UNSUPERVISED PROGRESSIVE LEARNING AND THE STAM ARCHITECTURE

Anonymous authors
Paper under double-blind review

ABSTRACT

We first pose the Unsupervised Progressive Learning (UPL) problem: learning salient representations from a non-stationary stream of unlabeled data in which the number of object classes increases with time. If some limited labeled data is also available, those representations can be associated with specific classes, thus enabling classification tasks. To solve the UPL problem, we propose an architecture that involves an online clustering module, called Self-Taught Associative Memory (STAM). Layered hierarchies of STAM modules learn based on a combination of online clustering, novelty detection, forgetting outliers, and storing only prototypical representations rather than specific examples. The goal of this paper is to introduce the UPL problem, describe the STAM architecture, and evaluate the latter in the UPL context.

1 INTRODUCTION

We start by posing a challenging problem, referred to as *Unsupervised Progressive Learning (UPL)* (see Figure 1). In the UPL problem, the agent observes a sequence (or stream) of unlabeled data vectors $\{\mathbf{x}_t\}_{t\in\mathbb{N}}$ with $\mathbf{x}_t \in \mathbb{R}^n$. Each vector $\mathbf{x}_t$ is associated with a class $k(x_t)$ and the vectors of class k follow a distribution F_k. The class information, however, is hidden from the agent. Occasionally, the agent may be given a small number of labeled examples of one or more classes. These examples are meant to associate "names" (i.e., class labels) with the learned representations enabling classification tasks in which the set of output classes stays constant ("persistent tasks") or increases ("expanding tasks").

We denote as L_t the set of class labels the agent has seen up to time t. This set is gradually increasing, meaning that the agent progressively learns about more classes. In the UPL context, the goal is to learn in an online manner salient representations of the unlabeled input stream so that the agent can, at any point in time t, classify a given set of test data based on the set of classes L_t it knows about so far. We require an online learner for pragmatic reasons: it would not be possible or desirable in practice to store and/or process all previously seen data and learn a new model offline every time there is a change in L_t. The online nature of the problem constraints the solution space: methods that require multiple passes over the training data and/or randomly sampled minibatches are not applicable in the UPL context.

We assume that the distribution F_k associated with class k may also change with time – but this is a slow and gradual process so that an online learner can track changes in F_k. Abrupt changes would require that the agent forgets what was previously learned about class k – we do not consider that possibility in this paper.

We do not add any further constraints on the structure of the data sequence. For instance, it is possible that the learner first observes a labeled example of class k at time t (and so $k \in L_t$) even though it has not seen any unlabeled examples of that class prior to t – this would require a transfer-learning capability so that the learner can classify k based on representations it has previously learned from other classes. Another interesting scenario is when the unlabeled data arrive in separated class phases, which are unknown to the agent, so that each phase includes data from only few new classes – this is a challenging task from the perspective of catastrophic forgetting because the learner should not forget previously learned classes for which it does not see any new examples. We consider such UPL scenarios in Section 3.

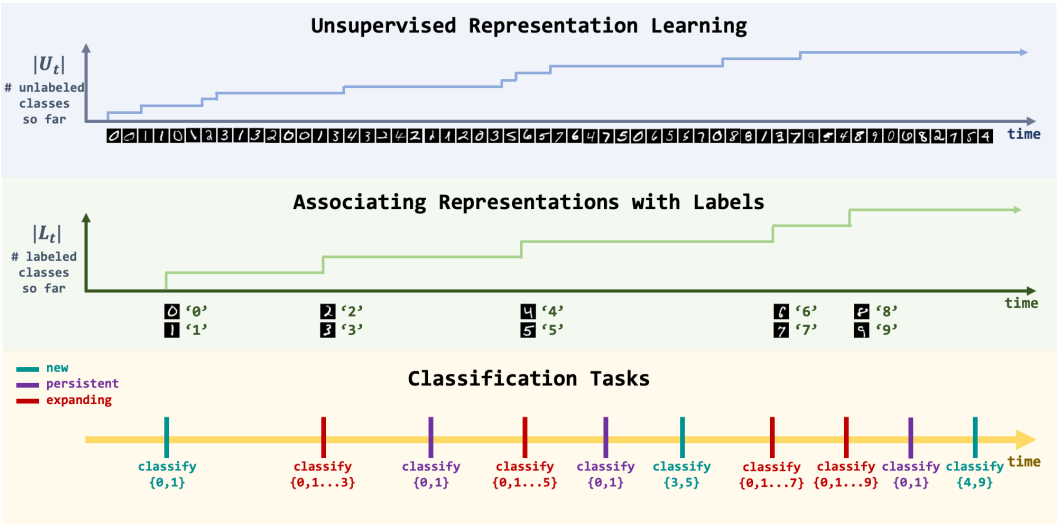

Figure 1: In the UPL problem, the agent observes a stream of unlabeled data in which the number of classes increases with time. The agent has to learn representations that distinguish the classes seen so far, without catastrophic forgetting and without the replay of previously seen data. If some limited labeled data is also available, the agent can associate the learned representations with classes, enabling persistent and expanding classification tasks.

It is plausible that UPL represents how animals learn, at least in the case of *perceptual learning* (Goldstone, 1998): they observe their environment, which is predominantly "unlabeled", and so they learn to gradually distinguish between a growing number of different object categories even when they do not have a way yet to name them. Later, some of those classes may be associated with words (in the case of humans) (Ashby and Maddox, 2005), or more generally, with a specific taste, odor, reward, fear, etc. (Watanabe *et al.*, 2001).

1.1 UPL VERSUS SIMILAR LEARNING PARADIGMS

1. Unsupervised and self-supervised learning: There have been great strides recently in learning data representations (Bengio *et al.*, 2013) via clustering (Caron *et al.*, 2018), generative models (Jiang *et al.*, 2017; Eslami *et al.*, 2016; Kosiorek *et al.*, 2018; 2019), and information theory (Hjelm *et al.*, 2019; Ji *et al.*, 2019). While these methods can learn representations without data labels, they still require prior information about the number of classes present in a given dataset (to set the number of cluster centroids or class outputs) and, in the continual learning case, they suffer from catastrophic forgetting unless some form of replay is used.

2. Few-shot learning (FSL) and Meta-learning: Such methods attempt to recognize object classes not seen in a training set with only a single (or handful) of labeled examples (Fei-Fei *et al.*, 2006; Snell *et al.*, 2017; Finn *et al.*, 2017; Ren *et al.*, 2018). FSL requires labeled data to learn good representations - whereas UPL only requires labeled data to associate already learned representations with new classes.

3. Semi-supervised learning (SSL): SSL addresses scarcity of available labeled data for model training by leveraging large amounts of unlabeled training data to boost performance (Springenberg, 2015; Oliver *et al.*, 2018; Miyato *et al.*, 2018; Kingma *et al.*, 2014). SSL requires both labeled and unlabeled data during the training process and in most cases it needs to store and process the labeled data repeatedly during the training process.

4: Continual learning (CL): Most CL methods rely on labeled data and knowledge of which tasks are learned or performed at any point in time(Hsu *et al.*, 2018; Parisi *et al.*, 2019; Kemker *et al.*, 2018; van de Ven and Tolias, 2019; Lopez-Paz and Ranzato, 2017). Further, the most effective CL mechanisms address catastrophic forgetting using stored examples or generative replay (Gepperth and Karaoguz, 2017; Shin *et al.*, 2017).

5. Transfer learning (TL): Such methods require pre-training on a large labeled dataset and so they are not applicable to UPL (Yosinski *et al.*, 2014).

6. Progressive learning/networks: In (labeled) progressive learning, a supervised classification model must be able to learn in an online manner without prior knowledge of the number of classes (Venkatesan and Er, 2016; Rusu *et al.*, 2016). However, the existing approaches in this area require supervision when a new class appears, and they suffer from catastrophic forgetting if they do not see data from a previous class for long periods of time.

2 STAM ARCHITECTURE AND LEARNING ALGORITHM

The learning approach that we pursue in this work is based on online clustering, novelty detection, two separate short-term and long-term memories, and storing only prototypical representations rather than specific examples. In the following, we describe the STAM architecture as a sequence of its seven major components.

The notation is summarized in Appendix A. The image preprocessing pipeline is minimal and is described in Appendix B. The reasons we decided to *not* pursue a deep learning approach are discussed in Appendix C.

1. Hierarchy of increasing receptive fields: An input vector $\mathbf{x_t} \in \mathbb{R}^{\mathbf{n}}$ (an image in all subsequent examples) is analyzed through a hierarchy of Λ layers. Instead of neurons or hidden-layer units, each layer consists of STAM units – in its simplest form a STAM unit functions as an online clustering module. Each STAM processes one $\rho_l \times \rho_l$ *patch* (subvector) of the input at that layer. The patches are overlapping, with a small stride (set to one pixel in our experiments), to accomplish translation invariance (similar to CNNs). The patch dimension ρ_l increases in higher layers – the idea is that the first layer learns the smallest and most elementary patterns while the top layer learns the largest and most complex patterns.

2. Online clustering: Every patch of each layer is clustered, in an online manner, to a set of centroids. These time-varying centroids form the *prototypical patterns* that the STAM architecture gradually learns at that layer. All STAM units of layer l share the same set of centroids $C_l(t)$ – again for translation invariance.[1] Given the m'th input patch $\mathbf{x_{l,m}}$ at layer l, the nearest centroid of C_l selected for $\mathbf{x_{l,m}}$ is

$$\mathbf{c_{l,j}} = \arg\min_{c \in C_l} d(\mathbf{x_{l,m}}, \mathbf{c}) \tag{1}$$

where $d(\mathbf{x_{l,m}}, \mathbf{c})$ is the Euclidean distance between the patch $\mathbf{x_{l,m}}$ and centroid $\mathbf{c}$.[2] The selected centroid is updated based on a learning rate parameter α, as follows:

$$\mathbf{c_{l,j}} = \alpha\, \mathbf{x_{l,m}} + (1-\alpha)\mathbf{c_{l,j}}, \quad 0 < \alpha < 1 \tag{2}$$

A higher α value makes the learning process faster but less predictable. We do not use a decreasing value of α because the goal is to keep learning in a non-stationary environment rather than convergence to a stable centroid. If the centroid $\mathbf{c_{l,j}}$ is selected by more than one patches of the same input, the centroid is updated based on the closest patch to that centroid.

An online clustering algorithm that is similar to our approach (and asymptotically equivalent to k-means) can be implemented with a simple recurrent neural network of excitatory and inhibitory spiking neurons using strictly Hebbian learning, as shown recently (Pehlevan *et al.*, 2017).

3. Novelty detection: When an input patch $\mathbf{x_{l,m}}$ at layer l is significantly different than all centroids at that layer (i.e., its distance to the nearest centroid is a statistical outlier), a new centroid is created in C_l based on $\mathbf{x_{l,m}}$. We refer to this event as *Novelty Detection (ND)*. This function is necessary so that the architecture can learn centroids associated with new classes after they appear in the unlabeled data stream.

To do so, we estimate in an online manner the distribution of the Euclidean distance between input patches and their nearest centroid (separately for each layer). We sample a randomly chosen patch from each input vector, only considering the last 1000 inputs. The novelty detection threshold at layer l is denoted by $\hat{D}_l$ and it is defined as the 95-th percentile ($\beta = 0.95$) of the distance distribution.

[1] We drop the time index t from this point on but it is still implied that the centroids are dynamically learned over time.

[2] We have also experimented with the L1 distance metric with only minimal differences.

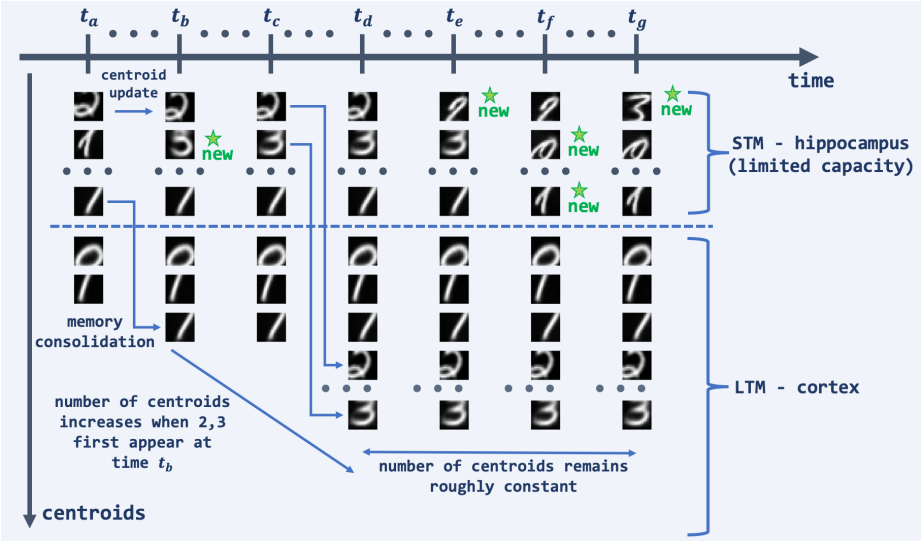

Figure 2: A hypothetical pool of STM and LTM centroids visualized at seven time instants. From t_a to t_b, a centroid is moved from STM to LTM after it has been selected θ times. At time t_b, unlabeled examples from classes '2' and '3' first appear, triggering novelty detection and new centroids are created in STM. These centroids are moved into LTM by t_d. From t_d to t_g, the pool of LTM centroids remains the same because no new classes are seen. The pool of STM centroids keeps changing when we receive "outlier" inputs of previously seen classes. Those centroids are later replaced (Least-Recently-Used policy) due to the limited capacity of the STM pool.

4. Dual-memory organization: New centroids are stored temporarily in a *Short-Term Memory (STM)* of limited capacity Δ (separate for each layer). Every time a centroid is selected as the nearest neighbor of an input patch, it is updated based on (2). If an STM centroid $\mathbf{c_{l,j}}$ is selected $s_{l,j} > \theta$ times, it is copied to the *Long-Term Memory (LTM)* for that layer. We refer to this event as *memory consolidation*. The LTM has (practically) unlimited capacity and the learning rate is much smaller (in our experiments, set to zero).

This memory organization is inspired by the Complementary Learning Systems framework (Kumaran *et al.*, 2016), where the STM role is played by the hippocampus and the LTM role by the cortex. This dual-memory scheme is necessary to distinguish between infrequently seen patterns that can be forgotten, and new patterns that are frequently seen after they first appear.

We initialize the pool of STM centroids at each layer using randomly sampled patches from the unlabeled stream (a single patch from each image to maximize diversity).

When the STM pool of centroids at a layer is full, the introduction of a new centroid (created through novelty detection) causes the removal of an earlier centroid. We use the Least-Recently Used (LRU) policy to remove atypical centroids that have not been recently selected by any input. Figure 2 illustrates this dual-memory organization.

5. Associating centroids with classes: Suppose that we have seen some labeled examples $X_L(t)$ from a set of classes $L(t)$ up to time t. In the UPL context, we only use such labeled examples to associate existing LTM centroids at time t (learned strictly from unlabeled data) with the set of classes in $L(t)$.

Given a labeled example of class k, suppose that there is a patch $\mathbf{x}$ in that example for which the nearest centroid is $\mathbf{c}$. That patch contributes the following association between centroid $\mathbf{c}$ and class k:

$$f_{\mathbf{x},\mathbf{c}}(k) = e^{-d(\mathbf{x},\mathbf{c})/\bar{D}_l} \tag{3}$$

where $\bar{D}_l$ is a normalization constant (calculated as the average distance between input patches and centroids).

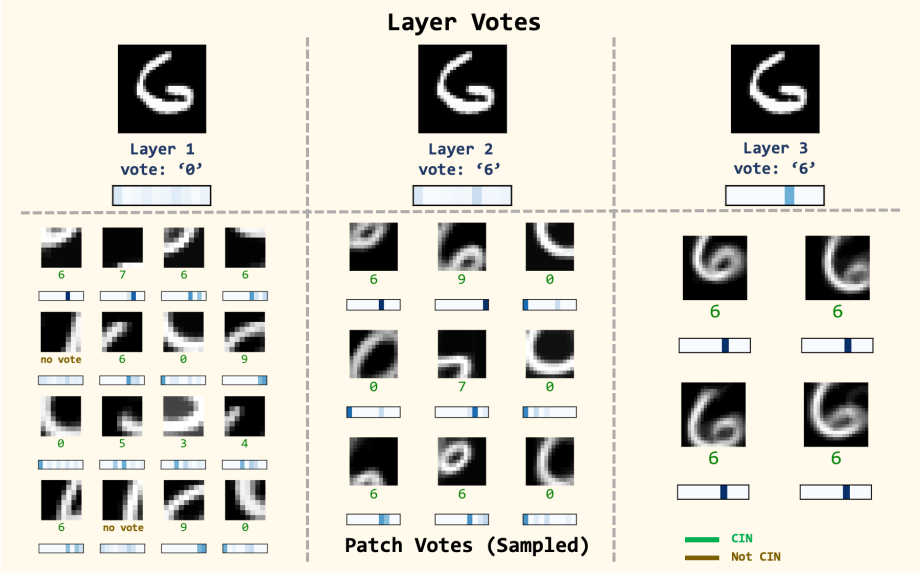

Figure 3: An example of the classification process. Every patch (at any layer) that selects a CIN centroid votes for the single class that has the highest association with. These patch votes are first aggregated at each layer. The final inference is the class with the highest vote across all layers.

The *class-association vector* $\mathbf{g_c}$ between centroid $\mathbf{c}$ and any class k is computed aggregating all such associations, across all labeled examples in X_L:

$$g_{\mathbf{c}}(k) = \frac{\sum_{\mathbf{x} \in X_L(k)} f_{\mathbf{x},\mathbf{c}}(k)}{\sum_{k' \in L(t)} \sum_{\mathbf{x} \in X_L(k')} f_{\mathbf{x},\mathbf{c}}(k')}, \quad k = 1 \dots L(t) \tag{4}$$

Note that $\sum_k g_{\mathbf{c}}(k)$=1.

6. Class informative centroids: If a centroid is associated with only one class k ($g_{\mathbf{c}}(k) = 1$), only labeled examples of that class select that centroid. At the other extreme, if a centroid is equally likely to be selected by examples of any labeled class, ($g_{\mathbf{c}}(k) \approx 1/|L(t)|$), the selection of that centroid does not provide any significant information for the class of the corresponding input.

We identify the centroids that are *Class INformative (CIN)* as those that are associated with at least one class more than expected by chance. Specifically, a centroid $\mathbf{c}$ is CIN if

$$\max_{k \in L(t)} g_{\mathbf{c}}(k) > \frac{1}{|L(t)|} + \gamma \tag{5}$$

where $\frac{1}{|L(t)|}$ is the chance term and γ is an additional significance term.

7. Classification using a hierarchy of centroids: At test time, we are given an input $\mathbf{x}$ of class $k(\mathbf{x})$ and infer its class as $\hat{k}(\mathbf{x})$. The classification task is a "biased voting" process in which every patch of $\mathbf{x}$, at any layer, votes for a single class as long as that patch selects a CIN centroid.

Specifically, if a patch $\mathbf{x_{l,m}}$ of layer l selects a CIN centroid $\mathbf{c}$, then that patch votes $v_{l,m}(k) = \max_{k \in L(t)} g_{\mathbf{c}}(k)$ for the class k that has the highest association with $\mathbf{c}$, and zero for all other classes. If c is *not* a CIN centroid, the vote of that patch is $v_{l,m}(k) = 0$ for all classes.

The vote of layer l for class k is the average vote across all patches in layer l (as illustrated in Figure 3):

$$v_l(k) = \frac{\sum_{m \in M_l} v_{l,m}(k)}{|M_l|} \tag{6}$$

where M_l is the set of patches in layer l. The final inference for input $\mathbf{x}$ is the class with the highest cumulative vote across all layers:

$$\hat{k}(\mathbf{x}) = \arg\max_{k'} \sum_{l=1}^{\Lambda} v_l(k) \tag{7}$$

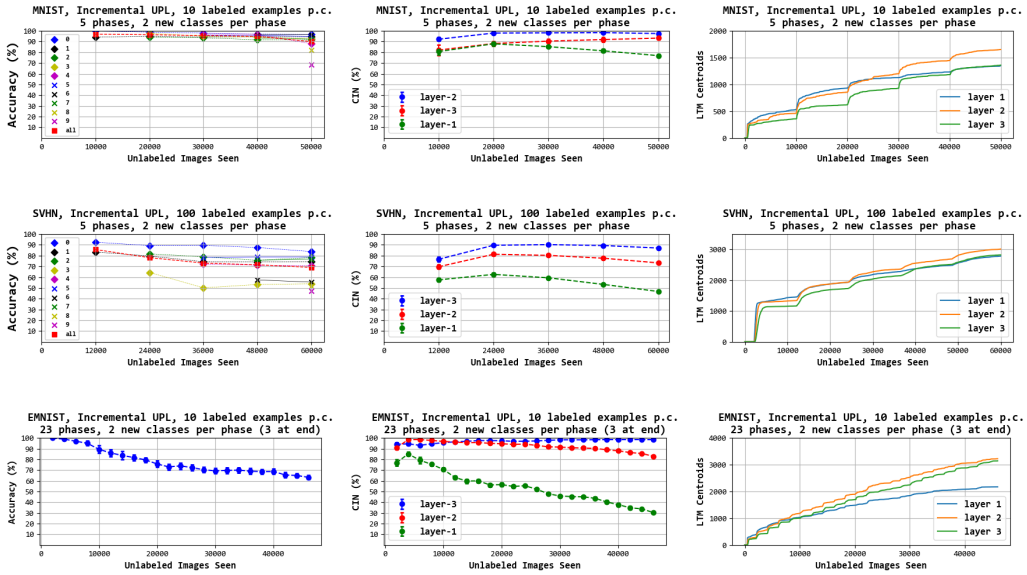

Figure 4: Incremental UPL evaluation for MNIST (row-1), SVHN (row-2), and EMNIST (row-3). Per-class and average classification accuracy (left); number of LTM centroids over time (center); fraction of CIN centroids over time (right). The task is expanding classification, i.e., recognize all classes seen so far.

3 EVALUATION

To evaluate the STAM architecture in the UPL context, we consider two different scenarios: **Incremental UPL** and **Uniform UPL**. In the Incremental UPL case, small groups of classes appear in successive *phases*. In the following results, new classes are introduced two at a time in each phase, and they are only seen in that phase. STAMs must be able to both recognize new classes when they are first seen in the stream, and to also remember all previously learned classes without catastrophic forgetting. In the Uniform UPL case, all classes appear with equal probability in the stream. The Uniform UPL scenario is more relevant when the distribution F_k of each class k may gradually change over time. The results for the Uniform scenario are presented in Appendix D.

The classification task that we focus on in the Incremental UPL case is *expanding*, meaning that in each phase we need to classify *all* classes seen so far. Given a few labeled examples for the classes that have been present in the stream up to time t, the algorithm is asked to perform object classification on a 1000-image test dataset. The datasets we evaluate on are MNIST (Lecun *et al.*, 1998), EMNIST (balanced split with 47 classes) (Cohen *et al.*, 2017), and SVHN (Netzer *et al.*, 2011).

For each classification task, we average results over five trials (different unlabeled data streams). In each trial, we have a randomly sampled hold-out set of 1500 images. Then, we sample from the remaining data to form the unlabeled stream. We perform each classification task five times, using randomly sampled test inputs from the hold-out set. So, each result is the average of 25 classification evaluations.

We use a 3-layer STAM hierarchy – all hyperparameters values are reported in Appendix A. The robustness of the results as we vary these hyperparameter values is shown in Appendix E.

3.1 INCREMENTAL UPL

As we introduce new classes to the incremental UPL stream (see Figure 4), the architecture recognizes previously learned classes without any major degradation in classification accuracy (left column). The average accuracy per phase is decreasing, which is due to the increasingly difficult expanding classification task. For EMNIST, we only show the average accuracy because there are 47 total classes. In all datasets, we observe that layer-3 (corresponding to the largest receptive field) contains the highest fraction of CIN centroids (center column). The ability to recognize new classes

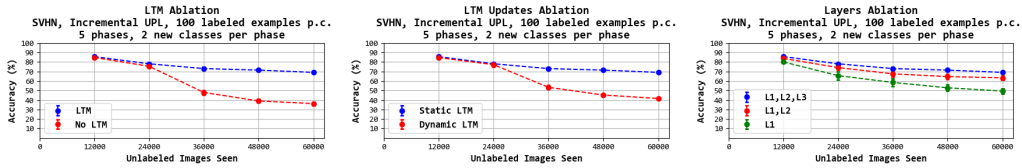

Figure 5: Ablation study: A STAM architecture without LTM (left), a STAM architecture in which the LTM centroids are adjusted with the same learning rate α as in STM (center), and a STAM architecture with removal of layers (right)

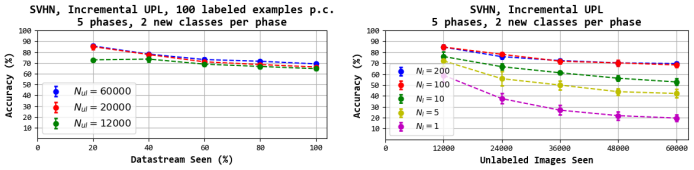

Figure 6: The effect of varying the amount of unlabeled data in the entire stream (left) and labeled data per class (right).

is perhaps best visualized in the LTM centroid count (right column). During each phase the LTM count stabilizes until a sharp spike occurs at the start of the next phase when new classes are introduced. This reinforces the claim that the LTM pool of centroids (i) is stable when there are no new classes, and (ii) is able to recognize new classes via novelty detection when they appear. In the EMNIST experiment, as the number of classes increases towards 47, we gradually see fewer "spikes" in the LTM centroids for the lower receptive fields, which is expected given the repetition of patterns at that small patch size. However, the highly CIN layer-3 continues to recognize new classes and create centroids, even when the last few classes are introduced.

3.2 Ablations

Several ablations are presented in Figure 5. On the left, we remove the LTM capabilities and only use STM centroids for classification. During the first two phases, there is little (if any) difference in classification accuracy. However, we see a clear dropoff during phases 3-5. This suggests that, without the LTM mechanisms, patterns from classes that are no longer seen in the stream are forgotten over time, and STAMs can only successfully classify classes that have been recently seen. We also investigate the importance of having static LTM centroids rather than dynamic centroids (center). Specifically, we replace the static LTM with a dynamic LTM in which the centroids are adjusted with the same learning rate parameter α, as in STM. The accuracy suffers drastically because the introduction of new classes "takes over" LTM centroids of previously learned classes, after the latter are removed from the stream. Similar to the removal of LTM, we do not see the effects of "forgetting" until phases 3-5. Note that the degradation due to a dynamic LTM is less severe than that from removing LTM completely. Finally, we look at the effects of removing layers from the STAM hierarchy (right). We see a small drop in accuracy after removing layer 3, and a large drop in accuracy after also removing layer 2. The importance of having a deeper hierarchy would be more pronounced in datasets with higher-resolution images or videos, potentially showing multiple objects in the same frame. In such cases, CIN centroids can appear at any layer, starting from the lowest to the highest.

3.3 Effect of unlabeled and labeled data

We next examine the effects of unlabeled and labeled data on the STAM architecture (Figure 6). As we vary the length of the unlabeled data stream (left), we see that STAMs can actually perform well even with much less unlabeled data. This suggests that the STAM architecture may be applicable even where the datastream is much shorter than in the experiments of this paper. A longer stream would be needed however if there are many classes and some of them are infrequent. The accuracy "saturation" observed by increasing the unlabeled data from 20000 to 60000 can be explained based on the memory mechanism, which does not update centroids after they move to LTM. As showed in the ablation studies, this is necessary to avoid forgetting classes that no longer appear in the stream. The effect of varying the number of labeled examples per class (right) is much more pronounced.

We see that the STAM architecture can perform well above chance even in the extreme case of only a single (or small handful of) labeled examples per class.

4 RELATED WORK AND DISCUSSION

Even though clustering has been used successfully in the past as a representation learning scheme (Coates *et al.*, 2011; Coates and Ng, 2012), its effectiveness gradually drops as the input dimensionality increases (Hinneburg *et al.*, 2000; Beyer *et al.*, 1999). In the STAM architecture, we avoid this issue by clustering smaller subvectors (patches) of the input data. If those subvectors are still of high dimensionality, another approach is to reduce the *intrinsic dimensionality* of the input data at each layer by reconstructing that input using representations (selected centroids) from the previous layer – we have experimented with this approach but not included it here because it is not required in the datasets and tasks we work with in this paper.

This paper does not compare the performance of STAMs with other methods because, to the extent of our knowledge, none of the existing approaches in the literature are directly applicable to the UPL problem. In a follow-up study, we plan to adapt the most relevant approaches so that they can be applied in the UPL context. One of these approaches is Incremental Classifier and Representation Learning (iCaRL) (Rebuffi *et al.*, 2017), which learns representations and classifiers from a progressive stream of *labeled* data and stores training examples for each class. In (Huang *et al.*, 2019), the goal is to do unsupervised few-shot classification and the authors evaluate various methods in a similar manner to UPL but without learning from a non-stationary stream. FearNet (Kemker *et al.*, 2018) replaces the storage of training examples with a generative model but still processes a labeled stream for both the classifier and generator. Bayesian Gradient Descent (BGD) (Zeno *et al.*, 2018) is a method for preventing catastrophic forgetting when the learner is unaware of the task schedule and so it cannot take any special action when the input data distribution changes. A comparison of methods for continuous online sequence learning, including the Hierarchical Temporal Memory model, was conducted by Cui et al. (Cui *et al.*, 2016) – those methods do not address the UPL problem however.

We firmly believe that in order to mimic human intelligence, learning methods should be able to learn in a streaming manner and in the absence of supervision. Animals do not "save off" labeled examples to train in parallel with unlabeled data, they do not know how many classes exist in their environment, and they do not have to replay/dream periodically all their past experiences to avoid forgetting them. The proposed STAM architecture addresses the desiderata that is often associated with Lifelong Learning:

1. Online learning: STAMs constantly update their centroids with every example. There is no separate training stage, and there is no specific task for which the network optimizes the features it learns. Any tasks that require classification will of course require one or few labeled examples so that the corresponding clusters that were formed previously are now associated with the name of a class. However, STAMs do not need these labeled examples to learn efficient data representations.

2. Transfer learning: The hierarchical nature of the proposed architecture means that features learned (in an unsupervised manner) at lower-level STAMs can be reused in different tasks that higher-level STAMs perform.

3. Resistance to catastrophic forgetting: The introduction of a new class or prototype will lead to the creation of new clusters at some STAMs in the hierarchy (e.g., layer-1 STAMs will learn new elementary visual features if we start feeding them natural images instead of MNIST examples – while a STAM at a higher-level would create a new cluster when it first starts seeing examples of scooters but without affecting the cluster associated with bicycles).

4. Expanding learning capacity: The learning capacity of a STAM architecture depends on two factors: the number of STAMs and the maximum number of centroids that each STAM can store in STM and LTM. The limited capacity constraint in the STM pool requires to forget recently created centroids that have not been recently updated with new examples. The unlimited capacity of the LTM pool of centroids, on the other hand, allows the system to gradually learn an unlimited number of classes, even if it does not see examples of all classes learned earlier.

5. No direct access to previous experience: A STAM only needs to store the centroids of the clusters it has learned so far. Those centroids correspond to prototypes, allowing the STAM to generalize. All previously seen exemplars are discarded.

REFERENCES

F. Gregory Ashby and W. Todd Maddox. Human category learning. *Annu. Rev. Psychol.*, 56:149–178, 2005.

Yoshua Bengio, Aaron Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, August 2013.

Kevin S. Beyer, Jonathan Goldstein, Raghu Ramakrishnan, and Uri Shaft. When is "nearest neighbor" meaningful? In *Proceedings of the 7th International Conference on Database Theory*, ICDT '99, pages 217–235, London, UK, UK, 1999. Springer-Verlag.

Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. In *The European Conference on Computer Vision (ECCV)*, September 2018.

Adam Coates and Andrew Y Ng. *Learning feature representations with k-means*, pages 561–580. Springer, 2012.

Adam Coates, Andrew Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 215–223, 2011.

Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik. Emnist: an extension of mnist to handwritten letters. *ArXiv*, abs/1702.05373, 2017.

Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. Continuous online sequence learning with an unsupervised neural network model. *Neural Comput.*, 28(11):2474–2504, November 2016.

S. M. Ali Eslami, Nicolas Heess, Theophane Weber, Yuval Tassa, David Szepesvari, Koray Kavukcuoglu, and Geoffrey E. Hinton. Attend, infer, repeat: Fast scene understanding with generative models. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 3233–3241, USA, 2016. Curran Associates Inc.

Li Fei-Fei, R. Fergus, and P. Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, April 2006.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, pages 1126–1135. JMLR.org, 2017.

R. Geirhos, P. Rubisch, C. Michaelis, M. Bethge, F. A. Wichmann, and W. Brendel. Imagenet-trained cnns are biased towards texture; increasing shape bias improves accuracy and robustness. *International Conference on Learning Representations (ICLR)*, 2019.

Alexander Gepperth and Cem Karaoguz. Incremental learning with self-organizing maps. *2017 12th International Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization (WSOM)*, pages 1–8, 2017.

Robert L Goldstone. Perceptual learning. *Annual review of psychology*, 49(1):585–612, 1998.

Alexander Hinneburg, Charu C. Aggarwal, and Daniel A. Keim. What is the nearest neighbor in high dimensional spaces? In *Proceedings of the 26th International Conference on Very Large Data Bases*, VLDB '00, pages 506–515, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *ICLR 2019*. ICLR, April 2019.

Yen-Chang Hsu, Yen-Cheng Liu, Anita Ramasamy, and Zsolt Kira. Re-evaluating continual learning scenarios: A categorization and case for strong baselines. In *NeurIPS Continual learning Workshop*, 2018.

Gabriel Huang, Hugo Larochelle, and Simon Lacoste-Julien. Centroid networks for few-shot clustering and unsupervised few-shot classification. *CoRR*, abs/1902.08605, 2019.

X. Ji, J. Henriques, and A. Vedaldi. Invariant infromation clustering for unsupervised image classification and segmentation. In *Proceedings of the International Conference on Computer Vision (ICCV)*, 2019.

Zhuxi Jiang, Yin Zheng, Huachun Tan, Bangsheng Tang, and Hanning Zhou. Variational deep embedding: An unsupervised and generative approach to clustering. In *Proceedings of the 26th International Joint Conference on Artificial Intelligence*, IJCAI'17, pages 1965–1972. AAAI Press, 2017.

Ronald Kemker, Marc McClure, Angelina Abitino, Tyler Hayes, and Christopher Kanan. Measuring catastrophic forgetting in neural networks. *AAAI Conference on Artificial Intelligence*, 2018.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

Diederik P. Kingma, Danilo J. Rezende, Shakir Mohamed, and Max Welling. Semi-supervised learning with deep generative models. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3581–3589, Cambridge, MA, USA, 2014. MIT Press.

Adam Kosiorek, Hyunjik Kim, Yee Whye Teh, and Ingmar Posner. Sequential attend, infer, repeat: Generative modelling of moving objects. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 8606–8616. Curran Associates, Inc., 2018.

Adam R. Kosiorek, Sara Sabour, Yee Whye Teh, and Geoffrey E. Hinton. Stacked capsule autoencoders, 2019.

Dharshan Kumaran, Demis Hassabis, and James L McClelland. What learning systems do intelligent agents need? complementary learning systems theory updated. *Trends in cognitive sciences*, 20(7):512–534, 2016.

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.

David Lopez-Paz and Marc'Aurelio Ranzato. Gradient episodic memory for continual learning. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6470–6479, USA, 2017. Curran Associates Inc.

Takeru Miyato, Shin-ichi Maeda, Shin Ishii, and Masanori Koyama. Virtual adversarial training: a regularization method for supervised and semi-supervised learning. *IEEE transactions on pattern analysis and machine intelligence*, 2018.

Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

Avital Oliver, Augustus Odena, Colin Raffel, Ekin D. Cubuk, and Ian J. Goodfellow. Realistic evaluation of deep semi-supervised learning algorithms. NIPS 2018 Proceedings, 2018.

German I. Parisi, Ronald Kemker, Jose L. Part, Christopher Kanan, and Stefan Wermter. Continual lifelong learning with neural networks: A review. *Neural Networks*, 113:54 – 71, 2019.

Cengiz Pehlevan, Alexander Genkin, and Dmitri B Chklovskii. A clustering neural network model of insect olfaction. In *2017 51st Asilomar Conference on Signals, Systems, and Computers*, pages 593–600. IEEE, 2017.

Sylvestre-Alvise Rebuffi, Alexander Kolesnikov, Georg Sperl, and Christoph H. Lampert. icarl: Incremental classifier and representation learning. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR'17, pages 5533–5542, 2017.

Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. In *Proceedings of 6th International Conference on Learning Representations ICLR*, 2018.

Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

Hanul Shin, Jung Kwon Lee, Jaehong Kim, and Jiwon Kim. Continual learning with deep generative replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2990–2999. Curran Associates, Inc., 2017.

Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. *CoRR*, abs/1703.05175, 2017.

Jost Tobias Springenberg. Unsupervised and semi-supervised learning with categorical generative adversarial networks, 2015.

Gido M. van de Ven and Andreas S. Tolias. Three scenarios for continual learning. *CoRR*, abs/1904.07734, 2019.

Rajasekar Venkatesan and Meng Joo Er. A novel progressive learning technique for multi-class classification. *Neurocomput.*, 207(C):310–321, September 2016.

Takeo Watanabe, José E Náñez, and Yuka Sasaki. Perceptual learning without perception. *Nature*, 413(6858):844, 2001.

Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3320–3328. Curran Associates, Inc., 2014.

Chen Zeno, Itay Golan, Elad Hoffer, and Daniel Soudry. Task agnostic continual learning using online variational Bayes, 2018.

A STAM NOTATION AND HYPERPARAMETERS

Table 1: STAM Notation

Symbol	Description
$\mathbf{x}$	input vector.
n	dimensionality of input data
M_l	number of patches at layer l (index: $m = 1 \dots M_l$)
$\mathbf{x}_{l,m}$	m'th input patch at layer l
C_l	set of centroids at layer l
$\mathbf{c}_{l,j}$	centroid j at layer l
$d(\mathbf{x}, c)$	distance between an input vector $\mathbf{x}$ and a centroid c
$\hat{c}(\mathbf{x})$	index of nearest centroid for input x
$\tilde{d}_l$	novelty detection distance threshold at layer l
$U(t)$	the set of classes seen in the unlabeled data stream up to time t
$L(t)$	the set of classes seen in the labeled data up to time t
k	index for representing a class
$g_{l,j}(k)$	association between centroid j at layer l and class k.
$\bar{D}_l$	average distance between a patch and its nearest neighbor centroid at layer l.
$v_{l,m}(k)$	vote of patch m at layer l for class k
$v_l(k)$	vote of layer l for class k
$k(\mathbf{x})$	true class label of input x
$\hat{k}(\mathbf{x})$	inferred class label of input x

Table 2: STAM Hyperparameters

Symbol	Default	Description
Λ	3	number of layers (index: $l = 1 \dots \Lambda$)
α	0.1	centroid learning rate
β	0.95	percentile for novelty detection distance threshold
γ	0.15	used in definition of class informative centroids
Δ	see below	STM capacity
θ	30	number of updates for memory consolidation
ρ_l	see below	patch dimension

Table 3: MNIST/EMNIST Architecture

Layer	ρ_l	Δ (incremental)	Δ (uniform)
1	8	400	2000
2	13	400	2000
3	20	400	2000

Table 4: SVHN Architecture

Layer	ρ_l	Δ (incremental)	Δ (uniform)
1	10	2000	10000
2	14	2000	10000
3	18	2000	10000

B IMAGE PREPROCESSING

Given that each STAM operates on individual image patches, we perform *patch normalization* rather than image normalization. We chose a normalization operation that helps to identify similar patterns despite variations in the brightness and contrast: every patch is transformed to zero-mean, unit variance before clustering.At least for the datasets we consider in this paper, grayscale images result in higher classification accuracy than color.

We have also experimented with ZCA whitening and Sobel filtering. ZCA whitening did not work well because it requires estimating a transformation from an entire image dataset (and so it is not compatible with the online nature of the UPL problem). Sobel filtering did not work well because STAM clustering works better with filled shapes rather than the fine edges produced by Sobel filters.

C WHY NOT USE DEEP LEARNING?

Given that deep neural networks have become the new orthodoxy in machine learning, we need to address the question: *why not use a deep learning approach to solve the UPL problem? Why to rely on online clustering instead?*

1) With few exceptions, deep learning approaches have a fixed architecture and capacity (measured in terms of their hidden unit parameters) and so they are not able to automatically grow as the network is presented with new classes or tasks. Methods such as Progressive Networks (Snell *et al.*, 2017) allow the network to grow but only when instructed that a new task/class is given – and by a pre-determined growth factor. The STAM architecture, on the other hand, grows by creating new centroids only when needed and by a growth factor that is determined in a self-supervised manner by the "degree of novelty" in the data.

2) Deep learning approaches require multiple passes over the training data, and thus they would need to either store and replay previously seen data or to learn a generative model that synthesizes realistic data when needed. Especially in semi-supervised methods such as VAT (Miyato *et al.*, 2018), it is critical that the limited given labeled data are processed repeatedly. In the STAM architecture, data is never stored both because that may be impractical and because the data distribution in the UPL context may gradually shift – storing old data can be misleading.

3) A neural network (deep or shallow) learns a nonlinear embedding of the input data in a low-dimensional continuous space in which it is more efficient, presumably, to perform classification, clustering, generative modeling, or other tasks. This compression of the input dimensionality however comes at a high cost: the latent features learned by a neural net are not interpretable, they may be derived from properties that are unrelated to the causal/defining properties of the corresponding classes (Geirhos *et al.*, 2019), and they can make the network susceptible to adversarial attacks. STAMs, on the other hand, use clustering to learn common patterns at a hierarchy of increasing receptive fields. These patterns can be easily interpreted by humans because they represent prototypes (common patterns) at each layer.

4) The embedding that a neural network learns is typically assumed to be time-invariant. This is problematic in the UPL context because the number of classes or tasks increases with time, without external supervision whenever that happens. It is not clear how to gradually adjust an embedding in an online manner without any external supervision when the classes/tasks change.

5) Learning in deep neural networks requires iterative nonconvex optimization processes, such as SGD, that can get stuck in local minima. Online clustering is a much simpler computation in which we only need to compute a certain distance metric between the input vector and all existing centroids – and then to update the nearest centroid based on the corresponding input.

D UNIFORM UPL

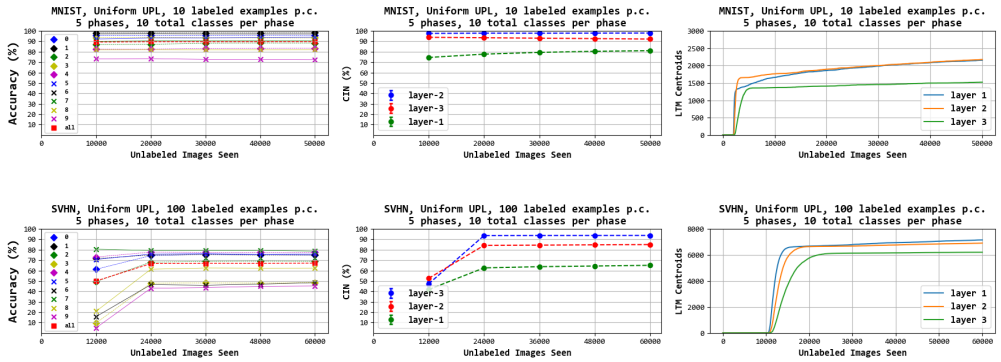

Figure 7: Uniform UPL evaluation for MNIST (row-1) and SVHN (row-2). Per-class/average classification accuracy is given at the left; the number of LTM centroids over time is given at the center; the fraction of CIN centroids over time is given at the right.

In order to examine if the STAM architecture can learn all classes simultaneously, but without knowing how many classes exist, we also evaluate the STAM architecture in a uniform UPL scenario (Figure 5). Note that LTM centroids converge to a constant value, at least at the top layer, Each class is recognized at a different level of accuracy, depending on the similarity between that class and others.

E ADDITIONAL HYPERPARAMETER SWEEPS AND ABLATIONS

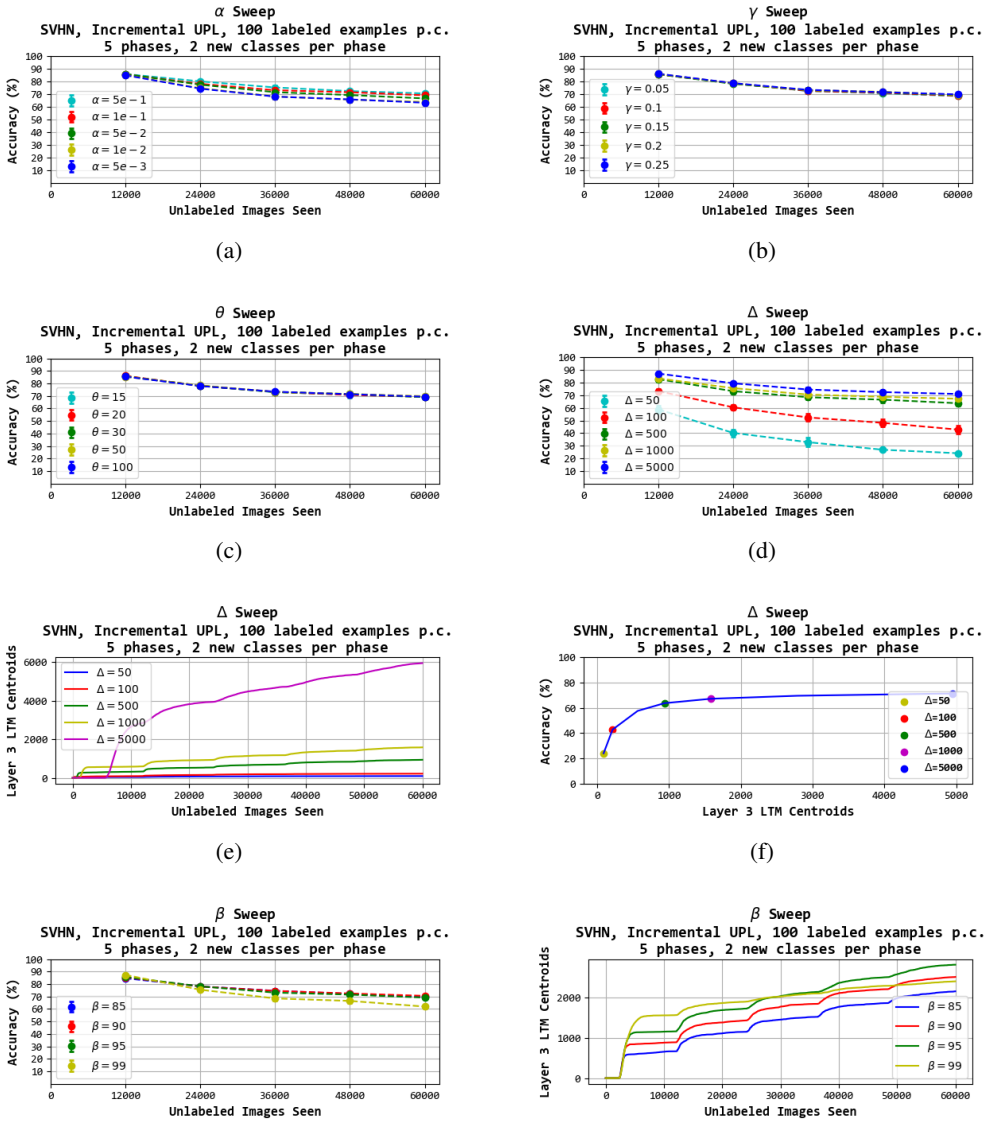

Figure 8: Hyperparameter sweeps for α, γ, θ, β, and Δ.

We examine the effects of STAM hyperparameters in Figure 8. (a) As we decrease the rate of α, we see a degradation in performance. This is likely due to the static nature of the LTM centroids - with low α values, the LTM centroids will primarily represent the patch they were intialized as. (b) As we vary the rates of γ, there is little difference in our final classification rates. This suggests that the maximum $g_{l,j}(k)$ values are quite high, which may not be the case in other datasets besides SVHN. (c) We observe that STAM is robust to changes in Θ. (d,e) The STM size Δ has a major effect on the number of learned LTM centroids and on classification accuracy. (e) The accuracy in

phase-5 for different numbers of layer-3 LTM centroids (and correspnding Δ values). The accuracy shows diminishing returns after we have about 1000 LTM centroids at layer-3. (g,h) As β increases the number of LTM centroids increases (due to a lower rate of novelty detection); if $\beta \geq 0.9$ the classification accuracy is about the same.

F COMPARISON WITH AUTO-ENCODER APPROACH

Even though the goal of this paper is not to present an extensive comparison between STAM and deep-learning methods, in this section we perform a limited such comparison with a simple autoencoder. Specifically, we train a convolutional autoencoder (CAE) with the images from the unlabeled data stream. The number of training epochs and the batch size are both set to 1, so that each unlabeled example is only used once. The encoder consists of three convolution layers with ReLU activations, embedding inputs into a 64-dimension latent space. The decoder consists of three transposed convolution layers with ReLU activations. The final layer uses linear activations and the network is trained to optimize Euclidean reconstruction error. The representations at the 64-dimension latent space are used to perform K nearest-neighbor (KNN) classification. The CAE architecture, given in Figure 10, is trained using Adam optimization (Kingma and Ba, 2014) with a learning rate of 1^{-4} and no decay.

The STAM architecture outperforms the CAE approach by a large margin (Figure 9, Incremental UPL, SVHN dataset). In follow-up work we plan to develop more sophisticated deep learning approaches for solving the UPL problem and compare them with STAMs under different conditions.

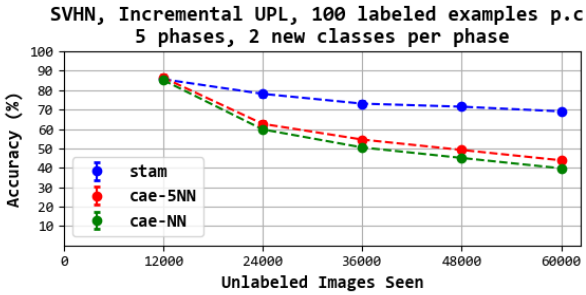

Figure 9: STAM versus a CAE-based baseline

G MEMORY FOOTPRINT ANALYSIS

The memory requirement of the STAM model can be calculated as:

$$M = \sum_{l=1}^{\Lambda} \rho_l^2 \cdot (|C_l| + \Delta) \tag{8}$$

For the 3-layer SVHN architecture with $|C_l| \approx 3000$ LTM centroids in every layer and $\Delta = 2000$, the memory footprint is $5,064,000$ pixels, equivalent to roughly 5000 grayscale SVHN digits. This memory requirement can be significantly reduced however. Figure 8(f) shows that the accuracy remains almost the same when $\Delta = 500$ and $|C_l| \approx 1000$. With these values the memory footprint reduces to about 950,000 pixels, equivalent to roughly 930 grayscale SVHN digits.

By comparison, the CAE architecture has $4,683,425$ trainable parameters, which should be stored at floating-point precision. With four bytes per weight, then STAM model would require $\frac{950000}{4683425 \times 4} \approx 5\%$ of the CAE's memory footprint. Future work can decrease the STAM memory requirement further by merging similar LTM centroids.

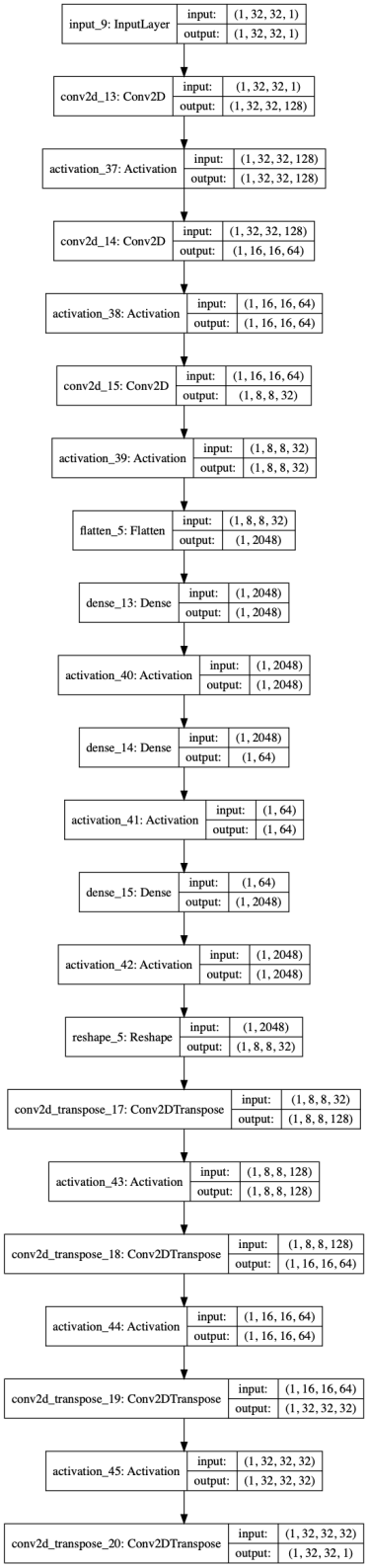

Figure 10: CAE architecture