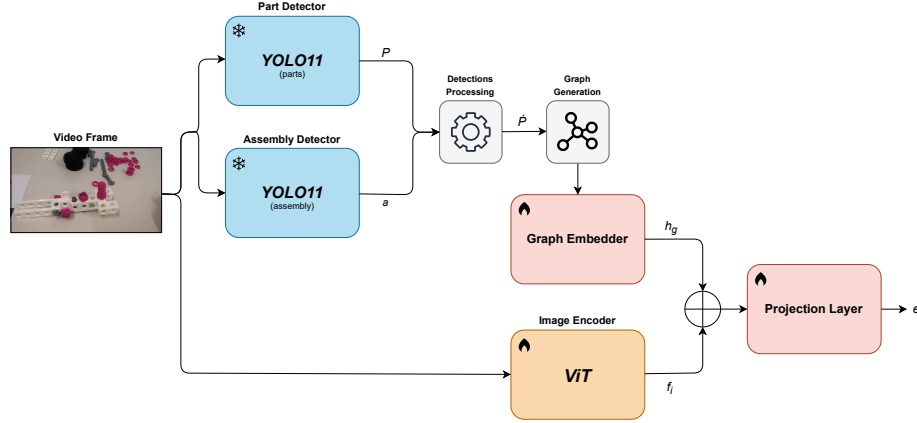


Figure 5.5: Multimodal approach with *naive* input fusion encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning; snowflake icon and blue color indicate components trained separately.)

projection layer, as the two embeddings have different sizes.

Based on these observations, a straightforward strategy to combine the two approaches is to fuse their respective embeddings into a single representation and use it as input to the projection layer. A first, and admittedly simple, fusion strategy consists of concatenating the two embeddings. Given that the image encoder (ViT) produces a 384-dimensional feature vector and the graph-based model outputs a 64-dimensional embedding, the resulting fused representation has a dimensionality of 448.

Figure 5.5 illustrates the complete encoding pipeline of this first multimodal approach, highlighting the parallel extraction of image and graph embeddings and their subsequent fusion prior to projection.

5.4.b Gated Fusion

The fusion approach based on simple concatenation is conceptually straightforward, but it naturally introduces a potential critical issue. The resulting 448-dimensional embedding is composed of 384 dimensions from the image embedding (approximately 86%) and only 64 dimensions from the graph embedding (approximately 14%), leading to a strongly unbalanced representation.

Even though the projection layer is an MLP with fully connected layers that, in principle, could learn to appropriately weight the two modalities during training, such a dimensional imbalance may bias the model toward relying predominantly on the image embedding. As a consequence, the resulting behavior

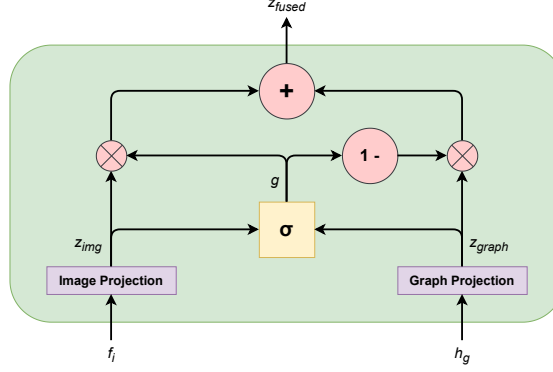


Figure 5.6: *Gated* input fusion module enabling learnable modulation of modality contributions to better exploit multimodal information.

may closely resemble that of the baseline approach, preventing the model from fully exploiting the complementary information provided by the graph-based representation. To address this issue, a more effective fusion strategy is required.

This problem has been encountered in prior work on multimodal learning. In particular, Arevalo et al. [56] introduced the *Gated Multimodal Unit* as an alternative to simple concatenation for fusing information from different modalities, demonstrating improved performance over naive fusion strategies.

Building upon this idea, a *gated fusion* approach was implemented. The fusion mechanism underlying this approach is illustrated in Figure 5.6 and is formally described by the following sequence of equations.

First, the image feature f_i and the graph embedding h_g are independently projected to a common fusion space of dimension 256 using a linear transformation followed by layer normalization and a ReLU activation function, which is standard practice. This step is described by Equations 5.1 and 5.2 and it is represented by the two purple blocks in the figure. The fusion dimensionality was chosen as an intermediate value between the original embedding sizes, allowing the model to better integrate structural information from the graph embedding while discarding image-related information that is not useful for the task.

$$z_{img} = \text{ReLU}(\text{LN}(\mathbf{W}_{img}f_i + \mathbf{b}_{img})) \quad (5.1)$$

$$z_{graph} = \text{ReLU}(\text{LN}(\mathbf{W}_{graph}h_g + \mathbf{b}_{graph})) \quad (5.2)$$

The core component of the fusion mechanism is the computation of the *gate*. First, the projected features z_{img} and z_{graph} are concatenated into a single feature vector z_{fusion} with twice the fusion dimensionality. This vector is then projected

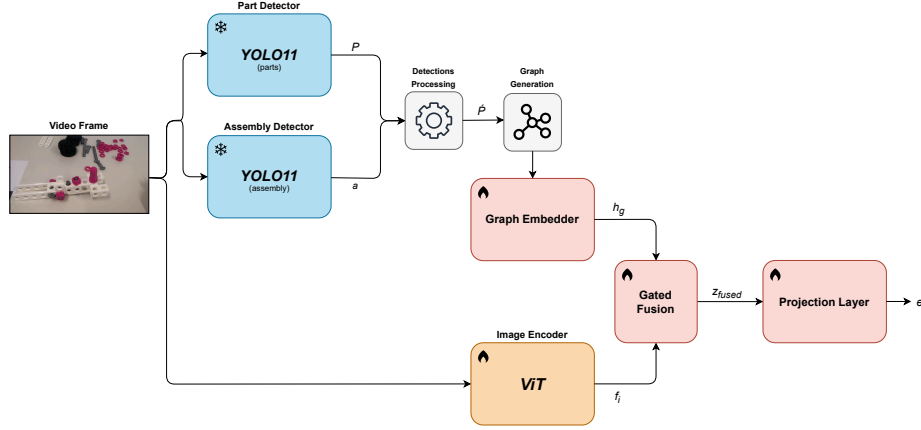


Figure 5.7: Multimodal approach with *gated* input fusion approach encoding pipeline. (Legend: flame icon indicates components directly trained for assembly-state recognition with red denoting training from scratch and orange fine-tuning; snowflake icon and blue color indicate components trained separately.)

back to the fusion space through a linear layer followed by a sigmoid activation, yielding a gating vector with values in the $[0, 1]$ range. This process is formalized in Equations 5.3 and 5.4 and it is represented by the yellow block in the figure (5.6).

$$\mathbf{z}_{\text{fusion}} = [\mathbf{z}_{\text{img}} ; \mathbf{z}_{\text{graph}}] \quad (5.3)$$

$$\mathbf{g} = \sigma(\mathbf{W}_g \mathbf{z}_{\text{fusion}} + \mathbf{b}_g) \quad (5.4)$$

Finally, the fused representation is computed as a convex combination of the projected image and graph features, weighted element-wise by the learned gate. This operation allows the model to dynamically balance the contribution of each modality on a per-dimension basis, as expressed in Equation 5.5, reflected by the figure (5.6) section involving red circles.

$$\mathbf{z}_{\text{fused}} = \mathbf{g} \odot \mathbf{z}_{\text{img}} + (\mathbf{1} - \mathbf{g}) \odot \mathbf{z}_{\text{graph}} \quad (5.5)$$

Figure 5.7 presents the complete encoding pipeline of the proposed multimodal approach with *gated fusion*.

As in the graph-only approach, the training strategy adopted in the baseline was retained for both multimodal models, as it was considered valid and fully compatible with the proposed architectures.

Similarly, assembly state prediction for both models is performed using the same query-reference similarity framework adopted in the other representation learning-based approaches.

Synthetic Dataset Generation Pipeline

As mentioned in the introduction (Chapter 1), scalability was one of the key aspects considered in the development of the proposed approach. Since the proposed method relies on an object detection based system capable of identifying assembled parts, training the detection models requires data annotated at both part and assembled-object levels. However, part-level annotations are not available in the IndustReal dataset. Moreover, even if such annotations were provided, approaches relying on manually annotated part-level labels would suffer from significant scalability limitations. Whenever new assembly configurations or components are introduced, extensive and costly re-annotation would be required, severely restricting the practical applicability of the method.

To overcome this limitation, one of the main contributions of this thesis is the design of a custom synthetic dataset generation pipeline, through which training data for the detection models can be generated with minimal manual effort.

The pipeline takes as input the 3D models of the assembly states, which are inherently available prior to production since products are designed in CAD form. Using these models, the pipeline automatically generates a dataset of rendered images depicting the assembly states from multiple plausible viewpoints, balanced across the different states, and simultaneously produces part-level and assembled-object-level bounding box annotations. An overview of the pipeline is illustrated in Figure 6.1.

Since rendering, dataset composition and labeling are performed automatically, scaling the approach to novel assembly states or new parts only requires adding the corresponding 3D models to the pipeline. By rerunning the generation process, a new dataset can always be easily obtained.

The development of the pipeline consisted of three main steps. First, since the primary data required to generate the images are 3D models, models represent-

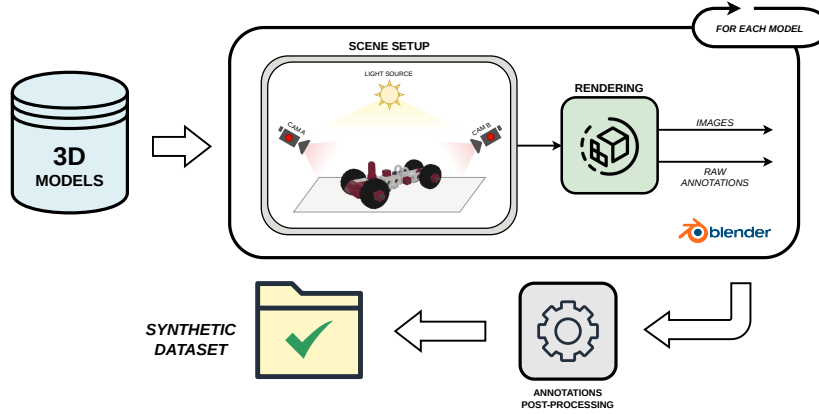


Figure 6.1: Synthetic dataset generation pipeline. Starting from a set of 3D models representing assembly states, the system automatically generates data by instantiating a scene with lighting and virtual cameras. For each camera, images and raw annotations are rendered and subsequently post-processed to obtain clean object- and part-level bounding boxes. The final dataset is produced by aggregating and splitting the generated samples according to user-defined parameters.

ing the different assembly states had to be carefully prepared for use. Second, it was necessary to define a procedure to set up a scene containing one of the models, the surrounding environment, and a camera, and then render this scene to obtain an image from the camera viewpoint together with the required bounding box annotations. Lastly, a large-scale rendering orchestration mechanism had to be implemented in order to automatically generate a complete annotated dataset with predefined properties.

Each of these steps was designed to ensure the desired characteristics in terms of realism, also with respect to the IndustReal dataset, needed to reduce the domain gap between synthetic and real data, viewpoint variability, balanced coverage of the different assembly states, and customizability of the pipeline. The following sections describe in detail how these three steps were addressed.

6.1 Assets Setup

The IndustReal dataset provides 3D models of the assembly states in the form of .fbx files. However, these models were not directly usable for the purposes of the proposed synthetic data generation pipeline, as they exhibited incorrect scales, dimensions, and object properties. For this reason, Blender¹, a powerful, free, and open-source 3D computer graphics software, was adopted to process

¹Official Blender organization website. (<https://www.blender.org>)

and adapt the models.

In particular, the 22 provided 3D models corresponding to the pre-defined assembly states were manually processed according to the following steps:

- Each asset was individually rescaled to ensure correct relative proportions between assets.
- The models were decomposed into parts and correctly named, so that each part appears as a separate object in Blender, a requirement for subsequent automatic annotation generation.
- Starting from copies of the original models, parts were selectively removed or added to create 19 additional assembly states, which are later used for generalization experiments. These novel states were also introduced by Tim J. Schoonbeek in [2]; however, since the corresponding 3D models were not publicly available, they were recreated to match the original configurations.
- The pose of each model was adjusted to obtain a natural resting configuration, ensuring that the assembled object lies on a reference plane at $z = 0$.
- When multiple physically plausible resting poses existed for a given assembly state, several model variants were created to avoid bias toward a single pose and to improve robustness.

In addition, a separate 3D model representing the supporting plane at $z = 0$, corresponding to the table visible in the real video frames, was created using a matching color and appearance.

Overall, 60 models were generated, accounting for the 22 real assembly states, 19 novel states and additional resting pose variants for both categories. A naming convention was applied: real state models use the “state” suffix, while novel models use the “inter” suffix, following [2].

6.2 Image Rendering

To define scenes in which assets could be loaded, while allowing control over camera poses and lighting conditions, and to render images for each configured camera, a rendering engine had to be selected. At the same time, the solution needed to be practical and easily manageable through scripting. For these reasons, the BlenderProc 2 library [57] was adopted.

BlenderProc 2 is a Python-compatible framework that enables automated and script-based rendering workflows and is built on top of Blender, which was already used for asset manipulation. Beyond rendering, BlenderProc 2 provides

automatic annotation generation capabilities, producing JSON files containing bounding box and segmentation annotations in COCO format. While the library supports additional output modalities, only the mentioned annotations were retained, as they are sufficient for the requirements of the proposed pipeline.

To render a single image using this library, a sequence of steps must be executed. First, the 3D models of the objects to be depicted are imported into the scene. Second, one or more light sources are configured to ensure proper illumination of the scene. Then, there is the most critical step, which is the definition of a camera pose for each image to render, as it determines the viewpoint from which the object is observed and, consequently, the visual appearance of the resulting image. Finally, rendering parameters are specified to control image properties. All steps that require further explanation are presented in detail in the following subsections.

6.2.a Light Sources

As light source, an “area” light was selected, as it provides uniform illumination over the object. The light was positioned above the scene (i.e., $x = 0$, $y = 0$, $z = 100$) and assigned a warm color temperature (i.e., $r = 1.00$, $g = 0.85$, $b = 0.70$), in order to closely match the lighting conditions present in the real recording setup used for capturing the IndustReal dataset videos.

6.2.b Camera Poses

The definition of camera poses required a more careful design, as the sampled viewpoints needed to satisfy two main criteria. First, besides being realistic, the synthetic images had to remain coherent with the real images of the IndustReal dataset. In particular, the viewpoints had to be physically plausible with respect to the head-mounted device used to record the real data, thus maintaining an egocentric perspective. Second, the sampled poses needed to exhibit sufficient variability in order to capture a wide range of viewpoints and perspectives. The camera pose sampling strategy described in this section was therefore designed to achieve an effective trade-off between realism and viewpoint diversity in practice.

To define a camera pose, both the position (x, y, z) in the world reference frame and the orientation of the camera with respect to the world frame are required. Rather than directly sampling these quantities, camera poses are specified indirectly through a set of intuitive parameters: a radius and an azimuth angle defining the camera position in the xy plane, the camera height z , a target point in the world reference frame (x, y, z) defining the camera look-at direction, and an

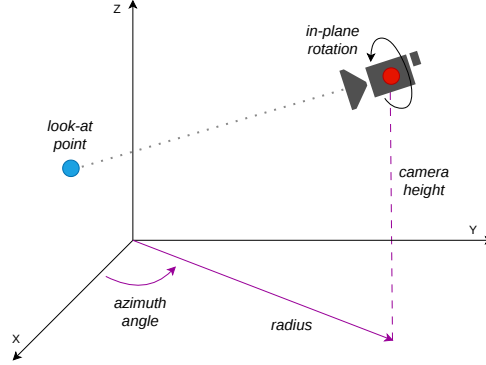


Figure 6.2: Visualization of the camera pose parameterization strategy. The camera position is defined in the xy plane through radius and azimuth, with height z , while orientation is determined by a look-at target point and an in-plane rotation (roll).

in-plane rotation value specifying the camera roll in radians. Given the radius, azimuth angle, and height, the camera position can be uniquely determined. Similarly, the camera rotation matrix is obtained from the look-at point and the in-plane rotation. This camera pose parameterization strategy is illustrated in Figure 6.2.

The sampling of these parameters is based on the Beta probability distribution, defined through the Beta function

$$B(\alpha, \beta) = \int_0^1 t^{\alpha-1} (1-t)^{\beta-1} dt, \quad \alpha > 0, \beta > 0 \quad (6.1)$$

with corresponding probability density function

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)} x^{\alpha-1} (1-x)^{\beta-1}, \quad x \in (0, 1) \quad (6.2)$$

This distribution was selected because, for $\alpha = \beta > 1$, it yields a unimodal density that favors values close to 0.5 while suppressing samples near 0 and 1. This behavior reflects the statistics of real video frames, where assembly states are most commonly observed in a canonical configuration: placed horizontally on a table in front of the operator and viewed from a reasonable distance. Consequently, most synthetic views are sampled near this default configuration, while more extreme viewpoints are generated less frequently.

The strength of this effect is controlled by the value of α (with $\alpha = \beta$), where larger values result in a stronger concentration around 0.5. In the proposed pipeline, different parameters are sampled using Beta distributions with varying

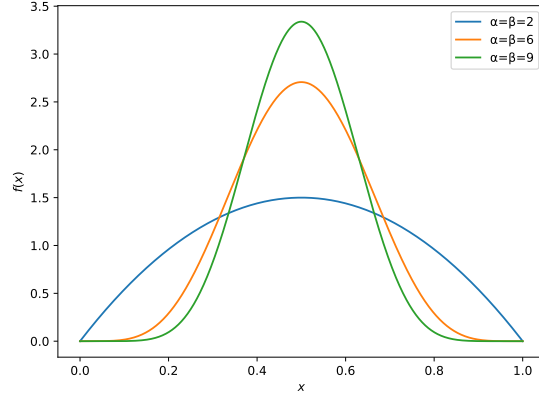


Figure 6.3: Beta probability density functions for the values of α and β adopted for camera pose generation.

concentrations, as some degrees of freedom exhibit greater variability in real scenarios than others. In practice, three values of α were selected, namely 2, 6, and 9. The corresponding probability density functions are shown in Fig. 6.3.

In practice, the sampling procedure is implemented by first defining a valid interval for each parameter. A value is then sampled from the assigned probability density function and mapped to the interval by scaling it by the interval width and adding it to the lower bound. Formally, given an interval $[v_{\min}, v_{\max}]$, the sampled value is computed as

$$v = v_{\min} + x \cdot (v_{\max} - v_{\min})$$

where $x \sim f(x; \alpha, \beta)$.

An exception to this strategy is the definition of the look-at point. Instead of directly sampling absolute coordinates, a default look-at point $(0, 0, 0.05)$, slightly above the origin, is defined, and independent offsets are sampled for each coordinate within predefined ranges. Similarly, for the radius and azimuth angle, an additional offset sampled from a separate distribution is added to the base sampled value. Although initially introduced for experimental purposes, this design choice was retained in the final pipeline as it enables finer control and improved flexibility in camera pose customization.

Table 6.1 summarizes the parameter ranges and corresponding sampling distributions used in the pipeline.

Figures 6.4a, 6.4b, 6.4c and 6.4d illustrate examples of rendered images that effectively demonstrate the diversity introduced by the camera pose sampling strategy.

In table 6.1, α denotes the shape parameter of the Beta distribution with $\alpha = \beta$,

Parameter	$v_{\min}$	$v_{\max}$	α
Radius	0.08	0.18	6
Radius offset	-0.07	0.07	2
Azimuth angle [$^{\circ}$]	0.0	180.0	6
Azimuth angle offset	-0.07	0.07	2
Height	0.15	0.22	2
Look-at x offset	-0.08	0.08	2
Look-at y offset	-0.05	0.05	6
Look-at z offset	-0.03	0.03	9
In-plane rotation	-0.08	0.08	U

Table 6.1: Sampling ranges and distribution parameters for camera pose generation. Values for $v_{\min}$ and $v_{\max}$ are expressed in meters.

while U indicates a uniform distribution. The in-plane rotation is intentionally sampled uniformly to exclude camera roll from the biased sampling strategy, reflecting the variability observed in real video frames. Figure 6.4i shows an image illustrating the in-plane rotation effect. Although subtle, the effect is noticeable by observing that the image viewpoint results from a slight counterclockwise roll of the camera.

Finally, with probability $1/2$, the sign of the y coordinate is flipped, ensuring that half of the rendered images depict the assembly from one lateral side and the other half from the opposite side, further increasing viewpoint diversity. Figures 6.4g and 6.4h show examples where the sampled camera pose is positioned on the right and left side of the assembly, respectively.

To render multiple images simultaneously, multiple camera poses can be sampled using the strategy described previously and added to the scene along with the assets and the light source.

6.2.c Camera Parameters

The images are then rendered using BlenderProc, which allows further customization. In this pipeline, images are rendered at 720p resolution (1080×720 pixels) to match the quality of the real video frames from the IndustReal dataset. Additionally, a slight motion blur is applied by setting the motion blur length to 0.005, which defines the exposure time of the virtual camera as a fraction of the time between consecutive rendered frames. This helps to more closely emulate the appearance of the real videos. Figure 6.4j is an example of image in which the motion blur effect is evident.

To further match the real data, an attempt was made to replicate the intrinsic

parameters of the camera used to capture the IndustReal videos. While the full specifications of the headset were not provided, the diagonal field of view is reported as $\theta_d = 96.1^\circ$ on the device page². Assuming an ideal pinhole camera model with square pixels and zero skew, the intrinsic matrix is given by

$$\mathbf{K} = \begin{pmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

Given the image dimensions $W \times H$, the diagonal length is

$$D = \sqrt{W^2 + H^2}$$

From pinhole camera geometry, the focal length f is computed as

$$\tan\left(\frac{\theta_d}{2}\right) = \frac{D/2}{f} \quad \Rightarrow \quad f = \frac{D/2}{\tan(\theta_d/2)}$$

Assuming a centered principal point, the offsets are

$$c_x = \frac{W}{2}, \quad c_y = \frac{H}{2}$$

With these settings, images are rendered and saved in JPEG format. Simultaneously, BlenderProc automatically generates bounding box and segmentation annotations for all assets in the scene for every rendered image.

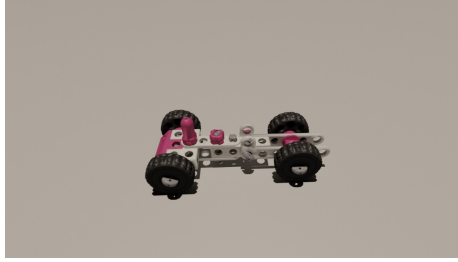
Figure 6.4 presents examples of images generated using the implemented rendering procedure. All aspects of assets setup and image rendering in the synthetic dataset generation pipeline are visible in these images. For instance, Figures 6.4e and 6.4f show different asset variants for the same assembly state, illustrating the preparation of multiple models for certain assembly states.

6.3 Dataset Organization

With the image rendering procedure defined, the pipeline execution was organized to ensure the desired balance and coverage of assembly states in the produced dataset, while viewpoint variability is enforced by the custom camera pose sampling strategy. The adopted procedure can be summarized as follows.

First, the overall size of the dataset and the proportion of real versus novel assembly states is specified. The portion of frames to render for each category

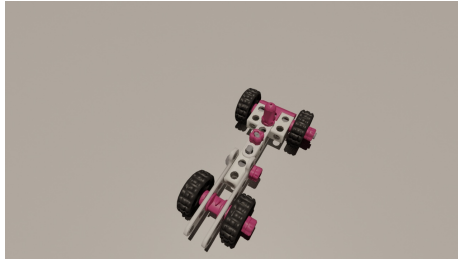
²Hololens 2 specifics. (<https://learn.microsoft.com/en-us/hololens/hololens2-hardware>)



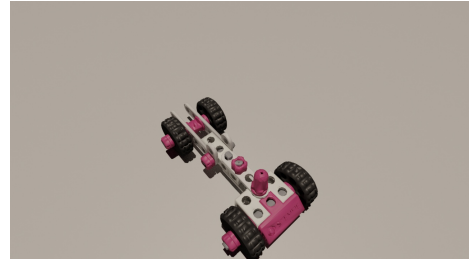
(a) state14_1d_003896.jpg



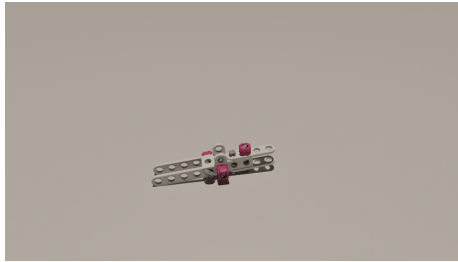
(b) state14_1d_003910.jpg



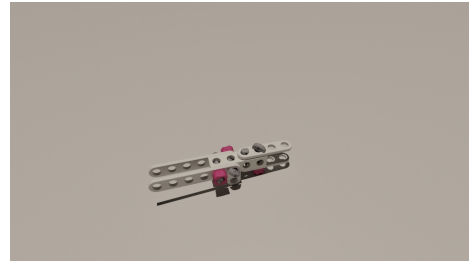
(c) state14_1d_003898.jpg



(d) state14_1d_003911.jpg



(e) state8_1f_004305.jpg



(f) state8_2c_004955.jpg



(g) inter13_1b_000517.jpg



(h) inter13_1b_000523.jpg



(i) state16_1f_009922.jpg



(j) state11_1k_013782.jpg

Figure 6.4: Examples of images obtained through the implemented synthetic dataset generation pipeline. The adopted filename format for rendered images encodes the depicted assembly state (e.g., state16, inter13), the specific asset for that state and rendering chunk (e.g., 1b, 1f, 1k, 2c), and a unique image identifier.

is then evenly divided among the assembly states within the category. For each state, this number is further split among the available assets, recalling that multiple assets may exist for a single assembly state to account for different natural poses.

Once an asset is selected, images can be rendered; however, due to GPU memory limitations, rendering is performed in chunks of customizable size rather than all at once. For each chunk, the scene is set up as previously described, and the images are rendered. After completing a chunk, the scene is cleared before loading the next set of assets, and the process continues until all frames for all assets are generated.

It should be noted that, at the time of writing, BlenderProc 2 exhibits minor bugs and suboptimal performance. To mitigate these issues, several measures were implemented. Rare failures in loading the plane were detected by checking if the background in rendered RGB images was black; in such cases, the asset loading was repeated until successful. Additionally, rendering is highly time-consuming even on high-end GPUs. To address this, it is possible to preselect subsets of assembly states and render them in parallel with different jobs across multiple GPUs. A custom script was then used to merge the resulting images and COCO annotation files into a single dataset.

The raw COCO annotations produced by BlenderProc 2 are not directly usable. Post-processing steps include removing annotations for the plane and generating bounding boxes for the entire assembled object. The latter is obtained by leveraging the segmentation of the plane: the “hole” in the plane corresponds to the full assembled object, which can then be trivially converted into a bounding box. These issues arise because, as previously anticipated, the rendering engine automatically generates annotations only for the loaded assets, which include the individual parts and the supporting plane, but not the assembled object as a whole.

Figure 6.5 illustrates how the annotations produced by BlenderProc are processed through the described post-processing pipeline to obtain bounding boxes for the assembled object, in addition to the already correct part-level annotations.

Finally, a conversion script was implemented to transform COCO annotations into YOLO Ultralytics format, enabling the training of YOLO-based object detection models.

By defining and implementing appropriate solutions for each required step, the desired synthetic dataset generation pipeline was obtained, incorporating all target properties and enabling the creation of training data for the necessary object detector models. As shown later in Chapter 8, datasets were generated

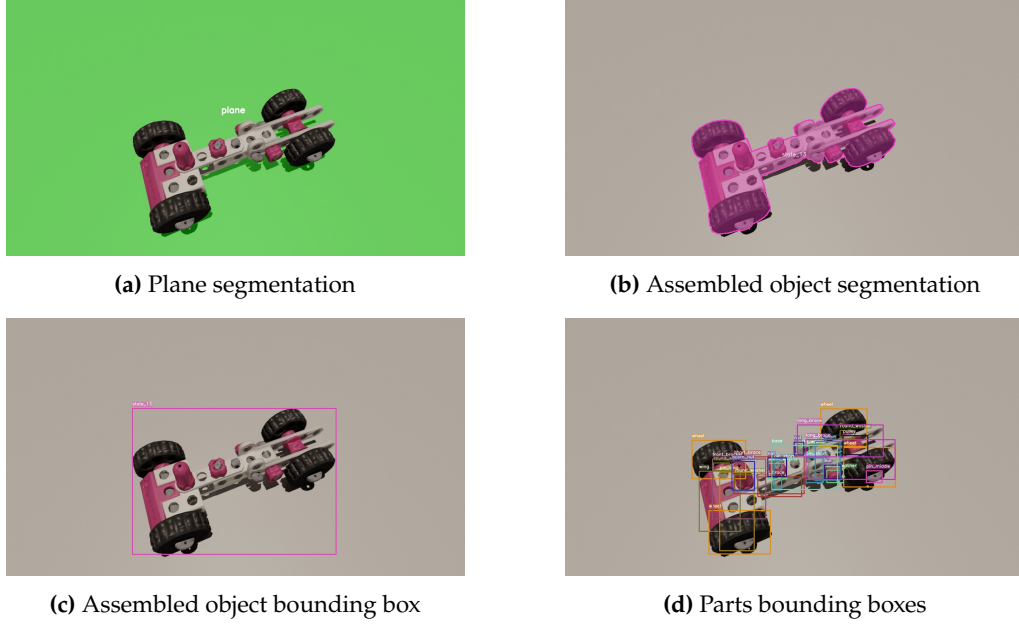


Figure 6.5: Example of BlenderProc-produced annotations post-processing performed to obtain assembled object and parts bounding boxes.

reaching up to 100K images, with approximately 2750 images per real assembly state and 2100 per novel assembly state.

Finally, it is worth noting that all parameters defined throughout the pipeline implementation are easily configurable through dedicated configuration files. This design choice makes the pipeline highly customizable and allows it to be readily adapted or reused for different requirements or application scenarios.

Sim-to-Real Adaptation

As discussed in previous chapters, the first and fundamental step of the proposed approach for the ASR task is assembled parts detection. This step is crucial for extracting structural information of assembled objects, which can be leveraged to accurately infer the corresponding assembly state.

Assembled parts detection is implemented through a two-detector system: one model for part detection and one for assembled object detection. Both models are trained via a fine-tuning strategy starting from weights pre-trained on the COCO dataset. For the system to be effective, fine-tuning must be performed on IndustReal-like data. To address this requirement, the synthetic dataset generation pipeline introduced in the previous chapter enables the creation of large-scale IndustReal-like data, along with corresponding annotations for both individual parts and assembled objects.

Although the pipeline is designed to closely replicate real video frames, discrepancies in image statistics (e.g., RGB value distributions) are unavoidable. These differences typically degrade the performance of object detection models when applied to real data. This effect is also observed in this work, where models trained on synthetic data exhibit significantly lower performance on real-world data. Quantitative results highlighting this issue are reported in Table 7.1 and are further discussed in Chapter 8.

This phenomenon is a well-known limitation in training deep learning models on synthetic data and is commonly referred to as the *sim-to-real gap*. Reducing this gap is therefore a key requirement for achieving models that generalize effectively to real-world data.

Test Data Type	Part Detector	Assembled Object Detector
Synthetic	0.8973	0.9710
Real	0.3788	0.0964

Table 7.1: Overall mAP_{50-95} performance of object detectors trained exclusively on synthetic data, evaluated on synthetic and real test sets. The substantial performance drop on real data highlights the presence of the sim-to-real gap.

7.1 Closing the Sim-to-Real Gap

As shown in previous works such as [1] and [58], model performance on real data can be significantly improved by augmenting the synthetic training set with a small fraction of manually annotated real images. Empirically, increasing the proportion of real data in the training and validation splits leads to improved performance on real data, with the largest gains observed when adding the first 1–2% of real samples. Performance improvements then gradually saturate, typically reaching a plateau around 5–10%, beyond which additional real data provides diminishing returns.

This method is effective because synthetic and real images depict the same objects under very similar configurations; the inclusion of real data mainly allows the model to learn the association between the visual properties of synthetic images and those of real images, enabling correct interpretation of equivalent structures despite differences in low-level image statistics.

To implement this method, an efficient annotation strategy was required to obtain real images with both part-level and assembled object-level annotations while minimizing manual effort. The adopted one differs between part and assembled-object cases, depending on what is already provided by the IndustrialReal dataset and what must be obtained manually.

For part-level annotations, the dataset does not already provide bounding boxes, making manual annotation from scratch impractical due to the large effort required. To address this, the part detector trained only on synthetic data was used to pre-annotate real images. The annotation process then consisted only of correcting these predictions, including adjusting bounding box position and size, fixing incorrect labels (false positives) and adding missing detections (false negatives). This process was further optimized through an iterative refinement strategy: once an initial subset of real images was corrected, it was used to partially adapt the detector to real data. The improved model was then used to pre-annotate additional images with higher accuracy, progressively reducing manual effort. Repeating this process enabled efficient large-scale annotation and ensured scalability, as previously learned parts can be automatically anno-



Figure 7.1: Comparison of real and synthetic images showing assembly state 11 from a similar viewpoint. The main difference is the presence of loose parts in the real image.

tated and only new part classes require manual labeling.

A specific consideration concerns the loose parts placed on the table around the assembled object. As shown in Figure 7.1, these components appear only in real images and are absent in the synthetic ones, where only assembled parts are present. This design choice of the synthetic dataset generation pipeline reflects the definition of an assembly state, which depends exclusively on assembled components and not on unused parts in the workspace. Therefore, explicitly modeling loose parts during synthetic data generation was unnecessary since there is no need to have a detector trained to recognize them.

As a consequence, part detectors trained only on synthetic data often fail to detect loose parts in real images. However, as already said, high detection performance on these is not needed. At the same time, completely ignoring them during real-data annotation could negatively impact training due to their visual similarity with assembled parts. For this reason, a balanced annotation strategy was adopted: false positives on loose parts were corrected, while false negatives were intentionally left unannotated. This approach prevents incorrect supervision while allowing the model to remain focused on assembled components, reducing annotation effort without compromising task-relevant performance.

Considering all aspects of the adopted annotation and training procedures for part detection, the overall approach can be effectively characterized as a weak-supervision strategy.

For assembled object annotations instead, the IndustReal dataset already provides bounding boxes for labels 1 to 23, corresponding to the 22 real assembly states and the *error state* class. The only missing category is the intermediate state (label 0), which required manual annotation. Since this involves a single large bounding box per image and only one class, the required effort was minimal.

All annotations were created using the Roboflow¹ online platform, which

¹Roboflow online platform. (<https://roboflow.com>)

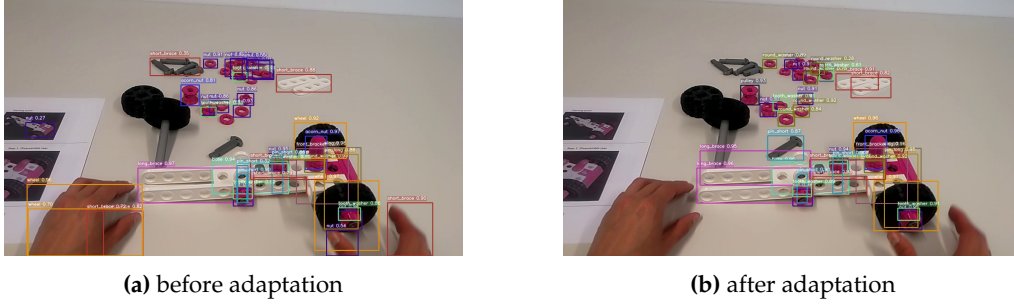


Figure 7.2: Comparison of part detection on a real image before and after closing the synthetic-to-real gap. Before adaptation, many false positives are produced (e.g., detections on the operator’s hands and duplicate predictions) along with false negatives (e.g., a missed *long brace*). After adaptation, these errors are largely mitigated, resulting in cleaner and more accurate detections.

provides efficient annotation tools, dataset management capabilities and Python API support for streamlined data upload and integration.

By adopting these model adaptation and data annotation strategies, improved versions of the object detection models were obtained, both for individual parts and for the assembled object, achieving effective performance on real video frames from the IndustReal dataset.

Figure 7.2 compares part detection results on a real image before and after closing the synthetic-to-real gap. Prior to this step, the detector produces several false positives (e.g., detections on the operator’s hands, duplicated predictions for the same component) and false negatives (e.g., the missed detection of one of the assembled *long braces*). After applying the proposed strategy, these issues are effectively mitigated, resulting in cleaner and more accurate part detections, as shown in the figure.

7.2 Detections Processing

As previously mentioned, when bridging the *sim-to-real gap*, the trained part detector generalizes to real data and becomes capable of detecting loose components as well. This behavior is undesirable in this context, as only assembled parts should be considered when inferring the assembly state. Therefore, a strategy to filter out detections corresponding to loose parts was required.

This issue was addressed by exploiting the availability of the second object detector trained to localize the assembled object as a whole. Part detections are filtered based on their spatial relationship with the assembly bounding box: a part prediction is retained only if at least 80% of its bounding box area lies within the bounding box of the assembled object; otherwise, it is discarded. Figure 7.3

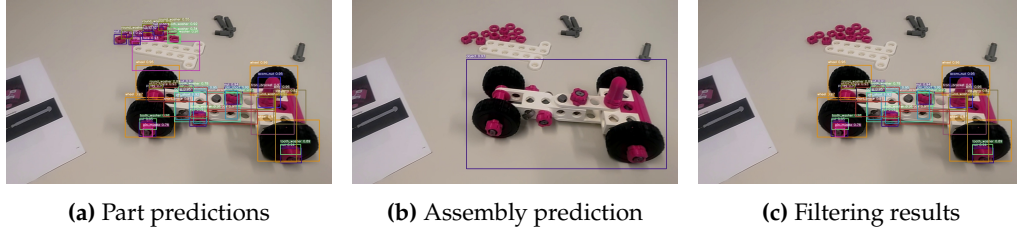


Figure 7.3: Example of step-by-step part prediction filtering procedure on a real image.

illustrate examples of predictions produced by the two detectors and show how their combination yields a filtered set of detections corresponding exclusively to assembled parts.

Several critical corner cases must be accounted for, as the filtering step may fail or return no valid predictions. This can occur for multiple reasons. In some rare *intermediate state* frames, no parts may be present on the table at all. Additionally, although infrequent, the part detector may fail to detect any components. In both cases, no part predictions are available to be retained. Similarly, if the object detector for the assembled object fails to produce a prediction, no assembly bounding box is available to filter part detections.

These cases must be handled explicitly to ensure that at least one prediction is available for every image. This requirement is essential for an assembly state recognition pipeline that relies on part-level detections, as the downstream model expects a valid input during both training and inference.

The issues were addressed as follows. If the assembled-object detector does not produce any prediction, a bounding box covering the entire image is assumed as the assembly bounding box. This choice effectively translates into retaining all part detections for that frame. Conversely, if no part detections remain after the filtering step, the assembly bounding box itself is treated as a part-level detection and assigned a fictitious 15th part category, denoted as *unknown*. This category signals to the subsequent model that the current frame depicts an undefined or ambiguous configuration. In practice, this situation predominantly occurs for *intermediate state* frames, where the assembly configuration is not well defined, making this strategy consistent with the underlying data semantics.

Algorithm 1 summarizes the complete detection processing and filtering procedure in the form of pseudocode.

Algorithm 1 Processing and Filtering of Detection Results

Inputs:

Part detections P
 Assembly detection a (may be None)
 Area overlap threshold τ

Output:

Filtered part detections $\hat{P}$

```

1: if  $a$  is None then
2:   Initialize  $a$  as a default detection
3:    $a.bbox \leftarrow (0, 0, image\_width, image\_height)$ 
4: end if
5:  $\hat{P} \leftarrow \emptyset$ 
6: for each  $p \in P$  do
7:   if  $area(p \cap a) \geq \tau \cdot area(p)$  then
8:      $\hat{P} \leftarrow \hat{P} \cup \{p\}$ 
9:   end if
10: end for
11: if  $\hat{P} = \emptyset$  then
12:    $a.class \leftarrow 'unknown'$ 
13:    $a.confidence \leftarrow 1.0$ 
14:    $\hat{P} \leftarrow \{a\}$ 
15: end if
16: return  $\hat{P}$ 
  
```

Object Detectors Evaluation

In the previous chapters, multiple approaches to address the Assembly State Recognition (ASR) task within a representation learning framework were presented. In particular, a baseline method, adopted from prior work [2], exploits visual feature extraction from video frames to learn representations. Building upon this, a novel graph embedding model operating on assembled parts graphs was introduced, together with two additional models that integrate the baseline and graph-based approach.

As discussed, all proposed methods rely on graphs constructed from assembled parts detections, obtained using two object detector models: one for individual parts and one for the assembled object. Therefore, prior to evaluating the ASR approaches, it was necessary to obtain object detectors capable of reliably operating on real images.

All required components to achieve this objective, including synthetic data generation, model selection, domain adaptation strategies and real data annotation procedures, were defined in the previous chapters. The next step was therefore to establish a consistent experimental setup, defining a default training and evaluation protocol to ensure both effective learning and fair comparison across experiments.

Training Protocol. As mentioned in Section 5.1, the Ultralytics framework was selected due to its flexibility and extensive control over training strategies. Training was performed by fine-tuning models pre-trained on the COCO dataset, rather than training from scratch. In particular, the backbone was frozen while only the neck and head were fine-tuned, a common strategy that accelerates convergence while maintaining strong performance.

Each model was trained for up to 300 epochs, with early stopping enabled

using a patience of 60 epochs to prevent unnecessary computation once performance plateaued. Since experiments involved datasets of varying sizes, both the batch size and the number of GPUs were adjusted accordingly to ensure efficient training. Specifically, the batch size was increased from 64 to 128 and 256 as dataset size grew, while the number of GPUs ranged from 1 up to 4.

The optimizer was automatically selected by the framework based on the training configuration, and in all experiments Stochastic Gradient Descent (SGD) was used. Validation was performed at the end of each epoch, and the best-performing checkpoint was automatically saved based on the highest validation mAP_{50-95} . Additional metrics, including mAP_{50} , precision and recall, were also computed in both overall and per-class form. For completeness, mAP_{50} evaluates detection performance in terms of Mean Average Precision (mAP) at an Intersection over Union (IoU) threshold of 0.50, while mAP_{50-95} averages performance over IoU thresholds ranging from 0.50 to 0.95, providing a more comprehensive assessment of localization accuracy.

Regarding data augmentation, numerous techniques are made available by Ultralytics. These are techniques that alter images or add modified versions of them to the dataset and are commonly used to improve model generalization to unseen contexts, particularly when objects may appear under varying environmental conditions. However, in the IndustReal dataset, the environment remains highly controlled and consistent, with fixed workspace geometry, lighting conditions and object proportions. Since the assembly parts exhibit precise geometries and relative configurations, preserving their visual and spatial fidelity is essential for accurate detection. For this reason, all data augmentation techniques were disabled, as they could introduce unrealistic variations and potentially degrade detection performance.

Evaluation Protocol. For object detector evaluation on validation and test sets, a custom Python script based on the `pycocotools`¹ library was adopted. This choice was made because the evaluation implementation provided by the Ultralytics framework does not strictly follow the canonical metrics definition, whereas `pycocotools` provides a standardized and widely accepted reference implementation. Using this library ensures fair, reproducible and directly comparable results. Additionally, adopting a custom evaluation script enabled full control over the evaluation procedure, allowing domain-specific and customized testing when needed.

The evaluation metrics considered were per-class and overall precision, recall, mAP_{50} and mAP_{50-95} .

¹Official `pycocotools` library GitHub repository. (<https://github.com/ppwwyyxx/cocoapi>)

With these training and evaluation protocols established, experiments could be conducted in a consistent and reliable manner. All experiments were performed using NVIDIA A40 GPUs.

8.1 Evaluation of Part Detector

To address the *sim-to-real gap* described in Chapter 7 and obtain a part detector model capable of operating on real data, the proposed approach consists of training the model on a mixture of synthetic and real images. However, the composition of this dataset cannot be arbitrary: the proportion of synthetic and real data, as well as the overall dataset size, must be carefully selected to maximize performance while limiting manual annotation effort to maintain scalability.

As a first step, the appropriate amount of synthetic data was investigated. In principle, increasing the dataset size improves model performance by exposing the model to a wider variety of examples. However, practical limitations arise from the synthetic data generation pipeline, which can produce only a finite variability of scenes. Beyond a certain size, image repetitions may occur, increasing the risk of overfitting, especially if similar samples appear across training and test sets. Furthermore, reducing the synthetic-to-real gap requires incorporating manually annotated real images, whose relative contribution decreases as the synthetic dataset grows, thereby increasing the annotation effort required to achieve effective adaptation.

The objective of this analysis was therefore to identify the optimal trade-off between synthetic dataset size and detection performance. To this end, multiple training datasets were generated, starting from 1K images and progressively expanding to 2K, 5K, 10K, 25K, 50K and 100K images. All datasets maintained a consistent 60%–40% split between real and novel assembly states and an 80%–20% train-validation split. Model evaluation was performed on a separate synthetic test set of 5K images, generated using the same state distribution and a different seed to ensure no overlap with training data.

For part detector experiments, the part-level annotations of these datasets were used, leading to the results shown in Figure 8.1.

The obtained results follow the expected trend: increasing the training dataset size leads to improved performance across all categories. The improvement is consistent up to approximately 25K images, after which the gains progressively diminish. Performance differences between classes remain evident regardless of dataset size. Parts such as *wheel*, *acorn_nut*, *front_bracket* and *long_brace* achieve high accuracy (mAP_{50-95} above 0.9) even with smaller datasets, thanks to their

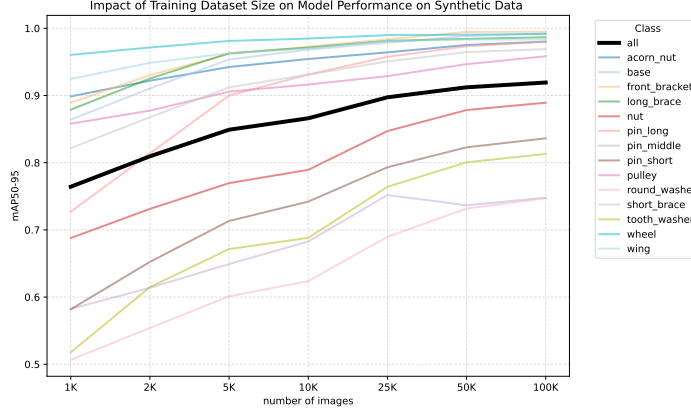


Figure 8.1: Part detector mAP_{50-95} performance (per class and overall) on a fixed synthetic test set, shown for different training dataset sizes.

distinctive shapes and clear visibility. In contrast, classes such as *pin_middle*, *round_washer* and *tooth_washer* remain more challenging (around 0.7 mAP_{50-95} even at 100K), mainly due to frequent occlusions and less distinctive visual features.

Overall, the results are satisfactory, with overall mAP_{50-95} approaching 0.9 starting from the 25K dataset size. To select the most suitable model for synthetic-to-real transfer, the 25K, 50K and 100K variants were compared in detail (Table 8.1). While larger datasets provide measurable improvements, the gains are generally limited and do not justify the significant increase in dataset size and annotation effort required for later real-data integration. Larger improvements observed for difficult classes such as *round_washer*, *tooth_washer*, *pin_short* and *nut* are likely influenced by image repetition between training and test sets, indicating overfitting rather than true generalization gains.

Considering the strong performance on synthetic data, the reduced risk of overfitting and the lower cost of real-data adaptation, the model trained on the 25K dataset emerged as the most suitable candidate for subsequent synthetic-to-real transfer. Nevertheless, a final selection based solely on synthetic performance would be incomplete. Since the ultimate objective is reliable operation on real images, an additional comparison directly on real data was required.

To this end, a dedicated real-image test set was created. A total of 100 real images were randomly selected, equally distributed across the 24 possible states (22 real states, the error state and the intermediate state) and manually annotated. The limited size of this set is due to the annotation strategy: for a fair and complete evaluation, all part-level annotations had to be exhaustively labeled, including loose parts. This full annotation process was significantly more

Class	25K	50K	100K	25K vs 50K	25K vs 100K
all	0.8973	0.9122	0.9193	-1.63%	-2.40%
nut	0.8471	0.8783	0.8893	-3.56%	-4.75%
wing	0.9815	0.9832	0.9834	-0.18%	-0.19%
acorn_nut	0.9642	0.9752	0.9802	-1.13%	-1.64%
pulley	0.9288	0.9466	0.9584	-1.88%	-3.08%
round_washer	0.6896	0.7317	0.7472	-5.75%	-7.70%
tooth_washer	0.7641	0.8006	0.8130	-4.56%	-6.02%
wheel	0.9899	0.9905	0.9917	-0.06%	-0.18%
base	0.9786	0.9885	0.9922	-1.00%	-1.38%
front_bracket	0.9836	0.9946	0.9948	-1.10%	-1.13%
short_brace	0.9507	0.9645	0.9691	-1.43%	-1.90%
long_brace	0.9813	0.9843	0.9867	-0.30%	-0.55%
pin_long	0.9576	0.9732	0.9800	-1.61%	-2.29%
pin_middle	0.7519	0.7366	0.7479	2.07%	0.52%
pin_short	0.7932	0.8229	0.8362	-3.61%	-5.15%

Table 8.1: Part detector mAP_{50-95} per class and overall performance on a fixed synthetic test set for training dataset sizes 25K, 50K and 100K.

time-consuming than the weak-supervision strategy previously described, thus constraining the number of usable images.

However, as discussed earlier, high detection performance is primarily required on assembled parts. Therefore, a single global evaluation over all parts would not be fully representative. For this reason, a double evaluation protocol was adopted for part-level detectors. Performance metrics were computed twice: first considering all ground-truth annotations and predictions (“all parts”) and then after filtering out loose parts from both (“assembled parts only”). To enable this filtering, an assembled-object bounding box was manually annotated for each image and used, as described in Section 7.2, to retain only ground-truth annotations and detections belonging to the assembled object.

This double evaluation on the real test set was conducted for the 100K, 50K, 25K and, for additional comparison, the 10K models. The obtained results are reported in Figure 8.2a for the evaluation on all parts and in Figure 8.2b for the assembled-parts-only evaluation.

From these results, several considerations can be drawn. First, model performance decreases noticeably compared to the previous evaluation on the synthetic test set: approximately $-0.25 mAP_{50-95}$ on assembled parts and up to $-0.55 mAP_{50-95}$ when considering all annotations. This degradation was expected due to the synthetic-to-real gap and the absence of loose parts in the synthetic training data.

Despite this drop, performance remains substantial. All models achieve over $0.6 mAP_{50-95}$ on assembled parts, even though no real images were used during

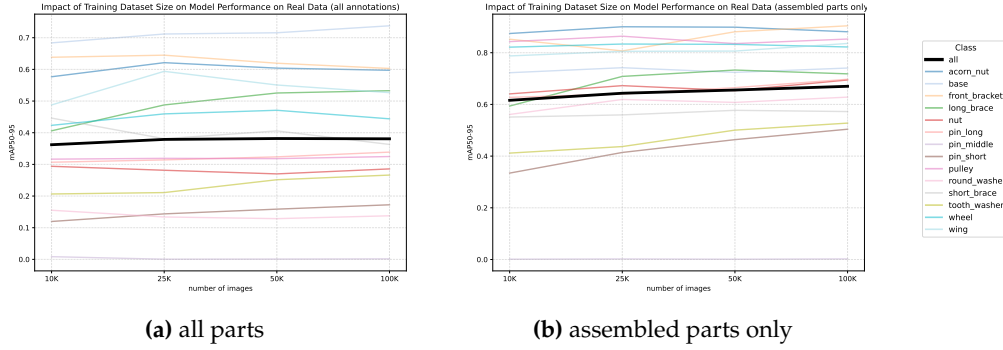


Figure 8.2: Part detector mAP_{50-95} performance (on all parts and on assembled ones only) on a fixed real test set, shown for different synthetic training dataset sizes.

training. This is a crucial result, as it confirms that the rendered synthetic images closely resemble real ones and demonstrates that effective adaptation to real data is feasible.

Furthermore, when considering all annotations, the performance differences between models trained on different dataset sizes are minimal. When restricting the evaluation to assembled parts only, a slight improvement can be observed with increasing training dataset size; however, this gain remains limited. In particular, the difference in overall mAP_{50-95} between the 25K and 100K variants is only 0.027, indicating diminishing returns from further increasing the synthetic dataset size.

Overall, the models exhibit very similar behavior on real data. Therefore, the final candidate selection is guided by the more meaningful differences observed during synthetic evaluation, as well as considerations of scalability and annotation cost. Based on these factors, the model trained on the 25K dataset was selected as the best candidate for subsequent synthetic-to-real adaptation.

The adaptation was performed by progressively adding real data in increments of 1% of the selected synthetic dataset size, corresponding to 250 images per step. Each increment was created by randomly sampling an equal number of images from the 24 assembly states available in the training split of the IndustReal dataset and annotating them at part level using the weak-supervision strategy previously defined in Section 7.1.

Experiments were conducted iteratively: after each addition of real data, the model was retrained and evaluated using the previously introduced double evaluation protocol. A gradual improvement in performance was expected, particularly for assembled parts, which represent the primary target of the detector. The adaptation process was stopped once performance saturation was reached on assembled parts.

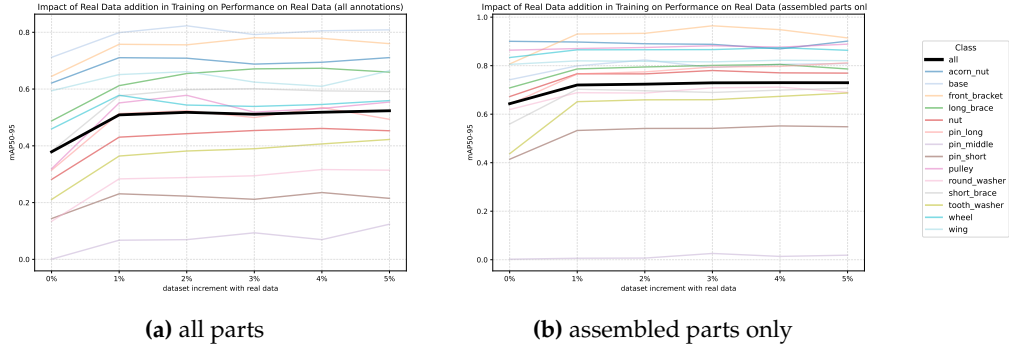


Figure 8.3: Part detector mAP_{50-95} performance (on all parts and on assembled ones only) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.

This saturation point was observed at +5% real data, corresponding to 1250 real images, which is consistent with theoretical expectations. The results are presented in Figures 8.3a and 8.3b and Tables 8.2 and 8.3 for both evaluation settings. Overall, the adaptation was successful, with mAP_{50-95} on assembled parts improving significantly from 0.64 to 0.73.

The largest performance gains were observed after adding the first 1% of real data, confirming the high effectiveness of even a small amount of real supervision. Subsequent increments provided progressively smaller improvements, until a slight decrease in assembled-parts performance was observed at 5%, indicating saturation. Conversely, performance on all parts continued to improve, including loose parts, despite their incomplete annotation in the added real data.

In conclusion, an effective part detector for real images was obtained by adapting the model trained on 25K synthetic images with the addition of only 5% real data (1250 images), achieving strong performance while maintaining scalability and limited annotation effort.

8.2 Evaluation of Assembled Object Detector

As done for the part detector, obtaining the optimal assembled object detector required determining the appropriate proportion and amount of synthetic and real training data. The first step consisted in selecting the most suitable synthetic dataset size by training models on synthetic datasets of increasing size and evaluating them on a common synthetic test set.

The same seven incremental versions of synthetic training dataset and test set previously generated for the part detector experiments were reused, this time leveraging the assembled object-level annotations instead of the part-level ones.

Class	0%	1%	2%	3%	4%	5%
all	0.3788	0.5090	0.5180	0.5112	0.5183	0.5236
nut	0.2814	0.4305	0.4428	0.4539	0.4612	0.4530
wing	0.5938	0.6510	0.6613	0.6244	0.6098	0.6643
acorn_nut	0.6212	0.7105	0.7087	0.6879	0.6944	0.7108
pulley	0.3191	0.5512	0.5780	0.5187	0.5316	0.5542
round_washer	0.1340	0.2835	0.2882	0.2946	0.3165	0.3142
tooth_washer	0.2109	0.3640	0.3817	0.3897	0.4067	0.4223
wheel	0.4594	0.5781	0.5436	0.5388	0.5457	0.5593
base	0.7117	0.7989	0.8229	0.7919	0.8053	0.8085
front_bracket	0.6449	0.7577	0.7558	0.7806	0.7786	0.7600
short_brace	0.3810	0.5765	0.5978	0.6011	0.5937	0.5913
long_brace	0.4877	0.6120	0.6547	0.6711	0.6733	0.6597
pin_long	0.3142	0.5136	0.5237	0.4996	0.5349	0.4934
pin_middle	0.0005	0.0674	0.0694	0.0935	0.0695	0.1238
pin_short	0.1437	0.2309	0.2228	0.2116	0.2354	0.2149

Table 8.2: Part detector mAP_{50-95} per class and overall performance (all parts) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.

Class	0%	1%	2%	3%	4%	5%
all	0.6432	0.7203	0.7237	0.7292	0.7299	0.7297
nut	0.6726	0.7666	0.7661	0.7800	0.7704	0.7693
wing	0.8053	0.8198	0.8180	0.8156	0.8224	0.8205
acorn_nut	0.9004	0.8975	0.8904	0.8880	0.8692	0.9012
pulley	0.8639	0.8705	0.8750	0.8822	0.8765	0.8887
round_washer	0.6191	0.6887	0.6870	0.7086	0.7116	0.6899
tooth_washer	0.4369	0.6516	0.6592	0.6597	0.6735	0.6874
wheel	0.8336	0.8653	0.8657	0.8663	0.8741	0.8634
base	0.7420	0.7992	0.8236	0.7923	0.8057	0.8089
front_bracket	0.8070	0.9305	0.9334	0.9644	0.9484	0.9145
short_brace	0.5594	0.7030	0.6966	0.6896	0.6992	0.7071
long_brace	0.7083	0.7865	0.7946	0.8011	0.8050	0.7864
pin_long	0.6403	0.7660	0.7744	0.7943	0.7970	0.8116
pin_middle	0.0021	0.0060	0.0067	0.0259	0.0137	0.0187
pin_short	0.4141	0.5328	0.5412	0.5415	0.5516	0.5481

Table 8.3: Part detector mAP_{50-95} per class and overall performance (assembled parts only) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.

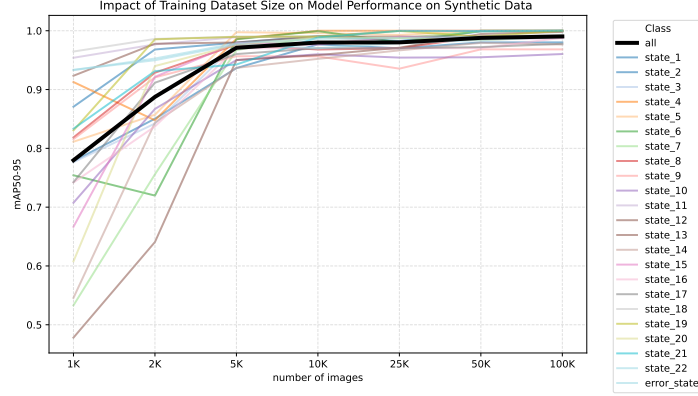


Figure 8.4: Assembled object detector mAP_{50-95} performance (per class and overall) on a fixed synthetic test set, shown for different training dataset sizes.

The obtained results are shown in Figure 8.4.

It is important to note that novel states in the synthetic data were not treated as separate classes. Novel states exist only in synthetic data and have no direct correspondence in real images; therefore, learning them as independent categories would introduce inconsistencies during real-data adaptation. Instead, all novel states were grouped under the *error_state* category defined in the IndustReal dataset (Section 4.1), which represents incorrect or invalid assembly configurations. Since novel states are generated by altering assembly assets of real states through part removal, addition, or displacement, assigning them to the *error_state* category is both conceptually and practically appropriate, allowing full exploitation of synthetic data while maintaining consistency with the real-world problem formulation.

The results show the expected performance improvement with increasing dataset size, with a steeper trend compared to the part detector. The overall mAP_{50-95} reaches approximately 0.97 already at 5K images and remains largely stable thereafter. This confirms that detecting the assembled object is significantly easier than detecting individual parts, likely due to its larger size and more distinctive structure.

To identify the best candidate, models trained on datasets of size 5K and above were compared in more detail (Table 8.4). Performance differences are minimal, with variations below 0.02 mAP_{50-95} , indicating diminishing returns from increasing the synthetic dataset size further. Since all models perform similarly on synthetic data, this evaluation alone is insufficient for final selection. Therefore, as done for the part detector, the final candidate selection was deferred to an additional evaluation on real data.

Class	5K	10K	25K	50K	100K
all	0.9710	0.9798	0.9802	0.9879	0.9903

Table 8.4: Assembled object detector overall mAP_{50-95} results on a fixed synthetic test set for training dataset sizes from 5K to 100K.

To create the real test set, a procedure similar to the one adopted for the part detector was followed. Images were sampled from the IndustReal test split, forming a larger dataset of 1000 images. The sampling aimed to equally represent the 24 possible states; however, some states were underrepresented or absent in the original test set. To compensate, the remaining images were uniformly sampled from the available real-state categories until the target size was reached.

Compared to the part detector real test set, this dataset is ten times larger. This was feasible because assembled object bounding box annotations are already largely available in the IndustReal dataset, and only intermediate-state annotations needed to be manually added, significantly reducing the annotation effort.

All trained models were evaluated on this real test set and the results are presented in Figure 8.5. Performance is reported in terms of mAP_{50} rather than mAP_{50-95} . This choice is motivated by the fact that the assembled object annotations provided in the dataset are imprecise at the bounding box level, although the class labels are correct. Therefore, using a more permissive metric is appropriate, as both training and evaluation rely on annotations with limited localization precision. Moreover, the assembled object detector is used only to filter assembled parts and the filtering procedure allows for spatial tolerance between predicted and ground truth bounding boxes (see Section 7.2).

The most evident result is the very low performance across all models, with overall mAP_{50} values below 0.1 even in the best case, corresponding to the 25K model. This highlights a significantly larger synthetic-to-real gap for assembled object detection compared to part-level detection when trained exclusively on synthetic data.

Considering both synthetic and real test set results, the 25K model was selected for adaptation. It achieves among the best performance on the synthetic test set and the highest performance on the real test set, despite the overall low absolute metric values.

As in the previous experiments, real annotated data chunks were iteratively added to the synthetic dataset and evaluation on the real test set was performed after each addition. This process was continued until saturation was

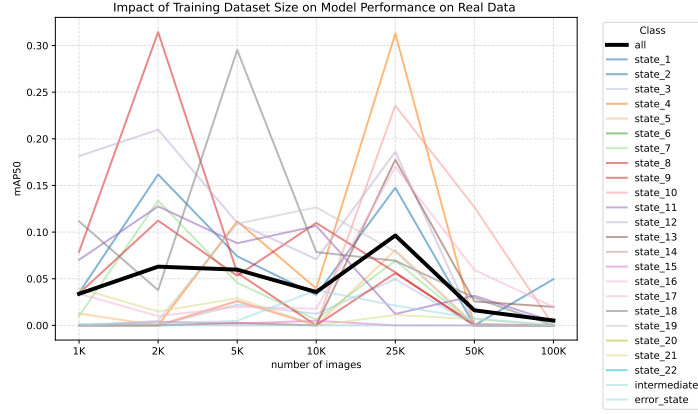


Figure 8.5: Assembled object detector mAP_{50} performance (per class and overall) on a fixed real test set, shown for different training dataset sizes.

fully reached, meaning that further additions of real data did not result in any measurable performance improvement.

The chunks were obtained following the same procedure used to construct the real test set. Each chunk had a size corresponding to 2% of the synthetic dataset, i.e. 500 images, and was sampled from the training split of the IndustReal dataset. The larger chunk size compared to the part detector experiments was motivated also in this case by the relatively easier acquisition of annotations for assembled objects.

Saturation was confirmed at the 10% addition level and the obtained results, in terms of mAP_{50} , are reported in Figure 8.6. From these results, several observations can be made. First, although the initial synthetic-to-real performance gap was significant, it was largely reduced with the first 2% addition of real data, with performance saturating shortly afterward. Similarly to the part detector experiments, the largest performance improvement was achieved with the first addition of real data.

Although performance had already started to slightly decrease at the 4% addition level, experiments were continued to ensure that saturation had truly been reached. Once the 10% addition level was reached, performance was verified to no longer improve and experimentation was therefore stopped. Table 8.5 reports the overall mAP_{50} performance values on the real test set for all evaluated data addition levels.

Based on these results, an effective assembled object detector for real images was obtained by adapting the model initially trained on 25K synthetic images with the addition of 10% real data (2500 images). This approach achieved strong performance while preserving scalability and requiring only limited additional

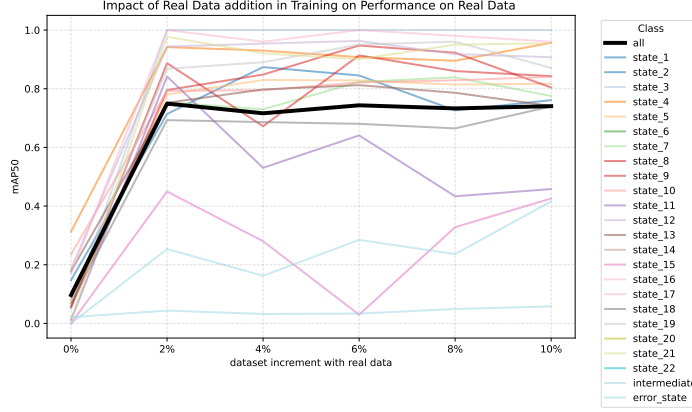


Figure 8.6: Assembled object detector mAP_{50} performance (per class and overall) on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset

Class	0%	1%	2%	3%	4%	5%
all	0.0964	0.7493	0.7163	0.7435	0.7329	0.7406

Table 8.5: Assembled object detector overall mAP_{50} results on a fixed real test set, shown for different amounts of real data added to the synthetic training dataset.

annotation effort, confirming the effectiveness of the proposed synthetic-to-real adaptation strategy in this scenario as well.

8.2.a Class Agnostic Assembled Object Detector

The obtained object detector model is not only capable of locating the assembled object in the video frames, but it can also associate it with an assembly state label among the 24 possible ones. However, this capability is not strictly required for the purposes of this work, as assembled object detection is used solely to distinguish assembled parts from loose ones.

An alternative approach would therefore consist of treating all assembled objects as a single generic category, denoted as *object*, rather than distinguishing between specific assembly states. This simplification could potentially improve localization performance, as the model would not need to solve the additional classification task of identifying the exact assembly state.

To validate this hypothesis, an experiment was conducted in which both the training dataset, composed of 25K synthetic images and 2500 real images, and the real test set of 1000 images were modified so that all assembled object annotations were assigned to the single generic category *object*. The model was then retrained

Detector Type	Recall ₅₀
Assembly-level	0.9220
Generic	0.9790

Table 8.6: Generic assembled object detectors Recall₅₀ results on a fixed real test set.

on this modified training dataset and evaluation was performed on the modified test set.

To enable a fair comparison, predictions from the original multi-class model were post-processed so that all predicted classes were mapped to the generic *object* category. As evaluation metric, Recall at an IoU threshold of 50% was adopted. This metric emphasizes the ground truth perspective, highlighting how often the model fails to detect an assembled object, while the 50% IoU threshold was selected for the same reasons brought in previous experiments.

The obtained results are reported in Table 8.6. As shown, both models achieve excellent performance as assembled object localizers, with the trained class agnostic model showing a slight improvement of approximately +0.057 in Recall. For this reason, the final model selected for the part filtering step proposed in this thesis is the retrained version that predicts the single generic *object* category.

8.3 Assembled Object Detector for ASR

Although the assembly-level detection capability was ultimately not used in the final pipeline, the availability of such a model enables another meaningful experiment. A possible approach to address the Assembly State Recognition (ASR) task is to use an object detector capable of both locating and, most importantly, labeling assembled objects in video frames, and to use it as an image classifier to assign assembly state labels.

An effective and scalable strategy to obtain such a detector is precisely the one previously adopted: training on a mixture of easily obtainable synthetic data, generated through the customizable synthetic dataset generation pipeline and a limited amount of real annotated data. This approach allows training a model capable of associating assembly state labels while maintaining scalability and limited annotation effort.

The ASR capabilities of this approach were evaluated by using the assembly-level detector as an image classifier on the IndustReal dataset test set. Standard precision, recall and F1-score metrics were computed for each assembly state appearing in the dataset, along with micro- and macro-averaged versions. These averages were computed both considering all states and considering only real

Assembly State	Precision	Recall	F1-Score
intermediate state	0.985	0.790	0.877
state 1	0.786	0.677	0.728
state 3	0.253	1.000	0.404
state 4	0.202	0.969	0.335
state 5	0.631	0.612	0.622
state 7	0.386	0.591	0.467
state 8	0.576	0.836	0.682
state 9	0.545	0.663	0.598
state 10	0.232	0.764	0.356
state 11	0.397	0.358	0.376
state 12	0.162	0.889	0.274
state 13	0.547	0.841	0.663
state 15	0.205	0.421	0.276
state 17	0.680	0.950	0.793
state 18	0.451	0.699	0.548
state 19	0.361	0.830	0.503
state 21	0.270	0.927	0.418
state 22	0.467	0.020	0.038
error state	0.085	0.079	0.082
Macro Avg (All)	0.343	0.538	0.377
Micro Avg (All)	0.774	0.774	0.774
Macro Avg (Real)	0.325	0.548	0.367
Micro Avg (Real)	0.450	0.692	0.546

Table 8.7: Assembled object detector performance, in terms of precision, recall and F1-score, on the IndustReal test set when used as image classifier.

assembly states (from 1 to 22). The obtained results are reported in Table 8.7.

Several important observations can be made from these results. First, while the detector can technically be used for ASR, the performance is particularly low. Only in a few cases does the F1-score exceed 0.7, and excluding the *intermediate state*, this occurs only for two real states (1 and 17). For all other states, performance is significantly lower, reaching a minimum F1-score of 0.038 for state 22. Since the dataset is dominated by frames depicting the *intermediate state*, macro-averaged metrics provide a more meaningful overall assessment. Considering all states, a macro F1-score of 0.377 is obtained, which further decreases to 0.367 when considering only real states. These results indicate that the approach is unreliable for consistent assembly state recognition.

Furthermore, the approach shows very poor robustness to error conditions. Performance on the *error state* class is extremely low, with an F1-score of only 0.082. This behavior is expected, since grouping all possible error configurations into a single class does not correspond to a coherent visual category that a detector can reliably learn, similarly to what occurs with the *intermediate state* class. In practice, the model can only learn to assign such labels as a form of fallback prediction when it is unable to confidently recognize a specific assembly state. The relatively high performance observed for the *intermediate state* class

suggests that the model adopts this behavior for that category. However, since such a default strategy can effectively apply to only one class, the same effect is not observed for the *error state* category.

Another critical limitation is the complete lack of generalization capability. The model is inherently restricted to the 24 assembly states seen during training and cannot be extended to recognize unseen states. This limitation is intrinsic to object detection approaches, which rely on predefined discrete categories.

For these reasons, a Representation Learning approach is adopted to improve performance, robustness and generalization to unseen assembly states.

Assembled Parts Graph Embeddings Evaluation

Having obtained the required object detectors, the proposed pipeline can be applied to extract assembled parts graphs from images. This enabled the conversion of each IndustReal sample into graph representation, allowing the training and evaluation of the proposed graph-based approaches and their comparison with the baseline method. This, in turn, enabled the assessment of representation quality in terms of the newly introduced spatial structural features extractable from the graphs and the visual features exploited by the baseline.

This chapter first presents the general training and evaluation protocols adopted for all representation learning approaches. Subsequently, a series of experiments is described in detail, highlighting, respectively, their performance on assembly states seen during training, their generalization to previously unseen states and their robustness to erroneous states.

Training Protocol. The model training protocol is defined by several factors, including batch composition, loss function behavior and learning rate management. In order to effectively train the representation learning methods, a controlled batch composition strategy was adopted. A purely random sampling strategy cannot guarantee sufficient diversity within batches, especially considering that intermediate states constitute the majority of the dataset. Moreover, without an adequate presence of real assembly states, the loss function would not be fully exploited. For these reasons, batches were manually constructed.

First, a fixed number of real assembly state classes to be included in each batch was defined. This ensures a constant and controlled presence of real states during training. Specifically, this number was set to 15. Therefore, at each batch creation step, 15 classes are randomly sampled from the 22 real assembly state classes, and samples belonging to those classes are used to populate the batch.

Second, both real and synthetic images were used jointly during training. For each sampled class, a fixed number of samples per domain was defined. This number was set to 8 for real images and 8 for synthetic images. As a result, each sampled class contributes 16 samples to the batch, equally divided between real and synthetic domains. This strategy serves two main purposes. First, it acts as a form of augmentation, improving generalization and helping the model learn representations that are robust across domains. Second, some assembly states do not appear in the real training split of the IndustReal dataset; in such cases, samples for those states are drawn exclusively from the synthetic dataset. The synthetic dataset used in this work is the custom-generated one of size 100K images, selected for its large variability.

Finally, samples corresponding to the “intermediate state” class were also included. Since this class is highly represented in the dataset and does not belong to the set of real assembly states, a fixed number of intermediate-state samples was added independently of the class sampling procedure. This number was set to 64 samples per batch.

As a result, each batch has a fixed size of: $15 \times (8 + 8) + 64 = 304$ samples.

Since batches are manually constructed, this sampling procedure does not guarantee complete coverage of the dataset within a standard epoch definition. Therefore, an epoch was defined as the completion of 100 consecutive batches. All models were trained for a maximum of 200 epochs, with an early stopping criterion set to 30 epochs without improvement.

The adopted loss function is the SupCon loss with ISIL modification presented in Section 4.2.b, which in practice was used with a temperature parameter of 0.07.

Regarding learning rate management, a staged scheduling strategy was implemented. During the first 15 epochs, a linear warmup phase was applied, gradually increasing the learning rate from 10^{-5} to 10^{-4} . This phase stabilizes early training dynamics and prevents abrupt updates in the embedding space.

After the warmup phase, a cosine annealing schedule with warm restarts was adopted. The optimizer used throughout training was AdamW, with a base learning rate of 10^{-4} and a weight decay factor of 10^{-4} . The cosine annealing schedule was configured with a restart period of 40 epochs, allowing the learning rate to periodically recover and facilitating improved exploration of the loss landscape.

Evaluation Protocol. As for evaluating the representation learning models, an appropriate evaluation protocol must be defined. This is not trivial, as the model outputs are embeddings rather than direct assembly state predictions. Therefore, evaluation must reflect how the models are actually intended to be

used to solve the Assembly State Recognition (ASR) task.

In practice, given an input image or an assembled-parts graph, the corresponding embedding (query) is obtained through model inference. This query embedding is then compared against a reference set of embeddings that have been previously computed and labeled. By identifying the most similar reference embedding, the label of that reference sample can be assigned to the query. In the adopted protocol, the reference set corresponds to the train split of the IndustReal dataset, while evaluation is performed on the validation and test splits. Cosine similarity is used as the distance metric in the embedding space.

Considering only the closest reference sample, standard classification metrics can be computed: Precision ($P@1$), Recall ($R@1$) and F1-score ($F1@1$). These metrics directly reflect the expected real-world usage scenario, where a single assembly state prediction is required for each sample.

However, since the approach is based on representation learning, it is also important to evaluate the global structure of the learned embedding space, i.e., how well samples of the same class are clustered together. This requires a metric that accounts not only for the nearest neighbor but for the overall ranking of relevant samples.

For this purpose, the Mean Average Precision at R ($mAP@R$) metric, introduced in [59], was adopted. This metric combines the concepts of Mean Average Precision and R -precision. R -precision is defined as follows: for each query, let R be the total number of reference samples belonging to the same class as the query. Among the R nearest retrieved samples, let r be the number of correctly retrieved samples (i.e., same class). The R -precision score is then $\frac{r}{R}$.

One limitation of R -precision is that it does not account for the ranking of correct retrievals. Therefore, $mAP@R$ is preferred. It is defined as Mean Average Precision computed over the top R nearest neighbors. For a single query:

$$mAP@R = \frac{1}{R} \sum_{i=1}^R P(i) \quad (9.1)$$

$$P(i) = \begin{cases} PR@i & \text{if the } i\text{-th retrieval is correct} \\ 0 & \text{otherwise} \end{cases} \quad (9.2)$$

This metric is more stable than $R@1$, rewards well-clustered embedding spaces and reflects system precision across different recall levels within the top R retrieved samples.

However, $mAP@R$ as defined considers all classes. This is not fully aligned with the adopted training strategy. Due to the ISIL modification of the loss function, the *intermediate state* class is excluded from positive pairs during training,

meaning that the model is not encouraged to cluster its samples together in the embedding space. Therefore, including this class in the clustering metric would artificially penalize the evaluation.

For this reason, a modified metric denoted as $mAP@R(+)$ is introduced, computed by considering only the well-defined assembly states. This allows for a fair evaluation aligned with the training objective. For completeness, $mAP@R(-)$ is also computed, considering only the *intermediate state* class. Although a high value of this metric is not explicitly desired, a very low value could indicate that intermediate-state embeddings are being mixed with regular assembly states, thus providing useful diagnostic insight.

It must be noted that, for computational efficiency reasons, only one out of every four *intermediate state* samples was considered when computing embeddings from every dataset split. As previously discussed, *intermediate state* samples constitute the largest portion of the dataset, making it impractical to include all of them in the evaluation. This subsampling strategy significantly reduces computational load while still preserving a sufficiently representative distribution of *intermediate state* embeddings for reliable evaluation.

Even after this subsampling, an inherent class imbalance between assembly states remains in the evaluated samples. Therefore, when computing overall metrics, macro-averaged versions were used instead of micro-averaged ones. This choice ensures that the reported metrics are more representative of the model performance across all assembly states, preventing dominant classes from disproportionately influencing the final evaluation results.

During training, model evaluation on the validation set was performed after each epoch. The models achieving the best results in terms of $F1@1$ and $mAP@R(+)$ were saved separately. Consequently, two checkpoints were retained for each training run, respectively, one corresponding to the best performance in terms of ASR capability, and another corresponding to the best assembly-state clustering performance.

With these training and evaluation protocols established, experiments could be conducted in a consistent and reliable manner. All experiments were performed using a single NVIDIA A40 GPU.

9.1 Performance on IndustReal Test Set

All four ASR models (baseline, graph-based, multimodal naive, and multimodal gated) were trained following the defined training and evaluation protocols. After training, the checkpoints achieving the best performance in terms of $F1@1$

Model	$P@1$	$F1@1$	$mAP@R$	$mAP@R(+)$	$mAP@R(-)$
Baseline	0.8454	0.4985	0.7773	0.5845	<u>0.8388</u>
Graph-based	0.7719	0.4157	0.4346	0.4303	<u>0.4151</u>
Multimodal - Naive	0.8510	<u>0.5164</u>	<u>0.7790</u>	0.6143	0.8315
Multimodal - Gated	<u>0.8477</u>	0.5329	0.8054	<u>0.5956</u>	0.8790

Table 9.1: Model performance on Assembly State Recognition using the IndustReal dataset test set, compared across multiple evaluation metrics. The best results are shown in bold, and the second-best results are underlined.

on the validation set were selected and evaluated on the IndustReal test set. Each experiment was repeated five times using different random seeds, so that each training and testing cycle was unique. This procedure enables a reliable estimation of model performance. The final results, averaged over the five runs for each model, are reported in Table 9.1.

To further support and interpret the quantitative results, the unsupervised UMAP [60] dimensionality reduction technique was applied to visualize the embedding spaces learned from the IndustReal train set. The resulting projections are shown in Figure 9.1. In these plots, embeddings corresponding to well-defined assembly states are represented as colored dots (each color indicating a different class), while *intermediate state* samples are shown as gray crosses.

Several observations can be drawn from the results. First, all approaches achieve considerable performance in Assembly State Recognition, with maximum $P@1$ and $F1@1$ values of 0.8510 and 0.5329 in the best cases, respectively. This confirms that a representation learning approach to ASR is not only feasible but also effective. The adopted feature projection layer, combined with the SupCon loss and the ISIL modification, proved to be appropriate design choices, enabling the learning of a highly representative embedding space tailored to the IndustReal domain.

In particular, the ISIL modification behaves as theoretically expected. By excluding *intermediate state* samples from the set of positive pairs in the loss computation, embeddings of this class are not forced into compact clusters, while still being separated from well-defined assembly states. This behavior is clearly visible in all embedding space visualizations (Figure 9.1), where intermediate samples form a large, non-compressed cloud, distinct from the compact colored clusters of well-defined states. This is also reflected in the metrics: in the best cases, $mAP@R$ reaches 0.8054, $mAP@R(+)$ 0.6143 and $mAP@R(-)$ 0.8790. These values indicate that meaningful reference embeddings are learned for both well-defined and intermediate states. It must be reminded that the latter are particularly challenging, as query embeddings belonging to such class may

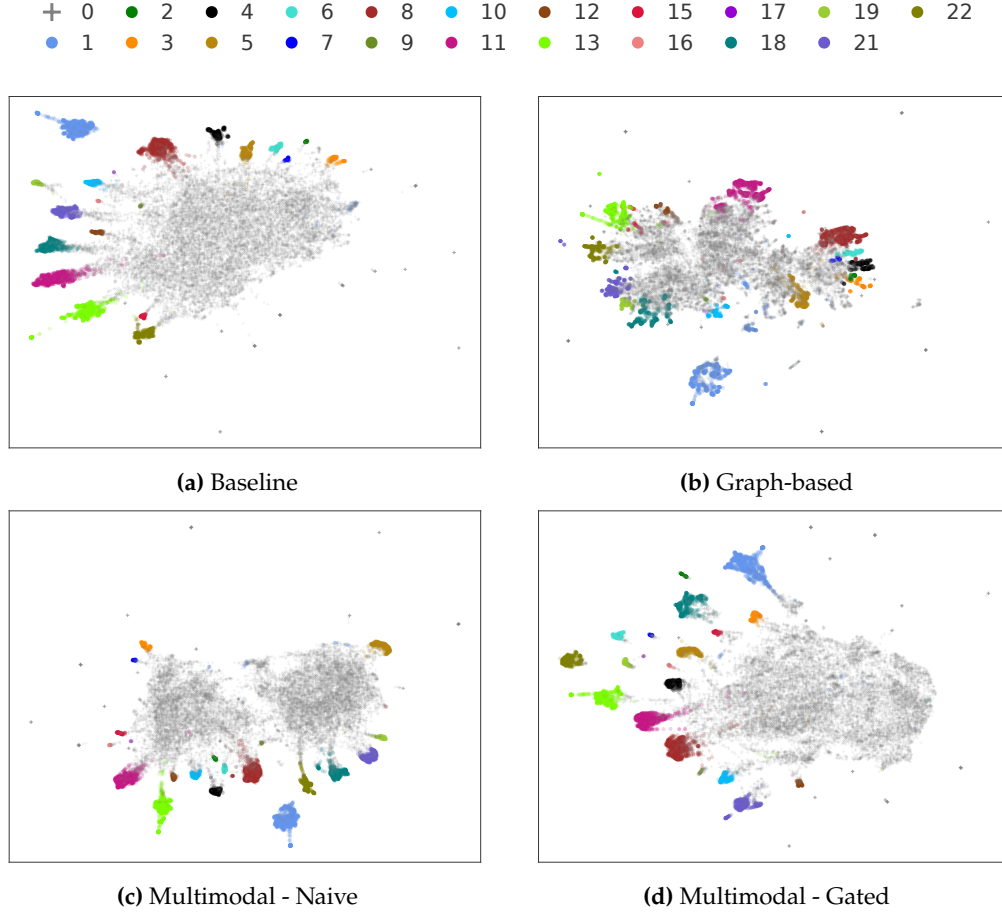


Figure 9.1: UMAP visualization of the embedding space learned by the four Assembly State Recognition models on the IndustReal training set. Intermediate states are depicted as gray crosses, and defined states as colored dots according to the legend.

represent transitions between well-defined assembly states that were not explicitly observed during training. These findings were previously reported in [2], and the approaches introduced in this thesis maintain and extend that effectiveness, demonstrating compatibility with the newly introduced assembled-parts graph representations.

Comparing the new approaches to the baseline provides insight into their respective contributions. The graph-based approach achieves solid but lower performance than the baseline, with a -15.6% drop in $F1@1$ and a -26.4% drop in $mAP@R(+)$. The most significant decrease appears in $mAP@R(-)$, with a -50.5% variation. This indicates that the assembled-parts detection graphs constitute meaningful and informative inputs for ASR. Even when relying solely on graph features, based on bounding box geometry, detection classes and con-

fidences, and processed through a GNN-based feature extractor, performance remains reasonably close to the baseline in terms of $F1@1$. However, the quality of the learned clusters is reduced. As visible when comparing the baseline embedding space (Figure 9.1a) with the graph-based one (Figure 9.1b), clusters of well-defined states are more compressed and better separated in the baseline case, whereas in the graph-based setting they are closer to each other and to the *intermediate state* cloud. This partial overlap explains both the moderate drop in $F1@1$ and the larger decrease in clustering metrics. As discussed in Chapter 5.4, this behavior is consistent with dataset labeling limitations, where correctly assembled states are labeled as *intermediate* when operators' hands, which are not detectable by the graph-based approach, are present in the frame, thereby limiting the full exploitation of the graph-only formulation.

The multimodal approaches were introduced precisely to mitigate this limitation. The naive fusion strategy, which combines image and graph features through simple concatenation, produces clear improvements over the baseline, with a +3.6% increase in $F1@1$ and a +5.1% improvement in $mAP@R(+)$.

The gated multimodal approach, adopting a more refined fusion mechanism, achieves even higher gains in $F1@1$, with a +6.7% improvement over the baseline. Interestingly, although the improvement in $mAP@R(+)$ is more modest (+1.9%), the variation in $mAP@R(-)$ is more favorable: -0.9% for naive fusion versus +4.8% for gated fusion. This suggests that, while the naive fusion yields slightly tighter clusters for well-defined states, the gated fusion better separates well-defined states from intermediate ones. This effect is visible in the embedding plots (Figures 9.1c and 9.1d), where the colored clusters in the naive fusion setting lie closer to the intermediate cloud than in the gated version.

Overall, the multimodal approach with gated fusion achieves the best performance on the IndustReal test set in terms of $F1@1$, making it the most effective solution for practical Assembly State Recognition. Moreover, even when not ranking first in certain auxiliary metrics, it consistently remains among the top-performing methods, being slightly outperformed by the naive fusion approach only in $P@1$ and $mAP@R(+)$ by marginal differences. This confirms the gated multimodal strategy as the most balanced and robust approach among those evaluated.

Moreover, these results highlight the difference in effectiveness between representation learning-based ASR approaches and classification-based ones. Comparing the best-performing representation learning model developed in this work, the multimodal approach with gated fusion, with the assembled-object detector used as an image classifier (presented in Section 8.2), a clear performance gap emerges. Using comparable evaluation metrics ($F1@1$ for the repre-

sensation learning approach and macro-averaged F1-Score for the classifier), the former achieves a performance improvement of +29.3%, from 0.3770 to 0.5329. This result demonstrates that representation learning-based solutions not only provide advantages in terms of generalization and robustness, but can also outperform classification-based methods even on the recognition of previously seen assembly states, the only aspect where classification approaches are expected to perform well.

9.2 Generalization to Unseen States

Other than evaluating the various methods on the same assembly states seen during training, it is also meaningful to test them on samples depicting unseen assembly states, in order to assess their generalization capabilities, one of the most desirable characteristics of an Assembly State Recognition solution.

To evaluate this aspect, models were tested in a realistic scenario in which ASR must be performed on new assembly states. In such a setup, labeled data is required to create a reference set of embeddings that can be used to label query samples. For this purpose, the custom synthetic dataset was used. As previously described, a large portion of this dataset consists of images depicting novel assembly states obtained by manually modifying the original 22 well-defined ones. These correspond to 18 new assembly states that were not seen during model training. For each of these 18 classes, 120 samples were selected, with the first 100 used as reference samples and the remaining 20 used as queries.

All samples were processed by the models to generate embeddings, and each query embedding was labeled by comparing it against the full set of reference embeddings using cosine similarity. Model performance was evaluated using: Precision ($P@1$), F1-score ($F1@1$) and $mAP@R$ metrics. Since all evaluated samples correspond to well-defined assembly states and no intermediate-state samples are involved, the distinction between $mAP@R(+)$ and $mAP@R(-)$ is not required.

The obtained results are reported in Table 9.2. Additionally, to better interpret the results, UMAP dimensionality reduction was applied to the embeddings of all available synthetic samples depicting the 18 novel assembly states. The resulting visualizations are shown in Figure 9.2.

From these results, several important observations can be made. First, generalization to unseen assembly states is clearly possible, with maximum values of 0.5925 in $mAP@R$, 0.9636 in $P@1$ and, most importantly, 0.8276 in $F1@1$. However, there are substantial differences between the approaches, highlighting significant variations in their generalization capabilities.

Model	$P@1$	$F1@1$	$mAP@R$
Baseline	0.6125	0.3712	0.1930
Graph-based	0.9636	0.8276	0.5925
Multimodal - Naive	0.7955	0.5468	0.3944
Multimodal - Gated	<u>0.8994</u>	<u>0.7407</u>	<u>0.5133</u>

Table 9.2: Model performance on Assembly State Recognition evaluated on additional synthetic images depicting assembly states unseen during training, compared across multiple evaluation metrics. The best results are highlighted in bold, and the second-best results are underlined.

Starting from the baseline, the model shows poor generalization performance, with large drops compared to its results on the IndustReal test set: -27.5% in $P@1$, -25.5% in $F1@1$ and -75.2% in $mAP@R$. This indicates that the model fails to properly distinguish embeddings of the new assembly states, resulting in significant overlap between clusters. This behavior is reflected in the very low $mAP@R$ value and is clearly visible in the corresponding embedding space visualization (Figure 9.2a), where clusters of visually similar states, consecutive ones, significantly overlap.

In contrast, the graph-based approach exhibits a remarkable improvement over the baseline, with increases of $+57.3\%$ in $P@1$, $+122.9\%$ in $F1@1$ and $+207.1\%$ in $mAP@R$ relative to the baseline. In this case, embeddings of different assembly states are clearly separated, as shown in Figure 9.2b, resulting in strong classification performance. Although some clusters exhibit non-homogeneous shapes or are split into subclusters, reducing overall clustering compactness, the separation between classes remains clear and effective.

This strong performance, exceeding even the graph-based model’s results on the IndustReal test set, can be explained by several factors. First, this evaluation does not include intermediate states, eliminating the labeling ambiguity that previously hindered performance. Second, all evaluated samples are synthetic, and therefore free from real-world noise such as loose parts, hands, or occlusions. This allows the object detector to operate with higher precision, producing more accurate and representative graphs, which in turn improves embedding quality and clustering performance.

These results confirm that the proposed graph-based formulation provides a highly expressive and generalizable representation for Assembly State Recognition, particularly when reliable part detections are available and class definitions are well separated.

The multimodal approaches achieve intermediate performance between the baseline and graph-based models, as expected. The naive fusion approach

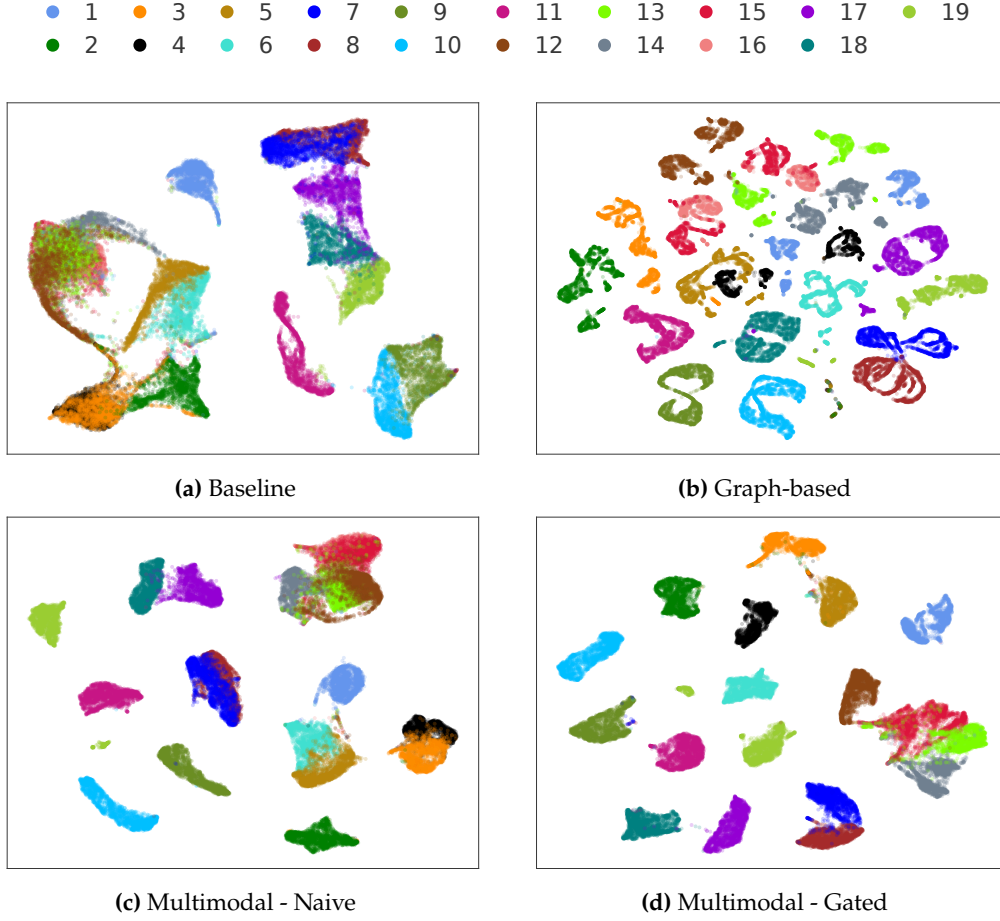


Figure 9.2: UMAP visualization of the embedding space generated by the four pre-trained Assembly State Recognition models from synthetic images depicting unseen assembly states. Embeddings corresponding to the novel assembly states are represented as colored dots according to the legend.

performs closer to the baseline, with differences of $+0.1756$ and -0.2808 in $F1@1$ relative to the baseline and graph-based models, respectively. This indicates that the model relies more heavily on image features, limiting its ability to fully exploit the generalization capabilities of the graph-based representation.

In contrast, the multimodal gated fusion approach performs much closer to the graph-based model, with differences in $F1@1$ of $+0.3695$ relative to the baseline and only -0.0869 relative to the graph-based model. This demonstrates that the gated fusion mechanism enables the model to more effectively balance and integrate image and graph features, allowing it to leverage the strengths of both modalities.

Embedding space visualizations further support these observations. In the

multimodal approaches (Figures 9.2c and 9.2d), clusters are more homogeneous in shape than in the graph-based case, but exhibit some overlap, particularly in the naive fusion model. This overlap is reduced in the gated fusion approach, reflecting its stronger generalization performance.

Overall, the best model in terms of generalization capabilities is the graph-based approach based solely on assembled-parts detection graphs, with the multimodal gated fusion model representing a strong alternative that achieves comparable performance while integrating both visual and structural information.

9.3 Generalization to Unseen Error States

The final property to be evaluated is robustness to errors. A robust ASR system should be able to distinguish erroneous assembly states from correct ones, enabling error detection and operator notification. Moreover, robustness should be assessed on unseen error states. This is because, as discussed in previous experiments, it is neither practical nor feasible to model or cluster all possible error configurations during training. For this reason, a custom evaluation protocol was designed.

As described in Section 4.1, each frame depicting an error state in the IndustReal test set is associated with additional information, namely the intended well-defined assembly state and the error category. Given a reference image of the intended correct state, a randomly sampled correct image of that same state, and an erroneous image corresponding to it, a robust model should produce embeddings such that the reference and correct images are more similar to each other than the reference and the error images.

To implement this evaluation, a custom reference set of 22 synthetic images was generated using the implemented synthetic data generation pipeline, each depicting a different well-defined assembly state in a standardized configuration (fixed pose and camera position). A few of these are reported in Figure 9.3 as example. A test set was then constructed by taking all error-state frames from the IndustReal test set and, for each one, adding a randomly sampled correct image of the intended state. Embeddings were computed for all images, and cosine similarity with the corresponding reference embedding was used to frame the task as a binary classification problem (correct vs. error). Mean Average Precision (mAP) was computed per error category, along with overall macro- and micro-averaged mAP and $aUROC$. It is important to note that none of the models was trained using *error state* samples, so all evaluated errors are completely unseen.

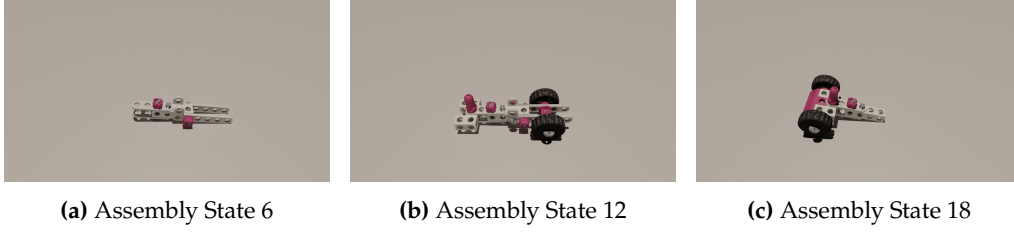


Figure 9.3: Examples of synthetic assembly state reference images generated for the model robustness to unseen errors test.

Metric	Baseline	Graph-based	MM - Naive	MM - Gated
$mAP_{\text{CAT I}}$	<u>0.996</u>	0.850	0.997	0.995
$mAP_{\text{CAT II}}$	0.894	0.739	<u>0.803</u>	0.778
$mAP_{\text{CAT III}}$	0.785	<u>0.777</u>	0.597	0.646
$mAP_{\text{CAT IV}}$	0.678	<u>0.710</u>	0.613	0.726
mAP_{MICRO}	0.833	0.782	0.757	<u>0.809</u>
mAP_{MACRO}	0.838	0.769	0.753	<u>0.786</u>
$auROC$	0.879	0.769	0.828	<u>0.859</u>

Table 9.3: Model performance on Assembly State Recognition in the robustness-to-errors binary verification test on the IndustReal test set, focusing on erroneous assembly states unseen during training. Per-class mean Average Precision (mAP) is reported first, followed by aggregated metrics micro-averaged mAP , macro-averaged mAP and area under the ROC curve ($auROC$). Best results are shown in bold and second-best underlined. (Legend: Cat I = missing component errors; Cat II = incorrect component orientation; Cat III = incorrect component placement; Cat IV = part-level errors.)

The results are reported in Table 9.3. Considering that a random binary classifier would achieve an expected mAP of 0.5, several observations can be made. The baseline already demonstrates strong robustness, with mAP_{MICRO} 0.833, mAP_{MACRO} 0.838 and $auROC$ 0.879. Performance is near optimal for category I, slightly lower for categories II and III, and lowest for category IV. This trend is expected, as category I errors (missing component) involve large structural differences, whereas category IV errors correspond to fine-grained part-level discrepancies.

The graph-based model shows lower performance than the baseline in categories I and II (-14.6% and -17.3% mAP , respectively), similar performance in category III, and improved performance in category IV ($+4.7\%$ mAP). When combining modalities, performance in category I is restored to top values, and improved relative to the graph-based model in category II, although still below the baseline. For category III, both multimodal approaches show reduced performance compared to the baseline, particularly the naive fusion (-23.9% mAP)

and to a lesser extent the gated fusion (-17.8%).

In category IV, naive fusion does not improve over the baseline and even decreases performance (-9.5% *mAP*), failing to exploit the robustness observed in the graph-based approach. In contrast, the gated fusion model successfully leverages this advantage, achieving a new best *mAP* of 0.726, corresponding to a $+7.1\%$ improvement over the baseline.

These results can be interpreted by analyzing the nature of the error categories. In categories II and III, all correct parts are present; the error lies solely in incorrect component orientation or placement. In graph terms, nodes corresponding to parts remain unchanged, and only edge features derived from bounding box geometric relationships differ. Since these features depend on part detection accuracy, any spatial imprecision in detections may limit the model's ability to distinguish subtle positional errors. Conversely, in categories I and IV, errors involve part absence or presence changes, which directly affect graph structure, making them more detectable by graph-based representations.

Overall, the baseline image-based approach achieves the highest robustness to errors, with the multimodal gated fusion model as a close second (considering micro *mAP*, macro *mAP* and *aUROC*). While the baseline remains the most robust overall, the proposed multimodal alternative represents a strong compromise, offering slightly reduced robustness to component orientation and placement errors but improved sensitivity to fine-grained part-level discrepancies.

Conclusions

This thesis addressed the Assembly State Recognition (ASR) task with the objective of designing a solution that satisfies the key properties of an optimal ASR system: high accuracy on known assembly states, strong generalization to unseen states, robustness to errors and scalability to changes in parts.

Achieving all these properties simultaneously makes the task particularly challenging. It was shown that naive solutions, such as training an object detector and directly using it as an image classifier, are inadequate. Such approaches not only perform poorly on known states, but also lack robustness to errors, exhibit no generalization capabilities and are not easily scalable.

A promising direction was identified in Representation Learning, which aims to learn an embedding space where assembly states can be inferred via similarity between sample representations rather than direct classification. Existing approaches already proposed suitable contrastive learning strategies and loss formulations. However, their reliance solely on image-based features limits the ability to explicitly capture structural information about the assembled object, which is fundamental for ASR.

To address this limitation, this thesis introduced a novel input representation based on assembled-parts detections. A synthetic-data-driven object detection pipeline was developed to locate and label the assembled parts composing the object in its current state. From these detections, a graph-based representation was constructed to explicitly encode spatial relationships between parts. To process these graphs, a dedicated Graph Neural Network (GNN) based model was implemented to generate meaningful embeddings.

The proposed representation proved highly informative, particularly when combined with image-based features through a multimodal architecture with gated input fusion. This approach demonstrated the best overall balance across

evaluation criteria, also enabling sensitivity to contextual elements such as hands or foreign objects in the scene.

Quantitatively, the multimodal gated approach improved ASR performance on known states by +6.7% in $F1@1$ compared to the baseline. In terms of generalization to unseen assembly states, the improvement was even more significant, with $F1@1$ increasing from 0.3712 to 0.7407, nearly doubling performance and highlighting the effectiveness of the proposed structural representation. Regarding robustness to errors, the method maintained performance comparable to the baseline overall, showing improved sensitivity to part-level presence or absence changes, at the cost of slightly reduced robustness to fine-grained orientation and placement errors.

Finally, scalability was addressed through the development of a synthetic dataset generation pipeline. By leveraging 3D assembly models, automatically labeled synthetic data can be generated for new parts and configurations, enabling efficient training of part detectors and ensuring adaptability of the overall system without excessive manual annotation effort.

In conclusion, the objectives of this thesis can be considered achieved. The proposed approach constitutes a strong and well-rounded solution to the ASR task, advancing toward the characteristics of an optimal system and improving upon existing methods in the literature across multiple critical dimensions.

10.1 Future Works

The proposed graph-based representation proved to be highly informative and effective, yet there is substantial room for further improvement to enhance its representational power.

Although the assembled-parts detector achieved high accuracy, its performance was not optimal, and this limitation was shown to affect downstream results, particularly in robustness-to-error evaluations. Since the graph construction process directly depends on detection outputs, inaccuracies in part localization or classification inevitably propagate to the graph structure and edge features. Improving detection precision would therefore lead to more reliable graph representations and, consequently, improved ASR performance, especially in fine-grained error detection scenarios.

Beyond refinement, the representation can also be expanded. In the current formulation, part detection is performed in 2D image space. Extending this to 3D detection, specifically 6D pose estimation (position and orientation in space), would allow encoding significantly richer structural information. By incorporating full spatial pose information into node and edge attributes, the resulting

graphs would more accurately reflect the true geometric relationships between assembled components. This would be particularly beneficial also for distinguishing subtle orientation and placement errors, which currently represent the most challenging error categories.

In addition to improving the graph-based representation itself, future work could explore alternative multimodal strategies that integrate additional informative modalities. For instance, as shown in the IndustReal dataset acquisition setup, augmented reality headsets such as the one used for recording capture multiple types of data beyond visual information. Among these, gaze information, tracked directly by the headset, represents a particularly promising modality. Previous work has shown that gaze signals can be meaningful for mistake detection in this context [61]. Integrating gaze information with visual and structural representations could therefore provide complementary cues about operator attention and intent, potentially improving assembly state recognition, especially when distinguishing between intermediate and well-defined states.

Overall, enhancing detection accuracy, incorporating 3D structural information and expanding multimodal integration represent promising paths to further improve the proposed ASR framework.



Bibliography

- [1] Tim J Schoonbeek et al. “Industreal: A dataset for procedure step recognition handling execution errors in egocentric videos in an industrial-like setting”. In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*. 2024, pp. 4365–4374.
- [2] Tim J. Schoonbeek et al. “Supervised Representation Learning Towards Generalizable Assembly State Recognition”. In: *IEEE Robotics and Automation Letters* 9.11 (Nov. 2024), pp. 9915–9922. ISSN: 2377-3774.
- [3] Ana Stanescu et al. “State-Aware Configuration Detection for Augmented Reality Step-by-Step Tutorials”. In: *IEEE International Symposium on Mixed and Augmented Reality, ISMAR*. Ed. by Gerd Bruder et al. IEEE, 2023, pp. 157–166.
- [4] Ana Stanescu et al. “Error Management for Augmented Reality Assembly Instructions”. In: *IEEE International Symposium on Mixed and Augmented Reality, ISMAR*. Ed. by Ulrich Eck et al. IEEE, 2024, pp. 690–699.
- [5] Kevin Murray et al. “Equipment Assembly Recognition for Augmented Reality Guidance”. In: *2024 IEEE International Conference on Artificial Intelligence and eXtended and Virtual Reality (AIxVR)*. IEEE, 2024, pp. 109–118.
- [6] Li-Chen Wu, I-Chen Lin, and Ming-Han Tsai. “Augmented reality instruction for object assembly based on markerless tracking”. In: *Proceedings of the 20th ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. Ed. by Chris Wyman and Cem Yuksel. ACM, 2016, pp. 95–102.
- [7] Shiyu Li et al. “GBOT: Graph-Based 3D Object Tracking for Augmented Reality-Assisted Assembly Guidance”. In: *IEEE Conference Virtual Reality and 3D User Interfaces VR*. IEEE, 2024, pp. 513–523.

- [8] Hannah Schieber and Shiyu Li et al. "ASDF: Assembly State Detection Utilizing Late Fusion by Integrating 6D Pose Estimation". In: *IEEE International Symposium on Mixed and Augmented Reality ISMAR*. Ed. by Ulrich Eck et al. IEEE, 2024, pp. 190–199.
- [9] Bin Wang et al. "Active Assembly Guidance with Online Video Parsing". In: *2018 IEEE Conference on Virtual Reality and 3D User Interfaces VR*. Ed. by Kiyoshi Kiyokawa et al. IEEE Computer Society, 2018, pp. 459–466.
- [10] Hangfan Liu et al. "TGA: Two-level Group Attention for Assembly State Detection". In: *2020 IEEE International Symposium on Mixed and Augmented Reality Adjunct, ISMAR*. IEEE, 2020, pp. 258–263.
- [11] Yongzhi Su et al. "Deep Multi-state Object Pose Estimation for Augmented Reality Assembly". In: *IEEE International Symposium on Mixed and Augmented Reality, ISMAR*. IEEE, 2019, pp. 222–227.
- [12] Bing Zhou and Sinem Güven. "Fine-Grained Visual Recognition in Mobile Augmented Reality for Technical Support". In: *IEEE Trans. Vis. Comput. Graph.* 26.12 (2020), pp. 3514–3523.
- [13] Dan Lehman et al. "Find the Assembly Mistakes: Error Segmentation for Industrial Applications". In: *Computer Vision - ECCV 2024 Workshops*. Ed. by Alessio Del Bue et al. Vol. 15626. Lecture Notes in Computer Science. Springer, 2024, pp. 211–227.
- [14] Francesco Ragusa, Antonino Furnari, and Giovanni Maria Farinella. "MECCANO: A multimodal egocentric dataset for humans behavior understanding in the industrial-like domain". In: *Comput. Vis. Image Underst.* 235 (2023), p. 103764.
- [15] Mattias Billast et al. "Human Motion Prediction on the IKEA-ASM Dataset". In: *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications VISIGRAPP*. 2023, pp. 906–914.
- [16] Fadime Sener et al. "Assembly101: A Large-Scale Multi-View Video Dataset for Understanding Procedural Activities". In: *IEEE/CVF Conference on Computer Vision and Pattern Recognition CVPR*. 2022, pp. 21064–21074.
- [17] Hao Zheng, Regina Lee, and Yuqian Lu. "HA-ViD: A Human Assembly Video Dataset for Comprehensive Assembly Knowledge Understanding". In: *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems NeurIPS*. 2023.

- [18] Ting Chen et al. "A simple framework for contrastive learning of visual representations". In: *International conference on Machine Learning*. PmLR. 2020, pp. 1597–1607.
- [19] Ting Chen et al. "Big Self-Supervised Models are Strong Semi-Supervised Learners". In: *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems*. 2020.
- [20] Jane Bromley et al. "Signature Verification Using A "Siamese" Time Delay Neural Network". In: *Int. J. Pattern Recognit. Artif. Intell.* 7.4 (1993), pp. 669–688.
- [21] Sumit Chopra, Raia Hadsell, and Yann LeCun. "Learning a Similarity Metric Discriminatively, with Application to Face Verification". In: *IEEE Computer Society Conference on Computer Vision and Pattern Recognition CVPR*. IEEE Computer Society, 2005, pp. 539–546.
- [22] Gregory Koch, Richard Zemel, Ruslan Salakhutdinov, et al. "Siamese neural networks for one-shot image recognition". In: *ICML deep learning workshop*. Vol. 2. 1. 2015, pp. 1–30.
- [23] Florian Schroff, Dmitry Kalenichenko, and James Philbin. "FaceNet: A unified embedding for face recognition and clustering". In: *IEEE Conference on Computer Vision and Pattern Recognition CVPR*. IEEE Computer Society, 2015, pp. 815–823.
- [24] Alexander Hermans, Lucas Beyer, and Bastian Leibe. "In Defense of the Triplet Loss for Person Re-Identification". In: *CoRR abs/1703.07737* (2017).
- [25] Aäron van den Oord, Yazhe Li, and Oriol Vinyals. "Representation Learning with Contrastive Predictive Coding". In: *CoRR abs/1807.03748* (2018).
- [26] Prannay Khosla et al. "Supervised contrastive learning". In: *Advances in neural information processing systems 33* (2020), pp. 18661–18673.
- [27] Hexiang Hu et al. "Learning Structured Inference Neural Networks with Label Relations". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition CVPR*. 2016, pp. 2960–2968.
- [28] Zhao-Min Chen et al. "Multi-Label Image Recognition With Graph Convolutional Networks". In: *IEEE Conference on Computer Vision and Pattern Recognition CVPR*. 2019, pp. 5177–5186.
- [29] Tianshui Chen et al. "Learning Semantic-Specific Graph Representation for Multi-Label Image Recognition". In: *2019 IEEE/CVF International Conference on Computer Vision ICCV*. 2019, pp. 522–531.

- [30] Hang Xu et al. "Spatial-Aware Graph Relation Network for Large-Scale Object Detection". In: *IEEE Conference on Computer Vision and Pattern Recognition CVPR*. 2019, pp. 9298–9307.
- [31] Hang Xu et al. "Universal-RCNN: Universal Object Detector via Transferable Graph R-CNN". In: *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI*. 2020, pp. 12492–12499.
- [32] Yi Lu et al. "Graph-FCN for image semantic segmentation". In: *CoRR* abs/2001.00335 (2020).
- [33] Hanzhe Hu et al. "Class-Wise Dynamic Graph Convolution for Semantic Segmentation". In: *Computer Vision - ECCV 2020 - 16th European Conference*. 2020, pp. 1–17.
- [34] Tong Zheng et al. "A Graph Aggregation Convolution and Attention Mechanism Based Semantic Segmentation Method for Sparse Lidar Point Cloud Data". In: *IEEE Access* 12 (2024), pp. 10459–10469.
- [35] Jianchao Wu et al. "Learning actor relation graphs for group activity recognition". In: *Proceedings of the IEEE/CVF Conference on computer vision and pattern recognition*. 2019, pp. 9964–9974.
- [36] Chihiro Noguchi and Toshihiro Tanizawa. "Ego-Vehicle Action Recognition based on Semi-Supervised Contrastive Learning". In: *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. IEEE, Jan. 2023, pp. 5977–5987.
- [37] Siyuan Qi et al. "Learning human-object interactions by graph parsing neural networks". In: *Proceedings of the European conference on computer vision (ECCV)*. 2018, pp. 401–417.
- [38] Keishi Nishikawa, Takaya Taniguchi, and Koji Sakata. "Graph Construction and Processing Towards Egocentric Action Recognition in Machine Inspection". In: *33rd IEEE International Conference on Robot and Human Interactive Communication (ROMAN)*. IEEE. 2024, pp. 1228–1235.
- [39] G. Hinton, R. Salakhutdinov, and Yoesoep Rachmad. "Reducing the Dimensionality of Data with Neural Networks". In: *Science* 313 (Aug. 2006), pp. 504–7.
- [40] Ross B. Girshick et al. "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation". In: *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*. 2014, pp. 580–587.
- [41] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 779–788.

- [42] Franco Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.
- [43] Thomas N. Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Networks". In: *5th International Conference on Learning Representations ICLR*. 2017.
- [44] Petar Velickovic et al. "Graph Attention Networks". In: *6th International Conference on Learning Representations ICLR*. 2018.
- [45] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*. 2017, pp. 5998–6008.
- [46] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *9th International Conference on Learning Representations ICLR*. 2021.
- [47] Olga Russakovsky et al. "Imagenet large scale visual recognition challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.
- [48] Teng Yang et al. "Self-supervised models are strong industrial few-shot defect classification learners". In: *European Conference on Computer Vision*. Springer. 2024, pp. 310–327.
- [49] Rahima Khanam and Muhammad Hussain. "YOLOv11: An Overview of the Key Architectural Enhancements". In: *CoRR abs/2410.17725* (2024).
- [50] Tsung-Yi Lin et al. "Microsoft coco: Common objects in context". In: *European conference on computer vision*. Springer. 2014, pp. 740–755.
- [51] Jørgen Bang-Jensen and Gregory Gutin. "Basic Terminology, Notation and Results". In: *Classes of Directed Graphs*. Ed. by Jørgen Bang-Jensen and Gregory Gutin. Springer Monographs in Mathematics. Springer International Publishing, 2018, pp. 1–34.
- [52] Adam Paszke et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).
- [53] Matthias Fey and Jan Eric Lenssen. "Fast Graph Representation Learning with PyTorch Geometric". In: *CoRR abs/1903.02428* (2019).
- [54] Tomás Mikolov et al. "Efficient Estimation of Word Representations in Vector Space". In: *1st International Conference on Learning Representations ICLR*. Ed. by Yoshua Bengio and Yann LeCun. 2013.

-
- [55] Tianle Cai et al. “Graphnorm: A principled approach to accelerating graph neural network training”. In: *International Conference on Machine Learning*. PMLR. 2021, pp. 1204–1215.
 - [56] John Edison Arevalo Ovalle et al. “Gated Multimodal Units for Information Fusion”. In: *5th International Conference on Learning Representations ICLR*. 2017.
 - [57] Maximilian Denninger et al. “BlenderProc2: A Procedural Pipeline for Photorealistic Rendering”. In: *Journal of Open Source Software* 8.82 (2023), p. 4901.
 - [58] Sven Burdorf, Karoline Plum, and Daniel Hasenklever. “Reducing the Amount of Real World Data for Object Detector Training with Synthetic Data”. In: *CoRR* abs/2202.00632 (2022).
 - [59] Kevin Musgrave, Serge Belongie, and Ser-Nam Lim. “A metric learning reality check”. In: *European Conference on Computer Vision*. Springer. 2020, pp. 681–699.
 - [60] Leland McInnes et al. “UMAP: Uniform Manifold Approximation and Projection”. In: *Journal of Open Source Software* 3.29 (2018), p. 861.
 - [61] Michele Mazzamuto et al. “Gazing into missteps: Leveraging eye-gaze for unsupervised mistake detection in egocentric videos of skilled human activities”. In: *Proceedings of the Computer Vision and Pattern Recognition Conference*. 2025, pp. 8310–8320.