Multiple Tasks and Multiple Agents

Ph.D. course on Collective Machine Intelligence

Antonio Carta
University of Pisa
Antonio.carta@unipi.it

Outline

- Multi-Task Learning
- Task-Incremental Learning
- Federated Learning
- Continual Federated Learning
- Multiple CL Agents

Methods:

- Task-aware Architectural Methods
- Knowledge Distillation

Multi-Task Learning

Joint training on multiple tasks

What is a Task?

- task: $\mathcal{T}_i \triangleq \{p_i(\mathbf{x}), p_i(\mathbf{y} \mid \mathbf{x}), \mathcal{L}_i\}$,
  - True data-generating distribution $p_i(x, y)$
  - loss $\mathcal{L}_i \leftarrow$ this is the evaluation loss function, not necessarily the training loss function
- usually, we have to some samples $\mathcal{D}_\mathrm{i} = \{(\mathbf{x}, \mathbf{y})_k \sim p_i(x, y)\}$
- Examples:
  - Different data: objects, classes, backgrounds, objectives, …
  - Different problems: classification, detection, segmentation, …

Problem Statement

MTL Objective:

$$\min_{\theta^{sh},\theta^1,\ldots,\theta^T} \sum_{i=1}^{T} \mathcal{L}_i\left(\{\theta^{sh},\theta^i\},\mathcal{D}_i\right)$$

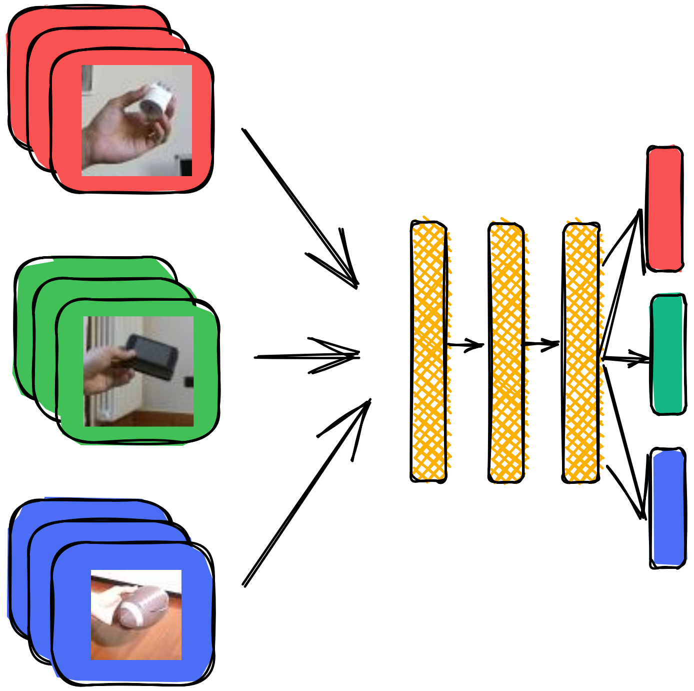
▨ Shared Layers used by multiple tasks

🟧 Task-specific layers exclusive to single tasks

- solve all the tasks concurrently
- share knowledge between tasks (θ^{sh})
- Separate task-specific components when necessary (θ^i)
- exploit tasks relationships to converge faster and generalize better

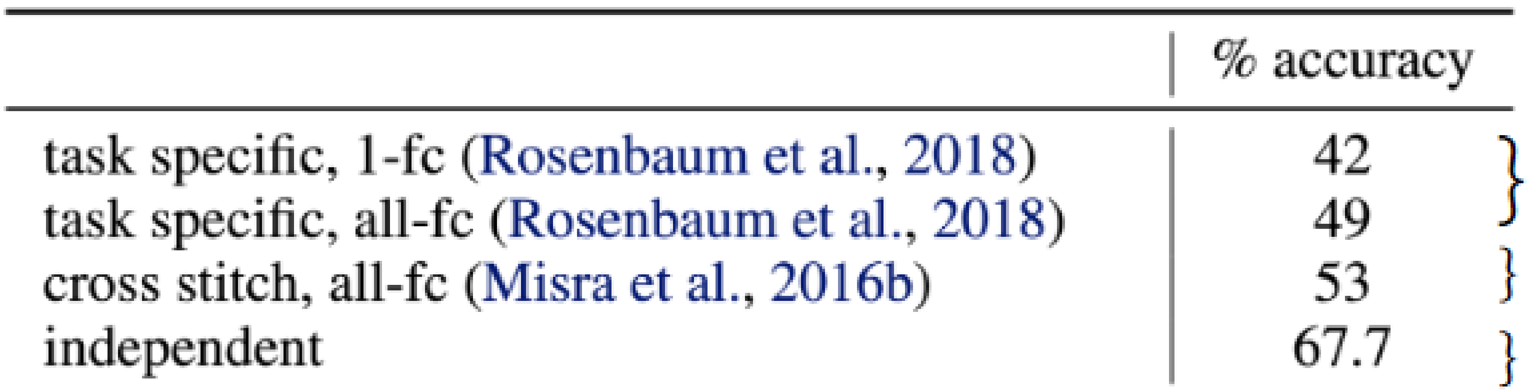
Critical Assumption:

- tasks share some common structure
  - helps learning multiple tasks jointly
  - it may also cause interference!



REMINDER: Transfer and Interference

- **Positive Transfer**: training tasks jointly (i.e. sharing weights) improves the performance on the single tasks
  - if the tasks are small the joint solution is more robust and less prone to overfitting
- **Negative Transfer**:
  - Sometimes independent models are better
  - cross-task interference, different rates of learning
  - representational capacity, MT nets need to be bigger



	% accuracy
task specific, 1-fc (Rosenbaum et al., 2018)	42
task specific, all-fc (Rosenbaum et al., 2018)	49
cross stitch, all-fc (Misra et al., 2016b)	53
independent	67.7

} multi-head architectures

} cross-stitch architecture

} independent training

Yu et al. Gradient Surgery for Multi-Task Learning. 2020

Naive MTL Optimization

Naive MT-SGD

Until convergence:

- sample tasks
- sample examples for each task
- SGD step: forward → backward → descent step (for all samples)

- NOTE: we implicitly balance over tasks instead of over samples
- NOTE: losses may have different magnitudes (e.g. in regression problems)

MTL Objective

- weighted objective $\min_\theta \sum_{i=1}^{T} w_i \, \mathcal{L}_i(\theta, \mathcal{D}_i)$
- how to choose the weights?
  - a predefined relative importance
  - balancing amount of data
  - heuristics
    - gradient of similar magnitudes (Chen et al. GradNorm. ICML 2018)
    - optimize for the worst task

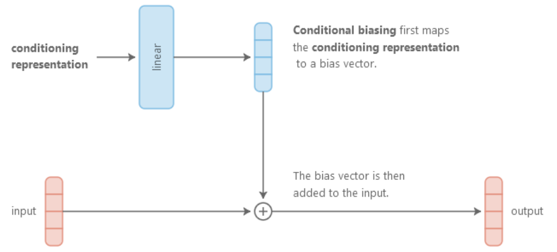
Weight Sharing - Task Conditioning

Task-specific and task-agnostic parameters control transfer

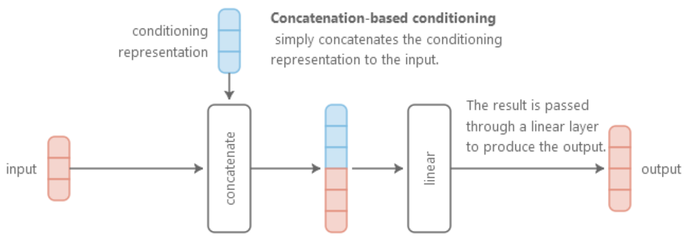
$$\min_{\theta^{sh},\theta^1,\dots,\theta^T} \sum_{i=1}^{T} \mathcal{L}_i\left(\{\theta^{sh},\theta^i\},\mathcal{D}_i\right)$$

- shared layers with simple task conditioning such as sum, concatenation, multiplication with task embedding z or gating with task label z
- Complex and adaptive forms of task conditioning are possible

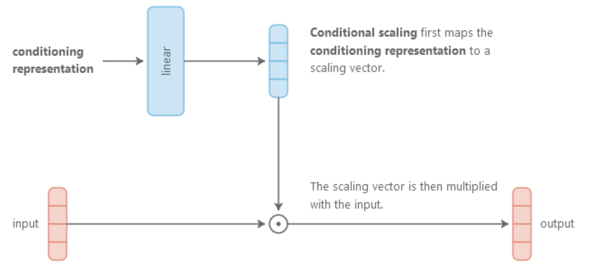
Sum conditioning

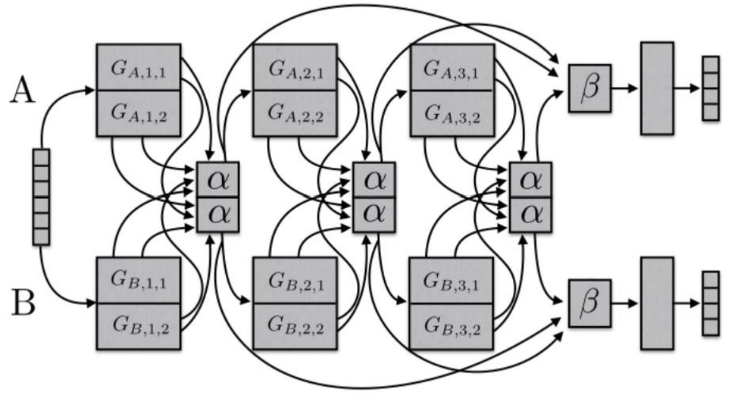

conditioning representation
linear
Conditional biasing first maps the conditioning representation to a bias vector.
The bias vector is then added to the input.
input
output

cat conditioning


conditioning representation
Concatenation-based conditioning simply concatenates the conditioning representation to the input.
input
concatenate
linear
The result is passed through a linear layer to produce the output.
output

Multiplicative conditioning


conditioning representation
linear
Conditional scaling first maps the conditioning representation to a scaling vector.
The scaling vector is then multiplied with the input.
input
output



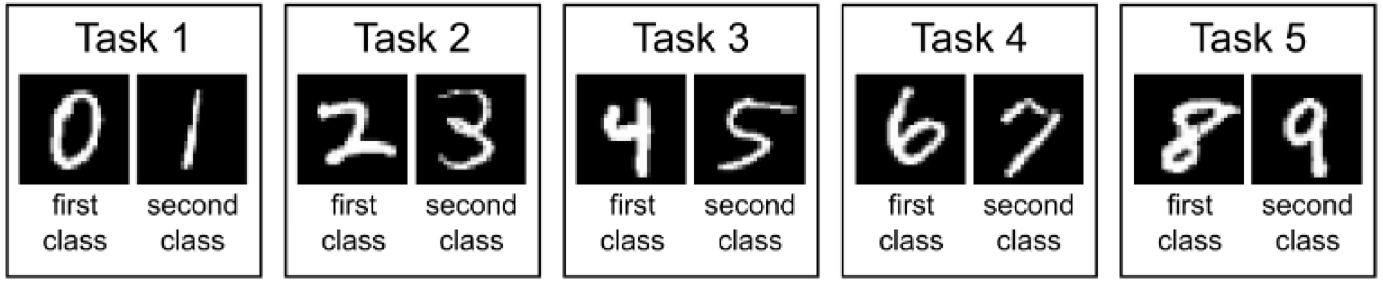
S, Ruder et a.l "Latent multi-task architecture learning". AAAI 2019
https://distill.pub/2018/feature-wise-transformations/

Task-Incremental Learning

Continual Learning with Architectural Methods

Task-Incremental Learning

- We want to solve task-incremental learning: learning multiple tasks incrementally
- We can exploit task labels to design task-aware model architectures
- **GOAL**: knowledge transfer with minimal interference



Modular Architectures

Idea:

- split the networks into several modules
- Connect modules to enable transfer
- Freeze/mask module to limit forgetting

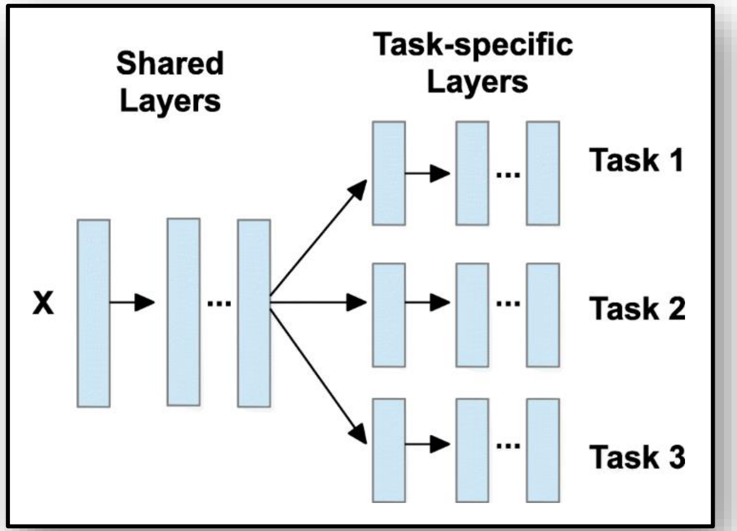
Opportunities

- Explicit separation between task-specific and shared components
- Eliminate forgetting (with freezing / task-specific components)

Challenges

- Limiting memory growth
- Requirements of task labels
- Forward transfer is impacted by some solutions (freezing / task-specific components)

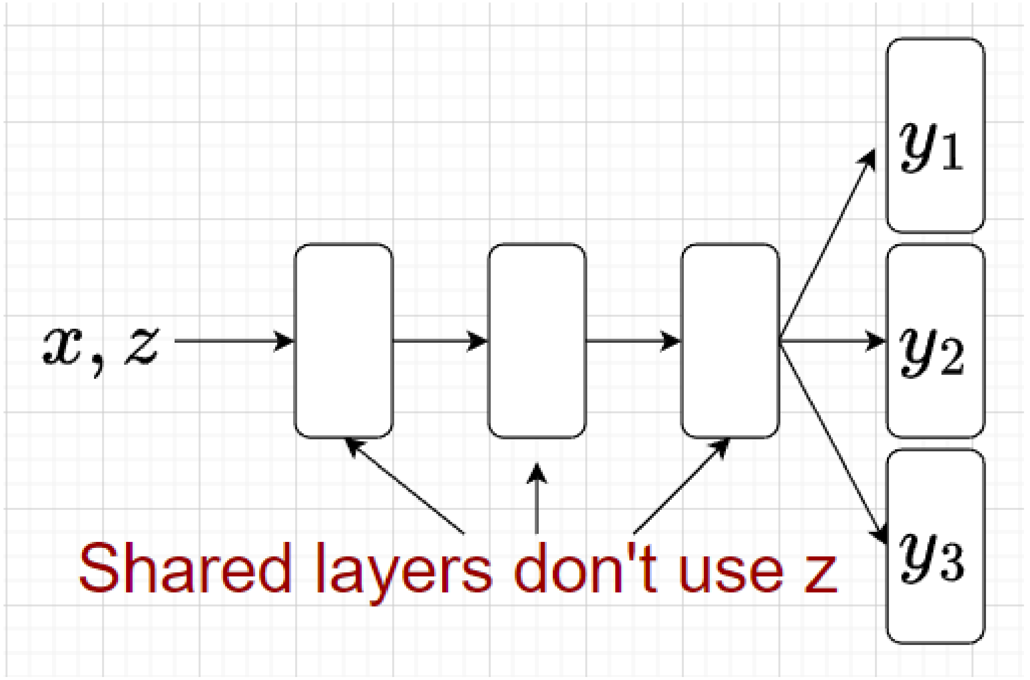
Conflicting requirements: a good method needs to balance memory occupation, eliminate forgetting, promoting forward transfer.



Base Model: Multi-Head

Multi-Head architectures have:

- a shared feature extractor
- a separate linear classifier (head) for each task
- the correct head is selected for each example via multiplicative gating
- The multi-head architecture is one of the big advantages of having task labels.
- We can also have task-dependent hidden layers (architectural methods)



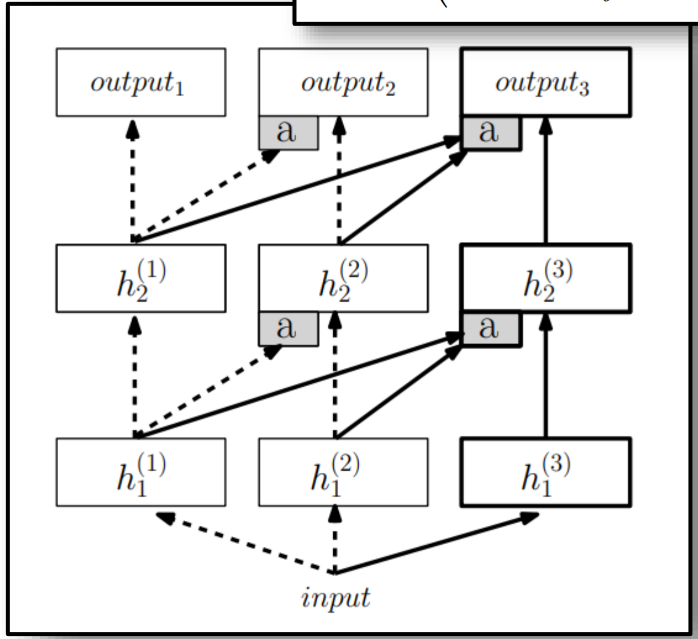
Now we also want a task-aware feature extractor!

Progressive Neural Networks (PNN)

A Basic Modular Architecture

- **Column**: Each new task adds its own "column" of features to each layer
- **Adapter**: New columns are connected to all the previous one via adapter
- **Inference**: task labels are used to activate the correct columns

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)}\right)$$



PNN – Column

Column: Each new task adds its own "column" of features to each layer

- Each column is connected to all the previous ones
- After training the column is **frozen**
- **Inference**: use task labels to activate the correct columns

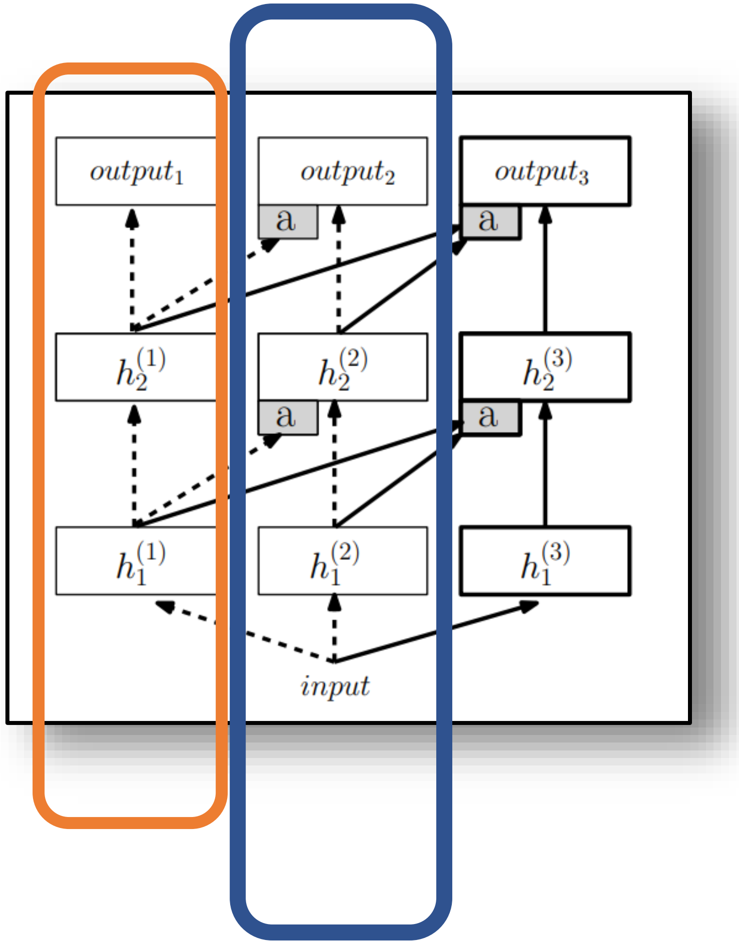
PNN Column

$$h_i^{(k)} = f\left(W_i^{(k)} h_{i-1}^{(k)} + \boxed{\sum_{j<k}} U_i^{(k:j)} h_{i-1}^{(j)}\right)$$

Connections to previous columns

PNN Column

previous columns



Pros and Cons

- **Good forward transfer**: each task can re-use previous columns
- **Inhibits forgetting** by freezing columns
- **Poor scaling in memory size**: quadratic due to adapters
- Requires **task labels**

Two open problems:

- How do we limit the **memory growth**?
- How do we choose which column to activate if don't have **task labels**?

Memory Growth: PNN Columns can be compressed

Good news: most of the capacity is not used!

- We can reduce the size of new columns over time
- We can compress them (e.g. after training)

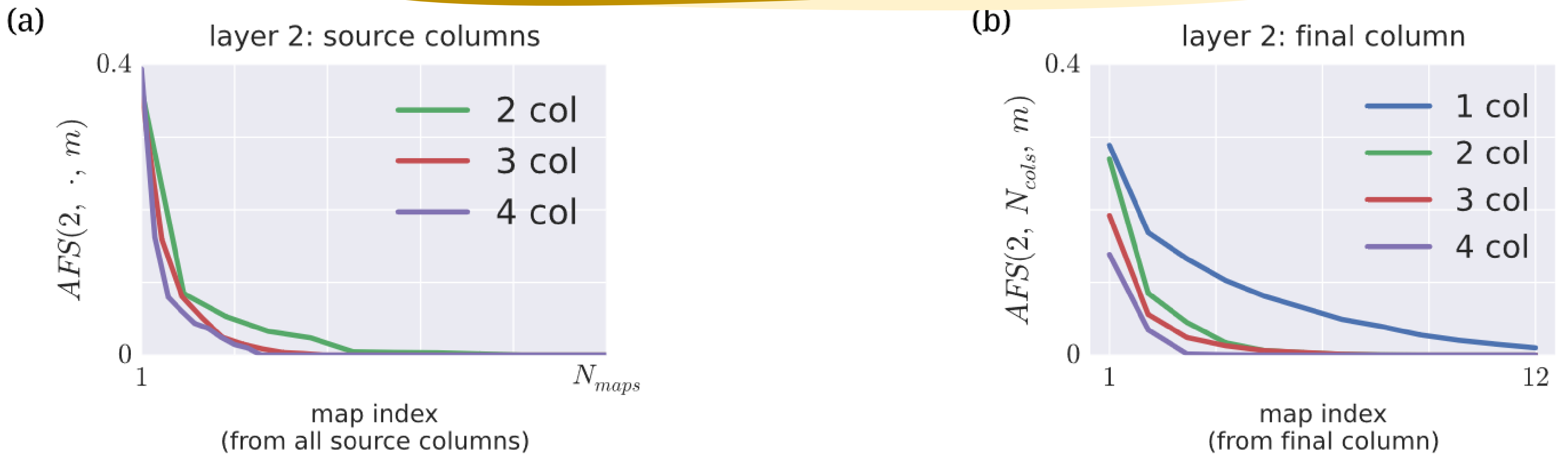


Figure 10: (a) Spectra of AFS values (for layer 2) across all feature maps from source columns, for the Atari dataset. The spectra show the range of AFS values, and are averaged across networks. While the 2 column / 3 column / 4 column nets all have different values of N_{maps} (here, 12, 24, and 36 respectively), these have been dilated to fit the same axis to show the proportional use of these maps. (b) Spectra of AFS values (for layer 2) for the feature maps from only the final column.

Task Inference

- **Modular architecture + task inference** to remove need for task labels
- **Task Inference**: classifier that given an input predicts the task label
- Often predicting the task label is easier than predicting the class.
  - Example: identifying a language (task inference) is easier than predicting the next word of an incomplete sentence (solving the task).
  - We can use a proxy signal: reconstruction error, pattern of activations, …
  - We can use a simple classifier, easier to train continually

Expert Gate

IDEA: *Modular network with gating and task inference*

- **Expert**: a module of the network trained on a single task
- **Gate**: an undercomplete autoencoder for each task
- **Inference**: use the expert model associated with the most confident autoencoder

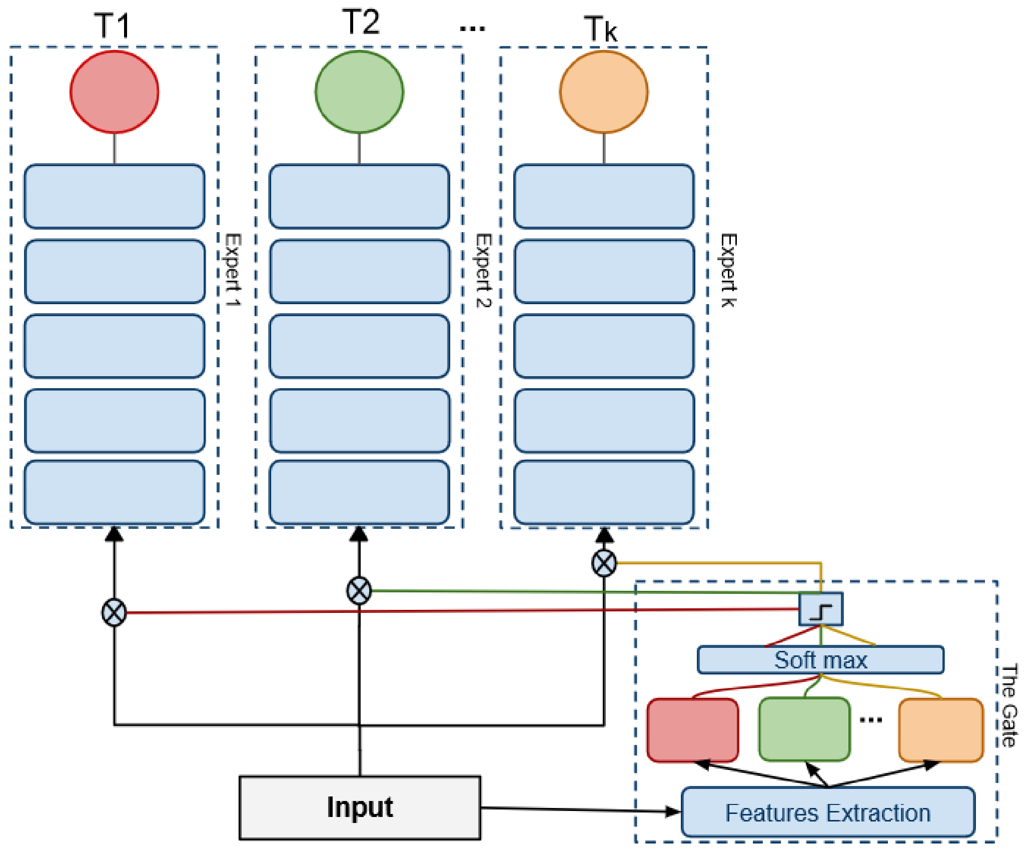


Figure 1. The architecture of our Expert Gate system.

Expert Gate: Gate

- **Pretrained Input Features**: input x to the expert and autoencoder is the output of the last CONV layer of AlexNet pretrained on ImageNet
- **Gate Architecture**: standardization + an undercomplete autoencoder for each task
- **Inference**: use the expert model associated with the most confident autoencoder
  - $\boldsymbol{er_i}$ reconstruction error for task i
  - $\boldsymbol{t}$ temperature
- **Limitations**:
  - Requires pretrained network
  - The reconstruction error is not always a good task predictor. Autoencoders are very good at reconstructing unseen data.

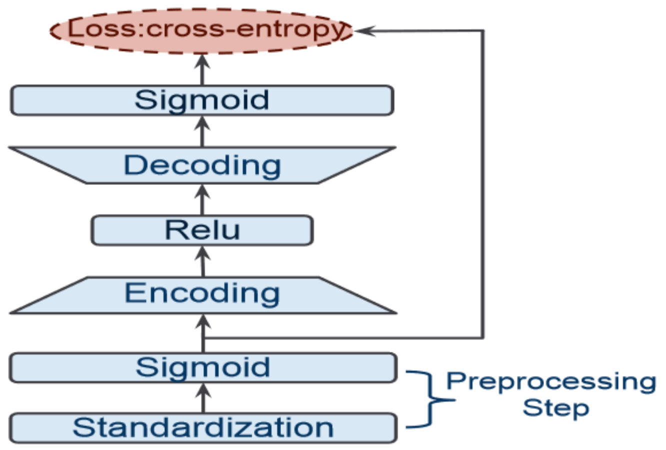


Figure 2. Our autoencoder gate structure.

$$p_i = \frac{\exp(-er_i/t)}{\sum_j \exp(-er_j/t)}$$

Masking – Motivations

increase the memory occupation over time

- We know that deep networks are overparameterized
- **SOLUTION**: use a fixed large network and select a subset of units for each task
- **ADVANTAGES**:
  - Similar to modular networks but less expensive
  - Binary masks are easy to compress
  - Induces sparsity

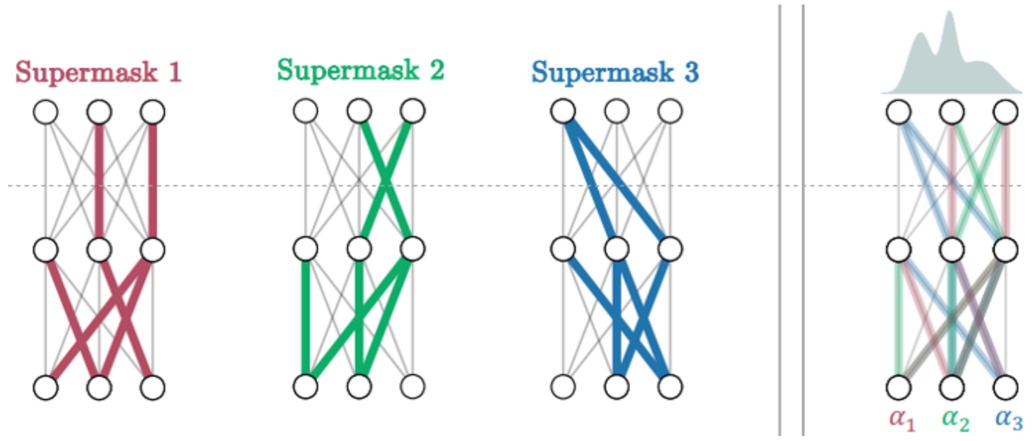
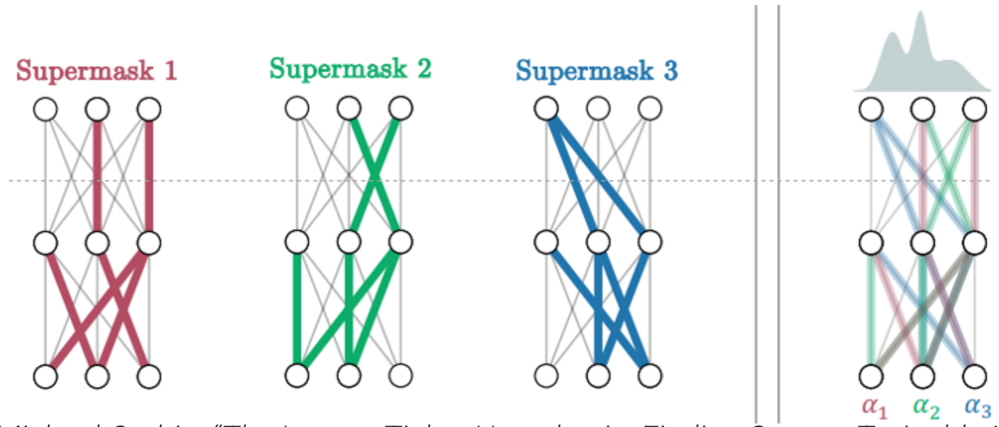


Image from Supermasks in Superposition, Wortsman et al. 2020.

Motivations – Lottery Ticket Hypothesis

HYPOTHESIS - Lottery Ticket Hypothesis:
dense, randomly-initialized, feed-forward networks contain subnetworks (**winning tickets**) that—when **trained in isolation**−reach test accuracy comparable to the original network in a similar number of iterations

WARNING: This is just a hypothesis, not a formal theorem
PROBLEMS:
- How do we optimize binary masks during continual learning?
- How do we do task inference?



Frankle, Jonathan, and Michael Carbin. "The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks," ICLR 2019 22

Masking with Pruning Methods

We can use pruning methods to find a mask

Magnitude Pruning

- Train a network
- Sort the weights in a layer by their absolute magnitude
- Cut the lowest p%

Variation: **Iterative Magnitude Pruning (IMP)**, where the process is repeated multiple times, each time pruning p% and retraining.

PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning, Mallya et al 2017.

PackNet

Magnitude Pruning, task-aware

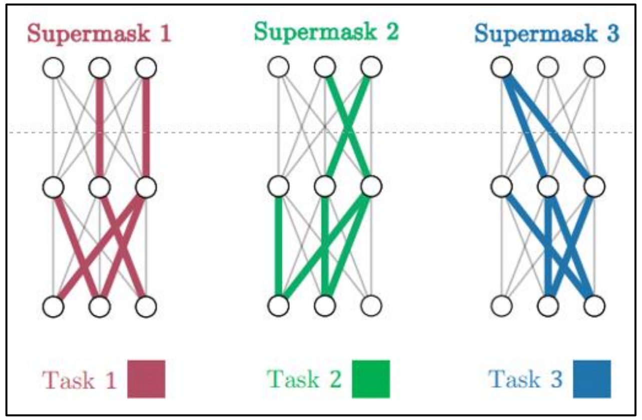
- **Model**: masked layers $\boxed{\mathbf{y} = (\mathbf{W} \odot \mathbf{m})\mathbf{x}}$
- **Inference**: use task labels to choose mask.
- **Training**:
  - start from a **Pretrained Model.**
  - for each task:
    - **Finetune**: the weights of the dense network (unmasked) on the new task
      - frozen parameters are fixed
    - **Pruning**: prune away a certain fraction of the weights of the network, i.e. set them to zero
    - **Retrain**: to regain accuracy after pruning (half epochs)
    - **Freeze**: Task parameters are frozen.

- 1.5× more expensive than simple finetuning

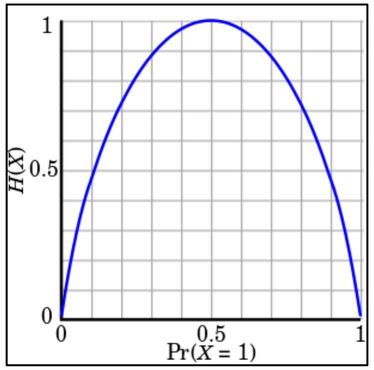
PackNet: Adding Multiple Tasks to a Single Network by Iterative Pruning, Mallya et al 2017. 24

Task Inference with Sparse Models (SupSup)

- After training, we have one mask (model) for each task
- In a task-agnostic setting, how do we choose the task label?
- Good heuristic: select the most confident model
  - WARNING: Keep in mind that neural networks may be highly overconfident, so this method doesn't always work
- We can use the entropy to measure the confidence



The entropy measures can be used to measure the confidence. For example, the entropy of a coin flip is maximal when p(H)= 0.5



Entropy Eq:

$$\mathrm{H}(X) := -\sum_{x\in\mathcal{X}} p(x)\log p(x) = \mathbb{E}[-\log p(X)]$$

Image source: wikimedia
Supermasks in Superposition, Wortsman et al. 2020.

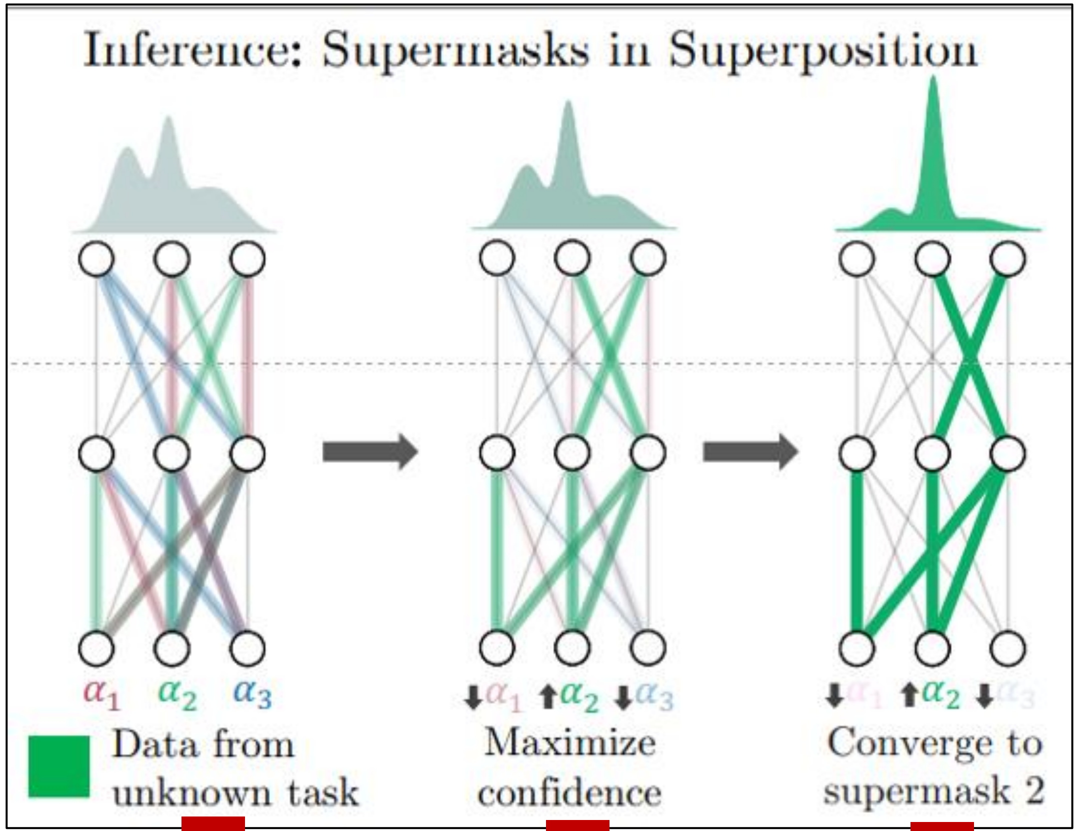
SupSup – Superposition

Superposition: a weighted sum of all the masks

- An approximation of the ensemble output
- $\mathbf{p}(\alpha) = f\left(\mathbf{x}, W \odot \left(\sum_{i=1}^{k} \alpha_i M^i\right)\right)$
- Requires a single forward pass

ONE-SHOT TASK INFERENCE:

- (1) Compute $\mathbf{p}(\alpha)$
- (2) Compute gradient with respect to entropy and do an SGD step on α
- (3) Choose the mask s.t. $\arg\max i \left(-\frac{\partial \mathcal{H}(\mathbf{p}(\alpha))}{\partial \alpha_i}\right)$
  - This is a single step of SGD
  - You could optimize α until convergence but one step is sufficient



Inference: Supermasks in Superposition
α_1 α_2 α_3
🟩 Data from unknown task
$\downarrow\alpha_1$ $\uparrow\alpha_2$ $\downarrow\alpha_3$
Maximize confidence
$\uparrow\alpha_2$
Converge to supermask 2

1

2

3

Conclusion

- **We can solve TIL** with
  - Architectural methods that expand the model over time
  - Sparse models
- **Good solutions to prevent forgetting but poor transfer**
  - Some methods are just a smart version of the basic ensemble of independent models
- **Task inference** removes the requirements of task labels
  - Input-based task predictors
  - Confidence-based task inference

Federated Continual Learning

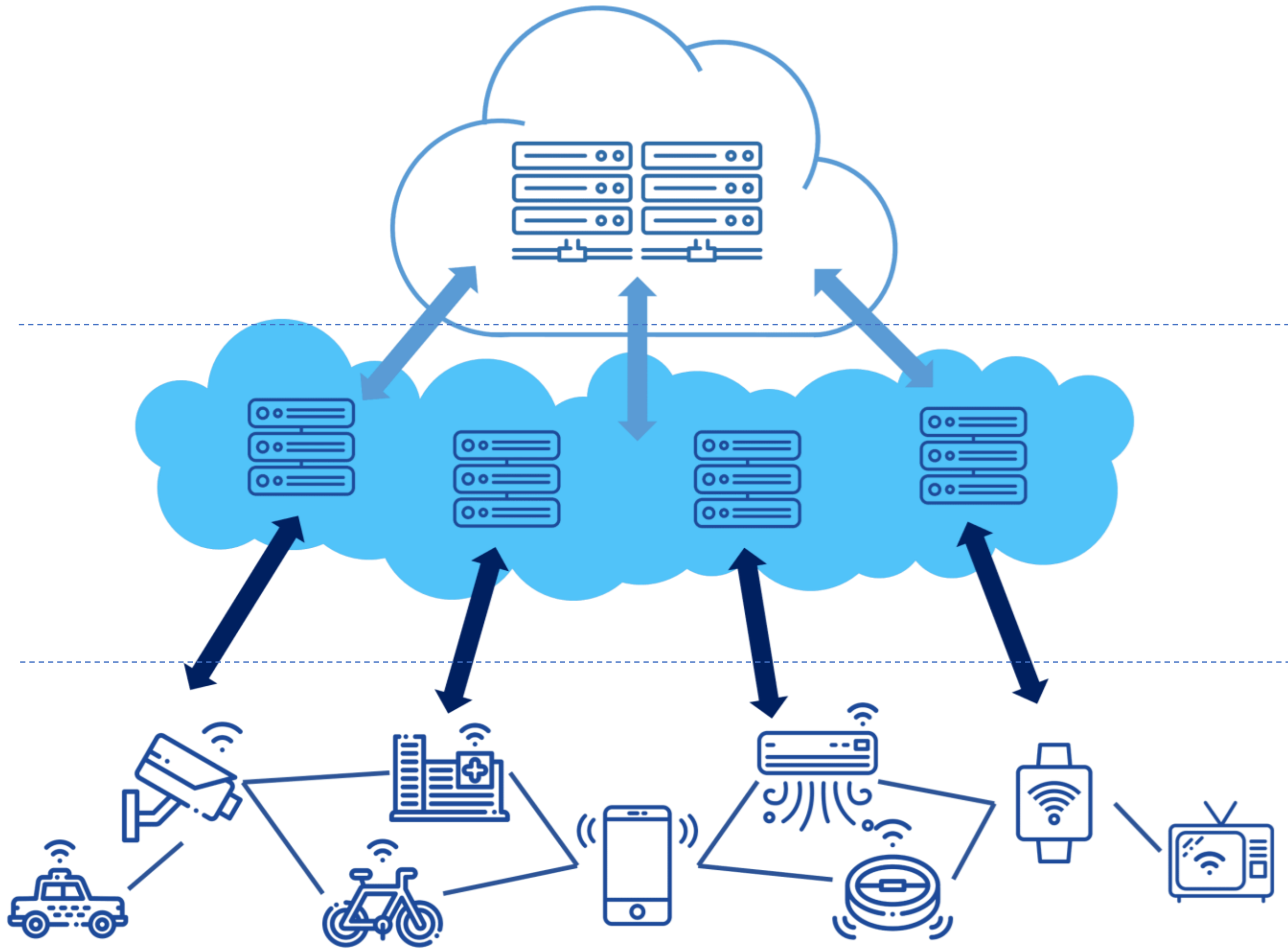
with slides from Valerio De Caro

Goal

- Learn with a large number of devices
- Learning algorithm is controlled by a centralized server
- There is a common goal (e.g. learning a single or multiple tasks)

For simplicity, we focus on the learning problem and ignore
- Communication costs
- Implementation issues
- Infrastructural issues

Cloud-Edge Continuum



Cloud

- Thousands of nodes
- Massive computational resources
- High Bandwidth, low latency
- Permanent availability

Fog

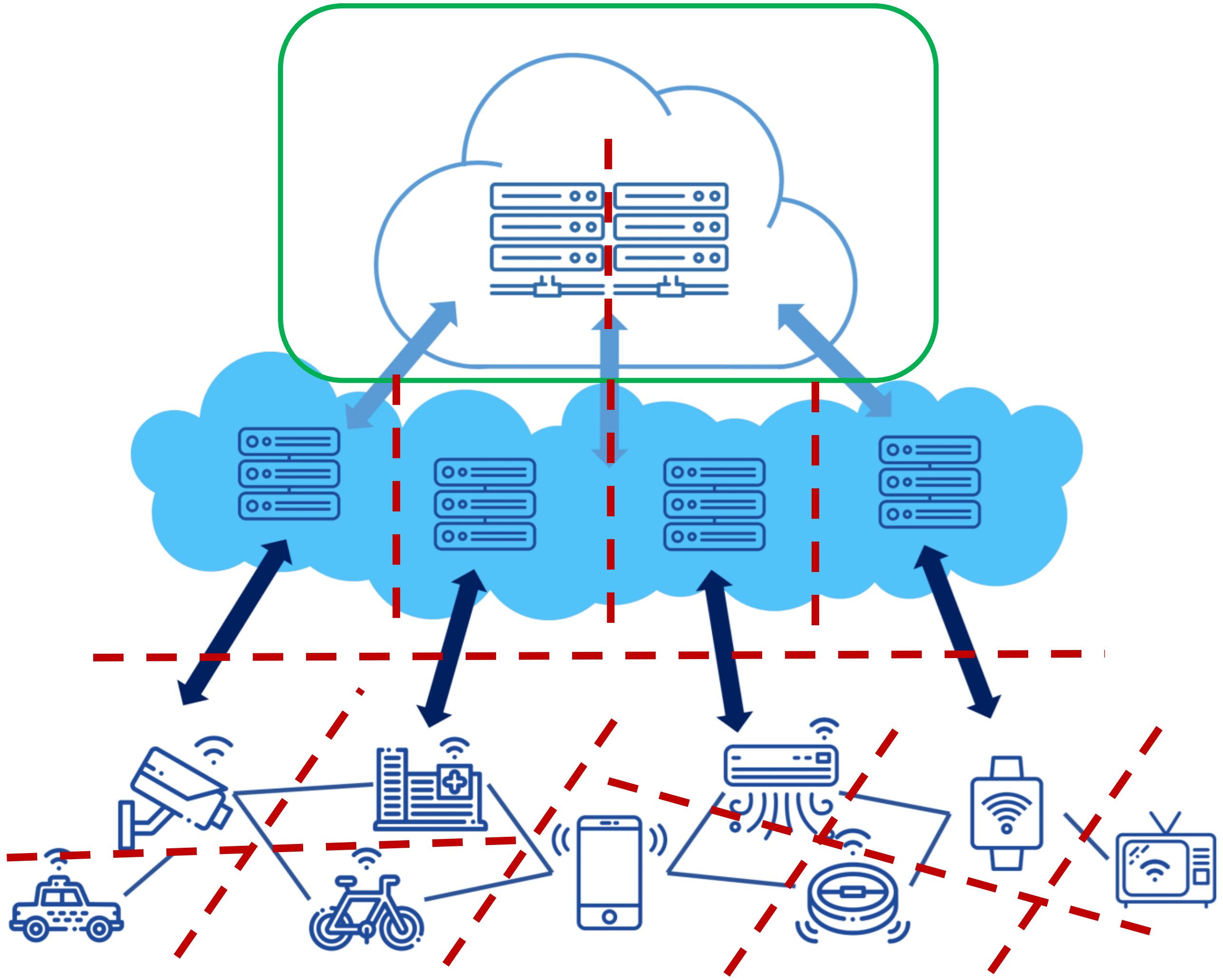
- Millions of nodes
- Moderate computational resources
- Moderate bandwidth and latency
- High availability

Edge

- Billions of nodes
- Low computational resources
- Low bandwidth and high latency
- Limited availability
- Inherently prolific data source

- A plethora of available resources for running ML tasks

Distributed vs Federated



- Distributed Learning:
  - Cloud-scale resources
  - All the data available on cloud
  - It's your usual learning process, just scaled up and faster

– – = Data is regulated by Privacy

- Federated Learning:
  - Semantic and system heterogeneity
  - Leverages the whole continuum, optimizing the utilization of the available resources
  - Complies with privacy constraints

Federated Learning

- Learning a global model consists in minimizing the following function:

$$F(\mathbf{x}) = \mathbb{E}_{\boxed{i\sim\mathcal{P}}}[F_i(\mathbf{x})] \Rightarrow F_i(\mathbf{x}) = \mathbb{E}_{\boxed{\xi\sim\mathcal{D}_i}}[f_i(\mathbf{x},\xi)]$$

Client Distribution: denotes client availability and resources, i.e., **system heterogeneity**

Unobservable *

Local Data Distribution: denotes heterogeneity of local data across clients, i.e., **statistical heterogeneity**

*Thus, we approximate the learning problem by **Empirical Risk Minimization**:

$$F^{ERM}(\mathbf{x}) = \sum_{i=1}^{|\mathcal{C}|} p_i F_i^{ERM}(\mathbf{x}) \Rightarrow F_i^{ERM}(\mathbf{x}) = \frac{1}{|D_i|}\sum_{\xi\in D_i} f_i(\mathbf{x},\xi)$$

Approximates system heterogeneity and statistical heterogeneity

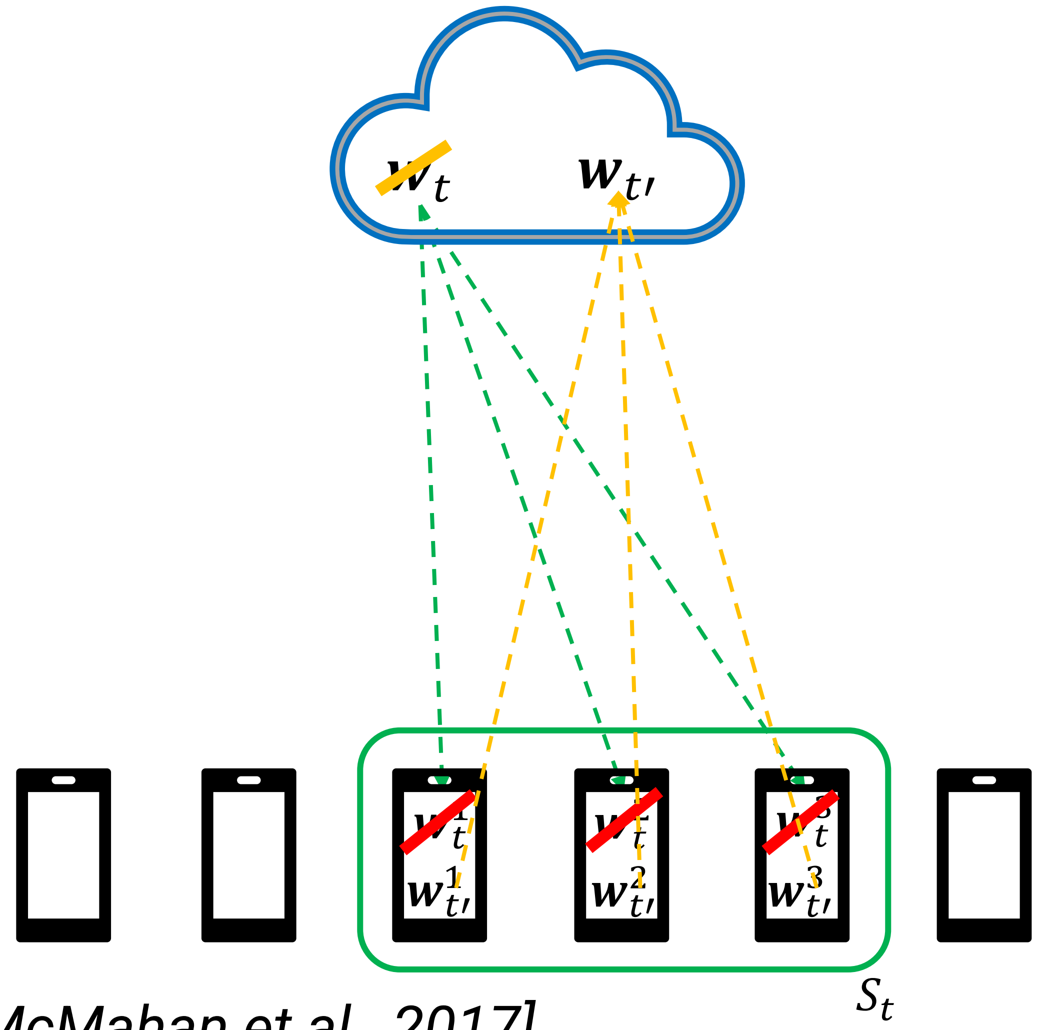
! No data points are fed to $F^{ERM}(\mathbf{x})$, thus **there is no direct evaluation of the global model**

Federated Averaging

Algorithm 1 FederatedAveraging. The K clients are indexed by k; B is the local minibatch size, E is the number of local epochs, and η is the learning rate.

```
Server executes:
    initialize $w_0$
    for each round $t = 1, 2, \dots$ do
        $m \leftarrow \max(C \cdot K, 1)$
        $S_t \leftarrow$ (random set of $m$ clients)
        for each client $k \in S_t$ in parallel do
            $w_{t+1}^k \leftarrow$ ClientUpdate$(k, w_t)$
        $w_{t+1} \leftarrow \sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$

ClientUpdate$(k, w)$:   // Run on client $k$
    $\mathcal{B} \leftarrow$ (split $\mathcal{P}_k$ into batches of size $B$)
    for each local epoch $i$ from 1 to $E$ do
        for batch $b \in \mathcal{B}$ do
            $w \leftarrow w - \eta \nabla \ell(w; b)$
    return $w$ to server
```



A Naïve approach to Federated Optimization [McMahan et al., 2017]

Federated Continual Learning

Federated Continual Learning

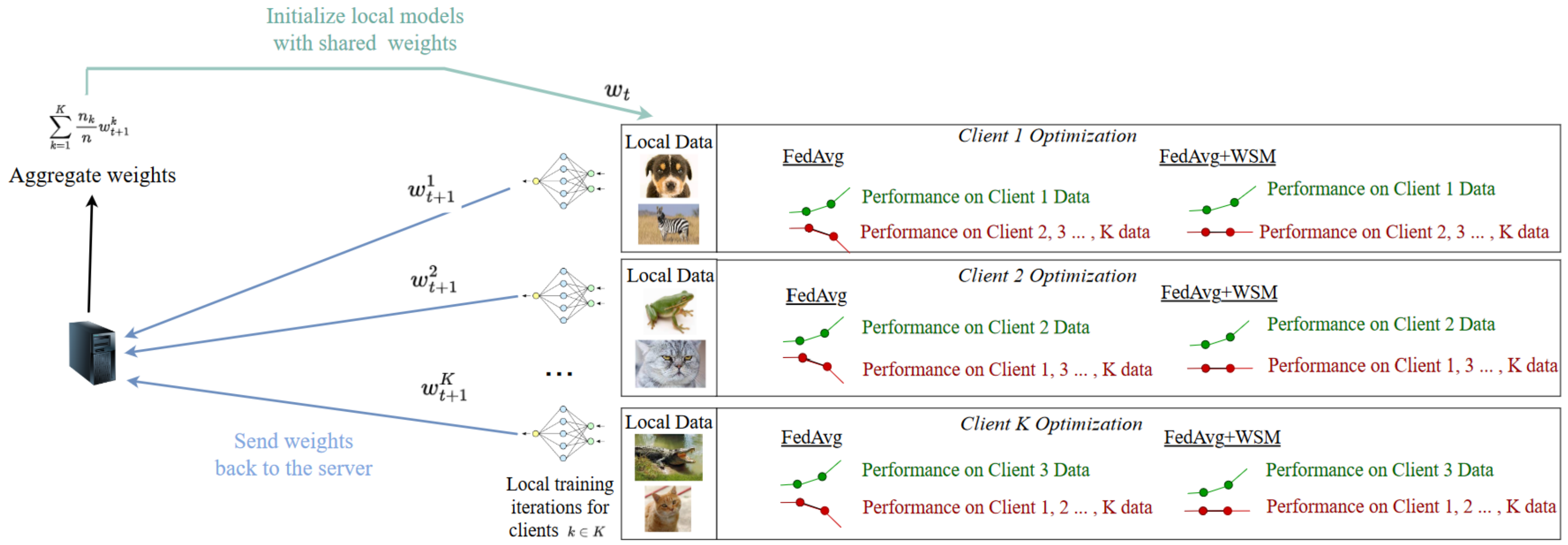
- Multiple Clients
- Each learning from a stream of tasks (assume task labels are available)
- We can have local forgetting (client, local model) and global forgetting (server, global model)

Continual Federated Learning

- **FL** methods fail in simple heterogeneous settings.
- **Local forgetting** happens in heterogeneous FL if the local models are not aggregated often enough, resulting in a local drift and forgetting of the global knowledge.

Open question: can continual learning improve federated learning in heterogeneous settings?

- **[1]** proposes **WSM loss**, a weighted cross-entropy to mitigate this problem



Initialize local models with shared weights
w_t
$\sum_{k=1}^{K} \frac{n_k}{n} w_{t+1}^k$
Aggregate weights
w_{t+1}^1
w_{t+1}^2
w_{t+1}^K
Send weights back to the server
Local training iterations for clients $k \in K$
Local Data
Client 1 Optimization
FedAvg
Performance on Client 1 Data
Performance on Client 2, 3 ... , K data
FedAvg+WSM
Performance on Client 1 Data
Performance on Client 2, 3 ... , K data
Client 2 Optimization
Performance on Client 2 Data
Performance on Client 1, 3 ... , K data
Client K Optimization
Performance on Client 3 Data
Performance on Client 1, 2 ... , K data

Figure 1: *Illustration of catastrophic forgetting within client rounds.* A global model with knowledge of all classes is sent to all clients participating in a given FL round. Local training increases the client model performance on the client's local distribution but tends to simultaneously decrease performance with respect other clients distributions which leads to poor aggregation and overall model performance.

[1] G. Legate et al. "Re-Weighted Softmax Cross-Entropy to Control Forgetting in Federated Learning." CoLLAs '23

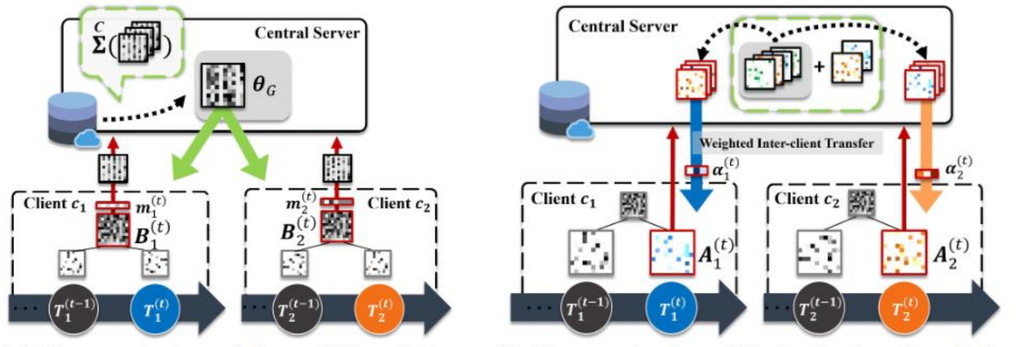
FedWelt

We can use task-incremental architectures just like we did for TIL!

- **Objectives**:
  - Minimize communication
  - Exploit task similarity
  - Avoid task interference
  - Define a different model for each client (clients may share only part of the model)
- **Modularized task-based model**:
  - Global parameters
  - Local base parameters
  - Task-adaptive parameters

Local client model:

$$\boldsymbol{\theta}_c^{(t)} = \mathbf{B}_c^{(t)} \odot \mathbf{m}_c^{(t)} + \mathbf{A}_c^{(t)} + \sum_{i \in \mathcal{C}_{\setminus c}} \sum_{j < |t|} \alpha_{i,j}^{(t)} \mathbf{A}_i^{(j)}$$



(a) Communication of General Knowledge (b) Communication of Task-adaptive Knowledge

Figure 3. Updates of *FedWeIT*. **(a)** A client sends sparsified federated parameter $\boldsymbol{B}_c \odot \boldsymbol{m}_c^{(t)}$. After that, the server redistributes aggregated parameters to the clients. **(b)** The knowledge base stores previous tasks-adaptive parameters of clients, and each client selectively utilizes them with an attention mask.

J. Yoon et al. "Federated Continual Learning with Weighted Inter-Client Transfer." ICML '21

Limitations of Federated Learning

- Requires frequent communication to aggregate models
- Only useful if clients are solving the same tasks
- Requires a server to orchestrate the learning algorithm
- More in general, the server is in control
  - What happens if we remove this assumption and assume that each device is an independent entity?
  - Can we share knowledge between devices?
  - How can we do the model aggregation?

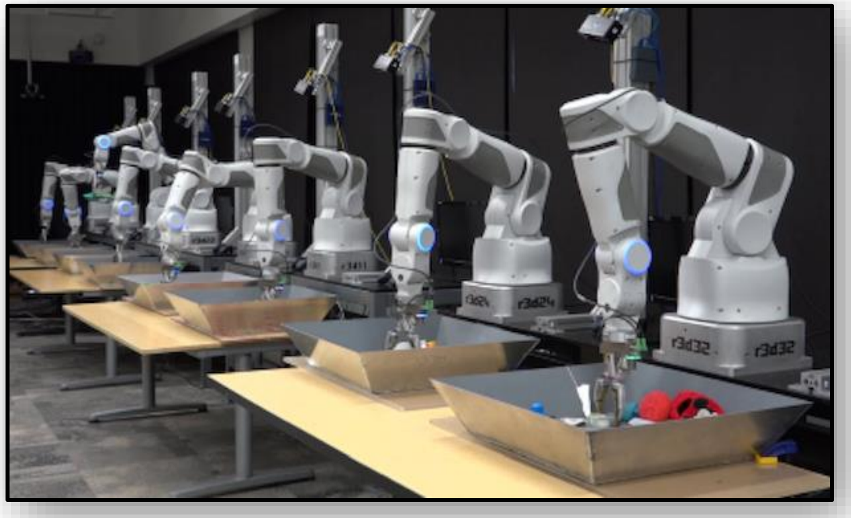
Towards Multiple Agents

Asynchronous and Independent Agents

Multiple agents:

- Learn tasks independently
- Do not have a single centralized server that orchestrates learning
- Want to «learn from each other» if possible, but at a minimal cost

- Related work: A Call to Build Models Like We Build Open-Source Software, Colin Raffel https://colinraffel.com/blog/a-call-to-build-models-like-we-build-open-source-software.html

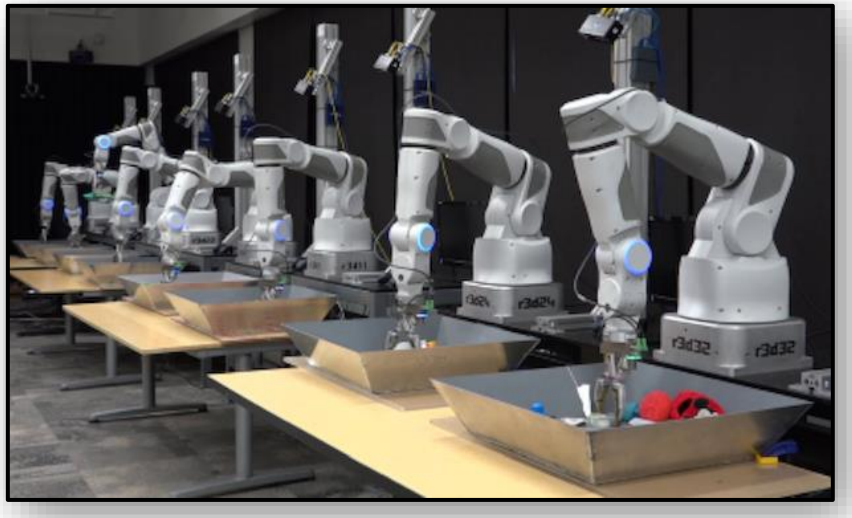


Multi-Agent Continual Learning

In a multi-agent setting, agents can “talk” to each other and share knowledge.

Desiderata

- **Reuse of expert knowledge**
- **Efficient** and **decentralized** learning
- **Independent agents** (unlike federated learning)
- **Privacy** (at will)



Ex-Model Continual Learning

Let's focus on the problem of **«knowledge consolidation»**
Instead of data, at each learning experience
the model receives an expert model.
We want to aggregate the models together

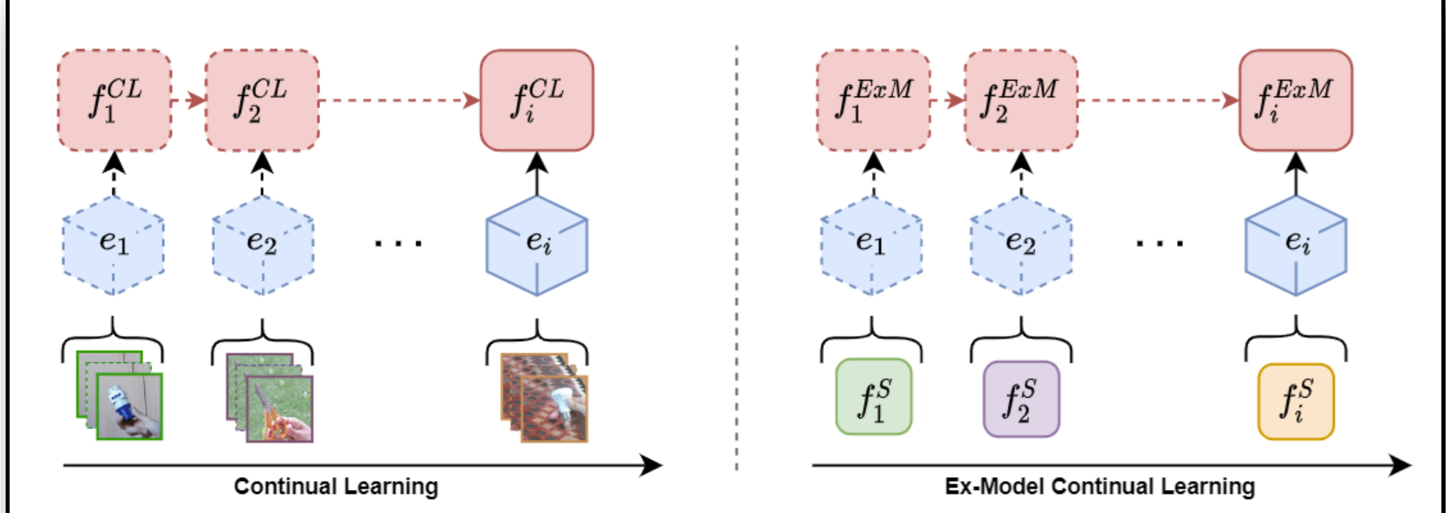


Figure 1. Ex-model Continual Learning. The stream is composed of expert models, without access to the original data.

Ex-Model: Continual Learning From a Stream of Trained Models. *Antonio Carta, Andrea Cossu, Vincenzo Lomonaco, Davide Bacciu*; CLVISION@CVPRW, 2022

Ex-Model CL

- **Model aggregation is the critical missing component in heterogeneous FL!**
  - We know how to train the local model (continual learning)
  - We know how to aggregate homogeneous models as long as the aggregation is frequent enough (homogeneous federated learning)
- **If we can aggregate independent models (Ex-Model CL)**
  - we can train on multiple tasks in parallel
  - Without frequent synchnonous aggregations
  - Allows decentralized training
  - related to model patching [1]

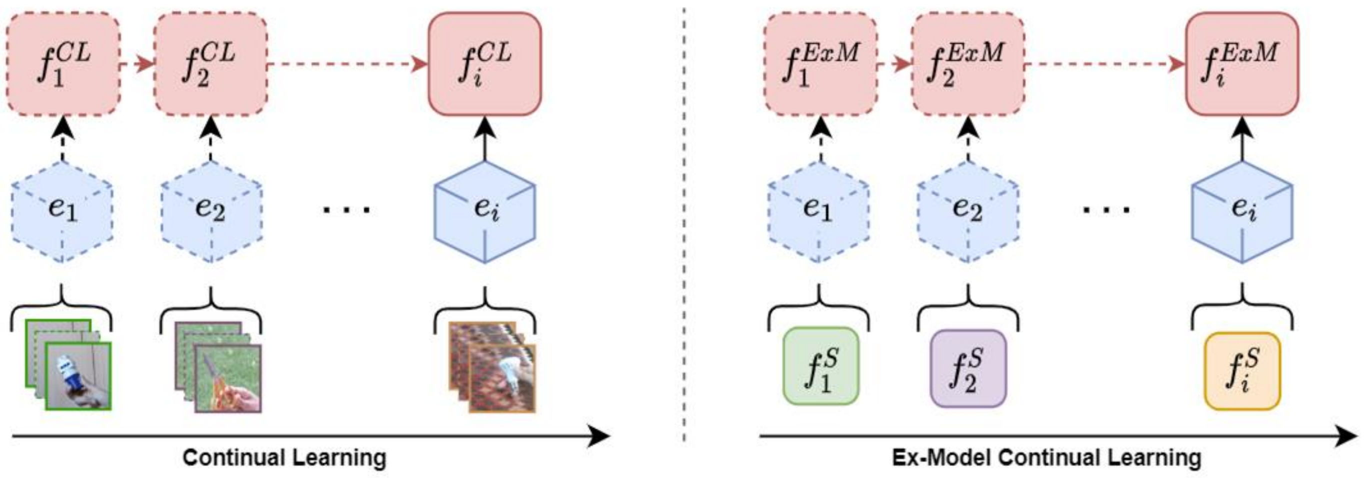


Figure 1. Ex-model Continual Learning. The stream is composed of expert models, without access to the original data.

A. Carta. "Ex-Model: Continual Learning From a Stream of Trained Models," CLVISION '22
[1] Raffel, Colin. "Building Machine Learning Models Like Open Source Software." Communications of the ACM 2023

Functional Regularization

How to share pretrained knowledge

Functional Regularization

- we have access to a model f_{i-1}^{CL} (the previous model) that learned experiences $S^{train}[1{:}\, i-1]$
- **IDEA**: let's replicate the old model behavior and update it only on the new examples
- **PROBLEMS**:
  - What objective do we use?
  - What data do we use?

$$f_i^{CL}(\boldsymbol{x}, k) = f_{i-1}^{CL}(\boldsymbol{x}, k), \quad \forall \boldsymbol{x} \in \mathbb{R}^N, k \neq i$$
$$f_i^{CL}(\boldsymbol{x}, i) = f_i^{Exp}(\boldsymbol{x}), \quad \forall \boldsymbol{x} \in \mathbb{R}^N.$$

Equations in a Multi-Task Scenario
x=input
k=task label
i=task label for new task
f^{Exp}=model for new task

First Eq: copy old CL model on the first i-1 tasks
Second Eq: copy new model on the new task

Knowledge Distillation (KD)

KD: Offline training method to replicate the output of a pretrained model

- **Teacher**: pretrained model
- **Student**: the new model that we want to train

KD is a general method with many applications outside CL:

- Example: Reducing the size of a model:
  - Example teacher: ResNet101 pretrained on ImageNet
  - Example student: ResNet18 trained with KD

Why does it work?

- Supervised training provides hard targets (i.e. the correct class)
- KD provides soft targets, which are more informative
  - Example: soft targets encode similarities between classes
  - Informally called «dark knowledge»

KD Objective – KL-Divergence and MSE

- The KL-Divergence measures the similarity between two probability distributions (teacher and student)
  - We are measuring the distance between the pdf of the teacher and the student
- $\hat{y}$ teacher, y student

$$E(\mathbf{x} \mid t) = -\sum \hat{y}_i(\mathbf{x} \mid t) \log y_i(\mathbf{x} \mid t)$$

- Alternative: MSE between the logits $||\hat{y} - y||_2^2$
  - Often more robust in CL

Learning Without Forgetting (LwF)

LwF implements functional regularization with:

- **Objective**: knowledge distillation
- **Data**: current data

Key Aspects

- Straightforward application of KD in CL
- Originally designed for Task-Incremental settings then extended to single task.
- Efficient: requires only an additional forward pass with the teacher.
- Easy to implement and commonly used

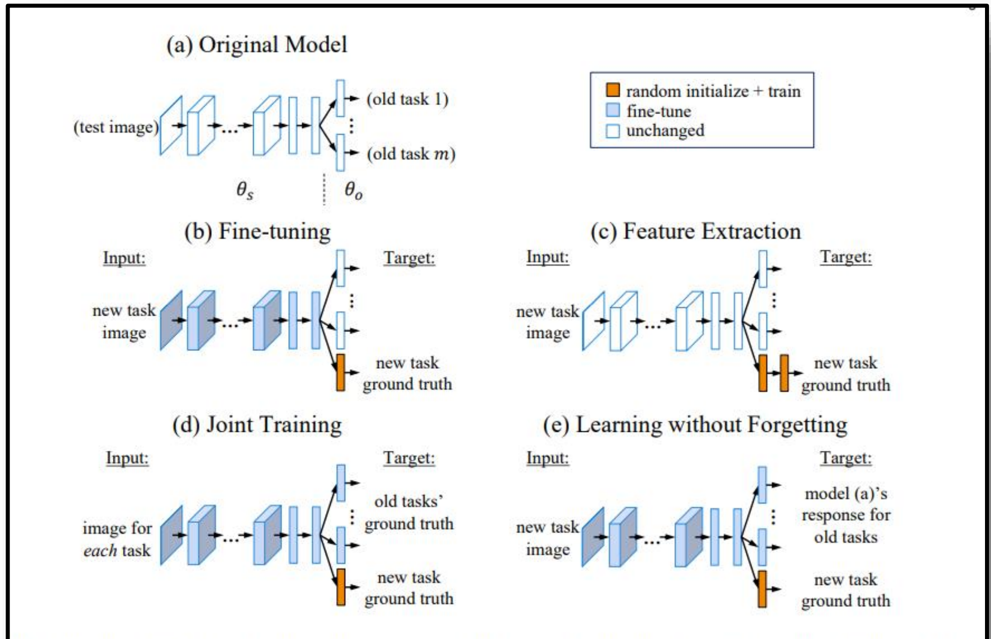


Fig. 2. Illustration for our method (e) and methods we compare to (b-d). Images and labels used in training are shown. Data for different tasks are used in alternation in joint training.

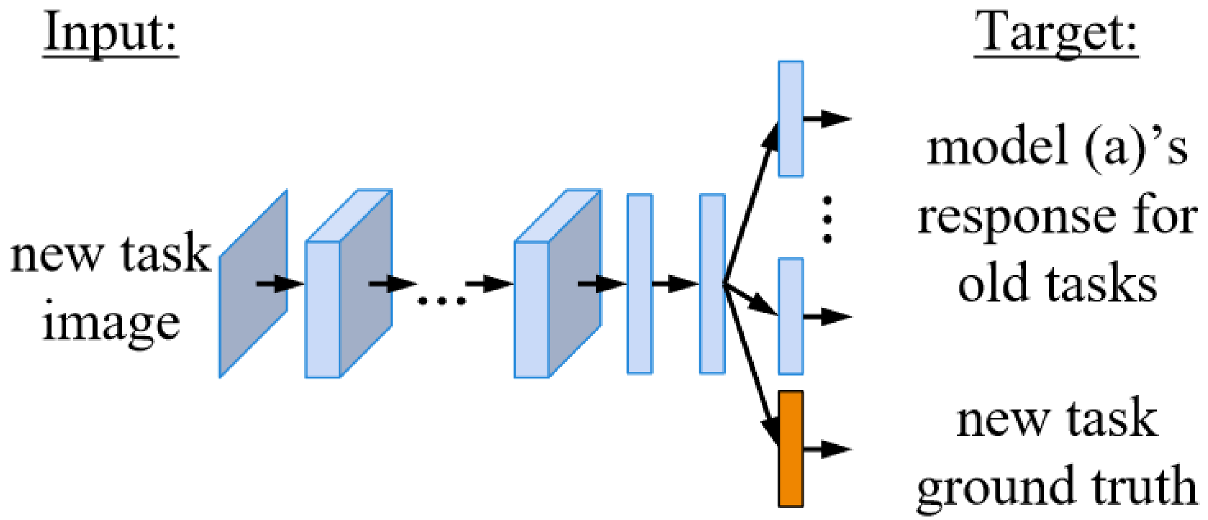
Learning without Forgetting, Li et al, TPAMI 2017
Distilling the knowledge in a neural network, Hinton et al, 2015.
Continuous Learning in Single-Incremental Tasks, Maltoni & Lomonaco, Neural Networks, 2019.

LwF in Task-Incremental CL

- **Multihead**: separate output layer for each task.
- **KD** computed on the old head using the new data (L_{old})
- **CE** computed on the new head using the new data (L_{new})

$$\mathcal{L}_{old}(\mathbf{y}_o, \hat{\mathbf{y}}_o) = -H(\mathbf{y}'_o, \hat{\mathbf{y}}'_o) = -\sum_{i=1}^{l} y'^{(i)}_o \log \hat{y}'^{(i)}_o$$

$$\mathcal{L}_{new}(\mathbf{y}_n, \hat{\mathbf{y}}_n) = -\mathbf{y}_n \cdot \log \hat{\mathbf{y}}_n$$



$$\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n)$$

Learning without Forgetting, Li et al, TPAMI 2017

Weight vs Functional Regularization

- **Prior-based methods (EWC) require that the new models don't move «too far» from the previous solutions**
    - We need a large model or a pretrained one to satisfy this requirement
    - It's only an approximation of the real objective
- **Functional regularization is much less restrictive**
    - The weights can change, as long as the output for the previous units is the same
    - The output of new units is completely unconstrained
    - The previous model provides the exact outputs that we want (no approximation)

Distillation in Practice

- Even outside CL, KD is surprisingly competitive

Implementation «tricks»

- Long aumentation pipelines
- Long training Schedules
- Consistent teacher and student inputs (i.e. same augmentation for both)

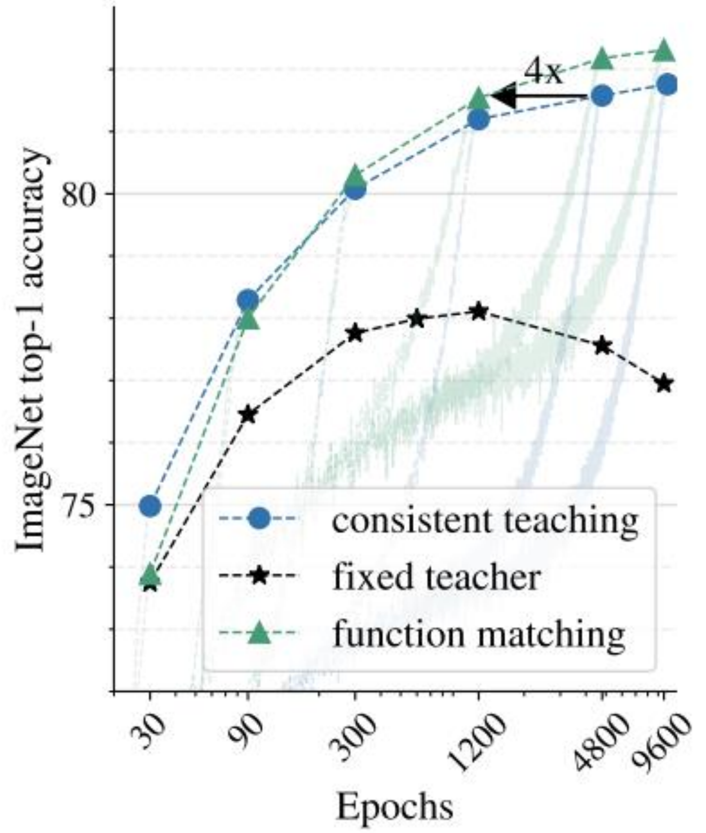
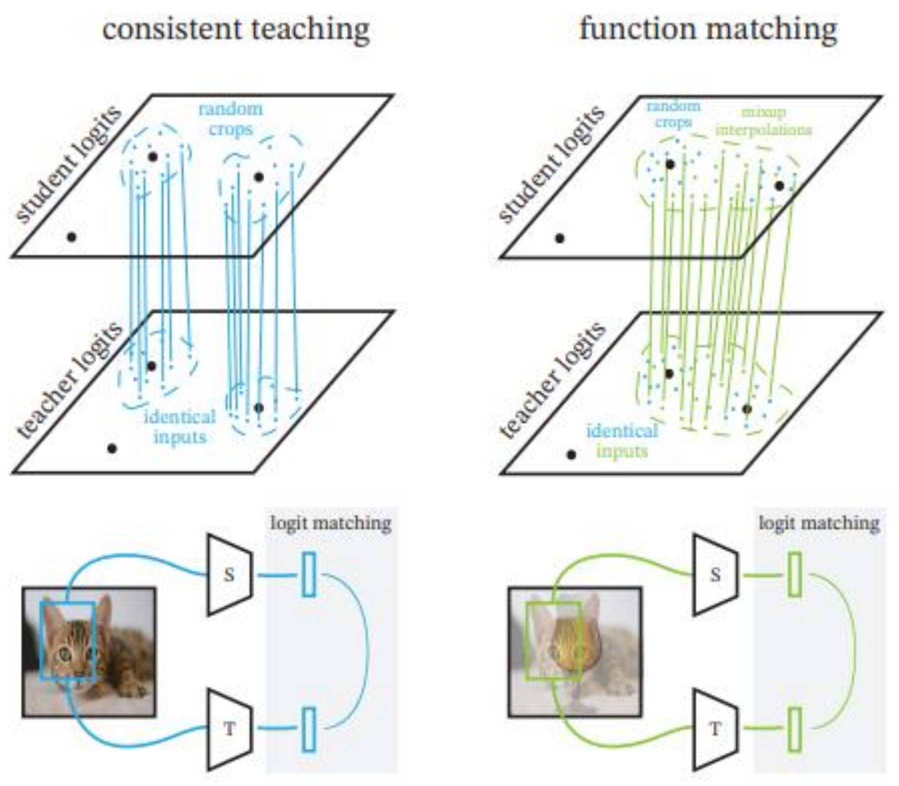


Figure 1. We demonstrate that distillation works the best when we train *patiently* for a large number of epochs and provide *consistent* image views to teacher and student models (green and blue lines). This can be contrasted to a popular setting of distilling with precomputed teacher targets (black line), which works much worse.

KD as Function Matching

KD is a **function matching** problem

- For any input, we know exactly what output we want
  - The data is much less important in this problem than in a typical ML problem
  - Having diverse data close to the domain of interest helps but it's not a necessity
- KD works with any inputs (with different convergence speeds)
  - The teacher training data
  - Out-of-domain-data with augmentations
  - A new task data (like in LwF!!!)



KD with a Single Image

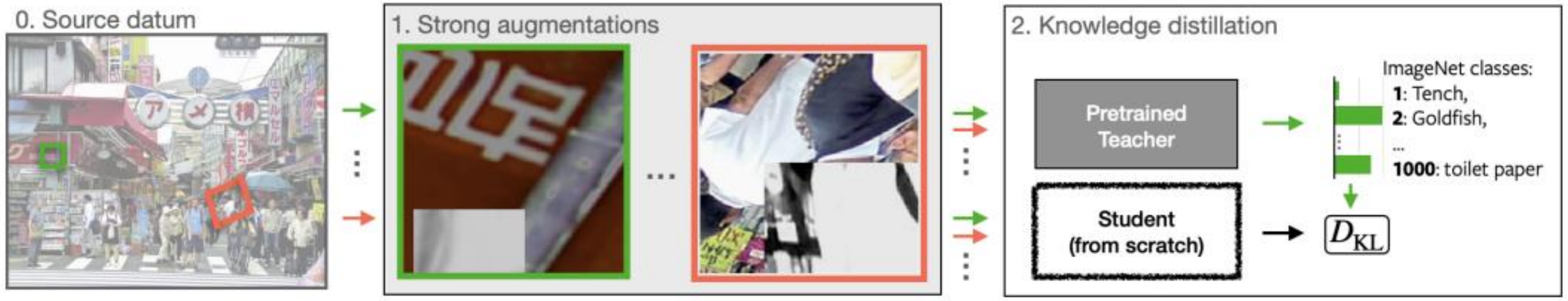


Figure 1: **Extrapolating from one image.** Strongly augmented patches from a single image are used to train a student (S) to distinguish semantic classes, such as those in ImageNet. The student neural network is initialized randomly and learns from a pretrained teacher (T) via KL-divergence. Although almost none of target categories are present in the image, we find student performances of $>69\%$ for classifying ImageNet's 1000 classes. In this paper, we develop this single datum learning framework and investigate it across datasets and domains.

LwF

- We need **task boundaries** to know **when** to store the previous model
- The data that we use for KD (new data) is different from the one we used to train the teacher (old data)
- **Augmentations** help KD, even when they distort the image

Multi-Agent Knowledge Distillation

Ex-Model CL

- **Model aggregation is the critical missing component in heterogeneous FL!**
  - We know how to train the local model (continual learning)
  - We know how to aggregate homogeneous models as long as the aggregation is frequent enough (homogeneous federated learning)
- **If we can aggregate independent models (Ex-Model CL)**
  - we can train on multiple tasks in parallel
  - Without frequent synchnonous aggregations
  - Allows decentralized training
  - related to model patching [1]

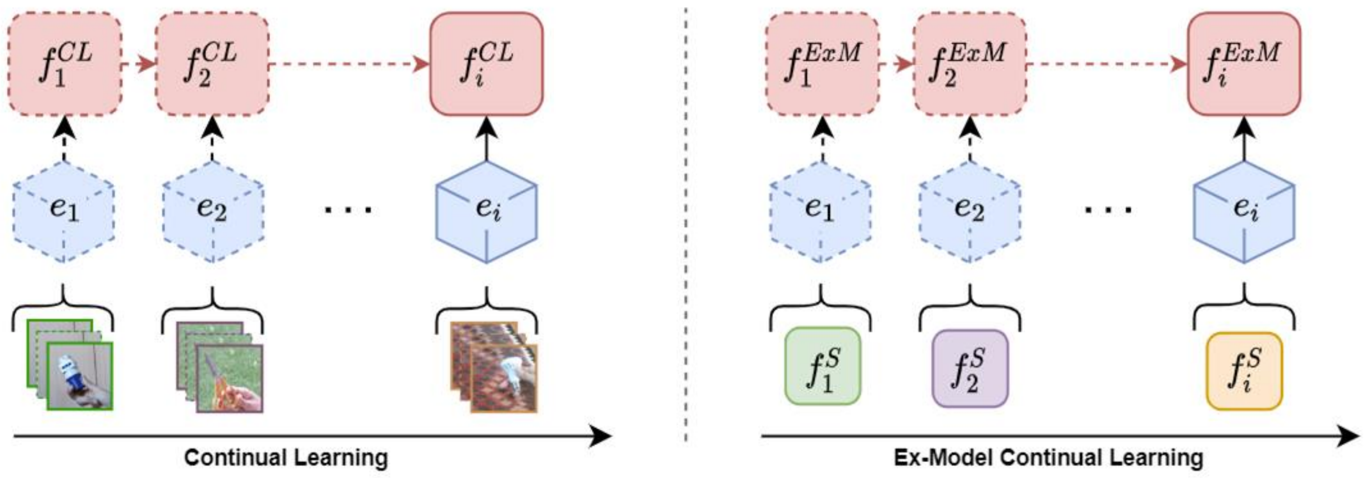


Figure 1. Ex-model Continual Learning. The stream is composed of expert models, without access to the original data.

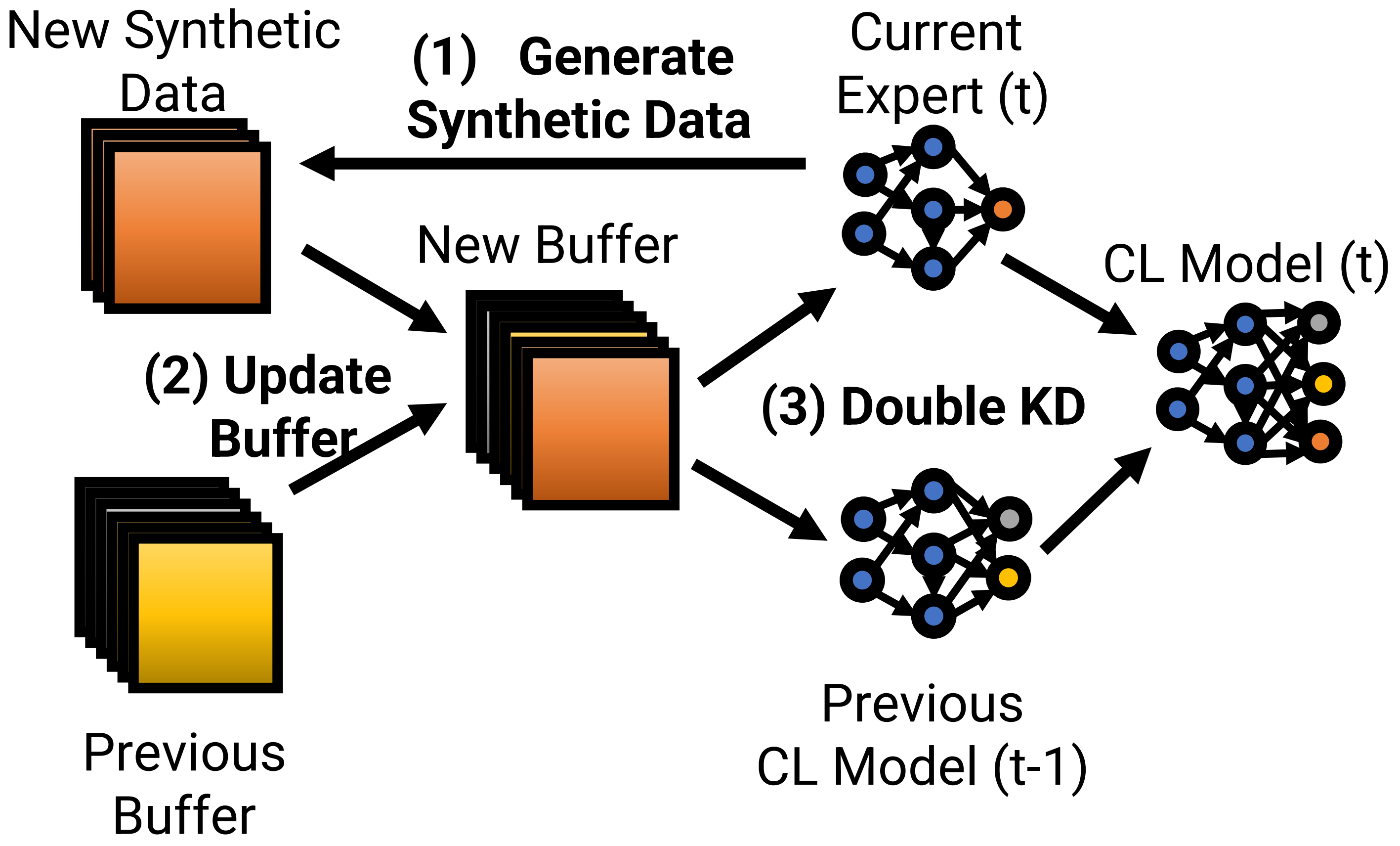
A. Carta. "Ex-Model: Continual Learning From a Stream of Trained Models," CLVISION '22

[1] Raffel, Colin. "Building Machine Learning Models Like Open Source Software." Communications of the ACM 2023

ExML Distillation (1)

Syntethic Data

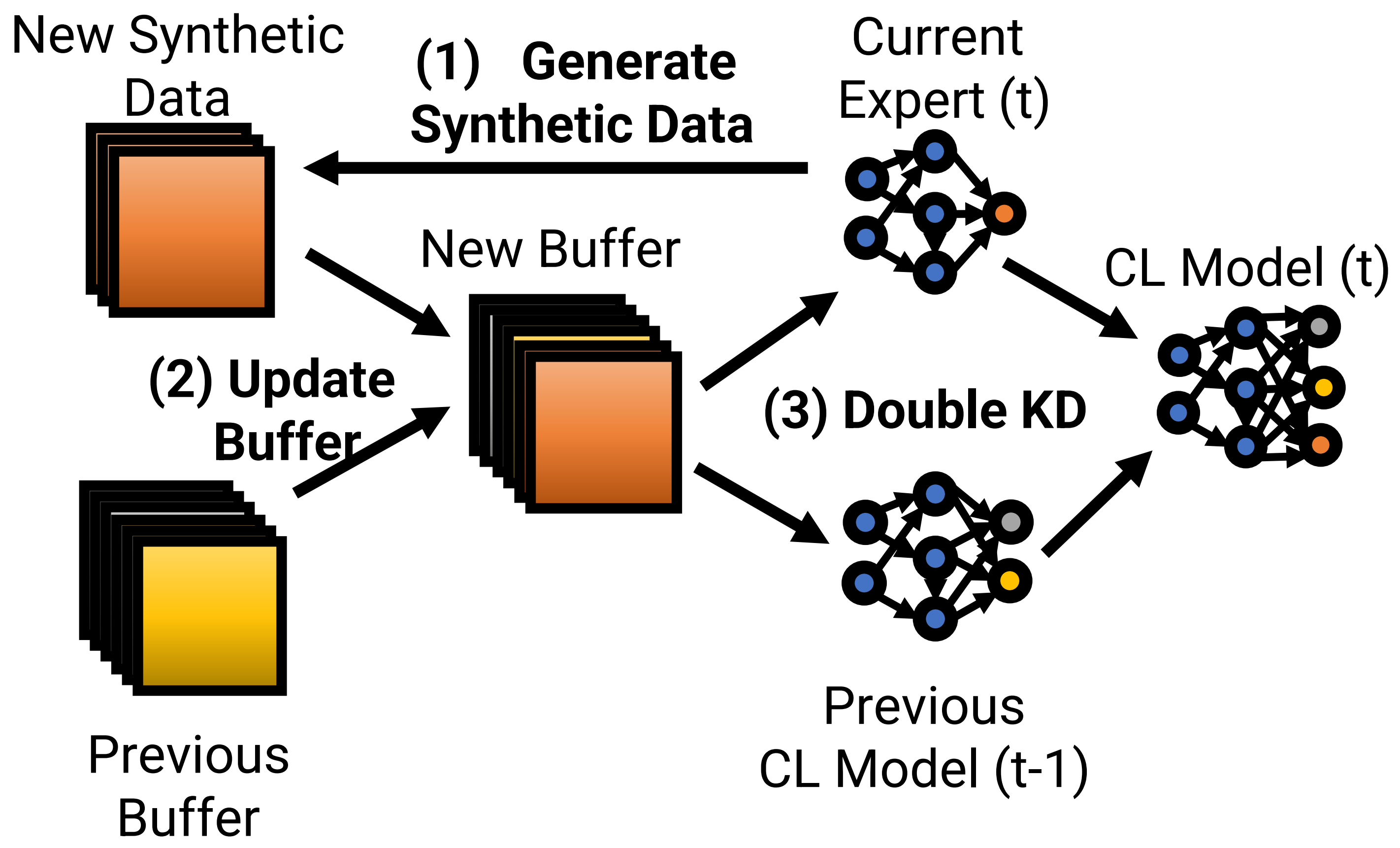
- Model inversion: optimize noise to resemble a class
- +natural image prior
- Alternative: ood data



ExML Distillation (2)

Rehearsal Buffer

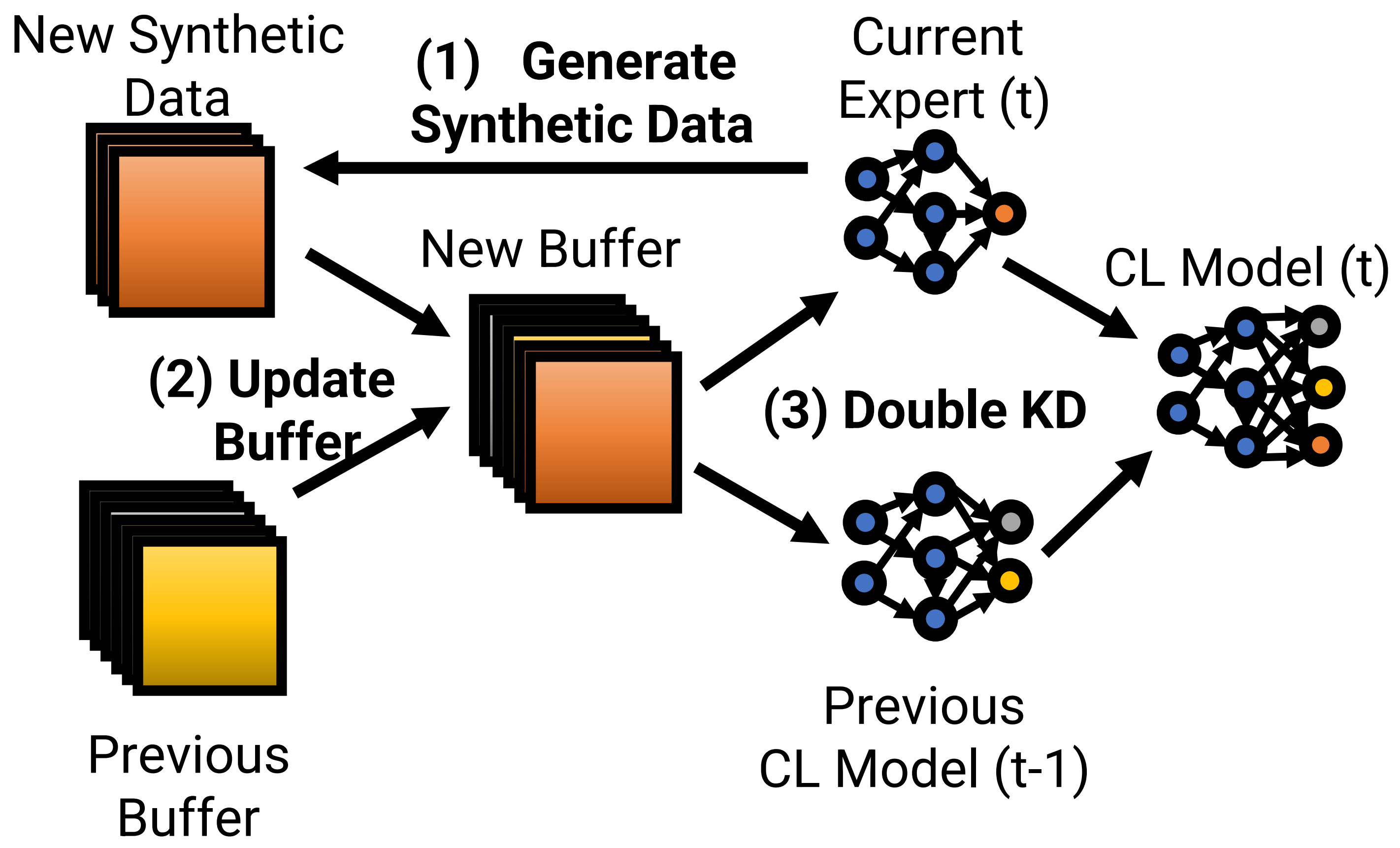
- Synthetic data is stored in a buffer
- Buffer is updated after every experience



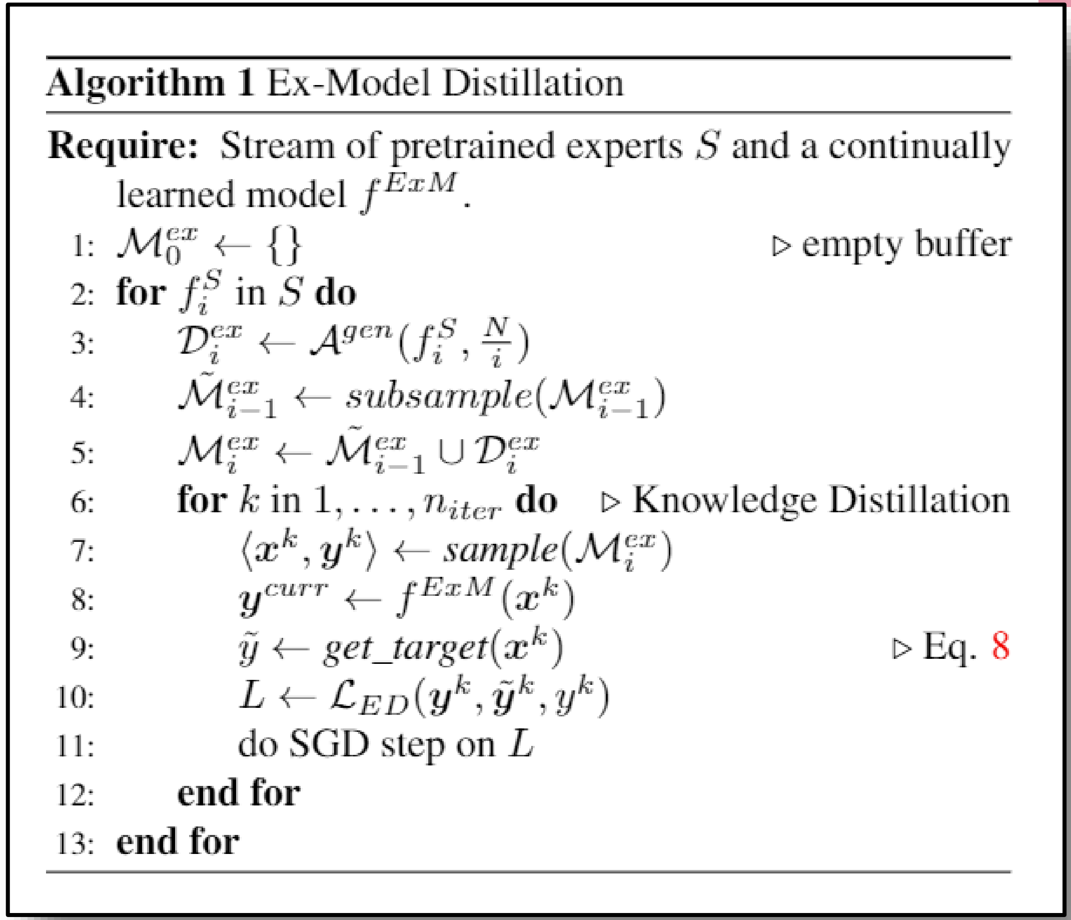
ExML Distillation (3)

Double Distillation

- Distillation from previous CL model and current expert
- Instance-based logits normalization to avoid bias towards one model



Ex-Model Distillation

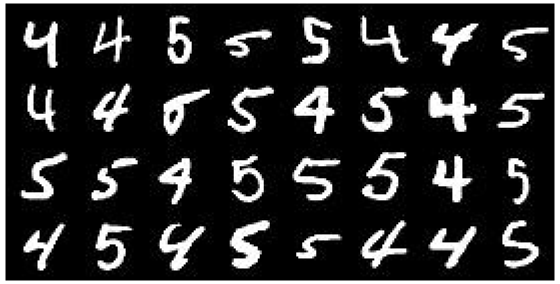


Algorithm 1 Ex-Model Distillation

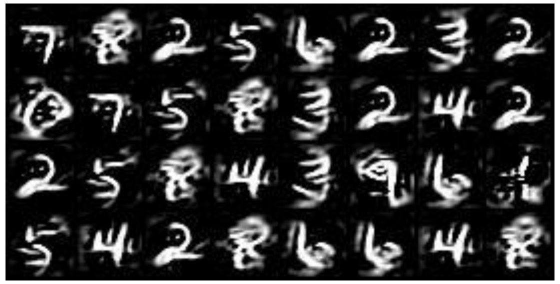
```
Require: Stream of pretrained experts $S$ and a continually
    learned model $f^{ExM}$.
 1: $\mathcal{M}_0^{ex} \leftarrow \{\}$                                   ▷ empty buffer
 2: for $f_i^S$ in $S$ do
 3:     $\mathcal{D}_i^{ex} \leftarrow \mathcal{A}^{gen}(f_i^S, \frac{N}{i})$
 4:     $\tilde{\mathcal{M}}_{i-1}^{ex} \leftarrow subsample(\mathcal{M}_{i-1}^{ex})$
 5:     $\mathcal{M}_i^{ex} \leftarrow \tilde{\mathcal{M}}_{i-1}^{ex} \cup \mathcal{D}_i^{ex}$
 6:     for $k$ in $1, \dots, n_{iter}$ do            ▷ Knowledge Distillation
 7:         $\langle \boldsymbol{x}^k, \boldsymbol{y}^k \rangle \leftarrow sample(\mathcal{M}_i^{ex})$
 8:         $\boldsymbol{y}^{curr} \leftarrow f^{ExM}(\boldsymbol{x}^k)$
 9:         $\tilde{y} \leftarrow get\_target(\boldsymbol{x}^k)$                ▷ Eq. 8
10:         $L \leftarrow \mathcal{L}_{ED}(\boldsymbol{y}^k, \tilde{\boldsymbol{y}}^k, y^k)$
11:         do SGD step on $L$
12:     end for
13: end for
```

ExML – Generated Samples

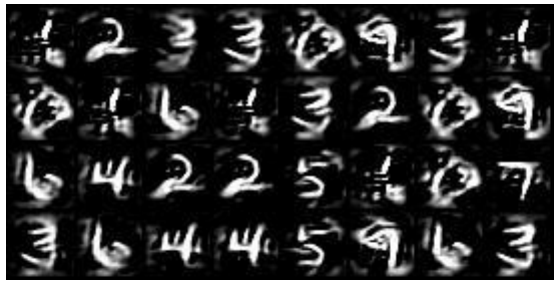
Original Data



Joint MNIST, Model Inversion

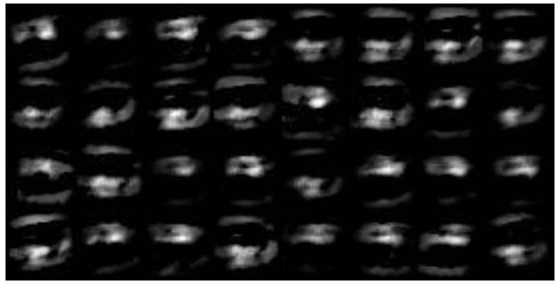


Joint MNIST, Data Impression

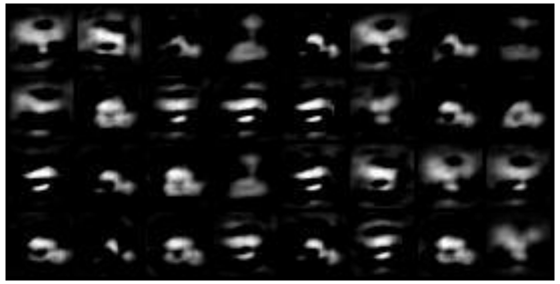


OOD Data

Split MNIST, Model Inversion



Split MNIST, Data Impression



ExML - Challenges

- We have only the expert model, not the original training data
- We cannot store all the experts in memory
- Experts are overconfident on out-of-distribution samples

Some of these limitations can be lifted with additional assumptions:

- Access to a subset of the original data
- A shared training protocol
- Sending the model every few epochs instead of only after convergence

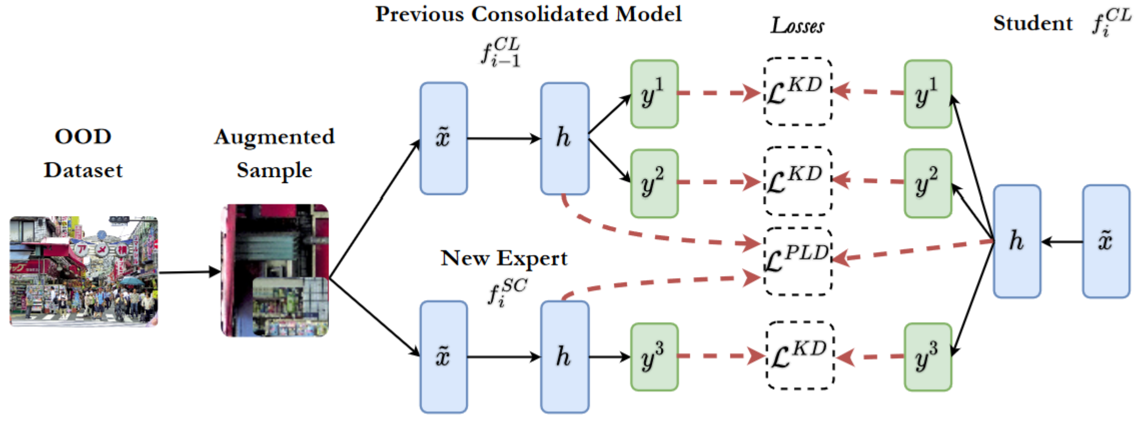
Data-Agnostic Consolidation (DAC)

Split learning into:

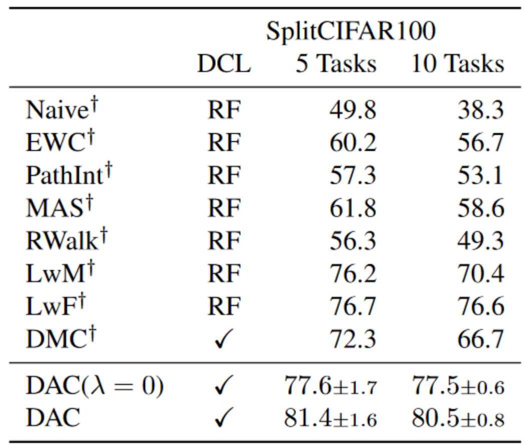
- *Adaptation*: learn new task
- *Consolidation*: aggregate models

Model consolidation with data-free knowledge distillation (DAC)

- Double Knowledge Distillation
  - Teachers: Previous CL model and New model
  - On the output
  - On the latent activations (Projected)
- Task-incremental method
- Surprisingly, **indipendent adaptation + sequential consolidation seems better than sequential adaptation** (i.e. what most CL methods are doing)



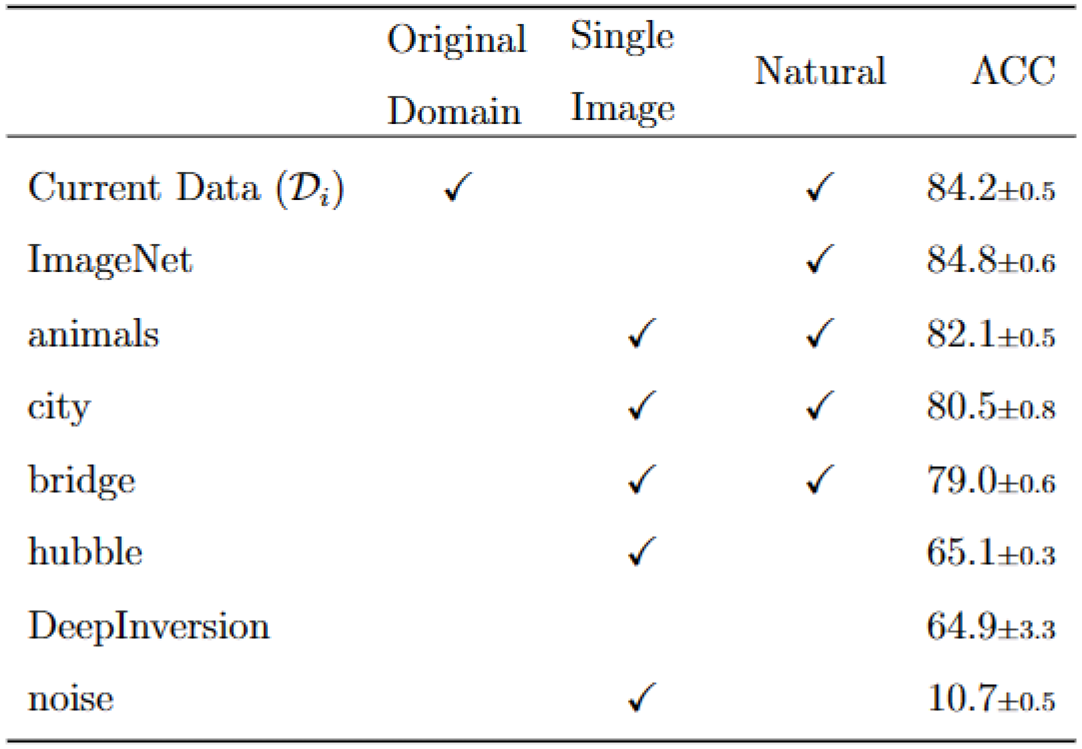
(a) Task-incremental SplitCIFAR100 after task 5 and 10. Baselines denoted by † are taken from (Masana et al., 2022)



		SplitCIFAR100	
	DCL	5 Tasks	10 Tasks
Naive†	RF	49.8	38.3
EWC†	RF	60.2	56.7
PathInt†	RF	57.3	53.1
MAS†	RF	61.8	58.6
RWalk†	RF	56.3	49.3
LwM†	RF	76.2	70.4
LwF†	RF	76.7	76.6
DMC†	✓	72.3	66.7
DAC($\lambda = 0$)	✓	77.6±1.7	77.5±0.6
DAC	✓	81.4±1.6	80.5±0.8

A single image is enough

Table 3: Test accuracy of DAC with different data sources on SplitCIFAR100 (10 Tasks).

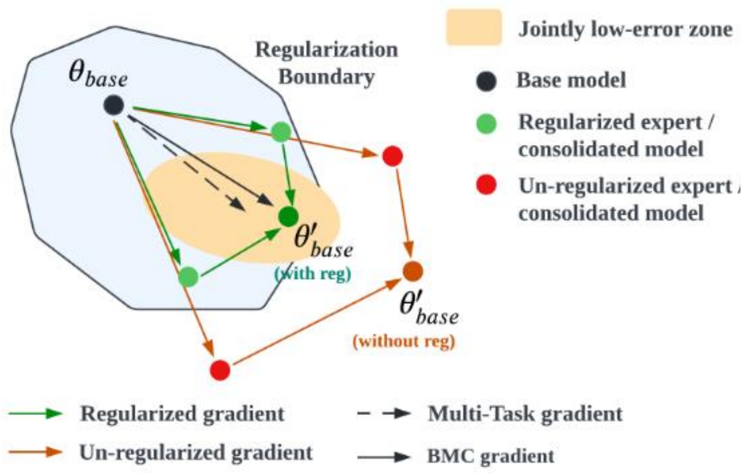


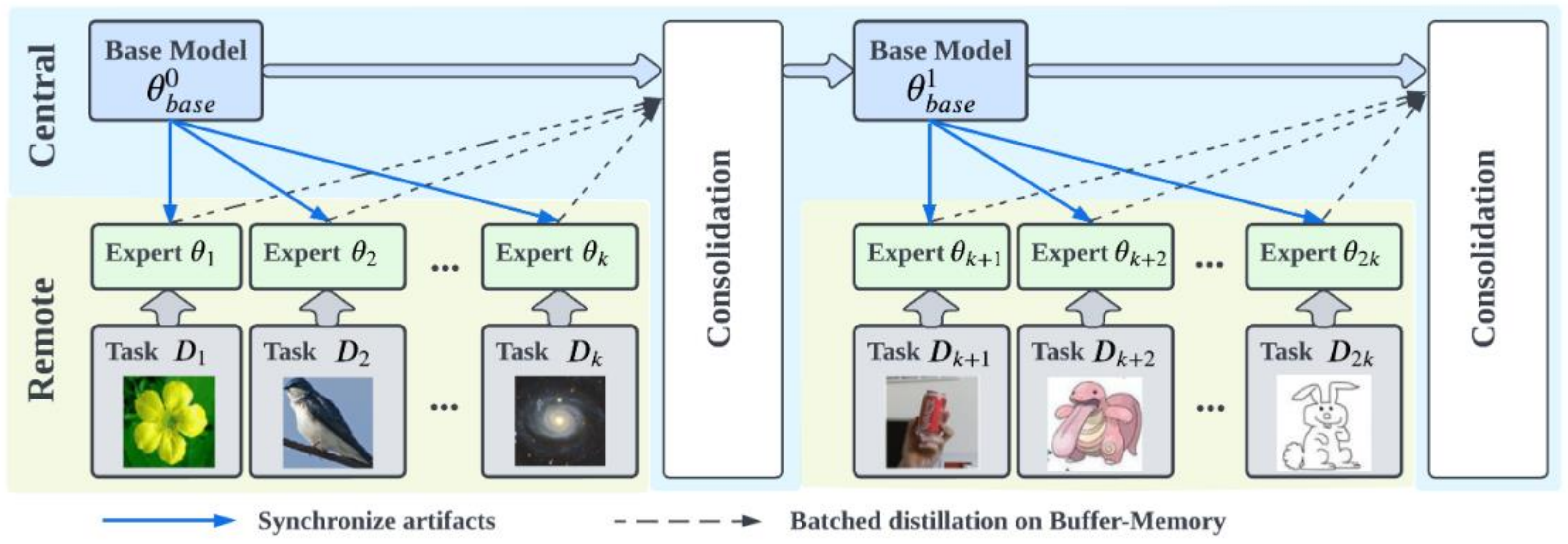
	Original Domain	Single Image	Natural	ACC
Current Data ($\mathcal{D}_i$)	✓		✓	84.2±0.5
ImageNet			✓	84.8±0.6
animals		✓	✓	82.1±0.5
city		✓	✓	80.5±0.8
bridge		✓	✓	79.0±0.6
hubble		✓		65.1±0.3
DeepInversion				64.9±3.3
noise		✓		10.7±0.5

Batch Model Consolidation

BMC:
- Regularized adaptation with distillation on the latent activations (teacher: base model)
- Replay data for batch consolidation

Sparse consolidation allows asynchronous learning in independent agents with light synchronization





I. Fostiropoulos et al. "Batch Model Consolidation: A Multi-Task Model Consolidation Framework." CVPR '23

Conclusion

Opportunities

Training Decentralized CL agents is a more general paradigm than federated learning:

Federated Learning requires **frequent sync** (large bandwidth) and a **shared single stakeholder training protocol**

- assumes **homogeneity in the model architecture**
- **not able to handle non-stationarity**
- Federated Learning can be seen a **constrained version of Ex-Model Continual Learning**
- opening the path for **Knowledge Sharing** between agents.