A Memory Transformer Network for Incremental Learning

Ahmet Iscen[1]
Tom Bird[2]
Mathilde Caron[1]
Alireza Fathi[1]
Cordelia Schmid[1]

[1]Google Research
[2]University College London

Abstract

We study class-incremental learning, a training setup in which new classes of data are observed over time for the model to learn from. Despite the straightforward problem formulation, the naive application of classification models to class-incremental learning results in the "catastrophic forgetting" of previously seen classes. One of the most successful existing methods has been the use of a memory of exemplars, which overcomes the issue of catastrophic forgetting by saving a subset of past data into a memory bank and utilizing it to prevent forgetting when training future tasks. In our paper, we propose to enhance the utilization of this memory bank: we not only use it as a source of additional training data like existing works but also integrate it in the prediction process explicitly. Our method, the Memory Transformer Network (MTN), learns how to combine and aggregate the information from the nearest neighbors in the memory with a transformer to make more accurate predictions. We conduct extensive experiments and ablations to evaluate our approach. We show that MTN achieves state-of-the-art performance on the challenging ImageNet-1k and Google-Landmarks-1k incremental learning benchmarks.

1 Introduction

The goal of Incremental Learning (IL) is to design models capable of dynamically learning when the input data becomes available gradually over time. The main challenge is to adapt to the new incoming data, while still being able to remember the previously learned knowledge. In the past, many IL approaches have been proposed to avoid *catastrophic forgetting*, which refers to the situation where an incremental learner ends up performing poorly on previously learned tasks [11]. One popular and successful way of approaching catastrophic forgetting is to keep a small subset of training data from earlier tasks, *i.e*. *exemplars* stored in a *memory bank*, and use them to augment the data when training on the new task, which is referred as *rehearsal* [27]. This approach has

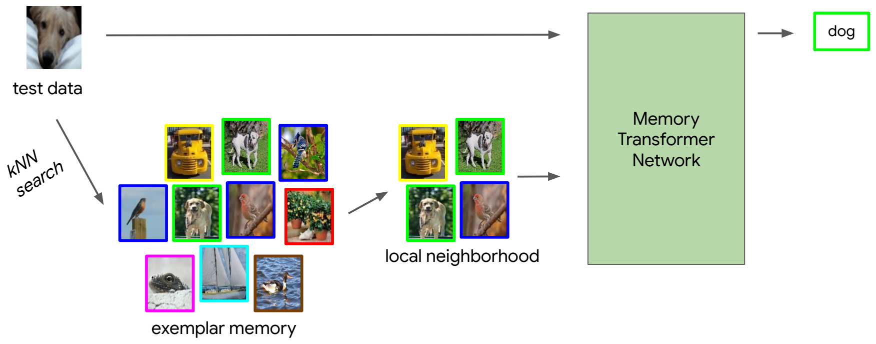

Figure 1: The Memory Transformer Network (MTN) uses a memory of data from previous and current tasks to help classify incoming data. Feature representations of images are stored in the exemplar memory. During the inference, kNN search is performed to find the local neighborhood of the test query vector. MTN then aggregates the information from the local neighborhood to make its prediction.

been shown to be very effective for IL and, consequently, the use of a memory to learn the model parameters constitutes a key component of most modern methods [1, 2, 14, 15, 26, 38].

In our work, we propose to go one step further and exploit the exemplars not only as a source of additional training data but also in the prediction process itself. Our hypothesis is that the relationships between a query (*i.e.*. a training or testing example) and the different elements of the memory bank generate a strong signal, which helps the incremental learner make a more robust and accurate prediction with respect to the given query. To this end, we propose the *Memory Transformer Network* (MTN) which looks at the local feature neighborhood of a query vector before making a prediction. MTN is a light-weight transformer that makes a class prediction for a given query by directly modeling the relationship between this query and the feature representations of the exemplars in a memory bank. Since conditioning on the entire memory bank would be too computationally demanding, we choose to feed MTN only a reduced set of exemplars (selected with nearest neighbour search).

Our approach allows rare patterns to be memorized explicitly, rather than implicitly in model parameters. The prediction of a query does not only depend on its representation anymore, it also depends on its nearest neighbors amongst the preserved exemplars. This makes the model more accurate by directly retrieving the corner cases from the memory, instead of dedicating the model parameters to remember them.

MTN is inspired from recent *memory transformer* architectures in language modeling [39] and video recognition [36], where the input sequence to the transformer follows a natural ordering, *e.g.* sequence of words and frames. In these frameworks, the memory typically contains previously seen words and frames which helps the system make predictions. MTN extends memory transformers to image classification, where we define the input sequence based on the local neighborhood of the query in the feature space.

To summarize, our contributions are as follows:

- We believe our approach is the first to propose a model with external memory for class-incremental learning. We do so by utilizing a transformer that combines the query input features with the memory exemplar features.

- Unlike the existing works on class-incremental learning, we leverage the exemplars not only during the model training, but also directly in the process of decision making by having the output distribution depend on the external memory.

- Our method achieves state-of-the-art results on ImageNet-1k [8] and Google-Landmarks-1k [35] datasets.

2 Related works

In this section we discuss previous works on class-incremental learning, looking at both exemplar-based and exemplar-free methods. We also discuss methods in the literature which augment neural networks with memory, which is at the heart of our method.

A popular method in class-incremental learning is to keep a subset of a task training data as exemplars, and use it in subsequent tasks to mitigate the forgetting effect as incremental training proceeds. Adding the exemplars to the current task training set is one effective way to prevent forgetting, and is referred to as rehearsal [27]. Rebuffi *et al.* propose the iCaRL method [26], which selects the exemplars via a simple algorithm known as herding and uses the mean exemplar vector for each class at inference time rather than the classifier used during training. Recent works extend iCaRL, and have focused on correcting the model bias towards tasks later on in incremental training. Some methods train additional parameters [38, 41], while others amend the loss function [1, 2, 14, 23, 32]. There have also been explorations into optimizing the exemplars themselves [19], and architectural changes like splitting the model into stable and plastic weights [20] to allow for fast task training but stable knowledge of past tasks. To reduce the memory footprint of saving exemplars as images, Iscen *et al.* [15] store features rather than images, adapting old exemplar features to be compatible with new features via a separate network.

There are also many works in class-incremental learning which do not utilize exemplars. Li and Hoiem [17] add a knowledge distillation loss to the usual classification loss, helping to prevent forgetting as tasks are seen. Lopez-Paz and Ranzato [21] modify the gradient update at a given task, and also pass updates to previously seen classes. There are also a variety of methods which learn a generative model to simulate data from past tasks, a method known as *pseudo-rehearsal*. Multiple methods explore using adversarial training to generate the pseudo-exemplars [29, 37], [40], whereas Smith *et al.* [30] use model-inversion to generate the fake data.

Santoro *et al.* [28] describe a memory-augmented neural network (MANN), in which a differentiable external memory is learned in order to improve neural network performance on meta-learning. There are a number of works in the NLP domain that allow the model to utilize an external memory bank of knowledge [10, 13]. Liu *et al.* [18] use a memory implemented through a Hopfield network and pretrained features on unsupervised learning. Mi and Faltings [22] use a MANN in incremental learning, on the application of recommendation systems. There are a number of works which resemble learned versions of nearest neighbors, predicting labels by calculating similarities of a query point to a support set of features and labels [24, 31, 34]. There are also recent works that use transformers to process the query points relation to memory. Gordo *et al.* [12] apply a transformer for retrieval, adapting a query point and its nearest neighbors. Doersch et al. [7] use a transformer in meta-learning, predicting a label for a query based on the spatially-aware similarities to labelled points from memory.

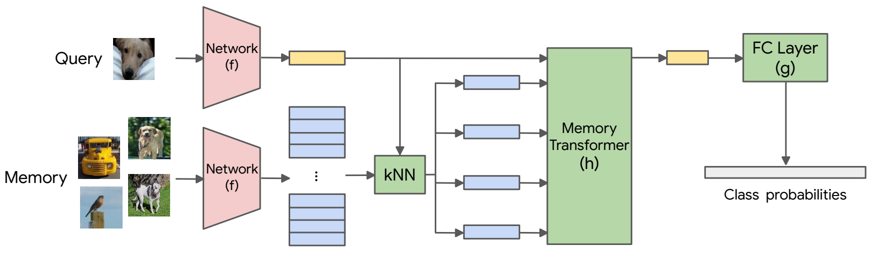

Figure 2: **Overview of the MTN model.** A query feature vector is provided as input to the MTN, along with the k nearest neighbors from a set of exemplar memory feature vectors. MTN adapts the input query feature by learning to model the relationship between the query vector and its nearest neighbors. We then pass the adapted feature to a linear head to calculate a set class scores (or logits).

3 Method

In this section, we begin by describing the problem definition and the notation used throughout this paper. We then introduce our method in Section 3.2.

3.1 Background and notation

We define the dataset $\mathcal{D} = \{(x,y) | x \in \mathcal{X}, y \in \mathcal{Y}\}$ which consists of a set of images $\mathcal{X}$ and their corresponding labels $\mathcal{Y}$. We assume that the labels $\mathcal{Y}$ belong to one of the classes in $\mathcal{C}$, which is a set of C classes.

In incremental learning, we do not have access to the entire dataset at once. Instead, the dataset is split into T tasks, where each task contains disjoint classes, *i.e.* $\mathcal{C}^1, \mathcal{C}^2, \dots, \mathcal{C}^T$, where $\mathcal{C} = \mathcal{C}^1 \cup \mathcal{C}^2 \cup \dots \cup \mathcal{C}^T$ and $\mathcal{C}^i \cap \mathcal{C}^j = \emptyset$ for $i \neq j$. At task t, we only have access to the data corresponding to the set of classes $\mathcal{C}^t$, and the training data in task t is defined by $\mathcal{D}^t = \{(x,y) | x \in \mathcal{X}^t, y \in \mathcal{Y}^t\}$.

The learned model typically consists of two parts: a *feature extractor* and a *classifier*. The feature extractor f maps each input image x to a d-dimensional feature vector, *i.e.* $\mathbf{q} := f(x) \in \mathbb{R}^d$. Consequently, the *classifier* $g : \mathbb{R}^d \to \mathbb{R}^C$ is applied to each vector $\mathbf{q}$ to obtain class prediction scores, or *logits*, denoted by $\mathbf{z}$, *i.e.* $\mathbf{z} := g(\mathbf{q}) \in \mathbb{R}^C$. We denote the model parameters used in the feature extractor and classifier as θ.

The best-performing recent methods in class-incremental learning have been rehearsal-based methods [1, 2, 14, 15, 26, 38]. That is, they rely on saving some subset of class training data $\mathcal{M}^t \subset \mathcal{D}^t$, *i.e.* the *exemplars*. Exemplars can be stored as images [1, 2, 14, 26, 38], or features [15]. The training data in task t contains all the exemplars from the previous tasks in order to prevent catastrophic forgetting, *i.e.* $\mathcal{D}^t_{\text{train}} = \mathcal{D}^t \cup \mathcal{M}^{1:t-1}$. Typically, the number of exemplars kept is significantly less than the total number of samples, *i.e.* $|\mathcal{M}^t| << |\mathcal{D}^t|$.

Recent works show that alternatives to the usual cross-entropy loss can benefit training in class-incremental learning [1, 2, 14]. In this work, we use the separated-softmax loss [1], which splits the classification loss into two terms:

$$\begin{aligned}\mathcal{L}_{SS}(x,y) = \; & \delta_{y \in \mathcal{C}^t} \mathcal{D}_{KL}(\mathbf{y}^t \;||\; \mathbf{p}^t) + \\ & \delta_{y \in \mathcal{C}^{1:t-1}} \mathcal{D}_{KL}(\mathbf{y}^{1:t-1} \;||\; \mathbf{p}^{1:t-1}),\end{aligned} \tag{1}$$

where $\mathbf{y}^t$ is the one-hot vector giving the label y in the task classes $\mathcal{C}^t$, and $\mathbf{p}^t = \text{softmax}(\mathbf{z}^t)$ is the vector of probabilities, over the same classes, produced by the model. We also augment the classification loss(Eq. (1)) with a task-wise distillation loss, which has been shown to be effective in preventing forgetting during class-incremental learning [4, 17]:

$$\mathcal{L}_{TKD}(x) = \sum_{i=1}^{t-1} \mathcal{D}_{KL}(\mathbf{p}^i_{t-1} \,||\, \mathbf{p}^i), \tag{2}$$

where $\mathbf{p}^i_{t-1}$ is obtained with the frozen model parameters θ_{t-1} from the end of the previous task training. Overall our objective is:

$$\mathcal{L}(x,y) = \mathcal{L}_{SS}(x,y) + \mathcal{L}_{TKD}(x). \tag{3}$$

3.2 Memory Transformer Network

Existing works only use exemplars to prevent catastrophic forgetting when training the model classifier g. Our work is based on the observation that, since we have to maintain the memory of exemplars, we can also utilize the exemplar memory to improve our predictions.

We define the memory by matrix V, an $M \times d$ matrix containing the feature vectors, extracted for each exemplar with a feature extractor f. We drop the task indices t here for simplicity. Typically, the memory V is used to rehearse the information from the previous tasks while learning the model parameters θ through the training process in incremental learning. The memory has no impact during the actual prediction, which follows the distribution $p_\theta(y|\mathbf{q})$, where y is the class label given the corresponding query vector $\mathbf{q}$.

However, given the catastrophic forgetting effect exhibited by neural networks when learning incrementally, we instead use V explicitly in the prediction process. That is, we model the output label distribution as $p_\theta(y|\mathbf{q},V)$, which now depends on the memory V as well as the query x. To incorporate the exemplar memory V into the prediction process, we add a layer between feature extraction and logit prediction, such that we adapt the features for the input $\mathbf{q}$ and the exemplar memory V: $\tilde{\mathbf{q}} = h(\mathbf{q},V)$. The output $\tilde{\mathbf{q}}$ is then passed to a classifier to generate logits: $\mathbf{z} = g(\tilde{\mathbf{q}})$. The MTN model is shown in Figure 2.

The transformer architecture [33] is a natural candidate for h. Transformers map a (possibly ordered) set of input features to a set of output features, where each output feature is dependent on all the input features through self-attention. This fits the modeling requirement for incorporating the exemplar memory set V into the prediction process. We want the adapted features to be able to attend to $\mathbf{q}$ and also to the exemplar memory features V.

The exemplar memory set V can grow to be of the order of tens of thousands of exemplars. Therefore, we only provide a subset of V as context for a given input $\mathbf{q}$. To choose a subset of V for given input $\mathbf{q}$, we first find the k-nearest-neighbors (kNNs) $\mathbf{q}$ from the memory V. Thus the transformer is tasked with performing a local attention of the input feature vector $\mathbf{q}$, dependent on its local neighborhood in the feature space.

4 Experiments

In this section, we first describe the experimental setup and implementation details. Then we compare our method to existing approaches for class-incremental learning. We also present detailed ablation studies on MTN showing the impact of different design choices.

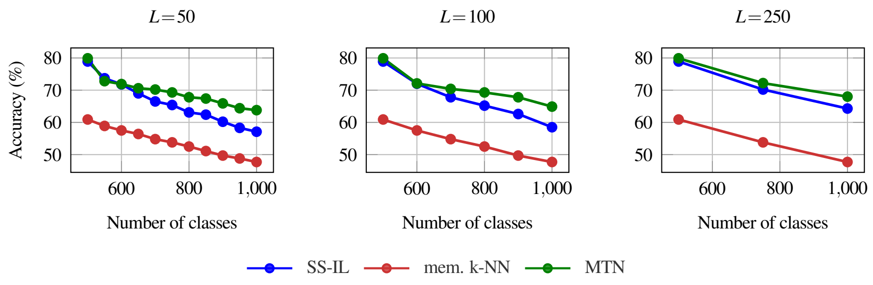

Figure 3: **Task accuracy.** Top-1 accuracy after each task is shown for MTN and baselines in ImageNet-1k dataset with SimCLR features for different number of classes per task (L).

4.1 Experimental Setup

Datasets. We assess MTN for class-based incremental learning separately on two datasets: ImageNet-1K [8] and Google Landmarks-1K [1]. Following standard practices [26], we use the *average incremental accuracy* as the evaluation metric, which is the top-1 accuracy averaged over T tasks. We now describe both datasets in more detail.

ImageNet-1k contains $C = 1000$ classes, 1.2 million training images, and 50k validation images. We follow the work of Hou *et al.* [14] and create the incremental tasks using the following approach. The first task contains half of the classes, *i.e.* $C/2 = 500$ classes. We then divide rest of the classes into tasks of $L = 50$, $L = 100$ or $L = 250$ classes.

Google Landmarks-1K is a subset of $C = 1000$ classes from Google Landmarks v2 [35], which originally contains 203,094 classes. We follow the work of Ahn *et al.* [1], and sample 1000 classes in the order of largest sample of images per class. Similarly to ImageNet-1K, the first task, contains 500 classes, and the other tasks are divided into $L = 50$, $L = 100$ or $L = 250$ classes.

Feature representations. In addition to the features learned from scratch with ResNet-18 backbone, we also use *fixed* features rather than back-propagating through the entire backbone, similarly to [16, 40]. However, we argue that using features that were pretrained for classification in a supervised way [16, 40] might confound our results since the backbone would have knowledge about all tasks at all times. For this reason, we choose to use representations obtained from no (or weak) supervision [6, 25]. Actually, these representations have been shown to be generic and excellent "off-the-shelf" descriptors, performing well on a large variety of tasks and datasets without any fine-tuning [3, 42].

In practice, we use CLIP [25] (ViT-B/32) features of $d = 512$, and SimCLR [6] (ResNet-50) features of $d = 2048$ as *fixed* features. For *learned* features, we use a ResNet-18 ($d = 512$) backbone learned with SS-IL [1] and updated at each task. We learn the classifier separately after the feature extractor is learned and its weights are frozen.

Implementation details. We train MTN with SGD optimizer, a learning rate of 0.1, weight decay of 0.0001 and momentum of 0.9. We train the model for 10 epochs per task. We use a batch size of 128, but also add 32 samples from the previous tasks to each batch [1]. We use the Ringbuffer method [5] to select the exemplars and construct the memory. There are M vectors in the memory at any given point. As new classes become available, some of the exemplars from the existing classes are removed to maintain a memory of size M. We typically set the memory size to $M = 20k$ and select top $k = 10$ nearest neighbors of each query as the input to the transformer. Our default setting is a multi-head self-attention transformer with 4 layers, 4 attention heads, and the output channel dimension of 128. We will release code and checkpoints to reproduce our results.

Method	Backbone	Memory usage: Train.	Pred.	ImageNet-1k 50	100	250	Landmarks-1K 50	100	250
PODNet [9]	R18$_{\text{(Scratch)}}$	✓		64.1	67.0	-	-	-	-
CCIL [23]†	R18$_{\text{(Scratch)}}$	✓		**65.8**	**67.6**	**69.1**	40.1	49.7	55.4
SS-IL [1]†	R18$_{\text{(Scratch)}}$	✓		57.0	62.5	67.1	54.9	60.0	64.7
MTN	🔒R18$_{\text{(SS-IL)}}$	✓	✓	60.8	64.8	68.4	**59.4**	**63.1**	**66.5**
mem. k-NN†	🔒-R50$_{\text{(SimCLR)}}$		✓	53.8	53.9	54.2	30.0	30.1	30.2
BIC [38]†	🔒-R50$_{\text{(SimCLR)}}$	✓		34.3	43.9	56.9	17.4	26.5	38.0
iCARL [26]†	🔒-R50$_{\text{(SimCLR)}}$	✓	✓	56.5	57.0	58.4	27.3	27.7	28.5
LUCIR [14]†	🔒-R50$_{\text{(SimCLR)}}$	✓		60.1	63.0	68.7	48.4	50.9	55.3
SS-IL [1]†	🔒-R50$_{\text{(SimCLR)}}$	✓		66.1	67.5	71.2	46.2	46.6	47.9
MTN	🔒-R50$_{\text{(SimCLR)}}$	✓	✓	**69.5**	**70.7**	**73.3**	**51.0**	**52.3**	**55.4**
mem. k-NN†	🔒-ViT-B$_{\text{(CLIP)}}$		✓	47.7	47.7	48.1	35.8	35.9	36.1
BIC [38]†	🔒-ViT-B$_{\text{(CLIP)}}$	✓		32.9	43.2	57.5	25.3	35.6	47.2
iCARL [26]†	🔒-ViT-B$_{\text{(CLIP)}}$	✓	✓	52.3	53.2	54.7	36.5	36.9	37.8
LUCIR [14]†	🔒-ViT-B$_{\text{(CLIP)}}$	✓		51.6	55.7	63.3	46.2	48.8	53.8
SS-IL [1]†	🔒-ViT-B$_{\text{(CLIP)}}$	✓		63.9	65.7	69.7	52.4	52.8	54.4
MTN	🔒-ViT-B$_{\text{(CLIP)}}$	✓	✓	**67.4**	**68.9**	**71.2**	**54.1**	**55.5**	**57.6**

Table 1: **Comparison with existing works.** We report the average incremental accuracy of MTN and other methods. We vary the number of classes in each task after the base task ($L = 50, 100, 250$). Reported numbers use different architectures: ResNet-18 ("R18"), ResNet-50 pre-trained with SimCLR ("R50$_{\text{(SimCLR)}}$") or ViT-Base/32 pretrained with CLIP ("ViT-B$_{\text{(CLIP)}}$"). 🔒 means that the features are kept fixed when learning the classifier. We specify when the memory is used during training as a source of additional data ("Train.") and explicitly during the prediction process ("Pred"). †: other methods run by us.

4.2 Comparison with Existing Works

We demonstrate the effectiveness of MTN on ImageNet-1k and Landmarks-1k, two standard incremental learning benchmarks in Table 1. For fair comparisons and to avoid confounding factors, we re-implement different methods [1, 14, 23, 26, 38] in a similar setup as MTN, i.e. with fixed pre-trained features from SimCLR and CLIP.

We propose and report a simple baseline, "mem. k-NN", that performs classification based on nearest neighbor search within the memory of exemplars. We set $k = 10$ for "mem. k-NN", as it gives the highest performance. We see that our MTN approach performs much better than the "mem. k-NN" baseline, $+21\%$ with SimCLR features (🔒 -R50$_{\text{(SimCLR)}}$), demonstrating the effectiveness of learning *how* to aggregate information from the memory for prediction.

Overall, we observe in Table 1 that MTN outperforms other existing methods by significant margins when trained on top of fixed representations and sets a new state-of-the-art for class-incremental learning on ImageNet-1k. The comparison with SS-IL is particularly interesting as we re-use the framework and loss proposed by SS-IL to train MTN, as described in Section 3.1. Concretely, the only difference between SS-IL and ours is the architecture of the classifier; SS-IL uses a linear layer while we use a transformer conditioned on the elements of the memory bank. We observe in Table 1 that MTN improves over SS-IL, showing the effectiveness of our prediction process based on memory transformer.

Consistently over all approaches, we see that using SimCLR features perform better on ImageNet-1k than Landmarks-1k. We hypothesize that this is due to the fact than ResNet-50 was pre-trained on ImageNet-1k dataset (without labels) and hence the representations are better suited for classification on this dataset whereas there is a domain discrepancy when applied to Landmarks-1k.

We also show that MTN can easily be applied to features learned from scratch. For ResNet-18

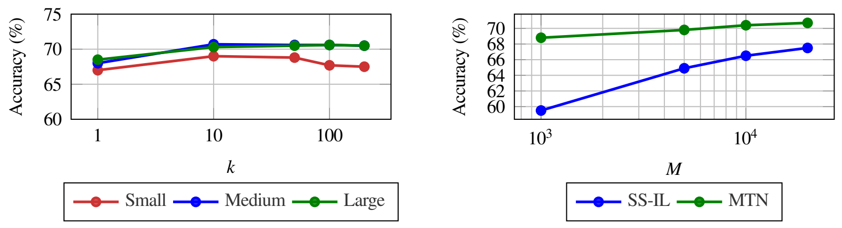

Figure 4: **Left: Impact of *k* and the architecture.** The average incremental accuracy for MTN is reported for different k and transformer architectures with $L = 100$ classes per task on ImageNet-1k. **Right: Impact of the memory size *M*.** Average incremental accuracy is shown for MTN and SS-IL for $L = 100$ classes per task on ImageNet-1k.

experiments, we train the feature extractor from scratch using SS-IL, and freeze the feature extractor at the end of each task. We then learn MTN as the classifier. Table 1 shows that MTN trained on top of ResNet-18 SS-IL features (🔒-R18$_{\text{(SS-IL)}}$) achieves up to $+4\%$ relative improvement compared to SS-IL without MTN. MTN applied on top of SS-IL does not outperform more recent baselines on ImageNet-1k, but we note that MTN architecture is not specific to SS-IL, can be applied to any other number of approaches, such as PODNet [9] or CCIL [23]. On the other hand, MTN applied on SS-IL features achieves a new state-of-the-art for Landmarks-1k dataset.

In Figure 3, we show the accuracy on all seen classes after each task. We report results for MTN and two baselines: "mem. k-NN" (that uses elements of the memory for prediction based on nearest neighbors) and SS-IL [1] which is trained similarly as ours but with a linear classifier instead of the MTN. We see that the accuracy for the SS-IL baseline starts at a similar level to MTN, but decreases at a faster pace as tasks are seen, which shows that it severely suffers from catastrophic forgetting. On the other hand, the accuracy of MTN does not drop quickly, resulting in a higher overall incremental accuracy.

4.3 Ablation studies

In this section, we evaluate different design choices of MTN like the number of nearest neighbors to select from the memory, details of the transformer architecture or the use of positional encoding. We also propose a qualitative evaluation of what is learned by MTN. All experiments are conducted with the SimCLR features (🔒-R50$_{\text{(SimCLR)}}$).

The number of kNN. We vary k the number of nearest neighbors to input to MTN. More precisely, the hyperparameter k is used to obtain the k-NN list $\text{NN}(\mathbf{q}; V)$ for a given query vector $\mathbf{q}$. As mentioned above, choosing k nearest neighbors from the memory is essential and keeps MTN efficient, as it becomes impractical to input the entire memory of $M = 20k$ vectors to the transformer. Figure 4 (Left) shows the impact of different k for MTN. We observe that the accuracy of MTN remains stable for different k as long as it is "large enough" (more than 10). We set $k = 10$ for all of our experiments.

Memory size. We vary the size of the memory M, and report the average incremental accuracy for MTN and SS-IL in Figure 4 (Right). We observe that MTN is more robust to smaller M, where as the accuracy for SS-IL rapidly decreases for smaller M. This shows that MTN is an excellent candidate for efficient incremental learning systems requiring very small memory footprint.

Transformer size. We examine the effect of the different transformer architectures in Figure 4 (Left). We compare across 3 architectures. The "small" architecture has 2 layers, 1 attention head, and the output channel dimension of 64. The "medium" architecture has 4 layers and attention

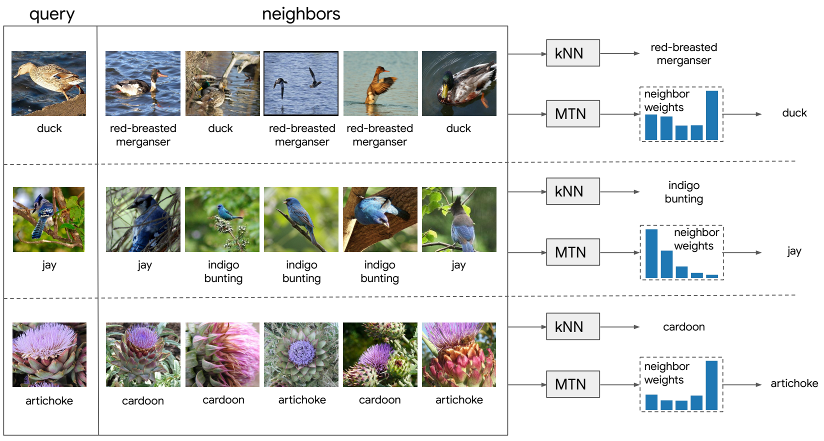

Figure 5: **Qualitative examples.** A visualization of examples from ImageNet-1K which kNN incorrectly classifies but MTN correctly classifies. We first show the kNN returned by the initial search. Note that the neighbors are ordered, with closest to the query on the left. We adapt the query and kNN features with MTN and show the similarity of the adapted features in the blue histogram, where each bar corresponds to a neighbor.

heads, and the output channel dimension of 128. The "large" architecture has 12 layers and attention heads, and the output channel dimension of 768. The "small" is less robust to the different values of k. There is no benefit going from "medium" to "large", as they perform similarly. We use the "medium" for all of our experiments.

Qualitative results. We present a qualitative visualization to inspect the behavior of MTN and get an intuition of how it learns to make more accurate predictions. In Figure 5, we look at how the similarity with different exemplars evolves after being processed by MTN. For each query image, we show its five nearest neighbors within the exemplars in the memory, i.e. before MTN processing, and report the prediction based on the vote of these neighbors. This is similar to baseline "mem. k-NN" in Table 1. Then, in the "MTN" rows of Figure 5, we show the similarity of these exemplars with the query as processed by MTN. Concretely we show in the histogram, the similarity between the MTN-adapted query vector $\mathbf{q}$, and the MTN-adapted k-NN features $\tilde{V}_{\mathrm{NN}_M(\mathbf{q})}$. We observe in Figure 5 how the features adapted by MTN have better similarity.

For sets of neighbors with the incorrect label as the majority, MTN is able to successfully up-weight neighbors that do have the correct label. We find it interesting to note that, even for images which appear at a high-level to be similar, MTN assigns most of the weight to one or two images. This suggests that MTN is able to successfully discriminate between confounding examples.

5 Conclusion

We introduce MTN, a method to improve exemplar-based class-incremental learning performance by utilizing the exemplars directly in the prediction process. We demonstrate that MTN allows to consistently improve class-incremental learning performance in various benchmarks. We show experimentally that MTN with generic pretrained features is competitive with state-of-the-art methods. In future work, we would like to evaluate MTN with unlocked features and train the full architecture end-to-end in an incremental manner.

References

[1] Hongjoon Ahn, Jihwan Kwak, Subin Lim, Hyeonsu Bang, Hyojun Kim, and Taesup Moon. Ss-il: Separated softmax for incremental learning. In *ICCV*, 2021.

[2] Eden Belouadah and Adrian Popescu. Il2m: Class incremental learning with dual memory. In *ICCV*, 2019.

[3] Mathilde Caron, Hugo Touvron, Ishan Misra, Hervé Jégou, Julien Mairal, Piotr Bojanowski, and Armand Joulin. Emerging properties in self-supervised vision transformers. In *ICCV*, 2021.

[4] Francisco M. Castro, Manuel J. Marín-Jiménez, Nicolás Guil, Cordelia Schmid, and Karteek Alahari. End-to-end incremental learning. In *ECCV*, 2018.

[5] Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K Dokania, Philip HS Torr, and M Ranzato. Continual learning with tiny episodic memories. *ICML*, 2019.

[6] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, 2020.

[7] Carl Doersch, Ankush Gupta, and Andrew Zisserman. Crosstransformers: spatially-aware few-shot transfer. In *NeurIPS*, 2020.

[8] Wei Dong, Richard Socher, Li Li-Jia, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, June 2009.

[9] Arthur Douillard, Matthieu Cord, Charles Ollion, Thomas Robert, and Eduardo Valle. Podnet: Pooled outputs distillation for small-tasks incremental learning. In *ECCV*, 2020.

[10] Thibault Févry, Livio Baldini Soares, Nicholas FitzGerald, Eunsol Choi, and Tom Kwiatkowski. Entities as experts: Sparse memory access with entity supervision. *arXiv preprint arXiv:2004.07202*, 2020.

[11] Ian J. Goodfellow, Mehdi Mirza, Da Xiao, Aaron Courville, and Yoshua Bengio. An empirical investigation of catastrophic forgetting in gradient-based neural networks. In *ICLR*, 2014.

[12] Albert Gordo, Filip Radenovic, and Tamara Berg. Attention-based query expansion learning. In *ECCV*, 2020.

[13] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. REALM: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*, 2020.

[14] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *CVPR*, 2019.

[15] Ahmet Iscen, Jeffrey Zhang, Svetlana Lazebnik, and Cordelia Schmid. Memory-efficient incremental learning through feature adaptation. In *ECCV*, 2020.

[16] Ronald Kemker and Christopher Kanan. Fearnet: Brain-inspired model for incremental learning. In *ICLR*, 2018.

[17] Zhizhong Li and Derek Hoiem. Learning without forgetting. In *ECCV*, 2017.

[18] Qun Liu and Supratik Mukhopadhyay. Unsupervised learning using pretrained cnn and associative memory bank. *arXiv preprint arXiv:1805.01033*, 2018.

[19] Yaoyao Liu, Yuting Su, An-An Liu, Bernt Schiele, and Qianru Sun. Mnemonics training: Multi-class incremental learning without forgetting. In *CVPR*, 2020.

[20] Yaoyao Liu, Bernt Schiele, and Qianru Sun. Adaptive aggregation networks for class-incremental learning. In *CVPR*, 2021.

[21] David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *NeurIPS*, 2017.

[22] Fei Mi and Boi Faltings. Memory augmented neural model for incremental session-based recommendation. In *IJCAI*, 2020.

[23] Sudhanshu Mittal, Silvio Galesso, and Thomas Brox. Essentials for class incremental learning. In *CVPR*, 2021.

[24] Tobias Plötz and Stefan Roth. Neural nearest neighbors networks. In *NeurIPS*, 2018.

[25] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *ICML*, 2021.

[26] Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. iCaRL: incremental classifier and representation learning. In *CVPR*, 2017.

[27] Anthony Robins. Catastrophic forgetting, rehearsal and pseudorehearsal. In *Connection Science*, page 7:123–146, 1992.

[28] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *ICML*, 2016.

[29] Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In *NeurIPS*, 2017.

[30] James Smith, Yen-Chang Hsu, Jonathan Balloch, Yilin Shen, Hongxia Jin, and Zsolt Kira. Always be dreaming: A new approach for data-free class-incremental learning. In *ICCV*, 2021.

[31] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

[32] Xiaoyu Tao, Xinyuan Chang, Xiaopeng Hong, Xing Wei, and Yihong Gong. Topology-preserving class-incremental learning. In *ECCV*, 2020.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[34] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *NeurIPS*, 2016.

[35] Tobias Weyand, Andre Araujo, Bingyi Cao, and Jack Sim. Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval. In *CVPR*, 2020.

[36] Chao-Yuan Wu, Yanghao Li, Karttikeya Mangalam, Haoqi Fan, Bo Xiong, Jitendra Malik, and Christoph Feichtenhofer. Memvit: Memory-augmented multiscale vision transformer for efficient long-term video recognition. *arXiv preprint arXiv:2201.08383*, 2022.

[37] Chenshen Wu, Luis Herranz, Xialei Liu, Yaxing Wang, Joost van de Weijer, and Bogdan Raducanu. Memory replay gans: learning to generate images from new categories without forgetting. In *NeurIPS*, 2018.

[38] Yue Wu, Yinpeng Chen, Lijuan Wang, Yuancheng Ye, Zicheng Liu, Yandong Guo, and Yun Fu. Large scale incremental learning. In *CVPR*, 2019.

[39] Yuhuai Wu, Markus Norman Rabe, DeLesley Hutchins, and Christian Szegedy. Memorizing transformers. In *ICLR*, 2022.

[40] Ye Xiang, Ying Fu, Pan Ji, and Hua Huang. Incremental learning using conditional adversarial networks. In *ICCV*, 2019.

[41] Shipeng Yan, Jiangwei Xie, and Xuming He. Der: Dynamically expandable representation for class incremental learning. In *CVPR*, 2021.

[42] Xiaohua Zhai, Xiao Wang, Basil Mustafa, Andreas Steiner, Daniel Keysers, Alexander Kolesnikov, and Lucas Beyer. Lit: Zero-shot transfer with locked-image text tuning. *arXiv preprint arXiv:2111.07991*, 2021.