# Supplementary Material: Memory-Efficient Incremental Learning Through Feature Adaptation

Ahmet Iscen[1], Jeffrey Zhang[2], Svetlana Lazebnik[2], and Cordelia Schmid[1]

[1] Google Research
[2] University of Illinois at Urbana-Champaign

## A Algorithm

An overview of our framework is described in Algorithm **??**.

**Algorithm 1** Memory-efficient incremental learning

**procedure** Algorithm(Training examples $\mathcal{X}$, labels $\mathcal{Y}$)
  Given $T$ tasks
  *** Train first task ***
  $\mathcal{X}^1, \mathcal{Y}^1 \in \mathcal{X}, \mathcal{Y}$ ▷ Data examples for first task
  $\theta, W = \textsc{Optimize}(L_{CE}(f_{\theta,W}(\mathcal{X}^1), \mathcal{Y}^1))$ ▷ (4)
  $h_\theta^1 = h_\theta$ ▷ Freeze feature extractor
  $\mathbf{M}^1 = h_\theta^1(\mathcal{X}^1)$ ▷ Store feature descriptors of images
  $\mathbf{M}^1 = \textsc{Herding}(\mathbf{M}^1)$ ▷ Reduce number of stored features
  **for** $t \in [2, \dots, T]$ **do**
    *** Train incremental tasks ***
    $\mathcal{X}^t, \mathcal{Y}^t \in \mathcal{X}, \mathcal{Y}$ ▷ Data examples for current task
    $\theta, W = \textsc{Optimize}(L(f_{\theta,W}(\mathcal{X}^t), \mathcal{Y}^t))$ ▷ (6)
    $h_\theta^t = h_\theta$
    $\phi_\psi = \textsc{Feature Adaptation}(h_\theta^t, h_\theta^{t-1}, \mathcal{X}^t, \mathcal{Y}^t)$
    $\mathbf{M}^t = h_\theta^t(\mathcal{X}^t)$ ▷ Store new feature descriptors
    $\mathbf{M}^t = \textsc{Herding}(\mathbf{M}^t)$
    $\mathbf{M}^t = \mathbf{M}^t \cup \phi_\psi(\mathbf{M}^{t-1})$ ▷ Adapt stored features
    $\tilde{W} = \textsc{Train Classifier}(\mathbf{M}^t, \mathcal{Y}^{1,\dots,t})$ ▷ (Sec. 4.3)

**procedure** FeatureAdaptation($h_\theta^{old}, h_\theta^{new}, \mathcal{X}, \mathcal{Y}$)
  *** Returns transformation function ***
  $\overline{\mathbf{V}} = h_\theta^{old}(\mathcal{X})$ ▷ Feature descriptors of old extractor
  $\mathbf{V} = h_\theta^{new}(\mathcal{X})$ ▷ Feature descriptors of new extractor
  $\psi = \textsc{Optimize}(L_{\text{FA}}(\overline{\mathbf{V}}, \mathbf{V}, \mathcal{Y}))$ ▷ (7)
  **return** $\phi_\psi$

## B Feature Adaptation Quality

We evaluate the quality of our feature adaptation process by measuring the average similarity between the adapted features and their *ground-truth value*. We compute the feature adaptation quality as explained in Section 5.4. However, we compute two distinct measurements this time. $\omega_{t-1}$ measures the average feature adaptation quality of features extracted in the previous task (*i.e.* $y \in \mathcal{C}^{t-1}$). This measurement does not

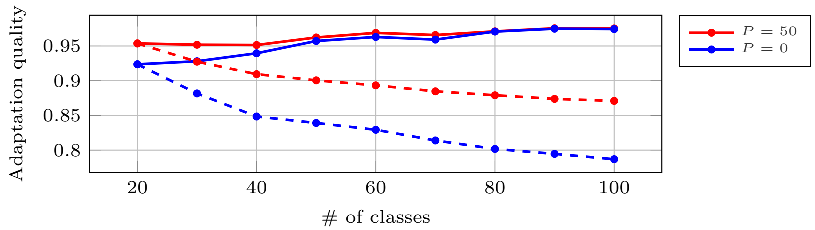


**Fig. 1.** Feature adaptation quality on CIFAR-100 for $M = 10$. $P$ refers to the number of images preserved in the memory. Solid and dashed lines correspond to vectors from previous ($\omega_{t-1}$) and first ($\omega_1$) task respectively.

track the quality over time, but shows feature adaptation quality between two tasks. $\omega_1$ measures the feature adaptation quality of features originally extracted in the first task (*i.e.* $y \in \mathcal{C}^1$), showing how much the adapted features can diverge from their optimal representation due to accumulated error. Adaptation quality is computed for all $L = 500$ feature descriptors per class.

Figure **??** shows the adaptation quality $\omega_{t-1}$ and $\omega_1$ on CIFAR-100 with $M = 10$. We report the quality measures for when $P$ number images are also preserved in the memory. We observe that $P = 0$ achieves $\omega_{t-1}$ greater than 0.9 in all tasks. It increases as more classes are seen, most likely due to the network becoming more stable. After 10 tasks, $\omega_1$ is still close to 0.8, indicating our feature adaptation is still relatively successful after training 9 subsequent tasks with no preserved images. Adaptation quality improves as $P$ increases, showing that preserving images also helps with learning a better feature adaptation.

## C Balanced Feature Classifier Training

Wu *et al.* [53] illustrated that training a classifier on fewer examples for previous classes introduces a bias towards new classes. To verify the robustness of our method under this setting, we investigate the effect of training our feature classifier on class-balanced and class-imbalanced training sets.

In our main experiments, we train our feature classifier $g_{\tilde{W}}$ with a balanced number of examples per class. We repeat our ImageNet-100 experiment in Table 1 without balancing the classifier training samples. In the unbalanced setting, the old classes contain 250 features per class in memory, whereas the new classes each contain $\sim 1300$ feature vectors. In the balanced setting, all classes contain 250 feature vectors. On ImageNet-100, we achieve 0.893 accuracy with class-imbalanced training, compared to 0.913 accuracy with class-balanced training (reported in Table 1). This shows that even though more training data is utilized in the class-imbalanced setting, the imbalanced class bias leads to a drop in overall performance.

Our method addresses this problem by building a large balanced feature set for training. Our feature adaptation method not only reduces the memory footprint compared to [53] (see Table 1), but also allows substantially more stored data points from old classes (250 features per class compared to 20 images per class for [53]). This may explain some improvement in our results over previous methods. Lastly, the significant increase in number of stored examples provides flexibility to remove examples to keep classes balanced.