Appendices

Appendix A Overview of RanPAC and comparison to other strategies

Figure A1 provides a graphical overview of the two phases in **Algorithm 1**. Table A1 provides a summary of different strategies for leveraging pre-trained models for Continal Learning (CL) and how our own method, RanPAC, compares.

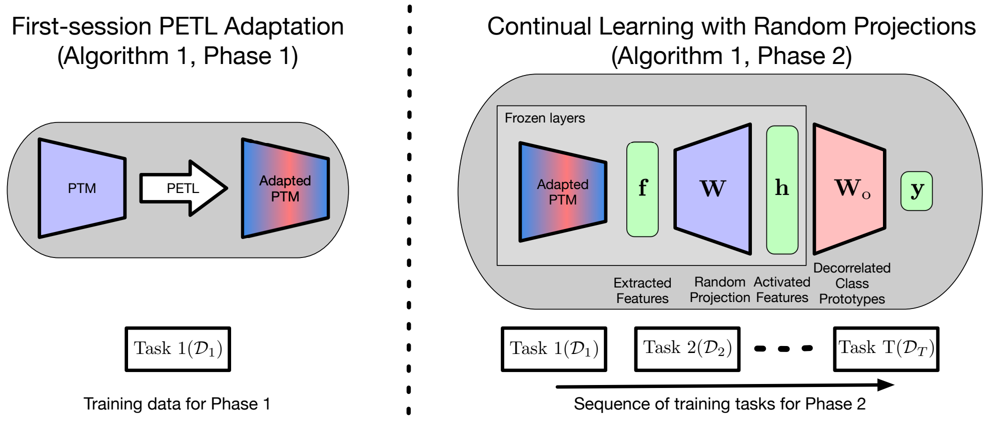

Figure A1: **Overview of RanPAC for CL classification.** In Phase 1 of **Algorithm 1** we optionally inject parameters for a Parameter-Efficient Transfer Learning (PETL) method into a frozen pre-trained model (PTM). The PETL parameters are trained only on Task 1 (the 'first-session') of a set of T continual learning tasks, to help bridge the domain gap, as in [64, 37]. Then in Phase 2, first, L-dimensional feature vectors, $\mathbf{f}$, are extracted from the network after completion of learning in Phase 1 (now frozen). Then, the extracted feature vectors are randomly projected to dimension M (typically $M > L$) using frozen weights $\mathbf{W}$, followed by nonlinear activation, ϕ, to obtain new feature vectors, $\mathbf{h} = \phi(\mathbf{f}^\top \mathbf{W})$. Training in Phase 2 is comprised of continual iterative updating of class prototypes and the Gram matrix, followed by a matrix inversion to compute $M \times N$ matrix $\mathbf{W}_\text{o}$, following the end of each Task's training. These weights can be thought of as decorrelated class prototypes that linearly weight the features in $\mathbf{h}$ to obtain class predictions in $\mathbf{y}$.

	Prompting [56, 55, 46, 54]	**Fine-tuning** [62, 47]	**CP** [20, 64, 37]	**CP + RP** RanPAC
No Rehearsal Buffer	✓	✓	✓	✓
Pre-trained model frozen	✓		✓	✓
Transformers and CNNs		?	✓	✓
Simplicity			✓	✓
Parameter-Efficient	✓		✓	✓
Theoretical support				✓
SOTA Performance				✓

Table A1: **Comparison of different strategies.** In Section 1, we categorised existing methods for leveraging pre-trained models for CL into three strategies: Prompting, Fine-tuning and Class-Prototypes (CP). Our own method, RanPAC, is a CP strategy in which we introduce a frozen Random Projection (RP) layer with nonlinear activation. Overall, we argue that CP methods provide the best combination of benefits, and within existing CP methods, RanPAC provides the strongest results, complemented by theoretical support for its use of RP and second-order statistics to decorrelate CPs. We note that rehearsal buffers have been used with some of the cited methods to boost performance, and could potentially also do so for methods where they have not yet been used, such as RanPAC.

Appendix B Theoretical support

B.1 Chernoff bound of the norm of the projected vectors

We provide further details for the discussion in Section 4.2. The norm of the vector projected using the Chernoff Bound can be written as:

$$\mathbb{P}\left(\left|\|\mathbf{W}^\top \mathbf{f}\| - \mathbb{E}_{\mathbf{W}}\left[\|\mathbf{W}^\top \mathbf{f}\|\right]\right| > \epsilon\sigma^2\right) \leq 2\exp\left(-\frac{\epsilon^2\sigma^2}{2M+\epsilon}\right). \tag{6}$$

This bound indicates the relation between the dimensionality and the expected variation in the norm of the projected vectors. For fixed σ and ϵ, as M increases, the right-hand side approaches 1, indicating that it is more likely for the norm of the projected vector to be in a desired distance to the expectation. In other words, these projected vectors in higher dimensions almost surely reside on the boundary of the distribution with a similar distance to the mean (the distribution is a better Gaussian fit).

B.2 The effects of increasing the projection dimensions

The Gram matrix of the projected vectors can be obtained by considering the inner product of any two vectors $\mathbf{f}, \mathbf{f}'$. As presented in Eqn. (3), this is derived as:

$$\begin{aligned}\mathbb{E}_{\mathbf{W}}\left[(\mathbf{W}^\top\mathbf{f})^\top(\mathbf{W}^\top\mathbf{f}')\right] &= \mathbb{E}_{\mathbf{W}}\left[(\mathbf{f}^\top\mathbf{W})(\mathbf{W}^\top\mathbf{f}')\right] \\ &= \mathbb{E}_{\mathbf{W}}\left[\sum_{i}^{M}\mathbf{W}_{(i)}^2\mathbf{f}^\top\mathbf{f}' + \sum_{i\neq j}^{M}\mathbf{W}_{(i)}^\top\mathbf{W}_{(j)}\mathbf{f}^\top\mathbf{f}'\right] \\ &= \underbrace{\sum_{i}^{M}\mathbb{E}_{\mathbf{W}}\left[\mathbf{W}_{(i)}^2\right]\mathbf{f}^\top\mathbf{f}'}_{=M\sigma^2\mathbf{f}^\top\mathbf{f}'} + \underbrace{\sum_{i\neq j}^{M}\mathbb{E}_{\mathbf{W}}\left[\mathbf{W}_{(i)}\right]^\top\mathbb{E}_{\mathbf{W}}\left[\mathbf{W}_{(j)}\right]\mathbf{f}^\top\mathbf{f}'}_{=0},\end{aligned} \tag{7}$$

where the second term is zero for any two zero-mean independently drawn random vectors $\mathbf{W}_{(i)}, \mathbf{W}_{(j)}$. We can derive the following from this expansion:

1. Using the Chernoff inequality, for any two vectors, we have

$$\begin{aligned}\mathbb{P}\left(\left|(\mathbf{W}^\top\mathbf{f})^\top(\mathbf{W}^\top\mathbf{f}') - \mathbb{E}_{\mathbf{W}}\left[(\mathbf{W}^\top\mathbf{f})^\top(\mathbf{W}^\top\mathbf{f}')\right]\right| > \epsilon' M\sigma^2\right) &\leq 2\exp\left(-\frac{\epsilon'^2 M\sigma^2}{(2+\epsilon')}\right) \\ \mathbb{P}\left(\left|(\mathbf{W}^\top\mathbf{f})^\top(\mathbf{W}^\top\mathbf{f}') - M\sigma^2\mathbf{f}^\top\mathbf{f}'\right| > \epsilon' M\sigma^2\right) &\leq 2\exp\left(-\frac{\epsilon'^2 M\sigma^2}{(2+\epsilon')}\right) \\ \mathbb{P}\left(\left|\frac{(\mathbf{W}^\top\mathbf{f})^\top(\mathbf{W}^\top\mathbf{f}')}{M\sigma^2} - \mathbf{f}^\top\mathbf{f}'\right| > \epsilon'\right) &\leq 2\exp\left(-\frac{\epsilon'^2 M\sigma^2}{(2+\epsilon')}\right).\end{aligned} \tag{8}$$

   This bound indicates that as the dimension of the projections increases, it is more likely for the inner product of any two vectors and their projections to be distinct. In other words, it is increasingly unlikely for the inner product of two vectors and their projections to be equal as M increases.

2. As M increases, it is more likely for the inner product of any two randomly projected instances to be distinct (i.e. the inner products in the projected space are more likely to be larger than some constant). That is because, using Markov's inequality, for a larger M it is easier to choose larger ϵ_2 that satisfies

$$\mathbb{P}\left(\left|(\mathbf{W}^\top\mathbf{f})^\top(\mathbf{W}^\top\mathbf{f}')\right| \geq \epsilon_2\right) \leq \frac{M\sigma^2}{\epsilon_2}. \tag{9}$$

B.3 Connection to least squares

The score of Eqn. (5) can be written in matrix form as

$$\mathbf{y}_{\text{test}} = \mathbf{h}_{\text{test}}(\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{C}, \tag{10}$$

where $\mathbf{h}_{\text{test}} := \phi(\mathbf{f}_{\text{test}}^\top \mathbf{W})$ are the randomly projected activations following element-wise nonlinear activation $\phi(\cdot)$. The form $\mathbf{W}_{\text{o}} := (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{C}$ arises in fundamental theory of least squares regression. We can write this as

$$\begin{aligned} \mathbf{W}_o &= (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{H}\mathbf{Y}_{\text{train}} \quad (11) \\ &= (\mathbf{H}\mathbf{H}^\top + \lambda\mathbf{I})^{-1}\mathbf{H}\mathbf{Y}_{\text{train}}, \quad (12) \end{aligned}$$

where $\mathbf{Y}_{\text{train}}$ is a $N \times K$ one-hot encoded target matrix, and $\mathbf{H}$ is as defined in Section 4.3. Eqn. (12) is well known [35, Eqn (7.33), p. 226] to be the minimum mean squared error solution to the l_2 regularized set of equations defined by

$$\mathbf{Y}_{\text{train}}^\top = \mathbf{W}_{\text{o}}^\top \mathbf{H}. \tag{13}$$

For parameter $\lambda \geq 0$, this is usually expressed mathematically as

$$\mathbf{W}_{\text{o}} = \text{argmin}_{\mathbf{W}}(||\mathbf{Y}_{\text{train}}^\top - \mathbf{W}^\top\mathbf{H}||_2^2 + \lambda||\mathbf{W}||_2^2). \tag{14}$$

Consequently, when not using CL, optimizing a linear output head using SGD with mean square error loss and weight decay equal to λ will produce a loss lower bounded by that achieved by directly computing $\mathbf{W}_{\text{o}}$.

B.4 Connection to Linear Discriminant Analysis, Mahalanobis distance and ZCA whitening

There are two reasons why we use Eqn. (2) for **Algorithm 1** rather than utilize LDA. First and foremost, is the fact that our formulation is mean-square-error optimal, as per Section B.3. Second, use of the inverted Gram matrix results in two convenient simplifications compared with LDA: (i) the simple form of Eqn. (2) can be used for inference instead of a form in which biases calculated from class-prototypes are required, and (ii) accumulation of updates to the Gram matrix and class-prototypes during the CL process as in Eqn. (4) are more efficient than using a covariance matrix.

We now work through the theory that leads to these conclusions. The insight gained is that they show that using Eqn. (2) is equivalent to learning a linear classifier optimized with a mean square error loss function and l_2 regularization, applied to the feature vectors of the training set. These derivations apply in both a CL and non-CL context. For CL, the key is to realise that CPs and second-order statistics can be accumulated identically for the same overall set of data, regardless of the sequence in which it arrives for training.

B.4.1 Preliminaries: Class-Prototypes

Class-prototypes for CL after task T can be defined as

$$\bar{\mathbf{c}}_y = \frac{1}{n_y}\sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathbf{h}_{t,n}\mathcal{I}_{t,n} \quad y = 1, \ldots, K, \tag{15}$$

where $\mathcal{I}_{t,n}$ is an indicator function with value 1 if the n–th training sample in the t–th task is in class y and zero otherwise, n_y is the number of training samples in each class and $\mathbf{h}_{t,n}$ is an M-dimensional projected and activated feature vector. For RanPAC we drop the mean and use

$$\mathbf{c}_y = \sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathbf{h}_{t,n}\mathcal{I}_{t,n} \quad y = 1, \ldots, K. \tag{16}$$

For Nearest Class Mean (NCM) classifiers, and a cosine similarity metric as used by [64], the score for argmax classification is

$$s_y = \frac{\mathbf{f}_{\text{test}}^\top \bar{\mathbf{c}}_y}{||\mathbf{f}_{\text{test}}|| \cdot ||\bar{\mathbf{c}}_y||}, \quad y = 1, \ldots, K, \tag{17}$$

where the $\bar{\mathbf{c}}_y$ are calculated from feature vectors $\mathbf{f}_{t,n}$ rather than $\mathbf{h}_{t,n}$. These cosine similarities depend only on first order feature statistics, i.e. the mean feature vectors for each class. In contrast, LDA and our approach use second-order statistics i.e. correlations between features within the feature vectors.

B.4.2 Relationship to LDA and Mahalanobis distance

The form of Eqn. (2) resembles Linear Discriminant Analysis (LDA) classification [35, Eqn. (4.38), p. 104]. For LDA, the score from which the predicted class for $\mathbf{f}_{\text{test}}$ is chosen is commonly expressed in a weighting and bias form

$$\begin{aligned} \psi_y &= \mathbf{f}_{\text{test}}\mathbf{a} + \mathbf{b} && (18) \\ &= \mathbf{f}_{\text{test}}^\top \mathbf{S}^{-1}\bar{\mathbf{c}}_y - 0.5\bar{\mathbf{c}}_y^\top \mathbf{S}^{-1}\bar{\mathbf{c}}_y + \log(\pi_y), \qquad y = 1, \ldots, K, && (19) \end{aligned}$$

where $\mathbf{S}$ is the $M \times M$ covariance matrix for the M dimensional feature vectors and π_y is the frequency of class y.

Finding the maximum ψ_y is equivalent to a minimization involving the Mahalanobis distance, $d_M := \sqrt{(\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y)^\top \mathbf{S}^{-1}(\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y)}$, between a test vector and the CPs, i.e. minimizing

$$\begin{aligned} \hat{\psi}_y &= d_M^2 - \log(\pi_y^2) && (20) \\ &= (\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y)^\top \mathbf{S}^{-1}(\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y) - \log(\pi_y^2) \qquad y = 1, \ldots K. && (21) \end{aligned}$$

This form highlights that if all classes are equiprobable, minimizing the Mahalanobis distance suffices for LDA classification.

B.4.3 Relationship to ZCA whitening

We now consider ZCA whitening [23]. This process linearly transforms data $\mathbf{H}$ to $\mathbf{D}_{\text{Z}}\mathbf{H}$ using the Mahalanobis transform, $\mathbf{D}_{\text{Z}} = \mathbf{S}^{-0.5}$. Substituting for $\mathbf{S}$ in Eqn. (21) gives

$$\begin{aligned} \hat{\psi}_y &= (\mathbf{D}_{\text{Z}}(\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y))^\top (\mathbf{D}_{\text{Z}}(\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y)) - \log(\pi_y^2) && (22) \\ &= ||(\mathbf{D}_{\text{Z}}(\mathbf{f}_{\text{test}} - \bar{\mathbf{c}}_y)||_2^2 - \log(\pi_y^2), \qquad y = 1, \ldots, K. && (23) \end{aligned}$$

This form implies that if all classes are equiprobable, LDA classification is equivalent to minimizing Euclidian distance following ZCA whitening of both a test sample vector, and all CPs.

B.4.4 Decorrelating Class-prototypes

The score in Eqn. (2) can be derived in a manner that highlights the similarities and differences to the Mahalanobis transform. Here we use the randomly projected features $\mathbf{h}$ instead of extracted features $\mathbf{f}$. We choose a linear transform matrix $\mathbf{D} \in \mathbb{R}^{M \times M}$ that converts the Gram matrix, $\mathbf{G} = \mathbf{H}\mathbf{H}^\top$, to the identity matrix, i.e. $\mathbf{D}$ must satisfy

$$(\mathbf{D}\mathbf{H})(\mathbf{D}\mathbf{H})^\top = \mathbf{I}_{M \times M}. \qquad (24)$$

It is easy to see that $\mathbf{D}^\top\mathbf{D} = (\mathbf{H}\mathbf{H}^\top)^{-1} = \mathbf{G}^{-1}$. Next, treating test samples and class prototypes as originating from the same distribution as $\mathbf{H}$, we can consider the dot products

$$\begin{aligned} \hat{s}_y &= (\mathbf{D}\mathbf{h}_{\text{test}})^\top (\mathbf{D}\mathbf{c}_y) \\ &= \mathbf{h}_{\text{test}}^\top \mathbf{G}^{-1}\mathbf{c}_y, \qquad y = 1, \ldots, K. && (25) \end{aligned}$$

Hence, the same similarities arise as those derivable from the minimum mean square error formulation of Eqn. (14). When compared with the Mahalanobis transform, $\mathbf{D}_{\text{Z}} = \mathbf{S}^{-0.5}$, the difference here is that the Gram matrix becomes equal to the identity rather than the covariance matrix.

LDA is the same as our method if all classes are equiprobable and $\mathbf{G} = \mathbf{S}$, which happens if all M features have a mean of zero, which is not true in general.

Appendix C Training and implementation

C.1 Optimizing ridge regression parameter in Algorithm 1

With reference to the final step in **Algorithm 1**, we optimized λ as follows. For each task, t, in Phase 2, the training data for that task was randomly split in the ratio 80:20. We parameter swept over 17 orders of magnitude, namely $\lambda \in \{10^{-8}, 10^{-7}, \ldots, 10^{8}\}$ and for each value of λ used $\mathbf{C}$ and $\mathbf{G}$ updated with only the first 80% of the training data for task t to then calculate $\mathbf{W}_{\text{o}} = (\mathbf{G} + \lambda\mathbf{I})^{-1}\mathbf{C}$.

We then calculated the mean square error between targets and the set of predictions of the form $\mathbf{h}^\top \mathbf{W}_\mathrm{o}$ for the remaining 20% of the training data. We then chose the value of λ that minimized the mean square error on this 20% split. Hence, λ is updated after every task, and computed in a manner compatible with CL, i.e. without access to data from previous training tasks. It is worth noting that optimizing λ to a value between orders of magnitude will potentially slightly boost accuracies. Note also that choosing λ only for data from the current task may not be optimal relative to non-CL learning on the same data, in which case the obvious difference would be to optimize λ on a subset of training data from the entire training set.

C.2 Training details

For Phase 2 in **Algorithm 1**, the training data was used as follows. For each sample, features were extracted from a frozen pretrained model, in order to update the $\mathbf{G}$ and $\mathbf{C}$ matrices. We then computed $\mathbf{W}_\mathrm{o}$ using matrix inversion and multiplication. Hence, no SGD based weight updates are required in Phase 2.

For Phase 1 in **Algorithm 1**, we used SGD to train the parameters of PETL methods, namely AdaptFormer [6], SSF [28], and VPT [21]. For each of these, we used batch sizes of 48, a learning rate of 0.01, weight decay of 0.0005, momentum of 0.9, and a cosine annealing schedule that finishes with a learning rate of 0. Generally we trained for 20 epochs, but in some experiments reduced to fewer epochs if overfitting was clear. When using these methods, softmax and cross-entropy loss was used. The number of classes was equal to the number in the first task, i.e. N_1. The resulting trained weights and head were discarded prior to commencing Phase 2 of **Algorithm 1**.

For reported data in Table 1 using linear probes we used batch sizes of 128, a learning rate of 0.01 in the classification head, weight decay of 0.0005, momentum of 0.9 and training for 30 epochs. For full fine-tuning (Table 2), we used the same settings, but additionally used a learning rate in the body (the pre-trained weights of the ViT backbone) of 0.0001. We used a fixed learning rate schedule, with no reductions in learning rate. We found for fine-tuning that the learning rate lower in the body than the head was essential for best performance.

Data augmentation during training for all datasets included random resizing then cropping to 224×224 pixels, and random horizontal flips. For inference, images are resized to short side 256 and then center-cropped to 224×224 for all datasets except CIFAR100, which are simply resized from the original 32×32 to 224×224.

Given our primary comparison is with results from [64], we use the same seed for our main experiments, i.e. a seed value of 1993. This enables us to obtain identical results as [64] in our ablations. However, we note that we use Average Accuracy as our primary metric, whereas [64] in their public repository calculate overall accuracy after each task which can be slightly different to Average Accuracy. For investigation of variability in Section F, we also use seeds 1994 and 1995.

C.3 Training and Inference Speed

The speed for inference with RanPAC is negligibly different to the speed of the original pre-trained network, because both the RP layer and the output linear classification head (comprised from decorrelated class prototypes) are implemented as simple fully-connected layers on top of the underlying network. For training, Phase 1 trains PETL parameters using SGD for 20 epochs, on $(1/T)$'th of the training set, so is much faster than joint training. Phase 2 is generally only slightly slower than running all training data through the network in inference mode, because the backbone is frozen. The slowest part is the inversions of the Gram matrix, during selection of λ, but even for $M = 10000$, this is in the order of 1 minute per task on a CPU, which can be easily optimized further if needed. Our view is that the efficiency and simplicity of our approach compared to the alternatives is very strong.

C.4 Compute

All experiments were conducted on a single PC running Ubuntu 22.04.2 LTS, with 32 GB of RAM, and Intel Core i9-13900KF x32 processor. Acceleration was provided by a single NVIDIA GeForce 4090 GPU.

Appendix D Parameter-Efficient Transfer Learning (PETL) methods

We experiment with the same three methods as [64], i.e. AdaptFormer [6], SSF [28], and VPT [21]. Details can be found in [64]. For VPT, we use the deep version, with prompt length 5. For AdaptFormer, we use the same settings as in [64], i.e. with projected dimension equal to 64.

Appendix E Datasets

E.1 Class Incremental Learning (CIL) Datasets

The seven CIL datasets we use are summmarised in Table A2. For Imagenet-A, CUB, Omnibenchmark and VTAB, we used specific train-validation splits defined and outlined in detail by [64]. Those four datasets, plus Imagenet-R (created by [55]) were downloaded from links provided at `https://github.com/zhoudw-zdw/RevisitingCIL`. CIFAR100 was accessed through torchvision. Stanford cars was downloaded from `https://ai.stanford.edu/~jkrause/cars/car_dataset.html`.

For Stanford Cars and $T = 10$, we use 16 classes in $t = 1$ and 20 in the 9 subsequent tasks. It is interesting to note that VTAB has characteristics of both CIL and DIL. Unlike the three DIL datasets we use, VTAB introduces new disjoint sets of classes in each of 5 tasks that originate in different domains. For this reason we use only $T = 5$ for VTAB, whereas we explore $T = 5$, $T = 10$ and $T = 20$ for the other CIL datasets.

	Original	CL version	N	# val samples	K
CIFAR100	[26]	[42]	50000	10000	100
Imagenet-R	[13]	[55]	24000	6000	200
Imagenet-A	[14]	[64]	5981	5985	200
CUB	[51]	[64]	9430	2358	200
OmniBenchmark	[63]	[64]	89697	5985	300
VTAB	[61]	[64]	1796	8619	50
Stanford Cars	[25]	[62]	8144	8041	196

Table A2: **CIL Datasets.** We list references for the original source of each dataset and for split CL versions of them. In the column headers, N is the total number of training samples, K is the number of classes following training on all tasks, and # val samples is the number of validation samples in the standard validation sets.

E.2 Domain Incremental Learning (DIL) Datasets

For DIL, we list the domains for each dataset in Table A3. Further details can be found in the cited references in the first column of Table A3. As in previous work, validation data includes samples from each domain for CDDB-Hard, and DomainNet, but three entire domains are reserved for CORe50.

Appendix F Additional results

F.1 Preliminaries

We provide results measured by Average Accuracy and Average Forgetting. Average Accuracy is defined as [30]

$$A_t = \frac{1}{t}\sum_{i=1}^{t} R_{t,i}, \tag{26}$$

where $R_{t,i}$ are classification accuracies on the i–th task, following training on the t-th task. Average Forgetting is defined as [3]

$$F_t = \frac{1}{t-1}\sum_{i=1}^{t-1} \max_{t' \in \{1,2,\ldots,t-1\}} (R_{t',i} - R_{t,i}). \tag{27}$$

	K	T	N	$N_t,\ t=1,\dots,T$	N_{val}	val set
CORe50 [29] (Nearly class balanced) ($\sim$ 2400 / class)	50	8	119894		44972	S3, S7, S10
				$N_1 = 14989$ (S1)		
				$N_2 = 14986$ (S2)		
				$N_3 = 14995$ (S4)		
				$N_4 = 14966$ (S5)		
				$N_5 = 14989$ (S6)		
				$N_6 = 14984$ (S8)		
				$N_7 = 14994$ (S9)		
				$N_8 = 14991$ (S11)		
CDDB-Hard [27] (Class balanced both train and val)	2	5	16068		5353	Standard
				$N_1 = 6000$ (gaugan)	2000	
				$N_2 = 2400$ (biggan)	800	
				$N_3 = 6208$ (wild)	2063	
				$N_4 = 1200$ (whichfaceisreal)	400	
				$N_5 = 260$ (san)	90	
DomainNet [40] (Imbalanced)	345	6	409832		176743	Standard
				$N_1 = 120906$ (Real)	52041	
				$N_2 = 120750$ (Quickdraw)	51750	
				$N_3 = 50416$ (Painting)	21850	
				$N_4 = 48212$ (Sketch)	20916	
				$N_5 = 36023$ (Infograph)	15582	
				$N_6 = 33525$ (Clipart)	14604	

Table A3: **DIL Datasets.** K is the number of classes, all of which are included in each task. T is the total number of tasks, N is the total number of training samples across all tasks and N_{val} is the number of validation samples, either overall (first row per dataset) or per task. Also shown are the number of training samples in each task, N_t, and the domain names for the corresponding tasks. Core50 was downloaded from `http://bias.csr.unibo.it/maltoni/download/core50/core50_imgs.npz`. CDDB was downloaded from `https://coral79.github.io/CDDB_web/`. DomainNet was downloaded from `http://ai.bu.edu/M3SDA/#dataset` – we used the version labelled as "cleaned version, recommended."

Note that for CIL, we calculate the $R_{t,i}$ as the accuracy for the subset of classes in $\mathcal{D}_i$.

For DIL, since all classes are present in each task, $R_{t,i}$ has a different nature, and is dependent on dataset conventions. For CORe50, the validation set consists of three domains not used during training (S3, S7 and S10, as per Table A3), and in this case, each $R_{t,i}$ is calculated on the entire validation set. Therefore $R_{t,i}$ is constant for all i and $A_t = R_{t,0}$. For CDDB-Hard and DomainNet, for the results in Table 3 we treated the validation data in the same way as for CORe50. However, it is also interesting to calculate accuracies for validation subsets in each domain – we provide such results in the following subsection.

F.2 Variability and performance on each task

Fig. A2 shows, for three different random seeds, Average Accuracies and Average Forgetting after each CIL task matching Table 1 in Section 5, without PETL (Phase 1). Due to not using PETL, the only random variability is (i) the choice of which classes are randomly assigned to which task and (ii) the random values sampled for the weights in $\mathbf{W}$. We find that after the final task, the Average Accuracy is identical for each random seed when all classes have the same number of samples, and are nearly identical otherwise. Clearly the randomness in $\mathbf{W}$ has negligible impact. For all datasets except VTAB, the value of RP is clear, i.e. RP (black traces) delivers higher accuracies by the time of the final task than not using RP, or when only using NCM (blue traces). The benefits of second-order statistics even without RP are also evident (magenta traces), with Average Accuracies better than NCM. Note that VTAB always has the same classes assigned to the same tasks, which is why only one repetition is shown. The difference with VTAB is also evident in the average accuracy trend as the number of tasks increases, i.e. the Average Accuracy does not have a clearly decreasing trend as the number of tasks increases. This is possibly due to the difference in domains for each Task making it less likely that confusions occur for classes in one task against those in another task.

Fig. A3 shows comparisons when Phase 1 (PETL) is used. The same trends are apparent, except that due to the SGD training required for PETL parameters, greater variability in Average Accuracy after the final task can be seen. Nevertheless, the benefits of using RP are clearly evident.

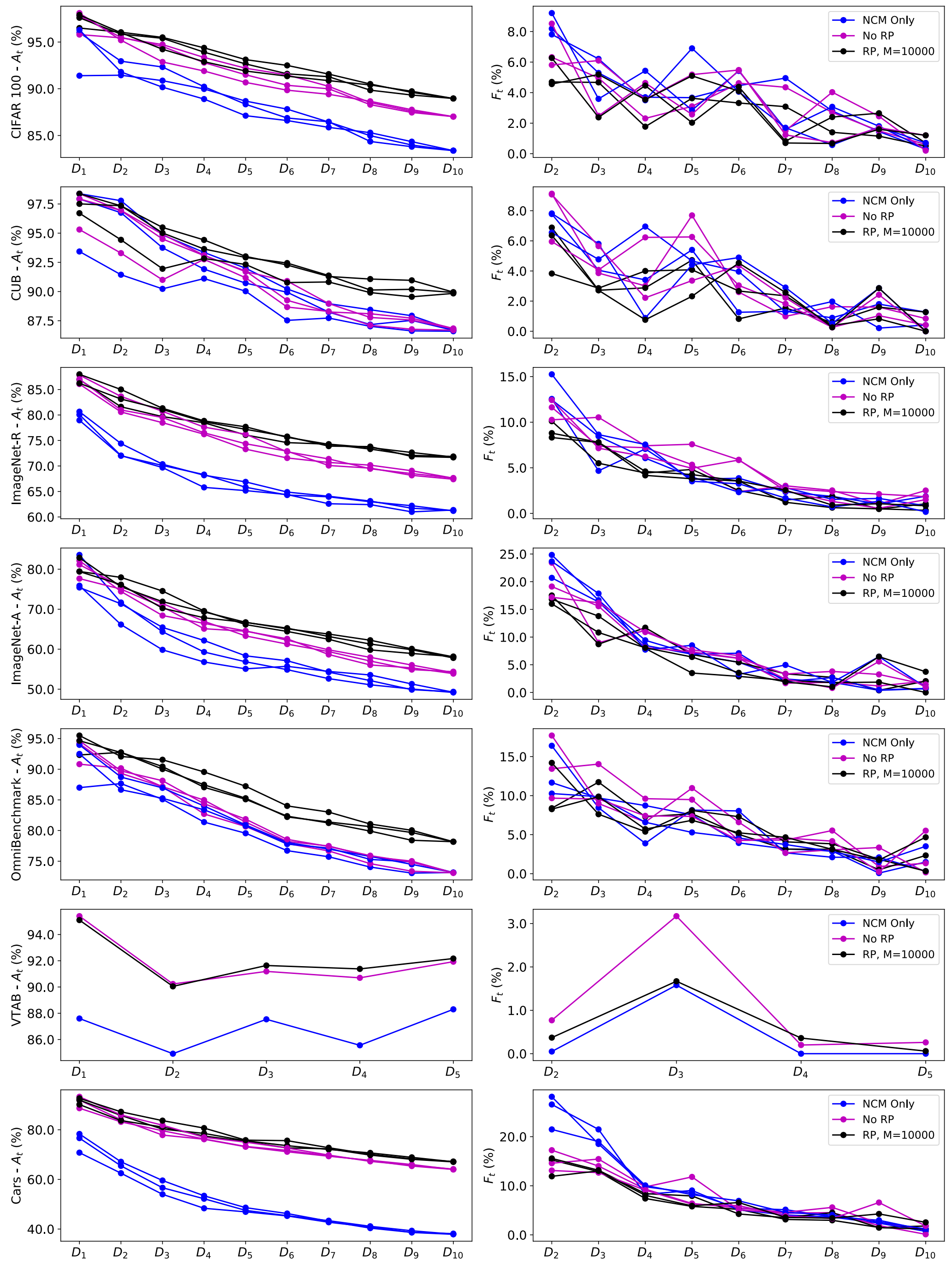

Figure A2: **Average Accuracies and Forgetting after each Task for CIL datasets (no Phase 1).** The left column shows Average Accuracies for three random seeds after each of T tasks for the seven CIL datasets, without using Phase 1. The right column shows Average Forgetting.

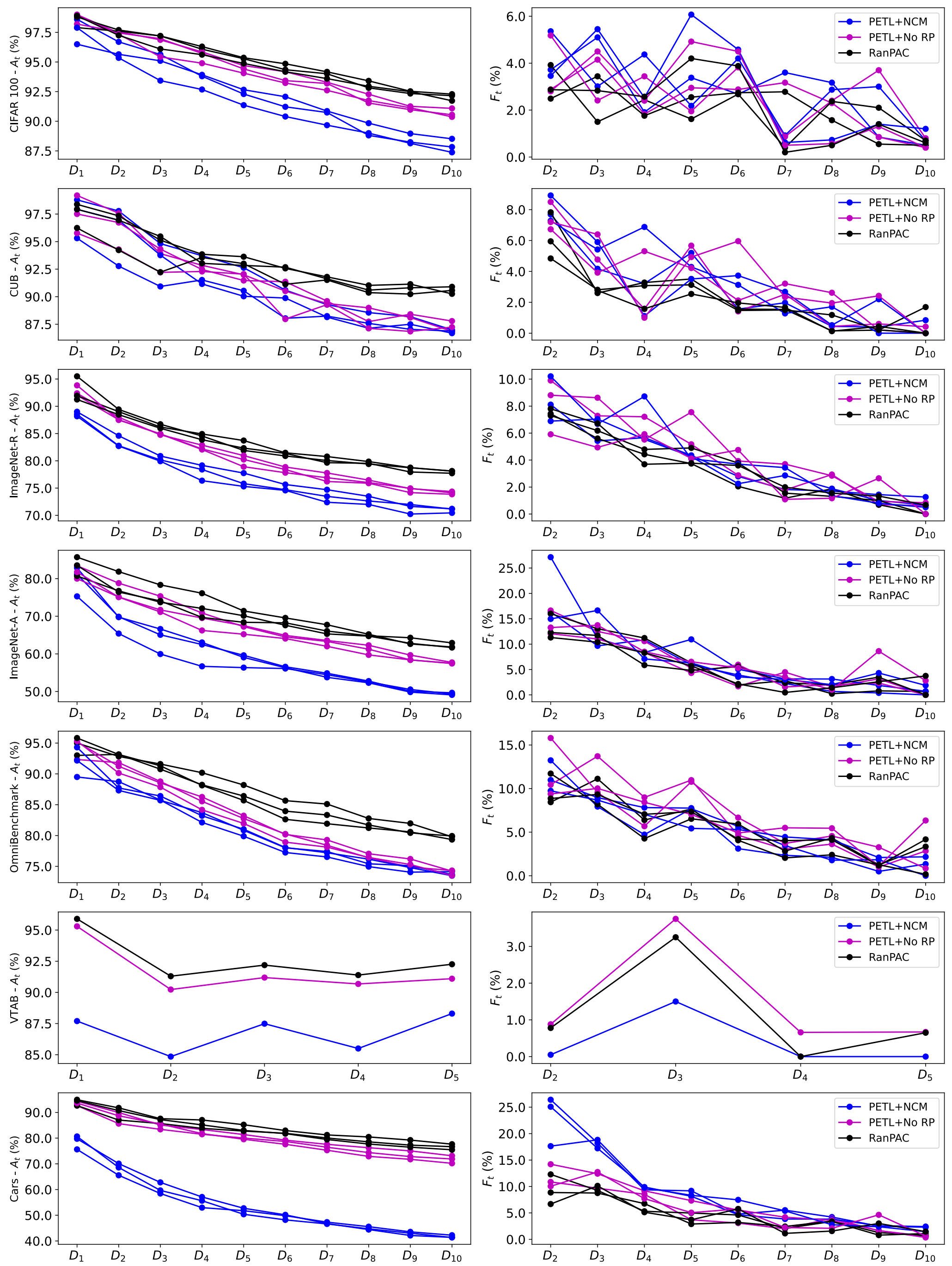

Figure A3: **Average Accuracies and Forgetting after each Task for CIL datasets (with Phase 1).** The left column shows Average Accuracies for three random seeds after each of T tasks for the seven CIL datasets, for the best choice of PETL method in Phase 1. The right column shows Average Forgetting.

Fig. A4 shows how accuracy on individual domains changes as tasks are added through training for the DIL dataset, CDDB-Hard. The figure shows that after training data for a particular domain is first used, that accuracy on the corresponding validation data for that domain tends to increase. In some cases forgetting is evident, e.g. for the 'wild' domain, after training on T_4 and T_5. The figure also shows that averaging accuracies for individual domains ('mean over domains') is significantly lower than 'Overall accuracy'. For this particular dataset, 'Average accuracy' is potentially a misleading metric as it does not take into account the much lower number of validation samples in some domains, e.g. even though performance on 'san' increases after training on it, it is still under 60%, which is poor for a binary classification task.

Fig. A5 shows the same accuracies by domain for DomainNet. Interestingly, unlike CDDB-Hard, accuracy generally increases for each Domain as new tasks are learned. This suggests that the pre-trained model is highly capable of forming similar feature representation for the different domains in DomainNet, such that increasing amounts of training data makes it easier to discriminate between classes. The possible difference with CDDB-Hard is that that dataset is a binary classification task in which discriminating between the 'real' and 'fake' classes is inherently more difficult and not reflected in the data used to train the pre-trained model.

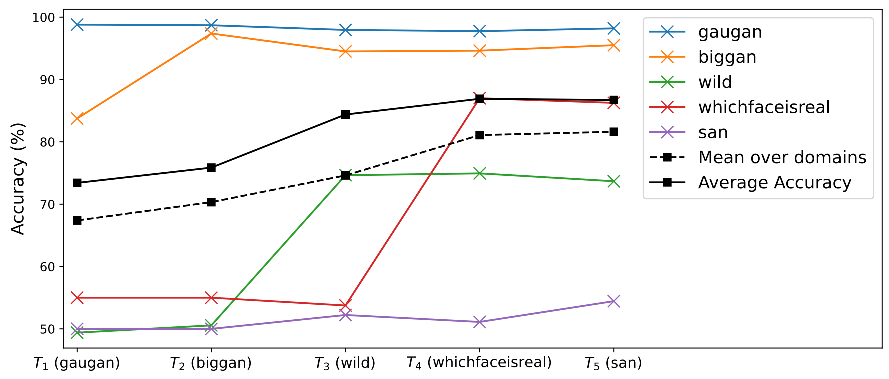

Figure A4: **Accuracies for DIL dataset CDDB-Hard.** Results are shown for the full RanPAC algorithm, using VPT as the PETL method, and $M = 10000$. Each task, T_i corresponds to learning on a new domain as shown on the x-axis. The accuracies shown in colors are those for individual domains, following training on the domain shown on the x-axis. 'Mean over domains' is the average of the five domain accuracies after each task.

F.3 Comparison of impact of Phase 1 for different numbers of CIL tasks

As shown in Fig. A2, when Phase 1 is excluded, the *final* Average Accuracy after the final task has negligible variability despite different random assignments of classes to tasks. This is a consequence primarily of Eqns. (4) being invariant to the order in which a completed set of data from all T tasks is used in **Algorithm 1**. As mentioned, the influence of different random values in $\mathbf{W}$ is negligible. The same effect occurs if the data is split to different numbers of tasks, e.g. $T = 5$, $T = 10$ and $T = 20$, in the event that all classes have equal number of samples, such as CIFAR100.

Therefore, in this section we show in Table A4 results for RanPAC only for the case where variability has greater effect, i.e. when Phase 1 is included, and AdaptMLP chosen. VTAB is excluded from this analysis, since it is a special case where it makes sense to consider only the case of $T = 5$. The comparison results for L2P, DualPrompt and ADaM are copied from [64].

Performance when AdaptMLP is used tends to be better for $T = 5$ and worse for $T = 20$. This is consistent with the fact that more classes are used for first-session training with the PETL method when $T = 5$. By comparison, $T = 20$ is often on par with not using the PETL method at all, indicating that the first session strategy may have little value if insufficient data diversity is available within the first task.

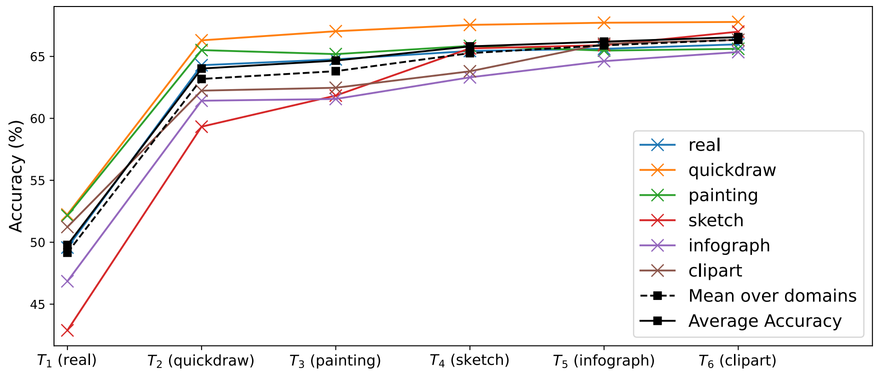

Figure A5: **Accuracies for DIL dataset DomainNet.** Results are shown for the full RanPAC algorithm, using VPT as the PETL method, and $M = 10000$. Each task, T_i, corresponds to learning on a new domain as shown on the x-axis. The accuracies shown in colors are those for individual domains, following training on the domain shown on the x-axis. 'Mean over domains' is the average of the six domain accuracies after each task.

		$T = 5$	$T = 10$	$T = 20$	RanPAC – No Phase 1
CIFAR100	RanPAC	92.4%	92.2%	90.8%	89.0%
	AdaM	88.5%	87.5%	85.2%	
	DualPrompt	86.9%	84.1%	81.2%	
	L2P	87.0%	84.6%	79.9%	
	NCM	88.6%	87.8%	86.6%	83.4%
ImageNet-R	RanPAC	79.9%	77.9%	74.5%	71.8%
	AdaM	74.3%	72.9%	70.5%	
	DualPrompt	72.3%	71.0%	68.6%	
	L2P	73.6%	72.4%	69.3%	
	NCM	74.0%	71.2%	64.7%	61.2%
ImageNet-A	RanPAC	63.0%	58.6%	58.9%	58.2%
	AdaM	56.1%	54.0%	51.5%	
	DualPrompt	46.6%	45.4%	42.7%	
	L2P	45.7%	42.5%	38.5%	
	NCM	54.8%	49.7%	49.3%	49.3%
CUB	RanPAC	90.6%	90.3%	89.7%	89.9%
	AdaM	87.3%	87.1%	86.7%	
	DualPrompt	73.7%	68.5%	66.5%	
	L2P	69.7%	65.2%	56.3%	
	NCM	87.0%	87.0%	86.9%	86.7%
OmniBenchmark	RanPAC	79.6%	79.9%	79.4%	78.2%
	AdaM	75.0%*	74.5%	73.5%	
	DualPrompt	69.4%*	65.5%	64.4%	
	L2P	67.1%*	64.7%	60.2%	
	NCM	75.1%	74.2%	73.0%	73.2%
Cars	RanPAC	69.6%	67.4%	67.2%	67.1%
	NCM	41.0%	38.0%	37.9%	37.9%

Table A4: **Comparison of CIL results for different number of tasks.** For this table, the AdaptMLP PETL method was used for RanPAC. The comparison results for L2P, DualPrompt and ADaM are copied from [64]; for ADaM, the best performing PETL method was used. In most cases, Average Accuracy decreases as T increases, with $T = 20$ typically not significantly better than the case of no PETL ("No Phase 1"). Values marked with * indicates data is for $T = 6$ tasks, instead of $T = 5$. Note that accuracies for comparison methods for Cars were not available from [64]

F.4 Task Agnostic Continual Learning

Unlike CIL and DIL, 'task agnostic' CL is a scenario where there is no clear concept of a 'task' during training [60]. The concept is also known as 'task-free' [44]. It contrasts with standard CIL where although inference is task agnostic, training is applied to disjoint sets of classes, described as tasks. To illustrate the flexibility of RanPAC, we show here that it is simple to apply it to task agnostic continual learning. We use the Gaussian scheduled CIFAR100 protocol of [56], which was adapted from [44]. We use 200 'micro-tasks' which sample from a gradually shifting subset of classes, with 5 batches of 48 samples in each micro-task. There are different possible choices for how to apply **Algorithm 1**. For instance, the 'first session' for Phase 1 could be defined as a particular total number of samples trained on, e.g. 10% of the anticipated total number of samples. Then in Phase 2, the outer for loop over tasks could be replaced by a loop over all batches, or removed entirely. In both cases, the result for $\mathbf{G}$ and $\mathbf{C}$ will be unaffected. The greater challenge is in determining λ, but generally for a large number of samples, λ can be small, or zero. For a small number of training samples, a queue of samples could be maintained such that the oldest sample in the queue is used to update $\mathbf{G}$ and $\mathbf{C}$, with all newer samples used to compute λ if inference is required, and then all samples in the buffer added to $\mathbf{G}$ and $\mathbf{C}$.

Here, for simplicity we illustrate application to Gaussian-scheduled CIFAR100 without any Phase 1. Fig. A6 shows how test accuracy changes through training both with and without RP. The green trace illustrates how the number of classes seen in at least one sample increases gradually through training, instead of in a steps like in CIL. The red traces show validation accuracy on the entirety of the validation set. As expected, this increases as training becomes exposed to more classes. The black traces show the accuracy on only the classes seen so far through training. By the end of training, the red and black traces converge, as is expected. Fluctuations in black traces can be partially attributed to not optimizing λ. The final accuracies with and without RP match the values for $T = 10$ CIL shown in Table 1.

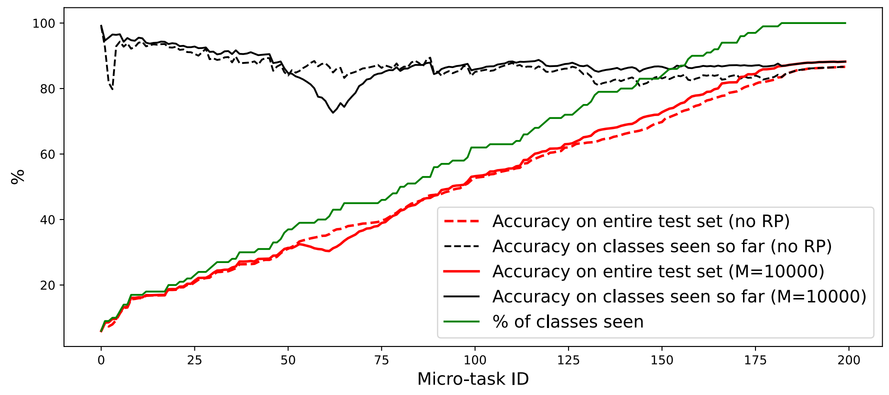

Figure A6: **Task agnostic example**. Application of Phase 2 of **Algorithm 1** to the Gaussian-scheduled CIFAR100 task-agnostic CL protocol.

F.5 Scaling with projection size

Table A5 shows, for the example of split CIFAR100, that it is important to ensure M is sufficiently large. In order to surpass the accuracy obtained without RP (see 'No RPs or Phase 1' in Table 1), M needs to be larger than 1250.

F.6 Comparison of PETL Methods and ViT-B/16 backbones

Fig. A7 shows how performance varies with PETL method and by ViT-B/16 backbone. For some datasets, there is a clearly superior PETL method. For example, for CIFAR100, AdaptMLP gives better results than SSF or VPT, for both backbones, for Cars VPT is best, and for ImageNet-A, SSF

M	**Accuracy**
100	71.6%
200	80.3%
400	83.9%
800	86.2%
1250	86.8%
2500	87.7%
5000	88.4%
10000	88.8%
15000	89.0%

Table A5: **Scaling with M for split CIFAR100.** The table shows final average accuracy for $T = 10$ and no Phase 1, with $\lambda = 100$ as a constant for all M, using ViT-B/16 trained on ImageNet21K.

is best. There is also an interesting outlier for VTAB, where VPT and the ImageNet-21K backbone failed badly. This variability by PETL method suggests that the choice of method for first-session CL strategies should be investigated in depth in future work.

Fig. A7 also makes it clear that the same backbone is not optimal for all datasets. For example, the ViT-B/16 model fine-tuned on ImageNet-1K is best for ImageNet-A, but the ViT-B/16 model trained on ImageNet-21K is best for CIFAR100 and OmniBenchmark. For Cars, the best backbone depends on the PETL method.

Fig. A8 summarises how the two ViT networks compare. For all datasets and method variants we plot the Average Accuracy after the final task for one pre-trained ViT network (self-supervied on ImageNet-21K) against the other (fine-tuned on ImageNet-1K). Consistent with Fig. A7, the best choice of backbone is both dataset dependent and method-dependent.

F.7 Experiments with ResNets

Unlike prompting strategies, our approach works with any feature extractor, e.g. both pre-trained transformer networks and pre-trained convolutional neural networks. To illustrate this, Table A6 and Table A7 shows results for ResNet50 and ResNet 152, respectively, pre-trained on ImageNet. We used $T = 10$ tasks (except for VTAB, which is $T = 5$ tasks). Although this is different to the $T = 20$ tasks used for ResNets by [64], the accuracies we report for NCM are very comparable to those in [64]. As with results for pre-trained ViT-B/16 networks, the use of random projections and second-order statistics both provide significant performance boosts compared with NCM alone. We do not use Phase 1 of **Algorithm 1** here but as shown by [37, 64], this is feasible for diverse PETL methods for convolutional neural networks. Interestingly, ResNet152 with RP produces reduced accuracies compared to ResNet50 on CUB, Omnibenchmark and VTAB. It is possible that this would be remedied by seeking an optimal value of M, whereas for simplicity we chose $M = 10000$. Note that unlike the pre-trained ViT-B/16 model, the pre-trained ResNets require preprocessing to normalize the input images. We found, however, that this is best removed for VTAB.

Method	**CIFAR100**	**IN-R**	**IN-A**	**CUB**	**OB**	**VTAB**	**Cars**
Phase 2	77.5%	56.8%	36.1%	70.9%	68.0%	88.8%	48.6%
Phase 2 (No RPs)	73.3%	55.0%	37.0%	62.6%	60.27%	87.7%	42.7%
NCM	61.5%	43.1%	25.8%	55.3%	57.4%	81.2%	26.2%

Table A6: **Results for ResNet50.** Results are final Average Accuracies for $T = 10$ tasks, except VTAB which is $T = 5$ tasks.

Method	**CIFAR100**	**IN-R**	**IN-A**	**CUB**	**OB**	**VTAB**	**Cars**
Phase 2	79.7%	59.0%	41.4%	66.6%	65.8%	88.0%	45.1%
Phase 2 (No RPs)	77.8%	57.1%	40.4%	59.4%	58.0%	87.8%	38.8%
NCM	70.1%	47.6%	31.9%	50.4%	56.1%	82.0%	25.1%

Table A7: **Results for ResNet152.** Results are final Average Accuracies for $T = 10$ tasks, except VTAB which is $T = 5$ tasks.

F.8 Experiments with CLIP vision model

To further verify the general applicability of our method, we show results for a CLIP [41] vision model as the backbone pre-trained model in Table A8. The same general trend as for pre-trained ViT-B/16 models and ResNets can be seen, where use of RPs (Phase 2 in **Algorithm 1**) produces better accuracies than NCM alone. Interestingly, the results for Cars is substantially better with the CLIP vision backbone than for ViT-B/16 networks. It is possible that data from a very similar domain as the Cars dataset was included in training of the CLIP vision model. The CLIP result for Phase 2 only (see ablations in Table 1) is also better for Imagenet-R than for ViT/B-16, but for all other datasets, ViT/B-16 has higher accuracy. Note that unlike the pre-trained ViT-B/16 model, the pre-trained CLIP vision model requires preprocessing to normalize the input images.

Method	**CIFAR100**	**IN-R**	**IN-A**	**CUB**	**OB**	**VTAB**	**Cars**
Phase 2	85.0%	76.5%	40.2%	79.4%	75.1%	90.5%	90.5%
Phase 2 (No RPs)	80.9%	67.4%	37.2%	72.7%	66.3%	89.8%	85.9%
NCM	77.1%	68.1%	32.6%	73.8%	67.8%	78.8%	85.6%

Table A8: **Results for CLIP-ViT-B/16.** Results are final Average Accuracies for $T = 10$ tasks, except VTAB which is $T = 5$ tasks. We used the pre-trained weights from OpenCLIP, finetuned on ImageNet-1K [19].

F.9 Experiments with regression targets using CLIP vision and language models

Until this point, we have defined the matrix $\mathbf{C}$ as containing Class-Prototypes (CPs), i.e. $\mathbf{C}$ has N columns representing averaged feature vectors of length M. However, with reference to Eqn. (13), the assumed targets for regression, $\mathbf{Y}_{\text{train}}$, can be replaced by different targets. Here, we illustrate this using CLIP language model representations as targets, using OpenAI's CLIP ViT-B/16 model.

Using CIFAR100 as our example, we randomly project CLIP's length-512 vision model representations as usual, but also use the length-512 language model's representations of the 100 class names, averaged over templates as in [41]. We create a target matrix of size $N \times 512$ in which each row is the language model's length 512 representation for the class of each sample. We then solve for $\mathbf{W}_{\text{o}} \in \mathcal{R}^{M \times 512}$ using this target instead of $\mathbf{Y}_{\text{train}}$.

When the resulting $\mathbf{W}_{\text{o}}$ is applied to a test sample, the result is a length-512 prediction of a language model representation. In order to translate this to a class prediction, we then apply CLIP in a standard zero-shot manner, i.e. we calculate the cosine similarity between the predictions and each of the normalized language model's length 512 representation for the class of each sample.

The resulting final Average Accuracy for $M = 5000$ and $T = 10$ is 77.5%. In comparison, CLIP's zero shot accuracy for the same data is 68.6%, which highlights there is value in using the training data to modify the vision model's outputs. When RP is not used, the resulting final Average Accuracy is 71.4%.

In future work, we will investigate whether these preliminary results from applying **Algorithm 1** to a combination of pre-trained vision and language models can translate into demonstrable benefits for continual learning.

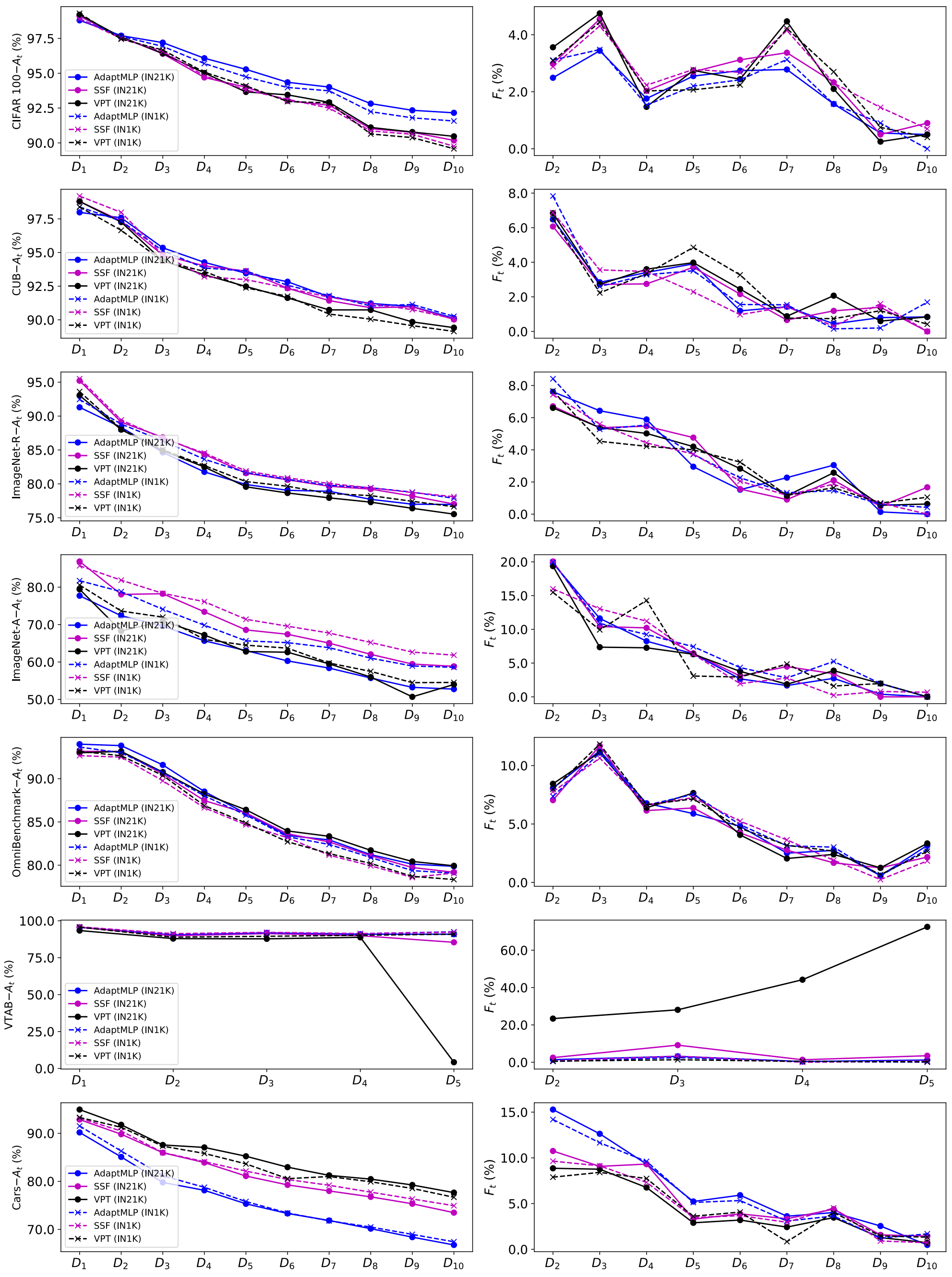

Figure A7: **Comparison of PETL Methods and two ViT models**. For the seven CIL datasets, and $T = 10$, the figure shows Average Accuracy and Average Forgetting after each task, for each of AdaptMLP, SSF and VPT. It shows this information for the two ViT-B/16 pre-trained backbones we primarily investigated.

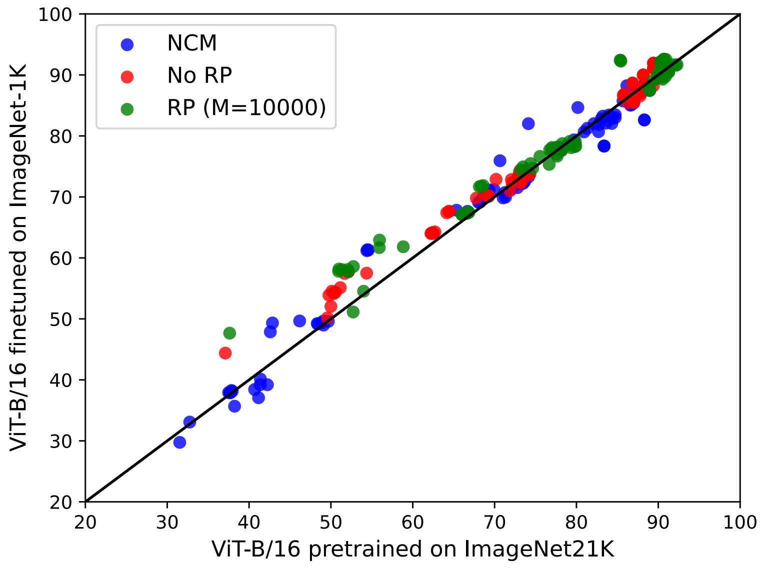

Figure A8: **Comparison of backbone ViTs.** Scatter plot of results for ViT-B/16 models pre-trained on ImageNet1K vs ImageNet21K.