Summary for continual&lifelong learning

Paper list

Survey

- *Three scenarios for continual learning*
- *A Comprehensive Survey of Continual Learning: Theory, Method and Application*
- *Deep Class-Incremental Learning: A Survey*

Classic

- *iCaRL: Incremental Classifier and Representation Learning*
- *Gradient Episodic Memory for Continual Learning*
- *Efficient Lifelong Learning with A-GEM*
- *DER: Dynamically Expandable Representation for Class Incremental Learning*
- 李宏毅机器学习*2022 lifelong learning*

Pre-trained

- *DyTox: Transformers for Continual Learning with DYnamic TOken eXpansion*
- *ELLE: Efficient Lifelong Pre-training for Emerging Data*
- *A Unified Continual Learning Framework with General Parameter-Efficient Tuning*
- *Revisiting a kNN-based Image Classification System with High-capacity Storage*
- *Revisiting Class-Incremental Learning with Pre-Trained Models: Generalizability and Adaptivity are All You Need*

Prompt-based

- *S-Prompts Learning with Pre-trained Transformers: An Occam's Razor for Domain Incremental Learning*
- *Learning to Prompt for Continual Learning*
- *DualPrompt: Complementary Prompting for Rehearsal-free Continual Learning*

Definition

持续/终身学习（下面简称持续学习）就是让模型从连续信息流中学习的一种方法，就是训练数据不会一次性提供，而是会一批一批按照顺序提供，让模型渐进式地学习且适应这些数据。

在持续学习中需要注意两方面的问题：

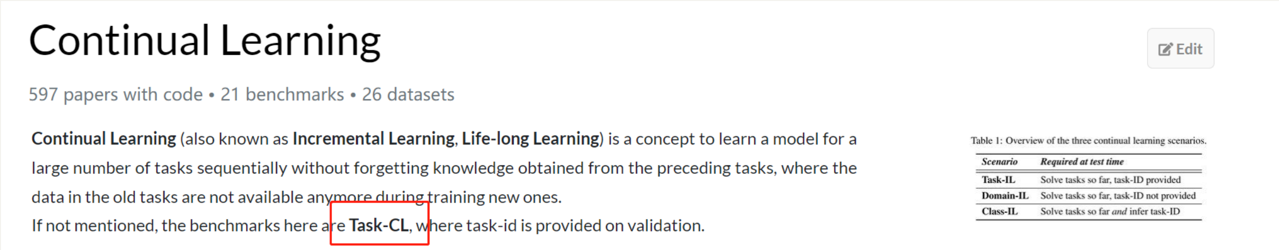
- 把之前学到的知识拓展到新任务/新数据上，也就是对新知识学的好 --> 可塑性
- 保持对于旧任务的学习能力，也就是不忘记旧知识 ---> 稳定性

👉**continual learning vs. incremental learning vs. lifelong learning**

三者如果不作严格区分的话，在论文里面其实意思是一致的，均可通用。

> *valse webinar:*
>
> - *Incremental:* 学了新东西不要忘了旧东西（*Task- Domain- Class-*）。
> - *Continual/Lifelong:* 可能会比*incremental*有更高的诉求，比如说要求对后面任务也能学习的更好（*zero-shot*能力），或者说学了后面的任务后，对以前的任务也能有所提升。
>
> 如果只是类别和任务的增加，用*incremental*表示更合适；但如果是数据增加，不涉及类别任务的增加，可能更偏向于*continual/lifelong*。
>
> 所以说*continual learning*其实是涵盖更广的一个范围，*lifelong*和*continual*的含义几乎一致。

然而，在paper with code这个网站上，incremental的范围会更广，continual learning单指task-CL，即测试时提供task ID，这种setting会比较简单。



Continual Learning

Edit

597 papers with code • 21 benchmarks • 26 datasets

Continual Learning (also known as **Incremental Learning**, **Life-long Learning**) is a concept to learn a model for a large number of tasks sequentially without forgetting knowledge obtained from the preceding tasks, where the data in the old tasks are not available anymore during training new ones.
If not mentioned, the benchmarks here are **Task-CL**, where task-id is provided on validation.

Table 1: Overview of the three continual learning scenarios.

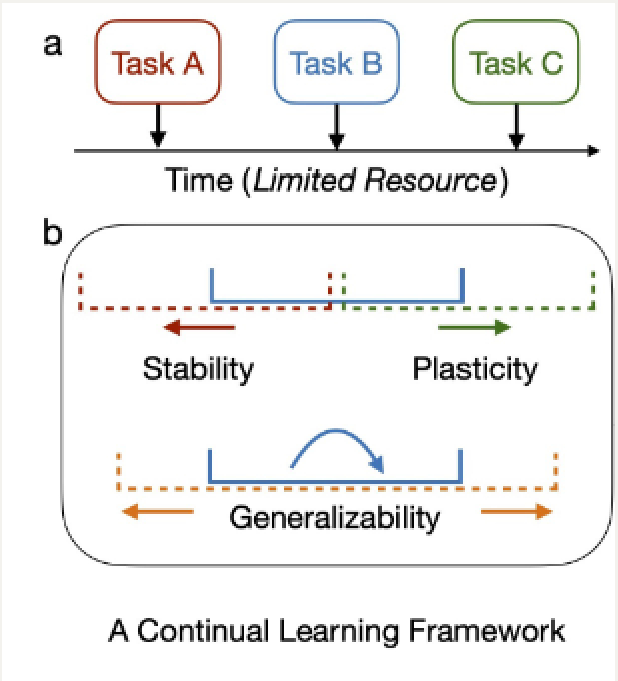
Scenario	*Required at test time*
Task-IL	Solve tasks so far, task-ID provided
Domain-IL	Solve tasks so far, task-ID not provided
Class-IL	Solve tasks so far *and* infer task-ID

👉continual learning vs. transfer learning

从目的上来说，前者指在新旧的任务上都希望能够表现得很好；后者仅希望是应用前面的知识，使得当前任务表现得更好。

Challenge

from *A Comprehensive Survey of Continual Learning: Theory, Method and Application*



- **Catastrophy forgetting (main)** (have no stability)
- **Intransigence** (have no plasticity)
- **resource efficiency:** 不可以用太多空间，把所有的之前的数据全部存起来

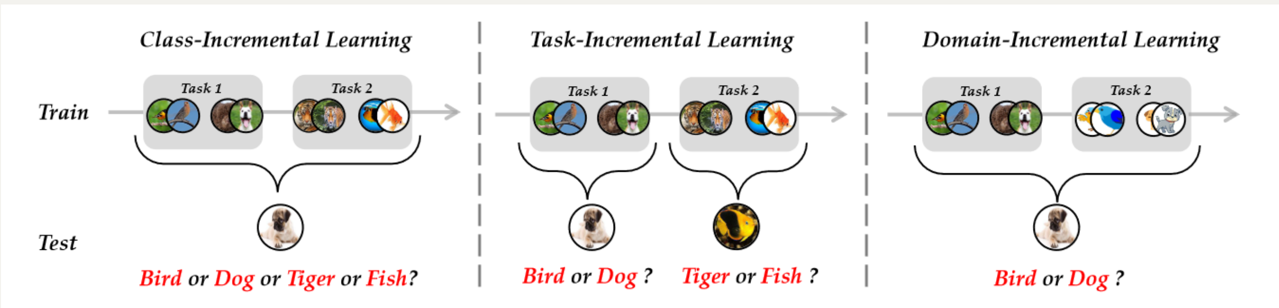
🎯 **goal**: good **generalizability** within and between task。需要在stability和plasticity中做一个trade-off，而且还要考虑资源的利用和限制。

Scenario

from *Three scenarios for continual learning*

Table 1: Overview of the three continual learning scenarios.

Scenario	*Required at test time*
Task-IL	Solve tasks so far, task-ID provided
Domain-IL	Solve tasks so far, task-ID not provided
Class-IL	Solve tasks so far *and* infer task-ID



最有挑战性的是class-IL，也就是说需要模型识别从开始训练到当前任务遇到的所有class，不区分这个class是哪个task提供的，而且大家似乎都愿意往class-IL的方向去做~

然后下面主要介绍的还是**class-IL**这个方面的吧，因为难度最大且相关工作比较多。

📚 Method

💡 **Train from scratch**

from *Deep Class-Incremental Learning: A Survey*

Algorithm Category	Subcategory		Reference
§3.1 Data-Centric Class-Incremental Learning	§3.1.1 Data Replay	Direct Replay	[35], [39], [40], [41], [42], [43], [44], [45], [46], [47]
		Generative Replay	[48], [49], [50], [51], [52], [53], [54], [55], [56], [57], [58]
	§3.1.2: Data Regularization		[40], [59], [60], [61], [62], [63]
§3.2 Model-Centric Class-Incremental Learning	§3.2.1 Dynamic Networks	Neuron Expansion	[64], [65], [66]
		Backbone Expansion	[20], [21], [22], [67], [68], [69], [70], [71]
		Prompt Expansion	[23], [72], [73], [74], [75], [76]
	§3.2.2: Parameter Regularization		[39], [77], [78], [79], [80], [81], [82], [83], [84]
§3.3 Algorithm-Centric Class-Incremental Learning	§3.3.1 Knowledge Distillation	Logit Distillation	[32], [85], [86], [87], [88], [89], [90], [91], [92]
		Feature Distillation	[93], [94], [95], [96], [97], [98], [99], [100]
		Relational Distillation	[101], [102], [103], [104], [105]
	§3.3.2 Model Rectify	Feature Rectify	[106], [107], [108], [109], [110], [111]
		Logit Rectify	[87], [93], [112], [113], [114]
		Weight Rectify	[115], [116], [117]

1. Data-Centric

- **Direct replay**

  都习惯于用一种叫Rehearsal的方式，存一些之前任务的示例样本到memory buffer中，便于今后任务的回顾。

  > *Rehearsal aims to approximate the observed input distributions over time and later resamples from this approximation to avoid forgetting.*

  - ER(Experience Replay)：直接从旧任务的数据中均匀采样，放到memory buffer中。或者采样离feature center最近的一些examplar放进去，然后和当前任务的数据一起训练。
  - iCaRL: 用栈式的思想，动态更新exampler sets，限制了内存的增长。loss进行了改进，对旧任务的exampler数据用distillation的思想（soft targets），对新任务的实际数据用hard target。

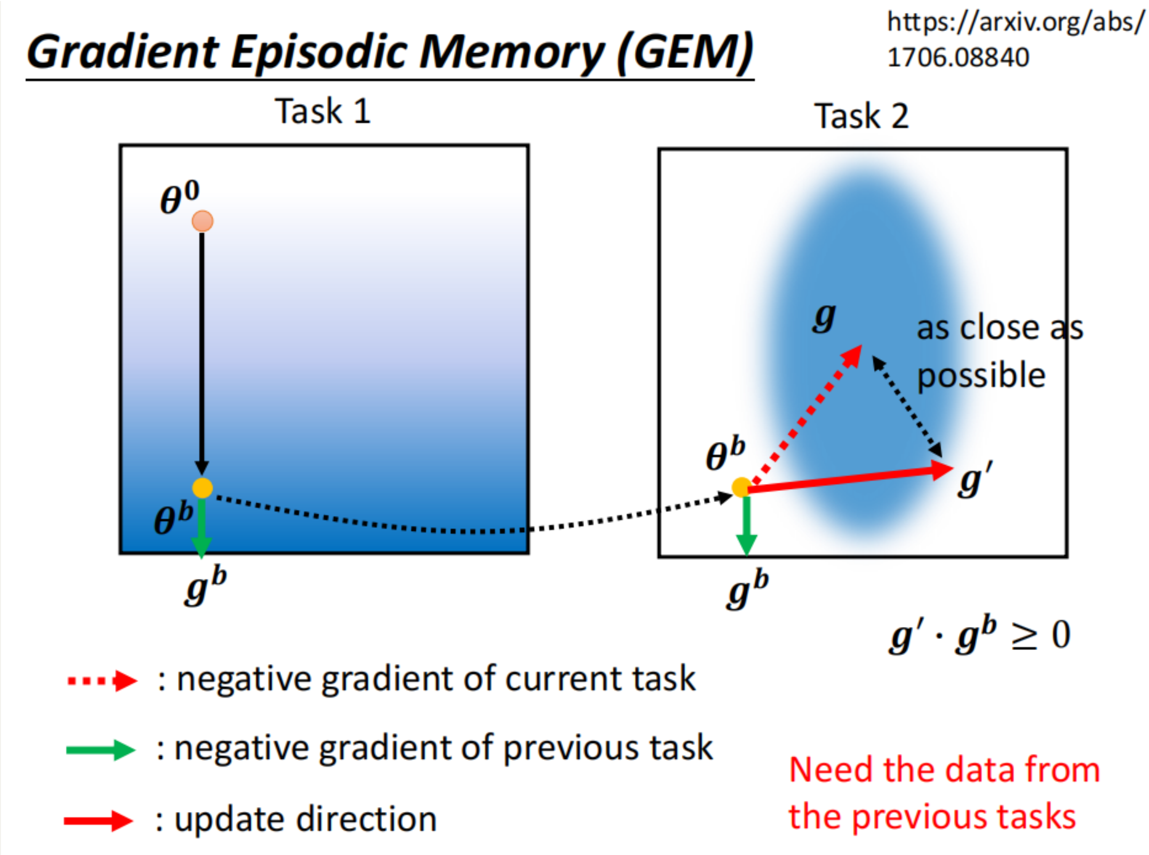
- **Generative Replay**

  会在分类模型之外额外构建一个生成模型，用于学习前后任务的数据分布。但是生成模型同时也会遭受灾难遗忘的问题，反过来又会继续加重分类模型灾难遗忘问题。生成的质量如何保证？

  - GR: 用GAN或者VAE进行伪样本生成，但是每个任务都有一个求解器，需要存这个求解器的参数。
  - FearNet：双记忆系统，分为长期记忆和短期记忆。

- **Data regulization**

  用数据的形态控制优化方向，也是取一定的旧数据放在memory buffer里面，然后计算一下新旧数据梯度，希望对新数据梯度更新的方向进行约束



- GEM & A-GEM: GEM就是满足一个模型在新旧数据上的梯度方向成锐角或直角这个公式就好，缺陷就是每一个训练完的task都要重新计算一遍对于旧任务数据的梯度，不高效。需要保存以前所有任务的gradient。

  $$\langle g, g_k \rangle := \left\langle \frac{\partial \ell(f_\theta(x,t),y)}{\partial \theta}, \frac{\partial \ell(f_\theta,\mathcal{M}_k)}{\partial \theta} \right\rangle \geq 0, \text{ for all } k < t$$

  A-GEM加速了优化过程，不用再和以前每一个任务的梯度一一作比较，只需要对旧任务的sample进行任意选择保存，然后在当前task上对这些sample求梯度，拿这个梯度当成旧任务梯度就行。

2. Model-Centric

- **Dynamic network**(表现力最强)

  主要思想就是对网络的结构进行拓展，按照新加进来的任务，添加一些新的结构或backbone。

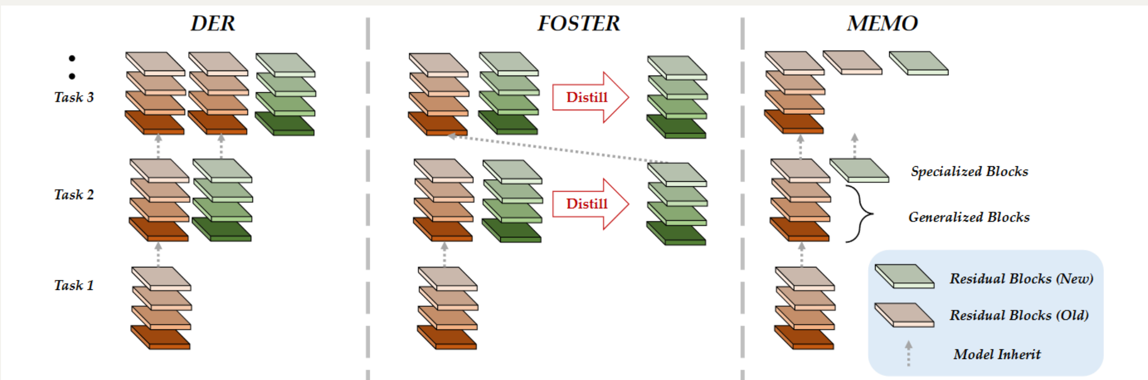


Figure 4: Illustration of network structure evolving in backbone expansion. **Left:** DER [20] expands a new backbone per incremental task. **Middle:** FOSTER [21] adds an extra model compression stage, which maintains limited model storage. **Right:** MEMO [22] decouples the network structure and only expands specialized blocks.

Table 2: Average and last accuracy performance comparison on CIFAR100. '#P' represents the number of parameters (million). The full table is reported in the supplementary.

Method	Base0 Inc5			Base0 Inc10		
	#P	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	#P	$\bar{\mathcal{A}}$	$\mathcal{A}_B$
Finetune	0.46	17.59	4.83	0.46	26.25	9.09
EWC	0.46	18.42	5.58	0.46	29.73	12.44
LwF	0.46	30.93	12.60	0.46	43.56	23.25
GEM	0.46	31.73	19.48	0.46	40.18	23.03
Replay	0.46	58.20	38.69	0.46	59.31	41.01
RMM	0.46	65.72	51.10	0.46	68.54	56.64
iCaRL	0.46	63.51	45.12	0.46	64.42	49.52
PODNet	0.46	47.88	27.99	0.46	55.22	36.78
Coil	0.46	57.68	34.33	0.46	60.27	39.85
WA	0.46	64.65	48.46	0.46	67.09	52.30
BiC	0.46	62.38	43.08	0.46	65.08	50.79
FOSTER	0.46	63.38	49.42	0.46	66.49	53.21
DER	9.27	67.99	53.95	4.60	69.74	**58.59**
MEMO	7.14	**68.10**	**54.23**	3.62	**70.20**	58.49
DyTox	10.7	68.06	52.23	10.7	71.07	58.72
L2P	85.7	84.00	78.96	85.7	89.35	83.39

- DER: 每添加一个新的任务，就增加一个backbone（一般是ResNet18）
- FOSTER： 训练过程中，内存最多只维持2个backbone，一个是先前任务总的backbone（保存旧知识），一个是新任务的backbone（学习的是更具判别性的新类知识），然后再把两个backbone的知识融合在一个backbone里面，做一个model compression。
- MEMO：保持模型浅层的特征知识共享，末层随着类别的增加而增加（末层更具有判别性和多样性）。
- TCIL（CIFAR100 SOTA）

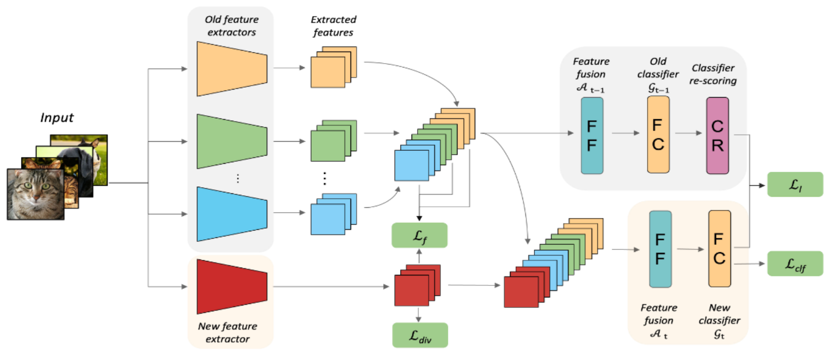
  

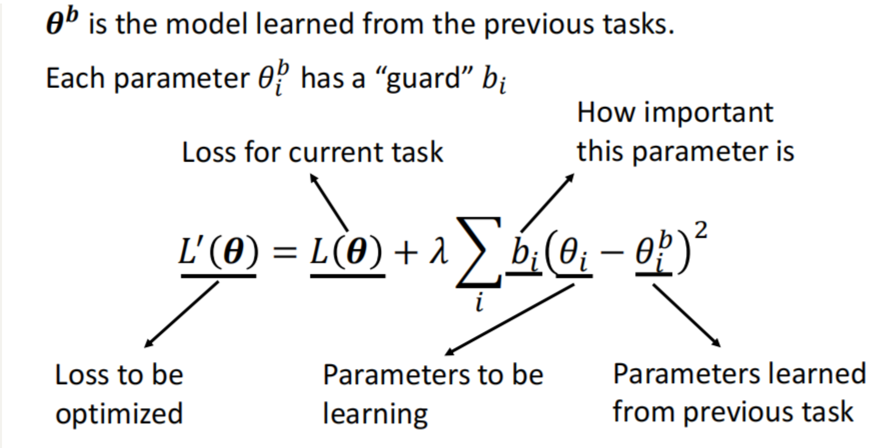
  Figure 2: An illustrative framework of the proposed TCIL. It uses a dedicated feature extraction sub-network for each specific task. $\mathcal{L}_f$ and $\mathcal{L}_l$ are knowledge distillation loss at feature and logit levels, respectively. $\mathcal{L}_{clf}$ is the classification loss, and $\mathcal{L}_{div}$ is the divergence loss for guiding the training of the feature extractors.

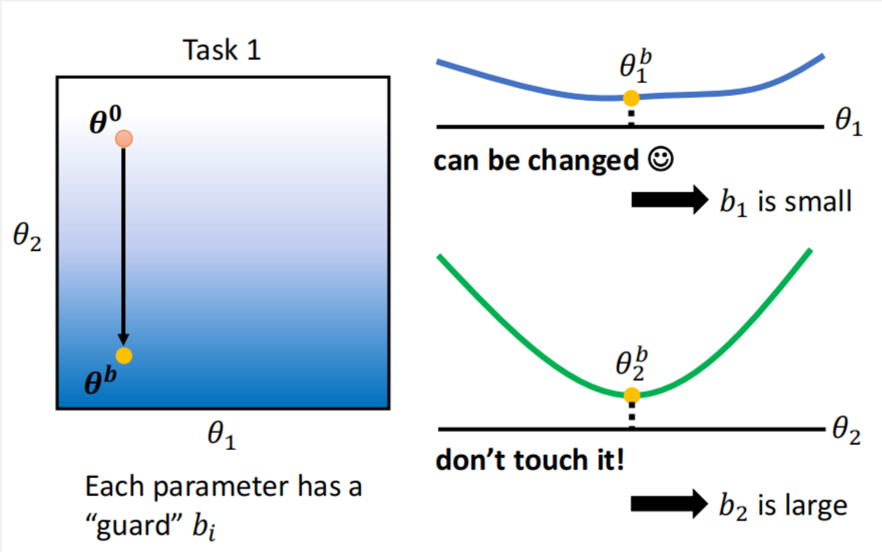
  延续了DER的思想，每个新Task增加一个feature extractor。用了特征蒸馏和logits蒸馏，对于增加得过大的模型，提出剪枝方式，为TCIL-lite。

- **Parameter regularization**

  根据模型参数对任务的贡献度，对参数更新的方式进行了限制。但是这类方式的表现能力已经远不如动态网络的方式了。

  - EWC: 通过计算fisher信息矩阵，获取每个参数对先前任务的重要程度，对于贡献度大的参数来说，对其更新所施加的惩罚也更大。





3. Algorithm-Centric

（和上面两种方式的界限不是特别的明显，甚至可以和model-centric融合）

- LwF （知识蒸馏，自蒸馏，旧任务蒸馏到新任务）
- BiC（矫正对新任务的偏好，用验证集（从训练集中分出来的）去矫正输出的bias，和预训练方式中的SSF有点像）

4. Discussion for training-from-scratch method

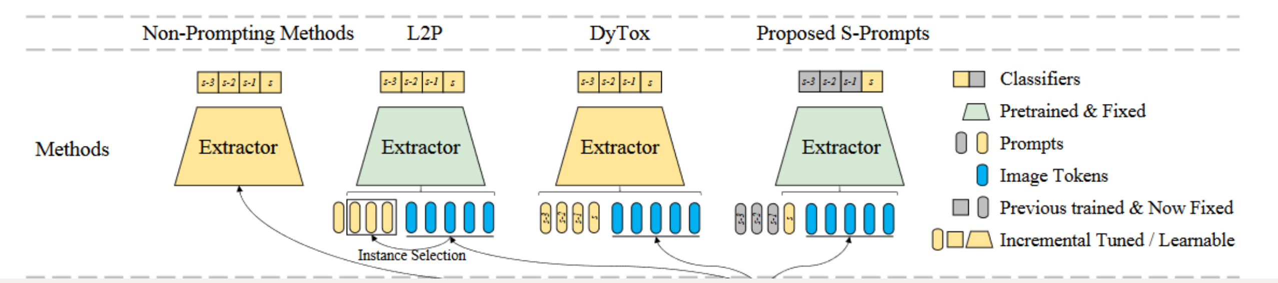
- 数据集：用的最多的都是CIFAR100、ImageNet100、ImageNet1000这种，算是比较小的，因为是train from scratch的。
- Dynamic Network的方法还是占据着表现力主导地位的
- 用预训练的方式来和这些方法对比实际上是不公平的（除非它们的backbone也使用预训练的方式），原因：
  - 这些模型是从头开始训练，挑战难度会更大
  - 预训练的数据集与实际训练和测试的数据集可能会有大量重合，导致其性能远高于这种scratch的模型。
  - 当给足够的空间资源时，动态网络是能够表现出很好的作用。但当资源限制时，就不一定能表现出最好的能力。因此需要做一个memory-agnostic

的对比。文中提出的AUC-A和AUC-L是很好地选择。

💡 Pre-trained

1. Prompt-based

和rehearsal-based的方式比较，prompt-based就是不需要exampler去记录过去的类别，而是用一些特定的prompt来代表每一个任务。主要思想是，prompt由几个token组成，是可调的，但是image token是不可调的。预训练模型大多数固定，有时候也可略微调整（DyTox）



- DyTox: 第一篇在continual learning中用ViT的文章（CVPR2022）每个task，制定一个独立的token和专属分类器（加起来的参数和整个预训练模型相比是很少的）。推理的时候，每个任务的token都将加到image token上（相当于原来的class token），然后送到token专属分类器里面进行分类。
- L2P：有一个prompt pool，里面是key-value paired，并且是固定数量的。input通过预训练模型提取得到query，然后匹配从prompt pool里面提取几个Token作为可训练的prompt，直接加入到image embedding前面（当作class token）进行训练。最后根据这个prompt对应的token，做平均得到预测类别。文章默认了这个prompt池里面的prompt既可以学习到共享知识，也能学到独立的知识。但文章还是用了一些rehearsal的方式，来和有buffer size的方法进行比较。文章没有说具体回放形式，应该是随机采样。

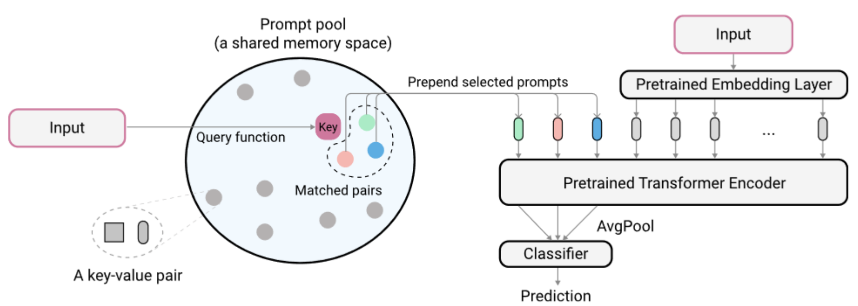
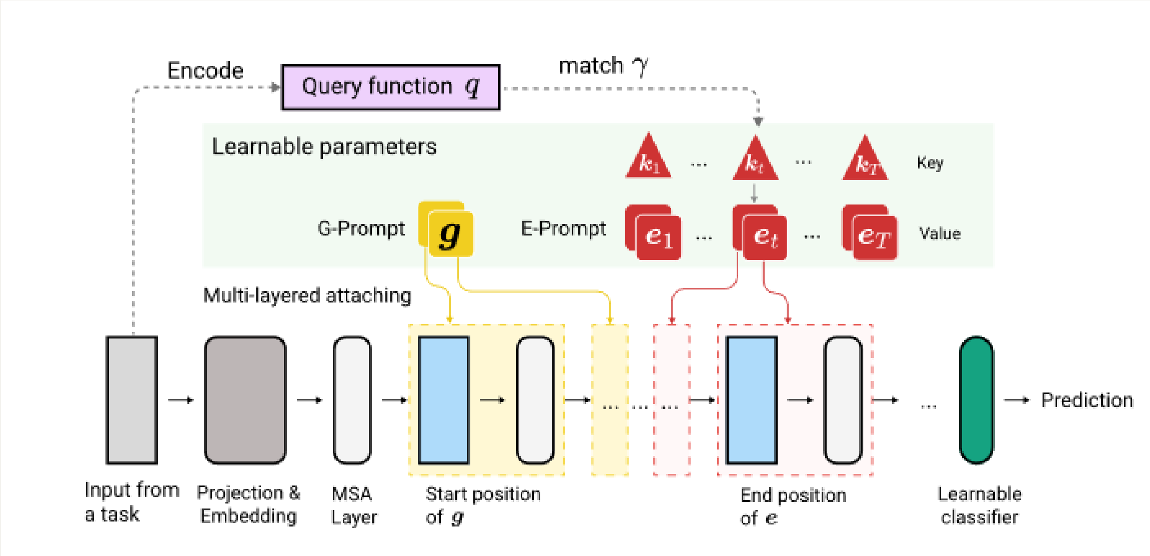


Figure 2. Illustration of L2P at test time. We follow the same procedure at training time: First, L2P selects a subset of prompts from a key-value paired *prompt pool* based on our proposed instance-wise query mechanism. Then, L2P prepends the selected prompts to the input tokens. Finally, L2P feeds the extended tokens to the model, and optimize the prompt pool through the loss defined in equation 5. The objective is learning to select and update prompts to instruct the prediction of the pre-trained backbone model.

- Dual-Prompt：强调是**rehearsal-free**的方式，并且把L2P里面的prompt pool进行了解耦。G-Prompt用来学习底层的，共享的特征，E-Prompt来处理比较高层的，具有判别性的特征。而且这些prompt插入的层不一样，G-Prompt主要插入在前两层，而E-Prompt插入在3-5层表现最好。G-Prompt长度5最好，E-Prompt长度20最好。



Method	Buffer size	Split CIFAR-100		Buffer size	Split ImageNet-R	
		Avg. Acc (↑)	Forgetting (↓)		Avg. Acc (↑)	Forgetting (↓)
ER [8]	1000	67.87±0.57	33.33±1.28	1000	55.13±1.29	35.38±0.52
BiC [62]	1000	66.11±1.76	35.24±1.64	1000	52.14±1.08	36.70±1.05
GDumb [47]	1000	67.14±0.37	-	1000	38.32±0.55	-
DER++ [4]	1000	61.06±0.87	39.87±0.99	1000	55.47±1.31	34.64±1.50
Co^{2}L [5]	1000	72.15±1.32	28.55±1.56	1000	53.45±1.55	37.30±1.81
ER [8]	5000	82.53±0.17	16.46±0.25	5000	65.18±0.40	23.31±0.89
BiC [62]	5000	81.42±0.85	17.31±1.02	5000	64.63±1.27	22.25±1.73
GDumb [47]	5000	81.67±0.02	-	5000	65.90±0.28	-
DER++ [4]	5000	83.94±0.34	14.55±0.73	5000	66.73±0.87	20.67±1.24
Co^{2}L [5]	5000	82.49±0.89	17.48±1.80	5000	65.90±0.14	23.36±0.71
FT-seq	0	33.61±0.85	86.87±0.20	0	28.87±1.36	63.80±1.50
EWC [20]	0	47.01±0.29	33.27±1.17	0	35.00±0.43	56.16±0.88
LwF [28]	0	60.69±0.63	27.77±2.17	0	38.54±1.23	52.37±0.64
L2P [60]	0	83.86±0.28	7.35±0.38	0	61.57±0.66	9.73±0.47
DualPrompt	0	**86.51**±0.33	**5.16**±0.09	0	**68.13**±0.49	**4.68**±0.20
Upper-bound	-	90.85±0.12	-	-	79.13±0.18	-

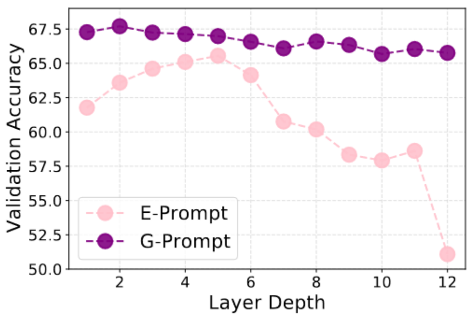


Fig. 3: Effects of position to attach prompts on Split ImageNet-R validation set. We empirically observe that attaching G- and E-Prompts to the 2nd and 5th MSA layer results in the best performance.

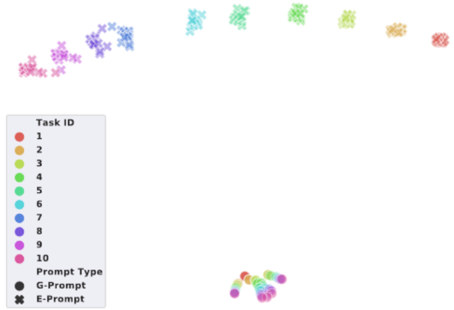
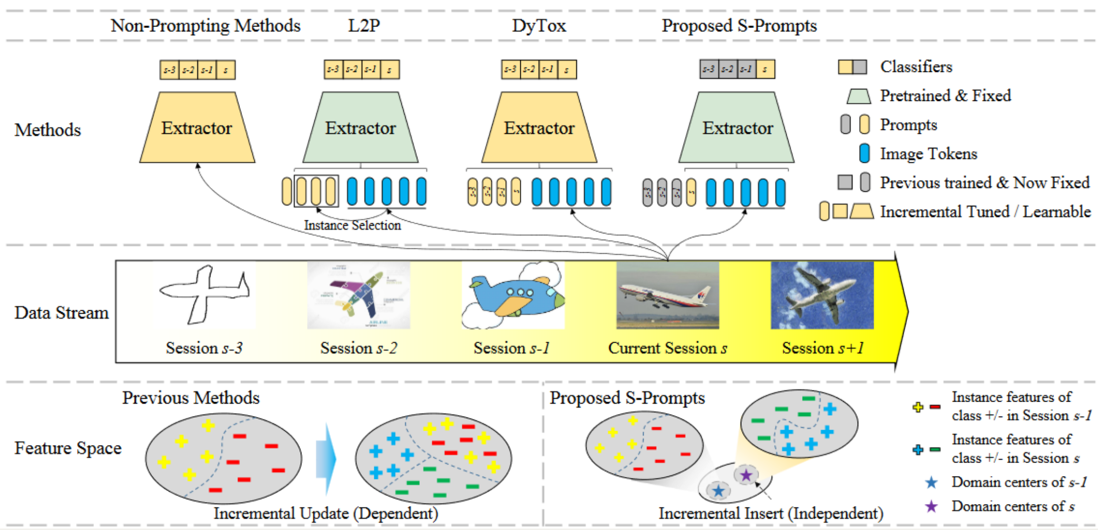
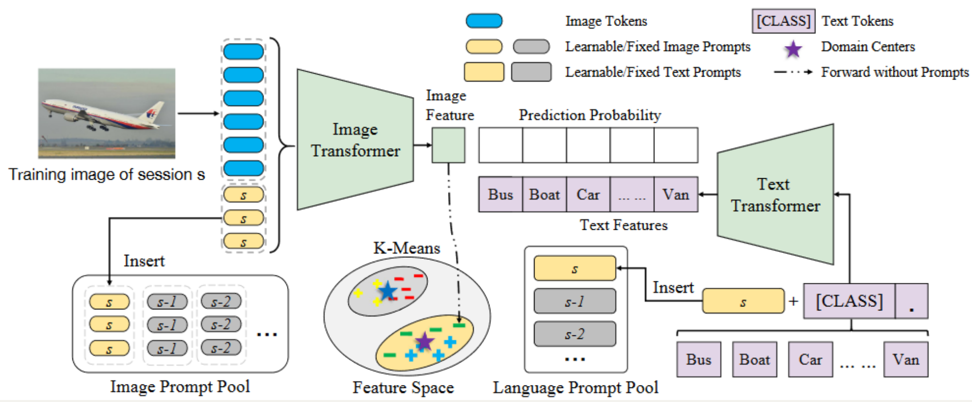


Fig. 4: t-SNE visualization of G- and E-prompts. Each point represents a prompt vector of dimension 768. E-Prompts are taken from the final model, while G-Prompts are taken from model snapshots after trained on each task.

- S-Prompt：和DyTox有点像，但这个是每一个域对应于一个（或几个）image token和一个tunable text prefix token，仅仅对域token和末端分类器进行微调。推理的时候用KNN选取距离图像特征最近的域特征中心（K-Means选出的）对应的Prompt。然后像CLIP那样进行相似度匹配。





2. Parameter-efficient tuning

- ADAM (Adapt and merge)　好像投稿没中，可能创新性还不足？

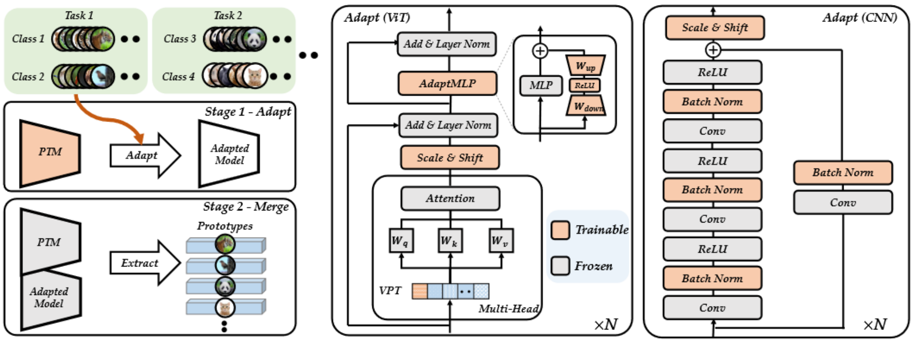


Figure 3. Illustration of ADAM. **Left**: the training protocol of ADAM. We adapt the PTM using the first stage training set $\mathcal{D}^1$ and then concatenate the embedding functions of PTM and the adapted model to maintain *generalizability* and *adaptivity*. The aggregated embedding function $[\phi^*(\cdot), \phi(\cdot)]$ is frozen throughout the following stages, and we extract the prototypes via Eq. 6 to set the classifier. **Middle**: adapting pre-trained ViT for CIL. We provide VPT Deep/Shallow, Scale & Shift, and Adapter for model adaptation. **Right**: adapting pre-trained CNN for CIL. We provide BN tuning and Scale & Shift for model adaptation. ADAM is a general framework that can be orthogonally combined with these adapting techniques. Red modules in the figure are trainable, while gray ones are frozen.

Method	CIFAR B0 Inc5		CUB B0 Inc10		IN-R B0 Inc5		IN-A B0 Inc10		ObjNet B0 Inc10		OmniBench B0 Inc30		VTAB B0 Inc10	
	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	$\bar{\mathcal{A}}$	$\mathcal{A}_B$	$\bar{\mathcal{A}}$	$\mathcal{A}_B$
Finetune	38.90	20.17	26.08	13.96	21.61	10.79	21.60	10.96	19.14	8.73	23.61	10.57	34.95	21.25
Finetune Adapter [10]	60.51	49.32	66.84	52.99	47.59	40.28	43.05	37.66	50.22	35.95	62.32	50.53	48.91	45.12
LwF [38]	46.29	41.07	48.97	32.03	39.93	26.47	35.39	23.83	33.01	20.65	47.14	33.95	40.48	27.54
L2P [72]	85.94	79.93	67.05	56.25	66.53	59.22	47.16	38.48	63.78	52.19	73.36	64.69	77.11	77.10
DualPrompt [71]	87.87	81.15	77.47	66.54	63.31	55.22	52.56	42.68	59.27	49.33	73.92	65.52	83.36	81.23
SimpleCIL	87.57	81.26	92.20	**86.73**	62.58	54.55	60.50	49.44	65.45	53.59	79.34	73.15	85.99	84.38
ADAM w/ Finetune	87.67	81.27	91.82	86.39	70.51	62.42	61.57	50.76	61.41	48.34	73.02	65.03	**87.47**	80.44
ADAM w/ VPT-Shallow	90.43	84.57	92.02	86.51	66.63	58.32	57.72	46.15	64.54	52.53	79.63	73.68	87.15	**85.36**
ADAM w/ VPT-Deep	88.46	82.17	91.02	84.99	68.79	60.48	60.59	48.72	67.83	54.65	**81.05**	**74.47**	86.59	83.06
ADAM w/ SSF	87.78	81.98	91.72	86.13	68.94	60.60	**62.81**	**51.48**	**69.15**	**56.64**	80.53	74.00	85.66	81.92
ADAM w/ Adapter	**90.65**	**85.15**	**92.21**	**86.73**	**72.35**	**64.33**	60.53	49.57	67.18	55.24	80.75	74.37	85.95	84.35

- Model adaptation: 用PET的方式，获得一个adapted model，同时保持原来的pretrained model。PET方式包括VPT、Scale&Shift（SSF）、Adapter还有Batch Normalization Tuning。
- Merge with pre-trained：直接出来一个预训练的特征，然后和上面adapted model的adaptive特征concat在一起。

- LAE (ICCV2023)

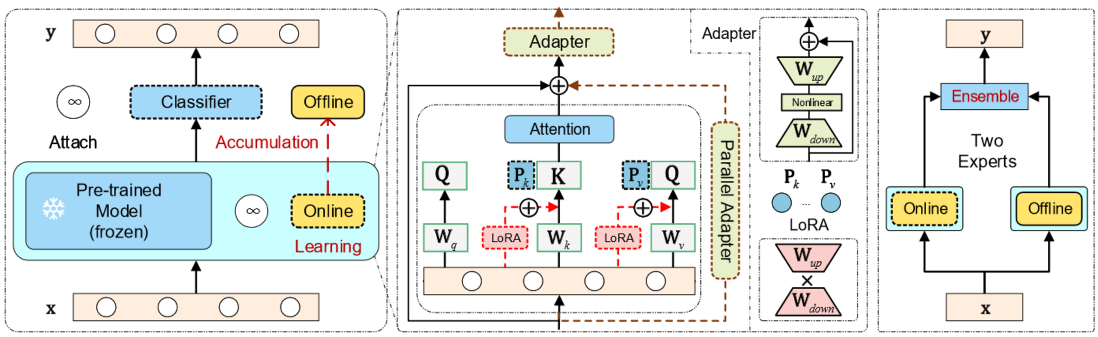


Figure 2: **Illustration of our LAE framework.** The left (a) is the training process, the right (c) is the inference flow, and the middle (b) lists some representative Parameter-Efficient Tuning (PET) modules attached to a transformer attention block, modules connected with dashed lines are optional and we use one of the PET modules in the experiments. There is no residual connection in Parallel Adapter. “Online” is a PET module to learn knowledge from the new task and “Offline” is a PET module to accumulate knowledge. The pre-trained model is omitted in the inference flow for concise.

用了Adapter、LoRA和Prefix tuning三种PET方式，还有3种创新设计：

- 校准不同PET模块（Adapter、LoRA、PF）的更新速度，让它们的适配速度与Adapter的保持一致
- 多任务知识聚合：也就是用online-PET模块，使用指数移动平均的方式更新offline-PET的参数
- 预测的时候，集成online-PET和offline-PET的energy，使用的是max函数。

3. Other

- **ELLE**: model expansion的思想，是NLP领域的，针对的是Transformer的结构调整

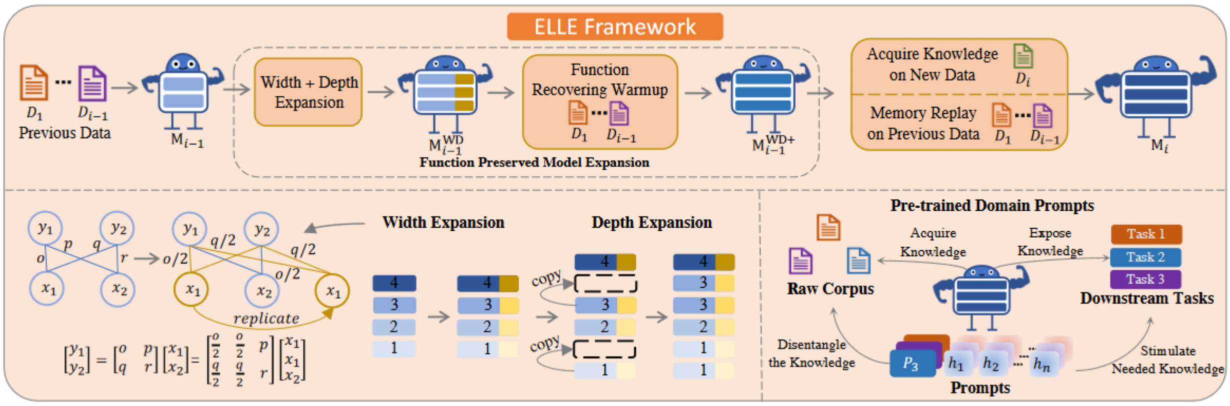


Figure 1: Illustration of ELLE when adapting an existing PLM $\mathcal{M}_{i-1}$ trained on previous data $\overline{\mathcal{D}}_{i-1}$ to a new corpus $\mathcal{D}_i$. We also visualize the mechanism of width / depth expansion and pre-trained domain prompts.

模型的宽度和深度都扩展了，通过了一个function recover的方式，来在扩张的模型上保持原有模型的功能。深度扩充就是把中间的一些层复制多一份，宽度扩充就是把中间的weight矩阵加宽。而且还添加了预训练的domain prompt，来告诉它处于哪个领域。

4. Discussion for pre-trained-method

- **combine with Neuroscience knowledge**：在introduction部分，有很多的from scratch文章是结合生物学思想的，比如说突触、海马体.... pre-trained文章里面我看好像只有dual prompt那篇文章用了这个想法，其它的都是直接从模型方面缺陷入手改进的。
- **transformer**的**decoder**是否可以用来做**generative**的**replay?** ：用generative和replay的那种思想，因为当前的pre-trained的方式全都是model-centric，较少有data-centric的
- **more challenging benchmark**: 在更具挑战性的**benchmark**上进行实验，而非那种from scratch的小实验。ADAM那篇文章里面提出了4个更具挑战性的benchmark，但在这些数据集上跑的方法比较少。