Politecnico di Torino

POLITECNICO DI TORINO

College of Computer Engineering, Cinema and Mechatronics

Master's Degree Thesis

Temporal Tracking of Waste Objects for New Item Identification in Smart Bins

Supervisors
prof. Bartolomeo Montrucchio
dr. Antonio Costantino Marceddu

Candidate
Matteo Gravile

July 2025

Abstract

This thesis addresses the problem of visual tracking of objects within images captured by smart bins to determine whether an object is new or has already been observed. Unlike traditional object detection methods, where each instance is recognized independently, this work focuses on the temporal dimension and the system's ability to retain the memory of previously seen objects, thus improving continuous and automated waste monitoring. The dataset used, consisting of approximately 7,000 images annotated in Common Object in Context (COCO) format, includes bounding boxes, segmentation masks, object categories (about 60 classes), and a custom "new" attribute indicating whether an object is new with respect to the temporal context. This attribute takes the value "yes" for newly appearing objects and "no" for those already seen. After an initial phase of annotation refinement and cleanup, three experimental architectures were designed and compared to correctly predict the "new" label as either "yes" or "no". The first architecture, used as a baseline, combines a ResNet50 visual feature extractor with a supervised Multilayer Perceptron (MLP) classifier that receives as input a set of geometric and similarity-based metrics (cosine similarity, Intersection over Union, centroid distance, area ratio) computed between the current object and those stored in memory. The second approach is a Memory-Augmented Network (MAN) that integrates a learnable Transformer module to retain a dynamic memory of previously observed objects. The current object is processed in relation to this memory to produce a contextualized representation, which an MLP classifies. Finally, a Siamese network was developed and trained using both pairs and triplets of objects to learn discriminative visual embeddings via supervised losses (Contrastive Loss and Triplet Loss). During inference, the similarity between the current object and those in memory is combined with geometric features and provided to an MLP for final classification. The results, evaluated using standard metrics such as accuracy, precision, recall, F1-score, and Area Under the Curve (AUC), clearly highlight the strengths and limitations of each method. The best-performing configuration was the Siamese network trained with triplet loss and followed by a supervised MLP, achieving 0.787 accuracy, 0.747 precision, 0.906 recall, 0.818 F1-score, and 0.85 AUC. This was followed by the Memory-Augmented Network with MLP (F1 = 0.781) and the baseline ResNet50 + MLP (F1 = 0.761). These results demonstrate the effectiveness of advanced architectures in accurately recognizing previously seen objects, even in visually complex and dynamic environments. Overall, this work contributes to the development of intelligent waste management systems capable of incorporating temporal reasoning into visual recognition, addressing real-world challenges with robustness and adaptability. The proposed solutions, based on supervised learning and memory-aware models, represent a promising step toward adaptive and continuous object recognition in urban and environmental scenarios.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Application context and motivations

The intelligent management of urban waste represents one of the most complex and relevant challenges in the current context of ecological transition and digital transformation. Increasing urbanization, combined with rising consumption and the spread of new production and distribution models, has led cities to urgently rethink their environmental infrastructure. In this scenario, separate waste collection and the precise monitoring of waste flows play a key role in achieving sustainability goals, reducing environmental impact, and improving operational efficiency.

To address these needs, numerous administrations and companies are adopting innovative technological solutions based on sensors, communication networks, and advanced analysis tools. Among these, so-called *smart bins* — intelligent containers equipped with electronic components and control algorithms — represent one of the most promising applications. These devices are not only capable of detecting the presence and volume of waste deposits, but also of dynamically interacting with the surrounding environment, adapting in real time to operating conditions and providing valuable data to service managers.

In particular, the integration of computer vision modules within smart bins has significantly expanded the application potential of such systems. The integrated cameras allow for visual analysis of the waste content, accurately identifying the type of disposed items and verifying the correctness of disposal operations. This capability supports a wide range of additional functionalities: from automatic recognition of materials (plastic, paper, glass, organic, metal) to anomaly detection, and the generation of updated statistics for the development of operational and informational strategies.

In some pilot projects, the data collected by such systems are already being used to feed pay-as-you-throw pricing models, optimize collection routes, activate automatic malfunction or overflow alerts, and even to launch personalized environmental education campaigns. In this way, the bin becomes an active node within a distributed urban infrastructure, capable of interacting with other systems and contributing to the construction of a true urban *collective intelligence* in the environmental field.

The actual availability of visual data that is updated and contextualized also represents an opportunity to increase transparency and user accountability. Real-time feedback, for example, can serve as an educational mechanism, showing users any disposal errors and suggesting correct behaviors. This aspect is particularly relevant in high-traffic public environments, such as schools or corporate cafeterias, where the effectiveness of waste collection also depends on the aggregate behavior of individuals.

However, despite technological advancements, many of the solutions currently in use still adopt a static approach to visual analysis. The *object detection* models used to classify objects within waste typically operate on single images, without considering the temporal sequence from

which these images are extracted. Each object is analyzed as if it were an isolated entity, with no history, regardless of whether it has already been seen in previous frames. This lack of continuity represents a structural limitation of the current approach, as it prevents the system from constructing a coherent representation of the evolution of objects over time.

As a result, the same object may be detected and counted multiple times, especially if it remains within the camera's field of view for an extended period or moves slowly. This phenomenon leads to a systematic overestimation of the actual number of disposals, generating redundant data and compromising the quality of the statistics. In high-usage contexts, this type of error can lead to incorrect operational decisions: a bin may appear full when it is not, or a user may be unfairly penalized. More generally, the system lacks the ability to correctly represent the temporal dynamics of the observed scene.

The problem of lacking visual memory is not exclusive to the waste management sector. It is a recurring challenge in many other fields of applied computer vision, where it is necessary to distinguish between recurring and new events or objects. In video surveillance systems, for instance, it is essential to recognize subjects that have already appeared in order to identify anomalous behavior or potential threats. In logistics and retail, tracking the recurrence of objects or people helps optimize the management of spaces and resources. In robotics as well, the ability to distinguish known elements from new ones is crucial for interacting with dynamic and unstructured environments.

In all these applications, the need emerges to equip systems with some form of visual memory, capable of keeping track of previous observations and supporting intelligent comparisons over time. Such a memory cannot be a simple static archive of images, but must include efficient data structures, compact representation models, and algorithms capable of performing robust comparisons between visual representations. Only through this type of mechanism can the system acquire a *temporal* view of the world, rather than just an instantaneous one.

In this scenario lies the work of Relearn, an Italian startup that has made digital sustainability and artificial intelligence applied to waste management its core focus. The company has developed a device called *Nando*, designed for direct installation on smart bins, capable of acquiring high-resolution images and applying real-time object detection techniques to recognize the types of disposed items. The system is already operational in various real-world contexts and can visually classify objects into more than 60 different categories.

The device is designed to operate in public and semi-controlled environments, providing both immediate feedback to users and aggregate data for statistical or decision-making purposes. However, in its current configuration, Nando adopts a frame-by-frame logic, lacking explicit mechanisms for handling temporality or object recurrence. This limitation can lead to a distorted view of the actual flow of disposals and compromise its statistical and operational accuracy.

1.2 Thesis objectives

In light of these considerations, the present work aims to extend the system's functionalities by integrating a temporal visual memory mechanism. The primary objective is to enable the system to distinguish between previously observed and new objects, thus overcoming the limitations of the currently adopted frame-by-frame approach.

This ability to recognize visual recurrence over time is crucial for improving the reliability of the collected data, reducing the overestimation of disposals, and strengthening the consistency of the metrics used in high-density operational contexts such as cafeterias, schools, public spaces, or high-traffic environments. The inability to distinguish between genuinely new objects and those already present in the camera's field of view results in a systematic overcounting of disposals, generating noisy data, statistical distortions, and potentially incorrect decisions by the system. This, in turn, negatively impacts operational efficiency, the credibility of user feedback, and the accuracy of the metrics used to guide maintenance, emptying, or environmental education interventions.

To address this issue, the thesis introduces an innovative module for the temporal identification of observed objects. The goal is to assign, for each detected object, a binary label indicating its novelty, distinguishing between `new: yes` (first appearance) and `new: no` (already observed object). This approach, which fits within the paradigm of temporal computer vision, enhances the functionality of existing systems by equipping them with a primitive yet effective form of "visual memory." The task, far from trivial, becomes even more challenging under real-world conditions such as lighting variations, partial occlusions, perspective deformations, and the presence of similar objects.

From a formal perspective, the problem has been framed as a binary supervised classification task. Each detected object o_t in an image I_t is associated with a `new` label, based on a comparison between its visual and geometric characteristics and those of previously observed objects $(o_{t-1}, o_{t-2}, \dots)$. The comparison is carried out through a dynamic visual memory, designed to be queryable, efficient, and compact enough to operate on resource-constrained devices, such as the embedded systems typical of smart bins.

A primary methodological goal is therefore the definition of a discriminative visual representation capable of capturing the salient features of objects while remaining robust to noise and non-significant variations. To this end, pre-trained feature extraction models such as ResNet50 will be used, alongside geometric descriptors such as position, size, centroid, area ratios, and IoU. The integration of these heterogeneous features aims to strengthen the system's ability to distinguish between similar but distinct objects, thereby improving decision quality.

In parallel, the thesis involves the design of an artificial visual memory, whose purpose is to store a synthetic yet informative trace of previously observed objects. Memory management will require the development of dynamic update and pruning strategies, aimed at avoiding the proliferation of redundant entries. Policies based on similarity thresholds (cosine, Euclidean), temporal decay, maximum duration, and spatial consistency will be considered, with the goal of balancing historical depth and responsiveness.

At the core of the system will be a decision module, responsible for estimating the probability that a currently detected object is actually new. To this end, three distinct architectures will be implemented and compared:

1. A supervised MLP classifier, fed with numerical metrics between current objects and those stored in memory.
2. A Transformer model with an attention mechanism, designed to retain internal memory and model relationships between objects over time.
3. A Siamese network, trained on both pairs and triplets of objects, capable of learning a discriminative embedding space.

Each architecture will be evaluated in terms of accuracy, stability, generalization capability, inference time, and computational footprint. Experimental tests will be conducted on a real dataset provided by Relearn, consisting of approximately 7,000 sequential images, initially labeled according to the COCO standard and later manually enriched with the `new` attribute. This annotation activity required a sequential review of the images, with the goal of constructing a reliable and coherent temporal history.

From a computational standpoint, the thesis will analyze in detail the system's compatibility with the typical constraints of embedded devices. Lightweight configurations for feature extraction (e.g., ResNet18), batchless modes for real-time inference, and pruning strategies for memory management will be tested. The goal is to achieve a stable, fast, and resource-efficient pipeline without sacrificing prediction quality.

An important part of the work will also be dedicated to the selection and management of the decision threshold for binary classification. Approaches based on fixed thresholds, adaptive dynamic thresholds (depending on the object type or environmental conditions), and thresholds

learned during training will be evaluated. This choice will significantly influence the robustness of the system in real-world scenarios, which are often highly variable.

Finally, particular attention will be paid to the generalizability of the proposed solutions. In addition to training and testing on the annotated dataset, evaluations will be carried out on unseen sequences acquired under conditions different from those used during training.

In summary, the objectives of the thesis are structured along three fundamental axes:

- The design of a system capable of distinguishing between new and already seen objects, improving temporal continuity in computer vision.
- The comparative experimentation of multiple architectural solutions (MLP, Siamese, Transformer) for temporal identification.
- The optimization of a lightweight and generalizable pipeline suitable for embedded and real-world operational environments.

The work aims to demonstrate that the introduction of the `new` attribute, if properly managed, can provide a crucial contribution to the development of intelligent systems capable of reasoning over time. From a broader perspective, this functionality can be considered a first step toward equipping computer vision systems with memory capabilities, bringing them closer to situated, contextual, and proactive intelligence.

1.3 Existing technologies and current limitations

In the context of computer vision applied to waste management, numerous systems have been developed that are capable of classifying disposed materials through object detection models. These systems are mainly based on pre-trained convolutional neural network architectures and allow for the correct identification of multiple object categories. However, most of them adopt a frame-by-frame approach, treating each image as independent from the others. This results in the absence of any form of temporal memory: each object is analyzed as if it were a new instance, even if it has already appeared in previous frames.

Such an approach introduces significant limitations in dynamic contexts, where objects may remain in the scene across multiple consecutive frames. Repeated detections lead to overcounting, reduced accuracy in aggregate statistics, and an incomplete understanding of the visual sequence. To address these issues, the literature has proposed more advanced approaches, including tracking, re-identification (Re-ID), and metric learning. These techniques, widely adopted in other application domains such as video surveillance, robotics, and retail, offer promising solutions for maintaining object identity over time.

Tracking algorithms attempt to associate a consistent identity to each detected object throughout the sequence. Solutions such as SORT and Deep SORT combine spatial data and visual representations, proving effective under controlled conditions. However, their use in smart bins is hindered by specific constraints, such as the presence of static or occluded objects, the discontinuous acquisition frequency of frames, and the absence of coherent trajectories. In these cases, tracking tends to lose object identities or introduce association errors.

Re-ID models, originally developed for person recognition in multi-camera scenarios, are based on visual embeddings compared using distance metrics. Although more robust to changes in perspective and lighting, these models are often trained on closed datasets and require significant computational infrastructure, making them poorly suited for embedded applications. Moreover, the open-world nature of waste makes classification based on predefined identities unsuitable for the problem.

Metric learning represents a third path, based on learning semantic spaces where similar objects are positioned close to each other. Siamese networks and Triplet Networks, trained on

pairs or triplets of objects, allow for flexible instance-to-instance comparisons. This methodology, used in numerous domains such as face recognition and signature verification, makes it possible to estimate object similarity without explicitly classifying them, making it useful for novelty estimation.

More recently, Transformers and neural networks with explicit memory have opened new perspectives for temporal analysis. These models, although promising due to their ability to model long-term dependencies, come with high computational costs. Their adoption in embedded environments therefore requires lightweight solutions, optimized through pruning, quantization, and compression techniques.

In summary, the literature review shows that existing solutions address the problem of visual memory only partially and are often not directly applicable to embedded and intermittent contexts such as that of smart bins. This thesis is positioned within this still partially explored research space, proposing a hybrid and lightweight approach adapted to the operational specificities of the Nando system.

In the following section, the main design and experimental contributions developed during this work will be presented in detail, illustrating the architectural choices, implementation methods, and validation strategies adopted.

1.4 Main contributions

In light of the objectives outlined in the previous section, this thesis has led to the development of a series of concrete and structured contributions aimed at filling the identified gap in computer vision systems applied to smart bins. The entire work focused not only on the introduction of a module for the temporal classification of detected objects, but also on the construction of a complete technical and methodological ecosystem capable of structurally addressing the numerous challenges involved in recognizing previously observed objects over time.

Rather than limiting itself to the theoretical design of the system, the thesis pragmatically tackled every aspect of the problem: from the formalization of the task to software implementation, from the definition of architectures to experimental validation, and from dataset review to its expansion. Every phase of the project was aimed at building a robust, generalizable, and efficient system, compatible with execution in real and constrained environments such as embedded systems.

This section presents the main contributions obtained, categorized by their nature: formal, architectural, experimental, and implementation-related. The aim is to provide a clear and systematic overview of the results achieved and their impact both from a technical and application-oriented perspective. These contributions are not isolated elements but integral parts of a cohesive pipeline, designed to address a real-world problem and with the potential to be extended to other domains in which the temporal dimension of visual analysis is crucial.

The first contribution consisted in the formalization of the problem as a binary supervised classification task, based on the concept of object "novelty." A complete framework was developed, capable of processing annotated sequences, maintaining a historical visual memory, and producing for each object a binary label (`new: yes/no`) based on visual and geometric comparisons. This formulation enabled not only the implementation of different neural architectures but also the modular integration of the system into Nando's operational pipeline, with special attention to compatibility with resource-constrained systems.

To address the technical challenge of temporal classification, three different neural architectures were implemented, each representing a complementary approach to the problem. The first uses a pre-trained ResNet50 as a feature extractor, followed by an MLP classifier fed with a set of heterogeneous metrics: Euclidean distance between centroids, cosine similarity between visual embeddings, Intersection over Union (IoU), and the ratio of bounding box areas. This model, designed as a baseline, demonstrated good discriminative capacity, particularly in controlled environments, while also being lightweight, interpretable, and easily deployable. An ablation study

was also conducted to analyze the relative importance of each metric, helping to understand the actual contribution of each feature to the decision-making process.

The second architecture is based on a Transformer module, integrated into a *Memory-Augmented Network* capable of dynamically modeling the visual sequence and comparing the current object with a historical context retained in memory. Thanks to the multi-head attention mechanism, the model is able to handle complex cases such as occluded, deformed, or ambiguous objects. The design required attention to sequence normalization, the choice of positional encoding, and the selection of objects to retain in memory. This approach proved particularly effective in real-world scenarios characterized by high variability.

Finally, a Siamese network was developed and used both with pairs (positive and negative) through a *Contrastive Loss*, and with triplets (anchor, positive, negative) via *Triplet Loss*, with the goal of learning a highly discriminative embedding space. In this space, visually similar objects are projected close to each other, while distinct objects are well separated. During inference, the current object is compared with those stored in memory based on the similarity between their respective embeddings, and this information is then processed by a final MLP. The training process required the controlled generation of pairs and triplets through dedicated scripts, as well as t-SNE analysis to evaluate the quality of the learned space. This solution, though more sophisticated, demonstrated notable generalization capability even under conditions of visual noise and poor lighting.

A fundamental element of the work was the reprocessing of the dataset provided by Relearn. Composed of approximately 7,000 images labeled according to the COCO standard, the dataset was extended with the introduction of the `new` label through a meticulous manual review, carried out by following the temporal sequence of the images. This process made it possible to build a reliable object history, which was essential for model training. The definition of "novelty" was based on objective criteria: spatial proximity, semantic class, visual similarity, and temporal order. This extended dataset can also be reused for further developments in the field of semantic temporal tracking.

From an implementation perspective, the entire system was developed in a cloud environment on an Azure virtual machine, using Python and JupyterLab. Specialized libraries were adopted: PyTorch for deep learning, OpenCV and NumPy for visual manipulation, Albumentations for augmentation, and Scikit-learn and Matplotlib for performance analysis. The code architecture was designed in independent and reusable modules, with the goal of ensuring extensibility and simplified maintenance.

Finally, the experimental evaluation phase was conducted rigorously. Cross-validation methods, stratified data splitting, and various standard binary classification metrics were used: precision, recall, F1-score, accuracy, and AUC. Particular attention was paid to error analysis, identifying recurring patterns in false positives and false negatives. In summary, the Siamese network achieved the best overall performance, while the Transformer model showed superior stability in the presence of environmental variations. The MLP model, on the other hand, proved to be a solid baseline, simple to train and robust under standard conditions.

Taken together, the main contributions of the thesis can be summarized as follows:

- Formalization of the temporal identification problem as a supervised binary classification task.
- Design, implementation, and comparison of three neural architectures: MLP, Memory-Augmented Network (Transformer), and Siamese network (with pairs and triplets).
- Manual annotation of the dataset provided by Relearn, with the introduction of the temporal label `new`.
- Development of a complete, modular, and scalable pipeline for both cloud environments and embedded devices.

- Comparative analysis of quantitative and qualitative performance on real data, with attention to generalization and efficiency.

These contributions demonstrate a body of work that combines theoretical research, software design, and experimental validation, delivering a concrete and innovative system for temporal object identification. The proposed solutions, originating from a specific use case, are generalizable to a wide range of applications where the temporal dimension and object persistence play a central role, particularly in fields such as collaborative robotics, intelligent surveillance, and urban cyber-physical systems. The project shows how, even in constrained contexts such as embedded devices, it is possible to introduce forms of "visual memory" capable of radically improving scene interpretation and real-time decision support.

Chapter 2

State of the Art

This chapter aims to provide a systematic and in-depth overview of the state of the art concerning the main methodologies, architectures, and tools in computer vision that constitute the theoretical and technological foundation for this work. The objective is twofold: on one hand, to offer a critical and up-to-date overview of the progress made in the fields of object detection, visual tracking, metric learning, and architectures with visual memory; on the other hand, to highlight the gaps that remain in the literature, particularly regarding the management of temporality and visual persistence in dynamic and unsupervised environments. Fully understanding the strengths and weaknesses of existing solutions is essential to justify the architectural and methodological choices made in this thesis project.

In recent years, automatic object detection and visual tracking have made remarkable progress thanks to the introduction of high-performance deep learning models, often based on convolutional neural networks (CNNs) or, more recently, Transformer architectures. These technologies have revolutionized visual processing of images and videos, enabling levels of accuracy and generalization previously unattainable with hand-crafted feature-based approaches. Such models are now capable of recognizing hundreds of visual categories with inference times compatible with real-time applications, contributing to a profound transformation of sectors such as urban surveillance, autonomous driving, intelligent retail, industrial automation, and mobile robotics. Systems like YOLOv5, EfficientDet, and DETR are emblematic examples of this evolution, combining computational efficiency and discriminative capability even in highly variable scenarios.

However, the application of these technologies to real-world, dynamic, and open scenarios—such as automated waste collection—poses still unresolved challenges. In unsupervised contexts, characterized by high heterogeneity of objects and frequent presence of visual noise (such as lighting variations, occlusions, and perspective distortions), the reliability of standard object detection techniques can be significantly compromised. In particular, most systems still operate in a frame-by-frame mode, analyzing each image as an isolated instance, without temporal continuity. This setting proves inadequate in contexts where it is necessary to distinguish between truly new objects and those already observed, as is the case in systems that continuously monitor visual flows—for example, when opening a smart bin or entering a monitored area.

The problem, known in the literature as visual recurrence or temporal identity reasoning, implies the need for computer vision systems to maintain a persistent representation of observed identities over time. This is not simply about storing images, but about building data structures and semantic representations capable of enabling reliable comparisons between objects over time. The absence of explicit memory compromises the system's ability to distinguish between recurring instances and novel ones, resulting in limitations in robustness, interpretability, and data quality. This aspect becomes especially critical in all cases where visual data is used to make operational decisions, activate feedback mechanisms, or feed predictive systems.

In response to these critical issues, scientific research has progressively proposed a variety of approaches to integrate the temporal dimension into vision systems. Among these, notable

techniques include multi-object tracking, re-identification (Re-ID), metric-based learning (metric learning), and more recently, neural architectures equipped with attention mechanisms and internal memory. Each of these research directions offers conceptual and practical tools to address the challenge of visual persistence in different ways: some focus on the spatial and temporal continuity of objects (tracking), others on learning semantic spaces that preserve similarity between visual instances (Siamese networks, Triplet loss), and others still on the explicit modeling of sequences (Transformers with memory).

The contribution of this chapter lies in providing the theoretical and methodological context that justifies these design choices. Starting from the foundations of object detection and its structural limitations, the chapter will delve into the main existing solutions for multi-object tracking, embedding techniques for visual comparison, and neural architectures that integrate memory and temporal attention. Particular attention will be paid to the relevance of these technologies in the domain of automated waste management, where accuracy, robustness, and the ability to generalize over time are essential requirements.

2.1 Object Detection: methods and limitations

Object detection, namely the automatic identification of objects in images and the estimation of their bounding boxes, is one of the fundamental tasks in computer vision. Accuracy, inference speed, and the ability to generalize in dynamic contexts have made this field one of the most active in the deep learning landscape. The introduction of convolutional neural networks (CNNs) marked a turning point, enabling end-to-end models capable of learning directly from visual data, surpassing previous techniques based on handcrafted feature extraction (e.g., HOG, SIFT).

Modern object detection first evolved through *two-stage* models, where detection is divided into two phases: the generation of region proposals and their subsequent classification.

The first successful model in this direction was **R-CNN** [1], which uses Selective Search to produce about 2000 proposals per image, each of which is independently classified by a CNN and then refined via linear regression.

Fast R-CNN [2] improved efficiency by sharing the convolution across the entire image and using ROI Pooling to extract features from each proposal, drastically reducing inference time.

With **Faster R-CNN** [3], the generation of proposals was integrated directly into the model through a Region Proposal Network (RPN), enabling end-to-end training.

However, despite their high accuracy, these models have several limitations:

- Latencies greater than 100 ms/frame, which are incompatible with real-time environments [4].
- Heavy architectures, difficult to deploy on embedded or edge devices [5].

As shown in Figure 2.1, the architectural differences between R-CNN, Fast R-CNN, and Faster R-CNN mainly concern proposal generation and feature sharing. The image is taken from [6].

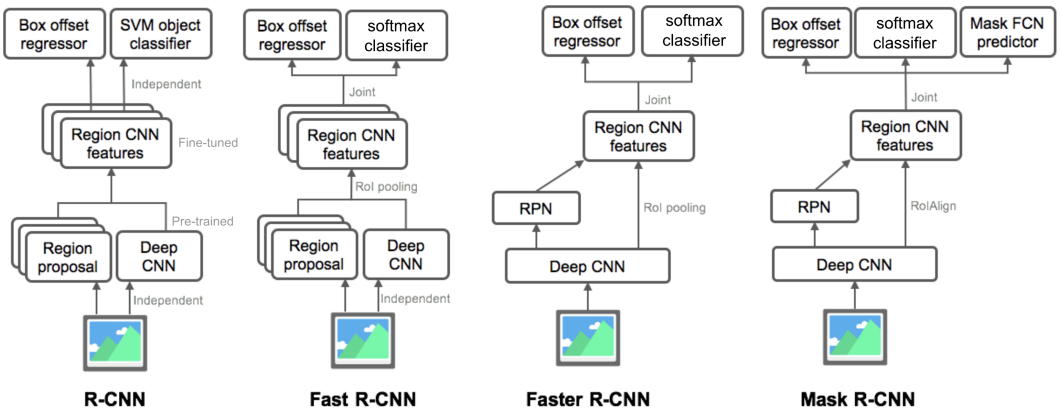


Figure 2.1: Comparison between the R-CNN, Fast R-CNN, and Faster R-CNN architectures. Adapted from [6].

To address the need for reduced complexity and latency, *one-stage* models were introduced, in which detection and classification occur in a single step.

The **SSD (Single Shot MultiBox Detector)** model [5] uses feature maps at different scales to detect objects of various sizes. This approach maintains good performance on small objects, making it useful in complex environments.

As illustrated in Figure 2.2, SSD and Faster R-CNN architectures show a clear trade-off between accuracy and latency: SSD models are much faster but less precise, whereas Faster R-CNN is more accurate at the cost of inference time.

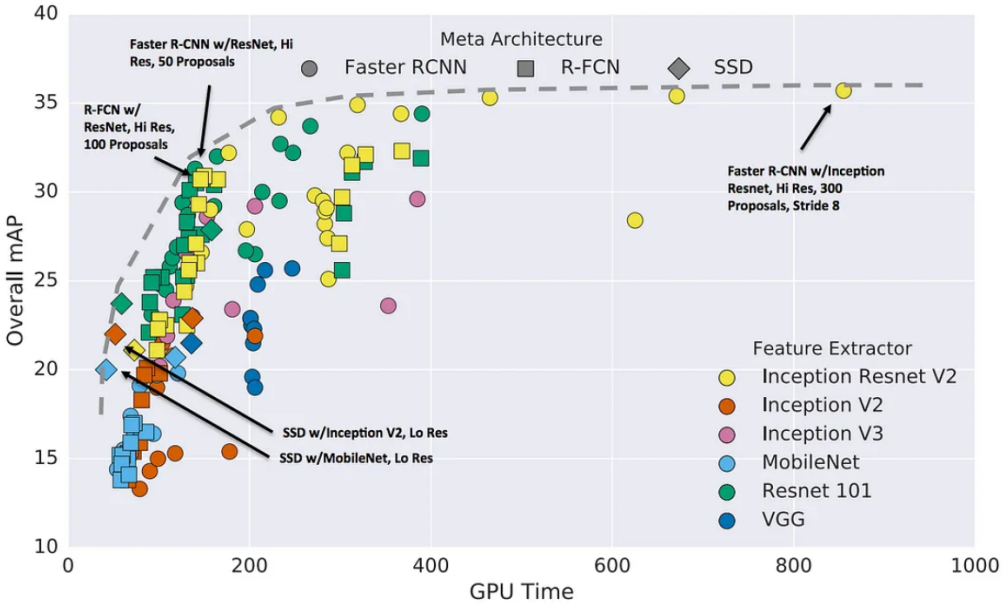


Figure 2.2: Comparison between SSD and Faster R-CNN in terms of accuracy (mAP) and GPU latency. Models are displayed based on the feature extractor used. Source: TensorFlow Object Detection API benchmark chart.

A significant breakthrough came with the **YOLO (You Only Look Once)** series: YOLOv1 [7] segments the image into grids and assigns multiple class and bounding box predictions. Later versions—up to YOLOv8—have improved both precision and efficiency through new data augmentation techniques, optimized backbones, and attention modules [8], [9].

The features that make YOLO a suitable model for embedded contexts like smart bins include:

- Very low inference times (up to 200+ FPS on modern GPUs).
- Support for lightweight backbones (e.g., CSPDarknet, MobileNet).
- Excellent trade-off between accuracy and response time in open-world environments.

However, some critical issues persist:

- Degraded accuracy in the presence of overlapping or very small objects [10].
- Unstable behavior on noisy or low-light inputs [11].

Another key distinction among models lies in the *loss functions* used to optimize object localization. SSD and Faster R-CNN typically use a combination of *classification loss* (e.g., cross-entropy) and *localization loss* (e.g., Smooth L1). YOLOv4 introduces CIoU loss, which also accounts for the relative orientation of bounding boxes [12]. Furthermore, while traditional models rely on predefined *anchor boxes*, some recent architectures experiment with anchor-free detection or learn anchors during training [13].

A significant distinction among more recent architectures concerns precisely the handling of anchor boxes. While SSD and Faster R-CNN employ predefined anchors of various sizes and aspect ratios, models such as CenterNet, FCOS, and CornerNet adopt anchor-free strategies, focusing instead on the direct prediction of object centers and dimensions. This approach eliminates the need for manual anchor tuning and has proven particularly effective on open-world or imbalanced datasets, where the morphological variety of objects is high. Moreover, anchor-free architectures tend to be more flexible and adaptable in dynamic scenarios, such as those related to waste sorting, where objects can vary significantly in shape, size, and position [13]–[15].

All the models mentioned above are built upon CNN-based architectures known as *backbones*, which are responsible for extracting high-level visual features. Early models used VGG-16 [16], but deeper and more modular backbones are now preferred, such as:

- **ResNet** [17]: uses residual blocks to ensure gradient propagation in deep networks.
- **CSPDarknet**: the native backbone of YOLOv5, with structural optimizations for embedded environments.
- **MobileNet, ShuffleNet**: designed to reduce parameters and accelerate inference.

As shown in Figure 2.3, the backbone integrated with FPN features a bottom-up and top-down pathway with lateral connections, ensuring efficient multiscale feature extraction. The residual block structure is embedded in the ResNet backbone. Adapted from [18].

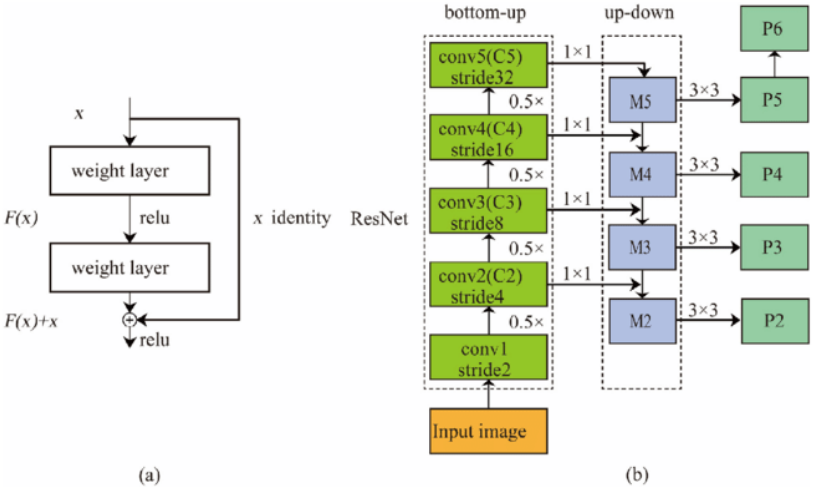


Figure 2.3: ResNet backbone with FPN structure (bottom-up + top-down with lateral connections) and residual blocks. Adapted from [18].

Despite significant advances, traditional object detection models exhibit a critical limitation for the context of this thesis: they operate according to a purely *frame-by-frame* logic. In other words, each frame is analyzed independently, and the system retains no memory or identity of previously seen objects.

In the context of a smart bin, this approach is limiting for at least three reasons:

- **Lack of persistent identity**: the same object, observed multiple times, is treated as a different entity each time.
- **Inability to estimate novelty**: no temporal or semantic information allows the model to determine whether an object is new.
- **Challenges in sequential scenarios**: the system cannot learn temporal patterns, which are crucial to recognize recurring behaviors (e.g., reappearance of the same object or tracking over time).

To demonstrate this, the **DSYOLO-Trash** algorithm [19], based on YOLOv5 with CBAM attention and CotNet, combined with DeepSORT for tracking, achieves a mAP of 97.3% on TrashNet, yet still operates on a frame-independent basis and does not explicitly address the dimension of visual recurrence.

As shown in Figure 2.4, Schneider et al. [20] propose an architecture based on ResNet50 and FPN with a decoder for semantic segmentation, designed for industrial environments with LiDAR data. However, the system does not incorporate any visual memory component.

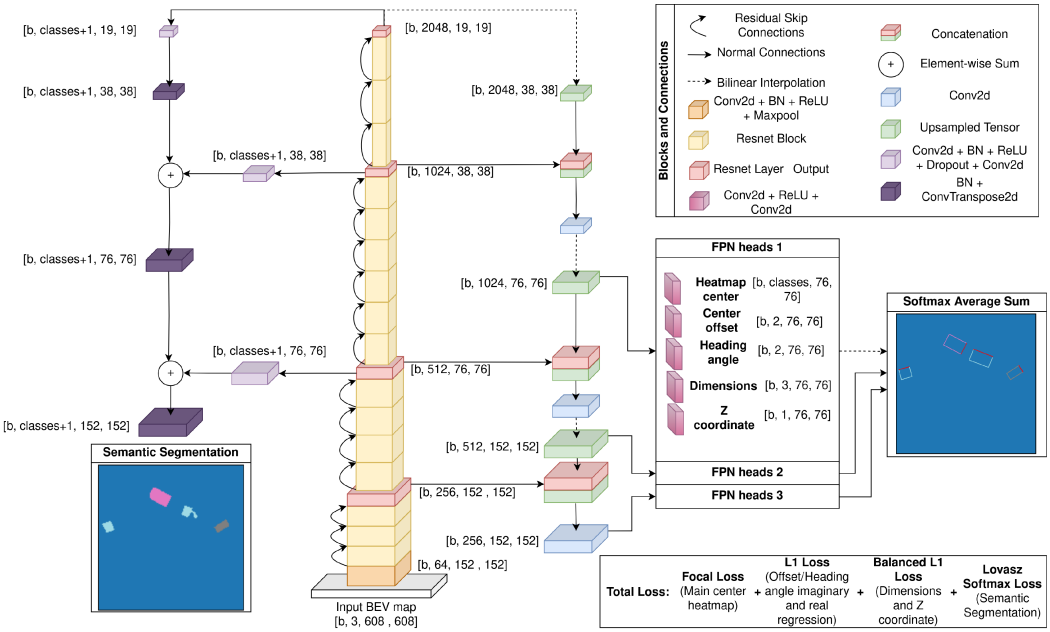


Figure 2.4: ResNet50 + FPN architecture with semantic segmentation module, similar to the one proposed by Schneider et al. for industrial environments.

These observations underscore the need to develop more advanced models capable of retaining information across sequences, comparing instances over time, and inferring "visual novelty." The following chapters will explore architectures specifically designed to fill this gap, through mechanisms of neural memory, temporal attention, and metric learning.

2.2 Visual tracking and Re-Identification

Following detection, the next step in a visual pipeline is *visual tracking*, which consists in maintaining consistent object identity across frames. In modern approaches such as *tracking-by-detection*, the bounding boxes generated by detectors like YOLO or SSD are associated over time through algorithms that integrate kinematic and visual information, with the goal of maintaining identity coherence over time—even in the presence of occlusions, interactions, or deformations.

One of the most well-known methods for multi-object tracking is **SORT** (Simple Online and Realtime Tracking) [21]. SORT uses a Kalman filter to predict the position and velocity of objects and applies the Hungarian algorithm to associate detections with active tracks. Although effective in low-dynamic and low-density scenarios, SORT relies solely on geometric and kinematic features, making it vulnerable to occlusions, scale variations, visual deformations, and temporary overlaps. In particular, the absence of visual appearance features severely limits its ability to distinguish between visually similar objects or to re-identify objects that reappear after temporal interruptions.

To overcome these limitations, **DeepSORT** [22] was proposed, which supplements the kinematic tracker from SORT with a visual descriptor based on a CNN pre-trained for *person re-identification*. Each detected object is assigned a visual embedding—a numerical vector that summarizes its salient visual characteristics—used to improve the association between tracks and detections. This multimodal approach allows for the combination of spatial prediction and visual similarity, greatly improving robustness in real-world scenarios and significantly reducing *identity switches*, with improvements estimated around 45% on standard benchmarks.

As shown in Figure 2.5, DeepSORT combines kinematic prediction (Kalman filter) with visual appearance descriptors (CNN feature extractor), using a cascade of matching with combined metrics to associate detections with active tracks.

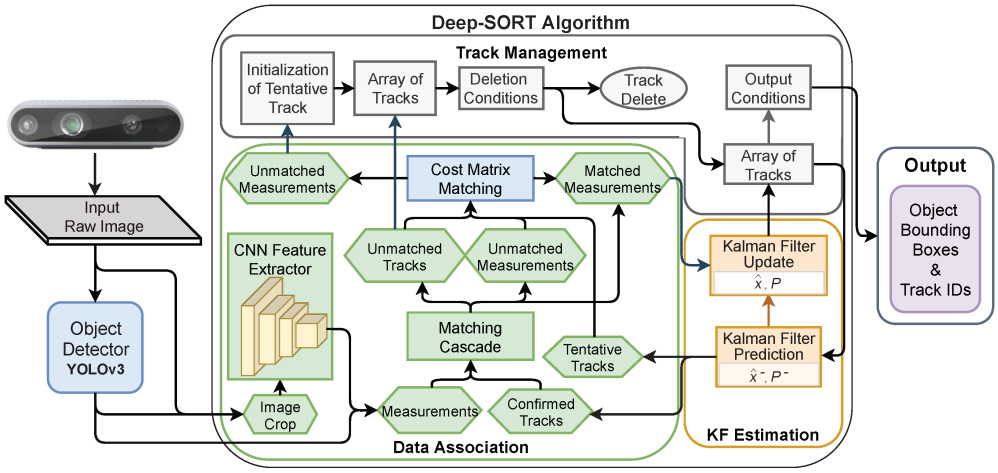


Figure 2.5: Modular architecture of DeepSORT: Kalman filter, visual feature extractor, cost matrix, and association using the Hungarian algorithm.

The DeepSORT system is based on re-ID networks, typically CNNs (e.g., ResNet-50), trained with similarity metrics such as triplet loss or contrastive loss on specific datasets like Market-1501. These embeddings enable effective object association even under challenging conditions such as crowded scenes, rapid perspective changes, or the presence of visually similar objects. In recent years, several studies have proposed even more performant architectures for re-ID, including OSNet, PCB+RPP, and RGA, which provide more discriminative and robust embeddings against intra-class variations. In particular, Koufos et al. (2021) showed that using these variants significantly reduces *identity switch* rates and increases track stability, even in complex sequences with visually similar objects [23].

As shown in Table 2.1, the OSNet, PCB+RPP, and RGA architectures provide more robust embeddings compared to ResNet-50-based models, improving key metrics such as mAP and Rank-1 on the Market-1501 and CUHK03 datasets.

Method	**Publication**	**Backbone**	**Market1501**		**CUHK03**		**Duke**		**MSMT17**	
			R1	mAP	R1	mAP	R1	mAP	R1	mAP
ShuffleNet	CVPR'18	ShuffleNet	84.8	65.0	38.4	37.2	71.6	49.9	41.5	19.9
MobileNetV2	CVPR'18	MobileNetV2	87.0	69.5	46.5	46.0	75.2	55.8	50.9	27.0
BraidNet	CVPR'18	BraidNet	83.7	69.5	–	–	76.4	59.5	–	–
HAN	CVPR'18	Inception	91.2	75.7	41.7	38.6	80.5	63.8	–	–
OSNet (ours)	ICCV'19	OSNet	93.6	81.0	57.1	54.2	84.7	68.6	71.0	43.3
DaRe	CVPR'18	DenseNet	89.0	76.0	63.3	59.0	80.2	64.5	–	–
PNGAN	ECCV'18	ResNet	89.4	72.6	–	–	73.6	53.2	–	–
KPM	CVPR'18	ResNet	90.1	75.3	–	–	80.3	63.2	–	–
MLFN	CVPR'18	ResNeXt	90.0	74.3	52.8	47.8	81.0	62.8	–	–
FDGAN	NeurIPS'18	ResNet	90.5	77.7	–	–	80.0	64.5	–	–
DuaATM	CVPR'18	DenseNet	91.4	76.6	–	–	81.8	64.6	–	–
Bilinear	ECCV'18	Inception	91.7	79.6	–	–	84.4	69.3	–	–
G2G	CVPR'18	ResNet	92.7	82.5	–	–	80.7	66.4	–	–
DeepCRF	CVPR'18	ResNet	93.5	81.6	–	–	84.9	69.2	–	–
PCB	ECCV'18	ResNet	93.8	81.6	63.7	57.5	83.3	69.2	68.2	40.4
SGGNN	CVPR'19	ResNet	92.3	82.8	–	–	81.0	68.2	–	–
Mancs	ECCV'18	ResNet	93.1	82.3	65.5	60.5	84.9	71.8	–	–
AANet	CVPR'19	ResNet	93.9	83.4	–	–	87.7	74.3	–	–
CAMA	ECCV'19	ResNet	94.7	84.5	66.6	64.2	85.8	72.9	–	–
IANet	CVPR'19	ResNet	94.4	83.1	–	–	87.1	73.4	75.5	46.8
DGNet	CVPR'19	ResNet	94.8	86.0	–	–	88.6	74.8	77.2	52.3
OSNet (ours)	ICCV'19	OSNet	**94.8**	**84.9**	**72.3**	**67.8**	**88.6**	**73.5**	**78.7**	**52.9**

Table 2.1: Comparison of OSNet vs altri metodi su Market1501, CUHK03, Duke, MSMT17 [24].

When applying these approaches to the context of waste management, specific challenges emerge that are not present in classical pedestrian tracking scenarios. Waste items deposited in

smart bins may disappear from view during the drop or container opening, remain static for long periods, or exhibit discontinuous and unpredictable movements. Moreover, the visual similarity between common objects, such as bottles, cups, or plastic containers, represents an additional source of ambiguity for conventional CNN embeddings. Under these conditions, the Kalman filter is unable to accurately predict trajectories, while visual similarity can lead to frequent identity switches between distinct yet similar objects.

Another limitation of systems like DeepSORT is the absence of an explicit visual memory. The model operates according to a purely local and frame-based logic, without maintaining a history of visual representations. As a result, it cannot assess whether an object observed at a given moment has already been seen previously. This lack of *visual recurrence* is particularly detrimental in dynamic or intermittent contexts, where an object might exit the scene and reappear after a few seconds, or appear in different positions without coherent motion.

In recent years, several approaches have been proposed in the literature that introduce memory mechanisms into multi-object tracking. Among them, *MeMOT* [25] uses a long-term memory structure to store temporal embeddings associated with tracks, improving the system's resilience to prolonged occlusions or interruptions. More recently, *MeMOTR* [26] introduced a *memory-attention* mechanism based on Transformers, capable of modeling long-range temporal relationships and improving metrics like HOTA by more than 7.9% on benchmarks such as DanceTrack. Other studies from 2024 and 2025 have developed even more compact and high-performing architectures, such as *FASTTrackTr* [27] and *Global Tracking Transformer* [28], capable of operating in real time on edge devices thanks to reduced attention mechanisms and compressible memory.

Although DeepSORT remains a well-established reference for multi-object tracking in relatively structured scenarios, its design principles prove less suited to the specific context of waste management. In particular, preliminary experiments conducted in this domain have highlighted limitations in the model's ability to handle visually similar, static, or incoherently moving objects, leading to numerous association errors and identity switches. Furthermore, the lack of persistent visual memory renders the system incapable of effectively distinguishing between new and previously seen objects, especially in the presence of occlusions, reappearances, or unsupervised scenes. These limitations point to the need for alternative solutions capable of explicitly modeling the temporal recurrence and semantic continuity of objects.

2.3 Visual memory and Memory-Based architectures

The frame-wise processing typical of traditional object detection and tracking models has already been highlighted in previous sections as one of the main limitations in dynamic scenarios. This section analyzes architectural solutions that introduce explicit memory mechanisms, with the aim of overcoming such constraints through temporal persistence and modeling of visual identity over time.

To address these limitations, recent literature has developed models equipped with *explicit memory* mechanisms, integrating persistent and dynamic representations of visual sequences. One of the most significant innovations in this area is the concept of **self-attention**, introduced by Vaswani et al. [29], which revolutionized the modeling of long-range dependencies within sequences. Applied to the visual domain, this technique led to the development of models like ViT [30], which treat images as sequences of patches, offering a richer understanding of spatial and semantic interactions.

Subsequently, video architectures such as *TimeSformer* [31] and *VideoSwin* [32] extended the Transformer paradigm to the temporal domain. Specifically, TimeSformer adopts separable attention along spatial and temporal dimensions to reduce computational complexity, while VideoSwin introduces hierarchical windows of local attention, making it scalable to long or high-resolution sequences while maintaining strong consistency across successive frames.

As illustrated in Figure 2.6, the diagram shows how video Transformers (e.g., TimeSformer) apply attention separately—first along the temporal dimension, then along the spatial dimension—and finally combine the two into a joint representation.

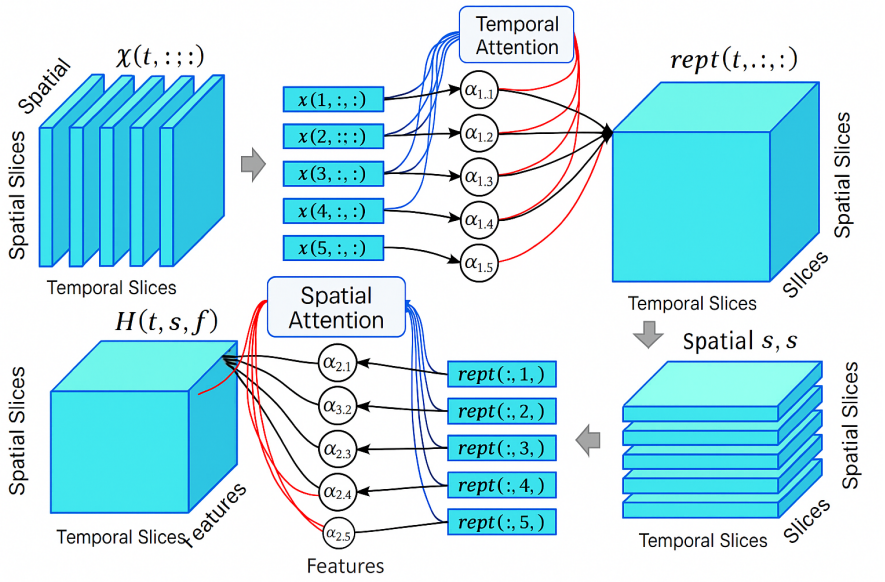


Figure 2.6: Spatio-temporal self-attention mechanism (time-then-space) used in video Transformer architectures [33] .

In parallel, the first **Memory-Augmented Neural Networks** (MANNs) emerged, such as the *Neural Turing Machine* [34] and the *Differentiable Neural Computer* [35], which combine differentiable external memory modules with neural read/write capabilities. These architectures have been successfully applied to complex tasks such as *Video Question Answering* [36], [37], where the model must maintain coherent narrative traces even in the presence of occlusions or discontinuous events. Although promising, these approaches still suffer from high computational costs, especially in real-time scenarios or on edge devices.

In the domain of **multi-object tracking**, the idea of visual persistence has been realized in models such as *MeMOT* [25], which maintains a long-term memory of embeddings associated with tracks, supporting correct reactivation after extended periods of visual absence. *MeMOTR* [26] extended this logic by integrating memory into the decoder of a Transformer, modeling complex temporal relationships and improving metrics such as HOTA by 7.9% on benchmarks like DanceTrack. Other advanced models, such as *Global Tracking Transformer* [28], use a shared visual cache with global attention, while *FASTTrackTr* [27] implements selective attention and compression to maintain high performance even in resource-constrained environments.

An important distinction between these models and conventional architectures is that memory is treated as an external, persistent, and updatable component, separate from the main network. This allows for the construction of a *continuous visual narrative*, enriched frame by frame, making the system more robust to environmental variations, occlusions, and perspective changes.

In parallel, research has addressed the problem of scalability and efficiency. Since full attention has quadratic complexity and differentiable memory structures are expensive to update, models like *MeMViT* [38] introduced *selective buffers* and compressed attention, updating memory only on *key frames*. Additional techniques include *token pooling*, *quantized memories*, FIFO strategies, and *controlled forgetting* to avoid memory saturation.

As shown in Figure 2.7, the MeMViT model uses a selective buffer updated only on key frames, allowing the system to retain visual context without processing every frame. This reduces computational cost while significantly extending temporal support.

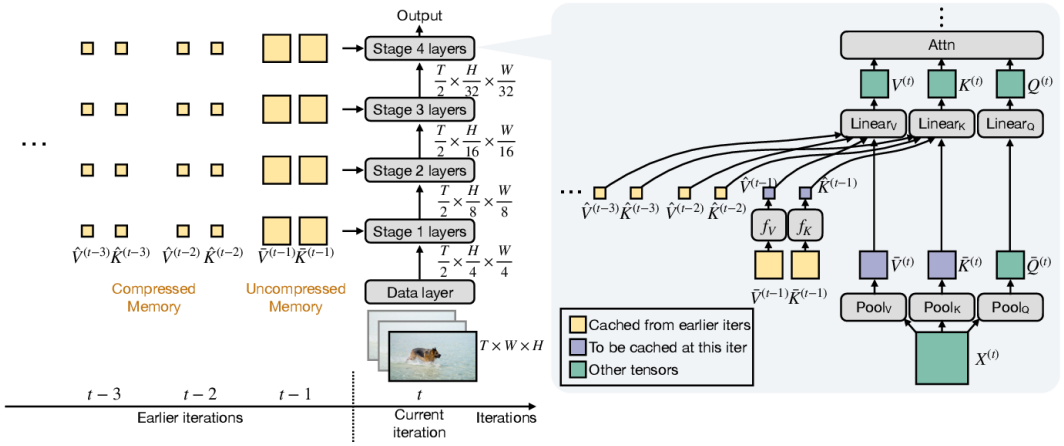


Figure 2.7: Comparison between traditional frame-wise pipeline and MeMViT's selective buffer, with memory updated only on key frames.

In the context of this thesis, an architecture is proposed that integrates detection and tracking with three hierarchical levels of visual memory: **short-term** ($\leq$ 1s), **medium-term** (5–10s), and **long-term** (entire operational cycle). As illustrated in Figure 2.8, each level is equipped with an independent temporal buffer, with specific *attention* and *forgetting* policies, enabling real-time comparison of current embeddings with the accumulated visual history. This structure facilitates recognition of previously observed objects, reduction of false positives, and implementation of a robust and contextual logic of *visual novelty* [39].

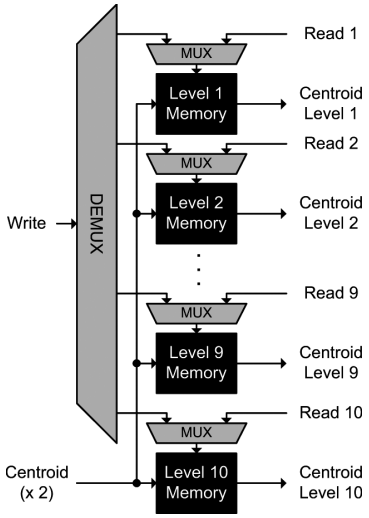


Figure 2.8: Hierarchical architecture of visual memory levels.

Studies on extended benchmarks (Kitti-Mem, MOT20-mem) indicate that the use of structured memory significantly reduces *identity switches*, with improvements of 20–30% compared to conventional trackers. The integration of *metric learning*, through Siamese networks or supervised triplet loss, further strengthens the system's ability to distinguish between truly new objects and visual recurrences.

All of these models represent a necessary transition toward systems endowed with *visual persistence*, capable of performing continuous inference on video streams. Memory-aware architectures represent the technological frontier for embedded applications, making it possible to address the challenge posed by smart bins: understanding whether an object is new or already seen, tracking it over time, and acting accordingly (e.g., ignoring duplicates, estimating quantities).

2.4 Metric learning and Siamese Networks

Metric learning is a paradigm in machine learning aimed at learning a distance function that reflects the semantic relationships between objects. Unlike traditional supervised learning, which seeks to classify inputs into predefined categories, metric learning constructs an embedding space in which similar objects are mapped close together and dissimilar ones far apart, according to a differentiable metric such as Euclidean distance or cosine similarity.

This approach is particularly effective in open-world or open-set contexts, where the system must generalize to new classes never seen during training while maintaining semantic consistency among visually similar instances. A prime example is facial recognition: faces of the same person should be close in the feature space even if they were not part of the training set.

The most well-known architectures for implementing metric learning are **Siamese Networks** and **Triplet Networks**, which share weights across parallel branches and learn a distance between inputs instead of performing explicit classification. The first modern formulation of the Siamese network was proposed by Bromley et al. [40], while its revival in deep learning is attributed to the work of Chopra et al. [41], and later to Schroff et al. with *FaceNet* (2015) [42].

Siamese Networks: consist of two identical subnetworks (with shared weights) that receive two images as input and produce two embedding vectors compared using a distance function. The goal is to minimize the distance between similar pairs (positives) and maximize it for dissimilar pairs (negatives). Training is performed using *contrastive loss* [43], which penalizes errors continuously based on the learned distance.

Triplet Networks: extend the Siamese logic by including an *anchor*, a *positive* (same class), and a *negative* (different class). The *triplet loss* enforces that the distance between anchor and positive is smaller than the distance between anchor and negative by at least a margin m. This margin can be fixed or learned. Triplet loss is used in high-profile systems for re-identification and the retrieval of similar objects in large-scale visual datasets.

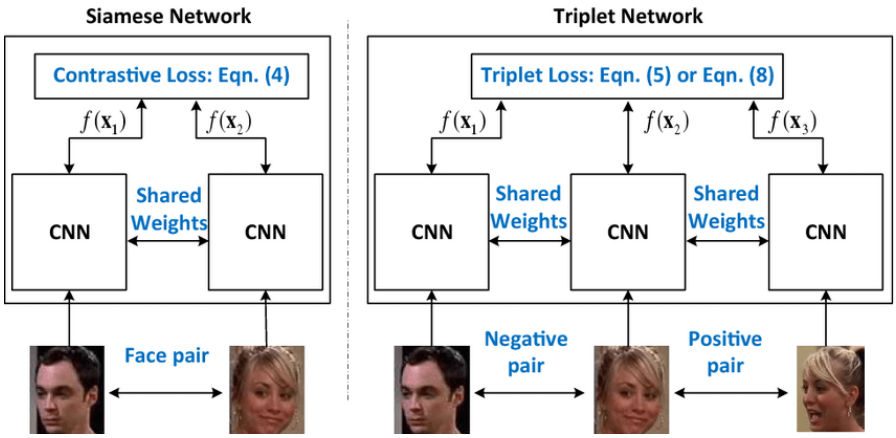


Figure 2.9: Comparison between Siamese Network (two branches with contrastive loss) and Triplet Network (three branches with triplet loss), with shared weights. Image from [42].

In recent years, numerous improvements to traditional loss functions have been proposed. The **batch hard triplet loss** [44] selects the hardest triplets within a batch to maximize useful gradient information. Other variants include **angular loss**, **arcface loss** [45], and **circle loss** [46], which enhance discriminative power in complex scenarios such as person re-ID, face recognition, and product matching.

The effectiveness of these architectures has been demonstrated in contexts with high intra-class and inter-class variability. In commercial applications, metric learning is used in biometric identity verification, deduplication of similar products in e-commerce, deduplication of frames in

video streams, and unsupervised visual clustering. The most widely used benchmarks include LFW, Market-1501, CUHK03, and SOP (Stanford Online Products).

In the domain of **intelligent waste management**, metric learning plays a crucial role in identifying previously seen objects—such as bottles, cans, or boxes reappearing in the video stream—and thus in correctly estimating the concept of *visual novelty*. Since many waste items are similar in shape and texture but not identical, the system's ability to position them correctly in the embedding space is a fundamental requirement.

For example, two plastic bottles may appear at different times, from different angles, but the system must recognize their semantic similarity and decide whether it is a new instance or a reappearance. This problem is further exacerbated in embedded environments with low resolution, fast motion, partial occlusions, and optical distortions, making traditional embeddings unstable.

For this reason, many recent architectures have integrated metric modules into tracking and detection systems. In **DeepSORT**, for example, the re-identification embedding is used directly during the association phase, avoiding errors based solely on position (Kalman). More recent works combine *Transformer-based encoders* with neural metric modules (e.g., MemViT, MeMOTR) capable of comparing a current object with a memory of past embeddings.

Studies such as Re-IDformer [47] and CrossFormer [48] propose hybrid architectures in which the visual backbone extracts embeddings compatible with metric learning, and the matching modules compute similarity between objects separated in time. These architectures are designed for low-power applications and optimized for continuous video datasets.

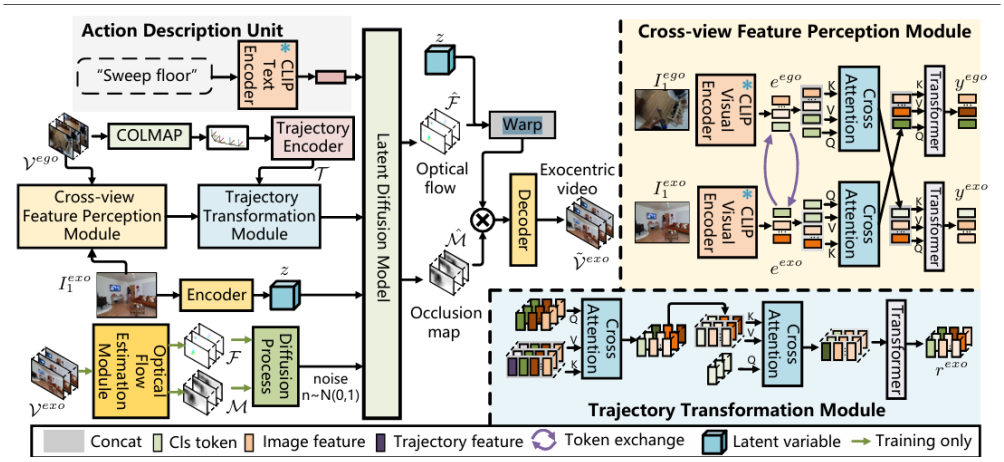


Figure 2.10: Architecture of a video stream metric learning system based on encoder, memory, and distance module. Image from [48].

An important topic in metric learning is the **quality of the pairs** or **triplets**: many recent architectures implement active *mining* strategies to select hard samples, avoiding the network learning only from trivial cases. Online mining algorithms (e.g., semi-hard triplet mining), semantic data augmentation, and guided sampling are essential to prevent overfitting.

In parallel, **lossless matching modules** have also been developed, where embeddings are directly quantized and compared via hashing, reducing computational cost. These techniques are particularly relevant for embedded systems such as smart bins, where memory and computational power are highly constrained.

In the context of this thesis, the use of metric learning is essential for building a pipeline capable of:

- visually associating objects across time,

- recognizing visual recurrences and reducing false positives,

- estimating whether an object has already been seen or represents a new instance.

The proposed architectures include Siamese modules that compare the current object with a historical buffer of recent embeddings. Similarity is computed using cosine similarity or Euclidean distance, with dynamic thresholds adapted based on distributional statistics.

Recent studies [49], [50] show how combining metric learning with visual attention and memory modules can drastically improve correct identification rates even in the presence of occlusions or partial visual variations. The model presented in this thesis draws inspiration from this paradigm to ensure robust and persistent object recognition in dynamic environments, leveraging the potential of metric networks without compromising latency.

2.5 Computer vision-based waste management systems

In recent years, growing environmental pressure and the need for urban automation have driven the integration of intelligent technologies into waste management systems. In this context, computer vision has proven to be a fundamental tool to support the automatic recognition, classification, and monitoring of waste, especially in unsupervised environments such as smart bins.

The first solutions based on convolutional neural networks introduced systems capable of classifying waste by material category (plastic, glass, paper, organic, metal) using lightweight CNN models. An example is *WasteNet*, a system designed to run in real time on edge hardware such as the Jetson Nano, achieving an accuracy of 97% on six waste classes [51]. However, while effective in static scenarios, these models did not address temporal dynamics or visual recurrence of objects—key elements for real-world environments.

Subsequently, a more systematic overview of architectures was proposed by Kunwar and Alade, who analyze various implementations of intelligent smart bins combining computer vision, IoT sensors, and inference modules on microcontrollers [52]. Their analysis includes the use of YOLO, MobileNet, EfficientNet, and ResNet for waste recognition, evaluating trade-offs between accuracy, latency, and deployability on low-power devices.

Architectural evolution has led to the adoption of increasingly efficient and accurate models for waste classification and detection. Kuang et al. proposed the use of YOLOv5 for multi-class waste classification on a real-world dataset acquired in both indoor and outdoor environments [53]. The model shows good performance even on partially occluded or deformed objects, but still operates frame-by-frame, with no form of temporal memory.

A further step forward was made by Mithra et al., who describe a YOLOv8-based pipeline equipped with a web interface and predictive modules to incentivize recycling through a rewarding system. Although the system is efficient, it does not implement tracking mechanisms or identity association, limiting itself to instantaneous classification of deposited objects [54].

In an effort to bridge this gap, several studies have begun integrating visual tracking algorithms. A representative example is the *DSYOLO-Trash* system, which combines YOLOv5 with attention modules (CBAM, CotNet) and the DeepSORT tracker to provide continuous estimation of the identity of detected waste in video streams [55]. Results on the TrashNet and MMTrash datasets show significant improvements in mAP and correct association rates, but the system remains frame-independent and does not implement visual memory or metric learning modules.

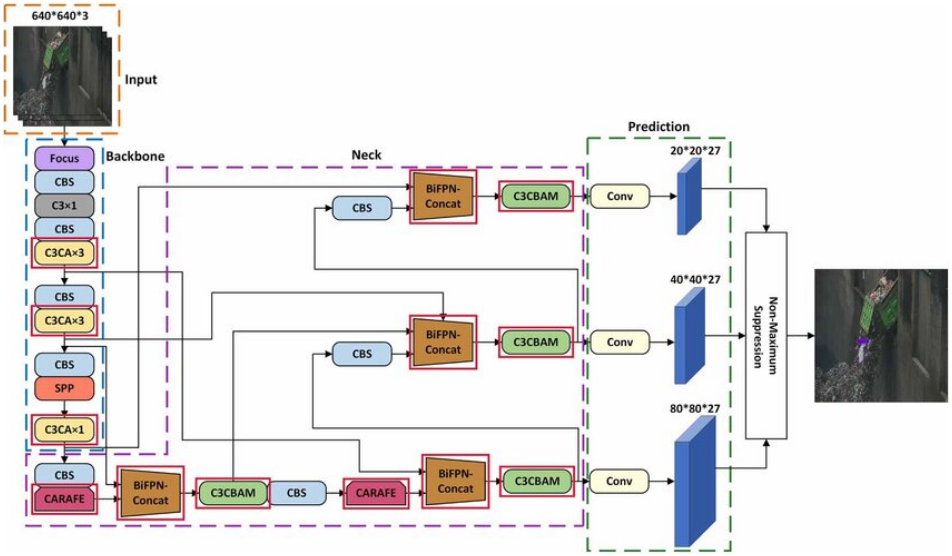


Figure 2.11: Architecture of the Waste-YOLO system for automatic waste classification. The model combines convolutional backbones, attention modules (CBAM), and head structure for real-time recognition on smart bins. Source: [56].

Similarly, Abo-Zahhad and colleagues propose a real-time edge system based on Raspberry Pi, YOLOv8, and ultrasonic sensors to monitor the fill level of waste bins, as well as to detect litter abandoned near the container [57]. The system integrates vision with the Internet of Things (IoT) to notify authorities in case of anomalies, but it does not address the issue of persistent object identity.

Most of the systems mentioned above focus exclusively on detection or waste classification, completely overlooking the problem of recurrence. In dynamic scenarios, such as smart bins placed in urban environments, it is crucial to recognize whether a detected object is new or has previously appeared in the scene, in order to avoid duplicate counting or estimation errors. This requirement calls for explicit visual memory modules, integrable into tracking architectures and combined with metric learning strategies.

In the literature, few studies have systematically addressed the issue of visual persistence. Some recent approaches have begun to explore the use of memory-augmented trackers, such as MeMOT or FASTTrackTr, for general multi-object tracking applications, but the specific application to waste management contexts remains largely unexplored [25], [27]. Moreover, no existing system integrates the concept of visual novelty—namely, the ability to distinguish between previously seen objects and new arrivals based on persistent visual memory and metrically informed embeddings.

The available benchmarks, such as TrashNet, WaDaBa, TACO, and RecycleNet, focus on classification and segmentation, but do not provide annotated datasets for identity tracking or temporal recurrence. This limits the ability to evaluate truly persistent and memory-aware systems, leaving a relevant research gap for embedded applications.

In light of the above, a clear **gap** in the literature emerges: the absence of integrated models that combine detection, tracking, and recurrent recognition of waste objects, operating in real time on embedded devices and equipped with visual memory capabilities.

This scenario represents the central motivation of the present thesis work. In the next chapter, three innovative architectures will be presented, directly addressing the identified limitations. These introduce a detection–tracking–metric learning pipeline equipped with a structured memory across multiple temporal levels, capable of recognizing previously seen objects, managing visual recurrence, and adapting to the computational constraints of real-world embedded waste management systems.

Chapter 3

Dataset and Methodology

3.1 Description of the original Dataset

The dataset used for the present thesis work was provided by **Relearn**, an innovative company committed to developing technological solutions for intelligent waste management. It serves as the experimental foundation upon which the methodologies for recurrent object recognition were designed, developed, and validated, with particular focus on robustness against visual variability, temporal discontinuity, and semantic invariance.

The images were collected through fixed cameras installed near bins designated for differentiated waste collection in real indoor environments, such as offices, public spaces, and corporate open spaces. Users, acting in a completely natural and unsupervised manner, deposit objects into the containers, generating heterogeneous visual streams in terms of type, orientation, angle, distance, occlusion, and ambient lighting. As such, the dataset represents an extremely realistic and diverse collection of scenes containing waste, which can be classified into a wide range of categories.

The ultimate goal of the data collection is not mere object classification, but rather the ability to establish an **identity relationship** between similar objects appearing in different images. This ability is central to applications such as waste traceability, deposit counting, and user behavior analysis.

The dataset is structured according to the standard **COCO** (Common Objects in Context) format, one of the most widely adopted and flexible formats in the field of computer vision. The main annotation file is a JSON object containing five main sections:

- **images**: list of metadata related to each image;
- **annotations**: detailed list of annotations associated with visible objects;
- **categories**: dictionary of semantic classes;
- **licenses** (optional): metadata related to image licensing (not present in this case);
- **info** (optional): general metadata about the dataset (not used).

Each image is described by a JSON object containing the following fields:

- `id`: unique numerical identifier of the image;
- `file_name`: relative path or filename containing the date and time of the image;
- `width`, `height`: image dimensions in pixels;

- `(any custom attributes)`: e.g., timestamp, source or camera location (not included in the provided dataset).

This information enables unique access to the image files and helps establish a geometric context useful during preprocessing or for computing spatial metrics.

Listing 3.1: Example of COCO structure with extensions for visual matching

```
{
  "images": [
    {
      "id": 19,
      "file_name": "real/2024_Week31_Tiny-0606_trash/data/2024-08-08___17-03-20.jpg",
      "width": 1280,
      "height": 1024
    }
  ],
  "annotations": [
    {
      "id": 57,
      "image_id": 19,
      "category_id": 44,
      "bbox": [430.36, 674.09, 263.67, 349.91],
      "segmentation": [[628.76, 1024.0, 674.09, 943.91, 694.03, 898.72, 678.08,
          731.25, 630.23, 709.98, 582.38, 674.09, 586.37, 797.7, 612.95, 811.0,
          615.61, 818.97, 558.45, 880.11, 525.22, 926.64, 430.36, 1024.0]],
      "area": 40930.0,
      "iscrowd": 0,
      "attributes": {
        "new": "no",
        "occluded": false
      }
    },
    {
      "id": 58,
      "image_id": 19,
      "category_id": 46,
      "bbox": [252.74, 501.3, 376.02, 522.7],
      "segmentation": [[600.99, 834.92, 557.12, 543.83, 491.99, 534.53, 482.69,
          502.63, 420.22, 501.3, 394.96, 511.93, 386.99, 526.55, 386.99, 543.83,
          252.74, 594.34, 320.53, 999.74, 324.48, 1024.0, 628.76, 1024.0]],
      "area": 148911.0,
      "iscrowd": 0,
      "attributes": {
        "new": "yes",
        "occluded": false
      }
    }
  ],
  "categories": [
    {
      "id": 44,
      "name": "crumbled tissue"
    },
    {
      "id": 46,
      "name": "paper cup"
    }
  ]
}
```

The `annotations` section is the richest and most central part of the dataset, consisting of a list of annotations, each representing a single object visually identified in a given image.

Each object has the following structure:

- **id**: globally unique identifier of the annotation within the dataset. It is assigned incrementally and sequentially for each annotated object, regardless of its visual identity. As a result, even if the same object appears in multiple images, it will receive a new `id` each time. Therefore, this field does not represent any temporal identity persistence, but only a local reference to the visual instance in that specific image.
- **image_id**: identifier of the image to which the object belongs. This allows the annotation to be directly associated with one of the entries in the `images` section.
- **category_id**: numerical identifier of the semantic class assigned to the object. This value refers to the `categories` section, which maps each `category_id` to a human-readable name.
- **bbox**: the object's bounding box, represented as an array `[x_min, y_min, width, height]`, where:
  - `x_min`, `y_min` are the coordinates of the top-left corner;
  - `width`, `height` define the dimensions of the rectangle.

  The bounding box is useful both for training models based on anchor boxes and as input for cropping or extracting local features.
- **segmentation**: a precise representation of the object's shape, via a list of vertices $(x_1, y_1, x_2, y_2, \dots)$ defining a polygon. The format also supports multiple segmentations (e.g., fragmented objects), but in the dataset at hand, a single manually annotated polygon per object is used.
- **area**: the surface area of the segmented object, automatically computed from the segmentation. It is used during validation to detect possible geometric anomalies.
- **iscrowd**: boolean flag indicating whether the object is a difficult-to-segment crowd or mass. In the provided dataset, it is always set to 0, as each object is annotated individually.
- **track_id** (custom): a custom field that uniquely identifies the same object across multiple images. It is essential for building positive object pairs (i.e., different representations of the same object) and is used for inter-frame matching.
- **attributes** (custom): a dictionary containing additional metadata:
  - `new`: a flag indicating whether the object is "new" with respect to previous images. It can take values such as "yes" or "no".
  - `occluded`: boolean indicating whether the object is partially covered by other elements in the scene.

  These additional attributes were used both for statistical analysis and for balancing the training data.

The richness of this structure makes the annotations suitable for multiple tasks: detection, instance segmentation, classification, tracking, and matching.

The `categories` field contains the definition of the semantic classes present in the dataset. Each category is described by:

- `id`: unique numerical identifier;
- `name`: class name (e.g., "plastic bottle", "paper cup", "crumbled tissue").

The provided dataset includes **53 categories**, varying in frequency and visual complexity.

The distribution of categories is highly unbalanced. Some classes appear very frequently (e.g., `paper cup`, `crumbled tissue`), while others are rare (e.g., `covid test`, `plastic dish`, `laptop charger`). Table 3.1 shows a representative excerpt of the statistics.

Table 3.1: Category statistics: frequency, recurrence, and temporal distribution

Class	**Annotations**	**Recurrences (new=no)**	**Distinct images**	**Consecutive**
ALUMINIUM CAN	1473	759	740	492
ALUMINIUM SHEET	281	126	265	125
ALUMINIUM TRAY	8	4	8	3
CIGARETTE BUTT	87	61	75	52
CIGARETTE PACK	87	51	78	44
COMPOSTABLE PACKAGING	282	141	173	116
CONDIMENT PACKETS	20	4	18	4
COVID TEST	1	1	1	0
CRUMBLED TISSUE	5576	2399	3082	2280
FACE MASK	9	6	5	3
GLASS BOTTLE	807	336	229	161
GLASS JAR	37	30	37	31
LAPTOP CHARGER	1	0	1	0
MEDS BLISTER	8	1	8	1
METAL CAP	51	43	43	36
ORGANIC SCRAPS	1071	442	669	424
PAPER BOWL	289	129	249	130
PAPER CUP	6890	3538	2527	1918
PAPER MAGASINE	48	23	46	24
PAPER PACKAGING	5034	2523	2727	2068
PAPER PLATE	41	17	34	15
PAPER SHEET	801	415	631	400
PAPER SUGAR BAG	191	79	130	72
PAPER TRAY	83	27	60	27
PIZZA BOX	115	56	90	66
PLASTIC BAG	149	74	136	82
PLASTIC BOTTLE	2548	1319	1222	896
PLASTIC BOWL	192	105	154	88
PLASTIC CAP	911	382	656	370
PLASTIC CUP	1927	1347	704	483
PLASTIC CUTLERY	263	112	207	104
PLASTIC DISH	10	4	10	2
PLASTIC GLOVES	9	3	8	3
PLASTIC PACKAGING	1935	899	1511	944
MIXED PAPER-PLASTIC PACKAGING	147	70	140	67
PLASTIC SNACK PACKAGING	2435	1124	1645	1060

(continued on next page)

(continued from previous page)

Class	Annotations	Recurrences (new=no)	Distinct images	Consecutive
PAPER FOOD PACKAGING	37	15	34	13
PLASTIC STICKS	460	334	148	118
PLASTIC STRAW	238	127	203	107
PLASTIC TRAY	169	75	148	71
RECEIPT	743	332	505	296
TEA BAG	54	18	43	17
TETRAPACK	190	125	166	113
TRANSPORT TICKET	22	11	20	10
WOODEN CUTLERY	169	52	137	51
WOODEN STICKS	746	310	516	281
TOBACCO PACK	11	6	11	6
PC ACCESSORIES	6	5	6	2
PLASTIC_JUG	77	41	56	34

Overall, the dataset comprises approximately **7000 images** and a total of **35,325 annotations**. The original image set included 7,150 files, but about 150 were excluded due to corruption or unreadability, as identified by an automated validation script.

A crucial aspect concerns the **sequentiality of the images**. Unlike video datasets, where temporal relationships are explicit and continuous, three scenarios are observed here:

- `Continuous sequences`: series of consecutive images of the same object during successive deposit actions;
- `Sparse repetitions`: objects reappearing in images far apart from each other, without any evident temporal order;
- `Isolated images`: presence of unique objects not observed elsewhere.

Data access was performed via an `AzureML` datastore, dynamically mounted within the working environment. After parsing the annotation JSON file, indexed data structures were created to:

- `Map each image` to its set of annotations;
- `Map categories` to semantic IDs;
- `Identify and group objects` by `track_id`;
- `Split the dataset` into training and validation sets (80%-20%) while preserving the class distribution.

The corrupted images (about 150) were automatically excluded.

The dataset provided by Relearn stands as a heterogeneous, realistic, and high-granularity corpus, ideally suited for complex tasks such as visual matching. The detailed annotations, enriched with non-standard extensions, enabled the development of deep learning models tailored to semantic and visual similarity. The next chapters will present the relabeling operations, mask normalization, and the architectures used to learn discriminative object representations.

3.2 Relabeling and Annotation structure

One of the key aspects of this work involved the critical analysis and partial relabeling of the annotations provided in the original dataset. Although the annotations were initially produced in COCO format with a high level of detail, it was deemed necessary to carry out a systematic process of verification, validation, and manual correction of the labels to ensure data reliability and semantic consistency.

The relabeling process focused in particular on the `attributes` field, and more specifically on the `new` attribute, whose meaning is crucial for the analysis of recurring patterns. In the original version of the dataset, all annotated objects were labeled as `new: yes`, regardless of their actual recurrence within the image sequence. While this approach may have served its purpose during the initial data collection phase, it was unsuitable for more refined analytical goals, such as evaluating object similarity or training models capable of distinguishing between new and previously seen objects.

For this reason, an extensive manual review of the images was initiated. The process involved visualizing each image through the annotation tool provided by the Azure platform, which allows for the overlay of segmented masks and metadata for each object. Through a sequential analysis of the images, a visual comparison was carried out for objects appearing in consecutive or logically connected frames, with the goal of identifying instances where the same object appeared more than once.

In such cases, the `new` attribute was changed from `yes` to `no`, indicating that the object had already been observed in previous images. This update enabled the distinction between truly new and already known objects, thus providing essential information for recurrence analysis and the construction of balanced datasets for matching tasks. While conceptually simple, this operation required meticulous and continuous inspection of each of the over 7,000 images and 35,000 objects in the dataset.

During this process, the `track_id` field was not used. Although available in the COCO format to support tracking tasks, it was not populated in the dataset provided by Relearn. The absence of this information made the `new` attribute even more important, as it served as the only indicator of an object's temporal recurrence. The revision of the `new` field was therefore essential not only for descriptive purposes but also for the construction of structured and reliable subsets on which to conduct experiments in similar-object recognition.

In addition to correcting the `new` labels, the manual review also included the verification of the semantic accuracy of the categories assigned to each object. In some cases, labeling errors were identified, such as objects assigned to incorrect or ambiguous categories (e.g., `paper cup` labeled as `paper packaging`, or `plastic bottle` labeled as `plastic bowl`). These anomalies were corrected manually to ensure greater semantic consistency and improve the quality of the training data for subsequent models.

The restructuring of the dataset did not involve any changes to the overall structure of the JSON file, which remained fully compatible with the COCO standard. However, the quality of the annotations was significantly improved through manual intervention. At the end of the review process, a new updated annotation file was generated, including all revised and relabeled objects. This file was used in all subsequent training and validation phases and served as the basis for generating image pairs for the visual matching task.

Finally, it is worth emphasizing that the absence of automation in the relabeling process made it possible to maintain accurate control over data consistency, minimizing the propagation of systematic errors. Although time-consuming, the manual effort provided an invaluable added value in terms of precision, semantic quality, and annotation reliability, making the dataset not only a solid experimental base but also a useful reference for future developments and validations of computer vision models applied to the context of urban waste.

3.3 Preprocessing and mask management

A fundamental phase in the experimental pipeline was the preprocessing of the images and their corresponding annotations, with particular attention to the management of segmentation masks. This step had a significant impact both on the efficiency of the trained models and on the quality of the visual representations used for matching recurring objects. Since the annotations followed the COCO format, each object in the dataset was equipped with a `bounding box`, and, when available, also a high-precision polygonal segmentation mask. Processing such information required a careful study of the methods for extracting, normalizing, and managing the regions of interest.

The first preprocessing step involved normalizing the image dimensions. Although the original dataset images shared a consistent resolution (e.g., 640x480 pixels), it was still necessary to scale the images and their respective masks to dimensions compatible with the neural architectures used, particularly during the training phase. In many experiments, the resolution was reduced to 224x224 pixels to ensure compatibility with pretrained models (such as ResNet or MobileNet) and to optimize GPU memory usage.

In parallel, automatic cropping of the objects was performed based on their bounding boxes. This made it possible to individually isolate each annotated object, thus obtaining a secondary dataset composed of cropped images containing only the object of interest. This procedure proved useful in multiple phases, such as constructing pairs for training Siamese architectures or extracting visual features independently of the background context. Cropping was accompanied by constant padding around the object to prevent overly aggressive crops that could introduce artifacts at the image edges.

The management of segmentation masks required dedicated handling. The masks provided in the dataset were expressed as lists of polygonal coordinates. These masks were converted into binary maps using libraries such as `pycocotools` and `opencv`. For each object, a binary mask was generated with the same dimensions as the original image, where pixels belonging to the object were marked with a value of 1, and the background with a value of 0. In cases of partially occluded objects, the resulting mask was fragmented, but still representative of the visible surface. These masks were subsequently used to:

- build custom inputs for mask-based architectures;
- compute structural similarity measures between objects, such as Intersection over Union (IoU);
- exclude background during the computation of visual embeddings.

During this phase, special attention was paid to coordinate harmonization, since bounding boxes and masks are often expressed in absolute units relative to the original image resolution. To enable comparison and processing by the models, all coordinates were converted into normalized form (values in [0,1]) with respect to the image size. This ensured independence from the original image scale and improved model generalization during inference stages.

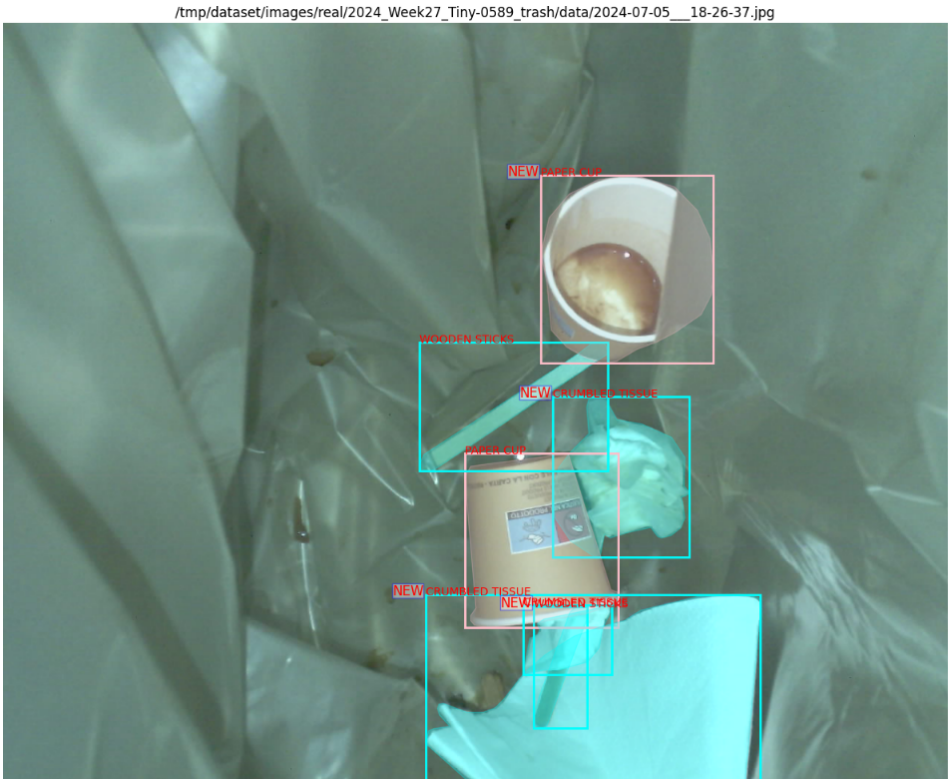


Figure 3.1: Example of an image annotated with bounding boxes (colored by category) and masks partially visible as overlays. Each object is accompanied by its respective class and attributes.

In addition to binary masks, experiments were also conducted with "masked" versions of the images—versions in which only the annotated object was kept visible, while the rest of the image was obscured (zero padding) or filled with a neutral color. The goal was to train the models to focus solely on the salient visual characteristics of the object, reducing the influence of the surrounding context. However, contrary to expectations, this strategy often led to a decline in matching performance, as removing the background deprived the model of contextual information useful for disambiguating visually similar objects.

Finally, all preprocessed data—cropped images, binary masks, masked versions, and corresponding annotations—were organized into indexed structures and saved in `.pkl` or `.npy` format depending on their complexity. This organization enabled fast access during the training phase, avoiding the need to dynamically recompute transformations for each data batch. Additionally, automatic scripts were created for generating positive and negative pairs (based on category or visual similarity) directly from the preprocessed masks.

Preprocessing and mask handling thus proved to be critical steps—not only to ensure the quality and consistency of the dataset, but also to support the implementation of advanced visual content-based matching strategies.

3.4 Initial approach: Manual image comparison

In the early stages of the project, a heuristic approach for recognizing recurring objects was developed, aimed at assessing the feasibility of the task using basic visual comparison tools, without relying on supervised learning models. This approach, entirely *rule-based*, was founded on geometric and visual metrics derived directly from the dataset annotations and from embeddings extracted via pre-trained neural networks.

The guiding principle was to compare each object in a new image with an archive of previously observed objects, in order to determine whether the object was "new" or a repetition of one already seen in earlier images. This decision was based exclusively on two indicators:

- the **cosine similarity** between visual embeddings;
- the **Intersection over Union (IoU)** between their respective bounding boxes.

Cosine similarity is a classical measure in the retrieval literature and is computed as:

$$\text{sim}_{\cos}(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \cdot \|\mathbf{b}\|}$$

where $\mathbf{a}$ and $\mathbf{b}$ are the embedding vectors associated with two objects.

IoU, on the other hand, is a geometric measure that evaluates the intersection between two bounding boxes, normalized by their union:

$$\text{IoU}(B_1, B_2) = \frac{|B_1 \cap B_2|}{|B_1 \cup B_2|}$$

For each object, the cosine similarity was computed between its embedding and those of all objects already stored in the `object memory`, a data structure that stored embeddings, bounding boxes, and categories for each previously observed instance. The candidate object was only compared against objects of the same semantic category, reducing the search space and avoiding comparisons between dissimilar classes (e.g., comparing a plastic bottle with an aluminum tray).

To determine whether an object had been seen before, a combination of the two metrics described above was evaluated. Several aggregation policies were tested: initially, independent thresholds were applied to each metric (e.g., $\text{sim}_{\cos} > 0.75$ **and** $\text{IoU} > 0.5$); later, a composite score was tested, computed as:

$$\text{score} = \alpha \cdot \text{sim}_{\cos} + (1 - \alpha) \cdot \text{IoU}$$

where α was a tunable hyperparameter (typically between 0.6 and 0.8) that controlled the weight given to the visual versus geometric component.

The system assigned a new `object_id` only when no object in memory achieved a score above a predefined threshold. Otherwise, the object was considered "already seen" and was assigned the same ID as the corresponding object. This allowed for the simulation of a primitive similarity-based tracking mechanism, without relying on temporal or contextual information.

The neural network used to extract embeddings was a ResNet-50 pre-trained on ImageNet. For each object, the image was cropped using the annotated bounding box, followed by normalization and resizing to 224×224 pixels. The embedding was extracted by removing the fully-connected layer and applying global average pooling to the output of the penultimate convolutional block.

In the absence of explicit tracking or populated `track_id` fields in the dataset, the `new` attribute was used as a proxy for ground truth to evaluate predictions. This allowed for the calculation of standard metrics such as precision, recall, accuracy, and F1-score. Reasonable performance was observed in simple scenes (few objects, no occlusion), while the approach frequently failed in more realistic scenarios where objects were partially occluded, rotated, or affected by different lighting conditions.

Special attention was given to the selection of thresholds. Systematic tests were conducted by varying the similarity and IoU thresholds over continuous intervals (e.g., $\text{sim}_{\cos} \in [0.6, 0.9]$, $\text{IoU} \in [0.3, 0.7]$) in order to identify the combination that maximized the F1-score across the entire dataset. In some preliminary experiments, a dynamic weight α was also tested, adjusted

according to the number of objects per class, but the marginal benefits did not justify the increased complexity.

The original code used for these experiments was partially lost, but the main logic was reconstructed from later versions in which the heuristic matching component was still present, although commented out. In these versions, additional features were introduced, such as the distance between the centroids of bounding boxes and the ratio of their areas, which were later used to train an MLP classifier. However, at this early stage, such features were not yet used, and classification relied exclusively on cosine similarity and IoU.

A simplified snippet of the code used for the heuristic approach is shown below:

Listing 3.2: Heuristic classification based on cosine similarity and IoU

```
# --- Modello di estrazione delle feature (ResNet50) ---
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
resnet = resnet50(weights=ResNet50_Weights.DEFAULT).to(device)
resnet.eval()
feature_extractor = torch.nn.Sequential(*list(resnet.children())[:-1])

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
        0.225]),
])

def extract_features(image):
    input_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        embedding = feature_extractor(input_tensor).squeeze().cpu().numpy()
    return embedding

def compute_iou(box1, box2):
    x1, y1, x2, y2 = box1
    x1_p, y1_p, x2_p, y2_p = box2

    x_inter1 = max(x1, x1_p)
    y_inter1 = max(y1, y1_p)
    x_inter2 = min(x2, x2_p)
    y_inter2 = min(y2, y2_p)

    inter_area = max(0, x_inter2 - x_inter1) * max(0, y_inter2 - y_inter1)
    area1 = (x2 - x1) * (y2 - y1)
    area2 = (x2_p - x1_p) * (y2_p - y1_p)
    union_area = area1 + area2 - inter_area
    return inter_area / union_area if union_area != 0 else 0

def validate_bbox(bbox, image_shape):
    x1, y1, w, h = map(int, bbox)
    x2, y2 = x1 + w, y1 + h
    h_img, w_img, _ = image_shape
    return [max(0, x1), max(0, y1), min(w_img, x2), min(h_img, y2)]

object_memory = []
results = []

similarity_threshold = 0.68
iou_threshold = 0.5
```

```
sorted_image_ids = sorted(img_map.keys())

for image_id in sorted_image_ids:
    img_data = img_map[image_id]
    img_path = img_data["path"]
    image = cv2.imread(img_path)

    print(f"Processing␣Image␣{image_id}␣-␣{img_path}")

    for ann in img_data["annotations"]:
        bbox = validate_bbox(ann["bbox"], image.shape)
        category_id = ann["category_id"]
        is_new_gt = 1 if ann["attributes"]["new"] == "yes" else 0

        x1, y1, x2, y2 = bbox
        cropped_img = Image.fromarray(cv2.cvtColor(image[y1:y2, x1:x2],
            cv2.COLOR_BGR2RGB))
        embedding = extract_features(cropped_img)

        predicted_new = 1 # default: new
        for obj in object_memory:
            if obj["category"] != category_id:
                continue
            iou = compute_iou(obj["bbox"], bbox)
            sim = 1 - cosine(obj["embedding"], embedding)
            if sim > similarity_threshold and iou > iou_threshold:
                predicted_new = 0
                break

        results.append({
            "image_id": image_id,
            "object_id": ann["id"],
            "category_id": category_id,
            "true_new": is_new_gt,
            "predicted_new": predicted_new
        })

        object_memory.append({
            "category": category_id,
            "embedding": embedding,
            "bbox": bbox
        })

print("\nFinals␣results:")
print(f"Accuracy:{accuracy_score(y_true,␣y_pred):.4f}")
print(f"Precision:{precision_score(y_true,␣y_pred):.4f}")
print(f"Recall:{recall_score(y_true,␣y_pred):.4f}")
print(f"F1-Score:{f1_score(y_true,␣y_pred):.4f}")
```

This snippet shows how each object is compared with previously observed objects of the same category using cosine similarity and IoU. If both values exceed predefined thresholds, the object is considered already seen (‘new=no‘); otherwise, it is classified as new (‘new=yes‘). After classification, the object is stored for future comparisons.

The system allowed for the collection of important preliminary insights:

- visual similarity between objects of the same class can vary significantly depending on viewing angle and lighting;
- IoU proves effective only when the object is positioned similarly within the image, but it is fragile in the presence of rotations or translations;
- overly strict thresholds lead to many false negatives (already seen objects classified as new), while overly permissive thresholds increase false positives;
- the approach fails to generalize on deformable (e.g., crumpled napkins) or poorly structured objects.

This phase provided a first unsupervised baseline, useful for quantifying the difficulty of the problem and guiding the design of subsequent models. Although modest, the obtained performance highlighted that geometric and visual distance metrics alone were insufficient to produce a robust and scalable system. This motivated the introduction of supervised machine learning techniques, described in the following chapters.

3.5 Architecture 1: MLP on cosine similarity

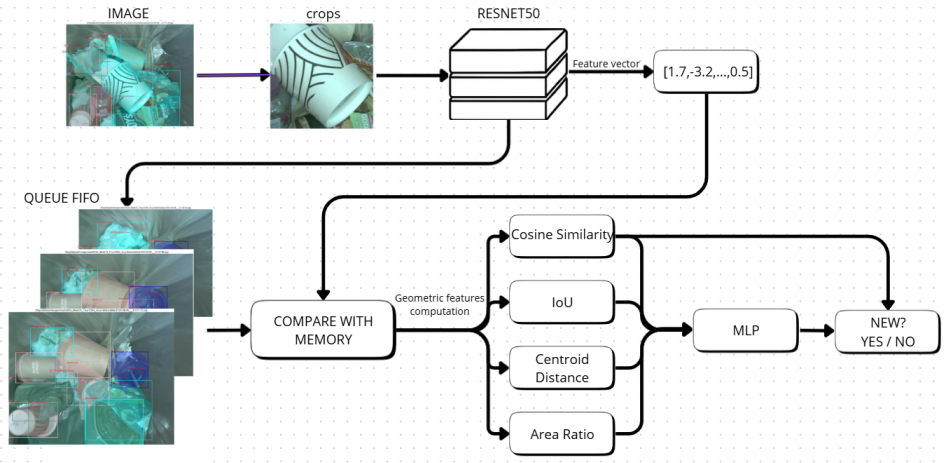


Figure 3.2: Architecture of Baseline

After completing the preliminary heuristic rule-based phase, it became evident that although this approach was useful for understanding the nature of the problem, it was limited in its ability to generalize to complex scenarios. This motivated the design of a first supervised model, with the goal of learning from the data the optimal conditions to distinguish "new" objects from those already seen.

The transition to machine learning-based models represented a fundamental step in the development of a more robust and adaptable system for visual object tracking in real-world environments, with particular focus on the context of waste sorting monitored through computer vision.

In this scenario, a static camera constantly frames a waste container and records changes in its contents. The goal remains to determine, for each visible object, whether it represents a new instance or a recurrence. This distinction is crucial for applications such as accurate disposal counting, behavior tracking, and anomaly detection.

Unlike the rigid thresholds adopted in rule-based approaches, the architecture described here relies on a Multilayer Perceptron (MLP) capable of inferring similarity relationships in a data-driven manner, using a compact representation of the relationship between a "query" object and a "candidate" object in the visual memory. The network receives as input a set of features derived

from the pair (embedding, IoU, centroid distance, area ratio, etc.) and outputs the probability that the two objects represent the same instance.

The rationale behind using an MLP lies in its ability to model non-linear relationships between features and target classes. This is particularly useful in the given context, where similarities between objects are not always explicitly codifiable through simple rules. For example, two objects may have a similar shape but differ in color or scale, or they may be partially occluded. In such cases, the MLP offers a robust solution for integrating different sources of information (visual embeddings, geometric metrics, etc.) and producing a reliable estimate of recurrence.

The core of the approach consists in building a supervised dataset to train the classifier. This dataset was derived from manually annotated data in JSON format following the COCO standard, in which each object is associated with a binary label `new`, specifying whether the object is new (1) or already seen (0). This type of annotation is valuable because it allows the problem to be formulated as a standard binary classification task, where the model must learn to predict whether an object has previously appeared based on a set of features.

The annotations were made available through a remote mount from AzureML, and once mounted, the dataset annotations were loaded from the `latest.json` file and organized into a map that links each image to its local path and corresponding annotations. This step is essential to construct a coherent data structure that allows for efficient iteration over images and associated polygons. Furthermore, a check is performed to verify the existence of each image file in order to discard any invalid references.

The following code snippet (Figure 3.3) shows the annotation parsing process and the construction of the image map.

Listing 3.3: Annotation parsing and image map construction

```
import os
import json

ANNOTATIONS_PATH =
    "annotations/office/waste-object-tracking/real/all/latest.json"

with open(os.path.join(MOUNT_FOLDER, ANNOTATIONS_PATH)) as f:
    annotations = json.load(f)

img_map = {
    i["id"]: {
        "path": os.path.join(MOUNT_FOLDER, "images", i["file_name"]),
        "annotations": []
    } for i in annotations["images"]
}

for a in annotations["annotations"]:
    img_map[a["image_id"]]["annotations"].append(a)

# Filtra immagini mancanti
valid_img_map = {}
for image_id, data in img_map.items():
    if os.path.exists(data["path"]):
        valid_img_map[image_id] = data
img_map = valid_img_map
```

The features selected to describe the relationship between a current object and a past one were carefully chosen, balancing computational simplicity and discriminative power. They include:

- **Cosine similarity**: obtained by comparing the visual embeddings of the two objects. The embeddings are high-dimensional vectors extracted from a ResNet50 pretrained on

ImageNet, and they represent a compact and rich description of the visual content of the cropped image. Cosine similarity provides a measure of how closely the two vectors point in the same direction, with values close to 1 indicating strong visual similarity.

- **Intersection over Union (IoU)**: a standard metric for evaluating the overlap between two bounding boxes. It is calculated as the ratio between the intersection area and the union area of the two boxes. A high IoU value indicates strong spatial overlap, and thus a strong clue that the same object is being observed at different points in time.
- **Euclidean distance between centroids**: measures the distance between the centers of the two bounding boxes. It is an intuitive and simple metric that proves useful in distinguishing distant objects (likely different) from nearby ones (likely the same).
- **Area ratio**: used to assess whether the relative sizes of the two objects are consistent. An object that is much smaller or larger than another may indicate too drastic a scale change to justify identity.

These four features represent an effective trade-off between visual and spatial information. The first two (cosine similarity and IoU) combine semantic and geometric dimensions, while the latter two offer additional geometric cues to refine the decision.

To collect the training data, an automated procedure was implemented that iterates over images in temporal order, building a FIFO memory of recently observed objects. Each new object is compared with those in the memory, and if a compatible match (same category) is found, the pair is added to the dataset with label 0; otherwise, the label is 1. This process enabled the generation of thousands of examples without the need for additional manual annotations.

Listing 3.4: Supervised dataset construction for the MLP model

```
# Setup percorsi
MOUNT_FOLDER = "/tmp/dataset"
IMAGES_FOLDER = os.path.join(MOUNT_FOLDER, "images")
ANNOTATIONS_PATH =
    "annotations/office/waste-object-tracking/real/all/latest.json"

# Caricamento annotazioni
with open(os.path.join(MOUNT_FOLDER, ANNOTATIONS_PATH)) as f:
    annotations = json.load(f)

img_map = {
    i["id"]: {
        "path": os.path.join(IMAGES_FOLDER, i["file_name"]),
        "annotations": []
    } for i in annotations["images"]
}
for a in annotations["annotations"]:
    img_map[a["image_id"]]["annotations"].append(a)

# Feature extractor da ResNet50
resnet = resnet50(weights=ResNet50_Weights.DEFAULT).to(device)
resnet.eval()
feature_extractor = torch.nn.Sequential(*list(resnet.children())[:-1])
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
        0.225]),
])
```

```
def extract_features(image):
    input_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        return feature_extractor(input_tensor).squeeze().cpu().numpy()

def compute_iou(box1, box2):
    xi1 = max(box1[0], box2[0])
    yi1 = max(box1[1], box2[1])
    xi2 = min(box1[2], box2[2])
    yi2 = min(box1[3], box2[3])
    inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)
    area1 = (box1[2] - box1[0]) * (box1[3] - box1[1])
    area2 = (box2[2] - box2[0]) * (box2[3] - box2[1])
    union_area = area1 + area2 - inter_area
    return inter_area / union_area if union_area > 0 else 0

def validate_bbox(bbox, image_shape):
    x, y, w, h = map(int, bbox)
    x2, y2 = x + w, y + h
    h_img, w_img, _ = image_shape
    return [max(0, x), max(0, y), min(w_img, x2), min(h_img, y2)]

# Costruzione dataset
FIFO_SIZE = 20
memory_queue = []
feature_rows = []
SCORE_THRESHOLD = 0.64

for image_id in sorted(img_map.keys()):
    img_data = img_map[image_id]
    image = cv2.imread(img_data["path"])
    if image is None:
        continue

    for ann in img_data["annotations"]:
        bbox = validate_bbox(ann["bbox"], image.shape)
        category_id = ann["category_id"]
        x1, y1, x2, y2 = bbox
        crop = image[y1:y2, x1:x2]
        if crop.size == 0:
            continue

        cropped_img = Image.fromarray(cv2.cvtColor(crop, cv2.COLOR_BGR2RGB))
        embedding = extract_features(cropped_img)

        best_match, best_score = None, -1

        for mem in memory_queue:
            if mem["category"] != category_id:
                continue
            cosine_sim = 1 - cosine(mem["embedding"], embedding)
            iou = compute_iou(mem["bbox"], bbox)
            score = 0.6 * cosine_sim + 0.4 * iou
            if score > best_score:
                best_match = mem
                best_score = score
```

```
                best_cosine, best_iou = cosine_sim, iou

        if best_score > SCORE_THRESHOLD and best_match:
            label = 0
            cx1, cy1 = (x1 + x2) / 2, (y1 + y2) / 2
            cx2, cy2 = (best_match["bbox"][0] + best_match["bbox"][2]) / 2,
                (best_match["bbox"][1] + best_match["bbox"][3]) / 2
            centroid_distance = np.sqrt((cx1 - cx2)**2 + (cy1 - cy2)**2)
            area1 = (x2 - x1) * (y2 - y1)
            area2 = (best_match["bbox"][2] - best_match["bbox"][0]) *
                (best_match["bbox"][3] - best_match["bbox"][1])
            area_ratio = area1 / area2 if area2 != 0 else 0
        else:
            label = 1
            best_cosine = best_iou = centroid_distance = area_ratio = 0

        feature_rows.append({
            "cosine_similarity": best_cosine,
            "iou": best_iou,
            "centroid_distance": centroid_distance,
            "area_ratio": area_ratio,
            "label": label
        })

        memory_queue.append({
            "embedding": embedding,
            "bbox": bbox,
            "category": category_id
        })
        if len(memory_queue) > FIFO_SIZE:
            memory_queue.pop(0)

pd.DataFrame(feature_rows).to_csv("mlp_baseline_features.csv", index=False)
```

The MLP model was structured in a simple yet effective way:

- an **input layer** with 4 neurons (one for each feature);
- a first **hidden layer** with 32 neurons and ReLU activation function;
- a second **hidden layer** with 16 neurons and ReLU activation;
- an **output layer** with 1 neuron and sigmoid activation, returning the probability that the object is "new."

The choice to use a relatively small network was due to the tabular nature of the data and the low dimensionality of the features. A deeper network would likely have led to overfitting without bringing significant performance improvements. Furthermore, the simplicity of the model allows for faster inference, which is an important requirement in real-time applications.

Before training the model, the features were normalized using a *StandardScaler*, which transforms each variable to have zero mean and unit standard deviation. This step is crucial to ensure that all features contribute equally to learning, preventing those with a larger numerical range from dominating the optimization process.

Listing 3.5: Definition and training of the MLP network

```
X_train_tensor = torch.tensor(X_train, dtype=torch.float32)
```

```
y_train_tensor = torch.tensor(y_train.values,
    dtype=torch.float32).unsqueeze(1)

train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)

class BaselineMLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(4, 32), nn.ReLU(),
            nn.Linear(32, 16), nn.ReLU(),
            nn.Linear(16, 1), nn.Sigmoid()
        )
    def forward(self, x):
        return self.model(x)

mlp_model = BaselineMLP()
optimizer = optim.Adam(mlp_model.parameters(), lr=1e-3)
criterion = nn.BCELoss()

for epoch in range(10):
    mlp_model.train()
    for batch_X, batch_y in train_loader:
        optimizer.zero_grad()
        loss = criterion(mlp_model(batch_X), batch_y)
        loss.backward()
        optimizer.step()
```

The dataset was then split into a training set and a validation set with an 80/20 ratio, ensuring that the class distribution was preserved (stratification). Training was performed for 10 epochs, with a batch size of 64, using the Adam optimizer and Binary Cross Entropy as the loss function. At each epoch, the model was evaluated on the validation set, and key metrics such as accuracy, precision, recall, and F1-score were monitored in order to select the best-performing configuration.

Listing 3.6: Loading, normalization, and splitting of the dataset

```
df = pd.read_csv("mlp_baseline_features.csv")

X = df[["cosine_similarity", "iou", "centroid_distance", "area_ratio"]]
y = df["label"]

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, stratify=y, random_state=42)
```

Finally, at the end of the training process, the trained model was saved in PyTorch format (.pth), along with the scaler, so that it could be easily reloaded for subsequent inference.The entire pipeline was designed to be modular, reusable, and extensible, making the model immediately available for operational use during the inference phase, which represents the next step in the pipeline and will be described in the following section.

Listing 3.7: Final evaluation and model saving

```
X_test_tensor = torch.tensor(X_test, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test.values, dtype=torch.float32).unsqueeze(1)
```

```
mlp_model.eval()
with torch.no_grad():
    y_pred_probs = mlp_model(X_test_tensor).squeeze()
    y_pred = (y_pred_probs >= 0.5).int()

print(classification_report(y_test, y_pred))

torch.save(mlp_model.state_dict(), "mlp_baseline_model.pth")
joblib.dump(scaler, "mlp_baseline_scaler.pkl")
```

3.5.1 Integration into the inference pipeline

After the training phase, the MLP model is integrated into the inference pipeline, which processes in real time the sequence of images captured by a static camera. This process constitutes one of the fundamental aspects of the architecture, as it transforms a supervised model into an active operational component capable of autonomously contributing to the tracking logic. Integration into the inference phase thus represents the crucial transition from experimental design to practical deployment.

The pipeline processes images one by one in temporal order, since each frame is captured by a fixed camera at regular intervals. For every new image processed, the system detects the present objects using the available annotations (or a detection system, if present), and for each object it performs a systematic comparison with previously seen objects stored in a FIFO (First-In, First-Out) memory structure. This memory keeps a history of the objects encountered in previous frames, with a maximum limit on the number of stored objects to prevent uncontrolled growth in computational cost.

The main objective of this phase is to determine, for each current object, whether there exists a compatible past object—i.e., one belonging to the same semantic category and with sufficiently similar geometric and visual characteristics. For each compatible pair (current object and FIFO memory object), the same four features used during training are computed: cosine similarity between visual embeddings, Intersection over Union (IoU) between bounding boxes, Euclidean distance between centroids, and the area ratio of the boxes. At the implementation level, the visual embedding of the current object is computed by passing the cropped image through a pre-trained ResNet50, truncated before the last fully connected layer, to obtain a representative vector of salient visual features.

Once computed, the features are preprocessed using the same scaler (StandardScaler) employed during training, in order to ensure statistical consistency with the data on which the model was optimized. This step is critical: without normalization, the feature distributions could differ between training and testing, significantly degrading model performance.

The normalized features are then passed to the MLP model, which returns a probability value between 0 and 1. This value represents the model's estimate that the object is "new" with respect to the observed memory. If the probability is below a predefined threshold (0.5 in the base case), the object is considered already seen (label 0) and is associated with the compatible candidate that produced the lowest probability (i.e., the one most likely to represent the same instance). On the other hand, if all probabilities are above the threshold, the object is assumed to be previously unseen, and a new identity (label 1) is assigned.

Regardless of the assigned classification, all objects from the new image are added to the FIFO memory, enriching the available context for subsequent images. In this way, the memory is continuously updated with every observed object, providing an incremental overview useful for future comparisons.

This decision mechanism allows the system to identify, at any given moment, the visual history of each object across frames, providing a form of semantic tracking of occurrences. Moreover,

using a trained model enables the system to overcome many of the limitations associated with direct comparisons and heuristic thresholds. For example, objects that are partially rotated, deformed, captured under different lighting conditions, or affected by visual noise can still be correctly recognized by the model, as long as their visual embeddings remain consistent with previous instances.

From a computational perspective, the inference phase is well optimized. The extraction of visual features is the most demanding operation, but it can be performed in batches to accelerate the process. Furthermore, the use of a FIFO memory with a limited length reduces the number of comparisons to perform at each frame, ensuring acceptable scalability even in real-time conditions. In the future, the entire system could be further optimized through parallelization or GPU-friendly implementations suitable for embedded environments.

Interestingly, the inference also plays a feedback role within the system: every object classified as "new" is stored in the FIFO, contributing to the system's evolving knowledge. This continuous cycle of comparison and update makes the system dynamic, adaptive, and constantly refreshed, helping to create a consistent temporal context for tracking. In other words, the pipeline implicitly learns the recent history of the visual content, basing its predictions on contextual memory.

The integration of the model into the inference pipeline is also fully modular. This means that the classifier can be easily replaced with a more sophisticated model (e.g., a deeper network or a classifier with visual attention) without altering the core logic of the pipeline. Similarly, the feature set can be expanded or enriched with new components (e.g., dominant color, texture, spatial frequency), enhancing the model's discriminative power without requiring structural changes to the architecture.

Another relevant aspect is that the model outputs a continuous probability, which provides an indication of the network's confidence in its predictions. This value can be interpreted to enrich the decision logic or simply monitored for diagnostic or analytical purposes, as will be further discussed in the next section.

Finally, the robustness of the integration also manifests in the handling of edge cases. Situations where an object appears highly deformed compared to its first appearance, or cases where similar objects are deposited close together, can still be resolved thanks to the model's ability to learn complex representations. The adoption of a learning-based approach, rather than one based on fixed rules, allows the system to adapt to unforeseen scenarios, extending its generalization capabilities.

This mechanism not only improves the system's accuracy and robustness, but also lays the foundation for even finer control of the decision logic, governed by the learned threshold that will be analyzed in the next section.

Listing 3.8: Integration of the MLP model into the inference pipeline

```
# === Setup iniziale ===
MOUNT_FOLDER = "/tmp/dataset"
IMAGES_FOLDER = os.path.join(MOUNT_FOLDER, "images")
ANNOTATIONS_PATH = os.path.join(MOUNT_FOLDER,
    "annotations/office/waste-object-tracking/real/all/latest.json")

with open(ANNOTATIONS_PATH) as f:
    annotations = json.load(f)

# Costruzione mappa immagini
img_map = {
    i["id"]: {"path": os.path.join(IMAGES_FOLDER, i["file_name"]),
        "annotations": []}
    for i in annotations["images"]
}
for a in annotations["annotations"]:
    img_map[a["image_id"]]["annotations"].append(a)
```

```
# Filtra immagini effettivamente presenti
filtered_img_map = {k: v for k, v in img_map.items() if
    cv2.imread(v["path"]) is not None}

# === Caricamento modelli ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Estrazione feature con ResNet50
resnet = resnet50(weights=ResNet50_Weights.DEFAULT).to(device).eval()
feature_extractor = torch.nn.Sequential(*list(resnet.children())[:-1])
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224,
        0.225]),
])

def extract_features(image):
    tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        return feature_extractor(tensor).squeeze().cpu().numpy()

def compute_iou(box1, box2):
    xi1, yi1 = max(box1[0], box2[0]), max(box1[1], box2[1])
    xi2, yi2 = min(box1[2], box2[2]), min(box1[3], box2[3])
    inter = max(0, xi2 - xi1) * max(0, yi2 - yi1)
    area1 = (box1[2]-box1[0]) * (box1[3]-box1[1])
    area2 = (box2[2]-box2[0]) * (box2[3]-box2[1])
    return inter / (area1 + area2 - inter) if inter else 0

# === Inizializza rete e scaler ===
class MLP(nn.Module):
    def __init__(self, input_dim=4, hidden_dim=32):
        super().__init__()
        self.model = nn.Sequential(
            nn.Linear(input_dim, hidden_dim), nn.ReLU(),
            nn.Linear(hidden_dim, 16), nn.ReLU(),
            nn.Linear(16, 1), nn.Sigmoid()
        )
    def forward(self, x): return self.model(x)

mlp_model = MLP().to(device)
mlp_model.load_state_dict(torch.load("mlp_baseline_model.pth",
    map_location=device))
mlp_model.eval()
scaler = joblib.load("mlp_baseline_scaler.pkl")

# === Inference loop ===
object_memory = deque(maxlen=20)
results = []

for image_id in sorted(filtered_img_map.keys()):
    image = cv2.imread(filtered_img_map[image_id]["path"])
    current_objects = []
```

```
    for ann in filtered_img_map[image_id]["annotations"]:
        bbox = validate_bbox(ann["bbox"], image.shape)
        crop = image[bbox[1]:bbox[3], bbox[0]:bbox[2]]
        cropped_img = Image.fromarray(cv2.cvtColor(crop, cv2.COLOR_BGR2RGB))
        embedding = extract_features(cropped_img)

        category_id = ann["category_id"]
        true_label = 1 if ann["attributes"]["new"] == "yes" else 0
        predicted_label = 1 # Assume nuovo finche non si trova match
        found = False

        for memory in object_memory:
            for mem_obj in memory:
                if mem_obj["category"] != category_id: continue

                sim = 1 - cosine(mem_obj["embedding"], embedding)
                iou = compute_iou(mem_obj["bbox"], bbox)
                cx1 = (bbox[0] + bbox[2]) / 2
                cy1 = (bbox[1] + bbox[3]) / 2
                cx2 = (mem_obj["bbox"][0] + mem_obj["bbox"][2]) / 2
                cy2 = (mem_obj["bbox"][1] + mem_obj["bbox"][3]) / 2
                centroid_dist = ((cx1 - cx2)**2 + (cy1 - cy2)**2)**0.5
                area1 = (bbox[2] - bbox[0]) * (bbox[3] - bbox[1])
                area2 = (mem_obj["bbox"][2] - mem_obj["bbox"][0]) *
                    (mem_obj["bbox"][3] - mem_obj["bbox"][1])
                area_ratio = area1 / area2 if area2 != 0 else 0

                features = [[sim, iou, centroid_dist, area_ratio]]
                features_tensor = torch.tensor(scaler.transform(features),
                    dtype=torch.float32).to(device)
                with torch.no_grad():
                    prob = mlp_model(features_tensor).item()

                if prob <= 0.5:
                    predicted_label = 0
                    found = True
                    break
            if found: break

        current_objects.append({"embedding": embedding, "bbox": bbox,
            "category": category_id})
        results.append({
            "image_id": image_id,
            "object_id": ann["id"],
            "category_id": category_id,
            "true_new": true_label,
            "predicted_new": predicted_label
        })

    object_memory.append(current_objects)

# === Valutazione finale ===
df = pd.DataFrame(results)
print("Accuracy:", accuracy_score(df["true_new"], df["predicted_new"]))
print("F1-Score:", f1_score(df["true_new"], df["predicted_new"]))
```

3.5.2 Decision making through learned thresholds

One of the most innovative elements introduced in Architecture 1 is the shift from a fixed-threshold decision system to one based on supervised statistical learning. In traditional computer vision contexts, it is common to use empirical thresholds to determine whether two objects represent the same instance. For example, an object might be considered "already seen" if the cosine similarity between its visual features and those of an object in memory exceeds 0.7, and their bounding boxes have an Intersection over Union (IoU) greater than 0.5. While these values are often derived from experimental observation or engineering intuition, they represent rigid compromises that are not necessarily optimal and are difficult to adapt to unforeseen conditions.

By using a Multilayer Perceptron (MLP), this binary and rigid logic can be replaced with a learned, continuous, and flexible function. During the training phase, the neural network learns a complex mapping between the four input features (cosine similarity, IoU, centroid distance, area ratio) and a class probability (new vs. already seen), optimizing a loss function (binary cross entropy) on the labeled data. In this scenario, the concept of a "threshold" is reinterpreted: it is no longer a single, static value, but the result of a nonlinear combination of multiple indicators, each weighted according to its learned importance.

Operationally, this means that the final decision no longer depends on the simultaneous satisfaction of isolated conditions, but rather on the continuous output of the model. This output—a probability between 0 and 1—expresses the system's confidence that the object is "new." The decision threshold applied to this output (typically 0.5) thus becomes a secondary parameter, easily adaptable to the application context, while the bulk of the decision-making complexity is absorbed by the model during the learning process.

This approach offers several advantages. First, it allows the modeling of complex interactions between features. For example, an object with a medium cosine similarity but a very small centroid distance might still be considered "already seen"; conversely, two objects with similar bounding boxes but very different visual embeddings should be treated as distinct. Such relationships, which could not be encoded using simple fixed thresholds, emerge naturally from the model training, which learns to discriminate based on the full feature configuration.

A second crucial aspect is the ability to calibrate the operational threshold on the model's output value. While 0.5 represents a neutral decision point (equally distant between the two classes), there is no restriction on shifting this threshold to suit specific needs. In contexts where it is a priority to avoid false positives (i.e., wrongly classifying a new object as already seen), the threshold can be raised to 0.6 or 0.7, increasing decision strictness. Conversely, in scenarios where avoiding false negatives (i.e., overlooking already seen objects) is more critical, the threshold can be lowered to 0.3 or 0.4. This type of control allows the system's behavior to be tailored to application-specific rather than purely statistical criteria.

Furthermore, the learned-threshold approach is more robust to variations in the data. Fixed thresholds are highly sensitive to changes in the data domain (e.g., new lighting conditions, changes in framing, new object categories), and require non-trivial manual retuning. The MLP model, if trained on a sufficiently large and diverse dataset, is instead capable of generalizing to such conditions while maintaining stable performance. In other words, the system's intelligence no longer resides in the engineer manually tuning the thresholds, but in the model's ability to learn regularities from observed data.

Another interesting characteristic of this approach is that the probabilistic output of the model can be analyzed beyond the mere decision-making process. It provides a continuous measure of confidence, which can be used to trigger fallback mechanisms, request confirmation, or enable advanced logging. For example, in the presence of a prediction with an output close to 0.5, the system may flag the decision as "uncertain" and retain it for post-processing review. This behavior is particularly relevant in high-criticality environments, such as industrial applications or quality control scenarios.

The differentiable nature of the model also allows for its extension or integration into more complex architectures. For instance, the MLP classifier could be replaced with a deeper network, integrated into an end-to-end pipeline alongside a segmentation network, or enhanced

through attention mechanisms that dynamically weigh the most relevant features for the decision. Moreover, the threshold itself can be made dynamic: instead of being a fixed value, it could be modulated based on context (e.g., object type, elapsed time, frequency of similar object occurrences), or even learned by a meta-model.

From a theoretical standpoint, the transition from fixed thresholds to a learned classifier aligns with the principles of modern machine learning. While rule-based approaches belong to the tradition of explicit programming, the use of data-driven statistical models is rooted in the belief that the optimal behavior of a complex system cannot be predetermined, but must emerge through observation. The architecture described here fully embodies this principle, demonstrating how even in seemingly simple problems—such as the visual recurrence of objects—significant advantages can be gained by adopting a paradigm shift.

In practical terms, the learned threshold also introduces an element of transparency into the system. Although the model is not as interpretable as an explicit rule, post-hoc analyses can be performed on the classifier's decisions by examining the feature configurations that lead to a given prediction. Furthermore, the distribution of probabilities output by the model can be visualized over a dataset to assess the separability of the two classes and to identify ambiguous or out-of-distribution cases.

Lastly, it is worth noting that the choice of the MLP classifier is neither arbitrary nor exclusive. It is a coherent choice given the complexity of the problem and the nature of the features, but other alternatives would also be possible, such as support vector machines, decision tree ensembles (e.g., random forest, gradient boosting), or even Bayesian models. However, the MLP stands out for its ability to model non-linear relationships, its computational efficiency, and its simplicity of implementation and training, making it a balanced choice for this baseline.

The replacement of a fixed threshold with a learned decision mechanism concretely improves system performance and marks a conceptual shift in the design of visual tracking pipelines. It allows the system to adapt to the actual characteristics of the data, to offer more flexible and reliable decisions, and to provide a solid foundation for future developments. In this sense, this architecture is not only an effective solution to the specific problem at hand but also a paradigmatic example of how artificial intelligence can elegantly and efficiently replace heuristics with learning.

3.6 Architecture 2: Memory-Augmented Network

The second proposed architecture, named **Memory-Augmented Network** (MAN), introduces a significant evolution over the previous one by incorporating a differentiable memory component based on a *Transformer Encoder* to explicitly model the recent history of observed objects. This choice stems from the need to enhance the representation of visual-temporal context and to overcome the limitations inherent in the use of static metrics and pairwise comparisons. The traditional paradigm of trackers based on fixed thresholds and local metrics often proves inadequate in realistic scenarios characterized by temporary occlusions, partial deformations, or variations in scale and viewpoint.

MAN was thus designed to enable a more robust and flexible approach, capable of maintaining an updated and adaptive representation of tracked objects by leveraging memory to capture visual dynamics across consecutive frames. This approach becomes particularly relevant in environments where image frequency is not constant or in the presence of short and heterogeneous sequences. In such cases, relying solely on rigid pairwise comparisons can lead to systematic association errors between similar but non-identical objects.

Whereas in the first architecture object comparison was performed through an MLP operating on isolated features (visual embeddings, IoU, centroid distance, area ratio), MAN introduces a more sophisticated mechanism of *contextual embedding*, where each current object (referred to as the *query*) is compared against an external memory composed of past objects (*key-value*) and dynamically updated through multi-head self-attentive interaction. The key difference is

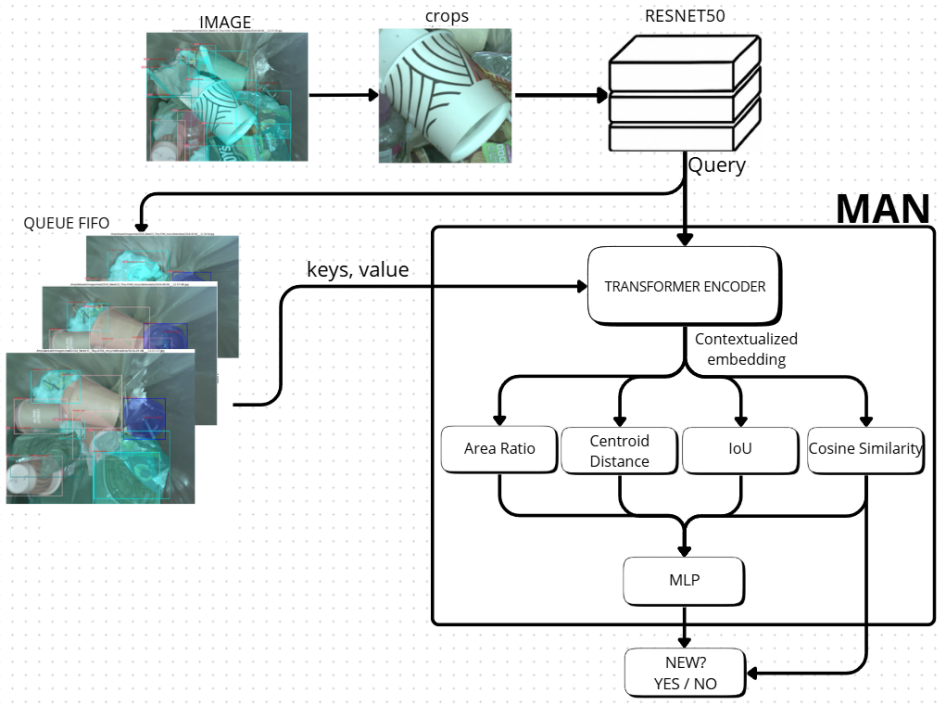


Figure 3.3: MAN architecture with Transformer

that, instead of analyzing each pair independently, the new approach considers the entire recent visual context simultaneously, ensuring consistency and stability even in conditions of perceptual ambiguity.

The core idea is to provide the system with an *explicit memory* of previously observed objects, capable not only of storing visual information but also of actively influencing the representation of the current object. The output of this interaction is a new, "enhanced" (contextualized) embedding, which incorporates information about what has been recently observed and how similar it is to the object under examination. This mechanism also allows for better handling of semantic drift phenomena, in which an object partially changes its appearance while maintaining the same semantic identity (e.g., a rotated or partially emptied bottle).

From an implementation standpoint, the memory is structured as a FIFO (First-In, First-Out) queue, in which the embeddings of objects observed in previous frames are stored. Each memory element includes not only the visual embedding but also geometric information (bounding box coordinates, area) and semantic category. The data structure used is a categorized dictionary, allowing efficient retrieval of compatible objects for each new query. At every new frame, the network performs a comparison between each current object and all compatible objects in memory (i.e., of the same semantic category). The object's representation is then refined through a *Transformer Encoder*, which uses the memory as context to modify the query. The maximum number of stored elements per category can be dynamically adjusted to balance historical coverage and computational efficiency.

The pipeline is structured into the following main steps:

1. Extraction of the annotated images and their corresponding objects using bounding boxes or segmentations.
2. Extraction of the crops (or masks) from the objects, followed by normalization and transformation using a pre-trained ResNet50.
3. Conversion of the crops into embedding vectors (512-dimensional), normalized using L2 norm.

Listing 3.9: Feature extractor based on ResNet50

```
import torch.nn as nn
import torch.nn.functional as F
from torchvision import models

class FeatureExtractor(nn.Module):
    def __init__(self, output_dim=512):
        super().__init__()
        base_model =
            models.resnet50(weights=models.ResNet50_Weights.DEFAULT)
        self.feature_extractor =
            nn.Sequential(*list(base_model.children())[:-1])
        self.projection = nn.Linear(2048, output_dim)

    def forward(self, x):
        features = self.feature_extractor(x) # [B, 2048, 1, 1]
        features = features.view(features.size(0), -1) # [B, 2048]
        projected = self.projection(features) # [B, output_dim]
        return F.normalize(projected, p=2, dim=1) # Normalizzazione L2
```

Listing 3.10: Image transformations for ResNet50

```
import torchvision.transforms as T

transform = T.Compose([
    T.Resize((224, 224)),
    T.ToTensor(),
    T.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

4. Organization of these embeddings into query-memory pairs and computation of the same four geometric features (previously described in Section 3.5): cosine similarity, IoU, centroid distance, and area ratio.

5. Application of the Transformer module that combines the query with the memory to generate a contextualized embedding.

6. Concatenation of the new embedding with the memory object and the geometric features, followed by final classification via MLP.

At the dataset level, the model is supervised using binary labels directly derived from the `new` attribute present in the COCO annotations. If an object is annotated as "new", its match with objects in memory is considered negative (label = 0); conversely, if it has already been observed, a positive match can be automatically inferred, reducing the need for manual annotation. To balance the dataset and avoid class imbalance issues, a stratified sampling strategy was introduced to ensure the presence of positive and negative pairs in balanced proportions. Additionally, matches between objects of different categories or with incompatible sizes were excluded to avoid semantic ambiguity and reduce noise in the data.

Listing 3.11: MAN dataset construction and FIFO memory management

```
from collections import deque
from tqdm import tqdm

# === PARAMETRI ===
MAX_MEMORY_IMAGES = 20
COSINE_THRESHOLD = 0.7
IOU_THRESHOLD = 0.3
```

```
MAX_IMAGES = 3500
FEATURE_DIM = 512

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
feature_extractor = FeatureExtractor(output_dim=FEATURE_DIM).to(device)
memory_queue = deque(maxlen=MAX_MEMORY_IMAGES)
man_dataset = []

# === LOOP SULLE IMMAGINI ===
for img_id in tqdm(image_ids, desc="Creazione␣dataset␣MAN"):
    img_data = img_map[img_id]
    img_path = img_data["path"]
    anns = img_data["annotations"]

    # Estrazione crop e feature
    crops, valid_anns = [], []
    for ann in anns:
        try:
            crop = extract_crop(img_path, ann, transform)
            crops.append(crop)
            valid_anns.append(ann)
        except:
            continue

    if not crops:
        continue

    batch = torch.stack(crops).to(device)
    with torch.no_grad():
        feats = feature_extractor(batch)

    # Confronta ogni oggetto con la memoria FIFO
    for i, ann in enumerate(valid_anns):
        f_q = feats[i]
        cat_q = ann["category_id"]
        is_new_q = ann["attributes"].get("new") == "yes"
        bbox_q = ann["bbox"]

        if len(memory_queue) == 0:
            continue

        for mem in memory_queue:
            for mem_obj in mem:
                if mem_obj["category_id"] != cat_q:
                    continue

                f_m = mem_obj["embedding"].to(device)
                bbox_m = mem_obj["bbox"]

                # Calcolo delle metriche
                cos = F.cosine_similarity(f_q, f_m, dim=0).item()
                iou = bbox_iou(bbox_q, bbox_m)
                cent_dist = centroid_dist(bbox_q, bbox_m)
                area_r = area_ratio(bbox_q, bbox_m)

                label = 0
```

```
                if not is_new_q and cos >= COSINE_THRESHOLD and iou >=
                    IOU_THRESHOLD:
                    label = 1

                # Aggiunta al dataset
                man_dataset.append({
                    "query": f_q.cpu(),
                    "memory": f_m.cpu(),
                    "cosine_sim": cos,
                    "iou": iou,
                    "centroid_dist": cent_dist,
                    "area_ratio": area_r,
                    "label": label
                })

    # Aggiorna la memoria FIFO
    current_objs = [{
        "embedding": feats[i].detach().cpu(),
        "category_id": ann["category_id"],
        "bbox": ann["bbox"]
    } for i, ann in enumerate(valid_anns)]
    memory_queue.append(current_objs)

torch.save(man_dataset, "man_train_dataset.pt")
```

This architecture offers several key advantages over the previous one:

- **Dynamic contextualization**: the query is modified based on the available memory, making the comparison context-dependent and improving discrimination in ambiguous cases.
- **Flexible decision-making**: the system learns to combine embeddings and geometric features without relying on rigid thresholds, adapting fluidly to different scenarios.
- **Extensibility**: the Transformer component can be easily scaled with additional layers or attention heads, or replaced with alternative memory mechanisms (e.g., attention pooling, recurrent mechanisms).

The final model therefore includes two main components:

- A **Memory Transformer**, which enhances the representation of the current object based on the context provided by the FIFO memory.
- A **Three-layer MLP**, which receives as input the contextualized embedding, the memory object's embedding, and the four geometric features, and outputs a probability indicating whether the two objects represent the same instance.

Training is performed on a dataset of `(query, memory)` pairs dynamically constructed from annotated sequences.

Listing 3.12: ManMLPDataset class for loading query-memory pairs

```
class ManMLPDataset(Dataset):
    def __init__(self, path):
        self.data = torch.load(path) # Carica i dati da file .pt

    def __len__(self):
        return len(self.data) # Numero di esempi
```

```
    def __getitem__(self, idx):
        sample = self.data[idx]
        q_feat = sample["query"] # Feature embedding della query
        m_feat = sample["memory"] # Feature dell'oggetto in memoria
        extras = torch.tensor([
            sample["cosine_sim"],
            sample["iou"],
            sample["centroid_dist"],
            sample["area_ratio"]
        ], dtype=torch.float32) # Feature geometriche
        label = torch.tensor(sample["label"], dtype=torch.float32) #
            Etichetta binaria
        return q_feat, m_feat, extras, label
```

Listing 3.13: Training loop for MAN+MLP

```
# === Parametri ===
BATCH_SIZE = 64
EPOCHS = 10
LR = 1e-4
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# === Dataset & DataLoader ===
train_dataset = ManMLPDataset("man_train_dataset.pt")
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)

# === Inizializzazione modello, loss e ottimizzatore ===

# Il modello ManWithMLP verra' mostrato nel dettaglio piu' avanti
model = ManWithMLP(feature_dim=512).to(DEVICE)
criterion = nn.BCELoss()
optimizer = optim.Adam(model.parameters(), lr=LR)

# === Ciclo di training ===
model.train()
for epoch in range(EPOCHS):
    total_loss = 0.0
    for q_feat, m_feat, extras, label in train_loader:
        q_feat, m_feat, extras, label = q_feat.to(DEVICE), m_feat.to(DEVICE),
            extras.to(DEVICE), label.to(DEVICE)
        optimizer.zero_grad()
        output = model(q_feat, m_feat, extras)
        loss = criterion(output, label)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_loss = total_loss / len(train_loader)
    print(f"[Epoch␣{epoch+1}]␣Loss:␣{avg_loss:.4f}")

torch.save(model.state_dict(), "man_mlp_model.pth")
```

The model's final predictions, expressed in probabilistic form, enable fine-grained control of the decision-making process, with thresholds adjustable based on the operational context. The entire architecture is designed for real-time environments, thanks to the modularity of the Transformer and the computational simplicity of the classifier. Performance can be further improved through temporal data augmentation techniques or by adopting hard negative mining

strategies, aimed at selecting challenging pairs to enhance the classifier's discriminative power.

The core innovation of this solution therefore lies in the model's ability to learn not only "how much" an object resembles a previously seen one, but also "in what context" such similarity occurs, delegating to the Transformer the responsibility of dynamically adapting internal representations. This concept will be analyzed in detail in the next subsection, dedicated to the use of the Transformer as an adaptive memory.

Listing 3.14: Final tracking with MAN+MLP and evaluation

```
# === Inizializzazione ===
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
feature_extractor = FeatureExtractor().to(device)
model = ManWithMLP().to(device)
model.load_state_dict(torch.load("man_mlp_model.pth", map_location=device))
model.eval()
feature_extractor.eval()
memory_queue = deque(maxlen=20)

# === Funzioni ===
def extract_crop(img_path, ann, transform):
    img = Image.open(img_path).convert("RGB")
    x, y, w, h = map(int, ann['bbox'])
    return transform(img.crop((x, y, x + w, y + h)))

def bbox_area(b): return b[2] * b[3]
def bbox_iou(boxA, boxB): ...
def centroid_distance(b1, b2): ...
def area_ratio(b1, b2): ...

# === Tracking ===
all_preds, all_labels, all_max_sim = [], [], []
for img_id in sorted(img_map.keys()):
    ...
    for i, ann in enumerate(valid_anns):
        ...
        for mem in memory_queue:
            ...
            with torch.no_grad():
                pred = model(query_feat, mem_feat, extras).item()
            scores.append(pred)

        is_new = True
        max_score = max(scores) if scores else 0.0
        if scores and max_score >= 0.5:
            is_new = False
        ...
        all_preds.append(1 if is_new else 0)
        all_labels.append(1 if ann["attributes"].get("new") == "yes" else 0)
        all_max_sim.append(max_score)
    ...
# === Metriche finali ===
from sklearn.metrics import accuracy_score, precision_score, recall_score,
    f1_score, roc_curve, auc

accuracy = accuracy_score(all_labels, all_preds)
precision = precision_score(all_labels, all_preds)
recall = recall_score(all_labels, all_preds)
```

```
f1 = f1_score(all_labels, all_preds)

fpr, tpr, thresholds = roc_curve(all_labels, all_max_sim, pos_label=0)
roc_auc = auc(fpr, tpr)
```

3.6.1 Using the Transformer as memory

One of the core components of the MAN architecture is the use of a **Transformer Encoder** as a differentiable memory mechanism. This choice is inspired by the effectiveness demonstrated by Transformers in modeling complex relationships among sets of interdependent elements—a capability initially developed in the field of Natural Language Processing (NLP), but which has found widespread application in other domains as well, such as computer vision and, more specifically, object tracking in video sequences. The Transformer can learn latent structures and dynamically weigh relationships between elements, making it an ideal choice for refining visual representations over time.

In the context of the Memory-Augmented Network, the Transformer is used to process the *query*—i.e., the embedding of the currently observed object—by leveraging as context a FIFO memory containing the embeddings of objects previously seen in earlier frames. This process fundamentally differs from rigid point-to-point comparisons: here, the representation of the current object is enhanced through *multi-head attention*, allowing the model to distribute importance across multiple stored objects, selecting and integrating useful information differently for each attention head.

The Transformer module is implemented using PyTorch's `nn.TransformerEncoder`. It is configured with two sequential layers and four independent attention heads (multi-head attention), maintaining an input dimensionality of 512, consistent with the output of the ResNet50 used to generate the visual embeddings. The entire module operates in *batch-first* mode, facilitating parallel sequence processing and ensuring compatibility with batched data flows during training and inference. Despite being relatively compact, this configuration is sufficient to capture complex interactions within the visual-temporal domain.

Listing 3.15: MemoryTransformer module to enhance the query with FIFO memory

```
class MemoryTransformer(nn.Module):
    def __init__(self, feature_dim=512, n_heads=4, n_layers=2):
        super().__init__()
        encoder_layer = nn.TransformerEncoderLayer(
            d_model=feature_dim, nhead=n_heads, dim_feedforward=1024,
                batch_first=True
        )
        self.encoder = nn.TransformerEncoder(encoder_layer,
            num_layers=n_layers)

    def forward(self, queries, memory_bank):
        if memory_bank.size(0) == 0:
            return queries # se la memoria e' vuota, ritorna la query
                originale

        memory = memory_bank.unsqueeze(0).expand(queries.size(0), -1, -1) #
            [B, M, F]
        queries = queries.unsqueeze(1) # [B, 1, F]
        concat = torch.cat([queries, memory], dim=1) # [B, 1+M, F]
        enhanced = self.encoder(concat) # [B, 1+M, F]
        return enhanced[:, 0, :] # Estrae la query potenziata
```

The operational flow of the Transformer within the MAN pipeline unfolds through five main steps:

1. The **query**, namely the embedding [F] of the current object, is reshaped into a tensor of shape [1, 1, F] to match the expected input format of the module.
2. The **memory**, i.e., the set of embeddings from objects in previous frames (up to a maximum of M = 20), is aggregated into a tensor of shape [1, M, F], where each row represents a distinct stored object.
3. The query and memory are concatenated along the temporal dimension (second dimension), producing a combined input of shape [1, M+1, F]. The first position corresponds to the query; the remaining positions correspond to memory elements.
4. This input is processed through the Transformer's self-attention and feed-forward mechanisms. Each embedding can interact with all others, updating itself based on contextual information. The query thus receives an injection of memory.
5. After processing, the first position of the resulting tensor is extracted, which corresponds to the *enhanced* query: a contextualized vector enriched with relations to the FIFO memory.

The result is a new embedding that reflects not only the intrinsic features of the observed object but also the visual-historical context in which it is situated. This is particularly useful in ambiguous situations—for example, when visually similar objects appear at different times or exhibit slight variations in pose, lighting, or occlusion. The Transformer is capable of adapting the contribution of each memory element based on global coherence, applying soft attention to automatically select the most relevant information.

An additional advantage of this strategy lies in its **end-to-end differentiability**. Every component—from memory construction to the generation of the final embedding—is included in the computational graph and can be optimized via backpropagation. This allows the system to automatically learn the optimal combination of historical patterns and visual representations, without relying on hand-coded heuristics or static similarity thresholds.

A notable implementation detail is the absence of traditional *positional encodings*, which are common in standard Transformers. In our case, memory objects are not organized in a structured sequence, as in natural language, but rather form an unordered set. This reflects the nature of the problem: the absolute temporal position is not relevant—only the relative quality and informativeness of each memory element.

During experimentation, it was observed that the Transformer behaves robustly even in the presence of noise or incomplete memory. When non-informative elements are present, the learned attention tends to ignore them, focusing on the most relevant ones. This emergent behavior proved particularly useful in real-world scenarios, where the quality of visual observations can vary drastically due to occlusions, scale or angle variations, and segmentation artifacts.

Finally, the architecture is designed to be scalable. It is possible to increase the number of Transformer layers to enhance abstraction capacity, raise the number of attention heads for finer segmentation of contextual relationships, or integrate masking modules that restrict the Transformer's visual field to semantic or spatial subsets of the memory. These extensions have not yet been included in the base version of the system, but they represent promising directions for the continuous improvement of the model.

The final output of this module is a rich, context-informed visual representation, which is then integrated with the embedding of the candidate object in memory and the geometric features for binary classification. This contextualized embedding forms the foundation upon which the subsequent similarity evaluation by the MLP is based.

Listing 3.16: ManWithMLP module: Transformer + MLP for binary classification

```
class ManWithMLP(nn.Module):
    def __init__(self, feature_dim=512):
        super().__init__()
        self.man = MemoryTransformer(feature_dim=feature_dim, n_heads=4,
            n_layers=2)
```

```
        self.mlp = nn.Sequential(
            nn.Linear(feature_dim * 2 + 4, 128), # query potenziata + memory
                + 4 feature
            nn.ReLU(),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, 1),
            nn.Sigmoid() # output [0,1]
        )

    def forward(self, q_feat, m_feat, extras):
        enhanced_q = self.man(q_feat, m_feat) # Potenzia la query con il
            contesto
        combined = torch.cat([enhanced_q, m_feat, extras], dim=1)
        out = self.mlp(combined).squeeze(1)
        return out
```

3.6.2 Matching with Masks or Bounding boxes

One of the most critical design decisions in visual tracking is the choice of spatial representation for objects. Specifically, within the Memory-Augmented Network architecture, we explored the alternative between using simple **bounding boxes** (bbox) and using **segmentation masks** for extracting image crops to be fed into the feature extractor and subsequent stages of the model.

Bounding boxes represent each object using an axis-aligned rectangle defined by coordinates `[x, y, w, h]`, where `x, y` denote the top-left corner, and `w, h` correspond to width and height. This is a simple representation, compatible with most public datasets (e.g., COCO), and enables fast and efficient extraction. The resulting crops are then transformed (resized, normalized) and projected into the feature domain through a pretrained ResNet50, as described in Section 3.5.

Segmentation masks, on the other hand, rely on more precise information: each object is described by a list of 2D coordinates defining its actual contour. From these polygons, a binary mask is constructed and used to keep only the relevant pixels visible while replacing the background with a neutral color (typically gray). The crop is extracted around the extended bounding box of the mask, optionally applying padding. This strategy reduces background noise and improves the visual clarity of the input, though at the cost of higher computational complexity.

Both strategies present advantages and limitations.

Use of Bounding Boxes (bbox):

- *Pro*: computational simplicity, direct compatibility with many datasets (e.g., COCO), fast extraction, no need for advanced geometric operations.
- *Con*: often includes portions of the background, which may mislead the model, especially in cluttered scenes or with adjacent objects.

Use of Segmentation Masks:

- *Pro*: more precise extraction, reduced visual noise from the background, better semantic consistency across examples of the same category.
- *Con*: increased computational complexity (mask rendering and composition), potential loss of useful contextual information (e.g., object-environment relations).

During the experimentation phase, both approaches were implemented and compared both visually and functionally. The two key functions employed are:

- `extract_object_crop()`: extracts the crop using the bounding box, by directly slicing the rectangular region from the original image.

Listing 3.17: Crop extraction from bounding box

```
def extract_object_crop(img_path, annotation):
    if not os.path.exists(img_path):
        raise FileNotFoundError(f"File␣not␣found:␣{img_path}")
    img = Image.open(img_path).convert("RGB")
    x, y, w, h = map(int, annotation['bbox'])
    cropped = img.crop((x, y, x + w, y + h))
    return cropped
```

- `extract_soft_masked_crop()`: builds a mask from the segmentation polygons, applies a neutral background, and performs a crop centered on the masked region.

Listing 3.18: Extraction with segmentation mask and neutral background

```
def extract_soft_masked_crop(img_path, annotation, padding=10,
    background_color=(128, 128, 128)):
    img = Image.open(img_path).convert("RGB")
    w_img, h_img = img.size

    # create binary mask
    mask = Image.new("L", (w_img, h_img), 0)
    draw = ImageDraw.Draw(mask)
    for seg in annotation['segmentation']:
        if len(seg) >= 6:
            draw.polygon(seg, fill=255)

    # gray background image
    background = Image.new("RGB", img.size, background_color)
    masked_img = Image.composite(img, background, mask)

    # crop with padding
    bbox = mask.getbbox()
    if bbox is None:
        return masked_img

    x0, y0, x1, y1 = bbox
    x0 = max(0, x0 - padding)
    y0 = max(0, y0 - padding)
    x1 = min(w_img, x1 + padding)
    y1 = min(h_img, y1 + padding)

    return masked_img.crop((x0, y0, x1, y1))
```

A visual comparison between the two strategies revealed clear differences, especially in crowded or occluded conditions: masks help separate adjacent objects, while bounding boxes provide greater spatial context, which is useful when the object undergoes structural variations or is only partially visible. However, if the segmentation is inaccurate or affected by artifacts (e.g., translucent or deformed objects), masking may compromise input quality.

From a training perspective, the pipeline was designed to allow easy switching between the two modes, thanks to a parameterized setting defined at script launch. The modular structure enabled the creation of two parallel versions of the dataset—bbox-based and mask-based—each organized as a collection of query-memory pairs described by:

- Query feature embedding.
- Memory feature embedding.
- Four geometric features: cosine similarity, IoU, centroid distance, area ratio.
- Binary label (0 for different objects, 1 for same object).

The comparison showed that embeddings generated from masks tend to exhibit lower intra-class variance, making it easier for the classifier to distinguish similar objects. However, due to their precision, they are also more sensitive to small variations in pose or perspective. Bounding boxes, on the other hand, while including background portions, preserve a broader context and can offer greater robustness against occlusions or scale changes.

From a computational standpoint, the mask-based approach introduces significant overhead. The operations required to build and apply the mask, blend it with the background image, and perform the final crop are more demanding than simple rectangular slicing. In real-time environments or on embedded devices, this aspect may be a critical limitation.

Ultimately, the choice between the two strategies depends heavily on the application context. The final version of the system adopts the bbox mode as the baseline, due to its greater operational stability and lower dependence on segmentation quality. However, the entire pipeline is compatible with both options and retains the flexibility to alternate between them as needed, allowing empirical evaluation of the performance associated with each configuration without modifying the system's core logic.

3.7 Architecture 3: Siamese Network

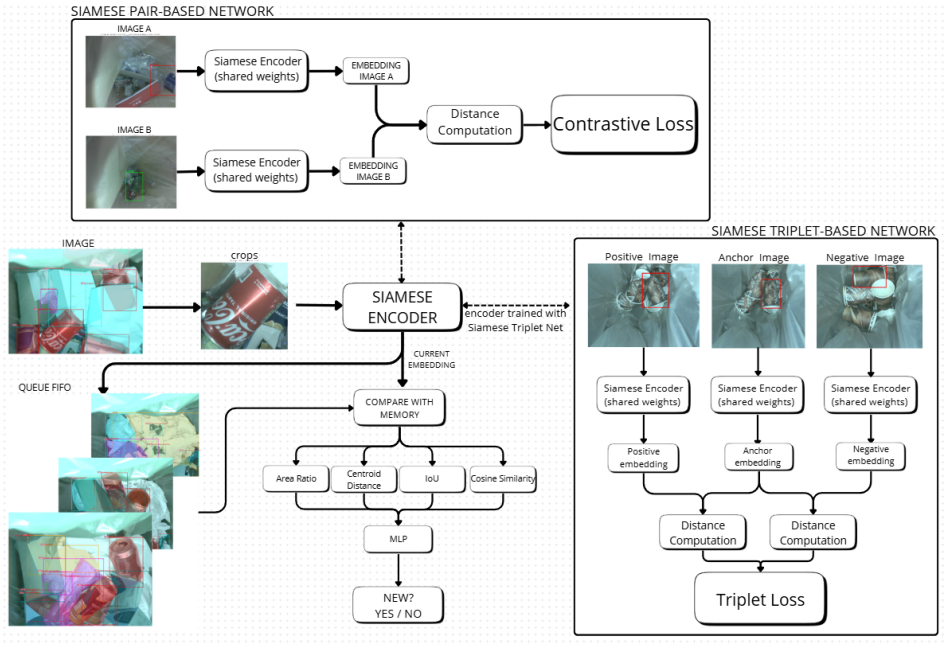


Figure 3.4: Siamese Network architecture based on pairs and triplets.

The third architecture developed in this work is based on the use of a **Siamese Network**, a classical yet highly effective approach in the domain of *metric learning*, designed to learn a similarity function between visual objects based on instance relationships rather than explicit class labels. This paradigm is particularly well-suited for *open-set* or *open-world* scenarios, where the number of object identities is neither fixed nor entirely known, and a system is desired that can generalize to new instances never seen during the training phase.

Unlike the two previous architectures discussed in this thesis, which directly produced a binary decision (match / no match) through a supervised classifier, the Siamese network aims to structure a latent space in which the geometric proximity between representations (embeddings) consistently reflects the semantic similarity between instances. In this space, similar objects (e.g., two different views of the same bottle) should be projected into adjacent regions, while distinct objects (e.g., two different bottles or a bottle and a glass) should be separated by a significant distance. This allows the system to assess similarity between objects in a continuous and flexible way, through the simple measurement of distance rather than rigid classification.

The implemented network is based on a modular and general-purpose architecture, designed to support both of the most common learning modes in metric learning: training on **pairs** of objects (using *contrastive loss*) and training on **triplets** in the form of *anchor-positive-negative* (using *triplet loss*). The decision to support both configurations was guided by empirical and theoretical considerations: the two strategies emphasize complementary aspects of the problem of learning a discriminative metric. While pairs promote absolute learning of distances, triplets encourage relative discrimination between similar and dissimilar examples.

At the core of the architecture is a **shared feature extractor**, based on a pre-trained `ResNet50` model on ImageNet, with the final fully connected layer (`fc`) removed. This backbone takes as input a visual patch (crop from bounding box or segmentation mask), resizes it to 224×224, and extracts a feature vector of dimension 2048. This representation captures salient visual information (shape, color, texture, etc.) in a compact and reusable space.

Listing 3.19: Siamese network with ResNet50 backbone and L2 normalization

```
class EmbeddingNet(nn.Module):
    def __init__(self):
        super().__init__()
        resnet = models.resnet50(pretrained=True)
        modules = list(resnet.children())[:-1]
        self.backbone = nn.Sequential(*modules)
        self.embedding = nn.Linear(2048, 512)

    def forward(self, x):
        x = self.backbone(x)
        x = x.view(x.size(0), -1)
        x = self.embedding(x)
        return F.normalize(x, p=2, dim=1)
```

Following the backbone, a **projection module** (projection head) is introduced, implemented as a Multilayer Perceptron (MLP) composed of two fully connected layers. The first layer reduces the dimensionality from 2048 to 512, followed by a `ReLU` activation and `Dropout` with probability $p = 0.3$; the second layer further compresses the space to 128 dimensions. Formally, the transformation can be expressed as:

$$z = W_2 \cdot \texttt{ReLU}(W_1 \cdot f_{\text{ResNet}}(x) + b_1) + b_2 \tag{3.1}$$

where $f_{\text{ResNet}}(x) \in \mathbb{R}^{2048}$ is the vector produced by the backbone, and $z \in \mathbb{R}^{128}$ is the final embedding. To ensure consistency in the distance metric and improve numerical stability during optimization, the vector z is normalized using the L2 norm:

$$\hat{z} = \frac{z}{\|z\|_2} \tag{3.2}$$

Normalization constrains all embeddings to lie on the surface of the unit hypersphere $S^{127} \subset \mathbb{R}^{128}$, effectively transforming Euclidean distance into a metric equivalent to angular (cosine) distance. This choice is particularly advantageous in the context of metric learning, as it makes

the representation space more homogeneous and prevents collapse or saturation phenomena in distance values.

From an operational standpoint, the network can be used interchangeably to analyze either pairs or triplets. In pair mode, the network processes two images (x_i, x_j), generating two embeddings $\hat{z}_i$ and $\hat{z}_j$; in triplet mode, three images are processed (x_a, x_p, x_n), producing the normalized embeddings $\hat{z}_a$, $\hat{z}_p$, and $\hat{z}_n$. While the loss logic varies between the two configurations, the architecture remains unchanged.

The entire model has been implemented in PyTorch in a modular and flexible way. The main components — backbone, projection head, loss, and sampling strategy — are encapsulated in separate classes, allowing for easy switching between different training configurations. For example, it is possible to replace ResNet50 with a lighter backbone (such as MobileNetV3) for embedded applications, or vary the final embedding size for downstream analysis.

One of the key features of the architecture is its ability to produce representations that can be efficiently compared during inference. Given a set of reference objects $\{x_j\}$ and a query object x_i, it is possible to compute the embeddings $\hat{z}_i$ and $\{\hat{z}_j\}$ and determine matching through the Euclidean distance between normalized vectors:

$$d_{ij} = \|\hat{z}_i - \hat{z}_j\|_2 \tag{3.3}$$

This operation, being a simple computation over vectors, is highly efficient and scalable, making it suitable for real-time applications or deployment on resource-constrained hardware.

Another important aspect of the architecture is its semantic neutrality: the network does not learn to classify objects into predefined categories, but rather to maintain consistency between representations of similar objects. This makes it particularly suitable for scenarios where categories change frequently, or are unknown or undefined a priori, as often occurs in object tracking, video surveillance, or mobile robotics.

The normalization, the architectural choice of the MLP, and the clear separation between the visual and embedding components finally allow for visual monitoring and analysis of the representation space.

The Siamese architecture developed in this work thus stands out for its simultaneous integration of pair- and triplet-based strategies, the adoption of a fully modular and reusable pipeline, and the ability to produce embeddings compatible with a wide range of downstream tasks. In order for this architecture to learn an effective discriminative metric, it is necessary to construct a supervised dataset based on coherent pairs and triplets. The following section describes in detail the process of automatically generating these data and the training procedures adopted.

3.7.1 Pair/Triplet generation and training

The construction of an appropriate dataset represents a fundamental step in training a Siamese network, as the quality and consistency of the pairs or triplets used directly determine the model's ability to learn an effective semantic embedding space. In this work, two different strategies were implemented for the automatic generation of supervised data: the first, based on the creation of labeled pairs, is designed for use with *Contrastive Loss*; the second instead produces *(anchor, positive, negative)* triplets for training with *Triplet Loss*. Both approaches were developed in accordance with the structure and temporal properties of the annotated dataset and contribute to different phases of training and evaluation.

Pair generation constitutes the first step in the data preparation pipeline. This method involves pairing two visual objects extracted from different images or from the same image, and assigning them a binary label based on their semantic and temporal similarity. The goal is to provide the Siamese model with examples of similar objects (label 1) and dissimilar ones (label 0), so that the contrastive loss function can act on the distance between the embeddings of the two images.

The dataset used is annotated in COCO format and includes, for each object, a bounding box, semantic category, and a `new` attribute, which indicates whether the object has been observed in previous frames or has appeared for the first time. Exploiting this temporal information, a pairing criterion was devised that takes into account the consistency of images over time.

The generation of positive pairs is based on comparing objects labeled as "not new" (`new = no`) in a current image with objects present in the two preceding images, with the aim of identifying the same object instance observed at different times. When an object from the previous frames is found belonging to the same category and with high *cosine similarity* between embeddings and an *Intersection over Union* (IoU) above an empirically fixed threshold (in this case 0.6), a positive pair is created. The underlying assumption is that a high IoU and class match within a short time window are strongly indicative of object recurrence.

Negative pairs, on the other hand, are generated in two complementary ways. On one hand, pairs are formed between objects from the same image or consecutive images that belong to different categories. On the other hand, pairs in which one of the two objects is marked as "new" are also considered. In both cases, the pairing provides a negative example that helps push the model to increase the distance between the respective embeddings.

Once the complete set of pairs was generated, a balancing procedure was applied to avoid an excessive predominance of negative examples, which are typically more numerous. To this end, a `neg_ratio` parameter was introduced to limit the maximum number of negative pairs included, generally maintaining a 1:1 ratio with the positive ones. The pairs are then divided into two disjoint sets, one for training and the other for validation, by performing a shuffle followed by a percentage-based split.

The resulting data is serialized in JSON Lines format. Each line contains the paths of the two images, the coordinates of the respective bounding boxes, the semantic categories, the IoU value, and the binary label associated with the pair. This data is then handled through a `SiamesePairsDataset` class implemented in PyTorch, which manages loading, cropping of the regions of interest, and transformation of the images via resizing and normalization. During training, the Siamese network receives the two crops as input and learns to distinguish similar from dissimilar pairs by optimizing the Contrastive Loss.

Listing 3.20: Generation of positive and negative pairs for training with contrastive loss

```
def generate_pairs(img_map, iou_threshold=0.3, neg_ratio=1.0, val_ratio=0.2):
    sorted_img_ids = sorted(img_map.keys())
    positive_pairs = []
    negative_pairs = []

    for idx, img_id in enumerate(sorted_img_ids):
        img_data = img_map[img_id]
        annotations = img_data['annotations']
        objs_new_no = [obj for obj in annotations if obj['attributes']['new']
            == 'no']

        for prev_idx in [idx - 1, idx - 2]:
            if prev_idx < 0:
                continue
            prev_img_id = sorted_img_ids[prev_idx]
            prev_annotations = img_map[prev_img_id]['annotations']
            for obj_a in objs_new_no:
                cat_a = obj_a['category_id']
                bbox_a = obj_a['bbox']
                for obj_b in prev_annotations:
                    if obj_b['category_id'] != cat_a:
                        continue
                    iou = bbox_iou(bbox_a, obj_b['bbox'])
                    if iou >= iou_threshold:
```

```
                    positive_pairs.append({
                        'img_a_path': img_data['path'],
                        'img_b_path': img_map[prev_img_id]['path'],
                        'bbox_a': bbox_a,
                        'bbox_b': obj_b['bbox'],
                        'category_id_a': cat_a,
                        'category_id_b': obj_b['category_id'],
                        'label': 1,
                        'iou': round(iou, 4)
                    })

        for next_idx in [idx + 1, idx + 2]:
            if next_idx >= len(sorted_img_ids):
                continue
            next_img_id = sorted_img_ids[next_idx]
            next_annotations = img_map[next_img_id]['annotations']
            for obj_a in annotations:
                cat_a = obj_a['category_id']
                new_a = obj_a['attributes']['new']
                for obj_b in next_annotations:
                    cat_b = obj_b['category_id']
                    new_b = obj_b['attributes']['new']
                    cond1 = cat_a != cat_b
                    cond2 = new_a == 'yes' and new_b == 'yes'
                    cond3 = new_a != new_b and new_a == 'no'
                    if cond1 or cond2 or cond3:
                        negative_pairs.append({
                            'img_a_path': img_data['path'],
                            'img_b_path': img_map[next_img_id]['path'],
                            'bbox_a': obj_a['bbox'],
                            'bbox_b': obj_b['bbox'],
                            'category_id_a': cat_a,
                            'category_id_b': cat_b,
                            'label': 0,
                            'iou': round(bbox_iou(obj_a['bbox'],
                                obj_b['bbox']), 4)
                        })

num_pos = len(positive_pairs)
num_neg = int(num_pos * neg_ratio)
negative_pairs = random.sample(negative_pairs, min(num_neg,
    len(negative_pairs)))

all_pairs = positive_pairs + negative_pairs
random.shuffle(all_pairs)

split_idx = int(len(all_pairs) * (1 - val_ratio))
train_pairs = all_pairs[:split_idx]
val_pairs = all_pairs[split_idx:]

os.makedirs("data_pairs", exist_ok=True)
with open("data_pairs/pairs_train.jsonl", "w") as f:
    for p in train_pairs:
        f.write(json.dumps(p) + "\n")
with open("data_pairs/pairs_val.jsonl", "w") as f:
    for p in val_pairs:
```

```
            f.write(json.dumps(p) + "\n")
```

Listing 3.21: Dataset for (positive/negative) pairs with bounding box transformation and cropping

```
class SiamesePairsDataset(Dataset):
    def __init__(self, jsonl_path, transform=None):
        self.pairs = [json.loads(line.strip()) for line in open(jsonl_path)]
        self.transform = transform if transform else self.default_transform()

    def __len__(self):
        return len(self.pairs)

    def __getitem__(self, idx):
        pair = self.pairs[idx]
        img_a = self.load_and_crop(pair['img_a_path'], pair['bbox_a'])
        img_b = self.load_and_crop(pair['img_b_path'], pair['bbox_b'])

        img_a = self.transform(img_a)
        img_b = self.transform(img_b)
        label = torch.tensor(pair['label'], dtype=torch.float32)
        return img_a, img_b, label

    def load_and_crop(self, img_path, bbox):
        img = cv2.imread(img_path)
        if img is None:
            raise ValueError(f"Errore␣nella␣lettura:␣{img_path}")
        x, y, w, h = map(int, bbox)
        crop = img[y:y+h, x:x+w]
        crop = cv2.cvtColor(crop, cv2.COLOR_BGR2RGB)
        return Image.fromarray(crop)

    def default_transform(self):
        return T.Compose([
            T.Resize((224, 224)),
            T.ToTensor(),
            T.Normalize([0.5]*3, [0.5]*3)
        ])
```

In parallel with the generation of pairs, a second module was developed for the automatic construction of supervised triplets, used in training a Siamese architecture based on the *Triplet Loss*. This loss function aims to structure the embedding space so that, for each *anchor* point, a *positive* point belonging to the same class is closer than a *negative* point belonging to a different class, with a minimum guaranteed margin.

Here as well, the construction logic leverages the temporal sequence of the images. For each current image, objects marked with `new = no` are selected as anchor candidates. For each of these, the system attempts to identify a positive object among those present in the two preceding images. The selection criterion for the positive is twofold: the object must belong to the same category and exhibit semantic-visual similarity suggesting it is the same physical object, estimated by combining cosine similarity and IoU (as with the pairs, also here 0.6).

If a valid positive is found, the system proceeds with the search for one or more negatives. These are selected from subsequent images or, alternatively, from the same current image, provided they belong to different categories or are marked as "new." Furthermore, to enrich the training process, multiple negatives per triplet may be included, up to a predefined maximum.

The final result is a heterogeneous collection of triplets, each composed of an anchor, a positive, and a negative, each represented by an image, a bounding box, and a category. The

entire set is then shuffled and split into training and validation, with the data saved into two separate files (`triples_train.jsonl` and `triples_val.jsonl`).

This strategy can be considered a form of *semi-supervised hard triplet mining*, as the selection of examples is guided by heuristic rules and semantic metadata (the `new` label), without performing a true search for the most difficult examples. This compromise allows for the automated construction of an informative triplet dataset, reducing the risk of overfitting or semantic ambiguity.

As in the previous case, the data is handled by a PyTorch class `TripletDataset`, which extracts the crops from the source images and applies the necessary transformations. The model trained with Triplet Loss receives the three samples as input and optimizes the objective function by minimizing the distance between anchor and positive, and maximizing it with respect to the negative beyond a certain threshold.

Listing 3.22: Generation of triplets anchored on existing objects

```
def generate_triplets1(img_map, iou_threshold=0.6, max_negatives=2,
    val_ratio=0.2):
    sorted_img_ids = sorted(img_map.keys())
    triplets = []

    for idx, img_id in enumerate(sorted_img_ids):
        img_data = img_map[img_id]
        annotations = img_data['annotations']
        objs_anchor = [obj for obj in annotations if obj['attributes']['new']
            == 'no']

        for obj_a in objs_anchor:
            cat_a = obj_a['category_id']
            bbox_a = obj_a['bbox']
            for prev_idx in [idx - 1, idx - 2]:
                if prev_idx < 0:
                    continue
                prev_id = sorted_img_ids[prev_idx]
                prev_ann = img_map[prev_id]['annotations']
                same_cat = [obj for obj in prev_ann if obj['category_id'] ==
                    cat_a]

                best_iou = 0
                best_obj = None
                for obj_p in same_cat:
                    iou = bbox_iou(bbox_a, obj_p['bbox'])
                    if iou > best_iou and iou >= iou_threshold:
                        best_iou = iou
                        best_obj = obj_p

                if best_obj:
                    negatives = []
                    for next_idx in [idx + 1, idx + 2, idx + 3]:
                        if next_idx >= len(sorted_img_ids):
                            continue
                        next_id = sorted_img_ids[next_idx]
                        next_ann = img_map[next_id]['annotations']
                        for obj_n in next_ann:
                            if obj_n['category_id'] != cat_a or
                                obj_n['attributes']['new'] == 'yes':
                                negatives.append((next_id, obj_n['bbox']))
```

```
                    if not negatives:
                        others = [obj for obj in annotations if
                            obj['category_id'] != cat_a]
                        for obj_n in others:
                            negatives.append((img_id, obj_n['bbox']))

                    if negatives:
                        for neg_img_id, neg_bbox in random.sample(negatives,
                            min(max_negatives, len(negatives))):
                            triplets.append({
                                "img_anchor_path": img_data['path'],
                                "bbox_anchor": bbox_a,
                                "img_positive_path": img_map[prev_id]['path'],
                                "bbox_positive": best_obj['bbox'],
                                "img_negative_path":
                                    img_map[neg_img_id]['path'],
                                "bbox_negative": neg_bbox,
                                "category_id": cat_a
                            })
                        break

    random.shuffle(triplets)
    split_idx = int(len(triplets) * (1 - val_ratio))
    train_triples = triplets[:split_idx]
    val_triples = triplets[split_idx:]

    os.makedirs("data_triplets", exist_ok=True)
    with open("data_triplets/triples_train.jsonl", "w") as f:
        for t in train_triples:
            f.write(json.dumps(t) + "\n")
    with open("data_triplets/triples_val.jsonl", "w") as f:
        for t in val_triples:
            f.write(json.dumps(t) + "\n")
```

Listing 3.23: PyTorch dataset for loading training triplets

```
class TripletDataset(Dataset):
    def __init__(self, triplet_file, transform):
        self.samples = [json.loads(line) for line in open(triplet_file)]
        self.transform = transform

    def load_crop(self, path, bbox):
        img = cv2.imread(path)
        if img is None:
            raise ValueError(f"Errore␣nel␣leggere␣l'immagine:␣{path}")
        x, y, w, h = map(int, bbox)
        crop = img[y:y+h, x:x+w]
        crop = cv2.cvtColor(crop, cv2.COLOR_BGR2RGB)
        return self.transform(Image.fromarray(crop))

    def __getitem__(self, idx):
        sample = self.samples[idx]
        anc = self.load_crop(sample['img_anchor_path'], sample['bbox_anchor'])
        pos = self.load_crop(sample['img_positive_path'],
            sample['bbox_positive'])
        neg = self.load_crop(sample['img_negative_path'],
            sample['bbox_negative'])
```

```
        return anc, pos, neg

    def __len__(self):
        return len(self.samples)
```

The combined adoption of pairs and triplets allows leveraging the strengths of both approaches. The generation of pairs proves particularly useful for stable and well-balanced pretraining, especially during the early phases of training or in scenarios with limited annotations. The construction of triplets, on the other hand, offers greater generalization potential in defining a structured semantic space, particularly when the model has already been initialized and its discriminative capacity needs to be refined.

Moreover, the generation infrastructure described here integrates seamlessly into the context of a visual tracking system, where it is essential to have compact and comparable representations of objects over time. The semantic richness of the embeddings learned through these methods translates directly into improved performance in distinguishing between recurring and novel objects, as will be discussed in the following chapters.

3.7.2 Loss functions, training strategies, and tracking phase

The training of the Siamese network is carried out in two operational modes, each associated with a different loss function: *Contrastive Loss* for training on pairs, and *Triplet Loss* for training on triplets. Both strategies aim to structure an embedding space in which the distance between vectors coherently reflects the semantic and visual similarity between objects. Unlike traditional classification approaches, where each input is assigned a discrete label, metric learning seeks to define a distance metric that can be generalized even to previously unseen objects.

Contrastive Loss is a fundamental objective function in the context of Siamese networks, as it enables learning an embedding space where the distance between two representations produced by the network indicates the degree of semantic similarity between the corresponding objects. This function was used in the training mode based on image pairs, each labeled as either "similar" ($y = 1$) or "dissimilar" ($y = 0$).

The mathematical formulation adopted is:

$$\mathcal{L} = y \cdot D^2 + (1 - y) \cdot \max(0, m - D)^2$$

where:

- $D = \|z_i - z_j\|_2$ is the Euclidean distance between the two normalized embeddings $z_i, z_j \in \mathbb{R}^{128}$ generated by the shared backbone of the network;
- $y \in \{0, 1\}$ is the binary label associated with the pair: 1 if the two objects belong to the same identity, 0 otherwise;
- $m > 0$ is a hyperparameter that defines the minimum desired margin between embeddings of dissimilar objects.

The behavior of the loss function can be analyzed separately for the two types of pairs:

- For **positive pairs** ($y = 1$), the loss simplifies to D^2, i.e., the squared distance between the two embeddings. This term forces the network to compress the embedding space for representations associated with the same instance or similar objects, minimizing the distance between them.
- For **negative pairs** ($y = 0$), the loss becomes $\max(0, m - D)^2$, a penalization term that activates only if the distance between the two embeddings is less than the margin m. In other words, it penalizes only the "easy" negatives, i.e., those erroneously mapped too close in the metric space, and ignores the contribution of already well-separated pairs.

The parameter m plays a critical role in regulating the behavior of the loss: values that are too small reduce the separation effect between different classes, while overly large margins may cause the function to diverge, especially in the presence of intra-class variability. In this work, the value $m = 1.0$ was empirically selected after a series of preliminary experiments.

Since all embeddings produced by the network are L2-normalized (i.e., $\|z\|_2 = 1$), they lie on the surface of the unit sphere $S^{127} \subset \mathbb{R}^{128}$. Consequently, the Euclidean distance between two normalized embeddings is directly related to the angle between them and can also be expressed in terms of cosine similarity:

$$\|z_i - z_j\|_2^2 = 2(1 - \cos(\theta_{ij}))$$

where θ_{ij} is the angle between the two vectors. This property makes the function particularly robust from a geometric standpoint and well-suited to capturing latent semantic relationships in a continuous manner.

Listing 3.24: Contrastive Loss for Siamese network

```
class ContrastiveLoss(nn.Module):
    def __init__(self, margin=1.0):
        super().__init__()
        self.margin = margin

    def forward(self, emb1, emb2, label):
        dist = F.pairwise_distance(emb1, emb2)
        loss_pos = label * dist.pow(2)
        loss_neg = (1 - label) * F.relu(self.margin - dist).pow(2)
        return (loss_pos + loss_neg).mean()
```

Triplet Loss is a more sophisticated loss function, introduced to overcome the limitations of *Contrastive Loss* in modeling relative relationships between examples. Instead of considering isolated pairs, it operates on groups of three examples, called triplets, composed of:

- An **anchor** x_a: the reference example;
- A **positive** x_p: an object similar to the anchor (same identity);
- A **negative** x_n: a different object (different identity).

The loss is defined as:

$$\mathcal{L} = \max\left(0, \|z_a - z_p\|_2^2 - \|z_a - z_n\|_2^2 + \alpha\right)$$

where:

- z_a, z_p, z_n are the normalized embeddings of anchor, positive, and negative;
- α is the positive margin that enforces a minimum distance between the anchor and the negative compared to the positive.

The intuition behind *Triplet Loss* is that the anchor should always be closer to the positive than to the negative by at least α. When this condition is met, the loss becomes zero (a "satisfied triplet"); otherwise, if the negative is mistakenly closer to the anchor than the positive (or too close), the loss becomes proportional to the degree of violation of the inequality.

This type of formulation allows the embedding space to be modeled according to a relative geometric structure, encouraging the organization of points into compact clusters by identity, separated by clear margins from different classes. The value of the margin α is a delicate

hyperparameter: a value too small fails to enforce sufficient class separation, while one too large may prevent convergence, especially in the presence of noisy data or semantic ambiguity. In this work, $\alpha = 0.2$ was adopted, an empirically validated compromise.

Moreover, *Triplet Loss* is particularly sensitive to the quality of triplet selection. Ideally, one would use so-called **hard triplets**, i.e., triplets in which the negative is very similar to the anchor. However, mining such examples is expensive and may induce instability in convergence. The approach implemented in this work can be considered a form of **semi-hard triplet mining**, in which triplets are constructed through temporal heuristics (co-occurrence analysis in frames) and semantic cues (high IoU, "new" status, category), ensuring a good balance between informativeness and numerical stability.

Listing 3.25: Triplet margin-based loss function

```
class TripletLoss(nn.Module):
    def __init__(self, margin=1.0):
        super().__init__()
        self.margin = margin

    def forward(self, anchor, positive, negative):
        d_pos = F.pairwise_distance(anchor, positive, p=2)
        d_neg = F.pairwise_distance(anchor, negative, p=2)
        loss = F.relu(d_pos - d_neg + self.margin)
        return loss.mean()
```

The combined effect of this loss function is to produce a highly structured embedding space, where each object instance forms a compact region, clearly separated from those of different objects, even when these exhibit high visual similarity.

Both training modes, based respectively on pairs and triplets, share an operational pipeline structured in five main stages, each designed to maximize the efficiency and semantic coherence of the learning process. The pipeline consists of:

- **Dataset loading:** The supervised data is organized in `JSON Lines` format, where each line describes a pair or triplet through image paths, bounding box coordinates, and associated metadata (categories, labels, IoU, etc.). These files are loaded via two custom classes: `SiamesePairsDataset` for pairs, and `TripletDataset` for triplets. Both classes extend `torch.utils.data.Dataset` and implement dynamic cropping logic: starting from the bounding box coordinates, the corresponding image patch is extracted and then transformed to be suitable for the network. These classes also support advanced operations such as automatic label balancing, conversion to PyTorch tensors, data-driven transformations, and multi-thread `DataLoader` integration to accelerate training.
- **Preprocessing and transformations:** Each image undergoes a series of standardized transformations, including:
  - Resizing to a fixed resolution of 224×224, compatible with ResNet;
  - Conversion to tensor and channel permutation (from HWC to CHW);
  - Channel-wise normalization using ImageNet statistics (mean: [0.485, 0.456, 0.406], std: [0.229, 0.224, 0.225]).

  These transformations ensure compatibility with the pre-trained backbone and promote more stable convergence. Images are then processed in mini-batches of 32, maintaining consistency in size and format.
- **Embedding generation:** The normalized crops are fed into the Siamese network, which consists of a `ResNet50` backbone pre-trained on ImageNet, with the final classifier removed. The resulting vector (dimension 2048) is passed to a **projection MLP module**, consisting of:

  - A fully connected layer from 2048 to 512 neurons;
  - `ReLU` activation and `Dropout` with $p = 0.3$;
  - A second fully connected layer from 512 to 128 neurons;
  - Final L2 normalization to project embeddings onto the unit sphere.

  The final output $z \in \mathbb{R}^{128}$ is thus a dense, normalized visual representation of the object, ready to be compared using Euclidean distance or cosine similarity.

- **Model optimization:** The optimization process is handled using the `Adam` algorithm, with an initial learning rate of $1 \cdot 10^{-4}$. The loss function depends on the training mode:
  - `ContrastiveLoss` for pairs, which encourages similar embeddings to move closer and dissimilar ones to move apart beyond the margin m;
  - `TripletLoss` for triplets, enforcing a geometric relationship between anchor, positive, and negative, controlled by margin α.

  Training is performed over 10 full epochs, with a validation phase at the end of each epoch. Validation is based on the average loss over the validation set. The model with the best validation loss is automatically saved via checkpointing. While early stopping was not explicitly used, conditional checkpointing prevented overfitting and ensured robustness against noise or local metric fluctuations.

- **Monitoring:** All key metrics (training and validation loss) were saved in CSV format at the end of each epoch. These values were then visualized using `Matplotlib` to detect anomalies, divergence, or saturation effects. These plots made it possible to empirically verify the stability of both losses and the behavior of different training setups.

At the end of training, the Siamese network was used as a general-purpose encoder to transform each annotated object (image + bounding box) into a vector representation $z \in \mathbb{R}^{128}$. These embeddings form the foundation for all subsequent stages: matching, tracking, and novelty classification.

Listing 3.26: Training of the Siamese network using pairs and contrastive loss

```
model = EmbeddingNet().to(DEVICE)
criterion = ContrastiveLoss(margin=1.0)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

train_losses = []
val_losses = []
best_val_loss = float('inf')

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    for img1, img2, label in train_loader:
        img1, img2, label = img1.to(DEVICE), img2.to(DEVICE), label.to(DEVICE)
        optimizer.zero_grad()
        emb1 = model(img1)
        emb2 = model(img2)
        loss = criterion(emb1, emb2, label)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_train_loss = total_loss / len(train_loader)
    train_losses.append(avg_train_loss)
```

```
    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for img1, img2, label in val_loader:
            img1, img2, label = img1.to(DEVICE), img2.to(DEVICE),
                label.to(DEVICE)
            emb1 = model(img1)
            emb2 = model(img2)
            loss = criterion(emb1, emb2, label)
            total_val_loss += loss.item()

    avg_val_loss = total_val_loss / len(val_loader)
    val_losses.append(avg_val_loss)

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), "best_siamese_pairs_model.pth")
```

Listing 3.27: Training of the Siamese network with best model saving

```
model = EmbeddingNet().to(DEVICE)
criterion = TripletLoss(margin=1.0)
optimizer = torch.optim.Adam(model.parameters(), lr=1e-4)

best_val_loss = float("inf")
train_losses, val_losses = [], []

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    for anc, pos, neg in train_loader:
        anc, pos, neg = anc.to(DEVICE), pos.to(DEVICE), neg.to(DEVICE)
        optimizer.zero_grad()
        loss = criterion(model(anc), model(pos), model(neg))
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_train_loss = total_loss / len(train_loader)
    train_losses.append(avg_train_loss)

    model.eval()
    val_loss = 0
    with torch.no_grad():
        for anc, pos, neg in val_loader:
            anc, pos, neg = anc.to(DEVICE), pos.to(DEVICE), neg.to(DEVICE)
            loss = criterion(model(anc), model(pos), model(neg))
            val_loss += loss.item()
    avg_val_loss = val_loss / len(val_loader)
    val_losses.append(avg_val_loss)

    if avg_val_loss < best_val_loss:
        best_val_loss = avg_val_loss
        torch.save(model.state_dict(), "best_siamese_model.pth")
```

Once the Siamese backbone has been trained and a stable, discriminative semantic embedding

space has been obtained, a dedicated module was implemented for binary recurrence classification. This module aims to predict whether an object observed in the current frame is a new instance (never seen before) or a recurrence of an object previously detected in past frames.

For each object o_t in a current frame t, represented by its bounding box b_t and corresponding embedding $z_t \in \mathbb{R}^{128}$, the best matching is searched among all embeddings stored in previous frames, within a fixed temporal window $[t-30, t-1]$. This strategy implements a limited temporal memory, useful for reducing computational complexity and maintaining temporal locality in the tracking process.

Among all possible matches (z_k, b_k) from previous frames, the one with the highest cosine similarity to z_t is selected:

$$\text{cosine\_sim}(z_t, z_k) = \frac{z_t \cdot z_k}{\|z_t\| \cdot \|z_k\|}, \quad \text{with} \quad z_t, z_k \in S^{127}$$

Since all embeddings are L2-normalized, cosine similarity is equivalent to the dot product, making the computation highly efficient.

Once the best match is identified, four descriptive features are computed and combined into a feature vector $(f_1, f_2, f_3, f_4) \in \mathbb{R}^4$:

1. f_1: **Cosine similarity** between z_t and z_k, capturing semantic coherence;
2. f_2: **Intersection over Union (IoU)** between b_t and b_k, used to assess spatial consistency;
3. f_3: **Euclidean distance between the centroids** of the two bounding boxes, indicative of geometric proximity in the frame;
4. f_4: **Area ratio** between the two boxes, penalizing matches with significantly different scales.

These features are designed to jointly capture three types of coherence: semantic (via embeddings), spatial (via IoU), and geometric (via centroid and area). The feature vector is then normalized using a `StandardScaler` fitted on the training data, in order to center and rescale feature distributions and stabilize the classifier's training.

Listing 3.28: Extraction of similarity features between bounding boxes for MLP training

```
for img_id in sorted(img_map.keys()):
    img_data = img_map[img_id]
    img_path = img_data["path"]
    annotations = img_data["annotations"]
    current_objects = []

    for obj in annotations:
        bbox = obj["bbox"]
        cat_id = obj["category_id"]
        true_new = 1 if obj["attributes"]["new"] == "yes" else 0
        emb = crop_and_embed(img_path, bbox, model, transform, DEVICE)
        # emb = crop_and_embed(img_path, bbox, model_pairs, transform,
            DEVICE) # se si usa la rete addestrata con coppie
        if emb is None:
            continue

        best = {"cosine_sim": -1.0, "iou": 0.0, "centroid_dist": 1e9,
            "area_ratio": 1.0}
        for past in memory:
            for past_obj in past["objects"]:
                if past_obj["category_id"] != cat_id:
```

```
                    continue
                cos_sim = F.cosine_similarity(emb, past_obj["embedding"],
                    dim=0).item()
                iou = bbox_iou(bbox, past_obj["bbox"])
                dist = np.linalg.norm(np.array(bbox_centroid(bbox)) -
                    np.array(bbox_centroid(past_obj["bbox"])))
                ar = bbox_area(bbox) / (bbox_area(past_obj["bbox"]) + 1e-6)
                if cos_sim > best["cosine_sim"]:
                    best.update({"cosine_sim": cos_sim, "iou": iou,
                        "centroid_dist": dist, "area_ratio": ar})

        features.append({
            "cosine_sim": best["cosine_sim"],
            "iou": best["iou"],
            "centroid_dist": best["centroid_dist"],
            "area_ratio": best["area_ratio"],
            "true_new": true_new
        })

        if true_new == 1:
            current_objects.append({
                "embedding": emb.detach(),
                "category_id": cat_id,
                "bbox": bbox
            })

    memory.append({"image_id": img_id, "objects": current_objects})
df_feat = pd.DataFrame(features)
df_feat.to_csv("mlp_features_dataset.csv", index=False)
```

Listing 3.29: PyTorch dataset for the MLP with feature normalization

```
class FeatureDataset(Dataset):
    def __init__(self, csv_path, scaler=None):
        df = pd.read_csv(csv_path)
        X = df[["cosine_sim", "iou", "centroid_dist", "area_ratio"]].values
        y = df["true_new"].values.astype(np.float32)

        if scaler is None:
            scaler = StandardScaler()
            X = scaler.fit_transform(X)
            self.scaler = scaler
        else:
            X = scaler.transform(X)
            self.scaler = scaler

        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.float32)

    def __len__(self):
        return len(self.y)

    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]
```

Listing 3.30: MLP for binary classification: new vs known object

```
class MLPClassifier(nn.Module):
```

```
    def __init__(self, input_dim):
        super().__init__()
        self.fc1 = nn.Linear(input_dim, 64)
        self.fc2 = nn.Linear(64, 32)
        self.out = nn.Linear(32, 1)

    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        return torch.sigmoid(self.out(x)).squeeze()
```

The binary classification model implemented to predict whether an object is "new" is a **Multilayer Perceptron (MLP)** with a compact yet effective architecture. The network takes as input the normalized feature vector (f_1, f_2, f_3, f_4) and outputs a probability $p \in [0,1]$ representing the likelihood that the object belongs to the "new" class.

The MLP architecture is defined as follows:

- Input layer: 4 neurons (one for each feature);
- First hidden layer: 64 neurons, `ReLU` activation;
- Second hidden layer: 32 neurons, `ReLU` activation;
- Output layer: 1 neuron, `sigmoid` activation.

The network is trained by minimizing the `Binary Cross Entropy Loss (BCELoss)`, which penalizes the discrepancy between the predicted probability p and the binary ground-truth label $y \in \{0,1\}$. Optimization is performed using the `Adam` optimizer with a learning rate of $1 \cdot 10^{-4}$ and a batch size of 64. Training is conducted over 20 epochs. The model achieving the highest accuracy on the stratified validation set is saved as the best checkpoint.

Listing 3.31: Training of the MLP with Binary Cross-Entropy Loss

```
model = MLPClassifier(input_dim=4).to(DEVICE)
criterion = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

train_losses, val_losses = [], []

for epoch in range(EPOCHS):
    model.train()
    total_loss = 0
    for x_batch, y_batch in train_loader:
        x_batch, y_batch = x_batch.to(DEVICE), y_batch.to(DEVICE)
        optimizer.zero_grad()
        y_pred = model(x_batch)
        loss = criterion(y_pred, y_batch)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()

    avg_train_loss = total_loss / len(train_loader)
    train_losses.append(avg_train_loss)

    model.eval()
    total_val_loss = 0
    with torch.no_grad():
        for x_val, y_val in val_loader:
```

```
            x_val, y_val = x_val.to(DEVICE), y_val.to(DEVICE)
            y_pred = model(x_val)
            val_loss = criterion(y_pred, y_val)
            total_val_loss += val_loss.item()

    avg_val_loss = total_val_loss / len(val_loader)
    val_losses.append(avg_val_loss)

torch.save(model.state_dict(), "mlp_classifier.pth")
# torch.save(model.state_dict(), "mlp_classifier_pairs.pth")
```

Listing 3.32: Evaluation of the MLP and saving of the ROC curve

```
val_loader = DataLoader(val_ds, batch_size=128)
model.eval()
y_true, y_pred, y_scores = [], [], []

with torch.no_grad():
    for x_batch, y_batch in val_loader:
        x_batch = x_batch.to(DEVICE)
        y_score = model(x_batch).cpu()
        y_pred_batch = (y_score >= 0.5).float()
        y_scores.extend(y_score.numpy())
        y_pred.extend(y_pred_batch.numpy())
        y_true.extend(y_batch.numpy())

acc = accuracy_score(y_true, y_pred)
prec = precision_score(y_true, y_pred)
rec = recall_score(y_true, y_pred)
f1 = f1_score(y_true, y_pred)
auc = roc_auc_score(y_true, y_scores)
```

The dataset used for training was obtained through a semi-supervised procedure based on the embeddings produced by the Siamese network and the original "new" labels provided in the COCO dataset. Specifically, for each annotated object, a positive or negative sample was generated depending on the presence of real matches in previous frames. To ensure proper class balance, a 1:1 ratio between "new" and "previously seen" examples was maintained using undersampling (in cases of negative prevalence) or oversampling (in the opposite case).

The architectural simplicity of the MLP is justified by the small number of input features and the inherently linear nature of the problem: the combination of semantic, spatial, and geometric coherence proved sufficient to effectively separate the two classes. Moreover, the modularity of the classifier allows for replacement with more complex models (e.g., Random Forest, XGBoost) if required in noisier or less structured scenarios.

The final classifier is subsequently used during the inference and tracking phase to determine, frame by frame, whether each new observation represents a new instance or a recurrence. A threshold value of $p \geq 0.5$ is used to binarize the prediction. The results are aggregated and saved for further evaluation.

Once the Siamese network and the MLP classifier have been trained, the entire system is integrated into a sequential inference pipeline, designed for temporal object tracking and binary classification of their state ("new" vs "recurring"). The approach is based on comparing each new observation with the recent visual memory, represented by a dynamic set of embeddings and associated metadata.

For each new frame F_t processed, the system performs the following operations in sequence:

1. **Embedding extraction:** For each annotated object in frame F_t, the region of interest (bounding box) is cropped from the image and undergoes preprocessing (resize to 224×224,

normalization). The resulting patch is then passed through the Siamese network to obtain the normalized embedding $z_t \in \mathbb{R}^{128}$.

2. **Memory search:** The embedding z_t is compared against all embeddings z_k stored in the temporary memory M, which contains information on objects observed in the 30 previous frames $F_{t-1}, F_{t-2}, ..., F_{t-30}$. Each memory entry is represented as a tuple (z_k, b_k, ID_k, f_k), where b_k is the bounding box, ID_k is the object identifier, and f_k is the frame index of origin.

3. **Feature computation:** For each comparison (z_t, z_k), a feature vector (f_1, f_2, f_3, f_4) is constructed, as previously described:

   - f_1: cosine similarity between z_t and z_k;
   - f_2: IoU between boxes b_t and b_k;
   - f_3: Euclidean distance between the centroids of the two boxes;
   - f_4: area ratio between the two boxes.

4. **Best match selection:** Among all pairs (z_t, z_k), the one with the highest cosine similarity f_1 is selected. In case of ties, the Euclidean distance between centroids is used as a secondary criterion.

5. **Classification:** The feature vector of the selected pair is normalized using the `StandardScaler` fitted on the training data and passed as input to the MLP classifier. The model outputs a probability $p \in [0,1]$ indicating the likelihood that the object is "new" (i.e., not present in memory).

6. **Decision:** If $p \geq 0.5$, the object is labeled as `predicted_new = 1`; otherwise, it is considered a recurrence and labeled as `predicted_new = 0`.

7. **Memory update:** The temporary memory is updated by inserting a new entry (z_t, b_t, ID, t) corresponding to the current object. If the memory has reached its maximum capacity (e.g., 30 frames), the oldest entries are discarded.

Throughout the entire inference phase, metadata relevant for both quantitative and qualitative analysis of the system are stored. This information is archived in a final CSV file, where each row corresponds to a single prediction and includes:

- `frame`: index of the current frame;
- `object_id`: unique identifier assigned to the object;
- `predicted_new`: binary output of the classifier (0 = recurring, 1 = new);
- `ground_truth_new`: ground truth label extracted from the COCO dataset;
- `max_similarity`: cosine similarity value of the best match;
- `IoU`: Intersection over Union value with the best match;
- `embedding_distance`: Euclidean distance between z_t and z_k;
- `MLP_output`: probability p produced by the classifier.

This closed-loop architecture enables continuous monitoring of the semantic evolution of visual entities across the temporal sequence, and dynamic management of object identity. The adoption of metric learning as a core mechanism ensures robust comparison between representations, regardless of variations in pose, lighting, or partial occlusions, while the supervised MLP component provides a discrete decision on novelty.

Listing 3.33: Temporal tracking with sliding window memory and prediction with MLP

```
for img_id in tqdm(sorted_ids, desc="Tracking␣con␣MLP"):
    img_data = img_map[img_id]
    img_path = img_data["path"]
    annotations = img_data["annotations"]
    current_objects = []

    for obj in annotations:
        bbox = obj["bbox"]
        cat_id = obj["category_id"]
        true_new = 1 if obj["attributes"]["new"] == "yes" else 0

        emb = crop_and_embed(img_path, bbox, s_model, transform, DEVICE)
        # emb = crop_and_embed(img_path, bbox, s_model_pairs, transform,
            DEVICE)
        if emb is None: continue

        best = {"cosine_sim": -1.0, "iou": 0.0, "centroid_dist": 1e9,
            "area_ratio": 1.0}
        for past in memory:
            for p in past["objects"]:
                if p["category_id"] != cat_id: continue
                cos_sim = F.cosine_similarity(emb, p["embedding"],
                    dim=0).item()
                iou = bbox_iou(bbox, p["bbox"])
                dist = np.linalg.norm(np.array(bbox_centroid(bbox)) -
                    np.array(bbox_centroid(p["bbox"])))
                ar = bbox_area(bbox) / (bbox_area(p["bbox"]) + 1e-6)
                if cos_sim > best["cosine_sim"]:
                    best.update({"cosine_sim": cos_sim, "iou": iou,
                        "centroid_dist": dist, "area_ratio": ar})

        x_feat = pd.DataFrame([best])
        x_scaled = torch.tensor(scaler.transform(x_feat),
            dtype=torch.float32).to(DEVICE)
        pred = mlp_model(x_scaled).item()
        # pred = mlp_model_pairs(x_scaled).item()
        is_new = int(pred >= 0.5)

        results.append({
            "image_id": img_id,
            "object_id": obj["id"],
            "category_id": cat_id,
            "true_new": true_new,
            "predicted_new": is_new,
            "cosine_sim": best["cosine_sim"],
            "iou": best["iou"],
            "centroid_dist": best["centroid_dist"],
            "area_ratio": best["area_ratio"],
            "mlp_output": pred,
            "bbox_x": bbox[0],
            "bbox_y": bbox[1],
            "bbox_w": bbox[2],
            "bbox_h": bbox[3]
        })
```

```
        if is_new == 1:
            current_objects.append({
                "embedding": emb.detach(),
                "category_id": cat_id,
                "bbox": bbox
            })

    memory.append({"image_id": img_id, "objects": current_objects})
```

The entire system is characterized by modularity, scalability, and adaptability. The decoupling between the embedding extraction phase and the classification phase allows for future improvements to each module independently — for instance, by adopting more advanced strategies for memory management (e.g., hierarchical or attention-based memory), or by replacing the MLP classifier with more complex or specialized models.

The result is an incremental recognition-oriented tracking system, effective in open-world scenarios where the number of object identities is unknown a priori, and capable of adaptively managing the appearance of new visual entities over time.

3.8 Hyperparameters and experimental setup

To complete the description of the three architectures developed — Baseline, Memory-Augmented Network, and Siamese Network — it is useful to provide a unified overview of the hyperparameters adopted, the computational environment, and the data preparation procedures, in anticipation of the experimental results presented in Chapter 4.

All experiments were conducted on a Microsoft Azure virtual machine equipped with an NVIDIA V100 GPU with 16 GB of memory, 56 virtual CPU cores, and 224 GB of RAM. The development environment used was JupyterLab, with Python 3.10 as the kernel, and main libraries including PyTorch 2.0, torchvision, NumPy, OpenCV, Matplotlib, Scikit-learn, and tqdm.

The dataset used was a custom-annotated version of COCO 2017, comprising 6,958 valid images. Each object is annotated with a bounding box, semantic category, and a binary `new` label indicating whether it represents a new instance or a recurrence. The dataset was split as follows: 80% for training, 20% for validation, and a separate subset of 1,000 images reserved for preliminary experiments and debugging.

The images and object crops were preprocessed by resizing to 224×224 pixels, normalization according to ImageNet statistics, and conversion into PyTorch-compatible tensors.

Below is a summary of the main configurations adopted for each architecture:

Baseline

- Backbone: ResNet50 (ResNet18 used in some variants)
- Embedding: 2048 dimensions
- Classifier: MLP (32-16-1) with ReLU and Sigmoid
- FIFO memory: 20 frames
- Decision strategy: fixed threshold or supervised MLP

Memory-Augmented Network

- Backbone: ResNet50 (softmask variant or ResNet18)
- Visual embedding: 512 dimensions
- Memory Transformer: static or learnable encoder (4 heads)
- Final classifier: MLP (64-32-1) with ReLU and Sigmoid
- FIFO memory: 30 frames

Siamese Networks

- Backbone: ResNet50 without final fully connected layer
- Final embedding: 128 dimensions (MLP 2048→512→128)
- Loss functions: Binary Cross Entropy (pairs), Triplet Loss (triplets)
- Margins: $m = 1.0$ for Contrastive Loss, $\alpha = 0.2$ for Triplet Loss
- Optimizer: Adam with learning rate 10^{-4}
- Batch size: 32 for the Siamese network, 64 for the final binary MLP
- Epochs: 10 for the backbone, 20 for the final classifier
- FIFO memory: 30 frames

All models were validated at the end of each epoch by computing the average loss on the validation set. The best-performing model was saved via checkpointing. For the binary classifier, a 1:1 balance between the `new` and `not new` classes was maintained using undersampling or oversampling techniques.

The main evaluation metrics include **accuracy**, **precision**, **recall**, **F1-score**, and **AUC**, computed both globally and on a per-frame basis. The dataset split was kept consistent across all architectures to ensure fairness in comparison.

Chapter 4

Results and Discussion

The evaluation of the effectiveness of the proposed architectures for temporal object tracking is based on a structured experimental protocol, which integrates multiple classification metrics, a coherent validation logic, and an inference mechanism based on visual temporal memory.

During the inference phase, each visible object in a new frame is compared with those stored in the FIFO memory, whose size varies depending on the architecture: 20 frames for the baseline and 30 for the Memory-Augmented Network and the Siamese network. This approach allows simulating the availability of a visual history to detect recurrences.

The comparison between the current object and those in memory generates a set of descriptive features — such as cosine similarity between embeddings, Intersection over Union (IoU), centroid distance, and area ratio — which are used to decide whether an object is new or already seen. The decision can be made through a fixed threshold or by means of a supervised MLP classifier, depending on the variant considered.

The evaluation metrics adopted to quantify the performance of the architectures in the binary classification task are as follows:

- **Accuracy**: percentage of correct predictions over the total.
- **Precision**: proportion of objects classified as "already seen" that actually are.
- **Recall**: proportion of "already seen" objects correctly recognized.
- **F1-score**: harmonic mean between precision and recall, indicative of the balance between the two metrics.
- **AUC (Area Under the ROC Curve)**: overall measure of the model's discriminative capability.

Metrics are computed on the validation set and, in some experiments, across multiple runs with different random seeds to assess their stability. The reported results represent the average of the observed performances.

All experiments were carried out using PyTorch, keeping the main parameters shared across architectures constant:

- Backbone: ResNet18 or ResNet50 pretrained on ImageNet
- Input: crops from bounding boxes resized to 224×224 pixels
- Optimizer: Adam or SGD, with a learning rate between 10^{-3} and 10^{-4}
- Batch size: ranging from 16 to 64

- Loss function: Binary Cross Entropy for supervised networks; Contrastive and Triplet Loss for metric learning

Finally, the adopted classification strategies include two configurations:

- Fixed threshold applied to cosine similarity or to the classifier output
- Supervised MLP combining visual and geometric features

The following sections present and discuss the results obtained by each architecture, comparing the various experimental variants from both a quantitative and qualitative perspective.

4.1 Experimental results

4.1.1 Baseline

As a starting point for comparing the different proposed architectures, a reference solution (baseline) was implemented, built around a simple yet effective principle, easily interpretable and low in computational cost. The approach, described in detail in Section 3.5, is based on two main components: a visual feature extractor, represented by a pretrained ResNet network, and a decision module — either supervised or heuristic — responsible for determining whether the current object has been previously observed or represents a new occurrence.

During inference, each object identified within an image is cropped based on the corresponding bounding box, normalized and resized, then passed through the ResNet network. The output of the backbone is a visual embedding that captures the semantic information of the object in a vector space. This embedding is compared with those stored from the last 20 images, thanks to a FIFO structure shared by all baseline variants. For each current object–memory object pair, four comparison features are computed:

- **Cosine similarity** between the current embedding and those in memory
- **IoU** between the respective bounding boxes
- **Euclidean distance between centroids**
- **Ratio between the areas of the two bounding boxes**

These features are then aggregated into a vector and used for the decision phase. In some configurations, this vector is passed to a supervised MLP classifier, which estimates the probability that the object has been "already seen"; in others, the decision is based on fixed thresholds applied to the cosine similarity or to a combined score with IoU. During experimentation, various configurations were tested, with the aim of evaluating the impact of architectural and decision-making choices on system behavior. In particular, the following scenarios were analyzed:

- **ResNet18 + cosine similarity**, with empirical similarity threshold between 0.50 and 0.75
- **ResNet50 + cosine similarity**, with the same reference thresholds
- **ResNet18 + cosine similarity + IoU**, with a combined score according to the formula: $score = 0.6 \cdot \text{cosine\_similarity} + 0.4 \cdot \text{IoU}$
- **ResNet50 + cosine similarity + IoU**, same combined scheme
- **ResNet18 + supervised MLP**, trained on the 4 features described above

- **ResNet50 + supervised MLP**, with the same classifier architecture

The design of the MLP classifier required an initial exploratory phase. Several designs were tested, inspired by visual classification and matching models. In particular, the first versions used the direct concatenation of visual embeddings, but showed poor stability and discriminative capability. Subsequently, attention shifted to the use of geometric and similarity features between pairs, appropriately normalized. After several iterations, the configuration that offered the best trade-off between simplicity and performance turned out to be a three-layer MLP, with the following structure:

- `Linear(4 → 32)` + ReLU
- `Linear(32 → 16)` + ReLU
- `Linear(16 → 1)` + Sigmoid

This network, trained with binary loss (Binary Cross-Entropy), proved particularly effective in synergistically combining the various features, managing to accurately predict the probability that an object had already been seen before.

In parallel, for the fixed-threshold variants, a tuning activity was carried out over a range of values between 0.50 and 0.75. Using lower thresholds, around 0.50–0.58, favored recall but introduced numerous false positives. Conversely, higher thresholds (above 0.70) led to an increase in false negatives and a worsening of recall. The optimal values, selected by maximizing the F1-score on the validation set, were found to be 0.64 for the `ResNet18 + cosine` configuration, 0.60 for `ResNet50 + cosine`, and 0.66 for the variants combined with IoU.

The results obtained for each configuration are reported in Table 4.1. The performances were calculated over the entire validation set and, where necessary, averaged over multiple runs to increase significance.

Table 4.1: Results of the baseline configurations: comparison between heuristic and supervised methods with different backbones.

Configuration	Backbone	Method	Accuracy	Precision	Recall	F1	AUC
Baseline (threshold 0.64)	ResNet18	Similarity	0.688	0.643	0.920	0.757	0.76
Baseline (threshold 0.60)	ResNet50	Similarity	0.696	0.665	0.895	0.765	0.77
Baseline (combined score)	ResNet18	Cosine + IoU	0.695	0.660	0.878	0.752	0.76
Baseline (combined score)	ResNet50	Cosine + IoU	0.708	0.674	0.884	0.766	0.78
Baseline (with MLP)	ResNet18	Supervised MLP	0.711	0.682	0.869	0.764	0.78
Baseline (with MLP)	**ResNet50**	**Supervised MLP**	**0.703**	**0.678**	**0.872**	**0.761**	**0.78**

From the comparison between the variants, it emerges that the fixed-threshold approach, although simple and efficient, is particularly sensitive to the choice of threshold value, with evident effects on the balance between precision and recall. The use of ResNet50 compared to ResNet18 results in a marginal performance increase, at the cost of higher computational demand. The integration of IoU in the combined score proved useful for improving geometric coherence, but did not yield benefits comparable to those obtained with a supervised classifier. The MLP variant, in fact, showed greater stability and generalization capacity, with higher values in terms of F1-score and AUC.

In particular, the MLP classifier proved capable of exploiting the various available features to compensate for visual ambiguities and similar geometries, providing more robust predictions even in complex scenarios. This observation suggests that, even in the absence of a sophisticated memory mechanism, the introduction of a learning component can bring tangible benefits in terms of final decision-making.

A further advantage of the baseline lies in its implementation simplicity, which makes it a good candidate for resource-constrained contexts, such as edge or embedded environments.

However, the limitations that emerged in handling temporal and contextual variability highlight the need to move toward more advanced solutions. Such solutions, as will be described in the following sections, are designed to explicitly integrate the historical context of observations and further enhance the system's generalization capabilities.

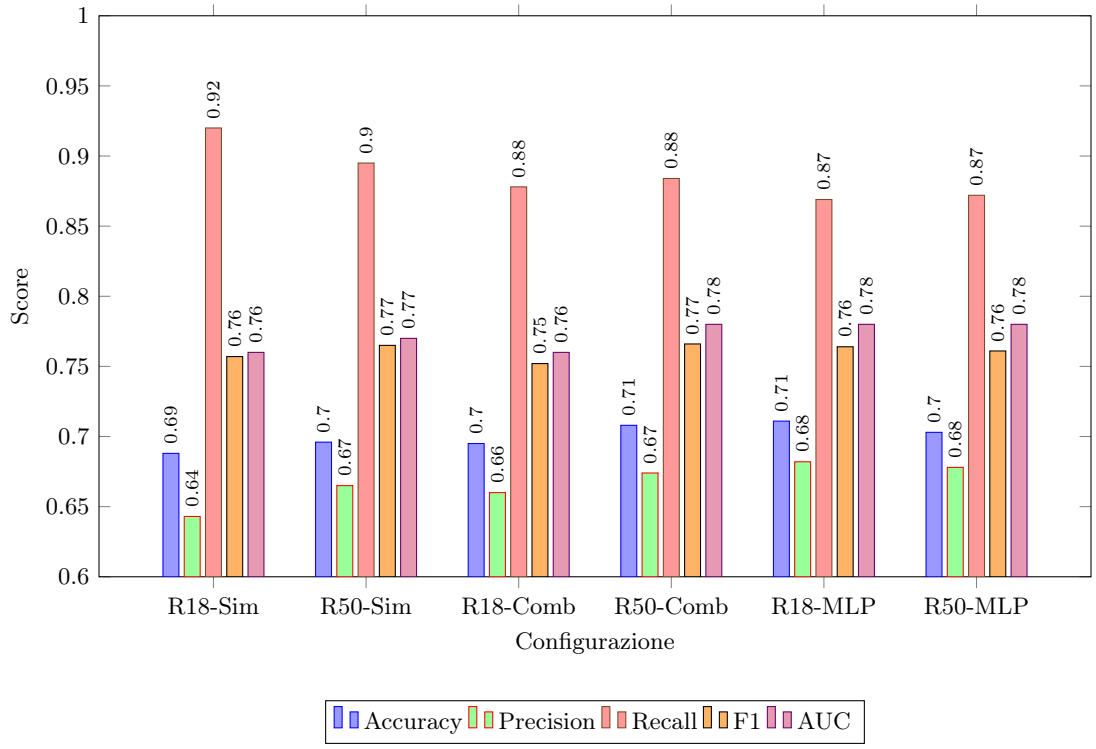


Figure 4.1: Comparison between the baseline configurations in terms of Accuracy, Precision, Recall, F1-score, and AUC.

Overall, the baseline proved useful not only as an initial reference point but also as a tool for analyzing the contribution of each system component. The simpler versions helped highlight the structural limitations of heuristic approaches, while the introduction of a supervised MLP network demonstrated that, even with a relatively simple design, it is possible to achieve significantly better performance.

This analysis thus laid the groundwork for the introduction of more sophisticated architectures, capable of explicitly and adaptively integrating temporal context, as described in the following sections.

4.1.2 Memory-Augmented Network (MAN)

The second architecture analyzed is a Memory-Augmented Network (MAN), designed to overcome the structural limitations of the baseline by introducing an explicit mechanism for contextual memory. As described in Section 3.6, the goal of this model is to exploit the history of past observations in a more structured way, through a Transformer-based attention module that allows the current object to be contextualized with respect to a visual memory.

The architecture consists of four main modules: (i) extraction of crops from bounding boxes, (ii) a CNN backbone for feature extraction, (iii) a Transformer encoder module serving as a learnable memory, and (iv) a final classifier, based on either a fixed threshold or a supervised MLP. For each object in an image, an RGB crop centered on the bounding box is extracted, with a slight padding to include visual context. The image is then normalized and resized to 224×224 pixels before being processed by the backbone.

Once again, various architectural variants were explored to understand the specific contribution of each component. The tested combinations include all possible choices of backbone, visual input, type of memory, and decision strategy. In particular, the following were evaluated:

- **Type of visual input**:
  - Bbox (Bounding-box)
  - Soft-mask (grayscale binary masks)
- **Backbone used for feature extraction**:
  - ResNet18
  - ResNet18 with fine-tuning on the target dataset
  - ResNet50
- **Type of Transformer memory**:
  - Static (non-learnable)
  - Learnable (trained jointly with the rest of the system)
- **Final decision**:
  - Fixed threshold on cosine similarity (empirical)
  - Supervised MLP, fed with visual and geometric features

As with the baseline, in this case too a FIFO memory is used to store the history of objects observed in the last 30 images. Each current object is compared with those in memory, which act as keys and values in the Transformer module. The Transformer output represents a new contextualized embedding, which can then be compared with the memory in terms of cosine similarity, or used as input to an MLP classifier.

At decision time, in addition to the maximum cosine similarity with memory objects, the same geometric features used in the baseline are also computed: maximum IoU, centroid distance, and area ratio. These four features are concatenated and passed to the supervised MLP for binary classification.

For each variant described, an independent experiment was conducted, keeping all other training and validation parameters constant. The results obtained are reported in Table 4.2.

Table 4.2: Results of the Memory-Augmented Network (MAN) variants with static or learnable memory and different decision strategies.

Configuration	Accuracy	Precision	Recall	F1	AUC
Softmask + ResNet18 + static memory + fixed threshold	0.684	0.645	0.905	0.756	0.76
Softmask + ResNet50 + static memory + fixed threshold	0.681	0.640	0.907	0.751	0.76
BBox + ResNet18 + static memory + fixed threshold	0.703	0.665	0.895	0.762	0.78
BBox + ResNet18 FT + static memory + fixed threshold	0.715	0.672	0.889	0.766	0.79
BBox + ResNet50 + static memory + fixed threshold	0.719	0.681	0.881	0.768	0.79
BBox + ResNet50 + learnable MAN + fixed threshold	0.727	0.690	0.875	0.768	0.80
BBox + ResNet50 + learnable MAN + MLP	**0.756**	**0.725**	**0.843**	**0.781**	**0.82**

The analysis of the results highlights significant differences among the tested variants, each of which allowed for a deeper understanding of the role of individual architectural components. One of the most relevant aspects emerged in the choice of visual input type. Initially, the system was designed using RGB crops obtained from the bounding boxes of the objects. Subsequently, two alternative experiments were conducted: the direct use of binary object masks, and, in a later

phase, the introduction of a soft-mask representation, in which the object was isolated within a window on a uniform gray background, without visual context.

Contrary to initial expectations, the mask-based variants — both binary and soft — yielded inferior results compared to full RGB crops. This behavior can be attributed to several factors. Although masks represent the exact shape of the object, they completely remove the surrounding spatial context. In a realistic scenario such as the one analyzed, where objects may slightly change shape, position, or be partially occluded, the visual context plays a fundamental role. Information such as the presence of adjacent objects, their spatial arrangement, and even the background color contribute to making the object recognizable over time. Moreover, masks overemphasize shape and size, which in this domain are features highly subject to variation (e.g., deformed, rotated, or truncated objects), effectively introducing noise rather than stability in the representation. The soft-masks, while partially preserving the bounding box area, suffered from the same limitation: the removal of context in favor of a uniform background reduced the global information useful for comparing the current object with previous observations.

The choice of backbone also had a significant impact on performance. ResNet50 proved to be more effective than ResNet18 in all scenarios, offering more robust and consistent representations. The greater depth and abstraction capability of ResNet50 allows it to capture complex visual patterns, making it more suitable for distinguishing similar objects with subtle differences. Although ResNet18, even with fine-tuning on the dataset, provides faster inference, it is less suited in the presence of visual variability or noisy conditions.

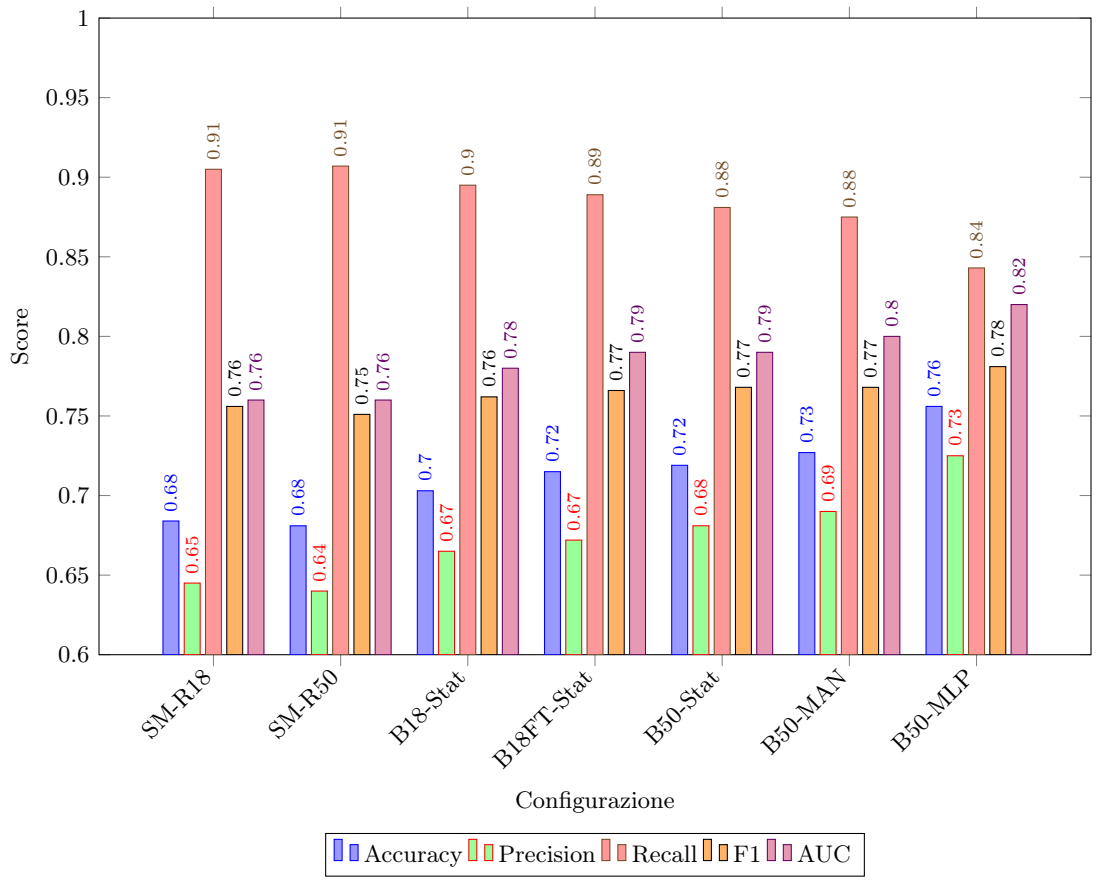


Figure 4.2: Comparison between MAN variants in terms of Accuracy, Precision, Recall, F1-score, and AUC.

Another critical element was the comparison between the static and the learnable Transformer module. In the static configuration, the Transformer acts as a simple projector of embeddings into

a latent space, but it lacks the ability to learn how to weigh the stored observations. This makes it less effective in distinguishing between relevant elements and noise. Conversely, the learnable version — trained end-to-end with the rest of the system — is capable of dynamically modeling the relationships between the current object and the memory. This enables a contextualized and adaptive representation, able to strengthen the importance of strong semantic matches and reduce the influence of less informative or misleading memories. This led to a clear increase in precision and consistency across consecutive frames.

Finally, the classification strategy also had a decisive impact. The fixed-threshold variants, similarly to what was observed in the baseline, are highly sensitive to the distribution of similarities and incapable of effectively integrating multiple features. The introduction of a supervised MLP, fed with contextualized cosine similarity together with geometric features (IoU, centroid distance, and area ratio), enabled more stable and accurate classification. In particular, the MLP showed better capabilities in handling ambiguous cases, such as partially overlapping objects or those with similar characteristics but different spatial relationships.

Overall, the combination of the four most effective architectural choices — RGB input, ResNet50, learnable Transformer, and supervised MLP — led to the configuration `BBox + ResNet50 + MAN Learnable + MLP`, which achieved the best performance across all main metrics. This version not only outperformed the baseline but also showed more stable behavior, reduced sensitivity to false positives, and greater adaptability to dynamic scenarios. Although more complex to train and slightly more computationally demanding, this architecture represents a substantial step forward in temporal memory management for visual object tracking.

4.1.3 Siamese Network

The third family of architectures analyzed in this work is based on Siamese networks, a paradigm widely used in visual instance similarity tasks. As detailed in Section 3.7, the approach adopted in this context does not merely aim to verify generic semantic affinity between two objects, but rather to answer a more subtle and concrete question: is this the same physical instance observed at different points in time? This type of challenge, which we might define as temporal identity recognition, requires sophisticated modeling of visual representations, capable of capturing continuity even in the presence of variations in scale, viewpoint, lighting, and partial occlusions.

The architecture is divided into two main variants: a binary Siamese network, trained using contrastive loss, and a triplet network, based on the optimization of the triplet loss. In both cases, the backbone used is a ResNet (either 18 or 50 version), shared between the network branches, responsible for extracting a normalized visual embedding from an RGB crop centered on the object to be analyzed. Training was made possible through the automatic generation of object pairs and triplets, using heuristics that leverage semantic coherence, spatial proximity, and temporal persistence, as described in Chapter 3.

Inference is based on comparing the current object with a set of recent observations stored in memory. Depending on the configuration, the decision can be made using a fixed threshold on cosine similarity or through a supervised MLP classifier, which takes as input a combination of visual and geometric features. In this case as well, the four features used are: maximum cosine similarity, IoU, distance between centroids, and area ratio — the same used in the baseline and MAN architectures, to ensure consistency in comparison. During the experiments, six distinct configurations were evaluated, obtained by combining: type of network (pairs or triplets), backbone (ResNet18 or ResNet50), and decision strategy (fixed threshold or supervised MLP). The results, reported in Table 4.3, clearly show how the architectural choice and the decision-making component significantly impact performance.

The comparison between the different configurations clearly highlights the superiority of the triplet-based approach, especially when paired with a supervised decision strategy. The best combination — `Triplets - ResNet50 + supervised MLP` — not only achieved the highest values across all key metrics (F1-score, AUC, Recall), but also showed greater stability across runs, confirming the robustness of the approach. This result underscores the strength of the

Table 4.3: Results of pairwise and triplet-based configurations: comparison between decision strategies and backbones.

Configuration	Accuracy	Precision	Recall	F1	AUC
Pairs – ResNet18 + fixed threshold	0.678	0.650	0.685	0.667	0.72
Pairs – ResNet50 + fixed threshold	0.694	0.667	0.680	0.673	0.74
Triplets – ResNet18 + fixed threshold	0.715	0.680	0.802	0.736	0.78
Triplets – ResNet50 + fixed threshold	0.738	0.700	0.845	0.765	0.81
Pairs – ResNet50 + supervised MLP	0.710	0.685	0.740	0.711	0.77
Triplets – ResNet50 + supervised MLP	**0.787**	**0.747**	**0.906**	**0.818**	**0.85**

relative learning paradigm introduced by the triplet loss, which does not merely evaluate absolute similarity between two elements but forces the network to structure a coherent embedding space, where proximity and distance relationships reflect identity continuity across objects.

In contrast, networks trained with contrastive loss exhibited limitations in terms of precision, due to difficulties in learning an effective discriminative threshold in the presence of visually similar but distinct objects. Specifically, the binary formulation of the supervised task proved more fragile, showing a higher rate of false positives in ambiguous cases, where visual appearance alone was not sufficient to distinguish between different instances. Nevertheless, these configurations proved valid as internal baselines within the Siamese family, offering a simple and interpretable structure.

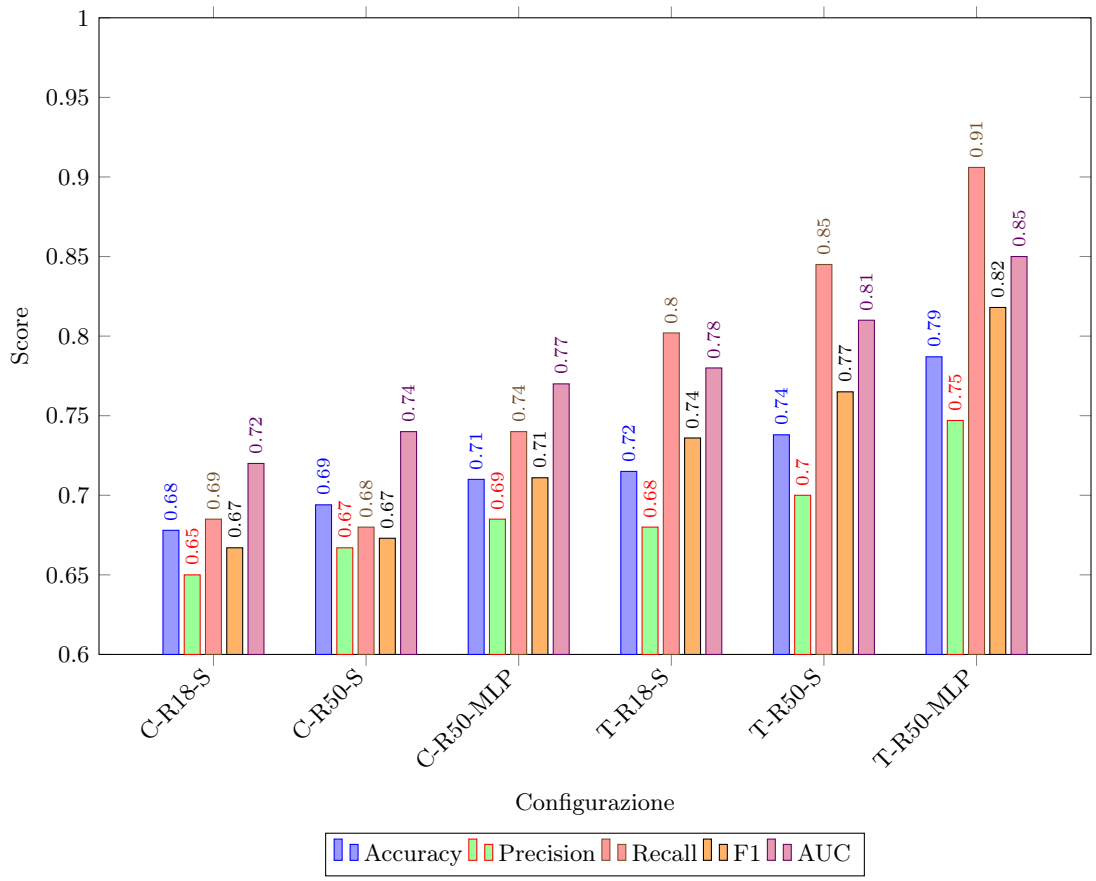


Figure 4.3: Comparison between Siamese configurations (Pairs and Triplets) in terms of Accuracy, Precision, Recall, F1-score, and AUC.

Another important finding from the experiments is the impact of the backbone. Using ResNet50 instead of ResNet18 consistently improved all metrics, suggesting that higher expressive capacity in the visual module allows for the generation of more robust embeddings, especially under complex conditions. The MLP classifier also showed clear benefits: its ability to integrate heterogeneous features significantly increased precision, mitigating the risk of incorrect associations caused by spurious similarity.

A visual comparison of the results is presented in Figure 4.3, which summarizes the performance of the six tested configurations. The clear separation between the triplet loss variants and all others is particularly evident in the recall, indicating the network's greater sensitivity in correctly identifying previously seen objects, even under partial transformations or occlusions.

Overall, Siamese networks — and in particular the triplet network combined with a supervised classifier — proved to be the most effective solution among those explored, both in absolute terms and relative to computational complexity. The ability to directly model the distance between instances, rather than classifying objects individually, enabled the network to naturally capture temporal dynamics. This turned out to be particularly advantageous in real-world, high-variability scenarios such as those represented in the test datasets, confirming the suitability of Siamese networks for applications involving object tracking, recurrent recognition, and identity continuity.

4.1.4 Comparison between architectures

After analyzing in detail the three families of proposed architectures, some comparative considerations can be drawn, highlighting the strengths, limitations, and structural differences between the models.

In terms of overall accuracy and the balance between precision and recall, the best configurations of each architecture showed competitive performance, but with noticeably different behaviors. Table 4.4 summarizes the results obtained from the best variants of the three families, evaluated using the main metrics: Accuracy, Precision, Recall, F1-score, and AUC.

Table 4.4: Comparison between the best variants of each architecture: baseline, memory-augmented network (MAN), and Siamese approach.

Architecture	**Accuracy**	**Precision**	**Recall**	**F1**	**AUC**
Baseline – ResNet50 + supervised MLP	0.703	0.678	0.872	0.761	0.78
MAN – Bbox + ResNet50 + learnable Transformer + supervised MLP	0.756	0.725	0.843	0.781	0.82
Siamese – Triplets + ResNet50 + supervised MLP	**0.787**	**0.747**	**0.906**	**0.818**	**0.85**

The baseline architectures, in the supervised MLP version, represented a simple yet surprisingly effective solution, especially due to the combination of visual and geometric features. However, the absence of an explicit temporal context and the reliance on thresholds or local classifiers make them sensitive to sudden variations and ambiguous objects.

The Memory-Augmented Network introduced a significant improvement thanks to the integration of a learnable memory. The Transformer module enabled the modeling of temporal relationships between objects, producing contextualized and flexible embeddings. This resulted in greater precision in difficult cases and improved overall stability, making the MAN architecture particularly suitable for realistic and dynamic scenarios. Nevertheless, the computational cost — both during training and inference — is noticeably higher than the baseline.

Siamese networks, and in particular the triplet network, proved very effective in building a coherent metric space for instance comparison. Although they do not include an explicit memory, the structure of the loss function allowed the network to learn how to distinguish between objects observed at different times based on their identity, rather than mere visual similarity. The addition of a supervised MLP in the decision phase further improved precision, narrowing the gap with memory-based architectures. The main limitation of the Siamese approach lies in the

higher complexity of data generation (pairs and triplets) and the need to carefully manage the balance between positives and negatives during training.

The configuration that achieved the best absolute performance was `Triplets - ResNet50 + supervised MLP`, closely followed by `MAN Learnable + ResNet50 + MLP`. Both architectures demonstrated high generalization capacity, robustness to visual variations, and temporal decision consistency. The supervised baseline, although simpler, provided an important reference point, especially for low-complexity computational contexts.

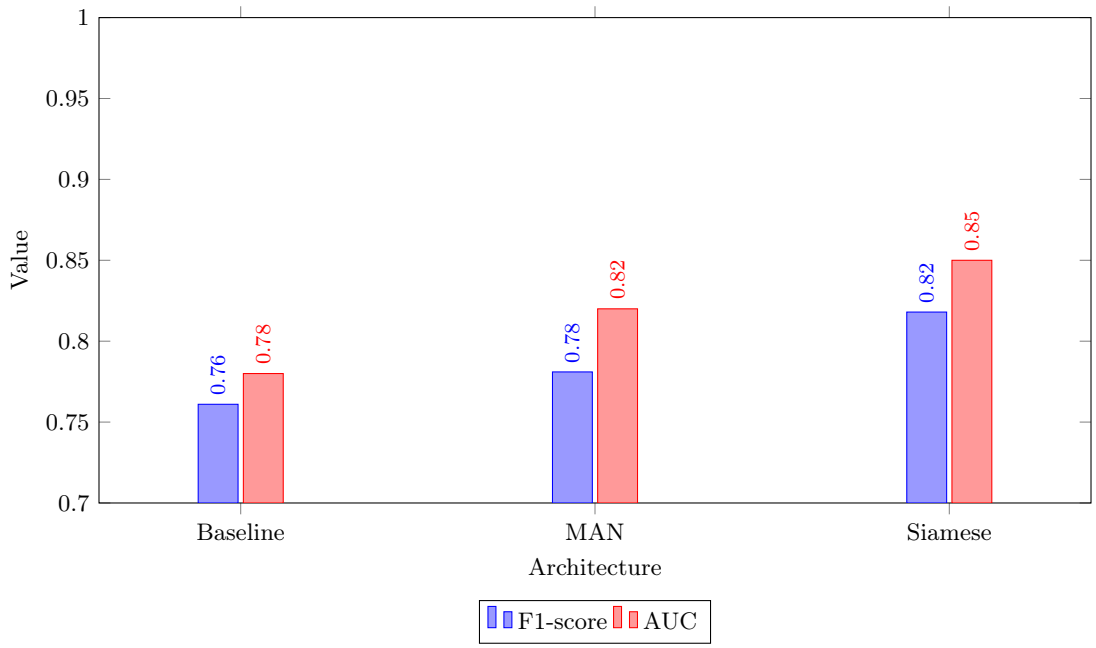


Figure 4.4: Synthetic comparison between the best configurations for each architecture in terms of F1-score and AUC.

The choice of the best architecture ultimately depends on the application context: if absolute precision and temporal stability are prioritized, Siamese networks and MAN represent the best options. Conversely, if lightness and simplicity are the main constraints, the supervised baseline offers an effective solution at a low computational cost. In the following sections, these aspects will be further explored in relation to the practical and industrial implications of the proposed system.

4.2 Qualitative visualizations of the results

In addition to quantitative analyses, it was useful to complement the evaluation with a qualitative assessment of the results obtained by the different architectures. Direct observation of the visual predictions allows us to capture aspects that are not immediately evident from the numerical results: temporal coherence, label stability, geometric correctness of associations, and sensitivity to local ambiguities. This section presents some representative examples, selected both among correctly handled cases and among the most problematic ones.

The images shown are taken from the validation set and illustrate the behavior of the three main architectures analyzed: the supervised baseline with MLP, the Memory-Augmented Network with learnable Transformer, and the Siamese network trained with triplet loss and MLP in the decision phase.

The first group of examples presents cases where all architectures produced correct and temporally stable predictions. These examples highlight the models' ability to maintain a consistent

identity for the same object across image sequences, even in the presence of slight changes in position or rotation.

Figure 4.5: Example of correct tracking for a glass jar: all architectures succeed in maintaining temporal coherence and recognizing the object as already seen, despite the change in perspective.

However, it is in ambiguous or incorrect cases that the differences between architectures become more evident. In particular, there were situations where the supervised baseline failed to recognize a previously seen object, especially in the presence of geometric changes or partial occlusions. In these same images, the MAN or the Siamese network managed to maintain the correct association, thanks respectively to the learnable visual memory and the learned metric structure.

A typical case is when an object undergoes a partial rotation or a change in scale: the baseline tends to treat it as "new," whereas the other architectures, having access to contextual or relational information, manage to identify it correctly.

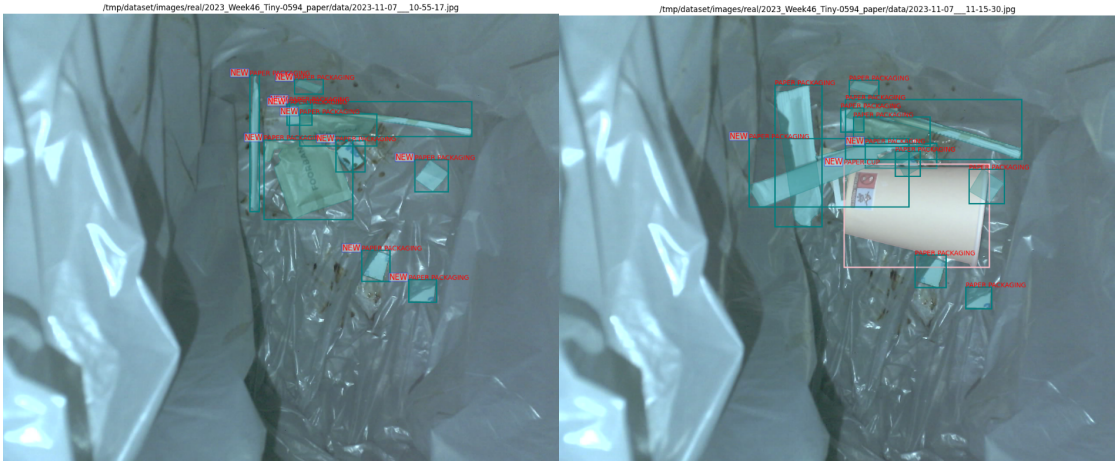
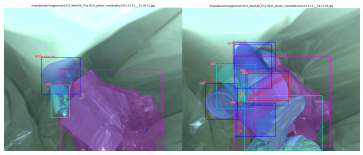


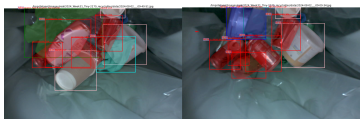
Figure 4.6: Ambiguous example: the paper packaging is rotated between two consecutive frames. The supervised baseline incorrectly labels the object as "new" in the second frame, while the MAN and Siamese architectures correctly maintain the identity.

Another example concerns situations where multiple similar objects appear simultaneously in the scene. In these contexts, it was observed that the MAN — thanks to the Transformer — can disambiguate identity by leveraging historical information, while the Siamese network relies more on metric and spatial coherence. The baseline, instead, may confuse the instances if visual similarity is high and geometric features do not provide sufficient contrast.

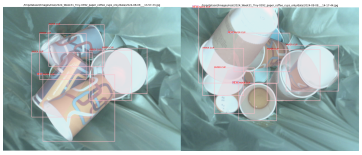
Finally, some recurring failure cases were analyzed. For example, when an object undergoes deformation or is partially occluded, all architectures show signs of weakness, but in different ways: the baseline tends to "reset" the identity, the MAN can be affected by noisy information in memory, and the Siamese network may collapse on visually similar but non-identical objects.



(a) Supervised Baseline
The object in the upper left (plastic cup) undergoes a position change and partial occlusion, causing the model to mistakenly treat it as "new."



(b) Memory-Augmented Network (MAN)
The simultaneous presence of similar objects (red cans) causes ambiguity in memory: a previously seen object is labeled as "new."



(c) Siamese Network
The cups are visually similar, but the lack of temporal memory leads to an incorrect identity association.

Figure 4.7: Failure examples of the three analyzed architectures. Each image highlights a structural limitation of the respective approach in maintaining visual identity consistency.

Thus, the qualitative visualizations confirm and enrich the quantitative findings. They show that the quality of predictions does not depend solely on the architecture type, but also on the nature of the variations that the object undergoes between frames. The most robust architectures are not necessarily those with the highest average metrics, but those that maintain consistency even in borderline cases. In this sense, MAN and the Siamese network prove more reliable in complex scenarios, while the baseline performs well only under regular or low-variability conditions.

The integration of these visualizations into the analysis process allowed for a more refined understanding of the models' behavior, highlighting not only "how much" but also "why" a network fails, and offering valuable insights for future improvements.

4.3 Error analysis

Error analysis is a crucial step for understanding the actual weaknesses of the analyzed architectures and for identifying the structural limitations of the models beyond aggregated metrics. This section classifies the main types of observed errors and discusses their most frequent causes, also in light of the qualitative visualizations.

Among the most common cases are the **false positives**, i.e., instances where a new object is erroneously associated with one already observed. This type of error typically occurs in the presence of visually similar objects belonging to the same category (e.g., two cans, two plastic bottles) but representing distinct entities. It is a particularly evident issue in the fixed-threshold baseline, where a high cosine similarity can easily lead to an incorrect association. Even in the MAN and the Siamese network, if not properly trained or lacking geometric features during the decision phase, false positives are observed in cases where the new object has shape, size, and color very similar to one already stored in memory.

A second recurring error is that of **false negatives**, namely the failure to recognize a previously seen object. In this case, the system classifies as "new" an object that is in fact the same instance as one that appeared in a previous image. These cases mainly occur when the object significantly changes position, scale, or orientation, or when it is partially occluded. Such variations introduce an apparent distance in the visual embedding, which may exceed the decision thresholds or confuse the classifier. Supervised architectures (MLP) have proven more robust to this type of error, thanks to their ability to combine visual and geometric information, whereas models relying solely on thresholds are particularly vulnerable.

Figure 4.8: Example of a false positive for each architecture. A new object (paper cup) is mistakenly associated with a previously seen one due to strong visual similarity.

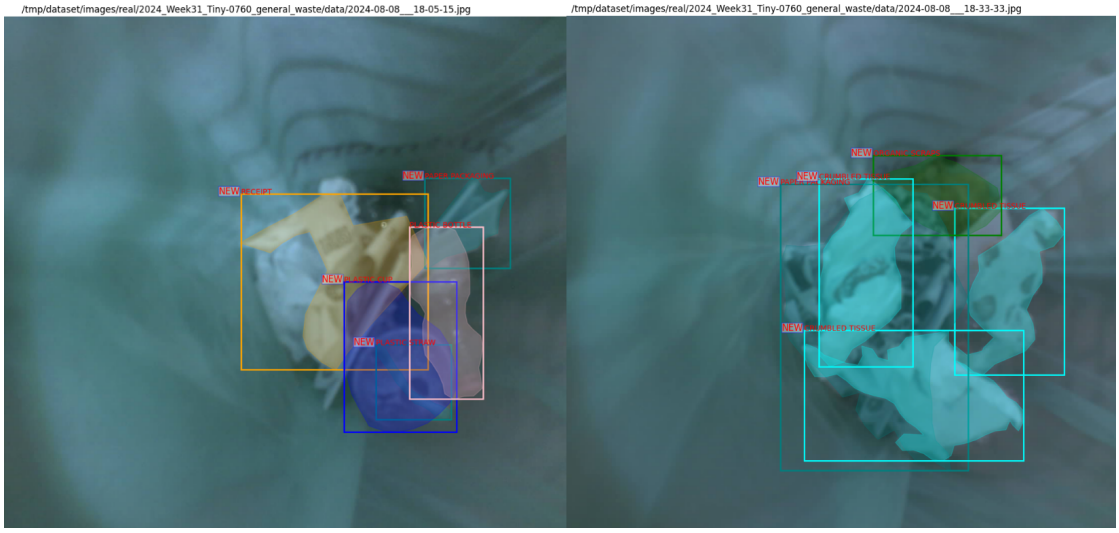


Figure 4.9: Example of a false negative: the same instance appears in two consecutive frames but is mistakenly labeled as "new." The variation in position and context leads to a high apparent distance in the embedding, exceeding the model's decision threshold.

An additional set of errors can be attributed to **spatial or contextual ambiguities**, which lead to inconsistent behavior by the models. These are situations in which multiple similar objects appear simultaneously, or in which an object undergoes partial occlusions or deformations, as frequently occurs with flexible or transparent materials. In such cases, the quality of the visual embedding tends to degrade, and the absence of contextual references leads to inconsistent predictions, even from more advanced models. The MAN, despite having a learnable memory, can suffer from noise accumulated in the FIFO, while the Siamese network, if not properly regularized, may collapse onto similar objects in memory.

Systematic errors related to visually weak categories are also observed, such as fragments of plastic, transparent bags, or irregular pieces of paper. In these cases, the shape is poorly defined and the visual appearance varies greatly from frame to frame. Supervised architectures struggle to generalize, while similarity-based comparisons become unreliable. Another systematic case involves **reflections, blurring, or uneven lighting conditions**, which alter the visual embedding and degrade the temporal coherence of memory.

Finally, it is important to mention **errors due to temporal misalignment**. In some

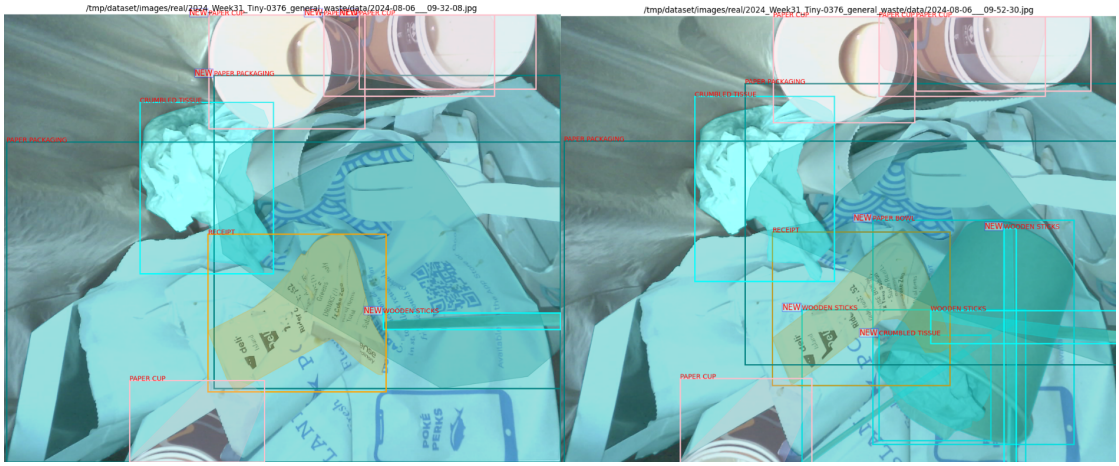


Figure 4.10: Example of inconsistent prediction in a crowded scene with spatial ambiguities. The simultaneous presence of similar objects and partial deformations reduces the quality of the visual embeddings. Even advanced models like MAN or the Siamese network exhibit uncertainty — respectively due to noise in the learnable memory or metric collapse on similar instances.

sequences, the image acquisition frequency and the dynamics of the environment lead to changes that are too rapid between one frame and the next. Under these conditions, even if the object is in fact the same, the visual change is such that it breaks continuity, causing the system to erroneously classify the object as "new." MAN and the Siamese network with MLP have shown some resilience to this problem, but in extreme cases, no model was able to maintain consistency.

Thus, the error analysis shows that, although advanced models significantly reduce the number of incorrect predictions compared to the baseline, there are still structurally critical situations that challenge even the most complex architectures. Errors related to loss of context, visual ambiguity, or temporal instability highlight the need for even more robust memory aggregation mechanisms and semantic representations.

4.4 Critical reflections on the proposed solutions

The experimental analysis made it possible to precisely evaluate the behavior of the proposed architectures in different scenarios, both from a quantitative and qualitative standpoint. This section presents a number of cross-cutting critical reflections that emerged during the development and validation phases, focusing on key aspects such as prediction stability, sensitivity to decision thresholds, generalization capability, and computational complexity.

A first relevant aspect is the **temporal stability of predictions**. Architectures based on local comparison (such as the baseline) exhibit good consistency under simple conditions but tend to lose stability in the presence of deformed objects, scale changes, or lighting variations. In such cases, the lack of an explicit temporal context leads to label oscillations ("new" – "already seen") even for the same instance. More advanced architectures — MAN with learnable memory and the Siamese network with triplet loss — show greater robustness thanks to their ability to model global information: the former leverages the context of past observations, while the latter constructs a coherent metric space that better handles local transformations. However, even in these cases, stability is not guaranteed in the presence of abrupt changes or visually weak objects.

Another critical issue is the **sensitivity to threshold selection**, which became particularly evident in the unsupervised variants. In all architectures relying on a fixed threshold on cosine similarity (or a combined score), performance is heavily influenced by the chosen value. Thresholds that are too low lead to false positives, while excessively high thresholds increase false negatives. This makes calibration complex and unstable, especially when the system is to be deployed on different datasets or under new operating conditions. By contrast, the use

of a supervised classifier (MLP) significantly reduced this sensitivity, proving more flexible in adapting the final decision based on visual and geometric features learned jointly.

Regarding **generalization capability**, all architectures performed well on common categories and objects well represented in the dataset. However, generalizing to rare, deformable, or visually ambiguous objects proved more challenging. In particular, the baseline — while effective on canonical objects — showed fragility when faced with less frequent or low-information instances. MAN was able to maintain a certain level of consistency even with rare objects, provided they appeared at least once in the sequence. The Siamese network benefited from its metric structure, which enables more continuous generalization, but only if the negatives during training were sufficiently diverse and balanced. This highlights the importance of a good sampling strategy in metric-based models.

Finally, it is important to consider the **computational complexity and resource requirements** of each solution. The supervised baseline, based on direct comparison and an MLP, stands out for its simplicity and low computational demands: it is a reasonable choice in resource-constrained environments such as embedded or edge systems. MAN, on the other hand, introduces a high computational cost, both for feature processing via the Transformer and for managing and updating the visual memory. Training requires more time and careful stability management. The Siamese network lies in an intermediate position: inference is fast and lightweight, but the generation and balancing of triplets during training represents a significant overhead. Moreover, the need to compare each object with all those in memory (or with a selected subset) can become costly as the sequence length increases.

In summary, each architecture presents specific advantages and limitations. Supervised models with memory or metric structure offer the best performance, but at the cost of increased complexity. The most suitable model therefore depends not only on the achieved metrics, but also on the operational context, available resources, and the requirements for temporal stability and adaptability.

4.5 Applications and practical implications in industrial contexts

From the perspective of **efficiency and scalability**, simpler supervised architectures—such as the baseline with MLP—are well suited for industrial scenarios where rapid response is required, even over long image sequences. The model requires a limited number of comparisons per object and can operate in real time on mid-range hardware. However, the lack of a more advanced memory structure makes these solutions less appropriate in complex scenarios where multiple similar objects appear or reappear at different times.

The Memory-Augmented Network, while offering superior performance, introduces a higher computational cost, mainly due to the Transformer module and FIFO memory management. In an industrial context, deploying such an architecture would require careful optimization or the use of more powerful hardware (e.g., GPU-equipped edge devices or multi-core CPUs in cloud or local gateways). The scalability of MAN is constrained by the memory sequence length and the inference complexity of the Transformer, but its potential in dynamic, multi-object environments remains significant.

The Siamese network with triplet loss presents an intermediate solution in terms of cost and performance. Once trained, inference is lightweight, and comparisons can be executed efficiently, making it suitable for embedded devices, provided they have sufficient computational capacity for batch embedding processing. The main challenge lies in preprocessing and dataset construction for training, which in a production system would require reliable automation or pre-annotated datasets for object temporal identity.

In terms of **compatibility with edge or embedded hardware**, the choice of architecture depends heavily on the constraints of the target system. The baseline with ResNet18 and MLP

can be integrated into existing computer vision pipelines with minimal overhead. Architectures based on ResNet50, learnable memory, or triplet networks instead require a careful assessment of the available computational budget and, if needed, model optimization techniques such as quantization, pruning, or distillation.

Finally, from the perspective of **operational and environmental impact**, a system capable of automatically recognizing previously seen objects enables reduction of counting errors, detection of anomalous behaviors (e.g., repeated insertion of the same waste item), and optimization of material classification. This could lead to smarter waste cycle management, increased collection efficiency, and reduced human intervention. In the future, such systems could be deployed not only in urban smart bins but also in automated sorting facilities, recycling centers, or industrial contexts where tracking reusable or recyclable objects over time is important.

Overall, the results obtained demonstrate that the proposed architectures are transferable to real-world applications, provided that trade-offs between accuracy, stability, and computational resources are considered. The ability to choose between lightweight and more sophisticated models provides important flexibility, which can be leveraged to adapt the system to the specific operational requirements of each industrial scenario.

Chapter 5

Conclusions and Future Work

5.1 Summary of the work

This thesis addressed the problem of temporal object recognition in image sequences, with particular focus on the context of smart bins—intelligent containers for waste collection. The proposed task consists in automatically determining, for each detected object in a visual sequence, whether it is a new occurrence or an instance already observed in a previous frame. This is a non-trivial problem that goes beyond standard object detection and requires the integration of memory and temporal tracking components.

The first part of the work focused on dataset preparation: starting from a corpus of approximately 7000 images annotated in COCO format, a thorough restructuring of the annotations was carried out, with special attention to the `new` attribute, used to label the "novelty" of objects. A subset of images was also created to support preliminary exploration of the architectures.

Subsequently, three families of architectures were designed and compared, each characterized by different approaches and underlying assumptions:

- A **supervised baseline**, where objects are compared with those in memory using visual and geometric features, and classified by a binary MLP;
- A **Memory-Augmented Network (MAN)**, which introduces a learnable Transformer module to represent the historical context of past observations;
- A **Siamese network** in two variants: a binary version with contrastive loss and a triplet-based version with triplet loss, designed to learn a metric space in which object distance reflects their temporal identity.

For each family, several variants were explored: changes in backbone (ResNet18 and ResNet50), input types (RGB crop, masks, soft-mask), decision strategies (fixed threshold or supervised MLP), and different memory or matching mechanisms. The non-trivial problem of automatic pair and triplet generation was also addressed, enabling Siamese models to learn not only visual similarity, but also the temporal continuity of objects.

The entire system was evaluated through a comprehensive and structured experimental analysis. Classical binary classification metrics (accuracy, precision, recall, F1-score, AUC) were used, supported by qualitative visualizations and detailed error analysis. This made it possible not only to compare the models, but also to better understand their limitations, behavior under non-ideal conditions, and ability to adapt to realistic scenarios.

5.2 Strengths and limitations

One of the most evident strengths of this work lies in its comparative approach, which enabled the study of the problem from multiple perspectives. The three proposed architectures are based on different assumptions (local decision-making, explicit memory, metric learning) and produced complementary results. This allowed for an in-depth analysis of the trade-offs between simplicity and power, generalization and specialization, local precision and global temporal coherence.

The supervised baseline demonstrated that, with intelligent use of visual and geometric features, good performance can be achieved even with lightweight models. This architecture represents a viable solution for industrial contexts with limited resources, where simplicity and computational efficiency are primary constraints.

The Memory-Augmented Network introduced a dynamic memory component capable of modeling the context of past observations. The learnable Transformer showed excellent ability in handling temporal variability, maintaining coherence across frames even in the presence of partially occluded or deformed objects. However, this architecture has inherently higher complexity, requiring greater computational resources, careful tuning, and robust memory management mechanisms.

The Siamese network — especially in its triplet variant with supervised classification — proved extremely effective in recognizing the temporal identity of objects. Its main strength lies in its ability to learn a coherent metric space, in which similar but distinct objects are clearly separated. Nevertheless, the generation of training data (pairs and triplets) requires a refined logic, and the model's quality strongly depends on the quality of positive and negative samples.

On the limitations side, the work highlighted several common weaknesses. Firstly, all architectures exhibit fragility in the presence of visually uninformative objects such as transparent plastic, crumpled paper, reflective or deformable materials. Furthermore, memory management — both in explicit form (MAN) and implicit (FIFO) — can introduce noise or uncertainty, especially when objects move, rotate, or undergo scale changes across frames.

Another critical aspect is the sensitivity to data distribution. Supervised models perform well only when trained on balanced and varied data. Some rare or underrepresented categories in the dataset can lead to systematic errors or weak generalization. In particular, the construction of pairs and triplets for the Siamese network required significant care: unbalanced sampling can quickly degrade the effectiveness of the loss function, leading to overfitting or underfitting.

Finally, the trade-off between accuracy and inference time represents a concrete challenge in terms of deployment. Although the MAN and Siamese architectures offer excellent performance, they require resources that may not be available on embedded or real-time systems. This calls for careful reflection on the scalability and optimization of models for practical use in production environments.

5.3 Possible improvements and extensions

The future directions of this work are broad and varied. On the architectural level, a first avenue involves the introduction of **long-term or hierarchical memory** mechanisms, to retain persistent traces of objects across extended temporal windows. This could be achieved through adaptive cache mechanisms or recursive structures such as Long-Term Memory Networks.

A second relevant extension concerns **multimodal fusion**: the integration of additional sensory channels (e.g., weight data, sounds, temperature) could enhance object discrimination in ambiguous cases, especially for similar materials.

Another promising area for improvement involves the **automatic generation of training data**. Particularly for Siamese networks, the adoption of auto-labeling techniques or self-supervised learning guided by temporal coherence could drastically reduce annotation costs, making the system more scalable. In parallel, exploring **semi-supervised** or **weakly-supervised** strategies would allow for more efficient use of the large volume of unlabeled data available.

In terms of adaptability, the introduction of **continual learning** or **few-shot learning** capabilities would be strategic: a system capable of updating itself online — retaining memory of previously seen objects and adapting to new classes or conditions — would be especially useful in dynamic environments such as urban waste management.

Additionally, a possible development direction includes the **systematic exploration of architectural variants** and **hyperparameter optimization**: modifying the depth of the backbone, experimenting with alternatives to triplet loss (e.g., quadruplet loss, N-pair loss), or exploring alternative attention mechanisms (e.g., Performer, Linformer) could lead to further performance improvements.

From an application perspective, the system could be extended to broader tasks, such as **material recognition**, **verification of proper disposal**, or **automatic estimation of volume/fill level**. An integrated system capable of tracking, classifying, and providing user feedback would be a valuable tool to promote recycling and improve waste management efficiency.

Finally, it will be essential to carry out **model optimization for deployment**, through techniques such as pruning, quantization, or knowledge distillation, to ensure compatibility with the strict hardware constraints typical of embedded and real-time applications.

In conclusion, the work carried out has allowed for a structured exploration of the temporal instance recognition problem, comparing different approaches and helping to clarify the limitations and potential of the main architectural solutions. The experimental evidence, the challenges addressed, and the improvement opportunities identified provide a solid foundation for future developments, with promising applications both in research and in concrete industrial scenarios.

Bibliography

[1] R. Girshick, «Rich feature hierarchies for accurate object detection and semantic segmentation», *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[2] R. Girshick, «Fast r-cnn», *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[3] S. Ren, K. He, R. Girshick, and J. Sun, «Faster r-cnn: Towards real-time object detection with region proposal networks», *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[4] Z. Zhao, P. Wang, S. Zhang, *et al.*, «Object detection in 20 years: A survey», *arXiv preprint*, 2019. eprint: `1905.05055`.

[5] W. Liu, D. Anguelov, D. Erhan, *et al.*, «Ssd: Single shot multibox detector», in *European Conference on Computer Vision (ECCV)*, Springer, 2016.

[6] L. Weng, «Object detection for dummies part 3: R-cnn family», *lilianweng.github.io*, 2017. [Online]. Available: `https://lilianweng.github.io/posts/2017-12-31-object-recognition-part-3/`.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, «You only look once: Unified, real-time object detection», in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[8] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, «Yolov4: Optimal speed and accuracy of object detection», *arXiv preprint*, 2020. eprint: `2004.10934`.

[9] G. Jocher *et al.*, *Yolov5 by ultralytics*, `https://github.com/ultralytics/yolov5`, 2020–2023.

[10] Z. Zou, Z. Shi, Y. Guo, and J. Ye, «Object detection with deep learning: A review», *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 11, 2019.

[11] C. Huang and e. a. Li, «Yolov4-based method for object detection in low-illumination environments», *Journal of Sensors*, 2022.

[12] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, «Distance-iou loss: Faster and better learning for bounding box regression», *AI Conference on Artificial Intelligence*, vol. 34, no. 07, 2020.

[13] Z. Tian, C. Shen, H. Chen, and T. He, «Fcos: Fully convolutional one-stage object detection», in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2019.

[14] X. Zhou, D. Wang, and P. Krähenbühl, «Objects as points», *arXiv preprint arXiv:1904.07850*, 2019.

[15] H. Law and J. Deng, «Cornernet: Detecting objects as paired keypoints», in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018.

[16] K. Simonyan and A. Zisserman, «Very deep convolutional networks for large-scale image recognition», *arXiv preprint*, 2014. eprint: `1409.1556`.

[17] K. He, X. Zhang, S. Ren, and J. Sun, «Deep residual learning for image recognition», *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[18] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, «Feature pyramid networks for object detection», in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[19] X. Ma, Y. Li, Z. Yang, and H. Lin, «Dsyolo-trash: Object detection and tracking for intelligent waste sorting systems», *Preprints.org*, 2024.

[20] T. Schneider, R. Urban, *et al.*, «Deep learning-based object recognition and segmentation using cnns for industrial environments with lidar projection», *Sensors*, vol. 24, no. 2483, 2024.

[21] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, «Simple online and realtime tracking», in *2016 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2016.

[22] N. Wojke, A. Bewley, and D. Paulus, «Simple online and realtime tracking with a deep association metric», in *2017 IEEE International Conference on Image Processing (ICIP)*, 2017, pp. 3645–3649. DOI: 10.1109/ICIP.2017.8296962. [Online]. Available: https://doi.org/10.1109/ICIP.2017.8296962.

[23] K. Koufos, T. Protopsaltis, S. Vrochidis, and I. Kompatsiaris, «A comprehensive benchmark and analysis for person re-identification embeddings in multi-object tracking», *Proceedings of the 18th International Conference on Informatics in Control, Automation and Robotics (ICINCO)*, 2021.

[24] K. Zhou, Y. Xiang, and S. Gong, «Omni-scale feature learning for person re-identification», in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2019.

[25] J. Cai, M. Xu, W. Li, *et al.*, «Memot: Multi-object tracking with memory», in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[26] R. Gao and L. Wang, «Memotr: Long-term memory-augmented transformer for multi-object tracking», in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[27] P. Liao, F. Yang, D. Wu, and B. Liu, «Fasttracktr: Towards fast multi-object tracking with transformers», *Multimedia Systems*, 2025, Preprint.

[28] X. Zhou, T. Yin, V. Koltun, and P. Krähenbühl, «Global tracking transformers», in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[29] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, «Attention is all you need», in *Advances in neural information processing systems*, vol. 30, 2017.

[30] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, «An image is worth 16x16 words: Transformers for image recognition at scale», in *International Conference on Learning Representations (ICLR)*, 2021.

[31] G. Bertasius, H. Wang, and L. Torresani, «Is space-time attention all you need for video understanding?», in *International Conference on Machine Learning (ICML)*, 2021.

[32] Z. Liu, J. Ning, Y. Cao, *et al.*, «Video swin transformer», *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[33] S. Li, S. Bak, P. Carr, and X. Wang, «Diversity regularized spatiotemporal attention for video-based person re-identification», *arXiv preprint arXiv:1803.09882*, 2018.

[34] A. Graves, G. Wayne, and I. Danihelka, «Neural turing machines», *arXiv preprint arXiv:1410.5401*, 2014.

[35] A. Graves, G. Wayne, M. Reynolds, *et al.*, «Hybrid computing using a neural network with dynamic external memory», *Nature*, vol. 538, no. 7626, 2016.

[36] J. Gao, K. Zeng, Y. Chen, and R. Nevatia, «Motion-appearance co-memory networks for video question answering», in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018.

[37] Z. Fang, L. Xu, and C. Xu, «Dawn: Temporal object tracking through memory enhanced attention for video question answering», in *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[38] X. Wu, C. Liu, and Z. Wang, «Memvit: Memory-augmented video transformer for efficient long-term video understanding», *arXiv preprint arXiv:2205.12559*, 2022.

[39] S.-Y. Chen *et al.*, «Flexible hardware architecture of hierarchical k-means clustering for large cluster number», *Journal of VLSI Signal Processing*, 2011.

[40] J. Bromley, I. Guyon, Y. LeCun, E. Säckinger, and R. Shah, «Signature verification using a "siamese" time delay neural network», in *Advances in neural information processing systems*, 1993.

[41] S. Chopra, R. Hadsell, and Y. LeCun, «Learning a similarity metric discriminatively, with application to face verification», in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, vol. 1, 2005.

[42] F. Schroff, D. Kalenichenko, and J. Philbin, «Facenet: A unified embedding for face recognition and clustering», *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2015.

[43] R. Hadsell, S. Chopra, and Y. LeCun, «Dimensionality reduction by learning an invariant mapping», in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, vol. 2, 2006.

[44] A. Hermans, L. Beyer, and B. Leibe, «In defense of the triplet loss for person re-identification», in *arXiv preprint arXiv:1703.07737*, 2017.

[45] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, «Arcface: Additive angular margin loss for deep face recognition», in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[46] Y. Sun, L. Zheng, Y. Yang, Q. Tian, and S. Wang, «Circle loss: A unified perspective of pair similarity optimization», in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[47] Y. Zheng, Q. Liu, and W. Hu, «Re-idformer: Transformer-based architecture for video object re-identification», *Journal of Visual Communication and Image Representation*, 2023.

[48] Y. Luo, W. Chen, and Z. Fang, «Crossformer: Cross-temporal transformer for video object matching in streaming applications», in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024.

[49] S. Zheng, X. Xu, and W. Lin, «Visual tracking via hybrid memory and metric alignment», *Neurocomputing*, vol. 525, 2023.

[50] T. Luo, J. Zhang, and J. Hu, «Temporal-aware metric learning for object re-identification in streaming environments», in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2024.

[51] O. Adedeji and Z. Wang, «Wastenet: A lightweight deep learning model for waste classification on edge devices», *Procedia Manufacturing*, vol. 35, 2020.

[52] R. Kunwar and T. Alade, «A review on smart waste bin systems using computer vision and iot», *Sustainability*, vol. 16, no. 2, 2024.

[53] C. Kuang, M. Zhao, and H. Li, «Optimizing yolov5 for real-time waste classification in indoor and outdoor scenarios», *Journal of Environmental Informatics Letters*, vol. 3, no. 1, 2024.

[54] S. Mithra and S. Raj, «Yolobin: Intelligent waste classification and incentive-based recycling system», *International Journal of Computer Applications*, vol. 177, no. 12, 2025.

[55] X. Ma, Y. Li, Z. Yang, and H. Lin, «Dsyolo-trash: Object detection and tracking for intelligent waste sorting systems», *Preprints.org*, 2024.

[56] H. Wang, L. Zhang, Y. Chen, and Y. Liu, «A yolo-based waste classification network with attention mechanisms for intelligent garbage sorting», *Mathematics*, vol. 12, no. 14, p. 2185, 2023. DOI: 10.3390/math12142185.

[57] M. Abo-Zahhad, H. El-Bakry, and A. Khalifa, «Smart overflow detection and classification of waste in bins using edge ai and iot», *Sensors*, vol. 25, no. 1, 2025.