

A APPENDIX

A.1 INTRODUCTION ABOUT COMPARED APPROACHES

- **Fine-tuning:** With a ResNet-32 trained from scratch, ResNet-32 is trained on every task with the same linear head using the cross-entropy loss for every new task. Hence, it suffers from severe catastrophic forgetting.
- **Joint (Oracle):** Incrementally training the model by replaying all samples from previous tasks at each incremental step as an upper bound of accuracy for continual learning.
- **LwF** Li & Hoiem (2016): intends to prevent forgetting by using knowledge distillation Hinton et al. (2015). Incorporating historical information into the present model creates a mapping between the last-stage model and the current model for every new job.
- **EWC** Kirkpatrick et al. (2017): Elastic Weight Consolidation (EWC) is a continual learning technique that prevents catastrophic forgetting by preserving important weights from previous tasks. It calculates the Fisher information matrix to identify critical parameters and adds a regularization term to the loss function for new tasks. This term penalizes changes to essential weights, balancing learning new tasks while retaining old knowledge. The updated loss function is:

$$L_{\text{total}} = L_{\text{new task}} + \lambda \sum_i F_i (\theta_i - \theta_i^*)^2 \quad (13)$$

Where F_i is the Fisher information matrix, θ_i^* represents the previous task weights, and λ is the regularization strength.

- **PASS** Zhu et al. (2021b): Prototype Augmentation and Self-Supervision for Incremental Learning, uses the prototypical network to classify the input classes and memorize the stored prototypes to update in each incremental stage with supervised-contrastive loss.

$$L_{\text{sup_contrastive}} = \sum_{i \in I} \frac{1}{|P(i)|} \sum_{p \in P(i)} -\log \frac{\exp(\text{sim}(z_i, z_p)/\tau)}{\sum_{a \in A(i)} \exp(\text{sim}(z_i, z_a)/\tau)} \quad (14)$$

- **LUCIR** Hou et al. (2019a): LUCIR reveals that forgetting is due to data imbalances between previous and new data and solves it with the help of cosine normalization, less-forget constraints, and inter-class separation. It also utilizes the knowledge distillation as Li & Hoiem (2016) with supervised contrastive loss during optimization.
- **IL2A** Zhu et al. (2021a): IL2A uses implicit semantic and class augmentation to mitigate the representation and classifier bias dilemmas with the help of spectral analysis of learning transferable and diverse representation.
- **FetrIL** Petit et al. (2023): FetrIL combines features extractor and pseudo-features generator to improve stability-plasticity with the help of geometric translation of new class features to recreate the representation of past classes
- **Elastic Feature Consolidation (EFC)** Magistri et al. (2024): EFC regularizes changes in directions in feature space most relevant for previously learned tasks and allows more plasticity in other directions with the help of the establishment of a pseudo-metric in feature space generated by a matrix termed the Empirical Feature Matrix (EFM). The Empirical Feature Matrix (EFM) Loss is given as the following:

$$\mathcal{L}_t^{\text{EFM}} = \mathbb{E}_{x \in X_t} \left[(f_t(x) - f_{t-1}(x))^T (\lambda_{\text{EFM}} E_{t-1} + \eta I) (f_t(x) - f_{t-1}(x)) \right] \quad (15)$$

where $f_t(x)$ and $f_{t-1}(x) \in \mathbb{R}^n$ represent the features of the sample x extracted from the current model and the previous model, respectively.

- **FeCAM** Goswami et al. (2024): FeCAM explores Bayesian prototypical networks for CIL, which generate new class prototypes using the frozen feature extractor and classify the features based on the Mahalanobis distance to the prototypes during inference.
- **SEED** Rypeść et al. (2023): SEED approximates the class prototypes using the Gaussian mixture model and computes the log-likelihood of the input features through the GMM as the inference. They use feature distillation as the regularizer.

- **RanPAC** McDonnell et al. (2024): RanPAC exploits parameter efficient fine-tuning (PEFT) to fine-tune Pre-Trained Model-based features extractor, trained on ImageNet-21k to extract the features and uses the random projection in the generated features to better estimate the non-linear decisions boundary with the optimizing ridge-regression.
- **EASE** Zhou et al. (2024b): uses an adapter to fine-tune PTM-based features extractor, trained in Imagenet-21k to extract the features and completes the uses of the prototypical network to infer the class during the inference. It uses cosine-similarity to complete the prototype of the old class in the new subspace. They trained task-specific adapters to distinguish the class.

A.2 IMPLEMENTATION DETAILS

We simulate ProCEED based on the standards of the widely used continual learning (CL) framework, FACIL, as outlined by Masana et al. (2022b). The comparison results are either reproduced from the official implementations provided by the authors or through the FACIL and PyCIL frameworks (Zhou et al., 2021). The code is written in Python Van Rossum & Drake Jr (1995), utilizing the PyTorch machine learning library Paszke et al. (2019). All simulations were run on an NVIDIA RTX A6000 Ada Generation GPU with 48GB of memory. ProCEED is trained using the Stochastic Gradient Descent (SGD) optimizer for 200 epochs per task, with a momentum of 0.9, weight decay of 0.0005, and an initial learning rate of 0.05. The learning rate is decreased by a factor of 10 after 60, 120, and 160 epochs. Additional hyperparameters include an α value of 0.99 and a temperature τ of 3. Knowledge distillation is applied using the L2 distance in the latent space embeddings. The model is configured with 5 experts by default, and the dimensionality of the latent space feature representation is 64. A manual hyperparameter search using Grid Search on a validation dataset is conducted to optimize ProCEED’s performance.

A.3 MORE ON DATASET SPLITS

- *Cold-start* CIL and DIL scenarios: we have evenly distributed the whole dataset in an equal number of classes in all incremental steps. Due to fewer initial classes, this approach weakens the feature extractor and is better for the assessment of the model’s plasticity. We have reproduced the baseline results for this setting using the FACIL Masana et al. (2022b) and PyCIL Zhou et al. (2021) frameworks for both CIL and DIL experiments. We have applied augmentation techniques: random crops, horizontal flips, cutouts, and AugMix Hendrycks et al. (2019) to train all the baselines.
- *Warm-start* CIL and DIL training strategy: we follow the same approach as Hou et al. (2019b), where the initial task contains 50% of the total classes, and the rest of the classes are split evenly. This approach yields a more robust feature-extracting backbone. Employing this setting allows the shared feature extractor to achieve improved results but has limited plasticity. We reproduced the results for this setting from the baselines provided by FeTrIL Petit et al. (2023). Both the cold-start and warm-start scenarios follow the *task-agnostic* evaluation setting, where the task-id is unknown during the inference.
- *Task-incremental scenarios*: This is a *task-aware* setting, where task-id is available during the inference. For this setting, we reproduced the baseline results from Wang et al. (2022d). We have used a pre-trained feature extractor from ResNet-32 He et al. (2016a). Similar augmentation techniques applied in the *cold-start* setting are used for this simulation as well.

A.4 FEATURE EXTRACTOR BACKBONE

Following the approach in Rypeś et al. (2023), we utilized the same multi-head modified ResNet-32 architecture. During training, the network’s final layer was replaced with an identity function. In the original ResNet blocks, ReLU activation functions are placed at the end of each block, which results in latent feature vectors composed of positive values. As a result, the random variables corresponding to each class are constrained within the interval $[0; \infty)^N$, where N is the dimension of the latent vector. However, Gaussian distributions require random variables defined over real-valued intervals. This restriction imposed by the ReLU activation limits the ability of the algorithm to represent classes

as multivariate Gaussian distributions. To mitigate this, we excluded the final ReLU activation in each block’s last layer of the ResNet architecture.

A.5 EVALUATION METRICS

In continual learning, many evaluation metrics rely on the **Accuracy Matrix** $R \in \mathbb{R}^{T \times T}$, where T is the total number of tasks (or domains). In the accuracy matrix R , the entry $R_{i,j}$ represents the accuracy of the model on task j after completing training on task i . In the proposed framework, we focus on the following specific metrics.

Average Incremental Accuracy (Avg. Acc.) up to task t is defined as the average accuracy across the first t tasks after incremental training on these tasks. We denote this metric as A_t and define it as:

$$A_t \triangleq \frac{1}{t} \sum_{i=1}^t R_{t,i}. \quad (16)$$

In this work, we have computed two different forms of accuracies based on Equation 12. *Task-aware* accuracy (TAw Acc), where the task-id is available during the inference and the model performs comparatively higher when it knows which task/domain is being evaluated. A more practical and difficult metric, *Task-agnostic* accuracy (TAg Acc), is when the model is evaluated without the task-id during inference. Furthermore, for the *Task-aware* setting, we have evaluated the Last Iterate accuracy (LTAw) to assess how much the model’s average performance deviates with respect to its last incremental session.

A.6 STORAGE REQUIREMENTS

The total number of parameters needed for ProCEED are:

$$|\theta_{\mathcal{F}}| + \mathcal{K} |\theta_{\mathcal{G}}| + \sum_{i=1}^{\mathcal{K}} \sum_{j=i}^{\mathcal{T}} |C_j| \left(N + \frac{N(N+1)}{2} \right) \quad (17)$$

where $|\theta_{\mathcal{F}}|$ and $|\theta_{\mathcal{G}}|$ represent the number of parameters of \mathcal{F} and \mathcal{G} functions, respectively. N is the dimensionality of the embedding feature space, \mathcal{K} is the number of experts, \mathcal{T} is the number of tasks, and $|C_j|$ is the number of classes in the j -th task. Table 5 shows the comparison of the model for various numbers of experts. We can see that the ProCEED performs consistently better even with fewer experts as compared to Rypešć et al. (2023); Wang et al. (2022a).

A.7 EUCLIDIAN NEAREST CLASS MEANS (NCM) CLASSIFIER

Class Representation (Centroids): For each class k , we compute the mean (centroid) of the feature vectors extracted by a function \mathcal{F}_k , parameterized by θ , on overall tasks. Let \mathcal{X}_t denote the data from task t and \mathcal{T} be the total number of tasks. The feature representation for class k is defined as:

$$f_k = \sum_{t=1}^{\mathcal{T}} \mathcal{F}_k(\mathcal{X}_t; \theta) \quad (18)$$

For each class, the centroid (mean vector) \mathbf{c}_k is then calculated as the mean of these feature vectors. Let the feature vectors of the training samples from class k be denoted as $\{f_i^k\}_{i=1}^{n_k}$, where n_k is the number of samples in class k . The centroid for class k , denoted as \mathbf{c}_k , is given by:

$$\mathbf{c}_k = \frac{1}{n_k} \sum_{i=1}^{n_k} f_i^k \quad (19)$$

Classification: Given a new sample with a feature vector f extracted as $f = \mathcal{F}(\mathcal{X}; \theta)$, we classify the sample by finding the nearest class centroid. The predicted class \hat{k} for the sample f is:

$$\hat{k} = \arg \min_k \|f - \mathbf{c}_k\| \quad (20)$$

where $\|\cdot\|$ represents the Euclidean distance.

Euclidean Distance: For each class k , the Euclidean distance between the sample's feature vector f and the class centroid \mathbf{c}_k is calculated as:

$$d(f, \mathbf{c}_k) = \|f - \mathbf{c}_k\| = \sqrt{\sum_{j=1}^d (f_j - c_{k,j})^2} \quad (21)$$

where d is the dimensionality of the feature space, and f_j and $c_{k,j}$ are the j -th components of the vectors f and \mathbf{c}_k , respectively.

We can summarize NCM classifier steps as follows:

1. Compute the feature vector $f_k = \sum_{t=1}^T \mathcal{F}_k(\mathcal{X}_t; \theta)$ for each class k .
2. Compute the centroid \mathbf{c}_k for each class k as the mean of the feature vectors.
3. For a new sample f , compute the distance $d(f, \mathbf{c}_k)$ for each class.
4. Assign the sample to the class with the nearest centroid:

$$\hat{k} = \arg \min_k d(f, \mathbf{c}_k) \quad (22)$$

A.8 BAYESIAN NEAREST CLASS MEANS (NCM) CLASSIFIER

Mahalanobis Distance: The Mahalanobis distance is commonly employed to quantify the distance between a data sample x and a distribution D . When the distribution is characterized by a mean μ and an invertible covariance matrix $\Sigma \in \mathbb{R}^{D \times D}$, the squared Mahalanobis distance can be expressed as:

$$D_M(x, \mu) = (x - \mu)^T \Sigma^{-1} (x - \mu) \quad (23)$$

In this expression, Σ^{-1} refers to the inverse of the covariance matrix.

The covariance matrix is symmetric and can be defined as follows:

$$\Sigma(i, j) = \begin{cases} \text{var}(i) & \text{if } i = j \\ \text{cov}(i, j) & \text{if } i \neq j \end{cases} \quad (24)$$

Here, $i, j \in 1, \dots, D$, where $\text{var}(i)$ indicates the variance of the data in the i^{th} dimension, and $\text{cov}(i, j)$ denotes the covariance between the i^{th} and j^{th} dimensions. The diagonal elements of the matrix represent the variances, while the off-diagonal elements represent the covariances. In the case of Euclidean space, we have $\Sigma = I$, where I is the identity matrix. Therefore, in Euclidean space, we assume equal variance across all dimensions and disregard the positive and negative correlations between variables. When modeling the feature distribution of classes with a multivariate normal feature distribution $\mathcal{N}(\mu_y, \Sigma_y)$, the probability of a sample feature x belonging to class y can be expressed as,

$$P(x|C = y) \approx \exp \left(-\frac{1}{2} (x - \mu_y)^T \Sigma_y^{-1} (x - \mu_y) \right) \quad (25)$$

It is straightforward to see that this is the optimal Bayesian classifier, since:

$$\arg \max_y P(Y|X) = \arg \max_y \frac{P(X|Y)P(Y)}{P(X)} = \arg \max_y P(X|Y)P(Y) \quad (26)$$

where the optimal boundary occurs at those points where each class is equally probable $P(y_i) = P(y_j)$.

Since the logarithm is a concave function and thus $\arg \max_y P(X|Y) = \arg \max_y \log P(X|Y)$:

$$\arg \max_y \log P(X|Y) = \arg \max_y \{ -(x - \mu_y)^T \Sigma_y^{-1} (x - \mu_y) \} = \arg \min_y D_M(x, \mu_y) \quad (27)$$

where the squared Mahalanobis distance $D_M(x, \mu_y) = (x - \mu_y)^T \Sigma_y^{-1} (x - \mu_y)$.

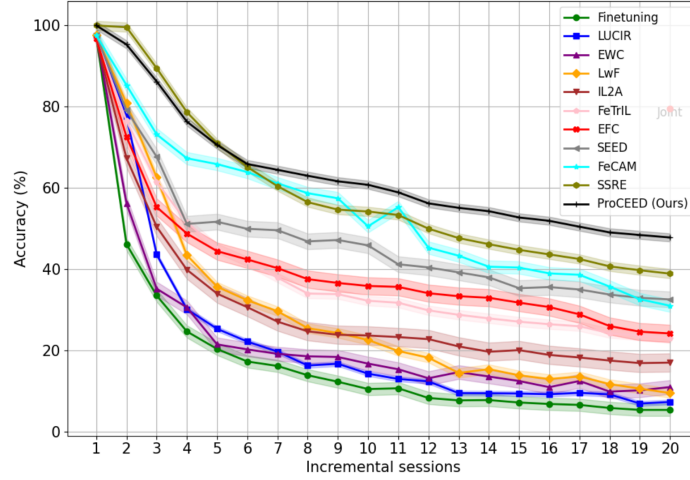


Figure 8: Average incremental accuracy measured after each task in two scenarios for each of five runs with seed (1993, 1994, 1995, 1996, 1997): (1) CIFAR100 (20 incremental tasks with 5 classes in each)

Table 7: Reproducing Table 1 with a different number of incremental tasks evaluated on CIFAR100, TinyImageNet200, ImageNetSubset100, and DomainNet using a *cold-start* scenario by introducing 500 exemplar samples from previous tasks. The best results are in bold.

Approach	CIFAR100			TinyImageNet		ImageNetSubset		DomainNet		
	$T=10$	$T=20$	$T=50$	$T=10$	$T=20$	$T=10$	$T=20$	$T=12$	$T=24$	$T=36$
Finetuning	29.93	22.74	12.41	25.28	19.35	30.48	24.15	23.78	19.33	16.93
EWC Kirkpatrick et al. (2017)	35.34	27.92	15.39	25.14	19.55	31.77	22.52	22.94	17.83	15.82
LwF Li & Hoiem (2016)	43.60	33.82	18.62	27.61	21.21	49.02	38.63	23.54	15.66	15.66
LUCIR Hou et al. (2019a)	41.17	28.46	15.23	29.21	21.73	39.07	25.69	24.07	17.56	14.52
IL2A Zhu et al. (2021a)	42.62	45.63	44.40	47.75	34.89	—	—	22.54	20.74	19.34
PASS Zhu et al. (2021b)	42.36	46.83	46.14	51.11	38.92	54.56	47.04	29.56	25.45	15.26
SSRE Zhu et al. (2022)	47.03	36.16	36.07	50.34	47.56	46.98	35.66	29.79	24.31	24.45
FeTrIL Petit et al. (2023)	47.16	42.96	39.30	55.57	49.09	48.56	39.37	41.32	35.76	34.14
SEED Rypseć et al. (2023)	64.90	61.01	36.91	50.92	43.39	69.72	67.71	48.64	38.32	34.12
EFC Magistri et al. (2024)	65.48	58.03	34.34	42.85	37.15	64.85	59.34	—	—	—
FeCAM Goswami et al. (2024)	66.32	63.85	41.61	50.34	44.85	62.03	48.73	—	—	—
ProCEED ^{NCM} A.7	71.66	60.99	51.09	49.08	48.77	69.72	69.21	50.26	48.84	45.54
ProCEED ^{Bayes} A.8	72.13	61.06	52.09	48.08	49.07	66.34	70.04	49.26	49.84	44.54
ProCEED ^{MLE}	76.56	65.23	55.34	56.00	53.15	73.55	72.46	54.33	54.15	51.34
Joint (Oracle)	79.00	79.52	80.77	67.74	69.34	83.23	84.64	64.08	65.43	69.72

A.9 ADDITIONAL RESULTS

A.9.1 MULTIPLE RUNS

In the main section, we report the average incremental accuracy which is the average of five multiple runs for each of the experiments. Figure 8 shows the standard deviation spread for each incremental step of the accuracy for different approaches.

A.9.2 MORE RESULTS

We also have presented the results of our model after introducing the 500 samples from previous tasks for a fair comparison by evaluating the ProCEED after incorporating the nearest class means (NCM) Janson et al. (2022) and Bayesian classifier Goswami et al. (2024) at the classification head. Table 7 shows the detailed accuracy of each CIL method with exemplar-based methods by using ResNet-18 He et al. (2016a) as a feature extractor backbone, where ProCEED is evaluated with NCM, Bayes (Mahalanobis), or maximum likelihood estimation (MLE) classifier at the classification head. The results show that incorporating the existing CIL heads also performs better after having a mixture-of-experts (MoE) model.

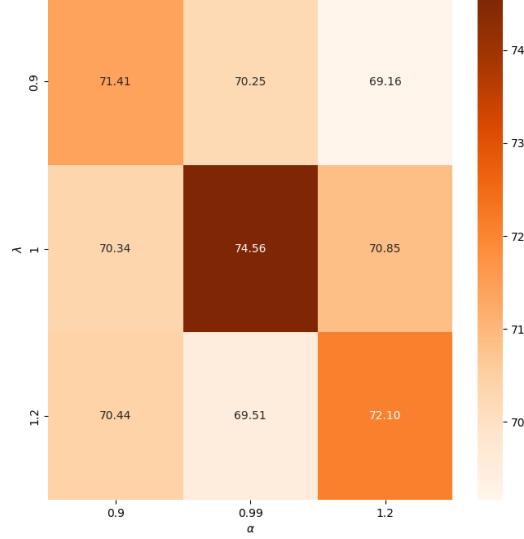


Figure 9: Average incremental accuracies of ProCEED on CIFAR100/10 after varying the hyperparameters of Equation 2.

A.9.3 ABLATION ON HYPERPARAMETERS

We also performed the ablation analysis to observe the effect range of hyperparameters α and λ in Equation 2. Figure 9 shows that the model performs best when $\alpha = 1$ and $\lambda = 0.99$. The choice of hyperparameters is also very critical playing a vital role in maintaining the plasticity of the model, during optimization as with wrong choice, it might result in drastic degradation in performance.

A.10 ALGORITHMS

Algorithm 1 Expert (Learning Model) Selection

Require: Training data D_t , Model θ , Number of experts \mathcal{K} .

- 1: Initialize an empty set of expert representations $P = \{\}$.
- 2: **for** each class c in task t **do**
- 3: **for** each expert k in \mathcal{K} **do**
- 4: Generate latent representations $p_{c,k}$ for class c .
- 5: Approximate $p_{c,k} \sim \mathcal{N}(\pi_k, \mu_{c,k}, \Sigma_{c,k})$ using the EM Algorithm Bishop (2006b)).
- 6: Append $(\mu_{c,k}, \Sigma_{c,k})$ to P .
- 7: **end for**
- 8: **end for**
- 9: Compute KL divergence between the distributions in P .
- 10: **Return** the expert with the highest KL metric (\hat{k}) among the classes for fine-tuning.
- 11:

$$\hat{k} = \arg \max_k \sum_{p_{i,k}, p_{j,k}} \mathcal{J}_{KL}(p_{i,k}, p_{j,k}).$$

Algorithm 2 Training and Fine-Tune for ProCEED

Require: Training data $\mathcal{D}_t = \{(\mathcal{X}, \mathcal{Y})\} | y \in \mathcal{C}_t$ for $t \in \mathcal{T}$, Model θ , Experts \mathcal{K} , Regularization parameter α , Distillation parameter λ , Learning rate η

```

1: for task  $t \in [1, 2, \dots, T]$  do
2:   if  $t > \mathcal{K}$  then
3:     Select the optimal expert  $\mathcal{F}_{\hat{k}}$  using Algorithm 1.
4:     Compute the cross-entropy loss  $\mathcal{L}_{CE}$  on the current task.
5:     Compute the knowledge-distillation loss  $\mathcal{L}_{KD}$  using Equation 2.
6:     Set total loss  $\mathcal{L}(\theta, \mathcal{D}_t) = (1 - \alpha) \cdot \mathcal{L}_{CE} + \alpha \cdot \mathcal{L}_{KD}$ .
7:   else
8:     Compute the cross-entropy loss  $\mathcal{L}_{CE}$  on the current task.
9:     Set total loss  $\mathcal{L}(\theta, \mathcal{D}_t) = \mathcal{L}_{CE}$ 
10:  end if
11:  Update parameters using SGD:  $\theta \leftarrow \theta - \eta \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_t)$ .
12:  Freeze feature extractor  $\mathcal{F}$  and remove Linear head  $\mathcal{A}$ .
13:  Remove linear head, extract and realign the prototypes up to  $\mathcal{D}_{t-1}$  using 8.
14:  Evaluate  $\mathcal{D}_i^{valid}$  upto  $i = 1, \dots, t$  on the model using 11.
15:  Move to the next task by appending the linear head with new classes and optimizing.
16: end for

```
