Supplemental Material for The CLEAR Benchmark: Continual LEArning on Real-World Imagery

Zhiqiu Lin[1] **Jia Shi**[1] **Deepak Pathak**[1*] **Deva Ramanan**[1,2*]
[1]Carnegie Mellon University [2]Argo AI

1 CLEAR Details

In this section, we describe additional details about the data curation pipeline used to construct CLEAR. YFCC100M is so large that even downloading the entire dataset requires weeks and storing all the media files would take over 10TB. Instead, we construct a carefully-tuned pipeline of successive dataset *triage* stages that prunes this massive datasource into more manageable subsets. Code is available at `https://github.com/linzhiqiu/continual-learning`.

(Re)constructing the YFCC100M temporal stream: We begin by downloading all the *metadata* files released by the YFCC100M creator [1], which is manageable (around 40GB after decompression). Each line of the metadata file contains the download link for an image (though some of the links may not be available) as well as its associated metadata including the upload timestamp. Next, we download the images in the original line order of the metadata file until we successfully download 7850000 images. Note that the metadata file provided by YFCC100M creator already shuffled the order of images and therefore we could treat this subset as iid samples from the real YFCC100M distribution spanning 10 years. We only download images with still accessible URLs and valid timestamps; in particular, the upload time of the image must be strictly later than its captured time. We then sort the 7850000 images by their upload timestamps to reconstruct the YFCC100M temporal stream, and define 11 time-period buckets such that each bucket contain the same number of 713626 images. The time span for each bucket is shown in Table 1.

Text-Prompt Engineering: We use CLIP to retrieve a small labeled dataset of images of our target visual concepts from each bucket independently that span the three super-categories discussed in main paper: **Product**, **Fashion**, and **Event**. We found it helpful to **engineer** the prompt to better capture some visual concepts. In particular, we use *sub*category queries to improve the precision of CLIP-based image retrieval, as shown in Table 2 for two classes (`laptop` for **computer**, and `sweater` for **pullover**). Higher precision reduces the cost of MTurk quality assurance (QA) since fewer images need to be rejected. There may also exist better prompts for retrieving the above 10 visual concepts, for example using other *sub*categories or using `"a photo of XXX"` as the prompt as suggested by the authors of CLIP for zero shot classification [13].

Dataset Collection: Using CLIP, for each of the bucket 1^{st} to 10^{th}, we retrieve the highest-scoring 600 images for each of the 10 categories. We also gather a `background` class per bucket containing 60 lowest-scoring images of each of the 10 categories. Images co-occurred in more than one category are discarded and we replace them by new images from the remaining highest-scoring images. Finally, all 11 categories including `background` class are assembled to a single bucket of 6600 images.

Crowdsourced Quality Assurance (QA): We use Amazon MTurk (`www.mturk.com`) for crowd-sourced quality assurance. We hire 3 workers per image and remove images that (1) do not align

[1]Since the original download link is no longer available on Yahoo, we reach out to the creator and use this command to download the YFCC100M metadata files from Amazon s3: `s3cmd get -recursive s3://mmcommons .`

35th Conference on Neural Information Processing Systems (NeurIPS 2021) Track on Datasets and Benchmarks.

Bucket Index	Earliest Timestamp	Latest Timestamp
0	2004-01-01	2007-04-09
1	2007-04-09	2008-03-21
2	2008-03-21	2008-12-23
3	2008-12-23	2009-09-07
4	2009-09-07	2010-06-03
5	2010-06-03	2011-02-05
6	2011-02-05	2011-09-24
7	2011-09-24	2012-06-05
8	2012-06-05	2013-02-03
9	2013-02-03	2013-09-08
10	2013-09-08	2014-04-28

Table 1: **Time span for 11 buckets.** For each bucket, we provide its earliest timestamp and latest timestamp. The average length of each bucket is about 0.8 year. We will provide all timestamps (as well as other YFCC100M metadata) in the public dataset. Note that bucket 0^{th} is longer than other buckets, because YFCC100M has relatively fewer images uploaded during 2004 to 2006.

Super-Category	Visual Concept	Engineered Prompt (CLIP)	Sub-Category Queries
Product	**computer**	`laptop`	laptop, desktop, tablet, mouse, keyboard
Product	**camera**	`camera`	SLR camera, film camera, digital camera, phone camera
Product	**bus**	`bus`	bus interior, bus exterior
Fashion	**pullover**	`sweater`	sweater, hoodies, sweatershirt
Fashion	**dress**	`dress`	skirt, ballgown, sundress, wedding dress
Event	**racing**	`racing`	car racing, bike racing, boat racing
Event	**cosplay**	`cosplay`	anime convention, Halloween, mascot
Event	**baseball**	`baseball`	baseball game, baseball field
Event	**soccer**	`soccer`	soccer game, soccer field
Event	**hockey**	`hockey`	ice hockey, field hockey

Table 2: **The 10 dynamic visual concepts.** For each visual concept, we attach its super-category, engineered prompt for CLIP-assisted image retrieval from YFCC100M, and a non-exhaustive list of examples per visual concept.

Figure 1: **Example images from CLEAR.** For each bucket (per column), we show a random sample from 5 of the classes (`computer`, `bus`, `camera`, `hockey`, `cosplay`) in CLEAR.

with the CLIP-labeled visual concept, or (2) contain more than one visual concepts. To achieve this goal, we ask MTurk workers to select all elements that apply to an image. A glimpse of the web user interface is provided in figure 3.

Specifically, a worker will be asked: `"Select one or more elements in this image. Or select 'None of the above'."`. If the worker want more detailed instruction, there is an instruction panel to the left as shown in figure 3. The worker will be provided with 11 options, and multiple answers could be selected unless the worker already select "None of the above", in which

Figure 2: **Examples of CLEAR images in** `background` **class.** We show 3 random `background` images (per column) for each of the bucket in CLEAR.

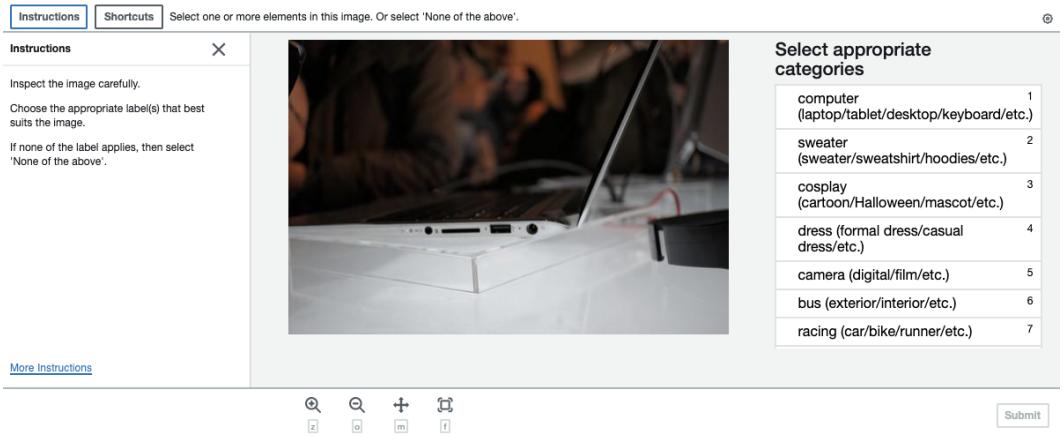


Figure 3: **MTurk user interface for image verification.** For each image, we hire 3 workers to select one or more elements in the image. The image is shown in the middle of the page; the detailed instruction is shown to the left; the options the worker could select is shown to the right. The worker could select multiple choices (unless he or she select the "None of the above" option). Note that we could not show all the available options in this figure due to the presence of a scrolling bar. Instead, we provide all the options given to worker with respect to each visual concept in Table 3.

case no other visual concept could be selected. After we receive the workers' feedback, we only keep images that have the majority (2 workers) agreeing on the single ground-truth label.

We also ask MTurk workers to identify whether those images contain sensitive contents. The web user interface for sensitive content removal is shown in figure 4. We ask the worker: `"Does this image contain sensitive content (e.g., pornography, abuse, hate speech, etc.)?"` Then the worker can choose from "Yes" or "No". We remove all images with more than 2 workers selecting "Yes" for sensitive content. This pipeline successfully surfaced three sensitive images contained in YCFF100M and we will report those cases to the creator of YFCC100M. Finally, for the remaining crowd-sourced verified images, we use random 300 images per visual concept to compose the final CLEAR dataset.

We paid Amazon MTurk $0.02 to label each image. An average worker could label 600 to 800 images per hour, so the estimated hourly wage is around $12 to $16 (not including the fee that Amazon MTurk deducted). The total amount we spent on worker compensation on the Amazon MTurk platform is $5323.

Examples of CLEAR images: Examples of images in our dataset can be found in Fig. 1 (for 5 of the classes) and Fig. 2 (for `background` class).

Future directions on dataset curation with CLIP: We hope our CLIP-assisted image retrieval and dataset curation process is simple enough to inspire future researchers to gather their own dataset

Visual Concept	MTurk Option Seen by Workers
computer	*computer (laptop/tablet/desktop/keyboard/etc.)*
camera	*camera (digital/film/etc.)*
bus	*bus (exterior/interior/etc.)*
pullover	*sweater (sweater/sweatshirt/hoodies/etc.)*
dress	*dress (formal dress/casual dress/etc.)*
racing	*racing (car/bike/runner/etc.)*
cosplay	*cosplay (cartoon/Halloween/mascot/etc.)*
baseball	*baseball (baseball/player/field/etc.)*
soccer	*soccer (soccer/player/field/etc.)*
hockey	*hockey (ice hockey/field hockey/etc.)*
background	*None of the above*

Table 3: **MTurk options for each of the 10 visual concepts plus the background class.** The worker can select either "None of the above", or select one or more answers out of the 10 visual concepts. We provide concrete examples in the prompt for each visual concept to help the workers make their choices with ease.

Figure 4: **MTurk user interface for detecting sensitive content.**

with a reduced cost. Given the superior "zero-shot" capability of CLIP on classifying Internet images, we expect CLIP to do well on most common visual concepts found in web image collections. More sophisticated prompt engineering strategies may also improve the precision of CLIP-based image retrieval.

One may also use the same design philosophy to gather a temporally-evolving image dataset with dynamic visual concepts from other domains such as Fashion [11].

Note that we only use the upload timestamps of YFCC100M images; YFCC100M also offer geo-stamps for the captured locations. These may also serve as useful and natural task boundary (regional differences) for gathering more practical continual learning dataset in a similar fashion. In particular, a concurrent work [1] uses both the time-stamps and geo-stamps of YFCC100M images to construct a continual image localization dataset.

2 Desiderata in CL Experiment Design

Our experiment design is partly inspired by the work of Farquhar et al.[4]. In particular, they criticize that existing experiment design for CL does not consider several important desiderata. In our work, we not only introduce a more realistic continual dataset, but also set up the experiments to reflect all the core desiderata brought up in [4]:

A: Cross-task resemblances. In CLEAR, the 10 temporally-sorted buckets with the same group of visual concepts form a natural sequence of 10 learning tasks that naturally and closely resemble each others.

B: Shared output head. In our experiments, we stick to image classification of the 10 visual concepts plus a `background` class and therefore there is no need to switch or modify the output head, i.e., incremental domain learning.

C: No test-time assumed task labels. In test time, the model does not know which time period a test image is from.

D: No unconstrained retraining on old tasks. We consider the scenario where there is a limited sample storage (e.g., one bucket of training images) for rehearsal purpose.

E: More than two tasks. CLEAR easily meets this desiderata as we have 10 buckets of images in CLEAR.

CLEAR further advocates an additional desiderata with our "streaming" evaluation protocol that reflects training and deploying a continual learning system in real world:

F: Train now, test in future. For streaming evaluation protocol, a model will always be evaluated on the data from future bucket. This protocol resembles training and deploying a model in real world CL scenarios. In practice, data annotation, model training, hyper-parameter tuning, and system deployment can all take a considerable amount of time. By the time the model is deployed, it will most likely be evaluated on "future" data that come from a distribution different from the one it was originally trained on. Such undeniable train-test domain shift has been largely ignored in prior work but it is crucial for real-world AI systems.

3 Evaluation Metrics

Prior works on continual learning have proposed a variety of metrics that summarize the accuracy matrix $\mathcal{R}$ in different ways [3, 4, 10], i.e. in [3]:

1. **Accuracy** averages diagonal entries as well as all elements below the diagonal.
2. **Backward Transfer** measures learning without forgetting by averaging the lower triangular entries.
3. **Forward Transfer** measures future generalization by averaging the upper triangular entries.

Formally, we calculate all five metrics as follows:

$$\textbf{Accuracy} = \frac{\sum_{i \geq j}^{N} \mathcal{R}_{i,j}}{\frac{N(N+1)}{2}} \tag{1}$$

$$\textbf{Backward Transfer} = \frac{\sum_{i > j}^{N} \mathcal{R}_{i,j}}{\frac{N(N-1)}{2}} \tag{2}$$

$$\textbf{Forward Transfer} = \frac{\sum_{i < j}^{N} \mathcal{R}_{i,j}}{\frac{N(N-1)}{2}} \tag{3}$$

$$\textbf{In-Domain Accuracy} = \frac{\sum_{i=1}^{N} \mathcal{R}_{i,i}}{N} \tag{4}$$

$$\textbf{Next-Domain Accuracy} = \frac{\sum_{i=1}^{N-1} \mathcal{R}_{i,i+1}}{N-1} \tag{5}$$

The above definitions slightly differ from [3], which calculates **Backward Transfer** by subtracting the diagonal term from each lower triangular entry below it, and then averaging. It further divides **Backward Transfer** into "Remembering" (which penalizes forgetting) and "Positive Backward Transfer" (which rewards improvement after learning on new tasks). For simplicity and consistency, we calculate **Backward Transfer** with (2), making it symmetric with **Forward Transfer**.

For iid evaluation protocol, we report all above metrics; for streaming evaluation protocol, because previous test sets have been repurposed to new train sets, we only report **Next-Domain Accuracy** and **Forward Transfer**.

4 Reservoir Sampling and Its Biased Version

In this section, we provide a detailed description and the pseudocode for the biased reservoir sampling algorithm.

"Bucket-level" reservior sampling procedure: Because in CLEAR the data comes in buckets, we treat all samples of the same bucket as having the same timestamp (we still release the precise upload date of each image to the public). In other words, we wait for all samples of the incoming buckets to arrive before we start the sampling procedure. Formally, assume we have N buckets of training data $\{S_t\}_{t=1}^{N}$ per N timestamp. The size of each bucket is $\{|S_t|\}_{t=1}^{N}$. Since we work with a buffer of fixed size k, the output of the algorithm is the replay buffer per timestamp $\{B_t\}_{t=1}^{N}$ with each $|B_t| \leq k$. The pseudocode of this bucket-level sampling procedure is given in Alg. 1.

Meanings of alpha in "bucket-level" sampling: There are two types of alpha: (1) **Fixed Alpha** that stays the same over time, and (2) **Dynamic Alpha** that changes according to the timestamp. Now in Alg. 1, for Fixed Alpha, α is still a constant non-negative real number, e.g., $\alpha \in \{0.5, 1.0, 2.0, 5.0\}$. However, for Dynamic Alpha, α changes with respect to **the number of seen samples in the stream so far**. In particular, we choose $\alpha \in \{\frac{0.25i}{k}, \frac{0.5i}{k}, \frac{0.75i}{k}, \frac{i}{k}\}$, where i is the total count of all seen samples thus far, e.g., if we are at timestamp t, then $i = \sum_{j=1}^{t} |S_j|$.

Algorithm 1: BiasedReservoirSampling($\{S_t\}_{t=1}^{N}, k, \alpha$)

Input : $\{S_t\}_{t=1}^{N}$: Incoming buckets of N timestamps
k: A fixed buffer size
α: The alpha value for weighting the probability (could be either fixed or dynamic)
Output : $\{B_t\}_{t=1}^{N}$: Training buffers of N timestamps

```
// Starting from an empty buffer
B_0 = {}
for t ← 1 to T do
    // Inherit the buffer from previous timestamp
    B_t ← B_{t-1};
    // T is a temporary buffer to store samples from current bucket S_t
    T ← {};
    // i is the total size of all seen buckets
    i ← Σ_{j=1}^{t} |S_j|;
    for s ∈ S_t do
        if |B_t| < k then
            // Buffer is not full yet so we add the new sample
            B_t ← B_t + {s} ;
        else
            p ~ Uniform(0, 1) ;
            if p ≤ α * k/i then
                // Add the new sample to temporary buffer T
                T ← T + {s} ;
        end
    end
    // Shuffle to ensure we replace random samples from B_t
    B_t ← RandomShuffle(B_t);
    // Remove first |T| samples from B_t
    B_t ← B_t[|T| : |B_t|];
    // Finally add T to B_t
    B_t ← B_t + T;
end
```

5 Additional Experiment Results

Train offline with all data with ResNet18: If we treat the entire labeled portion of CLEAR as a dataset and train on all its training data (70%) and test data (30%), then the final test accuracy is 81.1% for a ResNet18 from scratch (hyperparameters in Sec. 6). Most likely because CLEAR adopts real-world imagery, our new dataset by itself is a much more challenging dataset than MNIST or CIFAR10.

In the rest of this section, we enumerate a large set of additional shallow model experiments (linear and nonlinear two-layer MLP) that could not fit into the paper, especially on the pre-trained features with various baseline training methods and sampling strategies.

Training methods with shallow models: We adopt three simple training methods for continual training with shallow models over pre-trained features. For each timestamp:

1. **Napping**: Do not train unless for very first timestamp.
2. **From Scratch**: Train a linear layer from random initialization for each timestamp.
3. **Finetuning**: Use learned weight from previous timestamp as initialization for each timestamp.

Note that we already present in main paper some results while adopting the **Finetuning** method. In Table 4, we report iid evaluation protocol using linear classification with MoCo-YFCC-B0 features. In Table 5, we report streaming evaluation protocol using linear classification with MoCo-YFCC-B0 features. The results include both mean and standard deviation of 5 runs using different random seed per run. We notice several trends from the table: i) When we use **From Scratch** method, there is an obvious tradeoff between **Backward Transfer** and **Next-Domain/In-Domain Accuracy**. For example, when switching from uniform sampling ($\alpha = 1.0$) to selecting 75% recent samples ($\alpha = 0.75\%$) in iid protocol (Table 4), In-domain Accuracy improves from 88.4% to 89.0% and Next-domain Accuracy improves from 87.7% to 87.9%, whereas Backward Transfer drops from 88.9% to 88.3%. ii) However, when we use **Finetuning** strategy, it is always better to buffer more unseen samples from current bucket, which improves performance on all metrics. In short, with limited buffer, biased reservoir sampling that favors more-recent samples outperforms "classic" iid sampling from the data stream. That being said, future research on CLEAR could explore other sampling strategies to improve the performance under limited memory. We also include the results of unlimited buffer size (termed as "**Cumulative**") in the tables.

Pre-trained Models: For the results we present in paper, we train an unsupervised MoCo V2 model with ResNet50 backbone trained with the default hyperparameter and augmentation provided in the official codebase (`https://github.com/facebookresearch/moco`) on the entire bucket 0^{th} of around $0.7M$ images. In this section, we also provide results using features extracted with other pre-trained models. All models we used for feature extraction are listed below:

- **CLIP** [13]: Pre-trained CLIP model (ResNet50 backbone) from official codebase[2].
- **Pretrained-ImageNet** [15]: ResNet50 trained on labeled ImageNet [3].
- **MoCo-ImageNet** [2]:ResNet50 trained on ImageNet using MoCo V2[4].
- **BYOL-ImageNet** [5]: ResNet50 trained on ImageNet with BYOL[5].
- **MoCo-YFCC-B0** [2]: ResNet50 we trained on entire 0^{th} bucket with MoCo V2.

We also report the results of all 5 pre-trained models using both linear classification (Table 6) and non-linear classification (Table 7) under **iid evaluation protocol**. Similarly, we report the results of those pre-trained models using both linear classification (Table 8) and non-linear classification (Table 9) under **streaming evaluation protocol**.

Discussion: Out of all pre-trained models, Pretrained-ImageNet works the best and consistently outperform the rest of the pre-trained models on all metrics. MoCo-ImageNet and BYOL-ImageNet

[2] https://github.com/openai/CLIP

[3] `https://pytorch.org/vision/stable/models.html`

[4] `https://github.com/open-mmlab/OpenSelfSup/blob/master/docs/MODEL_ZOO.md`

[5] `https://github.com/open-mmlab/OpenSelfSup/blob/master/docs/MODEL_ZOO.md`

also works better than MoCo-YFCC-B0, presumably because they have been trained on images overlapping with the period of CLEAR. CLIP has the worst performance most likely because CLIP features are not well suited for transferring to other tasks [13] besides zero-shot classification. Nonetheless, no matter the pre-trained models we used, biased reservoir sampling (e.g., alpha = 0.75 * i/k or 1.0 * i/k) that rewards more recent samples consistently outperform naive reservoir sampling (e.g., alpha = 1.0) that treats the entire stream as an iid distribution.

Network	Buffer Size	Method	Alpha	Evaluation Metrics				
				In-Domain Acc	Next-Domain Acc	Acc	Backward Transfer	Forward Transfer
Linear	N/A	Nap	N/A	86.7% ± .0%	86.1% ± .1%	87.7% ± .2%	87.9% ± .1%	85.4% ± .1%
Linear	Cumulative	From Scratch	N/A	91.7% ± .1%	91.2% ± .1%	92.5% ± .1%	92.6% ± .1%	88.8% ± .1%
Linear	One Bucket	From Scratch	0.5	87.8% ± .0%	86.7% ± .0%	88.4% ± .0%	88.5% ± .0%	85.8% ± .0%
Linear	One Bucket	From Scratch	1.0	88.4% ± .0%	87.7% ± .0%	88.8% ± .0%	88.9% ± .0%	86.3% ± .0%
Linear	One Bucket	From Scratch	2.0	88.8% ± .0%	87.7% ± .0%	88.7% ± .0%	88.7% ± .0%	86.4% ± .0%
Linear	One Bucket	From Scratch	5.0	89.1% ± .0%	87.8% ± .0%	88.5% ± .0%	88.4% ± .1%	86.4% ± .0%
Linear	One Bucket	From Scratch	0.25 * i/k	88.6% ± .1%	87.8% ± .0%	88.8% ± .1%	88.8% ± .1%	86.3% ± .1%
Linear	One Bucket	From Scratch	0.50 * i/k	88.9% ± .1%	88.1% ± .1%	88.7% ± .1%	88.7% ± .1%	86.5% ± .1%
Linear	One Bucket	From Scratch	0.75 * i/k	89.0% ± .1%	87.9% ± .1%	88.4% ± .1%	88.3% ± .1%	86.5% ± .1%
Linear	One Bucket	From Scratch	1.00 * i/k	89.1% ± .1%	87.8% ± .0%	88.2% ± .1%	88.0% ± .1%	86.5% ± .1%
Linear	Cumulative	Finetuning	N/A	92.3% ± .0%	91.8% ± .0%	93.1% ± .0%	93.3% ± .0%	89.2% ± .0%
Linear	One Bucket	Finetuning	0.5	88.6% ± .0%	88.1% ± .0%	89.8% ± .0%	90.0% ± .0%	86.8% ± .0%
Linear	One Bucket	Finetuning	1.0	89.5% ± .0%	88.8% ± .0%	90.4% ± .0%	90.6% ± .0%	87.4% ± .0%
Linear	One Bucket	Finetuning	2.0	90.7% ± .0%	89.8% ± .0%	91.2% ± .0%	91.4% ± .0%	89.1% ± .2%
Linear	One Bucket	Finetuning	5.0	91.5% ± .0%	90.4% ± .0%	91.7% ± .0%	91.7% ± .0%	88.5% ± .0%
Linear	One Bucket	Finetuning	0.25 * i/k	89.7% ± .0%	88.8% ± .0%	90.5% ± .0%	90.7% ± .0%	87.1% ± .0%
Linear	One Bucket	Finetuning	0.50 * i/k	90.7% ± .0%	89.7% ± .0%	91.2% ± .0%	91.3% ± .0%	87.9% ± .0%
Linear	One Bucket	Finetuning	0.75 * i/k	91.3% ± .0%	90.1% ± .0%	91.6% ± .0%	91.6% ± .0%	88.2% ± .1%
Linear	One Bucket	Finetuning	1.00 * i/k	91.6% ± .0%	90.5% ± .0%	91.7% ± .0%	91.7% ± .0%	88.5% ± .0%

Table 4: **Linear Classification with IID Evaluation Protocol (Features Extracted by MoCo-YFCC-B0)**. With iid protocol, **Finetuning** consistently outperforms **From Scratch** and **Napping**. Furthermore, when there is a limited buffer size of one bucket of memory, biased reservoir sampling that favors more recent samples with higher values of alpha can improve the test time performance. Recall that this iid evaluation protocol requires a held-out test set, which is not needed in the streaming protocol 5.

Network	Buffer Size	Method	Alpha	Evaluation Metrics	
				Next-Domain Accuracy	Forward Transfer
Linear	N/A	Nap	N/A	$86.7\% \pm .0\%$	$85.7\% \pm .0\%$
Linear	Cumulative	From Scratch	N/A	$90.6\% \pm .1\%$	$88.7\% \pm .1\%$
Linear	One Bucket	From Scratch	0.5	$87.2\% \pm .0\%$	$86.1\% \pm .0\%$
Linear	One Bucket	From Scratch	1.0	$87.6\% \pm .1\%$	$86.5\% \pm .1\%$
Linear	One Bucket	From Scratch	2.0	$87.9\% \pm .1\%$	$86.8\% \pm .1\%$
Linear	One Bucket	From Scratch	5.0	$88.6\% \pm .1\%$	$87.6\% \pm .0\%$
Linear	One Bucket	From Scratch	0.25 * i/k	$87.0\% \pm .1\%$	$86.0\% \pm .1\%$
Linear	One Bucket	From Scratch	0.50 * i/k	$87.9\% \pm .1\%$	$86.5\% \pm .1\%$
Linear	One Bucket	From Scratch	0.75 * i/k	$88.5\% \pm .1\%$	$87.0\% \pm .1\%$
Linear	One Bucket	From Scratch	1.00 * i/k	$88.9\% \pm .0\%$	$87.3\% \pm .0\%$
Linear	Cumulative	Finetuning	N/A	$91.1\% \pm .0\%$	$89.3\% \pm .0\%$
Linear	One Bucket	Finetuning	0.5	$88.8\% \pm .1\%$	$87.6\% \pm .1\%$
Linear	One Bucket	Finetuning	1.0	$89.5\% \pm .0\%$	$88.1\% \pm .1\%$
Linear	One Bucket	Finetuning	2.0	$90.3\% \pm .1\%$	$88.8\% \pm .1\%$
Linear	One Bucket	Finetuning	5.0	$90.9\% \pm .0\%$	$89.2\% \pm .0\%$
Linear	One Bucket	Finetuning	0.25 * i/k	$89.7\% \pm .0\%$	$88.0\% \pm .1\%$
Linear	One Bucket	Finetuning	0.50 * i/k	$90.4\% \pm .1\%$	$88.6\% \pm .1\%$
Linear	One Bucket	Finetuning	0.75 * i/k	$90.8\% \pm .1\%$	$88.9\% \pm .0\%$
Linear	One Bucket	Finetuning	1.00 * i/k	$91.1\% \pm .0\%$	$89.2\% \pm .0\%$

Table 5: **Linear Classification with Streaming Evaluation Protocol (Features Extracted by MoCo-YFCC-B0)**. This table contains the complete results for Table 2 in main paper, including forward transfer and fixed value alpha experiments. Each entry shows the mean and std of 5 runs using different random seeds. The key takeaway is when there is a limited buffer size, contrary to the commonly followed strategy (training on the i.i.d. distribution sampled uniformly with reservoir sampling strategy), it is better to train on the recent data than from the past. The rest of the observation follows table 4, e.g., **Finetuning** outperforms **From Scratch** and **Napping** methods. This streaming evaluation protocol does not require a held-out test set, and it is a more realistic measure of a CL system deployed in the real world.

Network	Buffer Size	Method	Alpha	In-Domain Accuracy					Next-Domain Accuracy				
				CLIP	Pretrain-ImgNet	MoCo-ImgNet	BYOL-ImgNet	MoCo-YFCC	CLIP	Pretrain-ImgNet	MoCo-ImgNet	BYOL-ImgNet	MoCo-YFCC
Linear	N/A	Nap	N/A	$79.8\% \pm .2\%$	$93.3\% \pm .2\%$	$88.2\% \pm .3\%$	$91.1\% \pm .2\%$	$86.7\% \pm .1\%$	$79.4\% \pm .3\%$	$93.0\% \pm .2\%$	$87.7\% \pm .3\%$	$90.7\% \pm .2\%$	$86.1\% \pm .1\%$
Linear	Cumulative	From Scratch	N/A	$83.2\% \pm .2\%$	$95.5\% \pm .2\%$	$92.8\% \pm .2\%$	$94.7\% \pm .1\%$	$91.7\% \pm .1\%$	$83.0\% \pm .4\%$	$95.0\% \pm .1\%$	$92.3\% \pm .2\%$	$94.3\% \pm .1\%$	$91.2\% \pm .1\%$
Linear	One Bucket	From Scratch	0.5	$80.5\% \pm .2\%$	$94.1\% \pm .2\%$	$89.3\% \pm .2\%$	$92.1\% \pm .1\%$	$87.8\% \pm .1\%$	$80.3\% \pm .4\%$	$93.6\% \pm .2\%$	$88.4\% \pm .3\%$	$91.2\% \pm .3\%$	$86.7\% \pm .1\%$
Linear	One Bucket	From Scratch	1.0	$80.9\% \pm .1\%$	$94.2\% \pm .2\%$	$89.9\% \pm .3\%$	$92.4\% \pm .3\%$	$88.4\% \pm .1\%$	$80.4\% \pm .4\%$	$94.0\% \pm .5\%$	$88.9\% \pm .3\%$	$91.8\% \pm .4\%$	$87.7\% \pm .1\%$
Linear	One Bucket	From Scratch	2.0	$80.7\% \pm .1\%$	$94.3\% \pm .2\%$	$90.1\% \pm .2\%$	$92.7\% \pm .3\%$	$88.8\% \pm .1\%$	$80.1\% \pm .3\%$	$93.8\% \pm .3\%$	$89.1\% \pm .3\%$	$91.8\% \pm .2\%$	$87.7\% \pm .1\%$
Linear	One Bucket	From Scratch	5.0	$81.0\% \pm .2\%$	$94.4\% \pm .1\%$	$90.4\% \pm .2\%$	$92.8\% \pm .2\%$	$89.1\% \pm .1\%$	$80.0\% \pm .1\%$	$93.9\% \pm .1\%$	$89.4\% \pm .3\%$	$92.0\% \pm .2\%$	$87.9\% \pm .1\%$
Linear	One Bucket	From Scratch	0.25 * i/k	$80.7\% \pm .2\%$	$94.2\% \pm .1\%$	$90.1\% \pm .3\%$	$92.5\% \pm .2\%$	$88.6\% \pm .1\%$	$80.1\% \pm .3\%$	$93.9\% \pm .3\%$	$89.0\% \pm .3\%$	$91.8\% \pm .4\%$	$87.8\% \pm .1\%$
Linear	One Bucket	From Scratch	0.50 * i/k	$80.9\% \pm .3\%$	$94.3\% \pm .1\%$	$90.3\% \pm .2\%$	$92.7\% \pm .2\%$	$88.9\% \pm .1\%$	$80.3\% \pm .3\%$	$93.9\% \pm .1\%$	$89.2\% \pm .4\%$	$92.0\% \pm .1\%$	$88.1\% \pm .1\%$
Linear	One Bucket	From Scratch	0.75 * i/k	$81.1\% \pm .2\%$	$94.4\% \pm .1\%$	$90.5\% \pm .2\%$	$92.8\% \pm .3\%$	$89.0\% \pm .1\%$	$80.2\% \pm .1\%$	$93.8\% \pm .2\%$	$89.3\% \pm .5\%$	$91.9\% \pm .1\%$	$87.9\% \pm .1\%$
Linear	One Bucket	From Scratch	1.00 * i/k	$81.0\% \pm .2\%$	$94.3\% \pm .1\%$	$90.6\% \pm .2\%$	$93.0\% \pm .3\%$	$89.1\% \pm .1\%$	$79.8\% \pm .2\%$	$93.7\% \pm .1\%$	$89.4\% \pm .3\%$	$91.9\% \pm .3\%$	$87.8\% \pm .1\%$
Linear	Cumulative	Finetuning	N/A	$83.2\% \pm .2\%$	$95.3\% \pm .2\%$	$93.4\% \pm .2\%$	$94.4\% \pm .1\%$	$91.5\% \pm .0\%$	$83.0\% \pm .3\%$	$94.9\% \pm .0\%$	$93.1\% \pm .3\%$	$94.0\% \pm .1\%$	$90.6\% \pm .0\%$
Linear	One Bucket	Finetuning	0.5	$80.7\% \pm .3\%$	$94.3\% \pm .2\%$	$90.9\% \pm .3\%$	$92.6\% \pm .1\%$	$88.6\% \pm .0\%$	$80.3\% \pm .3\%$	$93.8\% \pm .3\%$	$90.3\% \pm .5\%$	$91.9\% \pm .1\%$	$88.1\% \pm .0\%$
Linear	One Bucket	Finetuning	1.0	$81.5\% \pm .2\%$	$94.7\% \pm .2\%$	$91.6\% \pm .2\%$	$93.2\% \pm .1\%$	$89.5\% \pm .0\%$	$81.3\% \pm .3\%$	$94.3\% \pm .4\%$	$90.8\% \pm .1\%$	$92.6\% \pm .3\%$	$88.8\% \pm .0\%$
Linear	One Bucket	Finetuning	2.0	$82.0\% \pm .1\%$	$94.7\% \pm .1\%$	$92.2\% \pm .3\%$	$93.6\% \pm .2\%$	$90.7\% \pm .0\%$	$81.7\% \pm .3\%$	$94.2\% \pm .2\%$	$91.4\% \pm .3\%$	$93.0\% \pm .2\%$	$89.8\% \pm .0\%$
Linear	One Bucket	Finetuning	5.0	$82.5\% \pm .2\%$	$94.8\% \pm .2\%$	$92.7\% \pm .2\%$	$93.9\% \pm .2\%$	$91.5\% \pm .0\%$	$82.1\% \pm .2\%$	$94.3\% \pm .1\%$	$91.9\% \pm .3\%$	$93.3\% \pm .2\%$	$90.4\% \pm .0\%$
Linear	One Bucket	Finetuning	0.25 * i/k	$81.5\% \pm .3\%$	$94.5\% \pm .1\%$	$91.7\% \pm .2\%$	$93.1\% \pm .2\%$	$89.7\% \pm .0\%$	$80.9\% \pm .3\%$	$94.2\% \pm .3\%$	$90.8\% \pm .4\%$	$92.6\% \pm .2\%$	$88.8\% \pm .0\%$
Linear	One Bucket	Finetuning	0.50 * i/k	$82.2\% \pm .2\%$	$94.7\% \pm .1\%$	$92.3\% \pm .2\%$	$93.6\% \pm .1\%$	$90.7\% \pm .0\%$	$81.7\% \pm .2\%$	$94.3\% \pm .2\%$	$91.5\% \pm .3\%$	$93.0\% \pm .1\%$	$89.7\% \pm .0\%$
Linear	One Bucket	Finetuning	0.75 * i/k	$82.7\% \pm .1\%$	$94.8\% \pm .1\%$	$92.6\% \pm .2\%$	$93.8\% \pm .2\%$	$91.3\% \pm .0\%$	$82.1\% \pm .2\%$	$94.3\% \pm .2\%$	$91.8\% \pm .3\%$	$93.3\% \pm .2\%$	$90.1\% \pm .0\%$
Linear	One Bucket	Finetuning	1.00 * i/k	$82.9\% \pm .1\%$	$94.8\% \pm .1\%$	$92.8\% \pm .2\%$	$94.0\% \pm .1\%$	$91.6\% \pm .0\%$	$82.0\% \pm .2\%$	$94.2\% \pm .1\%$	$92.1\% \pm .3\%$	$93.5\% \pm .2\%$	$90.5\% \pm .0\%$

Table 6: **Linear Classification with IID Evaluation Protocol with all pre-trained models**. We include the linear classification results using all 5 pre-trained models. Each entry shows the mean and std of 5 runs using different random seeds.

Network	Buffer Size	Method	Alpha	In-Domain Accuracy					Next-Domain Accuracy				
				CLIP	Pretrain-ImgNet	MoCo-ImgNet	BYOL-ImgNet	MoCo-YFCC	CLIP	Pretrain-ImgNet	MoCo-ImgNet	BYOL-ImgNet	MoCo-YFCC
MLP	N/A	Nap	N/A	80.6% ± .2%	93.4% ± .4%	88.9% ± .5%	91.2% ± .2%	87.5% ± .4%	80.2% ± .4%	93.1% ± .5%	88.3% ± .6%	90.8% ± .2%	86.9% ± .5%
MLP	Cumulative	From Scratch	N/A	84.1% ± .1%	95.9% ± .1%	93.8% ± .2%	94.7% ± .1%	92.8% ± .2%	83.9% ± .2%	95.6% ± .1%	93.4% ± .2%	94.3% ± .1%	92.3% ± .2%
MLP	One Bucket	From Scratch	0.5	81.3% ± .2%	94.3% ± .1%	89.8% ± .2%	92.0% ± .1%	88.6% ± .3%	81.1% ± .3%	93.9% ± .4%	89.1% ± .2%	91.2% ± .3%	87.8% ± .5%
MLP	One Bucket	From Scratch	1.0	81.7% ± .2%	94.4% ± .2%	90.5% ± .3%	92.5% ± .2%	88.9% ± .3%	81.1% ± .3%	93.9% ± .2%	89.5% ± .4%	91.7% ± .4%	88.1% ± .3%
MLP	One Bucket	From Scratch	2.0	81.6% ± .1%	94.6% ± .2%	90.7% ± .3%	92.7% ± .3%	89.4% ± .2%	81.0% ± .3%	94.1% ± .0%	89.8% ± .2%	92.0% ± .1%	88.5% ± .3%
MLP	One Bucket	From Scratch	5.0	81.7% ± .2%	94.6% ± .1%	90.9% ± .2%	92.6% ± .4%	89.8% ± .3%	80.7% ± .2%	94.0% ± .1%	90.0% ± .2%	91.6% ± .6%	88.5% ± .1%
MLP	One Bucket	From Scratch	0.25 * i/k	81.6% ± .2%	94.6% ± .1%	90.5% ± .4%	92.5% ± .1%	89.2% ± .3%	81.1% ± .2%	94.2% ± .3%	89.6% ± .4%	92.0% ± .2%	88.1% ± .3%
MLP	One Bucket	From Scratch	0.50 * i/k	81.6% ± .2%	94.6% ± .1%	90.8% ± .2%	92.6% ± .2%	89.5% ± .3%	80.9% ± .3%	94.1% ± .1%	90.0% ± .2%	92.0% ± .2%	88.5% ± .2%
MLP	One Bucket	From Scratch	0.75 * i/k	81.8% ± .2%	94.7% ± .1%	91.0% ± .2%	92.8% ± .2%	89.7% ± .3%	80.9% ± .3%	94.0% ± .1%	89.9% ± .3%	91.9% ± .2%	88.5% ± .3%
MLP	One Bucket	From Scratch	1.00 * i/k	81.7% ± .3%	94.7% ± .1%	91.0% ± .1%	92.8% ± .2%	89.7% ± .3%	80.5% ± .2%	94.1% ± .3%	89.9% ± .3%	92.0% ± .3%	88.3% ± .2%
MLP	Cumulative	Finetuning	N/A	83.2% ± .2%	95.8% ± .2%	93.9% ± .2%	94.6% ± .1%	92.7% ± .2%	82.9% ± .2%	95.4% ± .2%	93.5% ± .1%	94.2% ± .1%	92.3% ± .2%
MLP	One Bucket	Finetuning	0.5	80.7% ± .3%	94.6% ± .3%	91.5% ± .2%	92.5% ± .2%	89.8% ± .4%	80.2% ± .3%	94.1% ± .4%	90.7% ± .2%	91.7% ± .2%	89.2% ± .4%
MLP	One Bucket	Finetuning	1.0	81.8% ± .2%	94.7% ± .1%	92.1% ± .2%	93.2% ± .2%	90.3% ± .4%	81.3% ± .1%	94.3% ± .2%	91.6% ± .4%	92.6% ± .3%	89.8% ± .4%
MLP	One Bucket	Finetuning	2.0	82.3% ± .2%	94.9% ± .2%	92.7% ± .2%	93.6% ± .2%	91.3% ± .1%	81.8% ± .4%	94.4% ± .1%	92.1% ± .2%	93.1% ± .2%	90.7% ± .2%
MLP	One Bucket	Finetuning	5.0	83.4% ± .2%	95.0% ± .1%	93.1% ± .1%	94.0% ± .1%	91.9% ± .2%	82.7% ± .3%	94.6% ± .4%	92.5% ± .2%	93.4% ± .1%	91.1% ± .3%
MLP	One Bucket	Finetuning	0.25 * i/k	81.7% ± .3%	94.7% ± .2%	92.3% ± .2%	93.2% ± .2%	90.6% ± .3%	81.1% ± .4%	94.5% ± .2%	91.5% ± .3%	92.6% ± .3%	90.0% ± .3%
MLP	One Bucket	Finetuning	0.50 * i/k	82.5% ± .3%	94.7% ± .2%	92.8% ± .1%	93.6% ± .1%	91.3% ± .2%	81.9% ± .2%	94.3% ± .3%	92.2% ± .1%	93.1% ± .2%	90.5% ± .2%
MLP	One Bucket	Finetuning	0.75 * i/k	83.2% ± .2%	94.8% ± .2%	93.1% ± .1%	93.9% ± .1%	91.7% ± .2%	82.7% ± .2%	94.4% ± .3%	92.5% ± .1%	93.4% ± .2%	91.0% ± .2%
MLP	One Bucket	Finetuning	1.00 * i/k	83.4% ± .1%	94.7% ± .2%	93.3% ± .1%	94.1% ± .1%	92.0% ± .2%	82.6% ± .3%	94.3% ± .2%	92.7% ± .2%	93.5% ± .2%	91.3% ± .2%

Table 7: **Non-Linear (2 layers MLP) Classification with IID Evaluation Protocol with all pre-trained models**. We include the non-linear classification results using all 5 pre-trained models. Each entry shows the mean and std of 5 runs using different random seeds.

Network	Buffer Size	Method	Alpha	Next-Domain Accuracy				
				CLIP	Pretrain-ImgNet	MoCo-ImgNet	BYOL-ImgNet	MoCo-YFCC
Linear	N/A	Nap	N/A	80.1% ± .0%	93.6% ± .1%	88.8% ± .0%	91.4% ± .0%	86.7% ± .0%
Linear	Cumulative	From Scratch	N/A	83.5% ± .0%	95.4% ± .0%	92.7% ± .0%	94.5% ± .0%	90.6% ± .1%
Linear	One Bucket	From Scratch	0.5	80.9% ± .1%	94.1% ± .3%	89.4% ± .1%	92.0% ± .2%	87.2% ± .0%
Linear	One Bucket	From Scratch	1.0	81.1% ± .1%	94.3% ± .1%	89.7% ± .2%	92.4% ± .2%	87.6% ± .1%
Linear	One Bucket	From Scratch	2.0	80.9% ± .1%	94.4% ± .1%	89.8% ± .1%	92.6% ± .1%	87.9% ± .1%
Linear	One Bucket	From Scratch	5.0	80.7% ± .1%	94.3% ± .1%	90.1% ± .1%	92.7% ± .1%	88.6% ± .1%
Linear	One Bucket	From Scratch	0.25 * i/k	81.0% ± .1%	94.4% ± .1%	89.8% ± .2%	92.5% ± .1%	87.0% ± .1%
Linear	One Bucket	From Scratch	0.50 * i/k	81.0% ± .1%	94.4% ± .1%	90.1% ± .0%	92.7% ± .1%	87.9% ± .1%
Linear	One Bucket	From Scratch	0.75 * i/k	80.9% ± .1%	94.4% ± .1%	90.2% ± .0%	92.6% ± .1%	88.5% ± .1%
Linear	One Bucket	From Scratch	1.00 * i/k	80.4% ± .0%	94.3% ± .0%	90.1% ± .0%	92.6% ± .0%	88.9% ± .0%
Linear	Cumulative	Finetuning	N/A	83.6% ± .0%	95.3% ± .0%	93.3% ± .0%	94.3% ± .0%	91.1% ± .0%
Linear	One Bucket	Finetuning	0.5	81.4% ± .2%	94.3% ± .2%	91.1% ± .0%	92.4% ± .2%	88.8% ± .1%
Linear	One Bucket	Finetuning	1.0	81.8% ± .2%	94.5% ± .1%	91.6% ± .2%	92.9% ± .1%	89.5% ± .0%
Linear	One Bucket	Finetuning	2.0	82.3% ± .1%	94.6% ± .1%	92.0% ± .1%	93.5% ± .1%	90.3% ± .1%
Linear	One Bucket	Finetuning	5.0	82.5% ± .0%	94.7% ± .1%	92.4% ± .0%	93.8% ± .0%	90.9% ± .0%
Linear	One Bucket	Finetuning	0.25 * i/k	81.7% ± .2%	94.6% ± .1%	91.6% ± .1%	93.1% ± .1%	89.7% ± .0%
Linear	One Bucket	Finetuning	0.50 * i/k	82.3% ± .2%	94.7% ± .1%	92.0% ± .1%	93.6% ± .0%	90.4% ± .1%
Linear	One Bucket	Finetuning	0.75 * i/k	82.6% ± .1%	94.8% ± .1%	92.5% ± .1%	93.7% ± .1%	90.8% ± .1%
Linear	One Bucket	Finetuning	1.00 * i/k	82.5% ± .0%	94.7% ± .0%	92.5% ± .0%	93.8% ± .0%	91.1% ± .0%

Table 8: **Linear Classification with Streaming Evaluation Protocol with all pre-trained models**. We include the linear classification results using all 5 pre-trained models. Each entry shows the mean and std of 5 runs using different random seeds.

Network	Buffer Size	Method	Alpha	Next-Domain Accuracy				
				CLIP	Pretrain-ImgNet	MoCo-ImgNet	BYOL-ImgNet	MoCo-B0
MLP	N/A	Nap	N/A	81.1% ± .1%	93.8% ± .0%	89.8% ± .1%	91.3% ± .0%	87.8% ± .1%
MLP	Cumulative	From Scratch	N/A	84.4% ± .1%	95.9% ± .1%	93.8% ± .0%	94.6% ± .0%	92.6% ± .0%
MLP	One Bucket	From Scratch	0.5	81.7% ± .1%	94.2% ± .1%	90.3% ± .1%	91.9% ± .2%	88.6% ± .1%
MLP	One Bucket	From Scratch	1.0	81.9% ± .2%	94.5% ± .1%	90.6% ± .3%	92.3% ± .1%	89.2% ± .2%
MLP	One Bucket	From Scratch	2.0	81.7% ± .1%	94.6% ± .0%	90.8% ± .1%	92.6% ± .1%	89.6% ± .1%
MLP	One Bucket	From Scratch	5.0	81.5% ± .1%	94.6% ± .1%	90.9% ± .1%	92.6% ± .1%	89.5% ± .1%
MLP	One Bucket	From Scratch	0.25 * i/k	81.8% ± .1%	94.5% ± .1%	90.6% ± .2%	92.6% ± .1%	89.2% ± .1%
MLP	One Bucket	From Scratch	0.50 * i/k	81.8% ± .1%	94.6% ± .1%	90.9% ± .1%	92.7% ± .1%	89.5% ± .1%
MLP	One Bucket	From Scratch	0.75 * i/k	81.6% ± .1%	94.6% ± .1%	91.0% ± .1%	92.6% ± .1%	89.6% ± .1%
MLP	One Bucket	From Scratch	1.00 * i/k	81.3% ± .0%	94.5% ± .0%	90.9% ± .0%	92.5% ± .0%	89.4% ± .0%
MLP	Cumulative	Finetuning	N/A	83.3% ± .1%	95.9% ± .1%	93.9% ± .0%	94.4% ± .0%	92.3% ± .0%
MLP	One Bucket	Finetuning	0.5	80.6% ± .2%	94.4% ± .2%	91.6% ± .2%	92.4% ± .2%	89.6% ± .1%
MLP	One Bucket	Finetuning	1.0	81.2% ± .1%	94.8% ± .1%	92.1% ± .1%	93.0% ± .1%	90.4% ± .2%
MLP	One Bucket	Finetuning	2.0	82.0% ± .2%	95.3% ± .1%	92.6% ± .1%	93.6% ± .2%	91.1% ± .1%
MLP	One Bucket	Finetuning	5.0	82.8% ± .1%	95.3% ± .1%	93.0% ± .1%	93.9% ± .0%	91.7% ± .1%
MLP	One Bucket	Finetuning	0.25 * i/k	81.2% ± .2%	95.0% ± .1%	92.2% ± .2%	93.1% ± .1%	90.4% ± .1%
MLP	One Bucket	Finetuning	0.50 * i/k	82.1% ± .1%	95.2% ± .1%	92.8% ± .1%	93.6% ± .1%	91.1% ± .1%
MLP	One Bucket	Finetuning	0.75 * i/k	82.5% ± .1%	95.3% ± .1%	93.0% ± .1%	93.8% ± .1%	91.5% ± .0%
MLP	One Bucket	Finetuning	1.00 * i/k	82.7% ± .1%	95.4% ± .0%	93.1% ± .0%	93.9% ± .0%	91.8% ± .0%

Table 9: **Non-linear (2 layers MLP) Classification with Streaming Evaluation Protocol with all pre-trained models**. We include the non-linear classification results using all 5 pre-trained models. Each entry shows the mean and std of 5 runs using different random seeds.

6 Experiment Details

In this section provide all details required for replicating the results we shown in main paper and in supplement.

Augmentation: To extract 1024-dimensional L2 normalized features using pre-trained CLIP model, we stick to the default augmentation in their official codebase without any modification (`https://github.com/openai/CLIP`). To extract pre-trained features with the rest of the ResNet50 models, we use a standard augmentation scheme:

1. First resize the longer edge of the image to 224px. (*Resize(224)*)
2. Then perform a square center crop of 224px. (*CenterCrop(224)*)
3. Convert the pixel values to the range of (0,1). (*ToTensor()*)
4. Perform a normalization with ImageNet statistics of mean = (0.485, 0.456, 0.406) and std = (0.229, 0.224, 0.225). (*Normalize(mean, std)*)

When training fully-supervised CL baselines with ResNet18, we use the same above augmentation procedure while changing "CenterCrop(224)" to "RandomCrop(224) followed by a RandomHorizontalFlip()" during training time.

Shallow model architectures: For linear classification, the linear layer has input size same as the input features (1024 for CLIP features, and 2048 for other pre-trained ResNet50 features). For non-linear classification, we adopts a simple 2 layer multi-layer perceptron (MLP). The first layer of MLP transforms the input feature to a 2048 dimensional feature (regardless of the input size), followed by ReLU activation function and a hidden layer that transforms the hidden feature to output.

Hyperparameters for shallow model experiments: We adopt the standard cross entropy loss using a SGD optimizer with 0.9 momentum and no weight decay for all experiments. We use a starting learning rate of 1.0 for linear classification experiments (the only exception is for ImageNet-pretrained features we found out that learning rate 0.1 can improve the accuracy). We use a starting learning rate of 0.1 for MLP experiments. For all these shallow model experiments, we use batch of 256 and we apply a learning rate decay by a factor of 0.1 after the first 60 epochs, and train for a total of 100 epochs.

Deep CNN architectures: We use the default ResNet18 models in PyTorch library.

Hyperparameters for deep CNN experiments: For training the fully-supervised CL baselines, we trained a ResNet18 [6] initially from scratch for 70 epochs with a batch size of 64, SGD optimizer with momentum 0.9 and weight decay of 1e-5, initial learning rate of 0.01, and apply a learning rate decay by a factor of 0.1 every 30 epochs.

We conduct all our comparison experiments with the avalanche continual learning library (`https://avalanche.continualai.org/`). For LwF [8], we have alpha (distillation hyperparameter) with a list of float number start from 0, end with 2, step of 2/11, and temperate 1 (softmax temperature for distillation). For AGEM, we have one bucket pattern per bucket(number of patterns per training step in the memory), and one bucket sample size (number of patterns in memory sample when computing reference gradient) with reservoir sampling. For SI [16], we have lambda=0.0001. For EWC [7], we have ewc lambda (hyperparameter to weigh the penalty inside the total loss) being 0.4 and mode being "online". For CWR [9], we use the last layer as the cwr layer, which is automatically generated by the avalanche library. For GDumb [12] and ER [14], we have memeory size being number of training data in one bucket, which is 3300 for streaming protocol, 2310 for iid protocol.

Computing Resources: We conduct all the experiments using a single GPU (GeForce RTX 2080).

References

[1] Z. Cai, O. Sener, and V. Koltun. Online continual learning with natural distribution shifts: An empirical study with visual data. In *ICCV*.

[2] X. Chen, H. Fan, R. Girshick, and K. He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[3] N. Díaz-Rodríguez, V. Lomonaco, D. Filliat, and D. Maltoni. Don't forget, there is more than forgetting: new metrics for continual learning. *arXiv preprint arXiv:1810.13166*, 2018.

[4] S. Farquhar and Y. Gal. Towards robust evaluations of continual learning. 2018.

[5] J.-B. Grill, F. Strub, F. Altché, C. Tallec, P. H. Richemond, E. Buchatskaya, C. Doersch, B. A. Pires, Z. D. Guo, M. G. Azar, et al. Bootstrap your own latent: A new approach to self-supervised learning.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*.

[7] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[8] A. Li, A. Jabri, A. Joulin, and L. van der Maaten. Learning visual n-grams from web data. In *ICCV*.

[9] V. Lomonaco and D. Maltoni. Core50: a new dataset and benchmark for continuous object recognition. In *Conference on Robot Learning*, pages 17–26. PMLR, 2017.

[10] D. Lopez-Paz and M. Ranzato. Gradient episodic memory for continual learning. *NeurIPS*, 2017.

[11] U. Mall, K. Matzen, B. Hariharan, N. Snavely, and K. Bala. Geostyle: Discovering fashion trends and events. In *ICCV*.

[12] A. Prabhu, P. H. Torr, and P. K. Dokania. Gdumb: A simple approach that questions our progress in continual learning. In *ECCV*.

[13] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al. Learning transferable visual models from natural language supervision. *International Conference on Machine Learning*, 2021.

[14] D. Rolnick, A. Ahuja, J. Schwarz, T. P. Lillicrap, and G. Wayne. Experience replay for continual learning. *NeurIPS*, 2018.

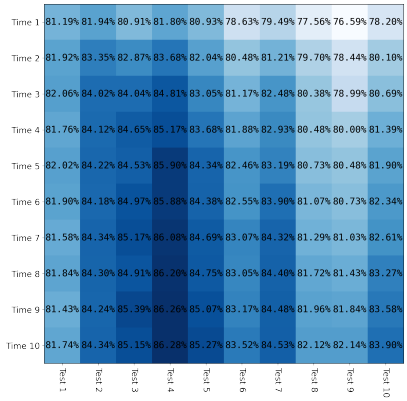
[15] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*.

[16] F. Zenke, B. Poole, and S. Ganguli. Continual learning through synaptic intelligence. In *International Conference on Machine Learning*, pages 3987–3995. PMLR, 2017.

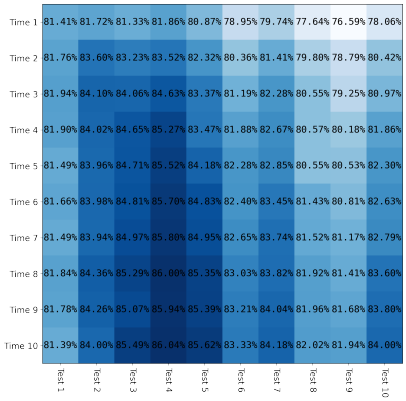
Appendix: Accuracy Matrices

We provide the accuracy matrices for all five pre-trained models, networks (linear vs MLP), sample storage (unlimited vs one bucket), training strategy (from scratch vs finetuning), and alpha value (1.0 vs 1.0 * i/k). The matrices are obtained under the iid evaluation protocol.

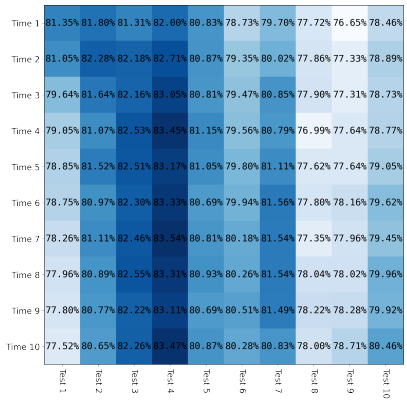
- Linear Classification
  - CLIP (Fig. 5)
  - Pretrained-ImageNet (Fig. 6)
  - MoCo-ImageNet (Fig. 7)
  - BYOL-ImageNet (Fig. 8)
  - MoCo-YFCC-B0 (Fig. 9)
- Non-Linear MLP Classification
  - CLIP (Fig. 10)
  - Pretrained-ImageNet (Fig. 11)
  - MoCo-ImageNet (Fig. 12)
  - BYOL-ImageNet (Fig. 13)
  - MoCo-YFCC-B0 (Fig. 14)



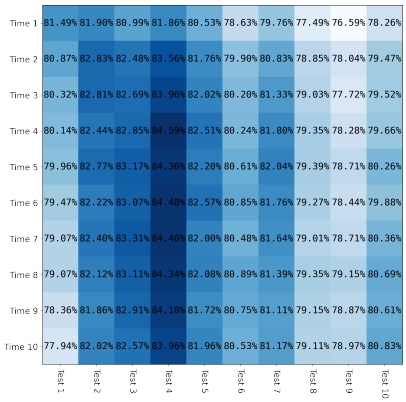
Cumulative, From Scratch



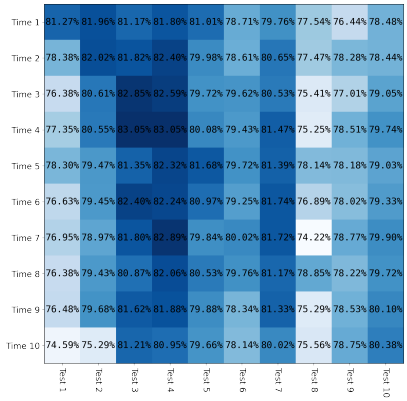
Cumulative, Finetuning



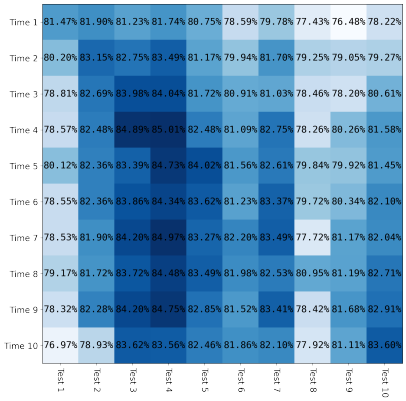
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

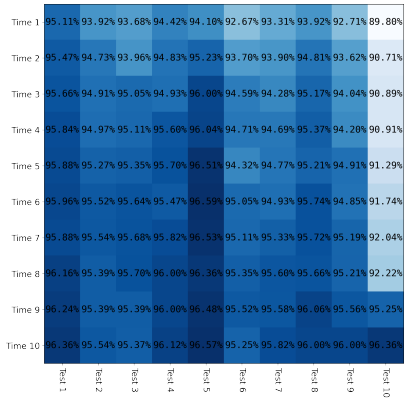


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

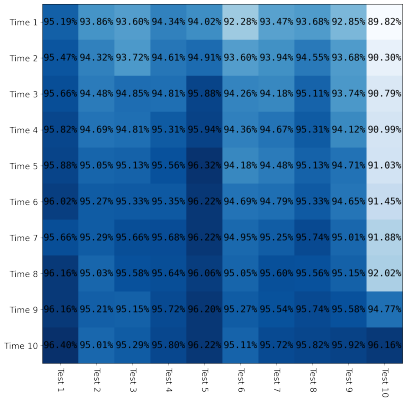


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

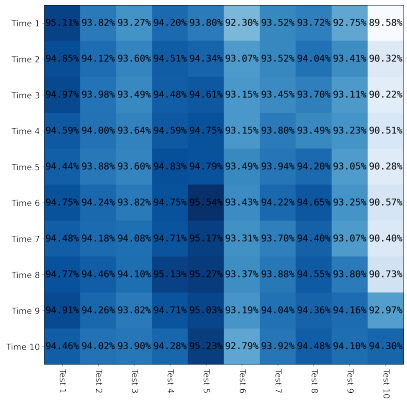
Figure 5: **Accuracy Matrix with Linear Classification (CLIP feature)**.



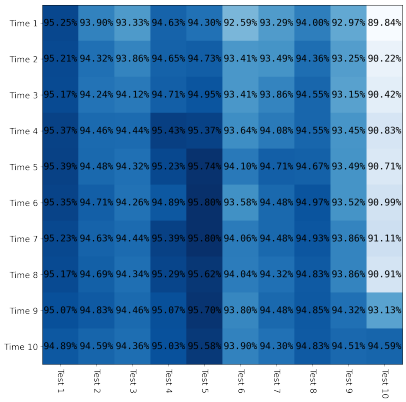
Cumulative, From Scratch



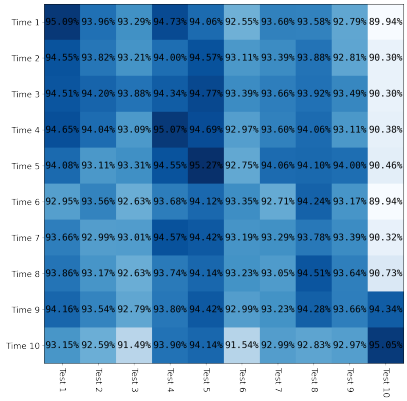
Cumulative, Finetuning



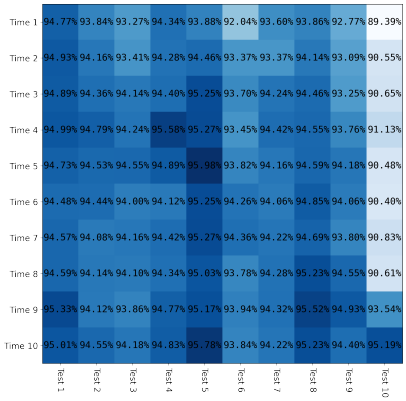
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

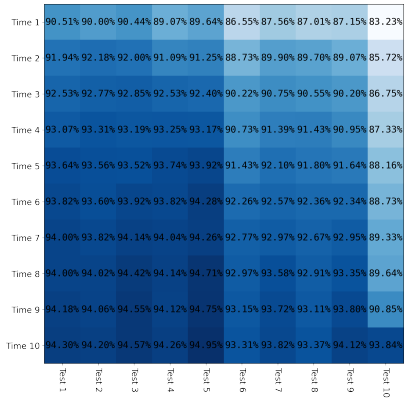


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

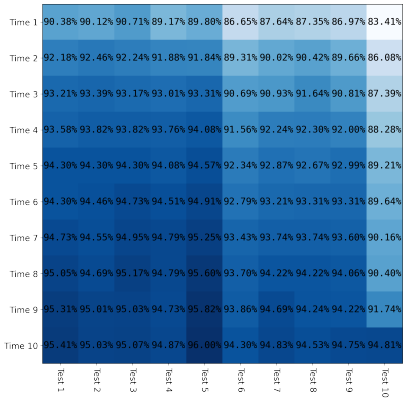


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

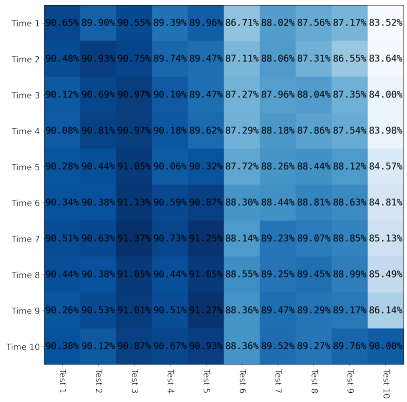
Figure 6: **Accuracy Matrix with Linear Classification (Pretrained-ImageNet feature)**.



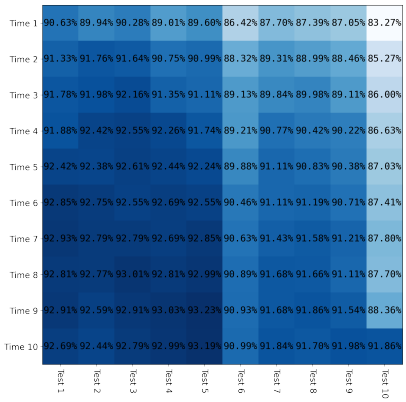
Cumulative, From Scratch



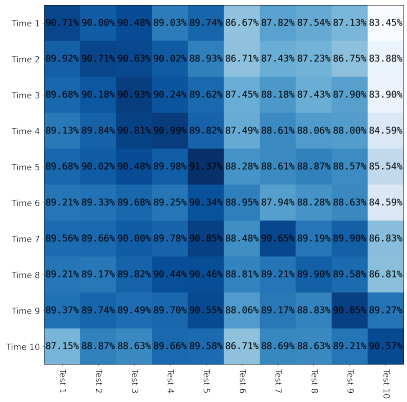
Cumulative, Finetuning



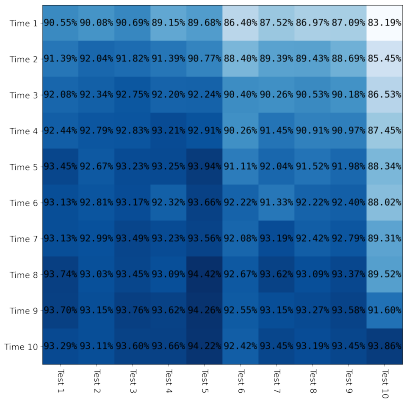
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**,Finetuning**

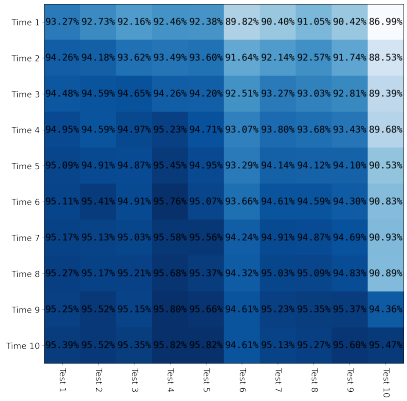


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

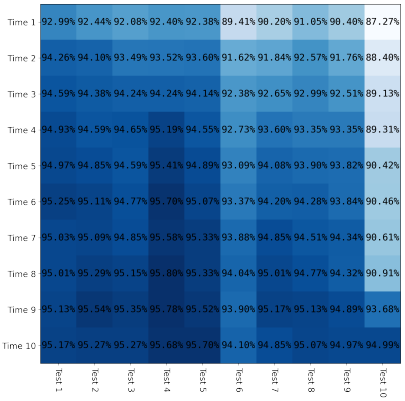


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

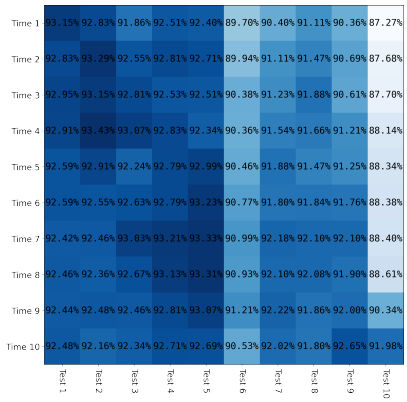
Figure 7: **Accuracy Matrix with Linear Classification (MoCo-ImageNet feature)**.



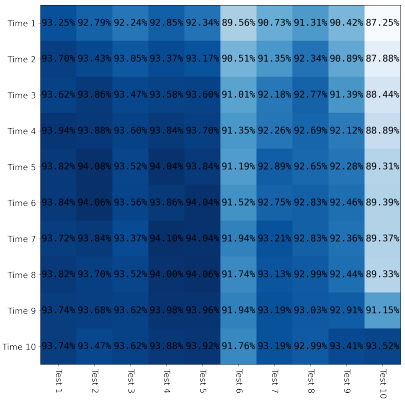
Cumulative, From Scratch



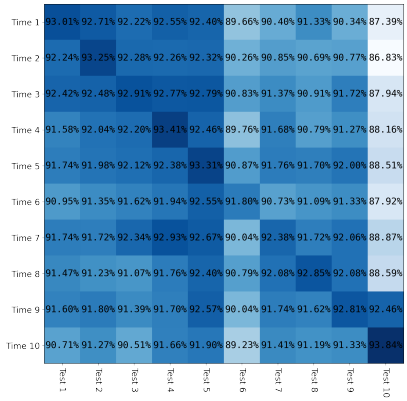
Cumulative, Finetuning



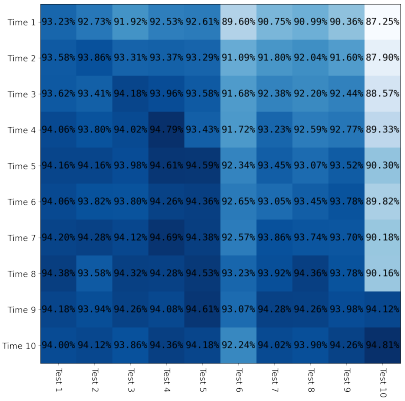
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

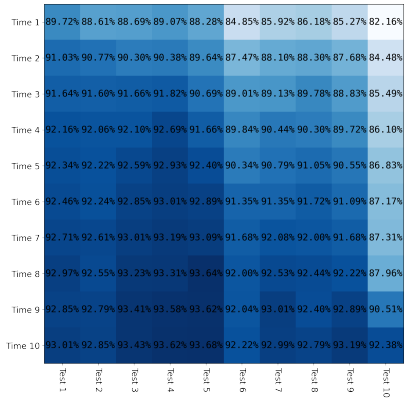


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

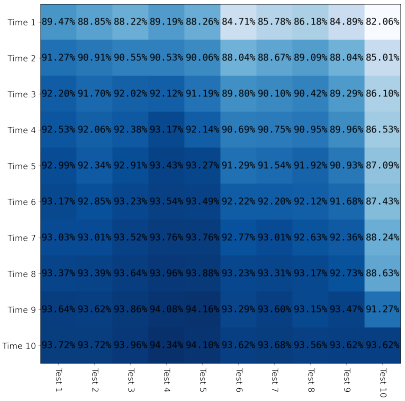


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

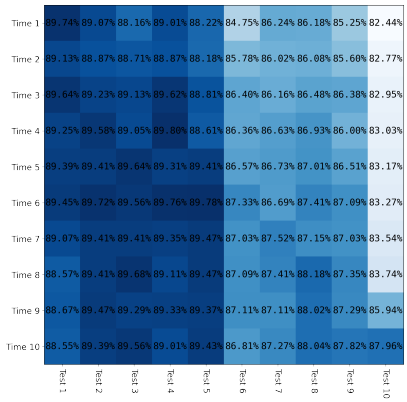
Figure 8: **Accuracy Matrix with Linear Classification (BYOL-ImageNet feature)**.



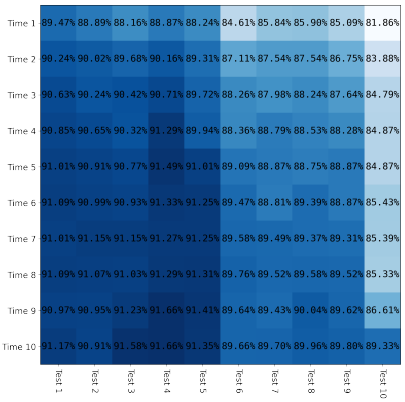
Cumulative, From Scratch



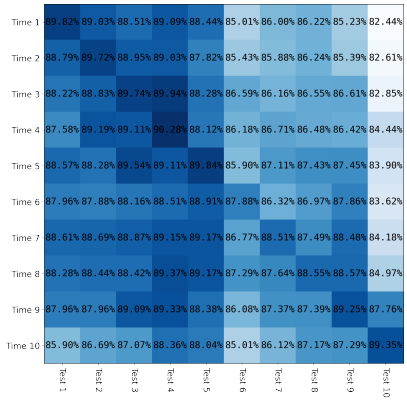
Cumulative, Finetuning



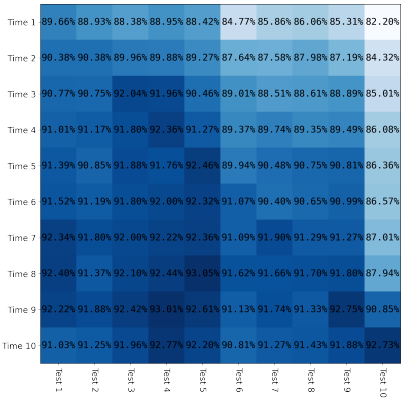
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

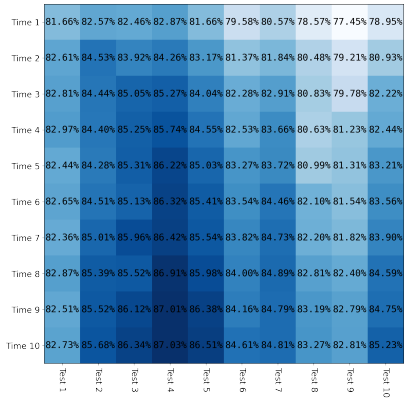


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

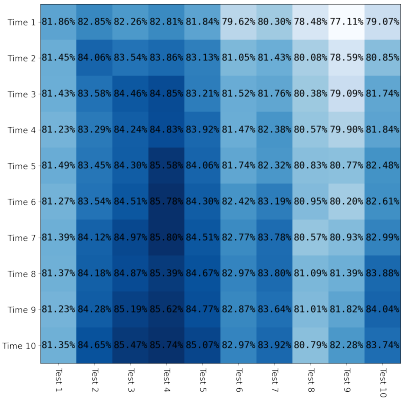


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

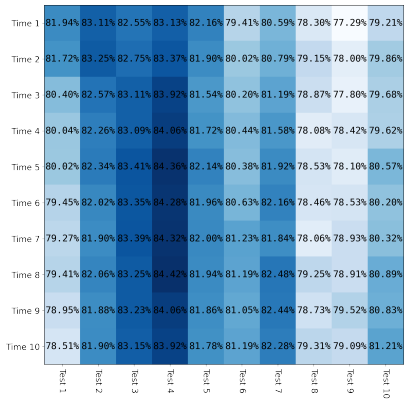
Figure 9: **Accuracy Matrix with Linear Classification (MoCo-YFCC-B0 feature)**.



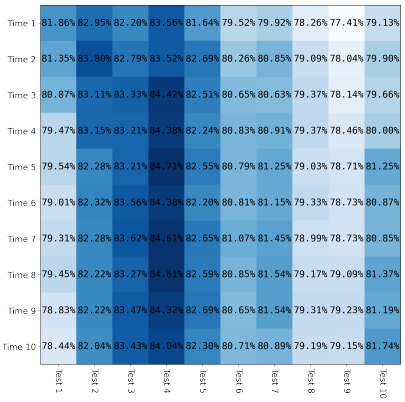
Cumulative, From Scratch



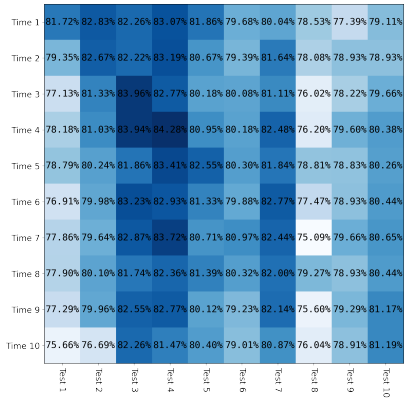
Cumulative, Finetuning



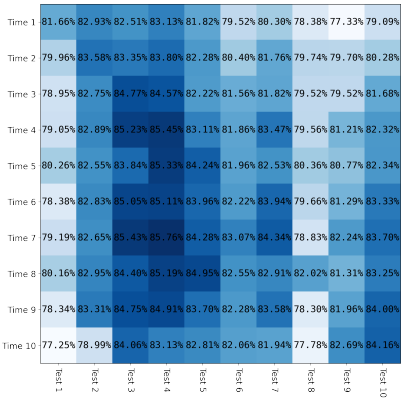
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

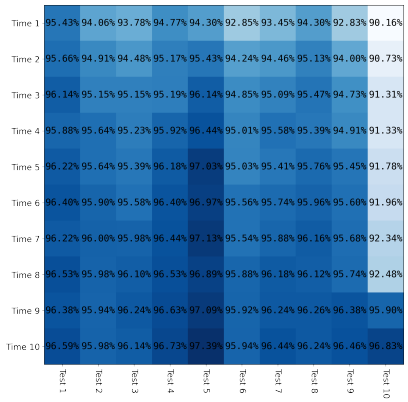


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

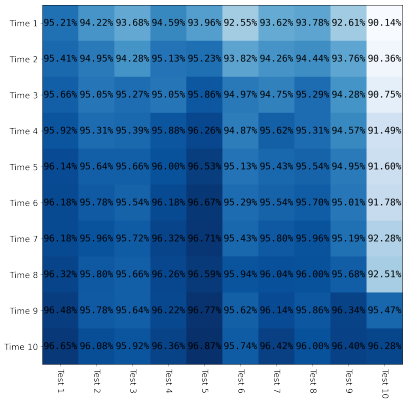


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

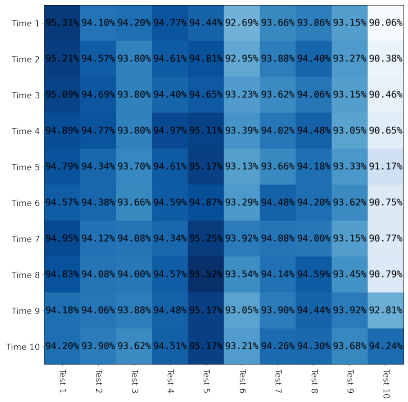
Figure 10: **Accuracy Matrix with Non-Linear (MLP) Classification (CLIP feature)**.



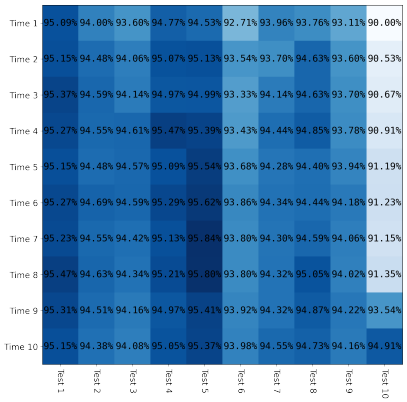
Cumulative, From Scratch



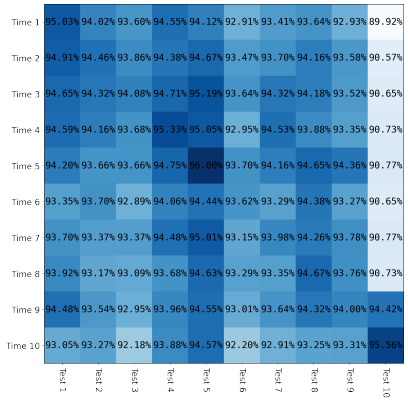
Cumulative, Finetuning



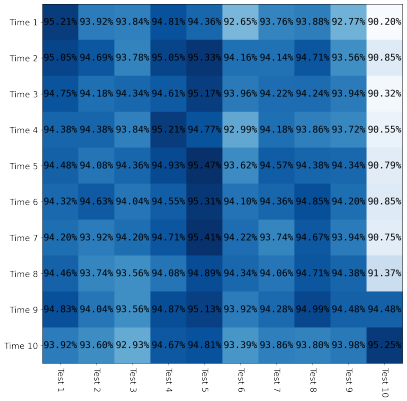
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

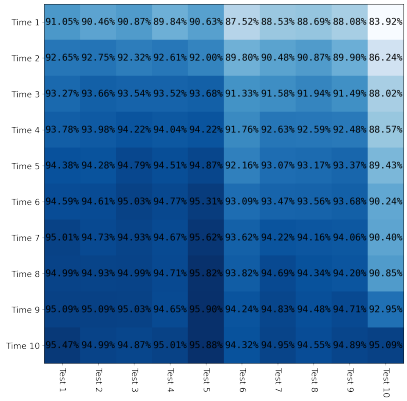


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

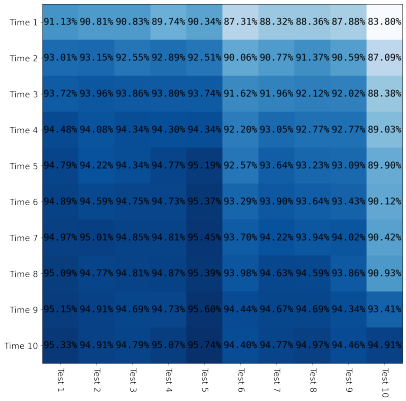


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

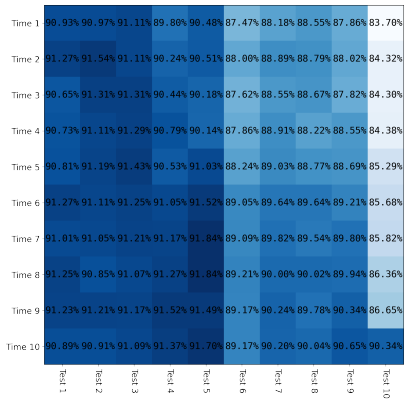
Figure 11: **Accuracy Matrix with Non-Linear (MLP) Classification (Pretrained-ImageNet feature)**.



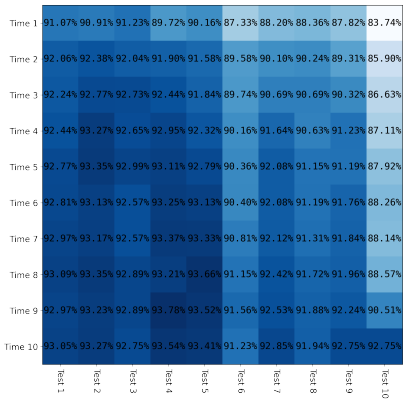
Cumulative, From Scratch



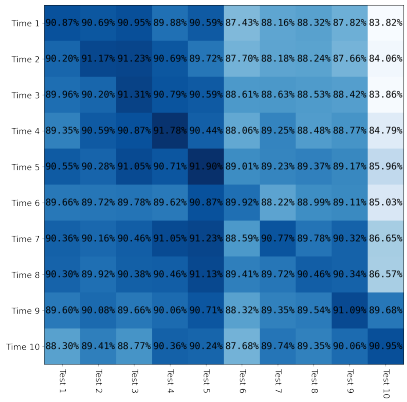
Cumulative, Finetuning



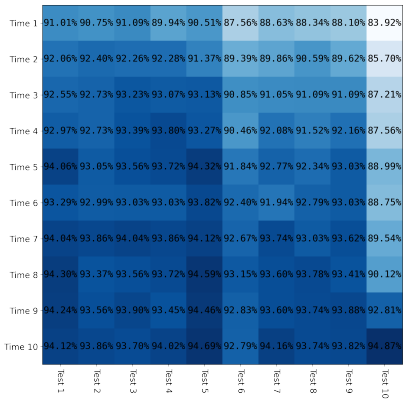
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

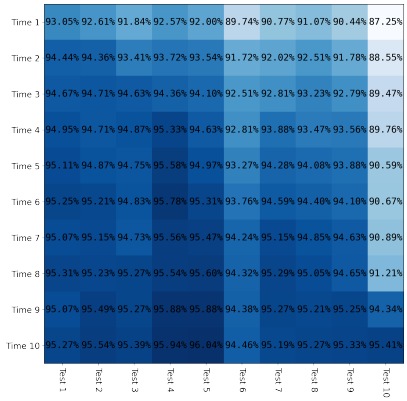


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

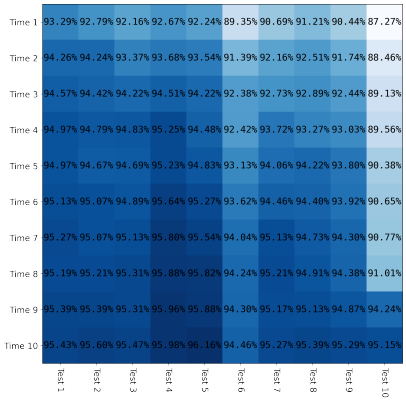


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

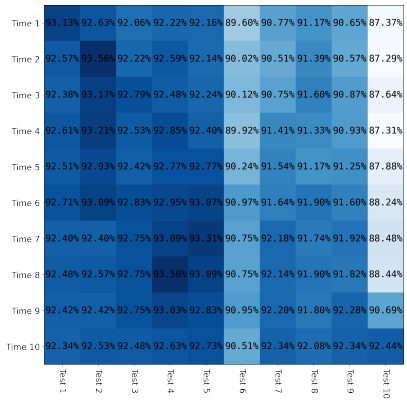
Figure 12: **Accuracy Matrix with Non-Linear (MLP) Classification (MoCo-ImageNet feature)**.



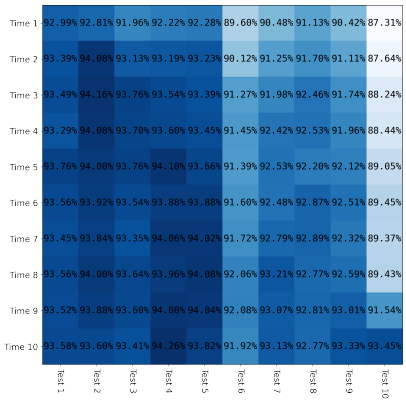
Cumulative, From Scratch



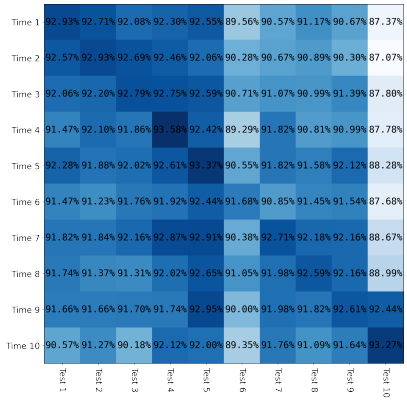
Cumulative, Finetuning



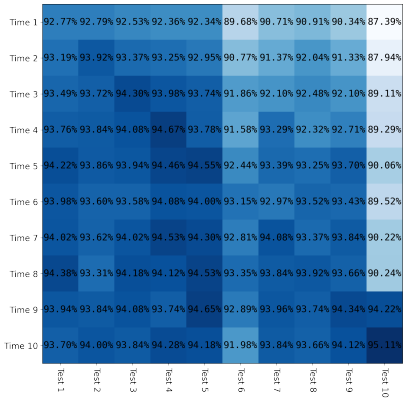
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**

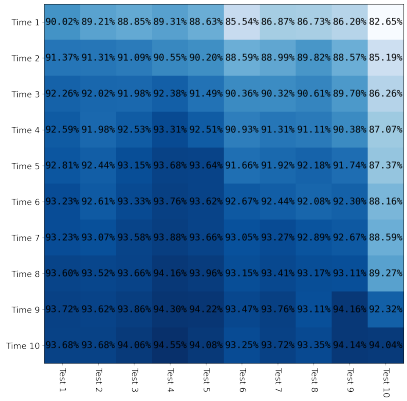


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**

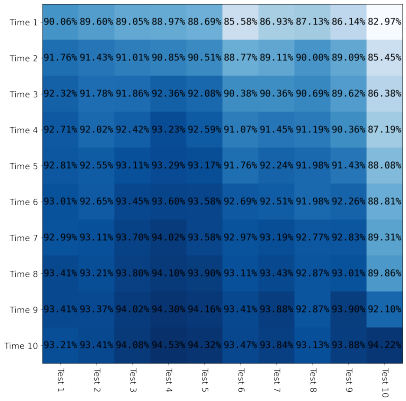


One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

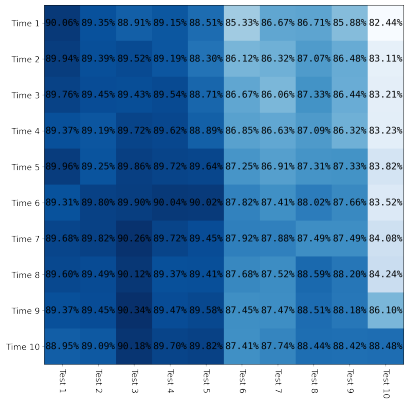
Figure 13: **Accuracy Matrix with Non-Linear (MLP) Classification (BYOL-ImageNet feature)**.



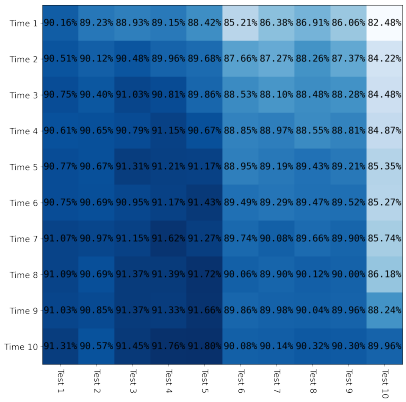
Cumulative, From Scratch



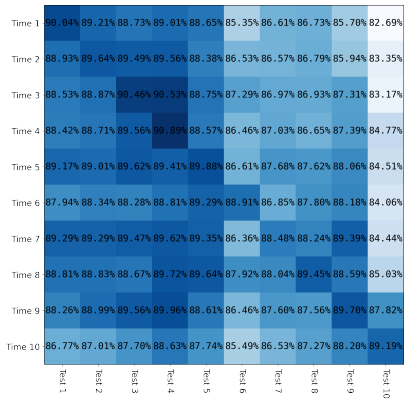
Cumulative, Finetuning



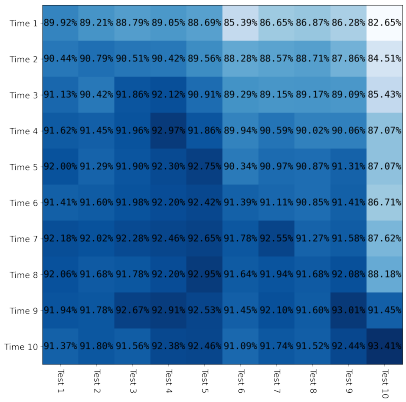
One Bucket, Alpha = 1.0**, From Scratch**



One Bucket, Alpha = 1.0**, Finetuning**



One Bucket, Alpha = $1.0 * \frac{i}{k}$**, From Scratch**



One Bucket, Alpha = $1.0 * \frac{i}{k}$**, Finetuning**

Figure 14: **Accuracy Matrix with Non-Linear (MLP) Classification (MoCo-YFCC-B0 feature)**.