Applied Intelligence (2023) 53:9115–9147
https://doi.org/10.1007/s10489-022-03886-6

A self-interpretable module for deep image classification on small data

Biagio La Rosa[1] · **Roberto Capobianco**[1,2] · **Daniele Nardi**[1]

Accepted: 10 June 2022 / Published online: 5 August 2022

Abstract

Deep neural networks are the driving force of the recent explosion of machine learning applications in everyday life. However, they usually require a lot of training data to work well, and they act as black-boxes, making predictions without any explanation about them. This paper presents Memory Wrap, a module (i.e, a set of layers) that can be added to deep learning models to improve their performance and interpretability in settings where few data are available. Memory Wrap adopts a sparse content-attention mechanism between the input and some memories of past training samples. We show that adding Memory Wrap to standard deep neural networks improves their performance when they learn from a limited set of data, and allows them to reach comparable performance when they learn from the full dataset. We discuss how the analysis of its structure and content-attention weights helps to get insights about its decision process and makes their predictions more interpretable, compared to the same networks without Memory Wrap. We test our approach on image classification tasks using several networks on three different datasets, namely CIFAR10, SVHN, and CINIC10.

Keywords Interpretable deep learning · eXplainable Artificial Intelligence · Memory augmented neural networks · Small data

1 Introduction

In the last decade, Artificial Intelligence has seen an explosion of applications thanks to advancements in deep learning. Despite their success, these techniques suffer from some important problems: they require a lot of data to work well [73], and they act as black-boxes, taking an input and predicting an output without providing any explanation about the decision process. The lack of transparency limits the adoption of deep learning in important domains like health-care [12] and justice, while the data requirement makes its generalization to real-world tasks harder. To overcome the data requirements, researchers propose several solutions that typically exploit additional resources like pre-trained models (e.g., transfer-learning [73]), unlabeled data (e.g., semi-supervised learning [19]), or prior knowledge (e.g., few-shot learning [82]). Conversely, the eXplainable artificial intelligence (XAI) community studies the transparency problem, developing methods that can explain the decision process of AI agents or developing a more interpretable AI. While there is an extensive literature on each topic, few works explore methods that can be used both on small data settings and that are more interpretable.

In this paper, we take a step in this direction, focusing on the domain of computer vision and image classification tasks, and proposing Memory Wrap, a self-interpretable module (i.e., a set of layers) that can be attached to deep learning models. We show that it improves the performance of the model to which it is attached without using any additional resources and it provides, at the same time, a way to inspect its decision process (Fig. 1).

In classical supervised learning settings, deep models use the training set only to adjust their weights, discarding it at the end of the training process. Instead, we hypothesize that, in small data settings, it is possible to strengthen

✉ Biagio La Rosa
larosa@diag.uniroma1.it

Roberto Capobianco
capobianco@diag.uniroma1.it

Daniele Nardi
nardi@diag.uniroma1.it

[1] Department of Computer, Control, and Management Engineering Antonio Ruberti, Sapienza University of Rome, Rome, Italy

[2] Sony AI, Zurich, Switzerland

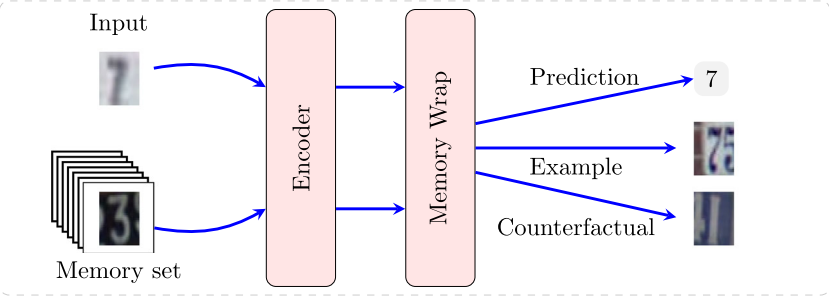

Fig. 1 Overview of Memory Wrap. The encoder takes as input an image and a memory set, containing random samples extracted from the training set. The encoder sends their latent representations to Memory Wrap, which outputs the prediction, an example-based explanation, and a counterfactual, exploiting the sparse content-based attention between inputs encodings

the learning process by re-using samples from the training set during inference. Taking inspiration from Memory Augmented Neural Networks [69], we propose to store a bunch of past training samples (called *memory set*) and combine them with the current input through sparse attention mechanisms [23] to help the neural network decision process. Since the network actively uses these samples during inference, we propose a method based on inspection of sparse content-based attention weights (Section 3.2) to extract insights and explanations about its predictions.

We test our approach on image classification tasks using CIFAR10 [36], Street View House Number (SVHN) [58], and CINIC10 [15] as datasets, obtaining promising results. Our contribution can be summarized as follows:

- we present Memory Wrap, a module for deep neural networks that uses a memory containing past training examples to enrich the input encoding;
- we extensively test its performance, using different backbone deep models on several small data settings, and show that it improves the accuracy of the backbone models in almost all the settings;
- we discuss how its structure makes the predictions more interpretable. In particular, we show that not only it is possible to extract the samples that actively contribute to the prediction, but we can also measure how much they contribute;
- we show how, by analyzing the samples that Memory Wrap actively uses at inference time, it is possible to inspect which features are important for the current prediction and to interpret and diagnose the model behavior;
- we study the main characteristics of the memory samples used by our approach, and divide them as candidates for example-based and counterfactual explanations. Moreover, we show some explanatory usage scenarios as an example.

The manuscript is organized as follows: Section 2 reviews existing literature, focusing on works that use similar methods and discuss the state-of-the-art in network explainability; Section 3 introduces our approach; Section 4 presents some experiments and their results for both performance and interpretability side; Section 5 analyzes the module and its components; and, finally, Section 6 discusses conclusions, limitations, and future directions.

2 Background

2.1 Memory augmented neural networks

Our work takes inspiration from current advances in Memory Augmented Neural Networks (MANNs) [23, 37, 69]. MANNs use a memory module to store and retrieve data during input processing through attention mechanisms. While initially designed to mitigate the problem of catastrophic forgetting on sequential tasks, researchers also apply them to different problems, like visual question answering [51], image classification [8], and meta-learning [65]. In the context of computer vision, MANNs are mainly applied to two types of problems: those that can be cast as sequential [5], like semantic segmentation [5], visual question answering [51], or video summarization [20], and few-shot learning. The latter aims at classifying never-seen objects into novel categories, given a pre-trained model on a set of different classes, which act as prior knowledge. In this case, MANNs can store examples of the novel classes to aid the network in the task [8, 65, 67, 78].

Few-shot learning includes several works that share our idea of using a memory set to strengthen the learning process, like Matching Networks [78], Prototypical Networks [67], Relation Networks [71] and Relational Embedding Network (RENet) [30]. They differ in how they exploit the samples in the memory set. Matching Networks

[78] use them for conditioning the encoding of both the input and the memory set through two LSTM networks. Additionally, they use the linear combination of the samples' labels to predict the class. Prototypical Networks [67] suggest avoiding the usage of LSTM networks because they introduce fictitious temporal dependencies. Hence, they compute prototypes for each class and perform the classification based on the distance between the prototypes and the current input. Finally, Relation Networks [71] and RENet [30] use feature maps of convolutional layers to enrich the input encoding. While the former concatenates the feature maps of both the input and the samples in the memory set, the latter enriches the input encoding by extracting correlations patterns between these feature maps. Finally, the enriched encoding is fed to a small convolutional network, which returns the prediction. Standard image classification settings are underexplored in the MANN literature, and represent an open problem for further research. The few published works on the topic, to the best of our knowledge, only target specific domains, such as bioimages classification [16] and defect pattern classification [29], lacking in generalizability. Typically, these works employ ad-hoc training procedures and learning paradigms tested only on shallow networks, making its generalizability and effectiveness on deep neural networks unclear.

Conversely, our contribution is a module that can be attached to any deep neural network and works on standard training procedures. As in the works on few-shot learning, Memory Wrap uses a memory set to aid the inference process, but, crucially, its architecture enables the interpretability of its behavior. In particular, differently from existing approaches, our module preserves the independence of the memory sample and input encodings, and uses only a subset of the memory set during the inference process. These features, alongside the module architecture, make the interpretation of the predictions easier, a feature not supported by the previous works.

2.2 eXplainable Artificial Intelligence

The field of eXplainable Artificial Intelligence aims at developing methods to help users in understanding the inner working mechanisms of black-box AI agents. [47] distinguishes between *transparent models*, where one can unfold the chain of reasoning (e.g., decision trees), and *post-hoc explanations*, which explain predictions without looking inside the box. Additionally, in the context of deep learning, some recent works propose the so-called self-interpretable deep learning models, which can be placed in between these categories. While they are not fully transparent models yet, they provide elements that can help users understand the decision process.

Examples of this category are architectures that learn and use prototypes, those that add constraints on the learning process of the latent space, and attentive models. The former learn sets of prototypes, which can then be used for interpretability purpose [44]. The typical workflow consists of first learning a set of prototypes that satisfy some constraints, then comparing the input with them, and finally computing the prediction based on the activated prototypes. By associating concepts to the prototypes and by analyzing the closest ones to the input, users can understand which parts of the inputs are important for the current prediction, thus extracting insights about feature attribution. ProtoPNet [10], NP-ProtoPNet [66], and TesNet [80] represent the last advancements of this category. The second set of architectures forces the network to learn more interpretable representations in the form of disentangled or concept aligned representations [11, 35, 77]. In this way, a user can understand by inspection what factors influence the decision process, since each activation captures only a given property. Attentive models include attention modules into their structure, making the elements on which the model is focusing more evident [43, 89]. We place our work in the last category, since it uses the attention mechanism and the structure of the model to provide insights into its decision process. Additionally, our approach presents some of the features of prototype-based explanations but replaces the learned prototypes with samples extracted from the dataset.

2.2.1 Example-based explanations

Example-based explanations are representative instances extracted from given data that help the user to understand how the network works [4]. Ideally, the instances should be similar to the input and, in classification settings, predicted in the same class. In this way, by comparing the input and the examples, a human can extract both similarities between them and features that the network uses to make predictions.

These explanations are usually connected to case-based reasoning and prototype learning. Prototype-learning includes the architectures that learn a set of prototypes, as described in the previous section. The prototypes are fixed latent representations against which the model compares the input and performs its computations [45]. By inspecting the prototypes, it is possible to understand which parts of the inputs have a strong influence on the decision process [10]. Additionally, by using techniques like K-Nearest Neighbours (K-NN) [14], one can retrieve similar samples to the prototypes over the latent space and use them as global example-based explanations.

Case-based reasoning approaches use a proxy model to explain the black-box by learning a mapping between them. An example is the usage of K-NN over the last latent space or enhanced versions [32] that assign different weights to

each dimension based on the features values [60] or their attributions [31].

As in the first set of methods, Memory Wrap uses the comparison between the input and some examples to increase the interpretability of its decision process. By design, samples used during the inference process and associated with the same prediction can be seen as good candidates for example-based explanations. Identifying these samples can aid the users in better understanding the decision process. While prototype learning provides global example-based explanations, Memory Wrap replaces the learned prototypes with dataset samples that are different each time, thus providing local explanations. Note that the model chooses these samples not because they are the optimal set of example-based explanations but because they are the most useful for computing the current prediction. In fact, they do not represent an alternative to post-hoc methods, which one can still apply on top of Memory Wrap, but a fast and cheap way to inspect its decision process.

2.2.2 Counterfactuals

Counterfactuals are specular to example-based explanations: in this case, the instances should be similar to the current input but classified in another class. By comparing the input to counterfactuals, it is possible to highlight differences and extract edits that one should apply to the current input to obtain a different prediction.

While it is feasible to get counterfactuals for tabular data by changing features and at the same time respect domain constraints [56], the task is more challenging for images and text. The difficulty is caused by the lack of formal constraints and the huge number of features involved. The possible solutions are adopting search methods that select some data samples as counterfactuals or generating them through generative models or perturbations. While the first approach has scalability problems on large state, the latter must deal with the generation of unrealistic samples or out-of-distribution samples [50, 79].

Both generative and perturbation-based approaches are based on the idea of minimizing a cost function that should take into account several factors, like the predictions on the perturbed instance, the desired outcome [79], the closeness of features [39], and the closeness to prototypes [50]. For example, [79] propose to guide the perturbation process by using a loss function that minimizes the difference between the predictions on the perturbed instance and the the desired outcome and the L1 norm of the perturbations. Recently, [50] improve the previous approach by adding a term that penalizes perturbations distant from a set of prototypes. One of the problems of perturbation-based methods and iterative process is their high latency due to the large search space. Liu et al. [49] try to mitigate this problem by combining Generative Adversarial Networks (GANs) and editing mechanisms in place of iterative processes. The results are promising, but – since GANs are black-boxes themselves – it is difficult to understand why a particular counterfactual is a good candidate or not. While our module shares the idea of the above-mentioned methods, i.e., using samples that are similar to the input but predicted in a different class as counterfactuals, our goal is not to select the optimal set of counterfactuals but to provide a more transparent decision process. Hence, as in the case of example-based explanations, Memory Wrap chooses candidate counterfactuals only among the samples actively used by the network during its decision process. When available, these samples can be treated as fast candidates for counterfactuals and used to inspect the module's behavior, achieving the double objective of improving the training process and providing insights about it (Section 4.3.2).

2.3 Image classification

Deep image classification is the task of learning a mapping between images and labels using a deep neural network. Its results are often used for improving the performance of related tasks such as detection [9], segmentation [21], image coloring [42], and text recognition [84, 86].

In the last ten years, the field has seen a considerable performance improvement, thanks to the explosion of deep learning techniques that replaced the hand-crafted filters used before. In particular, Convolutional Neural Networks [26, 28, 57, 64, 72, 74, 87, 88] are the most popular architectures, learning and applying filters through the chain of operations performed at each convolutional layer. Thanks to the availability of bigger and bigger datasets, the networks have grown in size, reaching millions of parameters, like in the case of the emerging class of architectures based on the Transformers [18]. However, the usage of large-scale datasets and the massive size of these networks require more and more resources and training time, thus making the development and the adoption of these networks harder [12].

The research community is actively working to solve this issue, proposing solutions that can be adopted on tasks that involve small data or low resources. Examples of such approaches are pre-trained models (e.g., transfer-learning [73]), unlabeled data (e.g., semi-supervised learning [19]), custom training paradigms, or novel blocks for specific architectures [76]. The first category includes transfer learning techniques and few-shot learning. Transfer learning consists of training a model on a large dataset, like ImageNet [63], and then using the learned weights as a starting point for the training of the same network on a smaller dataset. Instead, few-shot learning [82] aims at learning entirely novel classes

using only a few samples, starting from a pre-trained model on different classes that acts as prior knowledge. The second type of approach is applicable when unlabeled data are available and the distribution of data follows certain assumptions [19]. For instance, in the semi-supervised learning paradigm, the network can use its predictions to assign soft labels to the unlabeled data and use them during the training process. Finally, some works propose to modify the training process, introducing novel regularizers [6], or losses [3, 34, 41]. In contrast, we propose a module that can be directly attached to a deep neural network and does not use any pre-trained model, larger datasets, or prior knowledge. The module can be used without changing the training process in settings where we have no additional resources or information about the dataset. With respect to the approaches that modify the training process, our module is complementary, and thus future works could explore their practical combination.

While the domain of image classification includes several other subcategories, like hyperspectral image classification [27, 46] and medical imaging [55], in this paper, we focus on the case of natural image classification, which includes the most common benchmarks used in the settings considered in this paper. However, the structure of the proposed module could potentially be also used on other classification problems, adapting it to the new domains (Section 5.3).

3 Memory wrap

This section presents the proposed module, describing the elements on which it is based (Section 3.1), its structure (Section 3.2), and how to identify example-based explanations and counterfactuals for its predictions among the samples that impact the decision process (Section 3.3).

3.1 Preliminaries

3.1.1 Problem formulation

Given a training set of input-output pairs

$$X = \{(\mathbf{x}, \mathbf{y})\}_{i=1}^{n} \tag{1}$$

where n is the number of data points included in the training set, the objective is to learn a function f that maps a novel input $\mathbf{x}$ to its expected output $\mathbf{y}$:

$$f \ : \ \mathbf{x} \rightarrow \mathbf{y}, \ \ \mathbf{x} \in \mathbb{R}^p, \ \ \mathbf{y} \in \{0, 1\}^c \tag{2}$$

where c denotes the number of classes and p is the dimension of the inputs.

In this paper, we assume that n is small (i.e., $n = \{1000, 2000, 5000\}$), and f is a deep neural network that has to learn the mapping using only the given n available data.

Most deep neural networks f can be represented as the composition of two functions:

$$f_1 : \quad \mathbf{x} \rightarrow \mathbf{z} \tag{3}$$

$$f_2 : \quad \mathbf{z} \rightarrow \mathbf{y} \tag{4}$$

$$f : f_1 \circ f_2 = f_2(f_1(\mathbf{x})) \tag{5}$$

where the $\circ$ symbol denotes the composition of two functions, f_1 is the *encoder* that transforms the input from the input space to the latent space, and it is commonly referred to as feature extractor, and f_2 is the *classifier* that performs the classification based on the values of the latent representation of $\mathbf{x}$.

We assume that the network f is a black-box, i.e., it does not provide natively any way to inspect its decision process.

3.1.2 Content-based attention

Content-based attention has been introduced by [23], which named it *content-based addressing*, and it is defined as the weighted softmax ϕ_1 over the cosine similarity between a vector and a matrix. Formally, given the function D that computes the cosine similarity, a vector $\mathbf{k}$ of dimension d_1, and a matrix $\mathbf{M}$ of dimensions $d_2 \times d_1$, the content-based attention mechanism associates a score to each row r of the matrix using the following equation:

$$\begin{aligned} \mathbf{C}(\mathbf{M}, \mathbf{k}, \beta)[r] &= \phi_1(D(\mathbf{k}, \mathbf{M}), \beta) \\ &= \frac{exp(D(\mathbf{k}, \mathbf{M}[r, \cdot])\beta}{\sum_s exp(D(\mathbf{k}, \mathbf{M}[s, \cdot])\beta} \end{aligned} \tag{6}$$

where $\beta \in [0, 1]$ is a learned parameter that weighs the given module's importance in the context of multiple attention heads.

3.1.3 Sparsemax

Content-based attention, like most attention mechanisms, is based on the usage of softmax to compute the relevance of each input. Recent works highlight that using sparse functions in place of the softmax may improve performance and interpretability of attention modules [52, 54]. These functions assign zero probability to irrelevant input tokens, mitigating the problem of input dispersion [83].

Among the solutions proposed in literature, for flexibility reasons, we focus on the algorithm proposed by [13]. This approach, which is based on bisection methods [7, 48], finds the probability distribution that satisfies the following equation:

$$\phi_2(x_j) = \underset{\mathbf{p} \in \Delta^{n-1}}{\arg\min} \ \mathbf{p}^T \mathbf{x} + \mathbf{H}_\alpha^t(\mathbf{p}) \tag{7}$$

where $\Delta^{n-1}$is the probability simplex, α is a hyperparameter that controls the smoothness of the function, and $\mathbf{H}_\alpha^T$ is the Tsallis entropy [75], as described in (8).

$$\mathbf{H}_\alpha^t(\mathbf{p}) = \begin{cases} \frac{1}{\alpha(\alpha-1)} \sum_j (p_j - p_j^\alpha) & \alpha \neq 1 \\ -\sum_j p_j \log p_j & \alpha = 1 \end{cases} \tag{8}$$

By combining (8) and (7) we obtain the objective optimized by the softmax function and the resulting probability distribution does not contain zero values. Increasing the α leads to an increment of the sparseness until the maximum value of $\alpha = 2$, which corresponds to the sparsemax function.

To compute weights for $\alpha \neq 2$, we can use the following equation, which gives us the solution to the system:

$$\phi_2(x_j) = ReLU([(\alpha-1)\mathbf{x} - \tau\mathbf{1}]^{\frac{1}{\alpha-1}} \tag{9}$$

where τ is the Lagrange multiplier corresponding to the $\sum_i p_i = 1$ constraint. For further details about the algorithm and the proof of the derivation, please refer to the work of [13].

3.2 Proposed module

The goal of the proposed module is to make the decision process of the network f more transparent and improve its performance in small data settings. To achieve this goal, we design Memory Wrap to be a self-interpretable module that replaces the classifier f_2 in the black-box f. During the training process, Memory Wrap learns a function f_{MW} that takes as input the latent representations of two vectors and computes the prediction y_i. The input is composed of the latent representation of the current network input $\mathbf{x}_i$ and a set of latent representations of m samples randomly extracted from the training samples $S_i = \{\mathbf{x}^i_{m_1}, \mathbf{x}^i_{m_2}, .., \mathbf{x}^i_{m_m}\}$ called *memory set*, which act as memories of the training process.

$$y_i = f_{MW}(f_1(\mathbf{x}), f_1(\mathbf{S})) \tag{10}$$

The modified deep neural network becomes:

$$f : f_1 \circ f_{MW} = f_{MW}(f_1(\mathbf{x}), f_1(\mathbf{S})) \tag{11}$$

Since f_{MW} takes as input the latent representations produced by the encoder, the structure of the latter has an impact on the performance, so we expect that a better encoder architecture could improve further the performance of the module.

Memory Wrap uses a sparse version of the content-based attention mechanism and a classifier to combine the input representation and the memory set.

Our **sparse content-based attention** replaces the softmax ϕ_1 with the sparsemax function ϕ_2 [53], obtaining:

$$\begin{aligned} \mathbf{SC}(\mathbf{M}, \mathbf{k}, \beta)[r] &= \phi_2(D(\mathbf{k}, \mathbf{M}), \beta) \\ &= ReLU([(D(\mathbf{k}, \mathbf{M}) - \tau\mathbf{1}] \end{aligned} \tag{12}$$

where we set $\alpha = 2$ to use the sparsemax function and remove the β parameter (or equivalently set it to 1), since Memory Wrap includes just one attention module.

Now we will describe the entire workflow of the network f (Fig. 2). First, the encoder $f_1(x)$ encodes both the input and the memory set, projecting them in the latent space:

$$\mathbf{e}_{x_i} = f_1(\mathbf{x}_i) \tag{13}$$

$$\mathbf{M}_{S_i} = \{\mathbf{m}^i_1, \ldots, \mathbf{m}^i_m\} = \{f(\mathbf{x}^i_{m_1}), \ldots, f(\mathbf{x}^i_{m_m})\} \tag{14}$$

where $\mathbf{e}_{x_i}$ is the encoder representation of the input $\mathbf{x}_i$, and $\mathbf{M}_{S_i}$ is a matrix $\in \mathbb{R}^{m \times p}$ containing m samples, each of which of dimension p that depends on the output dimensions of the encoder. Then, these representations are fed to the sparse content-based attention module. The attention mechanism allows Memory Wrap to compute a score for each sample $\mathbf{m}^i_k$ in the memory set using the equation:

$$\mathbf{w} = ReLU([(D(\mathbf{e}_{\mathbf{x_i}}, \mathbf{M}_{\mathbf{s_i}})) - \tau\mathbf{1}] \tag{15}$$

At this point, similarly to [23], Memory Wrap computes the *memory vector* $\mathbf{v}_{S_i}$ as the weighted sum of memory set encodings, where the weights are the sparse content-based

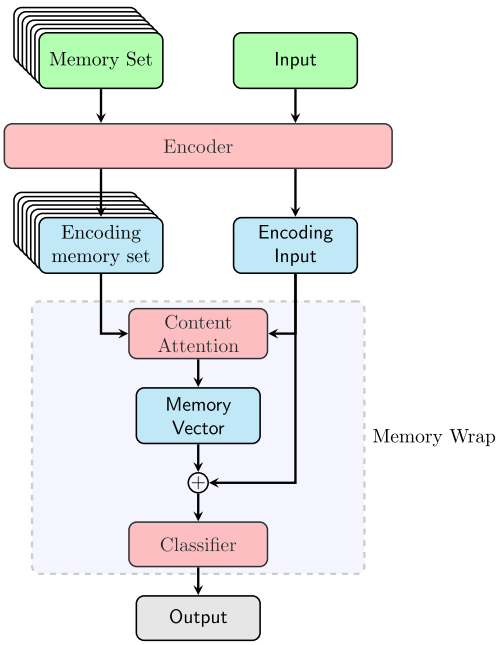

Fig. 2 Sketch of a deep neural network that includes Memory Wrap. The encoder maps the input and the memory set into the latent space. Then, Memory Wrap generates a memory vector based on the sparse content-based attention between them. Finally, the classifier takes both the memory vector and the input encoding, and it computes the output

attention weights computed before:

$$\mathbf{v}_{S_i} = \mathbf{M}_{S_i}^T \mathbf{w}. \tag{16}$$

Since the memory vector is computed using the sparsemax function, it includes information from a few memory samples. In this way, each sample contributes significantly, thus helping us to achieve output explainability. Conversely, the softmax would produce a vector representing all samples, but their contributions are flatter, making the importance estimation harder.

Finally, the **classifier** clf takes the concatenation of the memory vector and the encoded input, and computes the final output:

$$o_i = g(\mathbf{x}_i) = clf([\mathbf{e}_{x_i}, \mathbf{v}_{S_i}]). \tag{17}$$

In our case, we use a multi-layer perceptron with one hidden layer containing a number of units obtained by doubling the dimension of the input (Section 5.2). The role of the classifier is to exploit the memory vector to enrich the input encoding, using the *additional* features extracted from similar samples, possibly missing on the current input. On average, considering the whole memory set and thanks to the cosine similarity, strong features of the target class will be more represented than features of other classes, helping the network in the decision process.

3.3 Getting explanations

We aim at two types of explanations: example-based explanations and counterfactuals. The idea is to exploit the memory vector and content attention weights to extract explanations about model outputs, in a similar way to [38]. To understand how, let's consider the current input $\mathbf{x}_i$, the current prediction $f(\mathbf{x}_i)$, and the encoding matrix $\mathbf{M}_{S_i}$ of the memory set, where each $\mathbf{m}_j^i \in \mathbf{M}_{S_i}$ is associated with a weight w_j.

We can split the matrix $\mathbf{M}_{S_i}$ into three disjoint sets:

$$\mathbf{M}_{S_i} = M_e \cup M_c \cup M_z \tag{18}$$

where: $M_e = \{f_1(\mathbf{x}_{m_j}^i) \mid f(\mathbf{x}_i) = f(\mathbf{x}_{m_j}^i)\}$ contains encodings of samples predicted in the same class $f(\mathbf{x}_i)$ by the network and associated with a weight $w_j > 0$; $M_c = \{f_1(\mathbf{x}_{m_j}^i) \mid f(\mathbf{x}_i) \neq f(\mathbf{x}_{m_j}^i)\}$ contains encodings of samples predicted in a different class and associated with a weight $w_j > 0$; and M_z contains all the other samples, which are associated with a weight $w_j = 0$ (Fig. 3).

Note that M_z does not contribute at all to the decision process, and it cannot be considered for explainability purposes. Conversely, since M_e and M_c have positive weights, they can be used to extract example-based explanations and counterfactuals. Let's consider the sample $\mathbf{x}_{m_j}^i \in \mathbf{M}_{S_i}$ associated with the highest weight. A high weight of w_j means that the encoding of the input $\mathbf{x}_i$ and the encoding of the sample $\mathbf{x}_{m_j}^i$ are similar. If $\mathbf{x}_{m_j}^i \in M_e$, then it can be considered as a good candidate for an example-based explanation because it is an instance highly similar to the input and predicted in the same class, as defined in Section 2.2. Conversely, if $\mathbf{x}_{m_j}^i \in M_c$, then it could be considered as a counterfactual, because it is highly similar to the input but predicted in a different class.

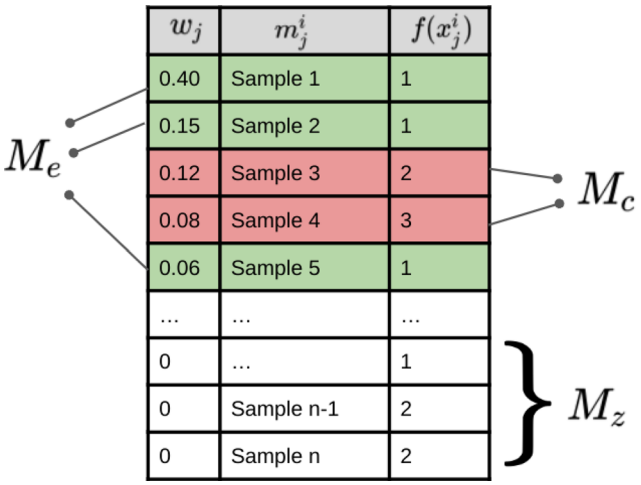

Fig. 3 An illustration to highlight how to interpret the samples included in the memory. Assuming that the current input is predicted as a member of class 1, we can distinguish between: the set M_e of candidates for example-based explanations as the samples predicted in the same class and associated with a weight greater than zero (green), the set M_c of candidates for counterfactuals as the samples predicted in a different class and associated with a weight greater than zero (red), and the set M_z of samples that have no impact on the decision process (white)

The key observation is that, since it has the highest weight, it will be heavily represented in the memory vector that will **actively** contribute to the inference, being used as input for the last layer. This means that common features between the input and the sample $\mathbf{x}_{m_k}^i$ are highly represented, and so they constitute a good example-based explanation. Moreover, because $\mathbf{x}_{m_k}^i$ is partially included in the memory vector, if it is a counterfactual, it is likely that it will be the second or third predicted class, giving also information about "doubts" of the neural network. In the next sections, we show how the identification and analysis of these important samples help us to diagnose and interpret the model behavior and its decision process.

4 Results

This section first describes the experimental setup, then it presents and analyzes the obtained performances, and finally, it shows how it is possible to interpret the decision process based on the memory samples used by Memory Wrap.

4.1 Setup

We train from scratch several deep neural networks and compare their performance with and without our proposed module on subsets of three popular datasets, Street View House Number (SVHN) [58], CINIC10 [15] and CIFAR10 [36]. We apply the protocol described in the Algorithm 1, training each network on a subset of each dataset.

Algorithm 1 Experimental protocol.

```
Require: datasets, networks        ▷ list of datasets and deep networks
Ensure: results                    ▷ variable to collect results
num_samples ← [1000,2000,5000]
results ← []
for model in networks do:
    for data in datasets do:
        for samples in num_samples do:
            accuracies ← []
            for run in range(1,15) do:
                reduced_dataset ← extract(data[train], samples)
                f←train(model, reduced_dataset)
                accuracy←eval(f, data[test])
                accuracies.append(accuracy)
            end for
            results.append([f, d, samples, accuracies])
        end for
    end for
end for
return results
```

In each experiment, we randomly split the training set to extract smaller datasets in the ranges {1000, 2000, 5000}. These sets correspond to the 1%, 2% and 5% of the labeled samples in CINIC10 and 2%, 4% and 10% of the labeled samples in the other datasets, thus simulating small data settings. Then, we train from scratch the chosen configuration of the deep neural network using the extracted subset, and we evaluate its performance on the test dataset. We consider the mean and the standard deviation of the accuracy over 15 experiments as the final result for each pair model-subset.

We test Memory Wrap on ResNet18 [26], EfficientNetB0 [74], MobileNet-v2 [64], GoogLeNet [72], DenseNet [28], ShuffleNet [88], WideResnet 28x10 [87], and ViT [18]. These networks are commonly used as backbone networks on the considered datasets, and their variants are among the top performers on the considered settings of training from scratch without extra data and prior knowledge[1].

[1] https://paperswithcode.com

The implementation of the networks relies upon the repositories by Kuang Liu[2], Oscar Knagg[3] and Omiita[4]. Memory Wrap can be installed as a Python package at https://pypi.org/project/memorywrap/.

Training procedure To train the models, we follow different training procedures based on the repository they belong to and the training procedure suggested by the authors of the papers. Specifically, WideResnet [87] is trained using the procedure explained in the reference paper, ViT [18] is trained for 200 epochs following the training procedure employed in the repository to which it belongs, while for all the other models, we use the official setup for CINIC10, and the settings of Huang et al. [28] for SVHN and CIFAR10. Hence, we train these models for 40 epochs in SVHN and 300 epochs in CIFAR10. In both cases, we apply the Stochastic Gradient Descent (SGD) algorithm, starting from a learning rate of 1e-1 and decreasing it by a factor of 10 after 50% and 75% of epochs. The images are normalized and, in CIFAR10 and CINIC10, we also apply an augmentation based on random horizontal flips. We do not use the random crop augmentation to avoid isolating a portion of the image containing only the background. In these cases, the memory retrieves similar examples based only on the background, thus pushing the network to learn useless shortcuts. Indeed, in preliminary experiments, we find that these biased shortcuts improve the performance of Memory Wrap on the lowest settings but nullify its impact in some configurations where the training dataset is larger and the effect of augmentation is more extensive. We acknowledge that this configuration is neither optimal for baselines nor for Memory Wrap and we can reach higher performance in both cases by choosing another set of hyperparameters tuned in each setting. However, this setup makes the comparison across different models and datasets quite fair.

4.2 Performance comparison

We start our investigation by comparing Memory Wrap against two groups of methods: memory-based neural modules and algorithms with a similar decision process in terms of interpretation of their results (i.e., variants of K-NN and an ablated version of Memory Wrap).

Using ResNet18, MobileNet, and EfficientNet as backbones, we perform these first tests on the SVHN dataset. The goal is to understand the advantages of Memory Wrap and select the best methods to be used in the next set of experiments.

[2] https://github.com/kuangliu/pytorch-cifar

[3] https://github.com/oscarknagg/few-shot

[4] https://github.com/omihub777/ViT-CIFAR

4.2.1 Memory-based modules

The first group includes Prototypical Networks [67] and Matching Networks [78], which are two popular baselines that, like Memory Wrap, replace the last layer with a memory module. They can be applied to any network and have an associated code available to the public. We also include in the comparison the models themselves without any additional layer (*std*).

Standard (Std) This baseline is the deep neural network without the replacement of the last layer with Memory Wrap. It is trained in the same manner as the network with Memory Wrap.

Prototypical networks This is the module proposed by [67], where we replace their shallow network with our backbone models. In this case, the network computes the prototypes for each class by taking the mean of the samples in the memory set, and then it will use the distance between them and the current input to compute the prediction.

Matching networks This is an adapted version of [78], where we replace the encoder with the networks considered in our work. It enriches the input embedding by using an LSTM network and it performs the classification based on a weighted linear combination of the labels in the memory set, using the distance between its samples and the current input as weights.

Note that the memory sets used in all the considered methods contain samples randomly extracted from the reduced training dataset. This prevents the network from accessing additional resources during the training process, thus violating the small data settings and making the comparison unfair.

Performance Figure 4 compares the performance reached by the methods when trained using 1000, 2000 and 5000 training samples respectively. The results allow us to understand the differences when using the memory set in different ways.

We can observe that Matching Networks reach the lowest performance on all the configurations and Prototypical Networks outperform them, confirming the results in the work of [67]. However, the performance of Prototypical Networks is still lower than the standard models and Memory Wrap. We can explain the results by analyzing the complexity of these approaches and the impact of each sample of the memory set. Matching Networks adopt the most complex type of encoding, since they use LSTMs to encode both the input and the memory set. Hence, the power of the approach depends on the goodness of the encoding of the LSTMs, which in turn depends on the quality and amount of training data. While this is not a problem in few-shot learning settings, where LSTMs are trained using the full dataset from a fixed set of classes, this is less effective in the scenario of small data. The low amount of training data can produce unstable or misaligned encoding, thus

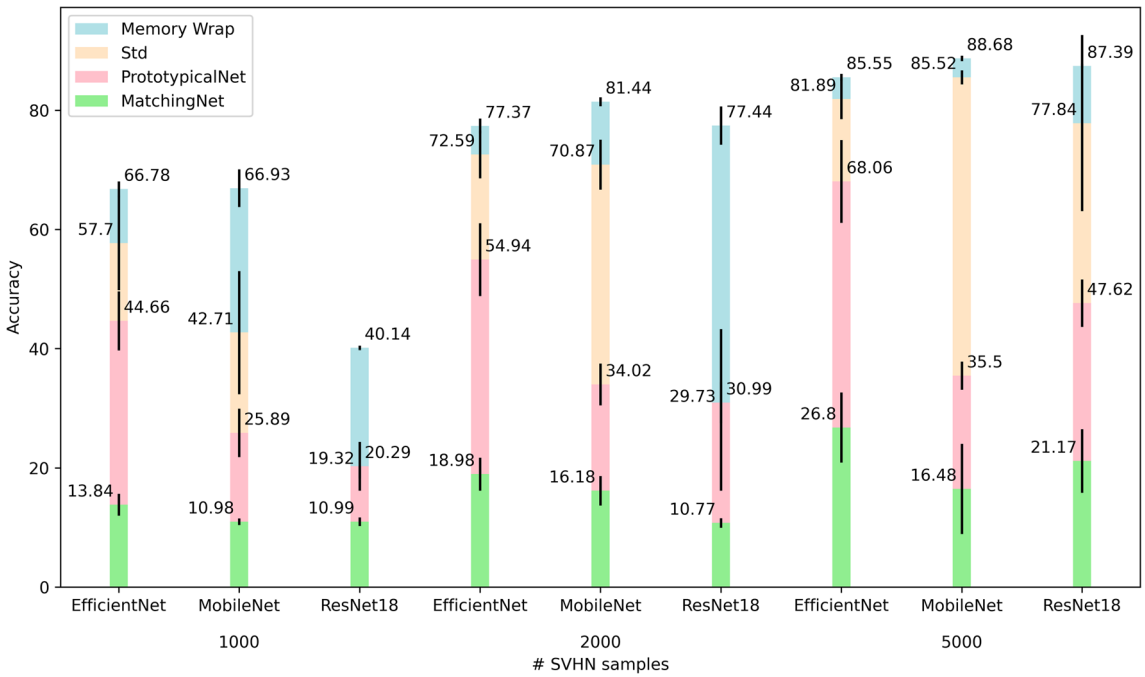

Fig. 4 A comparison between different ways of using a memory set to aid the inference process on the SVHN dataset. We compare the baseline models (std) against Matching Networks (MatchingNet), Prototypical Networks (PrototypicalNet) and Memory Wrap

impacting the performance of the model. Additionally, by design, all the samples in the memory set influence the encoding of both the set and the input. This means that the wrong encoding of a few samples has a huge impact on the behavior of the model itself. Prototypical Networks mitigate the first problem and outperform Matching Networks by encoding the input and the memory samples independently, using the backbone architecture. However, they are still affected by the second problem, since they compute the prototypes as an average of the samples in the memory set. This means that they depend on the quality of the memory set, and they have difficulties when dealing with outliers, since they can have atypical encoding. Moreover, when the number of classes is greater than a few, the encoding of the prototypes can be close, exacerbating the problem.

Memory Wrap outperforms all the others, reaching a data efficiency of 1.5x for EfficientNet and MobilNet and between 2x and 2.5x for ResNet. It achieves these results by adopting the encoding strategy of Prototypical Networks and mitigating the problems connected to the computation of prototypes. Indeed, it adaptively selects a subset of the samples in the memory set, ignoring misaligned points, and completely removes the problem of the impact of wrong encoding. Moreover, when the input is an outlier, a common scenario in small data settings, the models compare it to, and use information from, similar outliers rather than comparing it with an aggregated average, potentially distant from its encoding. Finally, since Memory Wrap does not use labels and can choose by itself which samples to focus on, the number of samples (and potentially of classes) has a lower impact on the performance. Another important feature is that, while Memory Wrap provides some possibilities to analyze its decision process (Section 4.3), the alternatives are close to black-boxes, especially when using LSTM to enrich the encoding, thus making the interpretation of their behavior hard.

4.2.2 Interpretable methods

The second group includes algorithms that perform the classification similarly to Memory Wrap, and they are comparable in terms of explanations that one can extract. In the same settings of the previous section, we compare Memory Wrap against: K-NN; a version of K-NN that uses the sparsemax function; and an ablated version of Memory Wrap. **K-NN** We consider the predictions obtained by applying the K-NN algorithm on the latent space of the standard baseline [17]. We pick 100 random samples at each iteration from the current training dataset, and we compute the distance between these and the current input. We select the predictions based on the mode of the top-k nearest neighbors. This baseline corresponds to the *baseline classifier* used in [78].

Major voting This baseline works like K-NN but replaces the K-NN algorithm with a sparsemax over the cosine distance. The predictions are chosen based on the mode of the samples that have the resulting weights greater than zero. The starting models are the standard baselines, like in the K-NN baseline.

Only Memory (OnlyMem) This is an ablated variant of Memory Wrap that uses the memory vector alone as the input to the classifier by removing the concatenation with the encoded input. Therefore, the output is given by

$$o_i = g(\mathbf{x}_i) = clf(\mathbf{v}_{S_i}) \tag{19}$$

In this case, the input is used only to compute the sparse content-based attention weights, which are then used to build the memory vector, and the network learns to predict the correct answer based on it. Because of the randomness of the memory set and the absence of the encoded input image as input of the last layer, the network is encouraged to learn more general patterns and not exploit the given image's specific features.

Performance Interestingly, the baseline of K-NN is quite competitive on several configurations, getting a more interpretable classification while lowering the performance of standard models by 1-2% of accuracy (Fig. 5). This is in line with the excellent results of this baseline reported across many tasks [78]. We can observe that the best performing value of the hyperparameter k changes model by model and configuration by configuration, thus making its tuning difficult. The Major Voting algorithm avoids the problem by using the sparsemax, which dynamically chooses the k value at inference time. However, its performance is nearly the same as the vanilla K-NN or even worse, likely due to the noise introduced by the additional samples included by the sparsemax function.

Memory Wrap and the variant OnlyMem combine the positive aspects of these baselines: they exploit similar examples to perform classification, thus making the decision process of the underlying networks more interpretable, and they use the sparsemax to dynamically choose the samples to be used for each input. The crucial difference is that they directly use the memory set and the similarity with the input during the optimization process to improve the learning process. In this way, the module allows the underlying model to learn how to exploit the neighbors' information. While both variants reach very good results, the additional information carried on by the encoder in Memory Wrap seems crucial to achieve the best possible performance, especially when dealing with the lowest number of samples, likely due to the additional shortcuts accessible only from the given input (e.g., the combination of rare features). Before analyzing the properties of Memory

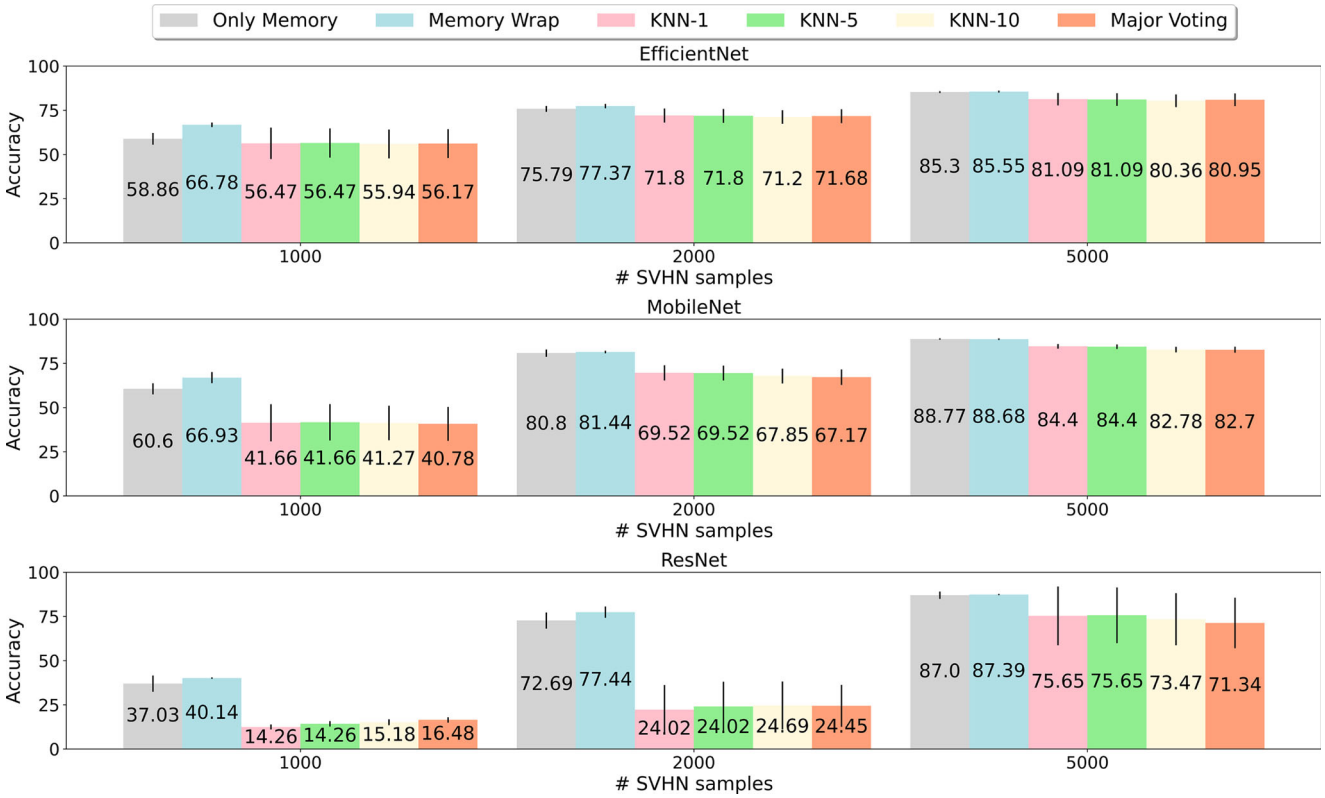

Fig. 5 A comparison between different algorithms with a similar interpretable decision process on the SVHN dataset. We compare K-NN using {1, 5, 10} as k value, a modified K-NN that uses a sparsemax (Major Voting), Memory Wrap and its ablated version (OnlyMem)

Wrap and the reasons behind the obtained results, we select the top performers across the tests, namely the standard baseline and the two variants of Memory Wrap, and show the results for the rest of the settings.

4.2.3 Complete experiments

First, we extend the test on SVHN to GoogLeNet [72], DenseNet [28], ViT [18], WideResnet [87], and ShuffleNet [88] (Fig. 6). Then, we report in Figs. 7 and 8 the results using all the encoders in CIFAR10 and CINIC10. Analyzing these results, firstly, we can observe that the amount of gain in performance depends on the underlying deep network: MobileNet shows the largest gap in all the datasets, while ViT shows the smallest one. Secondly, results depend on the dataset since the gains in each SVHN configuration are always more significant than the ones in CIFAR10 and CINIC10. We think that this depends on the structure of the dataset itself. Several features are in common between different classes in CINIC10 and CIFAR10 (e.g., material, color, etc.), and they are themselves common features in images. Therefore, for the memory module, it is harder to exploit them to distinguish between classes. Conversely, in SVHN, the intra-class variance is lower, and the differences between classes can be more easily exploited using the similarity with samples in the memory set.

Moreover, we can observe that adding Memory Wrap to a deep neural network reduces its variance of performance across different runs, making the learning process more stable. Regarding the ablated version Only Memory, it outperforms the standard baseline, reaching nearly the same performance as Memory Wrap in most settings. However, its performance appears less stable across configurations. In fact, they are lower than Memory Wrap in some SVHN and CINIC10 settings, lower than standard models in some configurations of DenseNet and ResNet, and sometimes the fail (e.g., ViT on SVHN). These results confirm our hypothesis that the additional information captured by the input encoding allows the model to exploit other shortcuts and to reach the best performance. Moreover, even though it uses only the memory set to compute the prediction, its interpretability is comparable to Memory Wrap (Appendix A).

4.3 Interpreting the memory wrap behavior

Now we are ready to discuss how Memory Wrap's structure helps us interpret its predictions. In this section, we consider MobileNet-v2 as our base network for simplicity, but the results are similar for all the considered models and

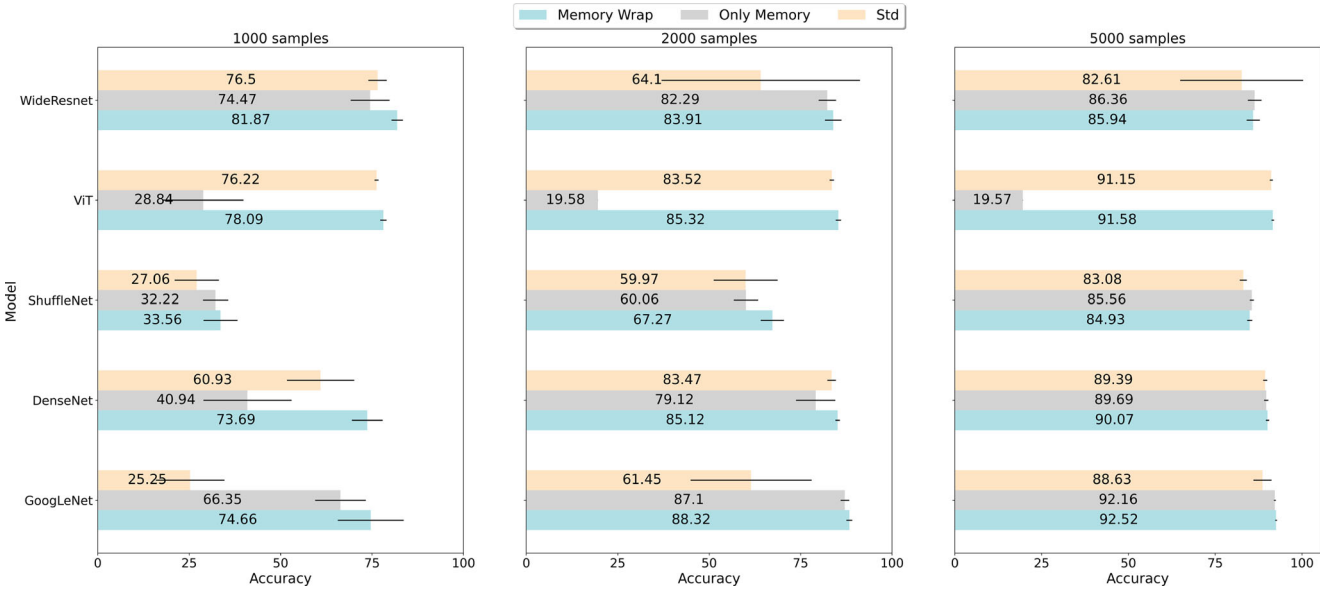

Fig. 6 Avg. accuracy and standard deviation over 15 runs of the standard model and two variants of Memory Wrap, when the training dataset is a subset of SVHN. For each configuration, we highlight in bold the best result and results that are within its margin

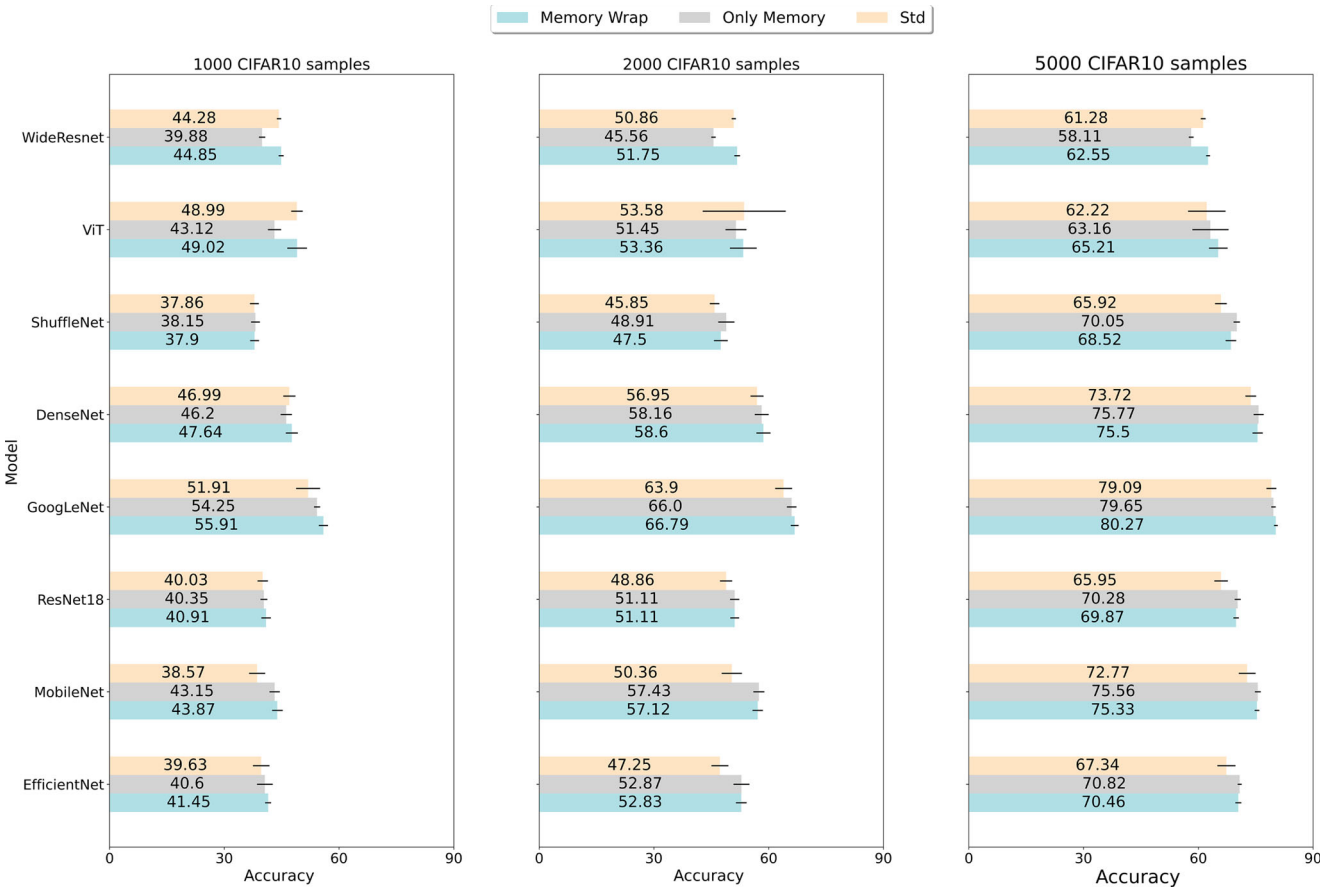

Fig. 7 Avg. accuracy and standard deviation over 15 runs of the standard model and two variants of Memory Wrap, when the training dataset is a subset of CIFAR10. For each configuration, we highlight in bold the best result and results that are within its margin

Fig. 8 Avg. accuracy and standard deviation over 15 runs of the standard model and two variants of Memory Wrap, when the training dataset is a subset of CINIC10. For each configuration, we highlight in bold the best result and results that are within its margin

configurations. The first step is to check which samples in the memory set have positive weights – the set $M_c \cup M_e$. Figure 9 shows this set sorted by the magnitude of sparse content-based attention weights for six different inputs of the three datasets. Each couple shares the same memory set as an additional input, but each set of used samples – those associated with a positive weight – is different. In particular, consider Fig. 9a, where the only difference among images

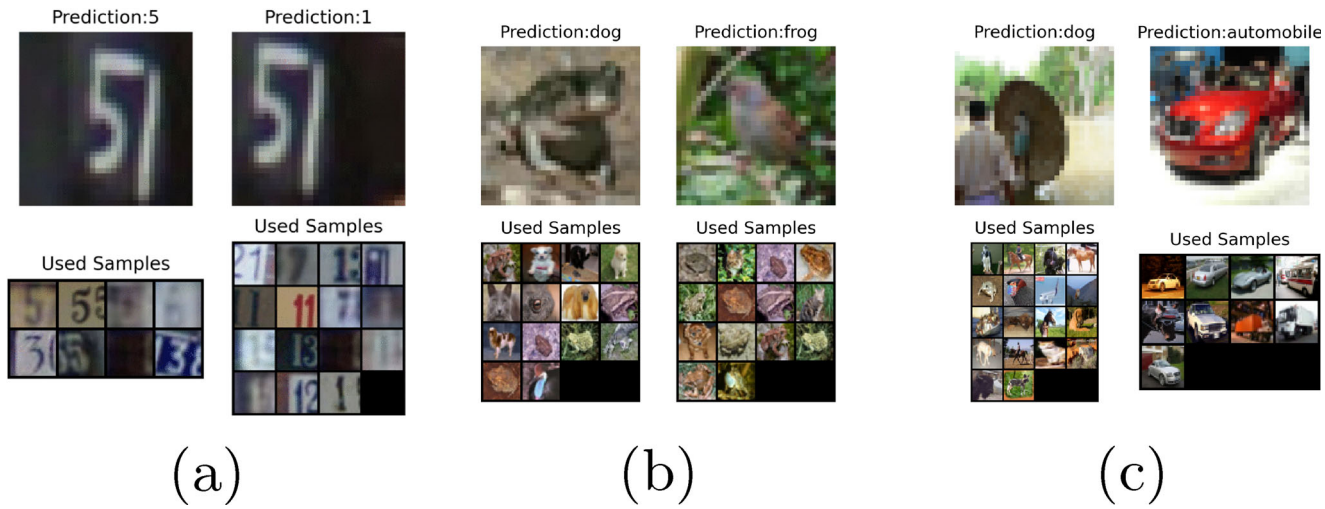

Fig. 9 Inputs (first rows) from SVHN (a), CIFAR10 (b), and CINIC10 (c), their associated predictions and an overview of the samples in the memory set that have an active influence on the decision process – i.e. the samples on which the memory vector is built – (second row)

is a lateral shift made to center the numbers. Despite their closeness in the input space, samples in memory are different: the first set contains images of "5" and "3", while the second set contains mainly images of "1" and a few images of "7". We can infer that the network probably focuses on the shape of the number in the center to classify the image, ignoring colors and the surrounding context. Conversely, in Fig. 9b the top samples in memory are images with similar colors and different shapes, thus telling us that the network is wrongly focusing on the association between background colors and the object color. These examples show that just the inspection of samples in the set $M_{ce} = M_c \cup M_e$ can give us some insights into the decision process. Finally, Fig. 9c gives us a hint of how the model separates the classes: the samples used to predict the image as an automobile include images of trucks, thus suggesting that these two classes are close in the representation space of the model.

Once we have defined the nature of the samples in the memory set that influence the inference process, we have to verify whether the sparse content-based attention weights ranking is meaningful for Memory Wrap predictions. To measure the reliability, we set the *prediction matching accuracy* as a measure that checks how many times the prediction obtained using as input the sample in the memory set matches the prediction associated with the current image. Intuitively, if a sample $\mathbf{x}^i_{m_k}$ influences significantly the decision process and if it can be considered as a good proxy for the current prediction $g(\mathbf{x}_i)$ (i.e a good example-based explanation), then $g(\mathbf{x}^i_{m_k})$ should be equal to $g(\mathbf{x}_i)$.

In Table 1, we compare the prediction matching accuracy reached by using as input the sample $\in M_{ce}$ with the highest weight ($Top_{M_{ce}}$), the sample $\in M_{ce}$ with the lowest weight ($Bottom_{M_{ce}}$), or a random sample. Additionally, we compute the prediction matching accuracy for the standard baseline, applying a similar mechanism. We extract 100 random samples from the training set, compute the cosine distance in the latent space, apply a sparsemax over the distances, and extract the samples with the highest and the lowest weight.

We observe that, in the networks with Memory Wrap, the sample with the highest weight reaches high accuracy, always greater than both the random selection and the sample with the lowest weight. As a result, the sparse content-based attention weights ranking is reliable and extracts good proxies for the predictions. The results show that the prediction matching accuracy of the bottom example increases when the model improves its performance.

Table 1 Avg. prediction matching accuracy and standard deviation comparison over 15 runs between the sample in the memory set with the highest sparse content-based attention weight ($\text{Top}_{M_{ce}}$), the example with the lowest weight but greater than zero ($\text{Bottom}_{M_{ce}}$) and a random sample (Random) for both MobileNet with Memory Wrap and without it (Standard)

Prediction matching accuracy %				
Model	Example	1000	2000	5000
SVHN				
Memory Wrap	$\text{Top}_{M_{ce}}$	**84.24 ± 1.22**	**90.59 ± 0.52**	**94.47 ± 0.22**
Standard	$\text{Top}_{M_{ce}}$	75.34 ± 3.77	84.45 ± 2.15	92.28 ± 1.07
Memory Wrap	$\text{Bottom}_{M_{ce}}$	**46.46 ± 1.77**	**57.39 ± 1.09**	**69.94 ± 1.37**
Standard	$\text{Bottom}_{M_{ce}}$	31.77 ± 4.89	44.00 ± 4.34	62.43 ± 3.36
	Random	11.76 ± 0.30	11.66 ± 0.17	11.71 ± 0.13
CIFAR10				
Memory Wrap	$\text{Top}_{M_{ce}}$	**82.04 ± 1.14**	**87.75 ± 0.72**	**91.76 ± 0.22**
Standard	$\text{Top}_{M_{ce}}$	72.39 ± 1.64	77.65 ± 1.76	85.56 ± 1.63
Memory Wrap	$\text{Bottom}_{M_{ce}}$	**46.01 ± 1.92**	**60.10 ± 1.29**	**69.94 ± 0.82**
Standard	$\text{Bottom}_{M_{ce}}$	32.42 ± 1.81	40.21 ± 2.77	52.61 ± 3.99
	Random	10.22 ± 0.28	10.23 ± 0.20	9.80 ± 0.43
CINIC10				
Memory Wrap	$\text{Top}_{M_{ce}}$	**76.31 ± 0.73**	**78.50 ± 0.50**	**78.45 ± 0.62**
Standard	$\text{Top}_{M_{ce}}$	69.75 ± 1.65	75.51 ± 1.27	74.11 ± 1.04
Memory Wrap	$\text{Bottom}_{M_{ce}}$	**37.01 ± 1.12**	**41.34 ± 0.73**	**37.55 ± 1.49**
Standard	$\text{Bottom}_{M_{ce}}$	29.66 ± 1.17	36.87 ± 1.45	30.73 ± 1.58
	Random	10.47 ± 0.19	10.30 ± 0.11	10.16 ± 0.12

The bold entries represents the results of the model that reaches the highest accuracy

This finding is consistent with the results of Table 3, and it is motivated by the fact that when Memory Wrap improves its performance, it learns to select more and more often samples of only the same class and, as a consequence, the set M_{ce} includes more often only them, hence the increment of the prediction matching accuracy of the sample with the lowest weight. Finally, Table 1 also shows that the model including Memory Wrap outperforms the standard baseline in all the datasets in terms of predictions matching accuracy, thus suggesting that including the memory set in the training process and encouraging the model to exploit is beneficial to obtain more precise example-based explanations.

4.3.1 Example-based explanations

In this section, we examine the samples in the set M_e, which are associated with a weight greater than zero and predicted in the same class of the input. We show a usage scenario on how to exploit them, and then we analyze their quality when used as example-based explanations by comparing them with post-hoc methods.

Usage scenario: bias detection In this test, we investigate whether it is possible to detect when the model is biased by exploiting the interpretability of the module. Therefore, we train EfficientNet augmented with Memory Wrap on a biased version of MNIST [40]. This dataset [2] includes a bias that correlates the background of images with their labels. We fix the correlation to 1 in the training set, introducing a one-to-one mapping between classes and background color. In this way, the model can achieve high accuracy by exploiting the bias, and it has little motivation to learn additional features of the images. At testing time, we remove the correlation, randomly selecting the background color for each image. Our goal is to train the model in a such way that its predictions are biased and show how we can use the memory set to detect the bias. Figure 10 shows how we can visually detect that the model is highly biased in a fast way: the samples in the set M_e used to perform the classification have the same background color as the input image, although the digit in the middle is different. This analysis tells us that the background is the main feature used by the model to classify the digit.

Quality estimation Here, we estimate the quality of the sample with the highest weight in the set M_e when used as an example-based explanation by comparing it to the explanations extracted by post-hoc methods. To measure the quality, we introduce and use the *input non-representativeness* and the *prediction non-representativeness* metrics. The first is the objective implicitly optimized by [32], and it is the L_1 loss between the logits of the current prediction and the logits of the selected explanation. Intuitively, a low score means that the model acts similarly in both cases, returning similar output distributions. Conversely, the prediction non-representativeness measures the cross entropy loss between the logits of the selected explanation and the predicted class used as the target class. This metric is equivalent to the non-representativeness metric proposed by the contemporary pre-print of [59], but considering a set of one explanation. We compare the best example-based explanation selected among the samples in the memory by two post-hoc methods, namely the CHP method [32] and the KNN* against which they compare (i.e., inspired by the pre-print written by [61]), a random selection among the samples associated with the same prediction of the current input, and the sample $x_i \in M_e$ associated to the highest weight.

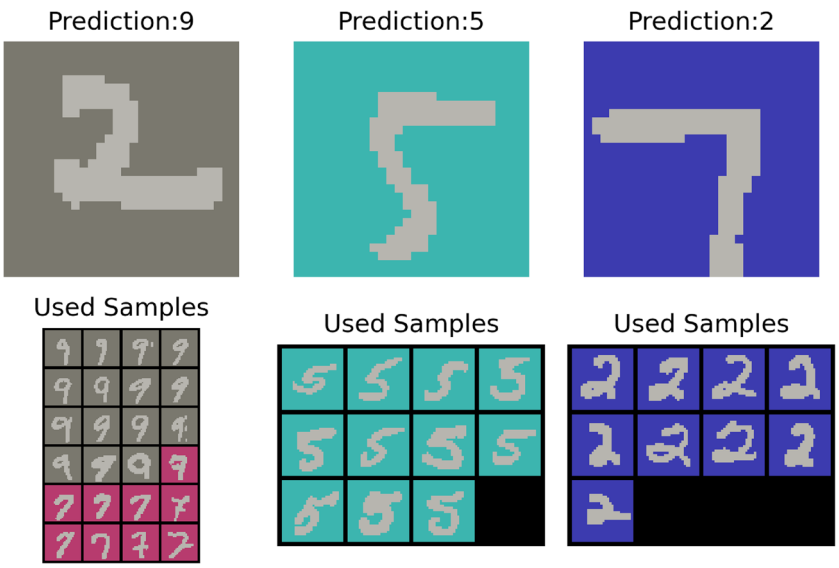

Fig. 10 An example of a model biased towards the background. The figure shows how the inspection of the memory samples used by the network makes clear the reasons behind the bad performance of the network

Figure 11 shows that, in terms of input non-representativeness, the post-hoc methods are the best choices to select the best sample. This is motivated by the fact that these methods are optimized to minimize this score. Memory Wrap achieves worse performance, but they are still significantly better than the random baseline. Conversely, Fig. 12 shows that the samples selected by Memory Wrap are slightly better than all the others in terms of class prediction non-representativeness and, surprisingly, sometimes the random baseline is better than the post-hoc methods. To understand the results, we can analyze the behavior of different methods on inputs where the model is uncertain and the logits among the classes are close. In these cases, CHP and KNN* extract samples where the model is uncertain too, and, consequently, the input non-representativeness score is low. The random baseline picks a random sample predicted in the same class, and for this reason, it is more difficult for it to extract one sample where the model acts in the same way. At the same time, in these inputs, the prediction non-representativeness scores of CHP and KNN* are high since they tend to select samples where the model is uncertain.

Conversely, in Memory Wrap, the final prediction is a balance of the samples in M_e and the counterfactuals in the set M_c. The samples included in M_e are representative of the predicted class, and they shift the prediction towards it, while the samples included in M_c act in the opposite direction. Hence, considering only the sample with the highest weight in the former set, it will maximize the prediction representativeness while having difficulties representing the uncertainty, likely encoded by counterfactuals or example-based explanations with lower weights.

4.3.2 Counterfactuals

This section examines the samples in the set M_c, which are samples in the memory set associated with a weight greater than zero and predicted in a different class with respect to the input. We show an example of a usage scenario, and then we compare them with counterfactuals generated by a generative method [50]. Note that, since it is not guaranteed that Memory Wrap returns a counterfactual at each time, we consider only the cases when this is available in the comparison.

Usage scenario: understanding prediction reliability In this usage scenario, we want to know whether the model predictions are reliable. The idea is to use the presence of samples included in M_c to detect when the prediction could be potentially unreliable. Intuitively, a high number of counterfactuals in the set of activated memory samples can be a sign of uncertainty of the model and a chance that the model is predicting the wrong class. At the same time, the relative position on the rank, obtained by sorting the weights of these samples, can also encode information about their importance in the decision process. To verify whether this is the case, we compute the Pearson product-moment correlation coefficients [68] between the correctness of the prediction (i.e., one if it is correct, zero otherwise) and two variables: the ratio between the number of samples in the set M_c and in the set M_e, and the relative index of the first counterfactuals. Table 2 shows that there is a significant positive correlation between the position of the first counterfactuals and the correctness of the prediction, hence the grater is the index,

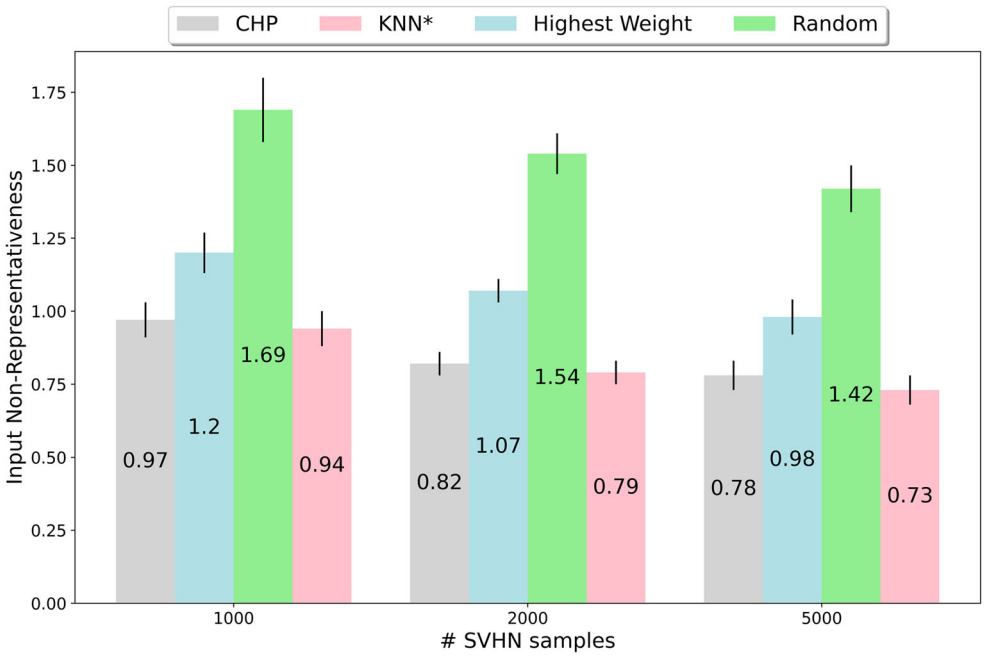

Fig. 11 A comparison in terms of input representativeness score between the CHP method (gray), KNN* (pink), the random baseline (green), and the sample with the highest weight in memory (blue). Lower is better

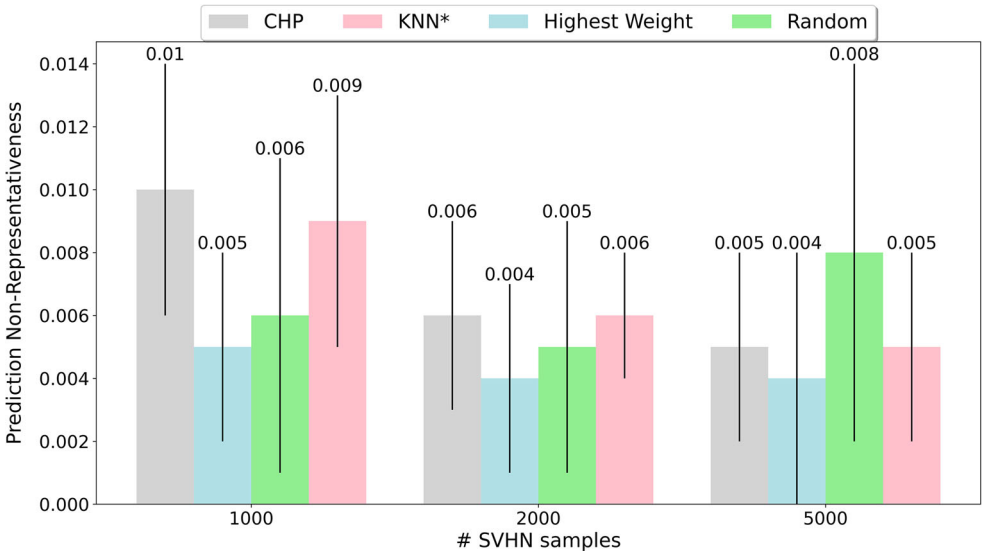

Fig. 12 A comparison in terms of prediction representativeness score between the CHP method (gray), KNN* (pink), the random baseline (green), and the sample with the highest weight in memory (blue). Lower is better

or equivalently the lower is its position in the rank, the more reliable is the prediction. Conversely, there is a negative correlation with the number of counterfactuals, hence the more counterfactuals in memory, the lower the prediction's reliability.

Once we have proved the correlation between the counterfactuals and the reliability of the predictions, we are ready to apply a similar test to the one used for the computation of prediction matching accuracy in Table 1. This time we consider two cases: when the sample with the highest weight is a counterfactual and when the memory set does not contain counterfactuals.

As shown in Table 3, when the sample with the highest weight is a counterfactual, then the model accuracy is much lower and its predictions are often wrong. Hence, one can use the presence of a counterfactual as the top contributor of the memory set to alert the user that the decision process could be unreliable. We also track their frequency (*coverage*), since, ideally, we want that these cases would be rare.

As in the case of prediction matching accuracy, we apply the test to both Memory Wrap and the standard baseline, configured as in the previous case. We can see (Table 3) that both the frequency and the accuracy are lower on the models augmented with Memory Wrap than the standard baseline, thus telling us that the procedure is more precise in alerting the user.

Conversely, when there are only example-based explanations in the memory set, the model is sure about its predictions and the accuracy is very high. We can observe that both the accuracy of models with Memory Wrap and the frequency of these cases are higher than the standard baseline, and thus the model is reliable in a higher number of cases and the procedure can detect them better.

Finally, note that, as expected, the number of cases where the model is detected as uncertain about its prediction decreases when we provide more examples in the training process, and at the same time, the number of cases where the model is sure increases.

Quality estimation As in the example-based explanations section, here, we estimate the quality of the counterfactuals selection based on the weights of Memory Wrap. We compare it to the counterfactuals generated by a recent method based on permutations [50] and guided by prototypes. Prototypes are computed as the mean of the samples of each class in the dataset. We start the comparison

Table 2 Avg. and standard deviation of the Pearson product-moment correlation coefficients between the correctness of the predictions and the number of counterfactuals (number) or the position of the first counterfactual (position)

Avg Pearson Coefficent			
Model	1000	2000	5000
Position	0.39 ± 0.02	0.40 ± 0.01	0.41 ± 0.01
Number	-0.42 ± 0.01	-0.45 ± 0.01	-0.46 ± 0.01

The score is computed as the mean over 15 runs using as encoder MobileNet-v2

Table 3 Accuracy reached by the model on SVHN when the sample with the highest weight in memory set is a counterfactual (Top Counter) and when there are no counterfactuals at all (No Counter)

Avg Accuracy % (Coverage%)				
Model	Counter	1000	2000	5000
Memory Wrap	Top	35.45 (19.32)	38.29 (12.06)	39.79 (7.56)
Standard	Top	28.00 (24.62)	40.95 (15.60)	47.81 (7.72)
Memory Wrap	No	95.20 (18.73)	97.00 (36.47)	97.84 (56.88)
Standard	No	86.80 (4.51)	95.51 (18.69)	97.42 (45.82)

We compare models with and without Memory Wrap (Standard). The accuracy is computed as the mean over 15 runs using as encoder MobileNet-v2

using the scores proposed by the same paper, namely the IM1 and IM2 scores [50].

The IM1 score measures the ratio between the reconstruction error of the counterfactuals using an autoencoder trained to recognize the samples in the input class and an autoencoder trained using only samples of the counterfactuals class. The lower the score, the more interpretable the counterfactual is. Table 4 shows that the scores are nearly the same for both approaches. However, we should consider that the score has been proposed for generative methods, which start from the input and shift towards the counterfactual class. In this case, it makes sense to reward a counterfactual closer to the counterfactual class rather than the input class.

Conversely, in our case, Memory Wrap already selects a sample predicted in a different class, and the score should reward samples closer to the input class. Hence, we propose the Inverted IM1 score (IIM1) that switches the positions of the numerator and denominator of the IM1 score. Therefore, the IIM1 score is given by:

$$\frac{\left\| x_{cf} - AE_i(x_{cf}) \right\|}{\left\| x_{cf} - AE_{cf}(x_{cf}) \right\|} \tag{20}$$

where x_{cf} is the counterfactual, AE_i is the autoencoder trained using only the samples of the input class of the current input x_i, and AE_{cf} is the autoencoder trained using only the samples of the counterfactual class. In this case (Table 4), the scores are a bit worse than IM1 scores of the post-hoc method, but this is understandable since Memory Wrap is not optimized to find the optimal counterfactual. Despite that, the difference is small, and counterfactuals returned by Memory Wrap can be considered as a fast approximation of good counterfactuals. Indeed, Memory Wrap needs less than two minutes to compute them for a dataset of 2000 samples, while a perturbation-based method, like the one considered, requires more than 400 minutes.

Table 4 Avg. IM1, IM2 and IIM1 scores comparison on SVHN dataset between the candidate counterfactuals associated with the highest weight and a post-hoc method based guided by prototypes

Counterfactuals Scores Avg.			
	Samples		
Score	Dataset	MemoryWrap	Proto [50]
IM1	1000	0.991	0.994
	2000	0.985	0.997
	5000	0.985	0.990
IM2(x10)	1000	1.703	2.623
	2000	1.674	2.707
	5000	1.698	2.770
IIM1	1000	1.014	–
	2000	1.015	–
	5000	1.013	–

The average is computed among the whole dataset of a random model for each configuration. Lower is better

Lastly, Table 4 also shows a comparison in terms of IM2 score. This score measures the similarity between the reconstructed counterfactual instances when using AE_{cf} and an autoencoder trained using all classes. A lower value means that the data distribution of the counterfactual class describes the counterfactual as well as the distribution over all classes. Thus, a low value implies that the counterfactual is interpretable. The scores of Memory Wrap are a lot better than the generative method, and the situation does not change if we compute it using AE_i in place of AE_{cf}. The motivation behind the large margin is that the samples returned by Memory Wrap are, by design, real samples and so inside the training distribution. Even though the returned counterfactuals are often edge cases, since they can be placed between two classes, they are still closer to the data distribution than the one produced by generative methods. Beyond the good results, these counterfactuals do not represent an alternative to post-hoc methods, especially when users are interested in different properties, like minimality, not supported by Memory Wrap.

4.3.3 Enhancing explanations

In this section, we list some possible ways to use the design and characteristics of Memory Wrap to enrich and enhance the explanations returned by classic methods.

Feature attribution Since the memory is actively used during the inference phase, we can use an attribution method to highlight the most important pixels for both the input image and the memory set for the current prediction. The only requirement is that the attribution method supports multi-input settings. We apply the technique of Integrated Gradients [70] using as a baseline a white image (Fig. 13). Here, for both Fig. 13a and d, the model predicts the wrong class. In Fig. 13d, the heatmap of the example-based explanation tells us that the model wrongly focuses on bird and sky colors, likely due to the unusual shape of the airplane. Indeed, it is very different from previously known shapes for airplanes, represented by the counterfactual with low weight and a heatmap that focuses only on the sky. Conversely, the example-based explanation is an image with a similar color for both the background and the bird, and the weight is much greater than the airplane, thus suggesting that the model is wrongly focusing on the colors. In Fig. 13a the model predicts the wrong class. Interestingly, in this case, the counterfactual is a sample of a different third class, thus meaning this input is challenging for the model. We hypothesize that this is due to the heavy blur effect, since the heatmaps of both the input and the example-based explanation focus on the bottom part of the digit, which is the most visible part of the current image.

On the opposite side, in Fig. 13c, the model predicts the correct class, focusing the attention on the head shape, a feature that is highlighted both in the input image and in the explanations. Finally, sometimes (Fig. 13b) counterfactuals are missing, and this means that the model is sure about its prediction, and it uses only examples of the same class. Note that the heatmaps are a bit noisy due to the lack of the application of VarGrad [1] or similar techniques to smooth them for computational reasons (Section 5.3).

Contrastive explanations Another method that can be applied and enhanced in our approach is the recently proposed *contrastive explanations* [22]. These explanations typically start from the input and a random image and extract the elements that make their predictions different [22], or highlight the parts that discriminate between the current prediction and another random class [81]. Memory Wrap has the potential to enhance these methods by exploiting the fact that it naturally selects suitable counterfactual classes and images, providing additional information. We leave for future research a way to extend these works to the multi-input settings and Memory Wrap.

5 Analysis

This section analyzes Memory Wrap, investigating the impact of its components and hyperparameters, the computational costs associated with its employment, its limitations, and the potential applicability on tasks different from image classification.

5.1 Ablation study

In this section, we study how the components of Memory Wrap, its hyperparameters, and the design choices impact the system's behavior. We will use MobileNet, ResNet, and EfficientNet for most experiments applied to SVHN, which has a shorter training time, allowing us to perform more experiments.

Representation power We start by applying the Major Voting algorithm (Section 4.2) to the models that include Memory Wrap. The results (Table 5) show that the training of Memory Wrap improves the representation power of the underlying encoder, helping it to separate the classes in the embedding space better, thus strengthening the training process. Additionally, this test helps weigh the contribution of different components. By comparing Table 5 and Fig. 5, we can see that ∼70% of the gain comes from the better representation learned, while ∼30% is due to the impact of the memory set at inference time.

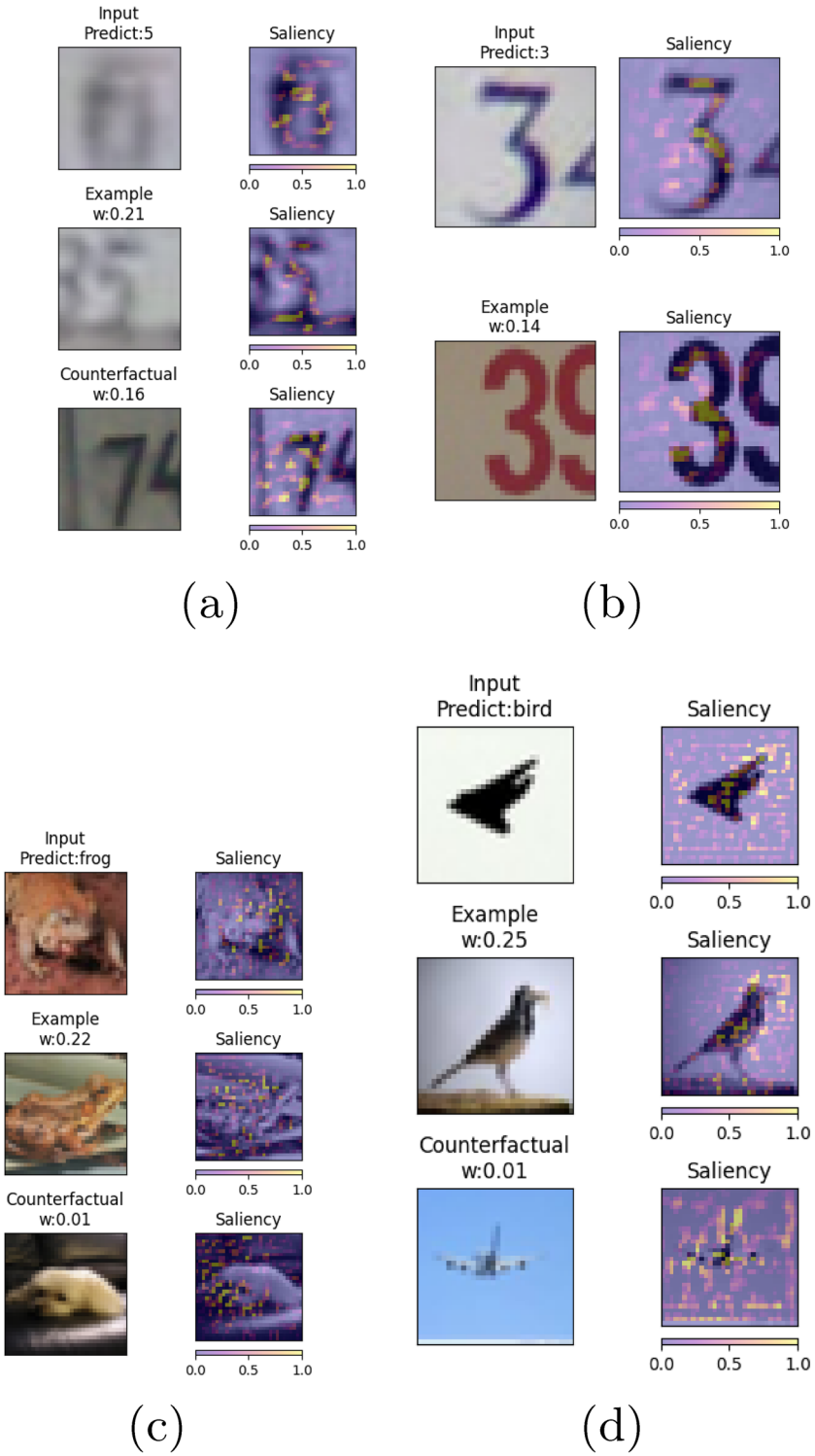

Fig. 13 Integrated Gradients heatmaps of the input, the example-based explanation associated with the highest weight in memory, and (eventually) the counterfactual associated with the highest weight. Each heatmap highlights the pixels that have a positive impact on the current prediction

Parameters We investigate whether these improvements come from the higher number of parameters introduced by Memory Wrap or the structure itself, comparing its performance to deeper variants of ResNet, DenseNet, and ShuffleNet. We consider ResNet34 (2.1M parameters against 1.3M of Memory Wrap), ShuffleNet doubling its size (1.2M parameters against 0.87M), and DenseNet-169 (2.6M against 2.1M) as deeper variants of the considered networks. Figure 14 tells us that this is not the case and that deeper variants often achieve worse performances

Table 5 Avg. accuracy and standard deviation over 15 runs reached applying the major voting algorithm over the embedding space learned by the encoders when trained using memory wrap

Major voting memory wrap avg. accuracy%			
	Samples		
Encoder	1000	2000	5000
EfficientNet	65.97 ± 1.33	76.34 ± 1.13	84.34 ± 0.62
MobileNet	64.63 ± 3.05	78.77 ± 0.84	86.69 ± 0.56
ResNet18	37.43 ± 4.49	67.49 ± 4.15	81.53 ± 0.81

due to overfitting, thus suggesting that Memory Wrap does not promote overfitting despite the higher number of parameters.

Number of samples The number of samples to keep in memory (i.e., 100) has been empirically chosen to be a trade-off between the number of samples for each class (10), the minimum number of data points in the training set (1000) across all the configurations, the training time, and the performance. The value is motivated by the fact that we want enough samples for each class in the memory set to get more representative samples for that class, but, at the same time, we do not want that the current sample is often included in the memory set and that the network exploits it. We see that the lower the number of samples, the lower the performance (Table 6). However, this comes at the cost of training and inference time, and the gap tends to vanish progressively. For example, an epoch of EfficientNetB0, trained using 5000 samples, lasts $\sim$ 9 seconds, $\sim$ 16 seconds, and $\sim$ 22 seconds when the memory contains respectively 20 samples, 300 samples, and 500 samples.

Distance In these tests, we compare the impact of the distance used to measure how different the input from each sample in the memory set is. We compare the performance over five runs using the cosine similarity and the L_2 distance. Figure 15 shows that cosine similarity is the best choice in our setup, outperforming L_2 in almost all the configurations. Moreover, the gap between using the L_2 distance in Memory Wrap and the standard baseline in some configurations is significantly lower than cosine similarity, thus suggesting that this is a crucial component of our module and confirming the findings of related works that use these similarities on memory networks [23, 78].

Sample selection ***Training time*** In this experiment, we test alternative selection mechanisms to decide which samples must be included in the memory set. We compare the random selection to a balanced selection of samples for each class, as well as a selection process inspired by the replay buffer of Deep Reinforcement Learning models.

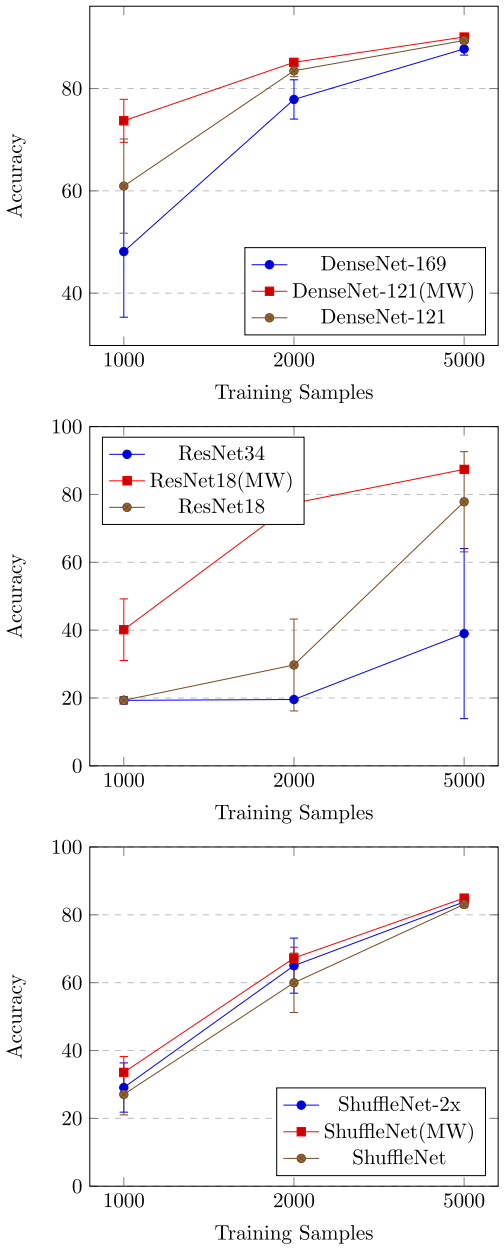

Fig. 14 Comparison between standard models (brown), Memory Wrap (MW) models (red), and deeper versions of standard models (blue) when the training dataset is a subset of SVHN

We consider two configurations for the latter: the *Replay-last* configuration samples the memory set by extracting 100 random samples from the last batch, while the *Replay-last-5* samples it from the pool of samples that include the previous five batches. The idea is to store memory samples recently used in the training process, keeping a limited size queue and sampling from it. Note that alternative mechanisms tailored to each input make the training process too slow or the memory footprint too high, destroying our previously described optimization (Section 4.1). For example, selecting the top 100 samples closer to each input requires that

Table 6 Avg. accuracy and standard deviation over 5 runs of the configuration of Memory Wrap trained using a variable number of samples in memory, when the training dataset is a subset of SVHN

Reduced SVHN Avg. Accuracy%				
		Samples		
Encoder	Memory	1000	2000	5000
EfficientNet	20	64.95 ± 2.46	75.80 ± 1.17	84.86 ± 0.99
	100	67.16 ± 1.33	77.02 ± 2.20	85.82 ± 0.45
	300	66.70 ± 1.58	77.97 ± 1.34	85.37 ± 0.68
	500	66.76 ± 0.98	77.67 ± 1.17	85.25 ± 1.02
MobileNet	20	63.42 ± 2.46	80.92 ± 1.42	88.33 ± 0.36
	100	68.31 ± 1.53	81.28 ± 0.69	88.47 ± 0.10
	300	65.08 ± 0.30	82.05 ± 0.75	88.93 ± 0.37
	500	69.88 ± 1.76	80.92 ± 1.74	88.61 ± 0.32
ResNet18	20	39.32 ± 7.21	72.54 ± 3.03	87.30 ± 0.41
	100	40.38 ± 9.32	74.36 ± 2.69	87.39 ± 0.45
	300	44.42 ± 10.97	74.63 ± 3.28	87.75 ± 0.62
	500	40.59 ± 12.27	76.97 ± 2.48	87.55 ± 0.35

each of them is encoded along with its own memory set, significantly increasing the memory footprint. Moreover, performing the selection on the representation space at each step increases the required time for training, thus making it infeasible.

In Table 7, we can observe that there are no clear winners and almost all configurations are equivalent. This means that different selection mechanisms are not able to bring enough benefits to be preferable to simple random selection. We can compare the properties of Memory Wrap and the benefits of alternative selection mechanisms to explain the slight difference between them.

We start considering the replay buffer: the advantage is that the model has been recently updated to recognize the samples in the memory set, and it can provide a better representation. But, since the similarity is computed with the current input, which is novel with respect to the weights, the benefits are not so significant. While balanced selection aims at providing, at each step, enough examples for each class to the memory set, Memory Wrap can already deal naturally with cases where it does not have enough information. Indeed, because the sparsemax dynamically selects the number of useful examples, it can also work with very few candidates. Additionally, the input encoding sent to the final layer can make up for cases where the memory set does not contain enough useful samples. Finally, note that these cases can be considered random noise in the training process, which is helpful for regularizing it.

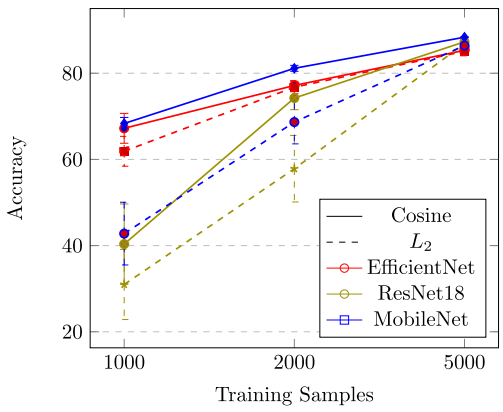

Fig. 15 Comparison between different encoders trained using respectively Cosine Similarity (solid lines) and L_2 distance (dashed lines) on a subset SVHN dataset

Inference time Once the model is trained, one can adopt a different mechanism for selection at testing time. We argue that the choice is context-dependent, and it should consider several factors.

For example, in applications where there are concerns about adversarial attacks, random selection could be a preferable option over a fixed selection. Random selection has the advantages of ensuring diversity (on average) and including only in-distribution data, but due to the randomness, it can produce two different example-based explanations for the same input. A simple solution to stabilize the explanations could be to compute a single centroid for each class, and then use them as the memory set, similarly to the Prototypical Networks. However, the fact that the memory stores real samples and uses them as separated entities is an essential factor for interpretability. When we compare the inputs with the memory set, we have multiple example-based explanations, and each of them is extracted from the same data distribution of input and has a clear semantic.

Table 7 Avg. accuracy and standard deviation over 5 runs Memory Wrap using different selection mechanisms to select samples in the memory set, when the training datasets is a subset of SVHN dataset

SVHN Avg. Accuracy %				
Encoder	Distance	1000	2000	5000
EfficientNet	Random	67.22 ± 3.47	77.16 ± 0.97	85.32 ± 0.90
	Balanced	65.80 ± 2.74	77.19 ± 1.27	85.80 ± 0.41
	Replay-Last	66.53 ± 1.80	76.36 ± 1.09	85.80 ± 0.49
	Replay-Last5	66.32 ± 1.70	77.56 ± 1.57	85.50 ± 0.54
MobileNet	Random	**68.34 ± 1.40**	81.14 ± 0.69	**88.36 ± 0.09**
	Balanced	**67.19 ± 1.77**	81.45 ± 1.30	**88.36 ± 0.68**
	Replay-Last	**66.92 ± 1.95**	81.17 ± 1.35	**88.51 ± 0.29**
	Replay-Last5	65.93 ± 1.98	82.09 ± 0.93	88.19 ± 0.26
ResNet18	Random	40.35 ± 9.31	74.24 ± 2.70	87.29 ± 0.39
	Balanced	40.74 ± 8.98	75.07 ± 4.20	**87.43 ± 0.25**
	Replay-Last	38.75 ± 11.90	**78.56 ± 1.84**	87.35 ± 0.85
	Replay-Last5	38.73 ± 14.25	**79.55 ± 1.16**	**87.70 ± 0.34**

For each configuration, we highlight in bold the best result and results that are within its margin

Conversely, the average of a class in the dataset is a non-real example, an out-of-distribution data point, and it is harder to understand. Indeed, because it is an aggregation of representations, we cannot easily visualize it, losing interpretability.

While this problem can be solved using K-medoids [62], this solution still uses only one sample for each class, thus lacking diversity. Indeed, the extracted medoid represents only a subspace of the input or latent space covered by the whole class. Thus, our suggestion is to use an algorithm that extracts several prototypes for each class [24, 33], ensuring that it captures diversity as much as possible both in terms of input space and latent space. In this paper, we opted for random selection at testing time to not bias the scores and to show the method's robustness.

Full dataset Finally, a natural question is whether the performance of Memory Wrap is better or worse than standard models when they both learn from the entire dataset (i.e., a more extensive dataset of 60000 samples). In these cases (reported in Table 8), they reach comparable performance most of the time. Hence, our approach is practical also when used with the entire dataset, thanks to the additional interpretability provided by its structure (Section 3.3).

5.2 Computational cost

This section briefly describes the changes in the computational cost when adding the Memory Wrap module to a deep neural network.

5.2.1 Parameters

The network size's increment depends mainly on the output dimensions of the encoder and the choice of the final layer. Let p be the number of network parameters until the last layer. In the standard baseline (without Memory Wrap), the

Table 8 Avg. accuracy and standard deviation over 15 runs of the baselines and Memory Wrap, when the training datasets are the whole SVHN, CIFAR10 and CINIC10 datasets. For each configuration, we highlight in bold the best result and results that are within its margin

Full Datasets Avg. Accuracy %				
Encoder	Model	SVHN	CIFAR10	CINIC10
EfficientNet	Standard	94.39 ± 0.24	**88.13 ± 0.38**	**77.31 ± 0.35**
	Memory Wrap	**94.67 ± 0.16**	**88.05 ± 0.20**	**77.34 ± 0.27**
MobileNet	Standard	**95.95 ± 0.09**	**88.78 ± 0.41**	**78.97 ± 0.31**
	Memory Wrap	95.63 ± 0.08	**88.49 ± 0.32**	**79.05 ± 0.15**
ResNet18	Standard	**95.70 ± 0.10**	**91.94 ± 0.19**	**82.05 ± 0.25**
	Memory Wrap	95.49 ± 0.11	91.49 ± 0.17	**82.04 ± 0.16**

last layer has dimensions (a_{std}, c) where a_{std} is the encoder output dimension and c is the number of classes, and so the total number of parameters is

$$p_{std} = p + (a_{std} \times c) \tag{21}$$

Now, consider the case of Memory Wrap with an MLP with 2 layers of dimension (a_{mw}, h) and (h, c). Because Memory Wrap takes as input both the input and the memory set encoding, then $a_{mw} = 2 \times a_{std}$, thus going from $a_{std} \times c$ to $a_{mw} \times (h + h \times c)$ parameters. Finally, we have to add the bias terms b_{MW} of the added neurons. Therefore, the total number of parameters is

$$p_{mw} = p + (a_{std} \times 2) \times h + (h \times c) + b_{MW} \tag{22}$$

Since we set $h = 2 \times a_{mw}$ to manage the additional complexity of the memory set, the increment is mainly caused by the a parameter. Table 9 shows the impact on MobileNet, ResNet18, and EfficientNet. EfficientNet, which has 320 units as the output layer, turns from a size of ∼3.6M parameters to ∼4.4M by adding Memory Wrap. Conversely, MobileNet, which has a larger output dimension of 1280, grows from ∼2.2M to ∼15.4M parameters.

The added parameters impact the space and time complexity due to the additional gradients that depend on the ratio between p and the added parameters. The impact is more significant for smaller networks, while for large networks, like Transformer, the impact is negligible.

Table 9 Number of parameters for the models with and without Memory Wrap

Number of parameters		
Model	Dimension	Parameters
EfficientNetB0	320	3 599 686
Only Memory	-	3 808 326
Memory Wrap	-	4 429 766
MobileNet-v2	1280	2 296 922
Only Memory	-	5 589 082
Memory Wrap	-	15 447 642
ResNet18	512	11 173 962
Only Memory	-	11 704 394
Memory Wrap	-	13 288 522

The column *dimension* indicates the number of output units of the encoder

5.2.2 Space complexity

Regarding the space complexity, in an ideal setting, one should provide a new memory set for each input during the training process. However, this makes the training and the inference process slower and the space requirements too high. Indeed, let m be the size of memory and n the dimension of the batch, the new input would contain $m \times n$ samples in place of n. For large batch sizes and many samples in memory, this cost can be too high. To reduce its memory footprint, we simplify the process by providing a single memory set for each new batch, maintaining the space required to a more manageable $m + n$. The consequence is that the testing batch size can influence performance at testing/validation time: a high batch size means a high dependency on random selection. To limit the instability, we fix a batch size at testing time to 500, and we repeat the test phase five times, extracting the average accuracy across all repetitions.

In our case, as explained in the Section 4.1, $m = c \times 10$ where c is the number of classes. Hence, the space complexity also depends on the number of classes, and for high values of c the cost could easily become prohibitive for standard workstations.

5.2.3 Time complexity

The training time added by the Memory Wrap modules depends on the number of training samples included in the memory set and the usage of a parallel structure. Indeed, in a sequential scenario, the encoder should first encode the input and then the memory set.

Let t be the number of seconds the encoder requires to encode a batch of n samples. If the number m of samples in the memory set is close ($m \sim n$), the network will need at least $t \times 2$ seconds to encode both of them. The higher the ratio $\frac{m}{n}$, the higher the impact on the training time.

To compute the overall time to train the network, we need to add the time required to backpropagate the error and update the weights. Since we use a single loss to train the neural network, we perform this operation only once for each step. In general, the time of the back-propagation is greater than the time needed for the forward call, thus, the factor of $t \times 2$ can be considered an upper bound of the overall time, as shown in the Table 10. The table shows the reference time for training EfficientNet, MobileNet, and Resnet for 40 epochs on the SVHN dataset on a V100 GPU card. We can observe that the training time of the models with the Memory Wrap module is $\sim \times 1,5$ slower than the network without it, but it is faster than training the same network using a double number of samples.

Table 10 Seconds needed to train each network for 40 epochs on the SVHN dataset

Training time (seconds)				
		Dataset samples		
Model	Variant	1000	2000	5000
EfficientNetB0	Std	55.21	108.31	266.98
	+ Memory Wrap	77.92	168.16	456.67
MobileNet-v2	Std	48.66	87.43	223.45
	+ Memory Wrap	76.89	156.86	400.08
ResNet18	Std	27.76	58.61	111.80
	+ Memory Wrap	47.68	92.18	205.20

5.3 Limitations

Memory footprint and training time The main limitation of Memory Wrap is that it increases the number of networks' parameters (Section 5.2.1), the memory footprint (Section 5.2.2), and the training process time (Section 5.2.3), due to the additional gradients towards the memory and the parameters needed to manage it. A possible solution to reduce the number of parameters would be to add a linear layer between the encoder and the Memory Wrap that projects data in a lower-dimensional space and preserves the performance as much as possible. However, one must tune this solution differently for each network, and it introduces a new hyperparameter. Despite the optimization that we made (Section 5.2.2), for large batch size, it could be problematic to keep the input, the memory set, and the gradients in memory in low resource scenarios. For example, in our application of Integrated Gradients, we avoid the usage of VarGrad [1] to smooth the heatmaps due to the massive requirement of memory needed to store gradients for input and memory set of all the generated perturbed instances. Moreover, the large memory footprint makes the training of Memory Wrap on dataset containing many classes difficult. For example, keeping the proportion of 10 samples for each class, the application on ImageNet [63] requires 10000 examples stored in the memory set, making its training difficult on standard workstations. Finally, the increased training time is caused mainly by the double forward pass of the encoder needed to parse both the input and the memory set. When there are enough resources available, one can recover the training time of standard models by parallelizing the encoding of input and the memory set. In this case, the only added time will be the one needed to compute the sparse content-based attention weights and perform the classification, but it is negligible with respect to the non-parallel structure. Another option at inference time is to fix the memory set a priori, computing its encodings only the first time, and use this set for the following inferences.

Bias amplification Another limitation is that the memory mechanism based on similarity could amplify the bias learned by the encoder. As shown in Section 3.3, the identification of such an event is straightforward, but currently, there are no countermeasures against it. A new adaptive or algorithmic selection mechanism of memory samples or a regularization method could mitigate the bias and improve the fairness of Memory Wrap.

5.4 Applicability to other tasks or domains

As explained in Section 3.2, at the implementation level, there are no special requirements on the underlying architecture. The only requirement is to access the latent representations of both the input and the memory set. Its application to other subcategories of image classification, like medical and hyperspectral image classification, where often data used for training deep learning systems is insufficient [25], should be straightforward, especially if they use variants of the networks tested in this paper [25].

Conversely, the extension to other domains, like audio classification and text classification, is an open question to investigate. While the experiments on ViT seem to suggest the applicability on networks based on different architectures, it is still unclear how the transformation involved to embed different types of data can impact the effectiveness of Memory Wrap. Moreover, the improved representation power (Section 5.1) could be beneficial for tasks that use high-level feature vectors in the pipeline, like text recognition in images [84] or image captioning.

Finally, adjusting the selection mechanism of the memory set, making it adaptive, or adding constraints into the learning process [85] of Memory Wrap, are all possible

future research directions to explore for the translation of Memory Wrap to different tasks.

6 Conclusion and future research

In this paper, we presented a module that improves the performance of deep neural networks in small data settings and aids the user to extract insights about the decision process. We showed how the module uses the training data to boost the performance and how to individuate and exploit example-based explanations and counterfactuals.

While Memory Wrap is a first step in this direction, we also encourage other researchers to focus on the topic of interpretable deep learning on small data, which are the most common setting in real-world scenarios but are surprisingly under-studied in literature. We think that the findings of this paper open up several future directions worth investigation, both for explanations and performance side. For example, future works could explore how to select memory samples that lead to optimal explanations, how to make an unbiased and fair inference process, or a way to reduce the current limitations of the module (Section 5.3).

Appendix A: Tables of results

This section presents the results of Section 4 in a table form (Tables 11, 12, 13, and 14) to better inspect the differences among configurations.

Table 11 Avg. accuracy and standard deviation over 15 runs of the baselines and Memory Wrap, when the training dataset is a subset of SVHN

Reduced SVHN Avg. Accuracy%				
		Samples		
Encoder	Model	1000	2000	5000
EfficientNet	Standard	57.70 ± 7.89	72.59 ± 4.00	81.89 ± 3.37
	K-NN$_{k=1}$	56.29 ± 8.89	72.01 ± 3.95	81.28 ± 3.51
	K-NN$_{k=5}$	56.47 ± 8.25	71.80 ± 3.95	81.09 ± 3.56
	K-NN$_{k=10}$	55.94 ± 8.17	71.20 ± 3.82	80.36 ± 3.58
	Voting	56.17 ± 8.20	71.68 ± 3.92	80.95 ± 3.56
	ProtoNet [67]	44.66 ± 4.94	54.94 ± 6.12	68.06 ± 6.95
	MatchNet [78]	13.84 ± 1.84	18.98 ± 2.77	26.80 ± 5.87
	OnlyMem	58.86 ± 3.30	75.79 ± 1.68	**85.30 ± 0.52**
	Memory Wrap	**66.78 ± 1.27**	**77.37 ± 1.25**	**85.55 ± 0.59**
MobileNet	Standard	42.71 ± 10.31	70.87 ± 4.20	85.52 ± 1.16
	K-NN$_{k=1}$	41.37 ± 10.50	69.63 ± 4.30	84.65 ± 1.32
	K-NN$_{k=5}$	41.66 ± 10.28	69.52 ± 4.23	84.40 ± 1.28
	K-NN$_{k=10}$	41.27 ± 9.78	67.85 ± 4.19	82.78 ± 1.58
	Voting	40.78 ± 9.57	67.17 ± 4.37	82.70 ± 1.70
	ProtoNet [67]	25.89 ± 4.06	34.02 ± 3.51	35.50 ± 2.35
	MatchNet [78]	10.98 ± 0.57	16.18 ± 2.48	16.48 ± 7.57
	OnlyMem	60.60 ± 3.14	**80.80 ± 2.05**	**88.77 ± 0.42**
	Memory Wrap	**66.93 ± 3.15**	**81.44 ± 0.76**	**88.68 ± 0.46**
ResNet18	Standard	19.32 ± 1.05	29.73 ± 13.54	77.84 ± 14.77
	K-NN$_{k=1}$	12.51 ± 1.35	22.25 ± 13.98	75.32 ± 16.60
	K-NN$_{k=5}$	14.26 ± 1.54	24.02 ± 14.08	75.65 ± 15.80
	K-NN$_{k=10}$	15.18 ± 1.65	24.69 ± 13.54	73.47 ± 14.72
	Voting	16.48 ± 1.48	24.45 ± 11.86	71.34 ± 14.25
	ProtoNet [67]	20.29 ± 4.09	30.99 ± 3.41	47.62 ± 3.99
	MatchNet [78]	10.99 ± 0.72	10.77 ± 0.79	21.17 ± 5.34
	OnlyMem	37.03 ± 4.61	72.69 ± 4.56	**87.00 ± 2.07**
	Memory Wrap	**40.14 ± 9.08**	**77.44 ± 3.21**	**87.39 ± 0.36**

For each configuration, we highlight in bold the best result and results that are within its margin

Table 12 Avg. accuracy and standard deviation over 15 runs of the standard model and two variants of Memory Wrap, when the training dataset is a subset of SVHN

Reduced SVHN Avg. Accuracy%				
		Samples		
Encoder	Type	1000	2000	5000
GoogLeNet	Standard	25.25 ± 9.39	61.45 ± 16.56	88.63 ± 2.60
	OnlyMem	**66.35 ± 6.93**	87.10 ± 1.17	92.16 ± 0.28
	Memory Wrap	**74.66 ± 9.01**	**88.32 ± 0.78**	**92.52 ± 0.25**
DenseNet	Standard	60.93 ± 9.21	83.47 ± 1.16	89.39 ± 0.60
	OnlyMem	40.94 ± 12.06	79.12 ± 5.36	**89.69 ± 0.63**
	Memory Wrap	**73.69 ± 4.20**	**85.12 ± 0.62**	**90.07 ± 0.49**

Table 12 (continued)

Reduced SVHN Avg. Accuracy%				
		Samples		
Encoder	Type	1000	2000	5000
ShuffleNet	Standard	27.06 ± 6.04	59.97 ± 8.75	83.08 ± 1.00
	OnlyMem	**32.22 ± 3.47**	60.06 ± 3.32	**85.56 ± 0.60**
	Memory Wrap	**33.56 ± 4.66**	**67.27 ± 3.18**	**84.93 ± 0.75**
ViT	Standard	76.22 ± 0.54	83.52 ± 0.60	91.15 ± 0.43
	OnlyMem	28.84 ± 11.01	19.58 ± 0.01	19.57 ± 0.02
	Memory Wrap	**78.09 ± 0.88**	**85.32 ± 0.73**	**91.58 ± 0.34**
WideResnet	Standard	76.50 ± 2.52	64.10 ± 27.13	82.61 ± 17.73
	OnlyMem	74.47 ± 5.33	**82.29 ± 2.38**	**86.36 ± 1.96**
	Memory Wrap	**81.87 ± 1.57**	**83.91** ± 2.26	**85.94** ± 1.89

For each configuration, we highlight in bold the best result and results that are within its margin

Table 13 Avg. accuracy and standard deviation over 15 runs of the standard model and two variants of Memory Wrap, when the training dataset is a subset of CIFAR10

Avg. Accuracy%				
		CIFAR10		
Encoder	Model	1000	2000	5000
EfficientNetB0	Standard	39.63 ± 2.16	47.25 ± 2.22	67.34 ± 2.37
	OnlyMem	40.60 ± 2.04	**52.87±2.07**	**70.82±0.52**
	Memory Wrap	**41.45±0.79**	**52.83±1.41**	**70.46±0.78**
MobileNet-v2	Standard	38.57 ± 2.11	50.36 ± 2.64	72.77 ± 2.21
	OnlyMem	**43.15±1.35**	**57.43±1.45**	**75.56±0.76**
	Memory Wrap	**43.87±1.40**	**57.12±1.36**	**75.33±0.62**
ResNet18	Standard	**40.03±1.36**	48.86 ± 1.57	65.95 ± 1.77
	OnlyMem	**40.35±0.89**	**51.11±1.22**	**70.28±0.80**
	Memory Wrap	**40.91±1.25**	**51.11±1.13**	**69.87±0.72**
GoogLeNet	Standard	51.91 ± 3.14	63.90 ± 2.21	79.09 ± 1.28
	OnlyMem	54.25 ± 0.80	66.00 ± 1.27	79.65 ± 0.59
	Memory Wrap	**55.91±1.20**	**66.79±1.03**	**80.27±0.49**
DenseNet	Standard	**46.99±1.61**	**56.95±1.68**	73.72 ± 1.41
	OnlyMem	**46.20±1.47**	**58.16±1.82**	**75.77±1.31**

Table 13 (continued)

Avg. Accuracy%				
		CIFAR10		
Encoder	Model	1000	2000	5000
	Memory Wrap	**47.64±1.58**	**58.60±1.85**	**75.50±1.33**
ShuffleNet	Standard	**37.86±1.16**	45.85 ± 1.26	65.92 ± 1.54
	OnlyMem	**38.15±1.14**	**48.91±2.12**	70.05 ± 0.84
	Memory Wrap	**37.90±1.15**	**47.50±1.79**	**68.52±1.38**
WideResnet	Standard	**48.99±1.51**	**53.58±10.87**	62.22 ± 4.90
	OnlyMem	43.12 ± 1.72	51.45 ± 2.75	**63.16±4.72**
	Memory Wrap	**49.02±2.57**	**53.36±3.51**	**65.21±2.43**
Vit	Standard	44.28 ± 0.57	50.86 ± 0.56	61.28 ± 0.64
	OnlyMem	39.88 ± 0.84	45.56 ± 0.59	58.11 ± 0.63
	Memory Wrap	**44.85±0.65**	**51.75±0.75**	**62.55±0.52**

For each configuration, we highlight in bold the best result and results that are within its margin

Table 14 Avg. accuracy and standard deviation over 15 runs of the standard model and two variants of Memory Wrap, when the training dataset is a subset CINIC10

Avg. Accuracy%				
		CINIC10		
Encoder	Model	1000	2000	5000
EfficientNetB0	Standard	29.50 ± 1.18	33.56 ± 1.26	45.98 ± 1.34
	OnlyMem	**30.46±1.17**	**36.17±1.54**	44.97 ± 0.95
	Memory Wrap	**30.45±0.64**	**36.65±1.03**	**47.06±0.91**
MobileNet-v2	Standard	29.61 ± 0.89	36.40 ± 1.58	50.41 ± 1.01
	OnlyMem	**32.46±1.07**	**39.90±0.82**	**52.51±0.77**
	Memory Wrap	**32.34±0.95**	**39.48±1.16**	**52.18±0.66**
ResNet18	Standard	31.18 ± 1.21	**37.67±0.98**	45.39 ± 1.07
	OnlyMem	30.79 ± 0.83	37.30 ± 0.57	**46.66±0.81**
	Memory Wrap	**32.15±0.68**	**38.51±0.96**	**46.39±0.67**
GoogLeNet	Standard	38.97 ± 1.16	47.83 ± 1.09	**58.47±0.91**
	OnlyMem	40.77 ± 0.78	48.53 ± 1.05	57.86 ± 0.55
	Memory Wrap	**42.19±0.92**	**50.47±0.77**	**58.98±0.68**

Table 14 (continued)

Avg. Accuracy%				
		CINIC10		
Encoder	Model	1000	2000	5000
DenseNet	Standard	**36.33±0.84**	41.78 ± 0.92	52.63 ± 0.95
	OnlyMem	35.64 ± 1.18	42.77 ± 0.69	**54.16±0.58**
	Memory Wrap	**37.02±0.95**	**43.55±1.05**	**53.59±0.61**
ShuffleNet	Standard	**28.32±0.85**	33.49 ± 0.93	46.36 ± 1.03
	OnlyMem	**28.68±0.93**	**35.33±1.09**	**48.25±0.90**
	Memory Wrap	**28.94±1.06**	**34.30±0.85**	47.33 ± 1.34
WideResnet	Standard	**35.66±0.65**	**41.68±0.79**	51.15 ± 0.93
	OnlyMem	32.23 ± 1.02	39.91 ± 0.71	51.12 ± 0.63
	Memory Wrap	**35.28±0.79**	**42.20±0.77**	**53.01±0.39**
Vit	Standard	**33.69±0.80**	38.94 ± 0.58	46.62 ± 0.39
	OnlyMem	30.86 ± 0.95	35.86 ± 0.54	44.09 ± 0.72
	Memory Wrap	**34.40±0.90**	**39.63±0.50**	**47.50±0.29**

For each configuration, we highlight in bold the best result and results that are within its margin

Appendix B: Results for the only memory variant

Table 15 shows how the variant Only Memory is not able to reach the same performance as standard models most of the time when trained using the complete datasets. We think this is due to this variant's difficulty in learning specific and rare patterns in images, caused by the lack of encoder input. Conversely, Tables 16 and 17 show that it has a similar representation power and similar interpretability to Memory Wrap in small data settings, making it a valid alternative in these scenarios.

Table 15 Avg. accuracy and standard deviation over 15 runs of the standard models and the Only Memory variant of Memory Wrap, when the training datasets are the whole SVHN, CIFAR10 and CINIC10 datasets

Full Datasets Avg. Accuracy %				
Encoder	Model	SVHN	CIFAR10	CINIC10
EfficientNet	Standard	94.39 ± 0.24	**88.13 ± 0.38**	**77.31 ± 0.35**
	OnlyMem	**94.63 ± 0.33**	86.48 ± 0.29	76.19 ± 0.25
MobileNet	Standard	**95.95 ± 0.09**	**88.78 ± 0.41**	**78.97 ± 0.31**
	OnlyMem	95.59 ± 0.11	86.37 ± 0.21	74.60 ± 0.13
ResNet18	Standard	**95.70 ± 0.10**	**91.94 ± 0.19**	**82.05 ± 0.25**
	OnlyMem	**95.82 ± 0.10**	91.36 ± 0.24	81.65 ± 0.19

For each configuration, we highlight in bold the best result and results that are within its margin

Table 16 Avg. accuracy and standard deviation over 15 runs reached applying the major voting algorithm over the embedding space learned by the encoders when trained using the Only Memory variant

Major Voting Memory Wrap Avg. Accuracy%			
	Samples		
Encoder	1000	2000	5000
EfficientNet	58.28 ± 3.25	74.73 ± 1.74	84.47 ± 0.59
MobileNet	59.08 ± 2.85	78.60 ± 2.17	87.07 ± 0.44
ResNet18	36.15 ± 4.06	65.84 ± 4.85	82.97 ± 3.43

Table 17 Avg. predictions matching accuracy and standard deviation comparison over 15 runs between the sample in the memory set with the highest sparse content-based attention weight ($\text{Top}_{M_{ce}}$), the example with the lowest weight but greater than zero ($\text{Bottom}_{M_{ce}}$), and a random sample (Random)

Predictions matching accuracy %				
Dataset	Example	1000	2000	5000
SVHN	$\text{Top}_{M_{ce}}$	81.43 ± 1.11	89.57 ± 0.98	93.83 ± 0.29
	$\text{Bottom}_{M_{ce}}$	41.37 ± 2.09	56.04 ± 2.65	68.88 ± 1.20
	Random	11.94 ± 0.60	11.76 ± 0.25	11.77 ± 0.18
CIFAR10	$\text{Top}_{M_{ce}}$	77.56 ± 0.93	82.46 ± 0.68	89.30 ± 0.43
	$\text{Bottom}_{M_{ce}}$	37.98 ± 1.48	45.19 ± 1.36	60.48 ± 0.88
	Random	10.68 ± 0.34	10.33 ± 0.26	10.16 ± 0.23
CINIC10	$\text{Top}_{M_{ce}}$	76.01 ± 0.73	78.34 ± 0.44	80.81 ± 0.91
	$\text{Bottom}_{M_{ce}}$	36.79 ± 1.42	39.93 ± 0.63	39.15 ± 0.65
	Random	10.91 ± 0.31	10.52 ± 0.18	10.35 ± 0.12

The scores are computed using the Only Memory variant on top of MobileNet-v2

Appendix C: Baselines using different training setup

In this section, we try to improve the performance of the Matching Networks and Prototypical Networks using the same hyperparameters used to achieve the best results in their respective repositories. Our setup modifications are a lower starting learning rate (1e-3) and a different optimizer (Adam). Table 18 show that these settings improve the performance of Matching networks and in some configurations of Prototypical Networks applied to ResNet. However, their accuracy is still lower than standard models and Memory Wrap.

Table 18 Avg. accuracy and standard deviation over 5 runs of the standard models, Matching Networks (MatchNet) and Prototypical Networks (ProtoNet) under different training configurations, when the training dataset is a subset of SVHN

Reduced SVHN Avg. Accuracy%				
		Samples		
Encoder	Model	1000	2000	5000
EfficientNet	Standard	57.70 ± 7.89	72.59 ± 4.00	81.89 ± 3.37
	ProtoNet [67]$_{SGD}$	44.66 ± 4.94	54.94 ± 6.12	68.06 ± 6.95
	ProtoNet [67]$_{ADAM}$	27.82 ± 4.27	43.09 ± 6.90	62.34 ± 4.31
	MatchNet [78]$_{SGD}$	13.84 ± 1.84	18.98 ± 2.77	26.80 ± 5.87
	MatchNet [78]$_{ADAM}$	18.03 ± 2.42	21.59 ± 2.58	26.37 ± 4.74
MobileNet	Standard	42.71 ± 10.31	70.87 ± 4.20	85.52 ± 1.16
	ProtoNet [67]$_{SGD}$	25.89 ± 4.06	34.02 ± 3.51	35.50 ± 2.35
	ProtoNet [67]$_{ADAM}$	21.26 ± 4.65	33.66 ± 5.89	35.27 ± 3.81
	MatchNet [78]$_{SGD}$	10.98 ± 0.57	16.18 ± 2.48	16.48 ± 7.57
	MatchNet [78]$_{ADAM}$	24.20 ± 2.14	28.53 ± 2.35	36.36 ± 5.01
ResNet18	Standard	19.32 ± 1.05	29.73 ± 13.54	77.84 ± 14.77
	ProtoNet [67]$_{SGD}$	20.29 ± 4.09	30.99 ± 3.41	47.62 ± 3.99
	ProtoNet [67]$_{ADAM}$	34.21 ± 9.56	37.65 ± 12.06	59.41 ± 7.05
	MatchNet [78]$_{SGD}$	10.99 ± 0.72	10.77 ± 0.79	21.17 ± 5.34
	MatchNet [78]$_{ADAM}$	20.27 ± 4.55	25.99 ± 4.68	29.46 ± 5.97

Acknowledgements This material is based upon work supported by Google Cloud platform.

Funding Open access funding provided by Università degli Studi di Roma La Sapienza within the CRUI-CARE Agreement.

Data Availability The datasets generated and analysed during the current study are available in the following repositories: CINIC10, https://github.com/BayesWatch/cinic-10; SVHN, http://ufldl.stanford.edu/housenumbers/; CIFAR10, https://www.cs.toronto.edu/~kriz/cifar.html; MNIST, http://yann.lecun.com/exdb/mnist/ Seeds and configurations used to generate the splits are available in the repository associated with the paper https://github.com/KRLGroup/memory-wrap. Memory Wrap can be installed as a Python package at https://pypi.org/project/memorywrap/.

Declarations

Competing interests All authors certify that they have no affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

1. Adebayo J, Gilmer J, Muelly M et al et al (2018) Sanity checks for saliency maps. In: Bengio S, Wallach H, Larochelle H (eds) Advances in neural information processing systems, Curran Associates, Inc, Vol 31
2. Bahng H, Chun S, Yun S, et al. (2020) Learning de-biased representations with biased representations. In: III HD, Singh A (eds) Proceedings of the 37th international conference on machine learning, proceedings of machine learning research, vol 119. PMLR, pp 528–539. https://proceedings.mlr.press/v119/bahng20a.html
3. Barz B, Denzler J (2020) Deep learning on small datasets without pre-training using cosine loss. In: Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)
4. Belle V, Papantonis I (2021) Principles and practice of explainable machine learning. Front Big Data 4:688969. https://doi.org/10.3389/fdata.2021.688969
5. Bercea CI, Pauly O, Maier A, et al. (2019) SHAMANN: Shared memory augmented neural networks. In: Lecture Notes in Computer Science. Springer International Publishing, Cham, pp 830–841. https://doi.org/10.1007/978-3-030-20351-1_65
6. Bietti A, Mialon G, Chen D, et al. (2019) A kernel perspective for regularizing deep neural networks. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 97. PMLR, pp 664–674. https://proceedings.mlr.press/v97/bietti19a.html

7. Blondel M, Martins A, Niculae V (2019) Learning classifiers with fenchel-young losses: generalized entropies, margins, and algorithms. In: Chaudhuri K, Sugiyama M (eds) Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics, Proceedings of Machine Learning Research, vol 89. PMLR, pp 606–615
8. Cai Q, Pan Y, Yao T, et al. (2018) Memory matching networks for one-shot image recognition. In: 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. IEEE, pp 4080–4088
9. Carion N, Massa F, Synnaeve G, et al. (2020) End-to-end object detection with transformers. In: Computer Vision – ECCV 2020. Springer International Publishing, Cham, pp 213–229. https://doi.org/10.1007/978-3-030-58452-8_13
10. Chen C, Li O, Tao D, et al. (2019) This looks like that: Deep learning for interpretable image recognition. In: Wallach H, Larochelle H, Beygelzimer A, et al. (eds) Advances in Neural Information. https://proceedings.neurips.cc/paper/2019/file/adf7ee2dcf142b0e11888e72b43fcb75-Paper.pdf. Accessed 1 Dec. 2021
11. Chen Z, Bei Y, Rudin C (2020) Concept whitening for interpretable image recognition. Nat Mach Intell 2(12):772–782. 10.1038/s42256-020-00265-z
12. Ching T, Himmelstein DS, Beaulieu-Jones BK, et al. (2018) Opportunities and obstacles for deep learning in biology and medicine. J R Soc Interf 15(141):20170–387. https://doi.org/10.1098/rsif.2017.0387. https://royalsocietypublishing.org/doi/10.1098/rsif.2017.0387
13. Correia GM, Niculae V, Martins AFT (2019) Adaptively sparse transformers. Assoc Comput Linguist. https://doi.org/10.18653/v1/d19-1223
14. Cover T, Hart P (1967) Nearest neighbor pattern classification. IEEE Trans Inf Theory 13(1):21–27. https://doi.org/10.1109/tit.1967.1053964
15. Darlow LN, Crowley EJ, Antoniou A et al (2018) Cinic-10 is not imagenet or cifar-10, University of Edinburgh, [dataset], Tech. rep. https://doi.org/10.7488/DS/2448
16. Ding W, Ming Y, Wang YK et al (2021) Memory augmented convolutional neural network and its application in bioimages. Neurocomputing 466:128–138. https://doi.org/10.1016/j.neucom.2021.09.012
17. Donahue J, Jia Y, Vinyals O et al (2014) Decaf: A deep convolutional activation feature for generic visual recognition. In: Xing EP, Jebara T (eds) Proceedings of the 31st international conference on machine learning, proceedings of machine learning research, vol 32. PMLR, China, pp 647–655. https://proceedings.mlr.press/v32/donahue14.html
18. Dosovitskiy A, Beyer L, Kolesnikov A et al (2021) An image is worth 16x16 words: transformers for image recognition at scale. International Conference on Learning Representations
19. van Engelen JE, Hoos HH (2019) A survey on semi-supervised learning. Mach Learn 109(2):373–440. https://doi.org/10.1007/s10994-019-05855-6
20. Feng L, Li Z, Kuang Z et al (2018) Extractive video summarizer with memory augmented neural networks Proceedings of the 26th ACM international conference on Multimedia. ACM, pp 976–983. https://doi.org/10.1145/3240508.3240651
21. Fu P, Xu Q, Zhang J et al (2019) A noise-resistant superpixel segmentation algorithm for hyperspectral images. Comput Mater Contin 59(2):509–515. https://doi.org/10.32604/cmc.2019.05250
22. Goyal Y, Wu Z, Ernst J, et al. (2019) Counterfactual visual explanations. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th international conference on machine learning, proceedings of machine learning research, vol 97. PMLR, pp 2376–2384. https://proceedings.mlr.press/v97/goyal19a.html
23. Graves A, Wayne G, Reynolds M et al (2016) Hybrid computing using a neural network with dynamic external memory. Nature 538(7626):471–476. https://doi.org/10.1038/nature20101
24. Gurumoorthy KS, Dhurandhar A, Cecchi G et al (2019) Efficient data representation by selecting prototypes with importance weights. In: 2019 IEEE International Conference on Data Mining (ICDM). IEEE. https://doi.org/10.1109/icdm.2019.00036
25. Song H, Yang W, Yuan H, Bufford H (2020) Deep 3d-multiscale densenet for hyperspectral image classification based on spatial-spectral information. Intell Autom Soft Comput 26(6):1441–1458. https://doi.org/10.32604/iasc.2020.011988
26. He K, Zhang X, Ren S, et al. (2016) Deep residual learning for image recognition. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). IEEE, pp 770–778
27. He L, Li J, Liu C et al (2018) Recent advances on spectral–spatial hyperspectral image classification: an overview and new guidelines. IEEE Trans Geosci Remote Sens 56(3):1579–1597. https://doi.org/10.1109/tgrs.2017.2765364
28. Huang G, Liu Z, Maaten LVD, et al. (2017) Densely connected convolutional networks. In: 2017 IEEE Conference on computer vision and pattern recognition (CVPR). IEEE, pp 4700–4708
29. Hyun Y, Kim H (2020) Memory-augmented convolutional neural networks with triplet loss for imbalanced wafer defect pattern classification. IEEE Trans Semicond Manuf 33(4):622–634. https://doi.org/10.1109/tsm.2020.3010984
30. Kang D, Kwon H, Min J, et al. (2021) Relational embedding for few-shot classification. In: Proceedings of the IEEE/CVF international conference on Computer Vision (ICCV), pp 8822–8833
31. Kenny EM, Keane MT (2019) Twin-systems to explain artificial neural networks using case-based reasoning: Comparative tests of feature-weighting methods in ANN-CBR twins for XAI. In: Proceedings of the twenty-eighth international joint conference on artificial intelligence. International Joint Conferences on Artificial Intelligence Organization, pp 2708–2715
32. Kenny EM, Keane MT (2021) Explaining deep learning using examples: Optimal feature weighting methods for twin systems using post-hoc, explanation-by-example in XAI. Knowl-Based Syst 233:107,530. https://doi.org/10.1016/j.knosys.2021.107530
33. Kim B, Khanna R, Koyejo OO et al (2016) Examples are not enough, learn to criticize! criticism for interpretability. In: Lee D, Sugiyama M, Luxburg U (eds) Advances in neural information processing systems, vol 29. Curran Associates, Inc
34. Kobayashi T (2021) t-vmf similarity for regularizing in-class feature distribution. In: Proceedings of IEEE conference on computer vision and pattern recognition (CVPR)
35. Koh PW, Nguyen T, Tang YS, et al. (2020) Concept bottleneck models. In: III HD, Singh A (eds) Proceedings of the 37th international conference on machine learning, proceedings of machine learning research, vol 119. PMLR, pp 5338–5348. https://proceedings.mlr.press/v119/koh20a.html
36. Krizhevsky A (2009) Learning multiple layers of features from tiny images. University of Toronto, [dataset], Tech. rep.
37. Kumar A, Irsoy O, Ondruska P, et al. (2016) Ask me anything: Dynamic memory networks for natural language processing. In: Proceedings of the 33rd international conference on international conference on machine learning, ICML'16, vol 48, pp 1378–1387. JMLR.org
38. La Rosa B, Capobianco R, Nardi D (2020) Explainable inference on sequential data via memory-tracking. In: Proceedings of the twenty-ninth international joint conference on artificial intelligence. International joint conferences on artificial intelligence organization, pp 2006–2013
39. Laugel T, Lesot MJ, Marsala C et al (2018) Comparison-based inverse classification for interpretability in machine

learning. In: Communications in computer and information science. Springer International Publishing, Cham, pp 100–111. https://doi.org/10.1007/978-3-319-91473-2_9
40. Lecun Y, Bottou L, Bengio Y et al (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324. https://doi.org/10.1109/5.726791
41. Lezama J, Qiu Q, Musé P et al (2018) OIÉ: orthogonal low-rank embedding - a plug and play geometric loss for deep learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)
42. Li H, Zeng W, Xiao G et al (2020) The instance-aware automatic image colorization based on deep convolutional neural network. Intell Autom Soft Comput 26(4):841–846. https://doi.org/10.32604/iasc.2020.010118
43. Li L, Wang B, Verma M et al (2021) Scouter: Slot attention-based classifier for explainable image recognition. In: Proceedings of the IEEE/CVF international conference on computer vision (ICCV), pp 1046–1055
44. Li O, Liu H, Chen C et al (2018a) Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions. In: Proceedings of the AAAI conference on artificial intelligence. AAAI Press, USA. https://ojs.aaai.org/index.php/AAAI/article/view/11771. Accessed 10 Dec. 2021
45. Li O, Liu H, Chen C, et al. (2018b) Deep learning for case-based reasoning through prototypes: a neural network that explains its predictions. In: Proceedings of the thirty-second AAAI conference on artificial intelligence and thirtieth innovative applications of artificial intelligence conference and eighth AAAI symposium on educational advances in artificial intelligence. AAAI Press, USA. AAAI'18/IAAI'18/EAAI'18
46. Li Y, Zhang H, Xue X, et al. (2018c) Deep learning for remote sensing image classification: a survey. WIREs Data Min Knowl Discov 8(6):e–1264. https://doi.org/10.1002/widm.1264
47. Lipton ZC (2018) The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. Queue 16(3):31–57
48. Liu J, Ye J (2009) Efficient euclidean projections in linear time. In: Proceedings of the 26th Annual International Conference on Machine Learning - ICML '09. ACM Press. https://doi.org/10.1145/1553374.1553459
49. Liu S, Kailkhura B, Loveland D, et al. (2019) Generative counterfactual introspection for explainable deep learning. In: 2019 IEEE Global Conference on Signal and Information Processing (GlobalSIP), pp 1–5
50. Looveren AV, Klaise J (2021) Interpretable counterfactual explanations guided by prototypes. Springer International Publishing, Cham, pp 650–665. https://doi.org/10.1007/978-3-030-86520-7_40
51. Ma C, Shen C, Dick A, et al. (2018) Visual question answering with memory-augmented networks. In: 2018 IEEE/CVF Conference on computer vision and pattern recognition. IEEE, pp 6975–6984
52. Malaviya C, Ferreira P, Martins AFT (2018) Sparse and constrained attention for neural machine translation. In: Proceedings of the 56th annual meeting of the association for computational linguistics (Volume 2: Short Papers). Association for Computational Linguistics. https://doi.org/10.18653/v1/p18-2059
53. Martins A, Astudillo R (2016a) From softmax to sparsemax: A sparse model of attention and multi-label classification. In: Balcan MF, Weinberger KQ (eds) Proceedings of the 33rd international conference on machine learning, proceedings of machine learning research, vol 48. PMLR, New York. USA, New York, pp 1614–1623
54. Martins AFT, Astudillo RF (2016b) From softmax to sparsemax: a sparse model of attention and multi-label classification. In: Proceedings of the 33rd international conference on international conference on machine learning - Volume 48. JMLR.org, ICML'16, pp 1614–1623
55. Miranda E, Aryuni M, Irwansyah E (2016) A survey of medical image classification techniques. In: 2016 International Conference on Information Management and Technology (ICIMTech), pp 56–61. https://doi.org/10.1109/ICIMTech.2016.7930302
56. Mothilal RK, Sharma A, Tan C (2020) Explaining machine learning classifiers through diverse counterfactual explanations. In: Proceedings of the 2020 Conference on Fairness Accountability, and Transparency. ACM. https://doi.org/10.1145/3351095.3372850
57. Muthusamy D, Rakkimuthu P (2021) Steepest deep bipolar cascade correlation for finger-vein verification. Appl Intell 52(4):3825–3845. https://doi.org/10.1007/s10489-021-02619-5
58. Netzer Y, Wang T, Coates A et al (2011) Reading digits in natural images with unsupervised feature learning. In: NIPS Workshop on Deep Learning and Unsupervised Feature Learning
59. phi Nguyen A, Martínez MR (2020) On quantitative aspects of model interpretability. https://arxiv.org/abs/2007.07584
60. Nugent C, Cunningham P (2005) A case-based explanation system for black-box systems. Artif Intell Rev 24(2):163–178. https://doi.org/10.1007/s10462-005-4609-5
61. Papernot N, McDaniel P (2018) Deep k-nearest neighbors: towards confident, interpretable and robust deep learning. https://arxiv.org/abs/1803.04765
62. Rdusseeun L, Kaufman P (1987) Clustering by means of medoids. In: Proceedings of the statistical data analysis based on the L1 norm conference, Neuchatel, Switzerland, pp 405–416
63. Russakovsky O, Deng J, Su H et al (2015) ImageNet Large Scale Visual Recognition Challenge. Int J Comput Vis (IJCV) 115(3):211–252. https://doi.org/10.1007/s11263-015-0816-y
64. Sandler M, Howard A, Zhu M et al (2018) Mobilenetv2: Inverted residuals and linear bottlenecks. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)
65. Santoro A, Bartunov S, Botvinick M et al (2016) Meta-learning with memory-augmented neural networks. In: Proceedings of machine learning research, vol 48. PMLR, pp 1842–1850
66. Singh G, Yow KC (2021) These do not look like those: An interpretable deep learning model for image recognition. IEEE Access 9:41,482–41,493. https://doi.org/10.1109/access.2021.3064838
67. Snell J, Swersky K, Zemel R (2017) Prototypical networks for few-shot learning. In: Proceedings of the 31st International conference on neural information processing systems. Curran Associates Inc., NIPS'17, pp 4080–4090
68. Stigler SM (1989) Francis galton's account of the invention of correlation. Stat Sci 4(2):73–79. https://doi.org/10.1214/ss/1177012580
69. Sukhbaatar S, Szlam A, Weston J, et al. (2015) End-to-end memory networks. In: Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2. MIT Press, NIPS'15, pp 2440–2448
70. Sundararajan M, Taly A, Yan Q (2017) Axiomatic attribution for deep networks. In: Proceedings of the 34th international conference on machine learning - Volume 70. JMLR.org, ICML'17, pp 3319–3328
71. Sung F, Yang Y, Zhang L et al (2018) Learning to compare: Relation network for few-shot learning. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)
72. Szegedy C, Liu W, Jia Y, et al. (2015) Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR), pp 1–9
73. Tan C, Sun F, Kong T, et al. (2018) A survey on deep transfer learning. In: Artificial Neural Networks and Machine Learning – ICANN 2018. Springer International Publishing, Cham, pp 270–279. https://doi.org/10.1007/978-3-030-01424-7_27

74. Tan M, Le Q (2019) EfficientNet: Rethinking model scaling for convolutional neural networks. In: Chaudhuri K, Salakhutdinov R (eds) Proceedings of the 36th international conference on machine learning, proceedings of machine learning research, vol 97. PMLR, pp 6105-6114
75. Tsallis C (1988) Possible generalization of boltzmann-gibbs statistics. J Stat Phys 52(1-2):479–487. https://doi.org/10.1007/bf01016429
76. Ulicny M, Krylov VA, Dahyot R (2019) Harmonic networks with limited training samples. In: 2019 27th European Signal Processing Conference (EUSIPCO). IEEE. https://doi.org/10.23919/eusipco.2019.8902831
77. Varshneya S, Ledent A, Vandermeulen RA et al (2021) Learning interpretable concept groups in CNNs. In: Proceedings of the thirtieth international joint conference on artificial intelligence. International joint conferences on artificial intelligence organization. https://doi.org/10.24963/ijcai.2021/147
78. Vinyals O, Blundell C, Lillicrap T, et al. (2016) Matching networks for one shot learning. In: Proceedings of the 30th international conference on neural information processing systems. Curran Associates Inc., NIPS'16, pp 3637–3645
79. Wachter S, Mittelstadt B, Russell C (2018) Counterfactual explanations without opening the black box: Automated decisions and the gdpr. Harv J Law Technol 31(2):841–887
80. Wang J, Liu H, Wang X, et al. (2021) Interpretable image recognition by constructing transparent embedding space. In: Proceedings of the IEEE/CVF international conference on computer vision (ICCV), pp 895–904
81. Wang P, Vasconcelos N (2020) Scout: Self-aware discriminant counterfactual explanations. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)
82. Wang Y, Yao Q, Kwok JT et al (2020) Generalizing from a few examples. ACM Comput Surv 53(3):1–34. https://doi.org/10.1145/3386252
83. Wiegreffe S, Pinter Y (2019) Attention is not not explanation. In: Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th International joint conference on natural language processing (EMNLP-IJCNLP). Association for Computational Linguistics. https://doi.org/10.18653/v1/d19-1002
84. Wu X, Luo C, Zhang Q et al (2019) Text detection and recognition for natural scene images using deep convolutional neural networks. Comput Mater Contin 61(1):289–300. https://doi.org/10.32604/cmc.2019.05990
85. Xing C, Wang M, Wang Z et al (2021) Diagonalized low-rank learning for hyperspectral image classification. IEEE Trans Geosci Remote Sens 60:1–12. https://doi.org/10.1109/tgrs.2021.3085672
86. Yasmeen U, Shah JH, Khan MA et al (2020) Text detection and classification from low quality natural images. Intell Autom Soft Comput 26(4):1251–1266. https://doi.org/10.32604/iasc.2020.012775
87. Zagoruyko S, Komodakis N (2016) Wide residual networks. In: Proceedings of the British Machine Vision Conference 2016. British Machine Vision Association. https://doi.org/10.5244/c.30.87
88. Zhang X, Zhou X, Lin M, et al. (2018) ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In: 2018 IEEE/CVF conference on computer vision and pattern recognition. IEEE, pp 6848–6856. https://doi.org/10.1109/cvpr.2018.00716
89. Zheng H, Fu J, Mei T et al (2017) Learning multi-attention convolutional neural network for fine-grained image recognition. In: 2017 IEEE International conference on computer vision (ICCV). IEEE. https://doi.org/10.1109/iccv.2017.557

Biagio La Rosa is a Ph.D. student at Sapienza University of Rome. His research focuses on the field of eXplainable Artificial Intelligence and in particular on exploring methods for interpreting deep neural networks. The application domains of his work include natural language processing and computer vision. He has been one of the organizers of the XAI4Debugging workshop at NeurIPS2021, a workshop that aimed at bringing together researchers of the visual analytics community and the eXplainable artificial intelligence community around the problem of explaining deep learning systems. He is active in the AI community, publishing international conferences and contributing to the organization of conferences.

Roberto Capobianco is a Senior Research Scientist at Sony AI as well as Contract Professor and member of the PhD board at the Department of Computer, Control, and Management Engineering "Antonio Ruberti" (DIAG), Sapienza University of Rome. He is the founder of the Knowledge, Reasoning and Learning Research Group in the Collaborative and Cognitive Robotics Lab in the same department. With long-standing machine and deep learning experience both in academia and industry, his research is focused on reinforcement learning, explainable AI, robot control and knowledge representation. He has recently organized the Explainable AI for Debugging Workshop at NeurIPS, as well as a Workshop on the Evaluation and Benchmarking of Human-Centered AI Systems. He has additionally contributed to the organization of other conferences as well as robotics competitions (RoCKIn and SciRoc challenges) in the context of EU projects.

Daniele Nardi received the Graduated degree in electrical engineering at Politecnico di Torino in 1981. He has been a Full Professor with the Faculty of Ingegneria dell'Informazione, Informatica, Statistica, Department of Computer, Control, and Management Engineering "A. Ruberti," Sapienza University of Rome, Rome, Italy, since 2000. He is currently the referent for the Master curriculum in Artificial Intelligence and Robotics, and he is teaching the course Artificial Intelligence of the master, since several years. He leads the research laboratory "Cognitive Robot Teams," addressing different research topics: Cognitive Robotics, Localization, Navigation, Perception, Cooperation in multirobot systems, Human Robot Interaction, multimodal interfaces and speech. The scientific and technical achievements are deployed in manifold application domains: Ambient Intelligence and robots to support elderly people, Disaster Response robots to explore and gather information from the environment, Cultural Heritage, Precision Agriculture, Soccer Player robots for RoboCup competitions. He has a publication record with several contributions in Artificial Intelligence and Robotics (H-index 50). He has been the principal investigator of several collaborative projects funded by FP7, H2020, and several other research funding institutions. Mr. Nardi is a EurAI Fellow, and was a President of RoboCup Federation from 2011 to 2014.