Transfer and Continual Learning

AI602: Recent Advances in Deep Learning

Lecture 9

Slide made by

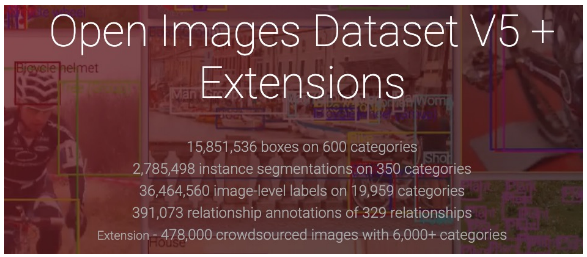
Sihyun Yu, Hyuntak Cha, Hankook Lee, Sangwoo Mo, Jongjin Park, Hyunwoo Kang

KAIST Graduate School of AI

Table of Contents

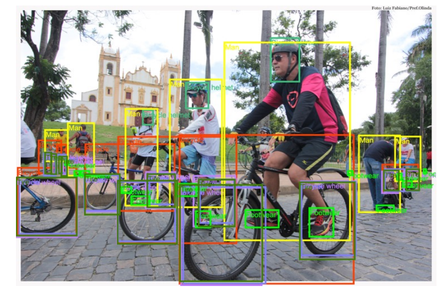
Table of Contents

- **Deep learning suffers from a lack of training samples**
  - Deep learning shows remarkable success in various fields of artificial intelligence (e.g., object classification, machine translation)
  - But, use (VERY) large labeled dataset



Open Images Dataset (9M images)

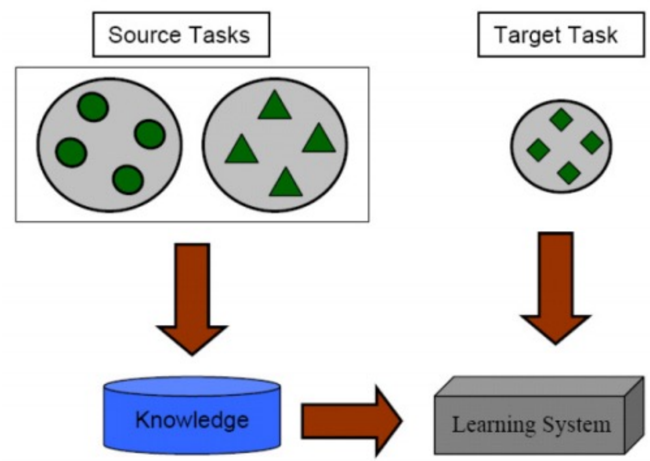
English Wikipedia (2.5B words)



>50 bounding boxes in an image

- **Collecting some annotations is too hard/expensive**
  - E.g., segmentation labels, bounding boxes, medical data
  - For a new task, only few samples are available

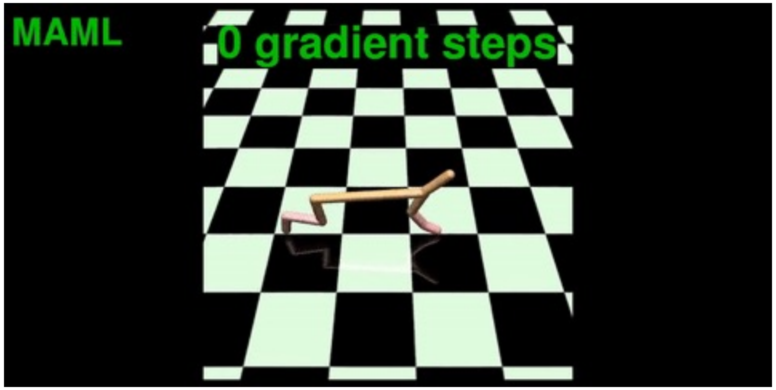
- ***Transfer learning*** aims to **transfer the knowledge** from **source** to **target domains & tasks**



* source: https://storage.googleapis.com/openimages/web/index.html , [Pan et al., 2010]

Robots learns skills and transfers that knowledge to other robots have different kinematics

Speech recognition: Learn from specific languages/accents transfer to learn different languages/accents



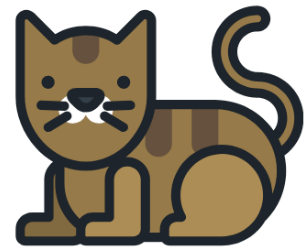
Simulated robots learn new movements from get transfer from previous learned task
(Top): from forward movements, learn backward move
(Bottom): learn faster movements from slow movements

* source: https://www.youtube.com/watch?v=_t1c_lrEH1k

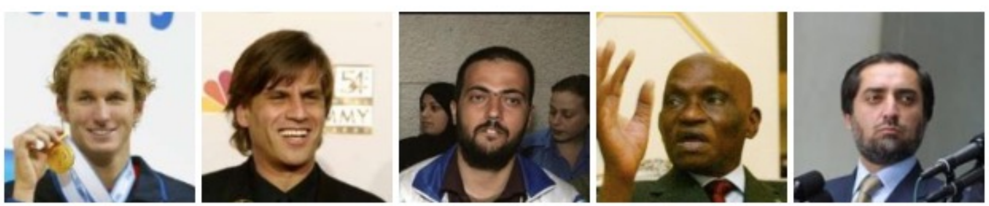
Domains & Tasks

- **Domain** $\mathcal{D} = \{\mathcal{X}, P(X)\}$
  - With a feature space $\mathcal{X}$ and a marginal probability distribution $P(X)$ for $X \in \mathcal{X}$
  - E.g., $\mathcal{X}$ is natural or cartoon image spaces / $P(X)$ is dog or cat distribution






- **Task** $\mathcal{T} = \{\mathcal{Y}, P(Y|X)\}$
  - With a label space $\mathcal{Y}$ and a conditional probability distribution $P(Y|X)$ for $Y \in \mathcal{Y}$
  - E.g., $\mathcal{Y}$ is digit (0, 1, …) or animal (dog, cat, …) spaces

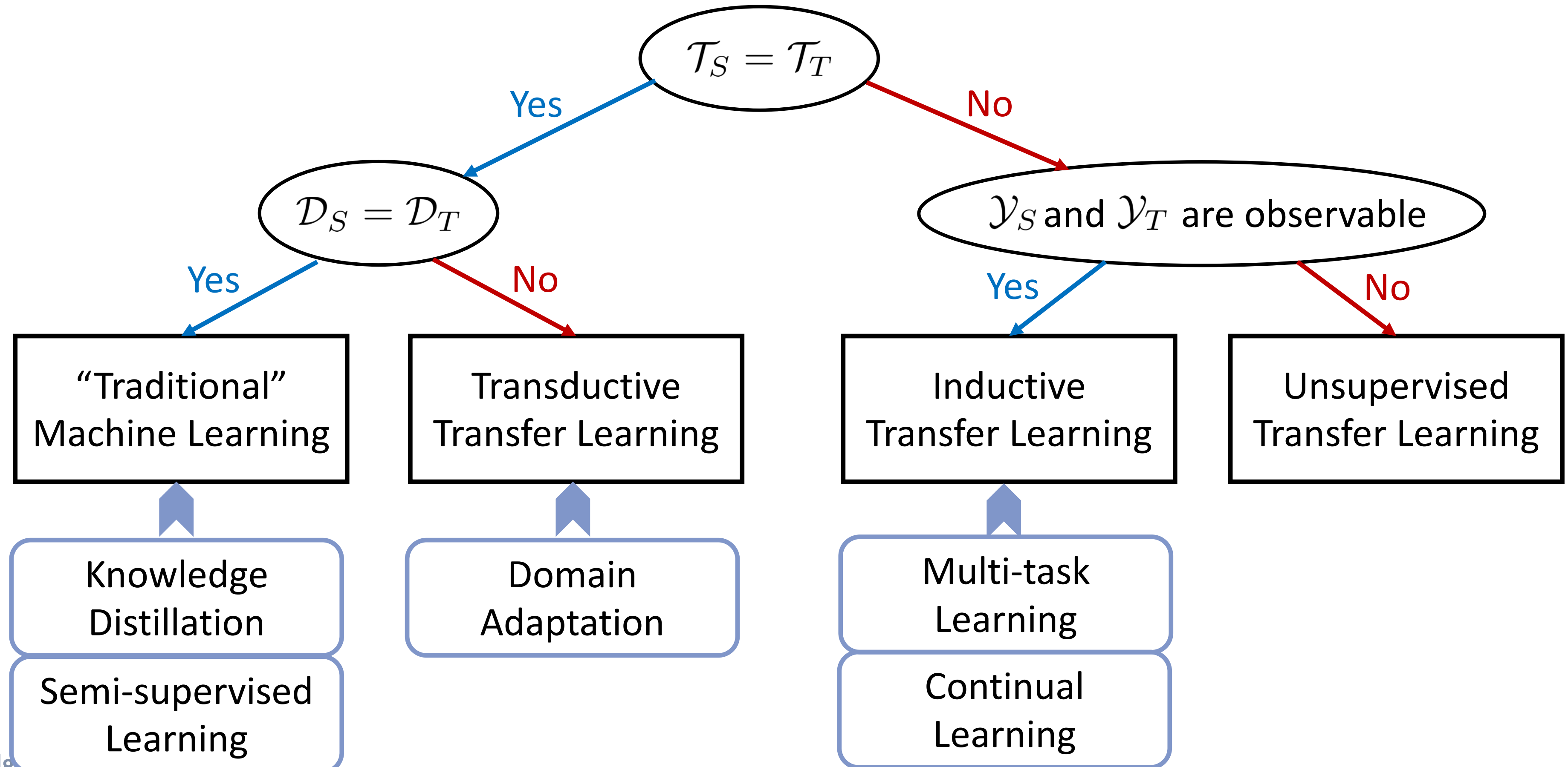


Age (e.g., 31, 49, 34, 50, 31)

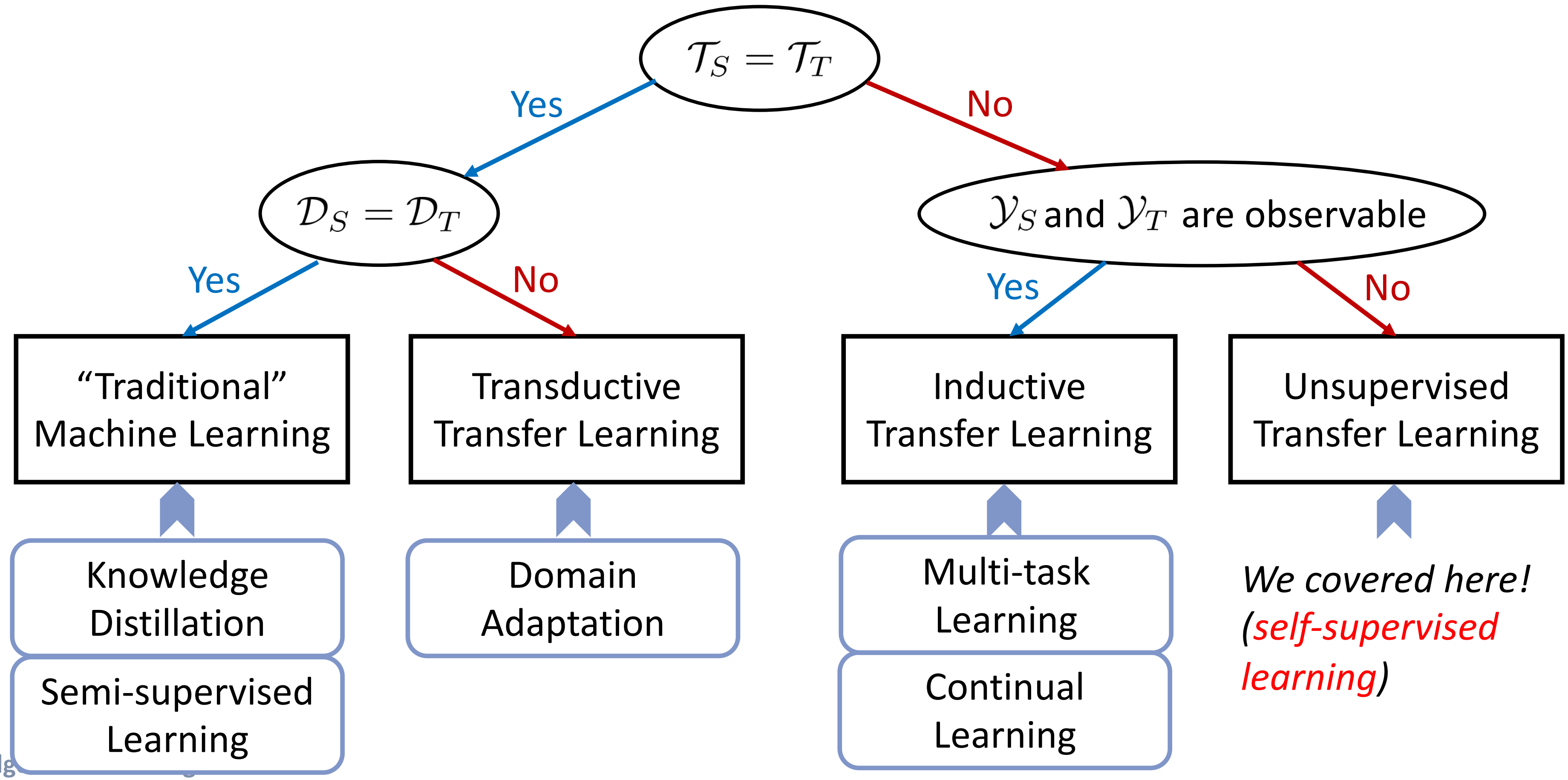
Person recognition
(e.g., John, Aaron, Adam, Will, John)

What is Transfer Learning?

- Definition of transfer learning [Pan et al., 2010]
  - Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$
  - ***Transfer learning*** aims to **improve** the learning of **the target predictive function** $f_T(\cdot)$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$

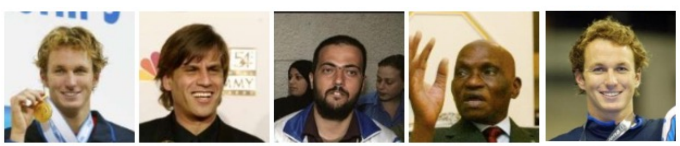


- Definition of transfer learning [Pan et al., 2010]
  - Given a source domain $\mathcal{D}_S$ and learning task $\mathcal{T}_S$, and a target domain $\mathcal{D}_T$ and learning task $\mathcal{T}_T$
  - ***Transfer learning*** aims to **improve** the learning of **the target predictive function** $f_T(\cdot)$ using the knowledge in $\mathcal{D}_S$ and $\mathcal{T}_S$ where $\mathcal{D}_S \neq \mathcal{D}_T$ or $\mathcal{T}_S \neq \mathcal{T}_T$

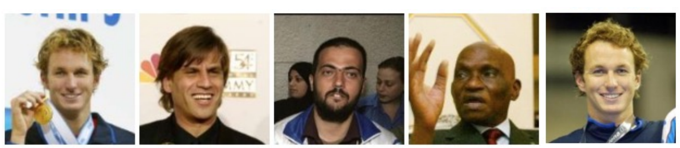


Type I: Same Tasks and Same Domain

- When tasks and domains are same, usually one can transfer knowledge for
  - Making target model that are smaller (model compression)
  - But, perform better than scratch learning
  - Using the knowledge transferred from the source model
  - ***Knowledge distillation***
    - Make a target model mimic the source model
      - Make outputs (or features) similar
      - Since tasks and domains are same, following a source/reference model is useful



Person recognition
(e.g., John, Aaron, Adam, Will, John)



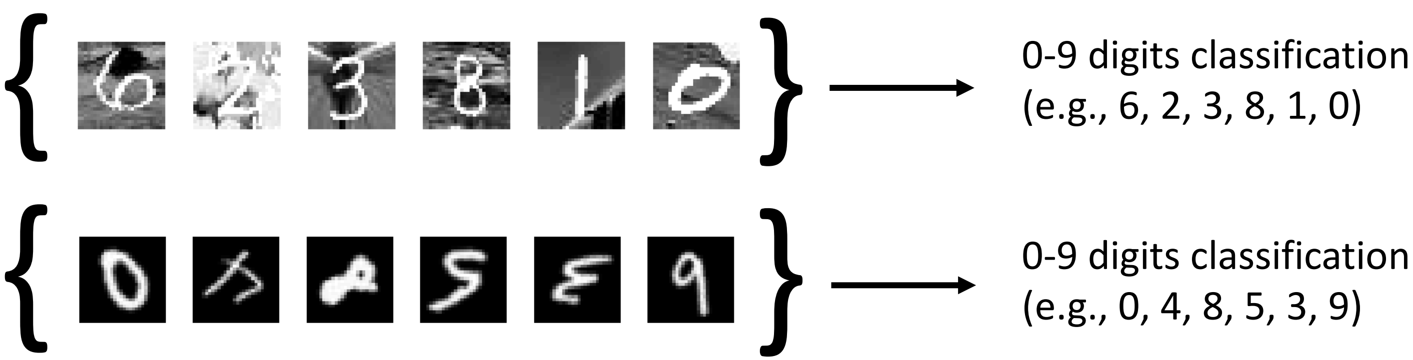
Person recognition
(e.g., John, Aaron, Adam, Will, John)

$\mathcal{D}_S = \mathcal{D}_T$

$\mathcal{T}_S = \mathcal{T}_T$

Type II: Same Tasks, but Different Domains (Transductive Transfer Learning)

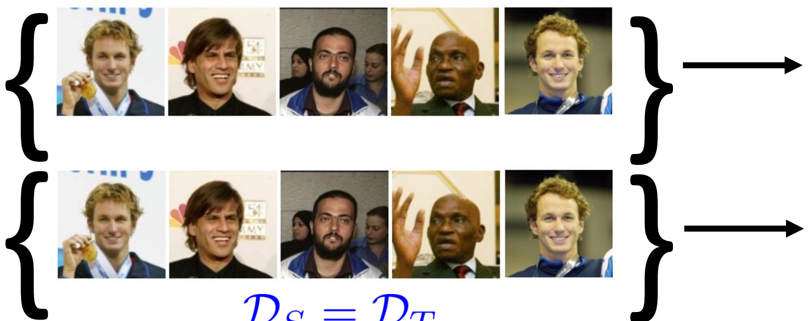
- Labels to predict are same but input data samples are different
  - Since tasks are same, by learning the features *invariant* to source and target domains, a target model can perform well
  - In many cases, target domain datasets do not have sufficient labels
    - By learning domain invariant features, source model's representations could be used for target domain
  - ***Domain adaptation***
    - Learn representations that confuse source and target domain inputs
    - Learn target representations that are similar to source domain



$\mathcal{D}_S \neq \mathcal{D}_T$ $\qquad$ $\mathcal{T}_S = \mathcal{T}_T$

Type III: Different Tasks (Inductive/Unsupervised Transfer Learning)

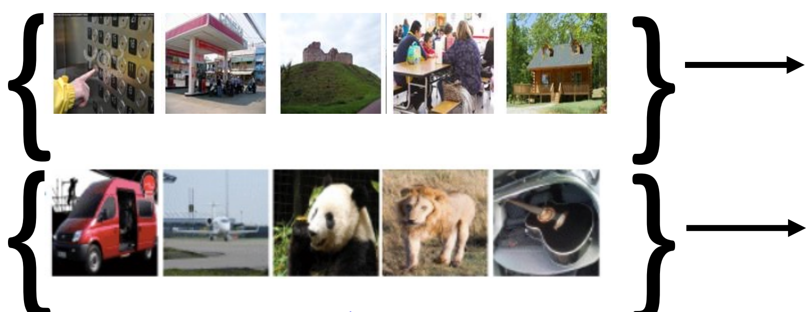
- Different tasks: different labels to predict
  - When tasks are different, feature extractors and output layers are need to be adjusted a lot for new tasks
  - ***Multi-task learning/fine-tuning*** are used to learn appropriate representations for target tasks from the source model's representations
  - ***Continual learning*** learns appropriate representations for target tasks without losing ones for past tasks.



Age (e.g., 31, 49, 34, 50, 31)

Person recognition
(e.g., John, Aaron, Adam, Will, John)

$\mathcal{D}_S = \mathcal{D}_T$ $\mathcal{T}_S \neq \mathcal{T}_T$



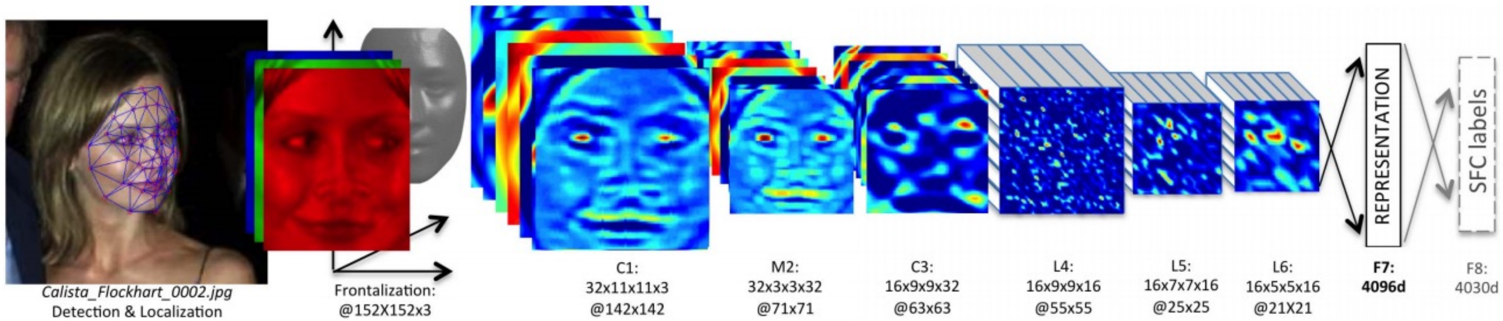
Scene classification
(e.g., elevator, gas station, castle, cafeteria, cabin)

Object classification
(e.g., car, airplane, panda, lion, guitar)

$\mathcal{D}_S \neq \mathcal{D}_T$ $\mathcal{T}_S \neq \mathcal{T}_T$

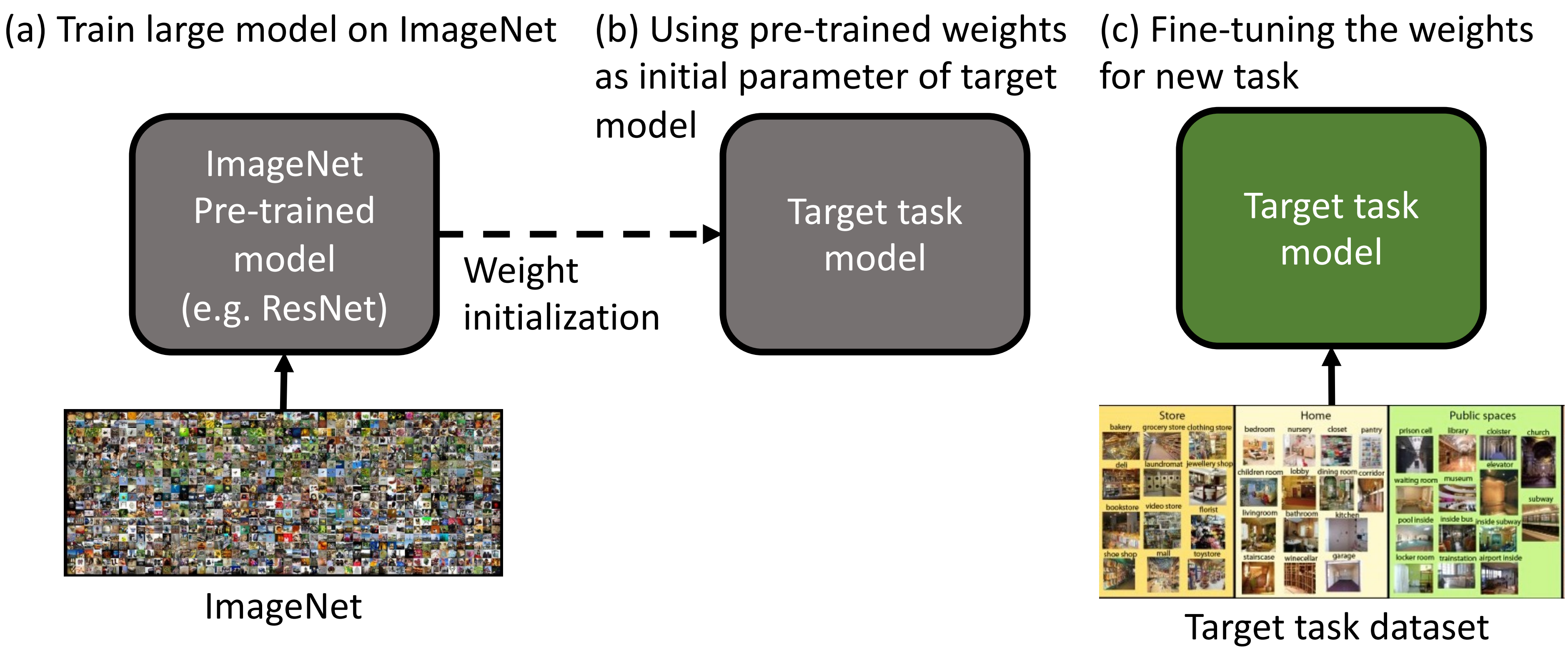
Table of Contents

- Convolutional layers are viewed as a feature extractor.
  - Lower convolutional layers capture low-level features. e.g. edges
  - Higher convolutional layers capture more complex, high-level features. e.g. eyes



- A source model pre-trained by a large dataset, e.g., ImageNet, is well-generalized, so one can expect it as a *good feature extractor or parameter initialization*.
  - To avoid overfitting, one can often *freeze* convolutional layers for small target datasets.
  - Can transfer to different domains and tasks
  - But, same architectures (at least for feature extraction part)

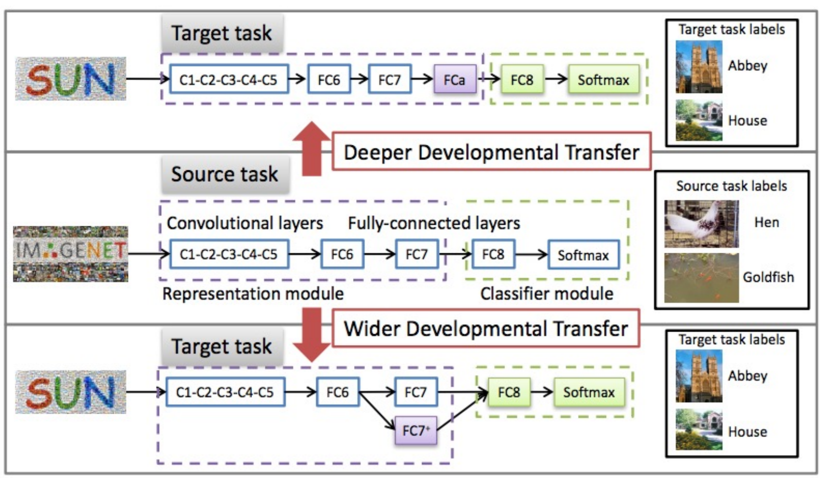
* source: [Yaniv et al., 2014]

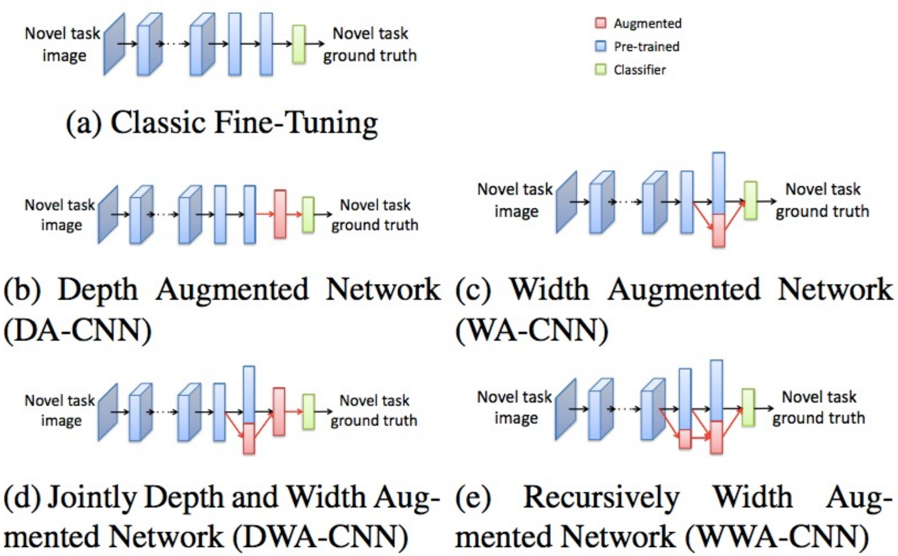


- Assumptions for fine-tuning approaches
  - **Features/Parameters** learned from some task are useful for **another tasks**
    - True in many artificial intelligence tasks (e.g. lower-level features of images such as edge)
- When do they **fail** to work
  - When **dataset** of source and target tasks are very **different**
  - When target tasks **have no (or very small) labeled training data**

Fine-Tuning with Increasing Target Model Capacity

- [Wang et al., 2017] increases the target model capacity in various ways
  - Channel-wise, depth-wise, (channel+depth)-wise
  - Using the pre-trained weights for all the layers except newly augmented layers/channels
  - Fine-tuning with target tasks
- Main idea at a high level
  - Using the pre-trained weight of source model to initialize the target model
  - Increase the capacity of target model in depth/channel-wise





* source: [Wang et al., 2017]

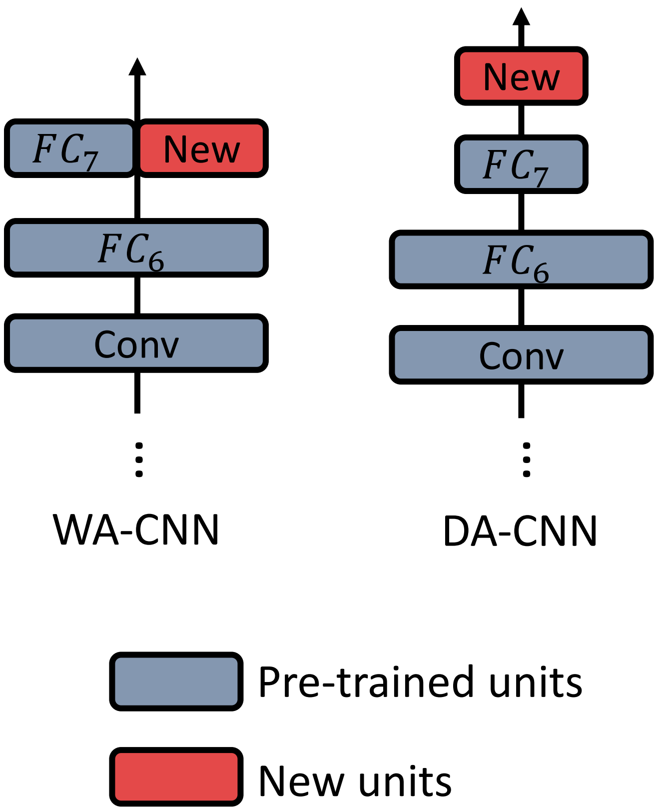
Experimental Results

- Evaluated on MIT-67, 102 Flowers, CUB200-2011, Stanford-40 with ImageNet pre-trained AlexNet
- Outperform most of task customized CNN or other multi-task learning methods
- Drawbacks:
  - Did not apply on architecture like ResNet (model without fully-connected layers)
  - Only augment the layers for fully-connected layers

Type	MIT-67		102 Flowers		CUB200-2011		Stanford-40	
	Approach	Acc(%)	Approach	Acc(%)	Approach	Acc(%)	Approach	Acc(%)
ImageNet CNNs	Finetuning-CNN	61.2	Finetuning-CNN	75.3	Finetuning-CNN	62.9	Finetuning-CNN	57.7
	Caffe [53]	59.5	CNN-SVM [32]	74.7	CNN-SVM [32]	53.3	Deep Standard [4]	58.9
	—	—	CNNaug-SVM [32]	86.8	CNNaug-SVM [32]	61.8	—	—
Task Customized CNNs	Caffe-DAG [53]	64.6	LSVM [30]	87.1	LSVM [30]	61.4	Deep Optimized [4]	66.4
	—	—	MsML+ [30]	89.5	DeCaf+DPD [7]	65.0	—	—
	Places-CNN [59]	**68.2**	MPP [55]	91.3	MsML+ [30]	66.6	—	—
	—	—	Deep Optimized [4]	91.3	MsML+* [30]	67.9	—	—
Data Augmented CNNs	Combined-AlexNet [18]	58.8	Combined-AlexNet [18]	83.3	—	—	Combined-AlexNet [18]	56.4
Multi-Task CNNs	Joint [22]	63.9	—	—	Joint [22]	56.6	—	—
	LwF [22]	64.5	—	—	LwF [22]	57.7	—	—
Ours	WA-CNN	66.3	WA-CNN	**92.8**	WA-CNN	**69.0**	WA-CNN	**67.5**

- Normalization and scaling activations are important for the performance improvement
  - Reconcile the learning pace of the new and pre-existing units
  - Normalization and scaling is more crucial in Width-augmented CNN (WA-CNN)
    - Without normalization and scaling, marginally better or worse than fine-tuning method

$$\hat{\boldsymbol{h}}^k = \underbrace{\gamma}_{\text{Scaling}} \boldsymbol{h}^k \underbrace{/ \left\| \boldsymbol{h}^k \right\|_2}_{\text{Normalization}}$$



Method	Scaling	New	FC_7-new	FC_6-new	All
Fine-tuning CNN	-	53.63	54.75	54.29	55.93
DA-CNN	w/o (rand)	**53.82**	**56.47**	56.25	57.21
	w/	53.51	56.15	**57.14**	**58.07**
WA-CNN	w/o (rand)	53.78	54.66	49.72	51.34
	w/o (copy+rand)	53.62	54.35	53.70	55.31
	w/	**56.81**	**56.99**	**57.84**	**58.95**

Performance on SUN-397 dataset by changing the fine-tuning layers from only new layer to all the layers

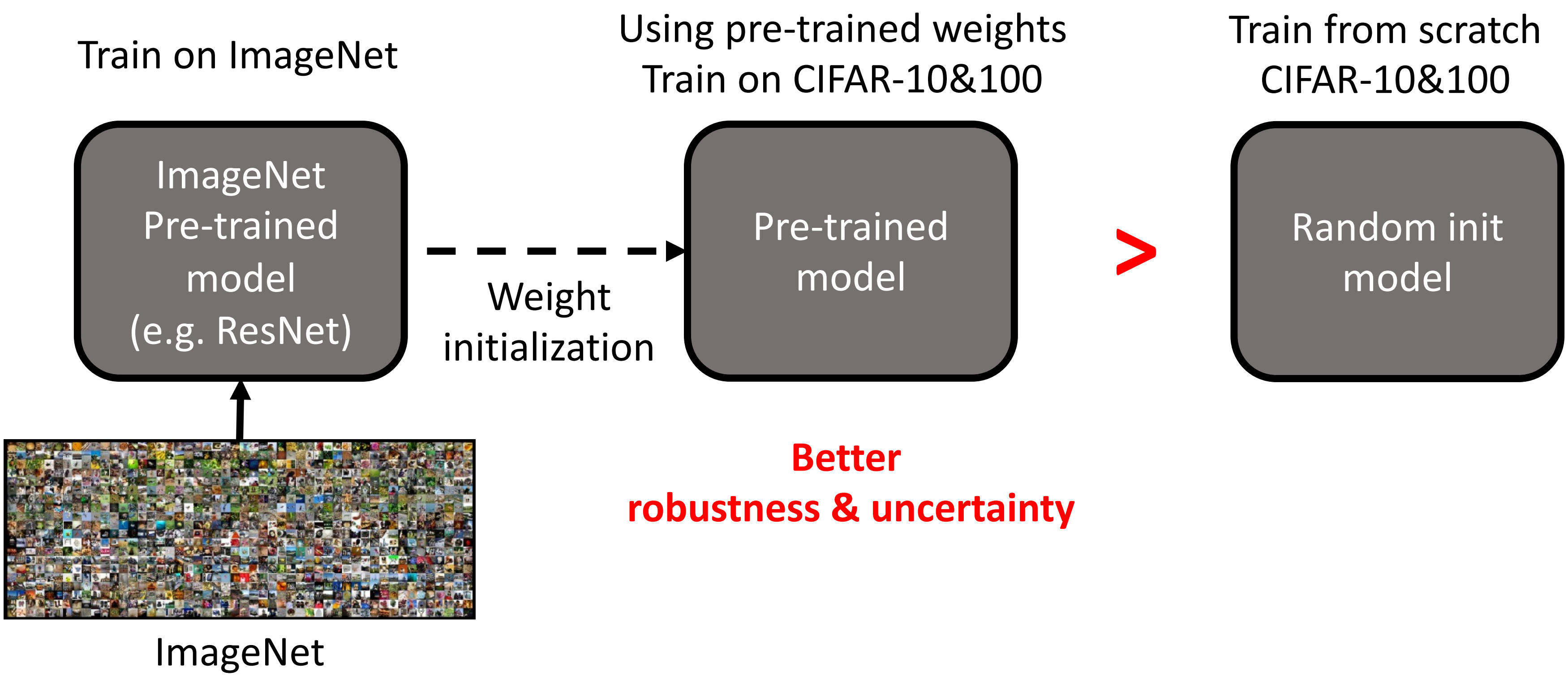
w/o (rand): new units are randomly initialized

w/o (copy+rand): initialize by copying FC_7, and add random noise

w/: with normalization and scaling

Using Pre-Training Can Improve Model Robustness and Uncertainty

- [Hendrycks et al., 2019] shows that re-training also improves other tasks such as **robustness and uncertainty**
- Considered various scenarios such as **label corruption, class imbalance, out-of-distribution detection**, etc.



Using Pre-Training Can Improve Model Robustness and Uncertainty

- **Label corruption:** when mis-labeled sample existed in train data

	CIFAR-10		CIFAR-100	
	Normal Training	Pre-Training	Normal Training	Pre-Training
No Correction	28.7	15.9	55.4	39.1
Forward Correction	25.5	15.7	52.6	42.8
GLC (5% Trusted)	14.0	7.2	46.8	33.7
GLC (10% Trusted)	11.5	6.4	38.9	28.4

- **Class imbalance**: when labels are imbalanced

Dataset	Imbalance Ratio / Method	0.2	0.4	0.6	0.8	1.0	1.5	2.0
		Total Test Error Rate / Minority Test Error Rate (%)						
CIFAR-10	Normal Training	23.7 / 26.0	21.8 / 26.5	21.1 / 25.8	20.3 / 24.7	20.0 / 24.5	18.3 / 23.1	15.8 / 20.2
	Cost Sensitive	22.6 / 24.9	21.8 / 26.2	21.1 / 25.7	20.2 / 24.3	20.2 / 24.6	18.1 / 22.9	16.0 / 20.1
	Oversampling	21.0 / 23.1	19.4 / 23.6	19.0 / 23.2	18.2 / 22.2	18.3 / 22.4	17.3 / 22.2	15.3 / 19.8
	SMOTE	19.7 / 21.7	19.7 / 24.0	19.2 / 23.4	19.2 / 23.4	18.1 / 22.1	17.2 / 22.1	15.7 / 20.4
	Pre-Training	8.0 / 8.8	7.9 / 9.5	7.6 / 9.2	8.0 / 9.7	7.4 / 9.1	7.4 / 9.5	7.2 / 9.4
CIFAR-100	Normal Training	69.7 / 72.0	66.6 / 70.5	63.2 / 69.2	58.7 / 65.1	57.2 / 64.4	50.2 / 59.7	47.0 / 57.1
	Cost Sensitive	67.6 / 70.6	66.5 / 70.4	62.2 / 68.1	60.5 / 66.9	57.1 / 64.0	50.6 / 59.6	46.5 / 56.7
	Oversampling	62.4 / 66.2	59.7 / 63.8	59.2 / 65.5	55.3 / 61.7	54.6 / 62.2	49.4 / 59.0	46.6 / 56.9
	SMOTE	57.4 / 61.0	56.2 / 60.3	54.4 / 60.2	52.8 / 59.7	51.3 / 58.4	48.5 / 57.9	45.8 / 56.3
	Pre-Training	37.8 / 41.8	36.9 / 41.3	36.2 / 41.7	36.4 / 42.3	34.9 / 41.5	34.0 / 41.9	33.5 / 42.2

- **Out-of-distribution detection**: detecting unseen samples in the test set

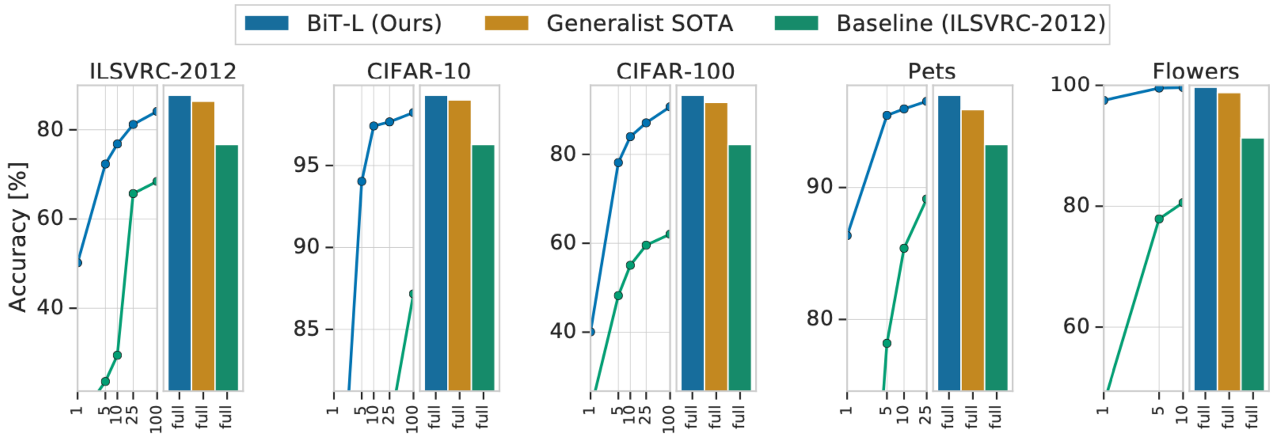
	AUROC		AUPR	
	Normal	Pre-Train	Normal	Pre-Train
CIFAR-10	91.5	94.5	63.4	73.5
CIFAR-100	69.4	83.1	29.7	52.7
Tiny ImageNet	71.8	73.9	30.8	31.0

* source: [Hendrycks et al., 2019]

Big Transfer (BiT): General Visual Representation Learning

- [Kolesnikov et al., 2019] shows that with a very large dataset, "general visual representation" can be learned
  - Authors pre-trained a classifier with JTF-300M dataset (or ImageNet-21K)
- Shows remarkable success on various dataset
  - Even with only a few label! (common failure case)
  - **Generalist SOTA:** pre-trained independently of the final task

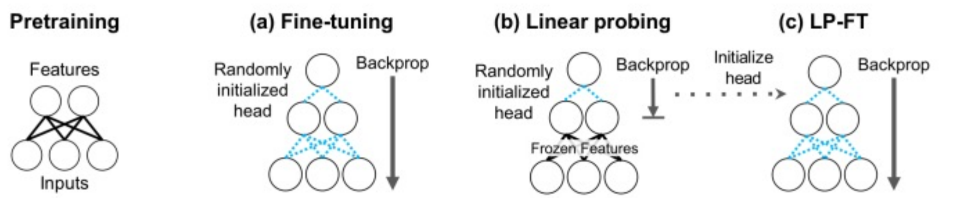
	BiT-L	Generalist SOTA
ILSVRC-2012	**87.54 ± 0.02**	86.4 [57]
CIFAR-10	**99.37 ± 0.06**	99.0 [19]
CIFAR-100	**93.51 ± 0.08**	91.7 [55]
Pets	**96.62 ± 0.23**	95.9 [19]
Flowers	**99.63 ± 0.03**	98.8 [55]
VTAB (19 tasks)	**76.29 ± 1.70**	70.5 [58]



# of labels

* source: [Kolesnikov, Beyer and Zhai et al.,2020]

- [Kumar et al., 2022] shows that fine-tuning underperforms out-of-distribution (OOD) when pretrained features are good and distribution shift is large
  - The authors show that linear probing then fine-tuning (LP-FT) can improve both in-distribution (ID) and OOD



	(a) Fine-tuning	(b) Linear probing	(c) LP-FT
ID test	85.1%	82.9%	**85.7%**
OOD test	59.3%	66.2%	**68.9%**

Average accuracies (10 distribution shifts)

- Tests CLIP pretrained ViT-B/16 on ImageNet and various ID and OOD settings:

Downstream ID	Corresponding OOD
”sketch” in DomainNet	“real”, “clipart”, and “painting” in DomainNet
Black bears and sloth bears in Living-17	Brown bears and polar bears in Living-17
CIFAR-10	STL-10 / CIFAR-10.1
ImageNet-1k	ImageNetV2, ImageNet-R, ImageNet-A, and ImageNet-Sketch

ID classification results

	CIFAR-10	Ent-30	Liv-17	DomainNet	FMoW	ImageNet	Average
FT	**97.3 (0.2)**	**93.6 (0.2)**	97.1 (0.2)	84.5 (0.6)	**56.5 (0.3)**	**81.7 (-)**	85.1
LP	91.8 (0.0)	90.6 (0.2)	96.5 (0.2)	89.4 (0.1)	49.1 (0.0)	79.7 (-)	82.9
LP-FT	**97.5 (0.1)**	**93.7 (0.1)**	**97.8 (0.2)**	**91.6 (0.0)**	51.8 (0.2)	**81.7 (-)**	**85.7**

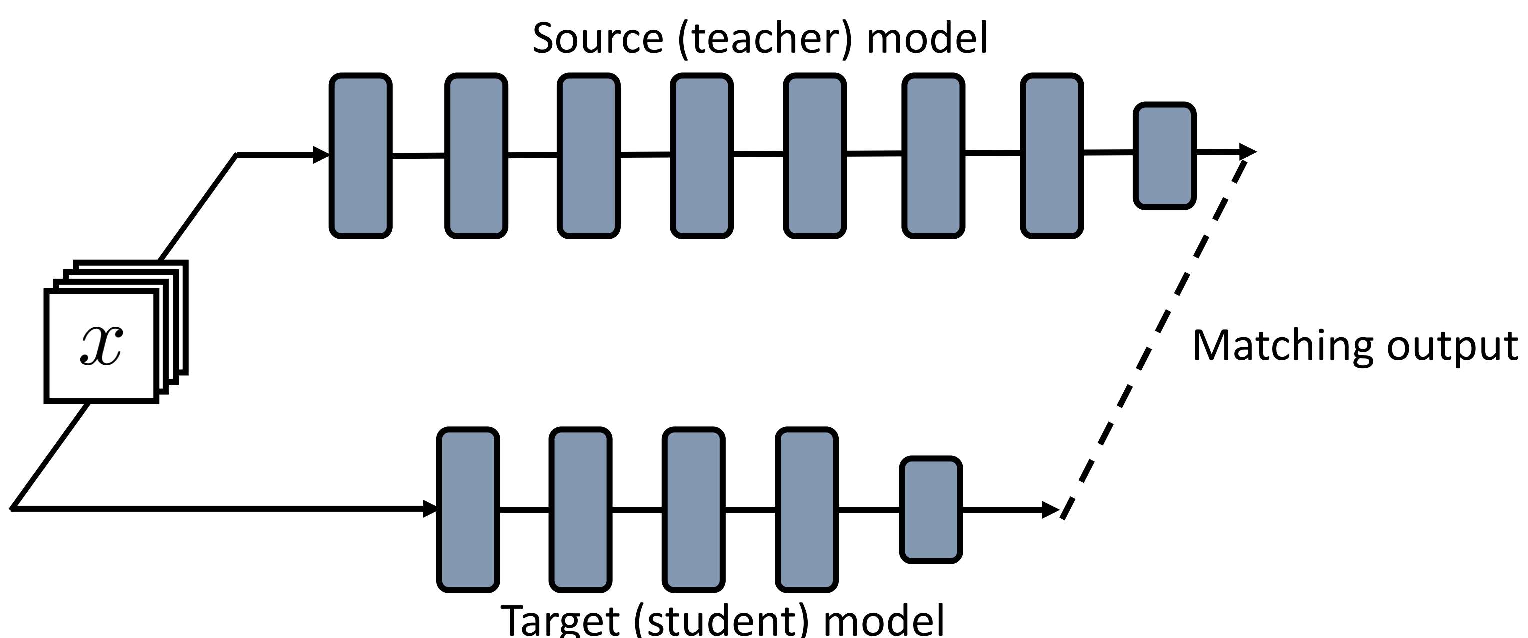
OOD classification results

	STL	CIFAR-10.1	Ent-30	Liv-17	DomainNet	FMoW
FT	82.4 (0.4)	92.3 (0.4)	60.7 (0.2)	77.8 (0.7)	55.5 (2.2)	32.0 (3.5)
LP	85.1 (0.2)	82.7 (0.2)	**63.2 (1.3)**	82.2 (0.2)	79.7 (0.6)	**36.6 (0.0)**
LP-FT	**90.7 (0.3)**	**93.5 (0.1)**	**62.3 (0.9)**	**82.6 (0.3)**	**80.7 (0.9)**	**36.8 (1.3)**

	ImNetV2	ImNet-R	ImNet-Sk	ImNet-A	Average
FT	**71.5 (-)**	52.4 (-)	40.5 (-)	27.8 (-)	59.3
LP	69.7 (-)	70.6 (-)	46.4 (-)	45.7 (-)	66.2
LP-FT	**71.6 (-)**	**72.9 (-)**	**48.4 (-)**	**49.1 (-)**	**68.9**

Table of Contents

- Learn a source model and distill its knowledge to a target model
  - Can lead to a better model with small architecture, or faster training
- Given a teacher network on domain $\mathcal{D}$, enhance the training of (usually **smaller**) a student network on ***same*** domain $\mathcal{D}$, using knowledge of a teacher network
- Done by **matching the output** of source and target models
  - Design a new loss term (e.g., MSE loss, KL divergence) for making source and target outputs similar in addition to the original loss term (e.g., cross entropy loss)



Knowledge Distillation: Matching Output of Source and Target Model

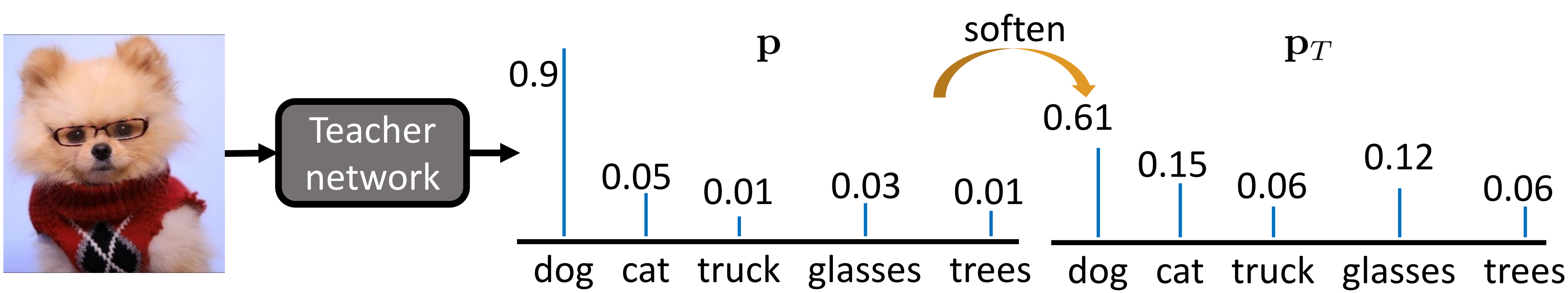
- [Hinton et al., 2015] proposes
  - Use temperature $T \geq 1$ to make a *softer* probability distribution over classes

$$q_{i,T} = \frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}$$

  where z_i, q_i are *the* i-th logit and probability, respectively
  - Use the *soft target* as additional labels to train student model

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{\text{ce}}(\mathbf{y}, \mathbf{q}) + \alpha T^2 \mathcal{L}_{\text{ce}}(\mathbf{p}_T, \mathbf{q}_T)$$

    where $\mathbf{y}$, $\mathbf{q}$ and $\mathbf{p}$ are ground-truth labels, target model outputs, and source model outputs, respectively. It is important to **multiply soft targets by T^2** because the magnitudes of the gradients produced by them scale as $1/T^2$. (derived in the next page)



Knowledge Distillation: Effect of Temperature Scaling

- Let C be a cross-entropy loss of softened labels.

$$C = \mathcal{L}_{\text{ce}}(\mathbf{p}_T, \mathbf{q}_T)$$

- The gradient of C, with respect to each target logit z_i, and source logit v_i :

$$\frac{\partial C}{\partial z_i} = \frac{1}{T}(q_i - p_i) = \frac{1}{T}\Bigg(\frac{\exp(z_i/T)}{\sum_j \exp(z_j/T)}) - \frac{\exp(v_i/T)}{\sum_j \exp(v_j/T)} \Bigg)$$

  - If the temperature is high compared with the magnitude of the logits,

$$\frac{\partial C}{\partial z_i} \approx \frac{1}{T}\left(\frac{1+z_i/T}{N+\sum_j z_j/T} - \frac{1+v_i/T}{N+\sum_j v_j/T} \right)$$

  - If we assume that the logits have been zero-meaned (i.e. $\sum_j z_j = \sum_j v_j = 0$)

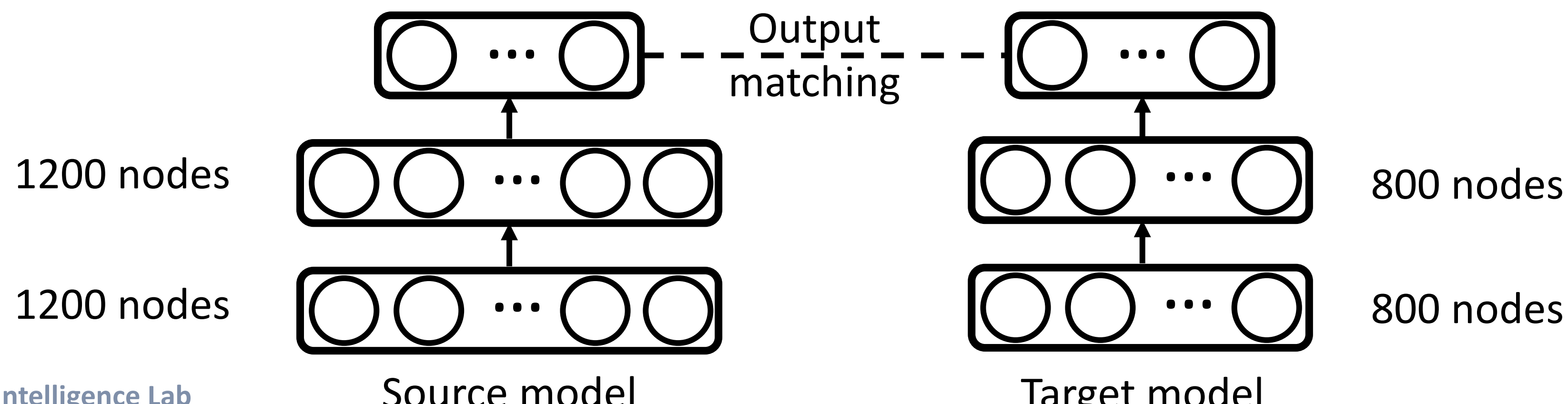
$$\frac{\partial C}{\partial z_i} \approx \frac{1}{NT^2}(z_i - v_i) = \underbrace{\boxed{\frac{1}{NT^2}}}_{\text{scaling}} \frac{\partial}{\partial z_i}\left(\boxed{\frac{1}{2}(z_i - v_i)^2}\right)$$

- **At high temperatures, the objective is equivalent to a quadratic function.**
  - **Distillation pays much more attention to logits that are negative than the average.**
  - This is potentially advantageous because these logits (which are not the correct label) are almost completely unconstrained by the classification loss.

Knowledge Distillation: Experimental Results

- MNIST experiments
  - Hand-written digits (28x28 grayscale images)
  - 60000 training, 10000 test images
  - Source model: 2 hidden layers MLP with 1200 hidden nodes
  - Target model: 2 hidden layers MLP with 800 hidden nodes

Model	Error rate (%)
Source model	0.67
Target model (without knowledge distillation)	1.46
Target model (with knowledge distillation, $T = 20$)	0.74

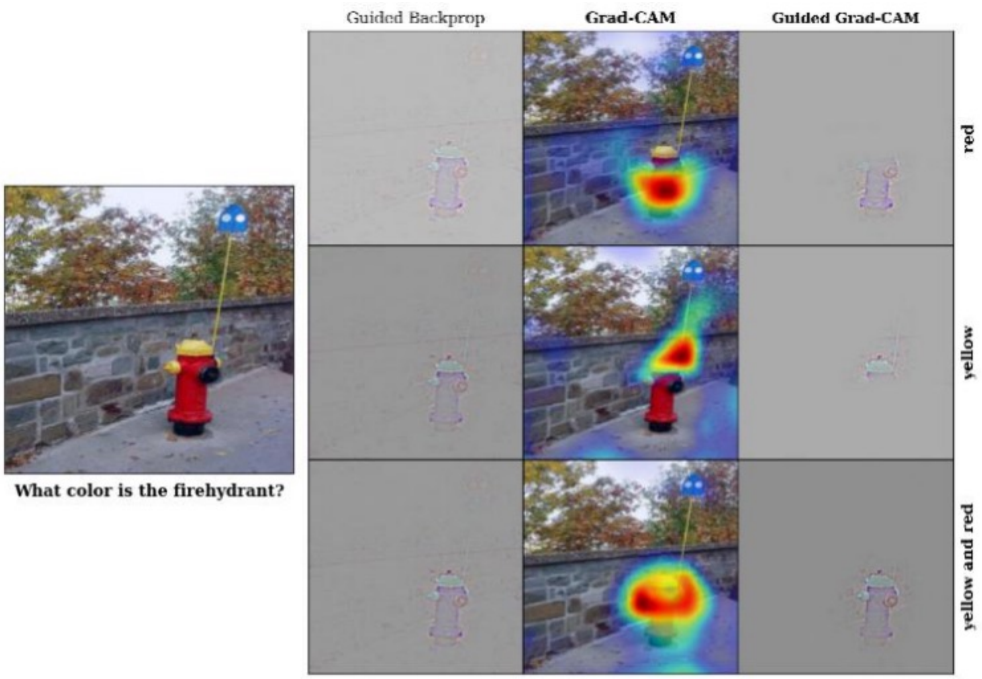


- Smaller target models get advantages by following larger source models

- Useful when target and source datasets/tasks are same
  - Performance may degrade when apply target dataset or task are changed

- **Main challenges:** what, when, and where to transfer
  - Decide the **form** of transferring knowledge
  - Decide **when** does transfer helps
  - Decide **which level** representations (layers) to transfer

- Visualizing **attention maps** in deep CNN is an open problem.
- Recently, a number of methods was proposed to improve attention maps.
  - e.g. Guided backpropagation [Springenberg et al., 2015], Grad-CAM [Selvaraju et al., 2016].
- In CNN models, the attention maps produced by intermediate features can be transferable knowledge.

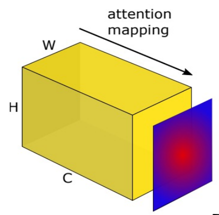


Visualization of VQA model.

* source: [Selvaraju et al., 2016]

- [Zagoruyko et al. 2017] matches the attention of intermediate features
  - Make a 2D attention map from feature activations with attention mapping function F

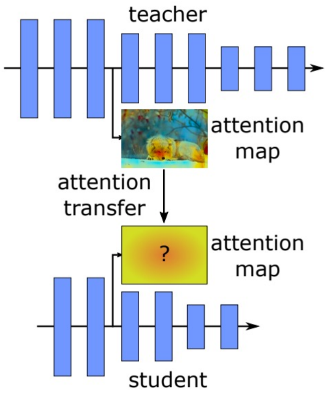
$$F(A_{h,w}) = \sum_{c=1}^{C} |A_{c,h,w}|^p$$

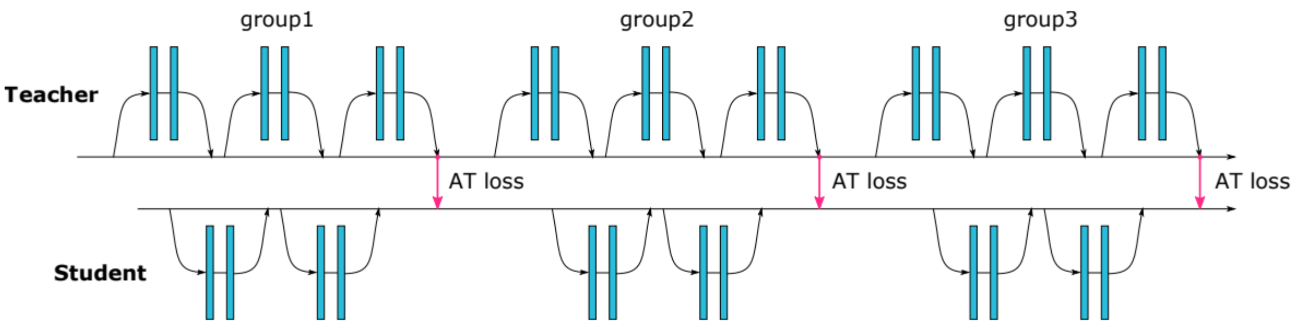


  - $p > 1$, feature activation $A_{c,h,w} \in \mathbb{R}^{C\times H\times W}$ (C channels, spatial size $H \times W$)
  - Train the original loss with the attention map matching regularization term

$$\mathcal{L}_{\text{at}}(\theta|\mathcal{D}) = \mathcal{L}_{\text{org}}(\theta|\mathcal{D}) + \frac{\beta}{2}\sum_{j\in\mathcal{I}} \left\| \frac{Q_{\mathcal{T}}^{j}(\theta,x)}{\|Q_{\mathcal{T}}^{j}(\theta,x)\|_2} - \frac{Q_{\mathcal{S}}^{j}(\theta,x)}{\|Q_{\mathcal{S}}^{j}(\theta,x)\|_2} \right\|_p$$

where $Q_T^j = vec(F(A_T^j))$ and $Q_S^j = vec(F(A_S^j))$ are respectively the j-th pair of target (student) and source (teacher) attention maps.





* source: [Zagoryuko et al., 2017]

- Attention transfer works better than original distillation methods or they can be used together
  - Hyper-parametric choices:
    - Choose proper attention mapping function
    - Layers to transfer the attention map

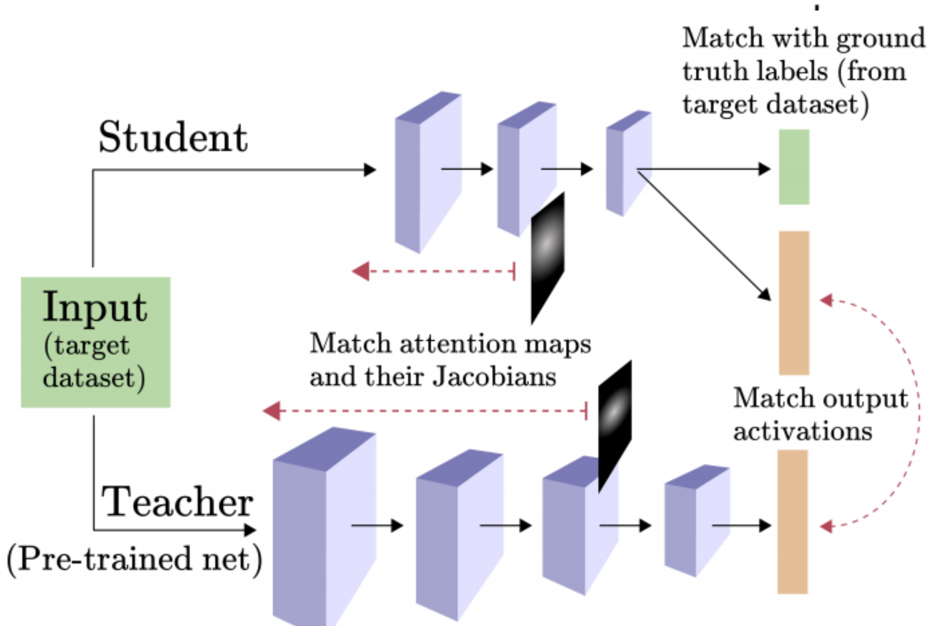
student	teacher	student	AT	F-ActT	KD	AT+KD	teacher
NIN-thin, 0.2M	NIN-wide, 1M	9.38	8.93	9.05	8.55	8.33	7.28
WRN-16-1, 0.2M	WRN-16-2, 0.7M	8.77	7.93	8.51	7.41	7.51	6.31
WRN-16-1, 0.2M	WRN-40-1, 0.6M	8.77	8.25	8.62	8.39	8.01	6.58
WRN-16-2, 0.7M	WRN-40-2, 2.2M	6.31	5.85	6.24	6.08	5.71	5.23

CIFAR-10 experiments. **AT**: attention transfer, **F-ActT**: full activation transfer, **KD**: knowledge distillation **AT+KD**: applying AT and KD at the same time. AT+KD is best in most cases (for student networks)

type	model	ImageNet→CUB	ImageNet→Scenes
student	ResNet-18	28.5	28.2
KD	ResNet-18	27 (-1.5)	28.1 (-0.1)
AT	ResNet-18	27 (-1.5)	27.1 (-1.1)
teacher	ResNet-34	26.5	26

Large-scale experiments. Using ImageNet pre-trained model, fine-tune source model with target dataset. Then, transfer to student model learning same target task.

- [Srinivas et al., 2018] proposes several Jacobian-based regularizations
  - Sobolev training [Czarnecki et al., 2017] demonstrated that using higher order (typically 1st order) derivatives along with the targets can help training.
  - [Srinivas et al., 2018] showed that matching Jacobians is a special case of previous distillation methods, when noise is added to the inputs.
- They added a new branch for distillation, and matched the output activations, attention maps, and their Jacobians (for the largest value of an attention map).



$$\mathcal{L}_{\mathrm{kd}} = (1-\alpha)\mathcal{L}_{\mathrm{ce}}(\mathbf{y},\mathbf{q}) + \alpha\mathcal{L}_{\mathrm{ce}}(\mathbf{p}_T,\mathbf{q}_T)$$

$$\mathcal{L}_{\mathrm{at}} = \sum_{j\in\mathcal{I}} \left\| \frac{Q_{\mathcal{T}}^{j}(\theta,x)}{\left\|Q_{\mathcal{T}}^{j}(\theta,x)\right\|_2} - \frac{Q_{\mathcal{S}}^{j}(\theta,x)}{\left\|Q_{\mathcal{S}}^{j}(\theta,x)\right\|_2} \right\|_p$$

$$\mathcal{L}_{\mathrm{Jac}} = \sum_{j\in\mathcal{I}} \left\| \frac{\nabla_x f^j(x)}{\|\nabla_x f^j(x)\|_2} - \frac{\nabla_x g^j(x)}{\|\nabla_x g^j(x)\|_2} \right\|_p$$

where f^j, g^j are max points of j-th attention maps of target and source model, respectively.

wise linear function;

The correct hypothesis is enriched with derivative information.

* source: [Czarnecki et al., 2017], [Srinivas et al., 2018]

- Matching Jacobians improves distillation performance in small data.

Distillation performance on the CIFAR100 dataset

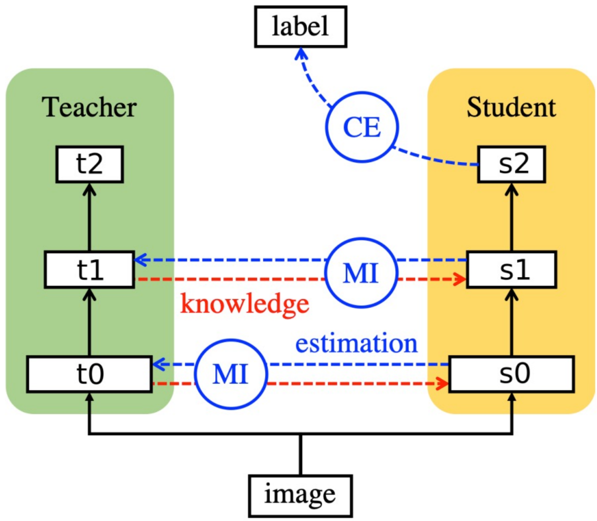
# of Data points per class →	1	5	10	50	100	500 (full)
Cross-Entropy (CE) training	5.69	13.9	20.03	37.6	44.92	54.28
CE + match activations	12.13	26.97	33.92	46.47	50.92	**56.65**
CE + match Jacobians	6.78	23.94	32.03	45.71	51.47	53.44
CE + match {activations + Jacobians}	**13.78**	**33.39**	**39.55**	**49.49**	**52.43**	54.57
Match activations only	10.73	28.56	33.6	45.73	50.15	56.59
Match {activations + Jacobians}	13.09	33.31	38.16	47.79	50.06	51.33

- Matching Jacobians improves performance of all case of transfer learning.
- None of the methods match the oracle performance of pre-trained model.

Transfer performance from Imagenet to MIT Scenes dataset

# of Data points per class →	5	10	25	50	Full
Cross-Entropy (CE) training on untrained student network	11.64	20.30	35.19	46.38	59.33
CE on pre-trained student network (Oracle)	**25.93**	**43.81**	**57.65**	**64.18**	**71.42**
CE + match activations (Li & Hoiem, 2016)	17.08	27.13	45.08	55.22	65.22
CE + match {activations + Jacobians}	17.88	28.25	45.26	56.49	66.04
CE + match {activations + attention} (Zagoruyko & Komodakis, 2017)	16.53	28.35	46.01	57.80	**67.24**
CE + match {activations + attention + Jacobians}	**18.02**	**29.25**	**47.31**	**58.35**	**67.31**

- [Ahn et al., 2019] **maximizes mutual information** between source/target models
  - Use the **variational information maximization** [Barber et al., 2003]
  - Instead of matching a specific form of feature representations



variational information maximization

$$\begin{aligned} I(t;s) &= H(t) - H(t|s) \\ &= H(t) + \mathbb{E}_{t,s}[\log p(t|s)] \\ &= H(t) + \mathbf{E}_{t,s}[\log q(t|s)] + \mathbf{E}_s[D_{\mathrm{KL}}(p(t|s)||q(t|s))] \\ &\geq H(t) + \mathbf{E}_{t,s}[\log q(t|s)] \end{aligned}$$

- Use a Gaussian distribution for modeling $q(t|s)$ with heteroscedastic mean $\mu(s)$ and homoscedastic variance $\sigma(s)$

$$-\log q(t|s) = \sum_{c,h,w} \log \sigma_c + \frac{(t_{c,h,w} - \mu_{c,h,w}(s))^2}{2\sigma_c^2} + \text{constant}$$

* source: [Ahn et al., 2019]

- Apply **Variational Information Distillation** (VID) to different locations
  - **VID-I**: between intermediate layers of teacher/student networks
  - **VID-LP**: between penultimate layers of teacher/student networks

Knowledge Distillation on CIFAR-10

M	5000	1000	500	100
Teacher	94.26	-	-	-
Student	90.72	84.67	79.63	58.84
KD	91.27	86.11	82.23	64.24
FitNet	90.64	84.78	80.73	68.90
AT	91.60	87.26	84.94	73.40
NST	91.16	86.55	82.61	64.53
VID-I	**91.85**	**89.73**	**88.09**	**81.59**
KD + AT	91.81	87.34	85.01	76.29
KD + VID-I	91.7	88.59	86.53	78.48

Transfer learning from ImageNet to CUB200

M	≈29.95	20	10	5
Student	37.22	24.33	12.00	7.09
fine-tuning	76.69	71.00	59.25	44.07
LwF	55.18	42.13	26.23	14.27
FitNet	66.63	56.63	46.68	31.04
AT	54.62	41.44	28.90	16.55
NST	55.01	41.87	23.76	15.63
VID-LP	65.59	54.12	39.20	27.86
VID-I	**73.25**	**67.20**	**56.86**	**46.21**
LwF + FitNet	68.69	58.81	48.86	31.30
VID-LP + VID-I	69.71	63.94	52.87	41.12

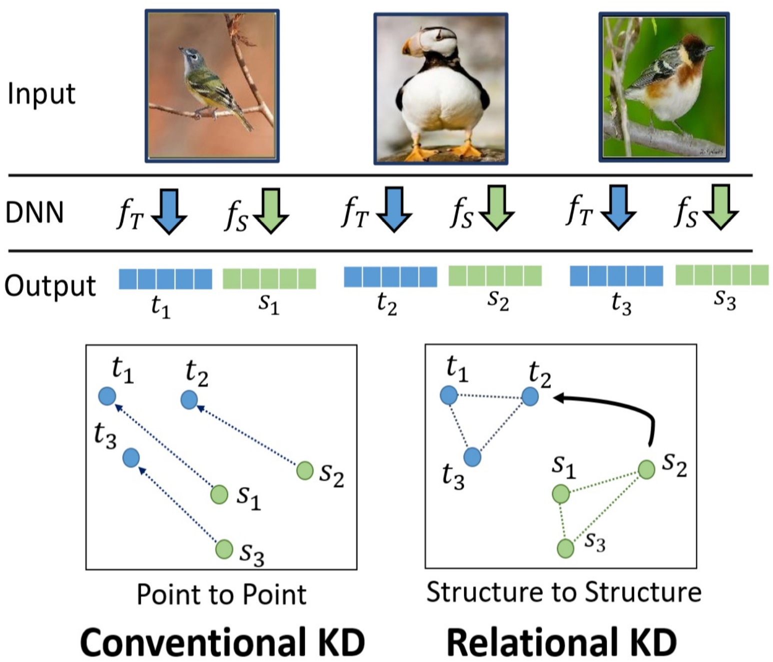
- **VID** can be applied between CNNs/MLPs
  - VID achieves state-of-the-art performance compared to other MLPs on CIFAR-10

Network	MLP-4096	MLP-2048	MLP-1024
Student	70.60	70.78	70.90
KD	70.42	70.53	70.79
FitNet	76.02	74.08	72.91
VID-I	**85.18**	**83.47**	**78.57**
Urban *et al.* [27]		74.32	
Lin *et al.* [17]		78.62	

Relational Knowledge Distillation

- [Park et al., 2019] transfers the mutual **relations of data examples**
  - Knowledge distillation (KD) only mimic the output of individual data point

- Author considers two types of relations: **distance & angle**



Distance: L2 distance

$$\psi_{\mathrm{D}}(t_i, t_j) = \frac{1}{\mu} \|t_i - t_j\|_2 ,$$

$$\mathcal{L}_{\text{RKD-D}} = \sum_{(x_i,x_j)\in\mathcal{X}^2} l_\delta\big(\psi_{\mathrm{D}}(t_i,t_j), \psi_{\mathrm{D}}(s_i,s_j)\big),$$

Angle: Cosine similarity

$$\psi_{\mathrm{A}}(t_i, t_j, t_k) = \cos\angle t_i t_j t_k = \langle \mathbf{e}^{ij}, \mathbf{e}^{kj} \rangle$$

$$\text{where} \quad \mathbf{e}^{ij} = \frac{t_i - t_j}{\|t_i - t_j\|_2}, \mathbf{e}^{kj} = \frac{t_k - t_j}{\|t_k - t_j\|_2}.$$

$$\mathcal{L}_{\text{RKD-A}} = \sum_{(x_i,x_j,x_k)\in\mathcal{X}^3} l_\delta\big(\psi_{\mathrm{A}}(t_i,t_j,t_k), \psi_{\mathrm{A}}(s_i,s_j,s_k)\big),$$

l_δ: feature matching loss (Huber, L2 etc.)

* source: [Park et al., 2019]

- Apply three types of **relational knowledge distillation (RKD)**
  - **RKD-D**: only considers distance relationship
  - **RKD-A**: only considers angular relationship
  - **RKD-DA**: considers both, distance and angular relationship

	Baseline (Triplet [31])	FitNet [27]	Attention [47]	DarkRank [7]	Ours		
					RKD-D	RKD-A	RKD-DA
$\ell 2$ normalization	O	O	O	O	O / X	O / X	O / X
ResNet18-16	37.71	42.74	37.68	46.84	46.34 / 48.09	45.59 / **48.60**	45.76 / 48.14
ResNet18-32	44.62	48.60	45.37	53.53	52.68 / **55.72**	53.43 / 55.15	53.58 / 54.88
ResNet18-64	51.55	51.92	50.81	56.30	56.92 / 58.27	56.77 / 58.44	57.01 / **58.68**
ResNet18-128	53.92	54.52	55.03	57.17	58.31 / 60.31	58.41 / **60.92**	59.69 / 60.67
ResNet50-512	61.24						

Recall@1 on CUB-200 dataset. The teacher is ResNet50-512 (model-d refers dimension)

	CIFAR-100 [15]	Tiny ImageNet [46]
Baseline	71.26	54.45
RKD-D	72.27	54.97
RKD-DA	72.97	56.36
HKD [11]	74.26	57.65
HKD+RKD-DA	**74.66**	**58.15**
FitNet [27]	70.81	55.59
FitNet+RKD-DA	72.98	55.54
Attention [47]	72.68	55.51
Attention+RKD-DA	73.53	56.55
Teacher	77.76	61.55

Accuracy (%) on CIFAR-100 and Tiny ImageNet.

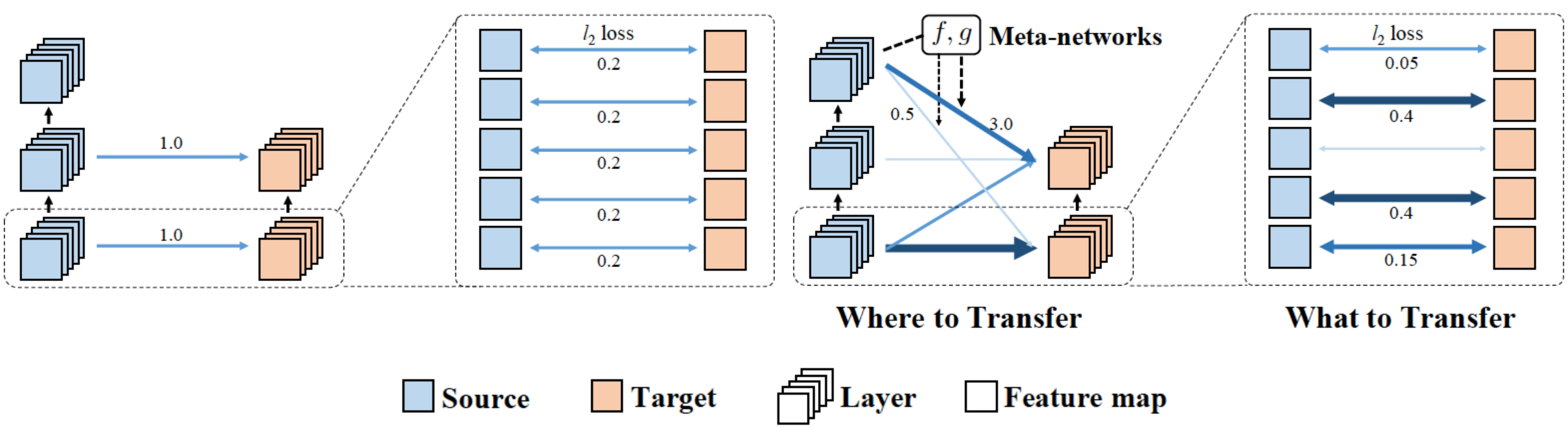
Teacher: ResNet-50, student: VGG11

HKD: Conventional knowledge distillation

- Previous methods **transfer hand-crafted and fixed source knowledge**
    - **Hand-crafted matching formulations**
        - E.g., **KL divergence** [Hinton et al., 2015] between output layers, **attention map** [Zagoruyko et al. 2017] between hidden feature maps
    - **Hand-crafted matching connections**
        - Transfer on output activations of each group of residual/convolutional blocks

- [Jang et al., 2019] automatically **finds what and where to transfer based on meta-learning** for maximizing transfer effect

Previous methods | **Learning What and Where to Transfer (L2T-ww)**



* source: [Jang et al., 2019]

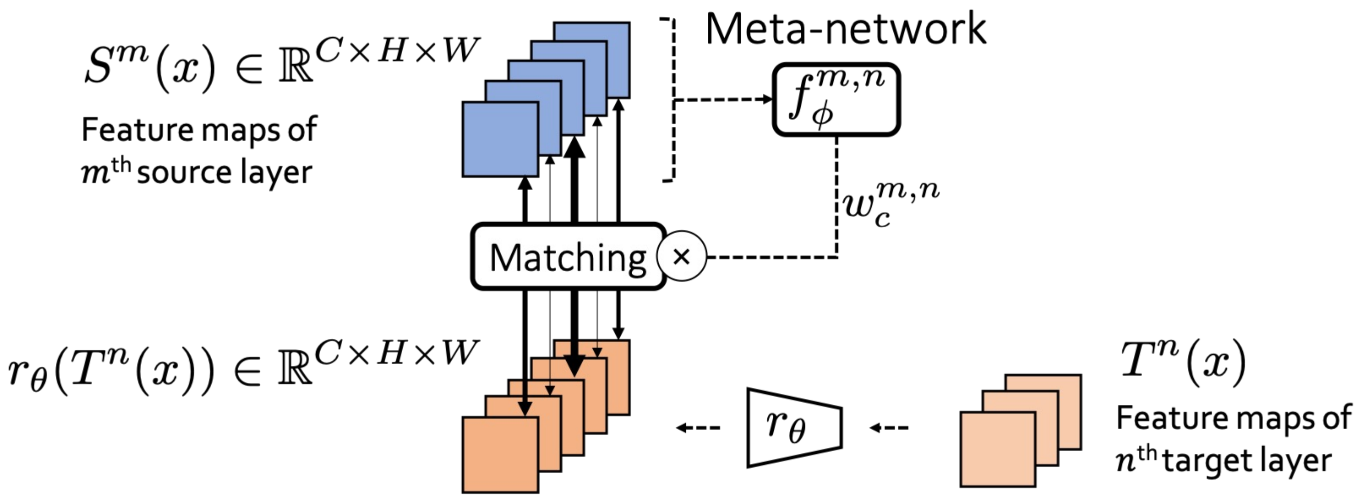
- [Jang et al., 2019] uses **meta-weighted feature matching** for transfer

- **Meta-network *f*** decides **useful channels** to transfer

Weight for channel *c*

$$\mathcal{L}_{\texttt{wfm}}^{m,n}(\theta|x, w^{m,n}) = \frac{1}{HW}\sum_{c} \boxed{w_c^{m,n}} \sum_{i,j} \boxed{\left(r_\theta(T_\theta^n(x))_{c,i,j} - S^m(x)_{c,i,j}\right)^2}$$

L2 distance at channel *c*



* source: [Jang et al., 2019]

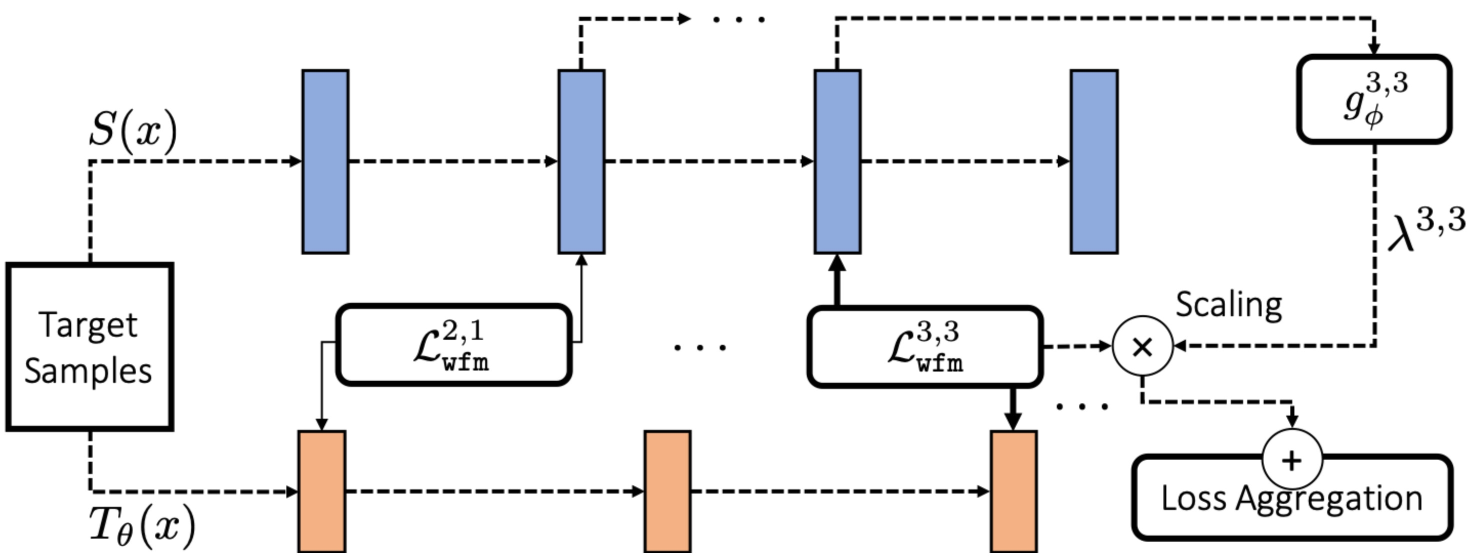
- [Jang et al., 2019] uses **meta-weighted feature matching** for transfer

- **Meta-network *g*** decides **useful pairs** of source/target layers to transfer

Weight for pair (m, n)

$$\mathcal{L}_{\mathtt{wfm}}(\theta|x,\phi) = \sum_{(m,n)\in\mathcal{C}} \boxed{\lambda^{m,n}}\boxed{\mathcal{L}^{m,n}_{\mathtt{wfm}}(\theta|x,w^{m,n})}$$

Transfer loss on pair (m, n)



* source: [Jang et al., 2019]

Q) How to learn meta-networks f, g?

- [Jang et al., 2019] proposes a bilevel scheme for training meta-parameters ϕ of meta-networks f, g

1. *Knowledge transfer*: for $t = 1, \dots, T$,
$$\theta_{t+1} = \theta_t - \alpha \nabla_\theta \mathcal{L}_{\texttt{wfm}}(\theta_t | x, \phi) \leftarrow \text{Transfer loss}$$
2. *One-step adaption*:
$$\theta_{T+2} = \theta_{T+1} - \alpha \nabla_\theta \mathcal{L}_{\texttt{org}}(\theta_{T+1} | x, y)$$
3. *Evaluation*:
$$\mathcal{L}_{\texttt{meta}}(\phi) = \mathcal{L}_{\texttt{org}}(\theta_{T+2} | x, y).$$
Task-specific loss
4. Update ϕ based on $\nabla_\phi \mathcal{L}_{\texttt{meta}}(\phi)$ using second-order gradients

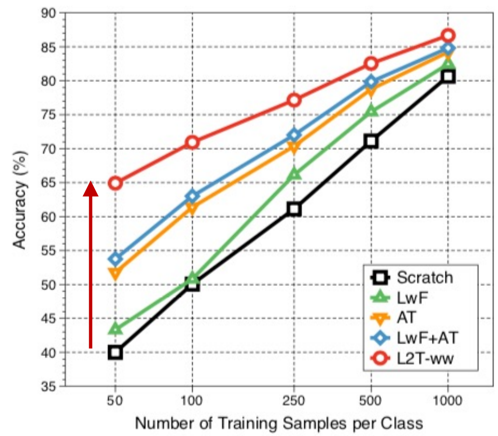
- Effective for learning ϕ with **a small number of steps T**
  - A popular bilevel scheme [Franceschi et al., 2018] **requires many steps**
- Joint-learning θ and ϕ without separate meta-learning phase

- **L2T-ww outperforms** previous methods **on various datasets, architectures**

Source task	TinyImageNet		ImageNet			
Target task	CIFAR-100	STL-10	CUB200	MIT67	Stanford40	Stanford Dogs
Scratch	67.69±0.22	65.18±0.91	42.15±0.75	48.91±0.53	36.93±0.68	58.08±0.26
LwF[6]	69.23±0.09	68.64±0.58	45.52±0.66	53.73±2.14	39.73±1.63	66.33±0.45
AT[1] (one-to-one)	67.54±0.40	74.19±0.22	57.74±1.17	59.18±1.57	59.29±0.91	69.70±0.08
LwF[6]+AT[1] (one-to-one)	68.75±0.09	75.06±0.57	58.90±1.32	61.42±1.68	60.20±1.34	72.67±0.26
FM[3] (single)	69.40±0.67	75.00±0.34	47.60±0.31	55.15±0.93	42.93±1.48	66.05±0.76
FM[3] (one-to-one)	69.97±0.24	76.38±1.18	48.93±0.40	54.88±1.24	44.50±0.96	67.25±0.88
L2T-w (single)	70.27±0.09	74.35±0.92	51.95±0.83	60.41±0.37	46.25±3.66	69.16±0.70
L2T-w (one-to-one)	70.02±0.19	76.42±0.52	56.61±0.20	59.78±1.90	48.19±1.42	69.84±1.45
L2T-ww (all-to-all)	**70.96**±0.61	**78.31**±0.21	**65.05**±1.19	**64.85**±2.75	**63.08**±0.88	**78.08**±0.96

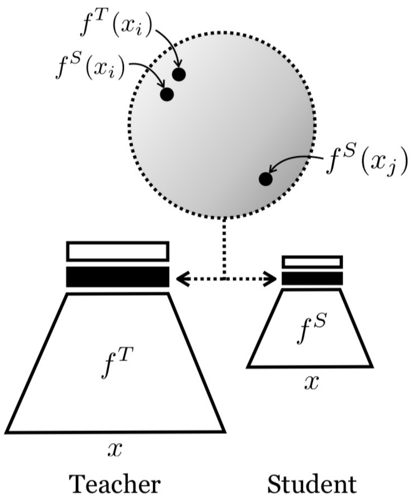
- **L2T-ww** can **aggregate multiple source** knowledge (left)
- **L2T-ww** can transfer knowledge effectively **on limited-data regime**

First source	TinyImageNet (ResNet32)			
Second source	None	TinyImageNet (ResNet20)	TinyImageNet (ResNet32)	CIFAR-10 (ResNet32)
Scratch	65.18±0.91	65.18±0.91	65.18±0.91	65.18±0.91
LwF[6]	68.64±0.58	68.56±2.24	68.05±2.12	69.51±0.63
AT[1]	74.19±0.22	73.24±0.12	73.78±1.16	73.99±0.51
LwF[6]+AT[1]	75.06±0.57	74.72±0.46	74.77±0.30	74.41±1.51
FM[3] (single)	75.00±0.34	75.83±0.56	75.99±0.11	74.60±0.73
FM[3] (one-to-one)	76.38±1.18	77.45±0.48	77.69±0.79	77.15±0.41
L2T-ww (all-to-all)	**78.31**±0.21	**79.35**±0.41	**79.80**±0.52	**80.52**±0.29



* source: [Jang et al., 2019]

- [Tian et al., 2020] transfers the **output similarity** of data points
  - Maximize the similarity of **same data point**, and minimize between **other points**



$f^T(x_i)$ and $f^S(x_i)$ is similar (**same sample**)
$f^T(x_i)$ and $f^S(x_j)$ is not similar (**other $N-1$ samples**)

- Contrastive-object **maximize the mutual information** between models

$$I(T;S) \geq \log(N) + \underbrace{\mathbb{E}_{q(T,S|C=1)}[\log h^*(T,S)]}_{\text{Maximize similarity}} + \underbrace{N\mathbb{E}_{q(T,S|C=0)}[\log(1-h^*(T,S))]}_{\text{Minimize similarity}}$$

$$h(T,S) = \frac{e^{g^T(T)' g^S(S)/\tau}}{e^{g^T(T)' g^S(S)/\tau} + \frac{N}{M}}$$

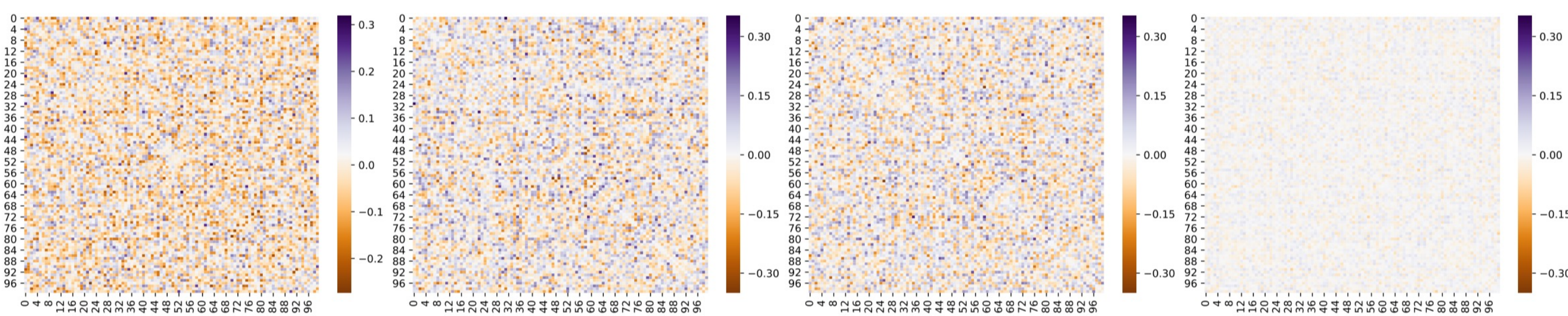
$h(T,S) \in [0,1]$ is a similarity measure
Where $T = f^T(x_i)$, $S = f^S(x_j)$ is the representation
and g^T, g^S is a linear layer of teacher and student, respectively

* source: [Tian et al., 2020]

- **CRD consistently outperforms** previous methods **on various architectures**

Teacher Student	WRN-40-2 WRN-16-2	WRN-40-2 WRN-40-1	resnet56 resnet20	resnet110 resnet20	resnet110 resnet32	resnet32x4 resnet8x4	vgg13 vgg8
Teacher	75.61	75.61	72.34	74.31	74.31	79.42	74.64
Student	73.26	71.98	69.06	69.06	71.14	72.50	70.36
KD*	74.92	73.54	70.66	70.67	73.08	73.33	72.98
FitNet*	73.58 (↓)	72.24 (↓)	69.21 (↓)	68.99 (↓)	71.06 (↓)	73.50 (↑)	71.02 (↓)
AT	74.08 (↓)	72.77 (↓)	70.55 (↓)	70.22 (↓)	72.31 (↓)	73.44 (↑)	71.43 (↓)
SP	73.83 (↓)	72.43 (↓)	69.67 (↓)	70.04 (↓)	72.69 (↓)	72.94 (↓)	72.68 (↓)
CC	73.56 (↓)	72.21 (↓)	69.63 (↓)	69.48 (↓)	71.48 (↓)	72.97 (↓)	70.71 (↓)
VID	74.11 (↓)	73.30 (↓)	70.38 (↓)	70.16 (↓)	72.61 (↓)	73.09 (↓)	71.23 (↓)
RKD	73.35 (↓)	72.22 (↓)	69.61 (↓)	69.25 (↓)	71.82 (↓)	71.90 (↓)	71.48 (↓)
PKT	74.54 (↓)	73.45 (↓)	70.34 (↓)	70.25 (↓)	72.61 (↓)	73.64 (↑)	72.88 (↓)
AB	72.50 (↓)	72.38 (↓)	69.47 (↓)	69.53 (↓)	70.98 (↓)	73.17 (↓)	70.94 (↓)
FT*	73.25 (↓)	71.59 (↓)	69.84 (↓)	70.22 (↓)	72.37 (↓)	72.86 (↓)	70.58 (↓)
FSP*	72.91 (↓)	n/a	69.95 (↓)	70.11 (↓)	71.89 (↓)	72.62 (↓)	70.23 (↓)
NST*	73.68 (↓)	72.24 (↓)	69.60 (↓)	69.53 (↓)	71.96 (↓)	73.30 (↓)	71.53 (↓)
CRD	**75.48** (↑)	**74.14** (↑)	**71.16** (↑)	**71.46** (↑)	**73.48** (↑)	**75.51** (↑)	**73.94** (↑)
CRD+KD	75.64 (↑)	74.38 (↑)	71.63 (↑)	71.56 (↑)	73.75 (↑)	75.46 (↑)	74.29 (↑)

- Visualization: ***difference*** of correlation matrices of student and teacher logits.
  - CRD shows **significant matching** between **student's and teacher's correlations**



(a) Student: vanilla (b) Student: AT (c) Student: KD (d) Student: ours (CRD)

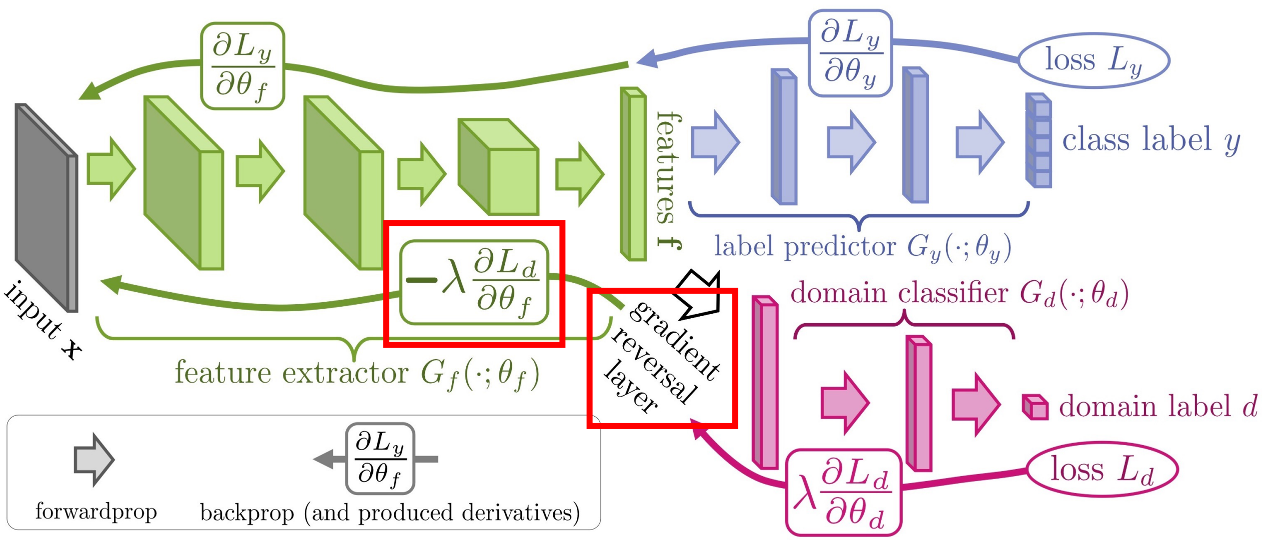
Table of Contents

- **Domain adaptation** aims to learn $f: X_T \to Y_T$ only using (X_S, Y_S) and X_T

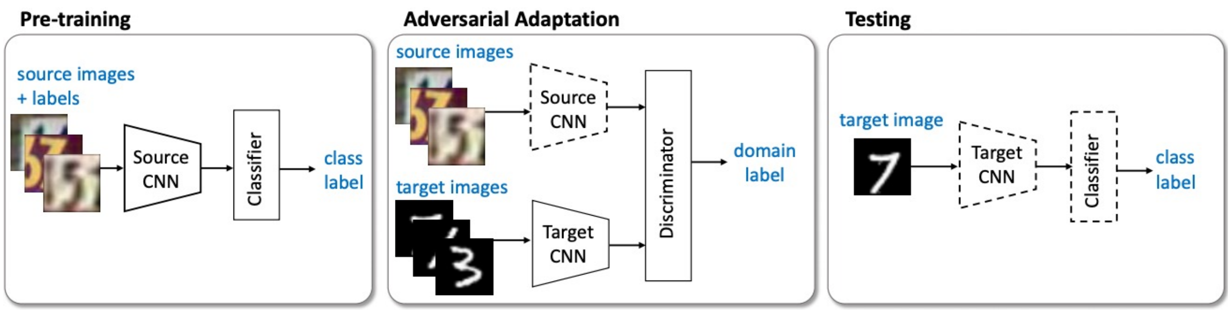
- There are **two general** approaches:
  - **Source/target feature matching:** Make features of X_S and X_T be similar
  - **Target data augmentation:** Generate target data (X_T', Y_T') using domain transfer

*Source: Ganin et al. "Unsupervised Domain Adaptation by Backpropagation", ICML 2015

- [Ganin et al., 2015]
  - **Goal:** Make features of source data X_S and target data X_T be similar
  - **Idea:** Train discriminator D which classifies domain label, and adversarially train network to fool discriminator fail to distinguish source/target feature
  - To this end, gradient from domain classifier is **reversely applied** for the network



- [Tzeng et al., 2017]
  - **Goal:** Make features of source data X_S and target data X_T be similar
  - Instead, one can alternatively update discriminator, similar to GAN scheme
  - Also, one can train separate feature extractors for source/target domain

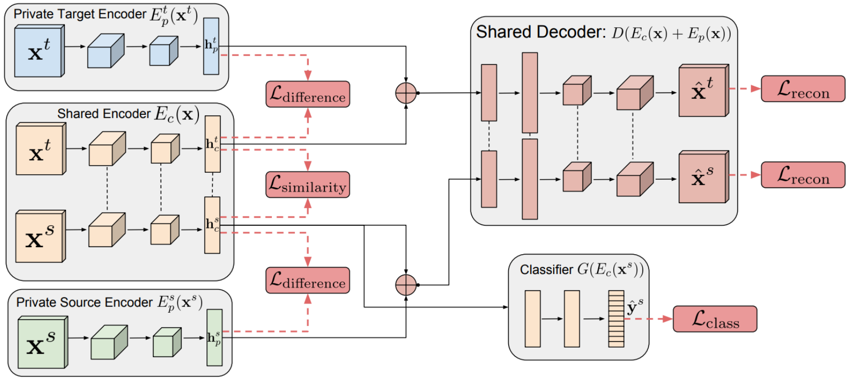


  - It is less **stable for train**, but shows **better performance** than gradient reversal

Method	MNIST → USPS	USPS → MNIST	SVHN → MNIST
Source only	0.752 ± 0.016	0.571 ± 0.017	0.601 ± 0.011
Gradient reversal	0.771 ± 0.018	0.730 ± 0.020	0.739 [16]
Domain confusion	0.791 ± 0.005	0.665 ± 0.033	0.681 ± 0.003
CoGAN	0.912 ± 0.008	0.891 ± 0.008	did not converge
ADDA (Ours)	0.894 ± 0.002	0.901 ± 0.008	0.760 ± 0.018

Domain Separation Network (DSN)

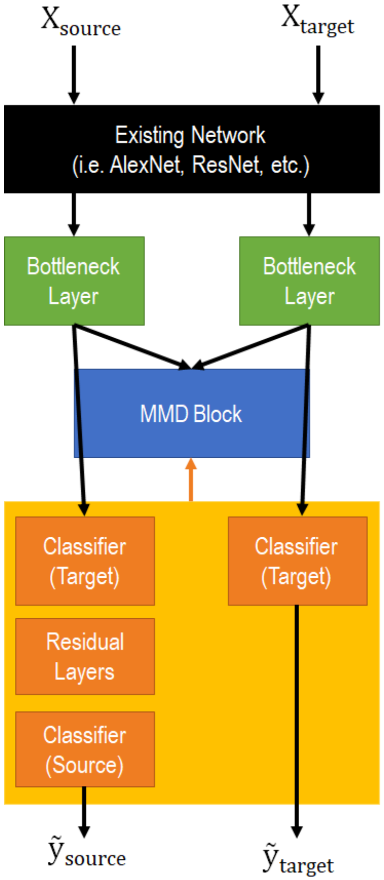
- [Bousmaliz et al., 2016]
  - **Motivation:** Is it rational to exactly match features for source/target data?
  - **Idea:** Consider style of each domain in addition to the **shared content**
  - To this end, train **shared content encoder** E_C and **private style encoders** E_S^S, E_S^T
  - Classifier ignores styles but only use **shared content** as an input



Model	MNIST to MNIST-M	Synth Digits to SVHN	SVHN to MNIST	Synth Signs to GTSRB
Source-only	56.6 (52.2)	86.7 (86.7)	59.2 (54.9)	85.1 (79.0)
CORAL [26]	57.7	85.2	63.1	86.9
MMD [29, 17]	76.9	88.0	71.1	91.1
DANN [8]	77.4 (76.6)	90.3 (91.0)	70.7 (73.8)	92.9 (88.6)
DSN w/ MMD (ours)	80.5	88.5	72.2	92.6
DSN w/ DANN (ours)	**83.2**	**91.2**	**82.7**	**93.1**
Target-only	98.7	92.4	99.5	99.8

Residual Transfer Network (RTN)

- [Long et al., 2016]
  - **Motivation:** Is it rational to exactly match classifiers for source/target data?
  - **Idea:** Define source classifier as a residual function of target classifier



$$f_S(x) = f_T(x) + \Delta f(x)$$

$$\|\Delta f(x)\| \ll |f_T(x)| \approx |f_S(x)|$$

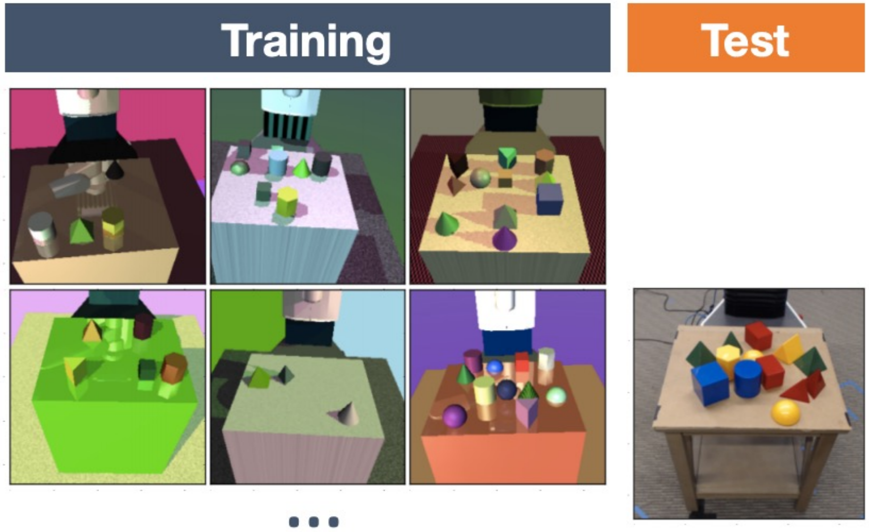
- To ensure that f_T learns structure of target domain, minimize entropy for target data, which is popular method for **semi-supervised** learning [Grandvalet & Bengio, 2004]

- Hence, in addition to (supervised) **classification loss** L and **feature matching loss** $D(X_S, X_T)$ (e.g., GAN loss), use (unsupervised) entropy loss H on target dataset

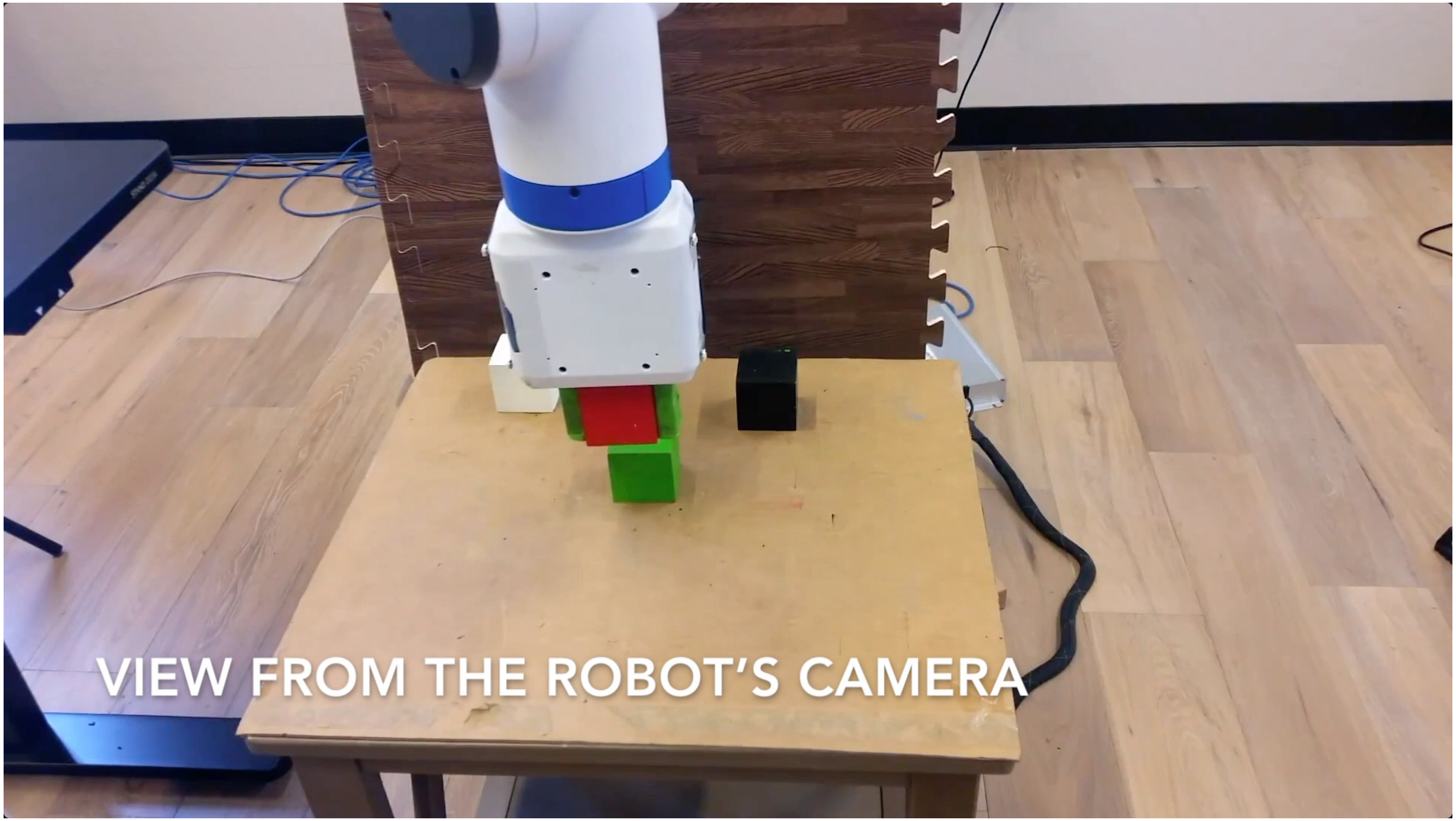
$$\mathcal{L} = \mathbb{E}_{x_s}[L(f_S(x_s), y_s)] + \gamma \mathbb{E}_{x_t}[H(f_T(x_t))] + \lambda D(X_S, X_T)$$

*Source: https://wiki.math.uwaterloo.ca/statwiki/index.php?title=Unsupervised_Domain_Adaptation_with_Residual_Transfer_Networks

- [Tobin et al., 2017]
  - **Motivation:** Source/target feature matching can be viewed as disentangling content and style (remove style of each domain but only keep common content)
  - **Idea:** In simulation-to-real (sim2real) setting, we can disentangle content by domain augmentation
    - Train NN on simulations with randomly generated styles
      $\Rightarrow$ style sums up, and only content remains



- **Results**
  - https://blog.openai.com/generalizing-from-simulation/

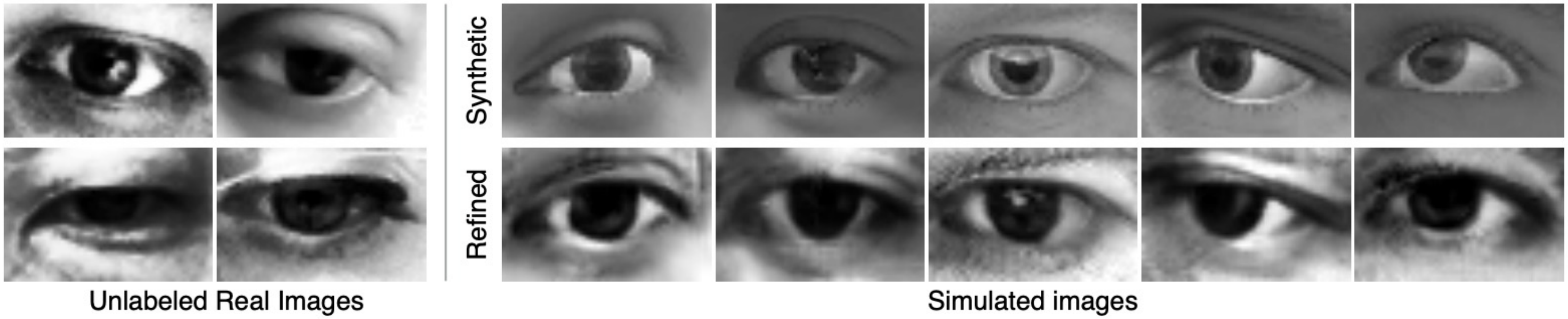


- **Domain adaptation** aims to learn $f: X_T \rightarrow Y_T$ only using (X_S, Y_S) and X_T

- There are **two general** approaches:
  - **Source/target feature matching:** Make features of X_S and X_T be similar
  - **Target data augmentation**: Generate target data (X_T', Y_T') using domain transfer

*Source: Ganin et al. "Unsupervised Domain Adaptation by Backpropagation", ICML 2015

- [Shrivastava et al., 2017]
  - **Idea:** Generate target data with domain transfer model $G: X_S \rightarrow X_T$
    - Given source data (x_s, y_s) and transfer model G, we can generate **labeled target data** $(x_t', y_t') = (G(x_s), y_s)$, and use it to train target network
  - Popular application is **augmenting real images** from **synthetic** images

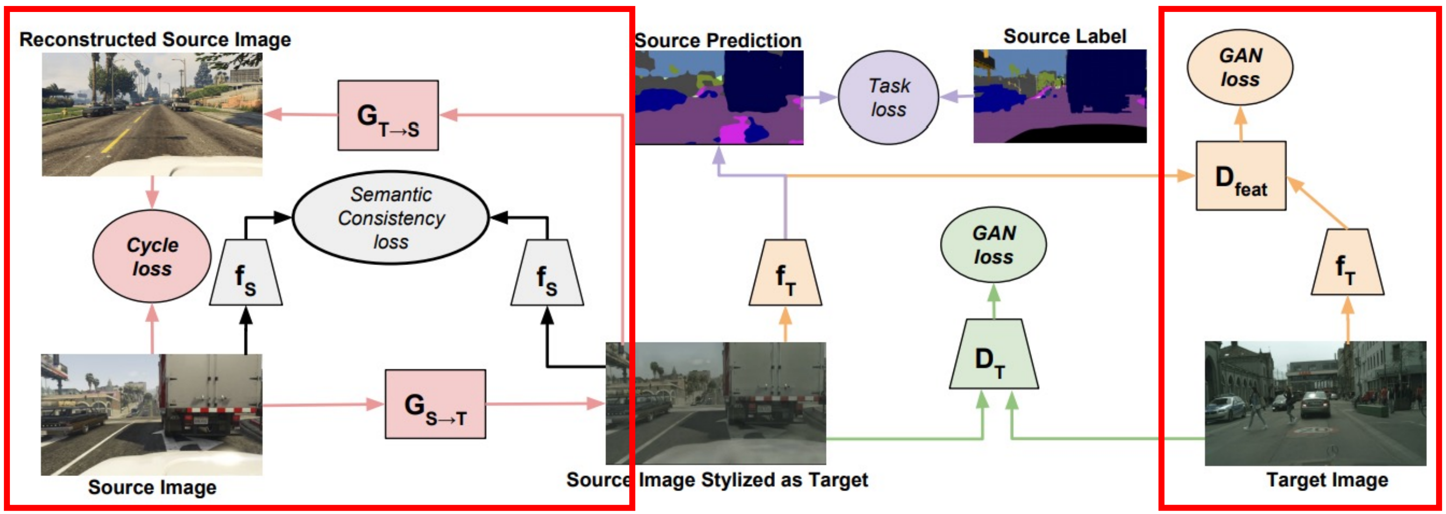


Training data	% of images within d
Synthetic Data	69.7
Refined Synthetic Data	72.4
Real Data	74.5
Synthetic Data 3x	77.7
Refined Synthetic Data 3x	**83.3**

*Source: Shrivastava et al. "Learning from Simulated and Unsupervised Images through Adversarial Training", CVPR 2017

- [Hoffman et al., 2017]
  - **Motivation:** Bridging gap between two approaches: source/target feature matching and target data augmentation?
  - Combine **ADDA** (feature matching via GAN) and **CycleGAN** (domain transfer)

target data augmentation (CycleGAN)



source/target feature matching

Source image (GTA5)

Adapted source image (**Ours**)

Target image (CityScapes)

Pixel accuracy on target

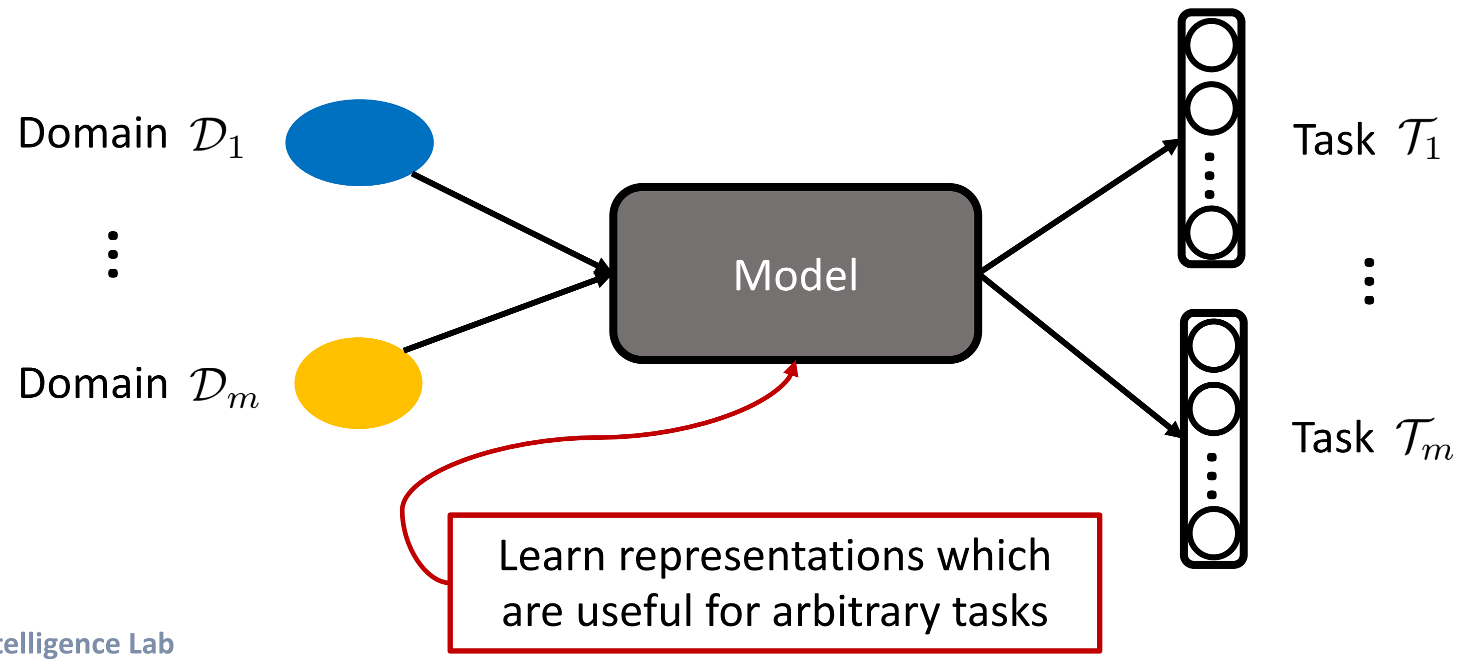
Source-only:	54.0%
Adapted (**ours**):	**83.6%**

Table of Contents

What is Multi-task Learning?

- [Zhang et al., 2017] defines the multi-task learning as following:
  - Given ***m* learning *tasks*** $\{\mathcal{T}_i\}_{i=1}^m$
    - where all the tasks or a subset of them are **related**,
  - ***Multi-task learning*** (MTL) aims to **improve** the learning of a model for $\mathcal{T}_i$ using the knowledge contained in all or some of the m tasks

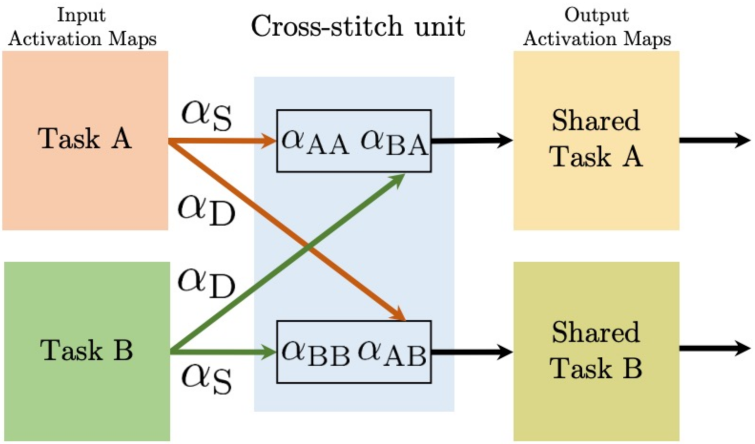
  - In the view of definition of transfer learning [Pan et al., 2010], all learning tasks $\{\mathcal{T}_i\}_{i=1}^m$ are considered as both source and target tasks

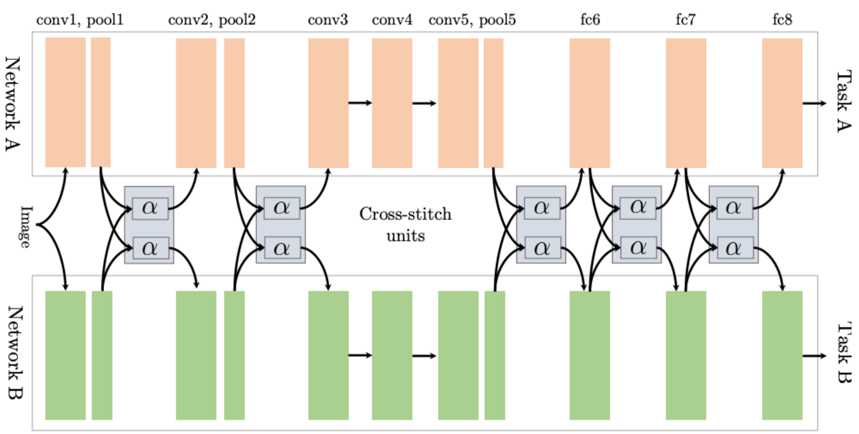


- [Misra et al., 2016] tries to find the best shared representations for multi-task learning with cross-stitch units

$$\begin{bmatrix} \tilde{x}_{\mathrm{A}}^{ij} \\ \tilde{x}_{\mathrm{B}}^{ij} \end{bmatrix} = \begin{bmatrix} \alpha_{\mathrm{AA}} & \alpha_{\mathrm{AB}} \\ \alpha_{BA} & \alpha_{\mathrm{BB}} \end{bmatrix} \begin{bmatrix} x_{\mathrm{A}}^{ij} \\ x_{\mathrm{B}}^{ij} \end{bmatrix}$$

  - $x_{\mathrm{A}}^{ij},\ x_{\mathrm{B}}^{ij}$ are activation map (at location i,j) of networks for task A, B, respectively
  - α is trained by backpropagation with different learning rates
  - Maintain one cross-stitch unit per channel





* source: [Misra et al., 2016]

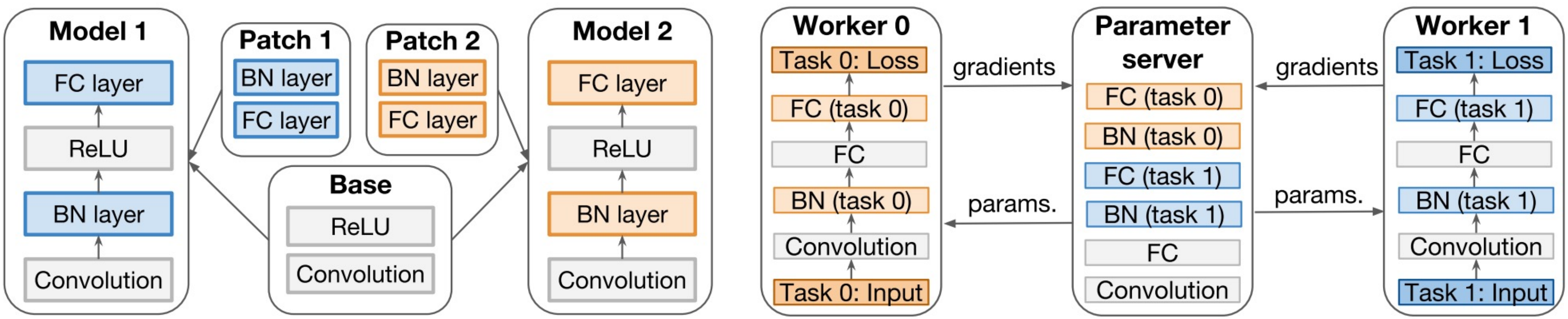
- Multi-task (Surface Normal / Segmentation) learning on NYU-v2 dataset
  - Cross-stitch uses 2 convolutional networks
  - Ensemble uses 4 convolutional networks (2 for each task)
  - It shows that **sharing information can improve** the performance

	Surface Normal					Segmentation		
	Angle Distance (Lower Better)		Within t° (Higher Better)			(Higher Better)		
Method	Mean	Med.	11.25	22.5	30	pixacc	mIU	fwIU
One-task	34.8	19.0	38.3	53.5	59.2	-	-	-
	-	-	-	-	-	46.6	18.4	33.1
Ensemble	34.4	18.5	38.7	54.2	59.7	-	-	-
	-	-	-	-	-	**48.2**	18.9	33.8
Split `conv4`	34.7	19.1	38.2	53.4	59.2	47.8	19.2	33.8
MTL-shared	34.7	18.9	37.7	53.5	58.8	45.9	16.6	30.1
Cross-stitch [**ours**]	**34.1**	**18.2**	**39.0**	**54.4**	**60.2**	47.2	**19.3**	**34.0**

- **Drawbacks**
  - Parameter-inefficiency because it requires **one CNN per each task**

K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning

- [Mudrakarta et al., 2019] introduces a single **odel-patch** for each task
  - One shared base model for all tasks
  - For multi-task learning, train model-patches and shared parts simultaneously
  - For transfer learning, freeze the shared parts / train new model-patch only
  - **Multiple networks share most weights** (>95% parameters)



- Two types of **model-patch**
  - **Scale-and-bias** (S/B) patch: a normalization layer (e.g., BN)
  - **Depth-wise-convolution** (DW) patch: depth-wise separable convolutional layers

* source: [Mudrakarta et al., 2019]

- **Despite using much fewer parameters**, competitive performance is achieved

Table 4: Multi-task learning with MobilenetV2 on ImageNet and Places-365.

Task	S/B patch + last layer	Last layer	Independently trained
Imagenet Places365	70.2% **54.3%**	64.4% 51.4%	71.8% 54.2%
# total parameters	3.97M	3.93M	6.05M

One patch for each task
Sharing Most weights

≈

One model for each task

- When transfer learning, **despite fine-tuning much fewer parameters**, it achieves nontrivial performance

Fine-tuned params.	Flowers		Cars		Aircraft	
	Acc.	#params	Acc.	#params	Acc.	#params
Last layer	84.5	208K	55	402K	45.9	205K
S/B + last layer	**90.4**	244K	**81**	437K	**70.7**	241K
S/B only (random last)	79.5	**36K**	33	**36K**	52.3	**36K**
All (ours)	93.3	25M	92.3	25M	87.3	25M
All (Cui et al., 2018)	96.3	25M	91.3	25M	82.6	25M

Table of Contents

- The naive approach to combining multi objective losses is to perform a weighted linear sum of the losses for each individual task.

$$\mathcal{L}_{\text{total}} = \sum_i \boxed{w_i} \mathcal{L}_i$$

- [Kendall et al., 2018] proposes that homoscedastic (i.e. task-dependent) *uncertainty* can be used as a weight for losses in a multi-task learning problem.
  - They adapted a likelihood as below, with a **noise scalar σ** . Note that the probability distribution becomes uniform as $\sigma \to \infty$.

For classification tasks $p(\mathbf{y}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \text{Softmax}(\frac{1}{\sigma^2}\mathbf{f}^{\mathbf{W}}(x))$

For regression tasks $p(\mathbf{y}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \mathcal{N}(\mathbf{f}^{\mathbf{W}}(\mathbf{x}), \sigma^2)$

  - Let's assume that the total likelihood can be factorized over the each output, given some sufficient statistics.

$$p(\mathbf{y}_1, ..., \mathbf{y}_K|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = p(\mathbf{y}_1|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) \dots p(\mathbf{y}_K|\mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$

- The log likelihood for output can be written as

For classification tasks
$$\log p(\mathbf{y}=c|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) = \frac{1}{\sigma^2}\mathbf{f}_c^{\mathbf{W}}(\mathbf{x}) - \log\sum_{c'}\exp\left(\frac{1}{\sigma^2}\mathbf{f}_{c'}^{\mathbf{W}}(\mathbf{x})\right)$$
$$\mathcal{L}_{\text{cls}}(\mathbf{W}) = -\log \text{Softmax}(\mathbf{y}, \mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$

For regression tasks
$$\log p(\mathbf{y}|\mathbf{f}^{\mathbf{W}}(\mathbf{x})) \propto -\frac{1}{2\sigma^2}||\mathbf{y}-\mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2 - \log\sigma$$
$$\mathcal{L}_{\text{reg}}(\mathbf{W}) = ||\mathbf{y}-\mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2$$

This constructions can be trivially extended to multiple outputs.

- If there are two regression tasks,

$$\mathcal{L}(\mathbf{W},\sigma_1,\sigma_2) = -\log p(\mathbf{y}_1,\mathbf{y}_2|\mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$
$$\propto \frac{1}{2\sigma_1^2}||\mathbf{y}_1-\mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2 + \frac{1}{2\sigma_2^2}||\mathbf{y}_2-\mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2 + \log\sigma_1\sigma_2$$

weighted sum
$$= \boxed{\frac{1}{2\sigma_1^2}\mathcal{L}_{1,\text{reg}}(\mathbf{W}) + \frac{1}{2\sigma_2^2}\mathcal{L}_{2,\text{reg}}(\mathbf{W})} + \log\sigma_1\sigma_2$$

- If the 1st task is a regression task, and the 2nd one is a classification task,

$$\mathcal{L}(\mathbf{W},\sigma_1,\sigma_2) = -\log p(\mathbf{y}_1,\mathbf{y}_2=c|\mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$
$$\propto \frac{1}{2\sigma_1^2}||\mathbf{y}_1-\mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2 + \log\sigma_1 - \log p(\mathbf{y}_2=c|\mathbf{f}^{\mathbf{W}}(\mathbf{x}))$$
$$= \frac{1}{2\sigma_1^2}||\mathbf{y}_1-\mathbf{f}^{\mathbf{W}}(\mathbf{x})||^2 - \frac{1}{\sigma_2^2}\log\text{Softmax}(\mathbf{y}_2,\mathbf{f}^{\mathbf{W}}(\mathbf{x})) + \log\sigma_1 + \log\frac{\sum_{c'}\exp\left(\frac{1}{\sigma_2^2}\mathbf{f}_{c'}^{\mathbf{W}}(\mathbf{x})\right)}{\left(\sum_{c'}\exp\left(\mathbf{f}_{c'}^{\mathbf{W}}(\mathbf{x})\right)\right)^{\frac{1}{\sigma_2^2}}}$$

weighted sum
$$\approx \boxed{\frac{1}{2\sigma_1^2}\mathcal{L}_{1,\text{reg}}(\mathbf{W}) + \frac{1}{\sigma_2^2}\mathcal{L}_{2,\text{cls}}(\mathbf{W})} + \log\sigma_1 + \log\sigma_2 \quad \text{as} \quad \sigma_2 \to 1\ .$$

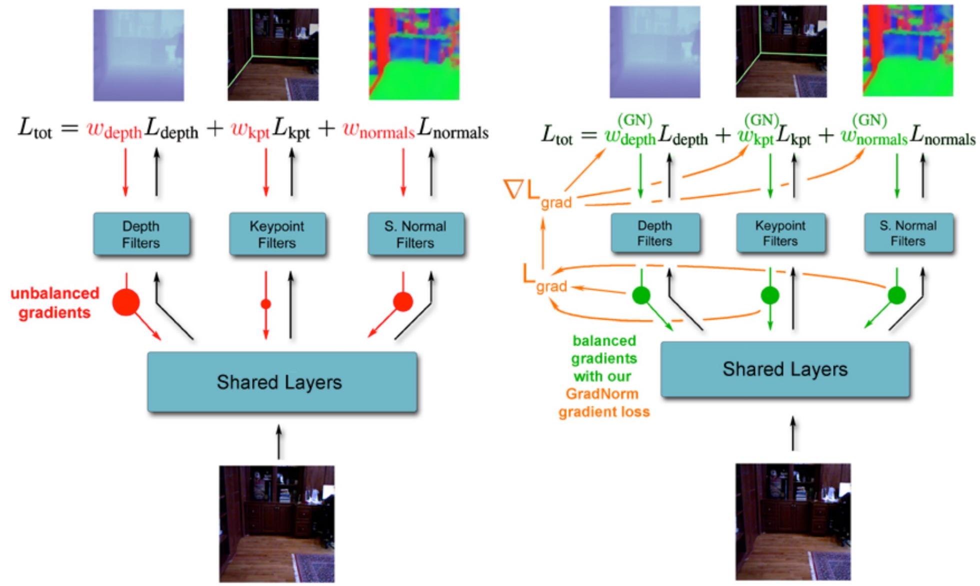
- In practice, the log variance $s := \log \sigma^2$ is trained by the network .
  - This term is added to weighted sum of original multi-task losses.
- In experiments, there are three tasks:
  - Semantic segmentation (classification)
  - Instance segmentation (regression)
  - Depth regression (regression)

Approx. optimal weights are found by grid search.

Loss	Task Weights Seg.	Inst.	Depth	Segmentation IoU [%]	Instance Mean Error [px]	Inverse Depth Mean Error [px]
Segmentation only	1	0	0	59.4%	-	-
Instance only	0	1	0	-	4.61	-
Depth only	0	0	1	-	-	0.640
Unweighted sum of losses	0.333	0.333	0.333	50.1%	3.79	0.592
Approx. optimal weights	0.89	0.01	0.1	62.8%	3.61	0.549
2 task uncertainty weighting	✓	✓		61.0%	**3.42**	-
2 task uncertainty weighting	✓		✓	62.7%	-	0.533
2 task uncertainty weighting		✓	✓	-	3.54	0.539
3 task uncertainty weighting	✓	✓	✓	**63.4%**	3.50	**0.522**

* source: [Kendall et al., 2018]

- [Chen et al., 2018]
- At time t, the weighted average for multi-task learning = $\sum_i w_i(t)\mathcal{L}_i(t)$
- **The gradient for a task might be dominant** when multi-task learning
  - It depends on task difficulties, loss functions, and so on
  - **Q)** What is correct balance for w_i?



- **Key Idea**: If a task is **not trained enough** ⇒ norm of its gradient **should be large**

* source: [Chen et al., 2018]

- **Gradient norm**
  - $G_W^{(i)}(t) = \|\nabla_W w_i(t) L_i(t)\|_2$: gradient norm of task *i*
  - $\bar{G}_W(t) = \mathbb{E}_i[G_W^{(i)}(t)]$: average gradient norm across all tasks

- **Training rates** for measuring current states of learning of tasks
  - Inverse training rates $\tilde{L}_i(t) = L_i(t)/L_i(0)$
  - Relative inverse training rates $r_i(t) = \tilde{L}_i(t)/\mathbb{E}_j[\tilde{L}_j(t)]$

- Large $r_i(t)$ ⇒ need to train more ⇒ need large gradients
  - **Our desired gradient norm**:

$$G_W^{(i)}(t) \mapsto \bar{G}_W(t) \times [r_i(t)]^\alpha$$

    where α is a hyperparameter
  - To balance the norms based on training rates, minimize L_{grad} over w_i

$$L_{\text{grad}}(t; w_i(t)) = \sum_i \left| G_W^{(i)} - \bar{G}_W(t) \times [r_i(t)]^\alpha \right|$$

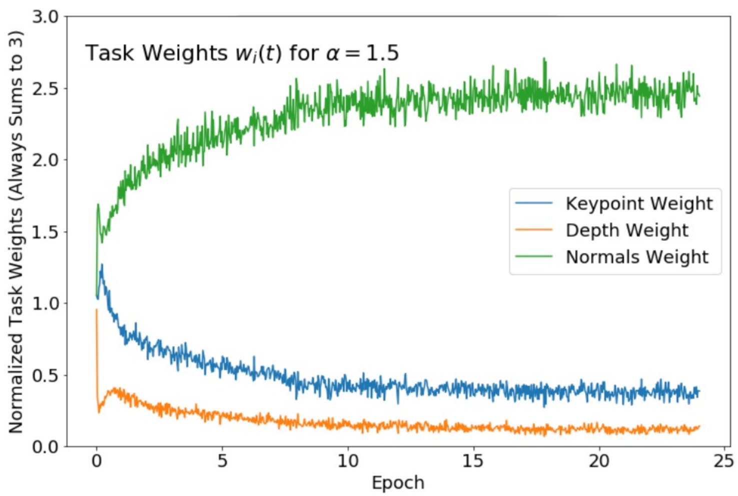
- Train on NYUv2+kepoint/segmentation dataset with 3 different tasks

Model and Weighting Method	Depth RMS Err. (m)	Seg. Err. (100-IoU)	Normals Err. (1-|cos|)
VGG Backbone			
Depth Only	1.038	-	-
Seg. Only	-	70.0	-
Normals Only	-	-	**0.169**
Equal Weights	0.944	70.1	0.192
GradNorm Static	0.939	**67.5**	0.171
GradNorm $\alpha = 1.5$	**0.925**	67.8	0.174

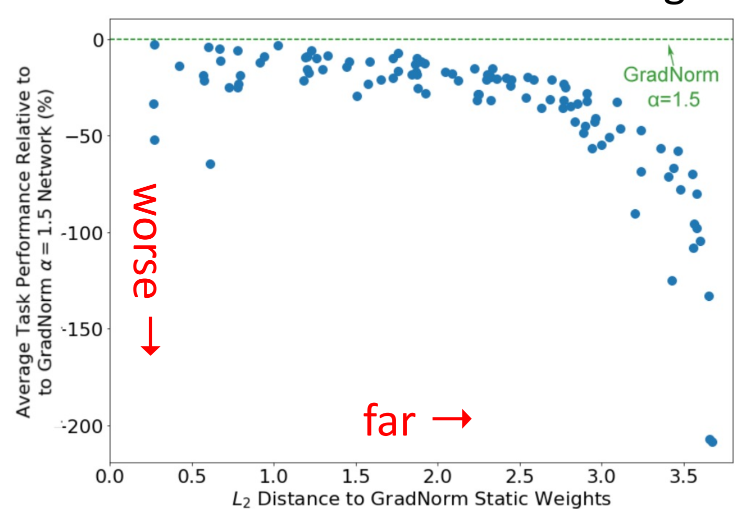
Model and Weighting Method	Depth RMS Err. (m)	Kpt. Err. (%)	Normals Err. (1-|cos|)
ResNet Backbone			
Depth Only	0.725	-	-
Kpt Only	-	7.90	-
Normals Only	-	-	**0.155**
Equal Weights	0.697	7.80	0.172
(Kendall et al., 2017)	0.702	7.96	0.182
GradNorm Static	0.695	7.63	0.156
GradNorm $\alpha = 1.5$	**0.663**	**7.32**	**0.155**

- If using farther weights from GradNorm, then worse results are obtained

Weights during training



Performance with various weights



Multi-task Learning as Multi-objective Optimization

- [Sener et al., 2018]
- The loss function for multi-task learning is generally the weighted summation

$$\min_\theta \sum_{t=1}^{T} w_t \mathcal{L}_t(\theta)$$

- **For finding weights,** expensive grid search or heuristics are required
  - Heuristics: [Kendall et al., 2018], [Chen et al., 2018]

- **Pareto optimality** (multi-objective optimization formulation)
  - A solution θ *dominates* $\bar{\theta}$ if $\mathcal{L}_t(\theta) \leq \mathcal{L}_t(\bar{\theta})$ for all tasks t
  - A solution $\theta^\star$ is called *Pareto optimal* if there is no θ that dominates $\theta^\star$
  - The Pareto optimal solution can be considered as a solution for multi-task learning
  - **Q)** How to find the Pareto optimal solutions?

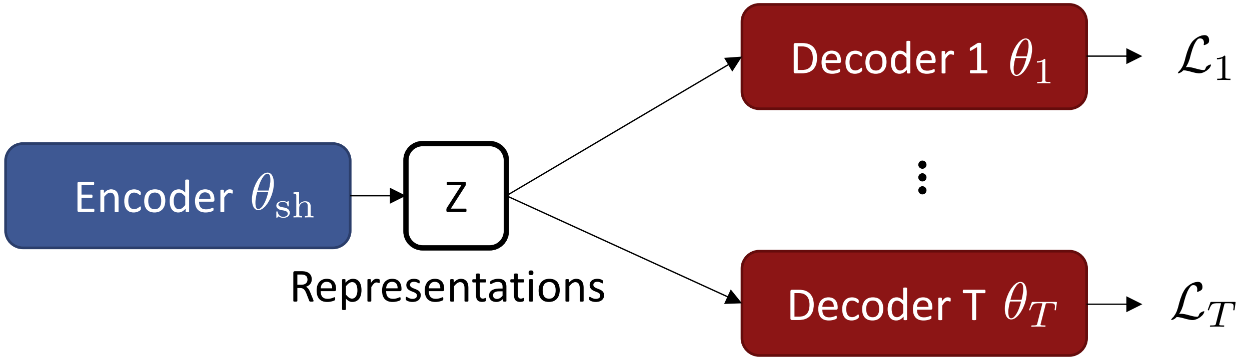
- **Multiple Gradient Descent Algorithm** (MGDA)

$$\min_{\alpha_1,\ldots,\alpha_T} \left\{ \left\| \sum_{t=1}^{T} \alpha_t \nabla_{\theta_{\text{sh}}} \mathcal{L}_t(\theta_{\text{sh}}, \theta_t) \right\|_2^2 \,\middle|\, \sum_{t=1}^{T} \alpha_t = 1, \alpha_t \geq 0 \right\}$$

  - Its solution gives **Pareto stationary** (necessary for optimality) **solutions or a descent direction that improves all tasks**
  - It can be efficiently solved by Frank-Wolfe algorithm (detail is omitted)

Multi-task Learning as Multi-objective Optimization

- **Issue:** MGDA needs to compute $\nabla_{\theta_{\text{sh}}} \mathcal{L}_t(\theta_{\text{sh}}, \theta_t)$ for each task t
  - Linear scaling of the training time
- **Solution:** Use encoder-decoder architectures
  - **One shared encoder** for all tasks
  - **One separate decoder** for each task
  - Encoder-decoder architectures are typically used for multi-task learning



  - Then, we can state an upper bound and minimize it efficiently

$$\left\| \sum_{t=1}^{T} \alpha_t \nabla_{\theta_{\text{sh}}} \mathcal{L}_t(\theta_{\text{sh}}, \theta_t) \right\|_2^2 \leq \left\| \frac{\partial Z}{\partial \theta_{\text{sh}}} \right\|_2^2 \left\| \sum_{t=1}^{T} \alpha_t \nabla_Z \mathcal{L}_t(\theta_{\text{sh}}, \theta_t) \right\|_2^2$$

Independent to α_t

Multi-task Learning as Multi-objective Optimization

- 40 binary tasks on CelebA dataset (lower is better)
  - This multi-objective optimization [Sener et al., 2018] outperforms uniform scaling, heuristic weights [Kendall et al., 2018], [Chen et al., 2018]
  - Grid search is not available because there are too many tasks

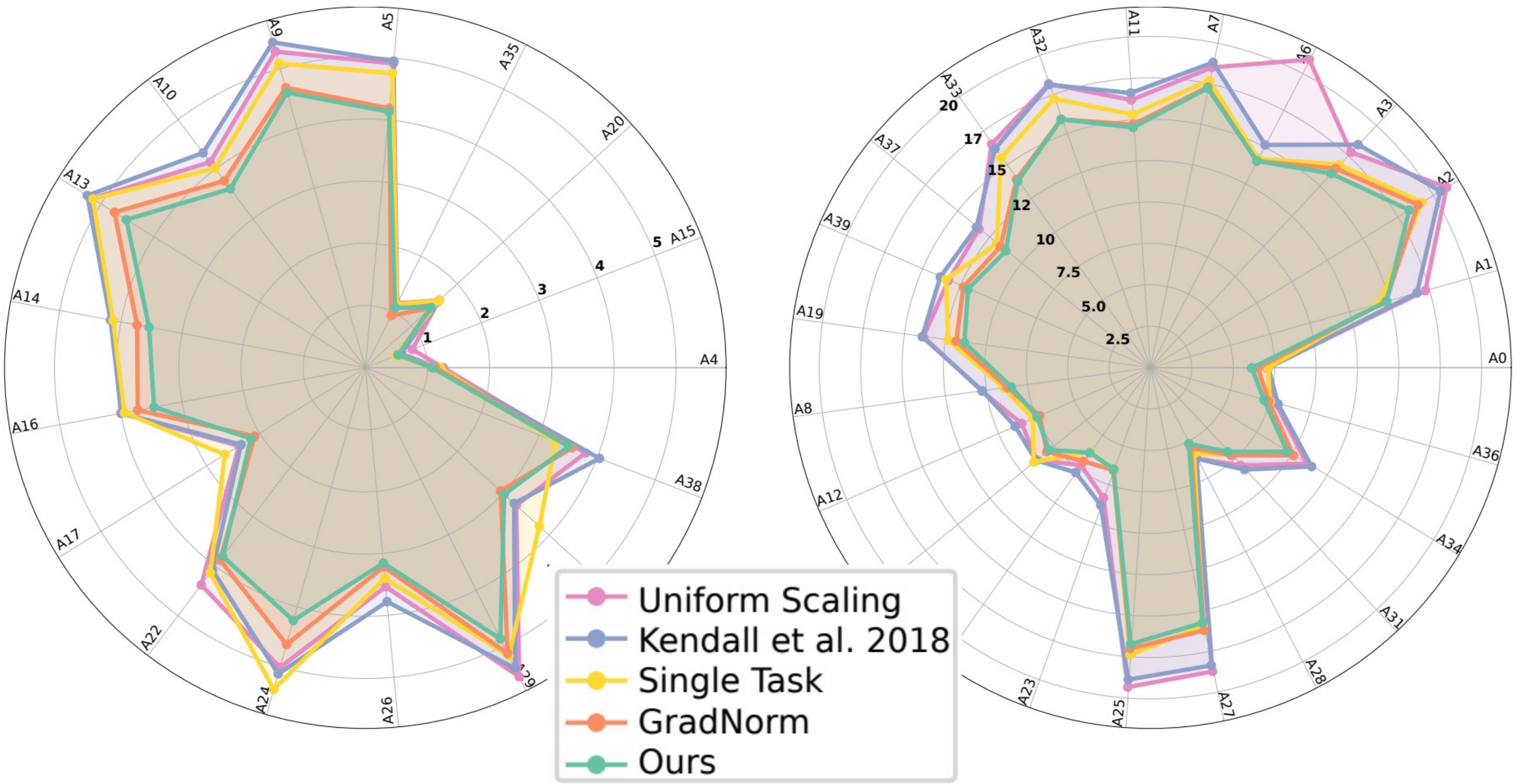
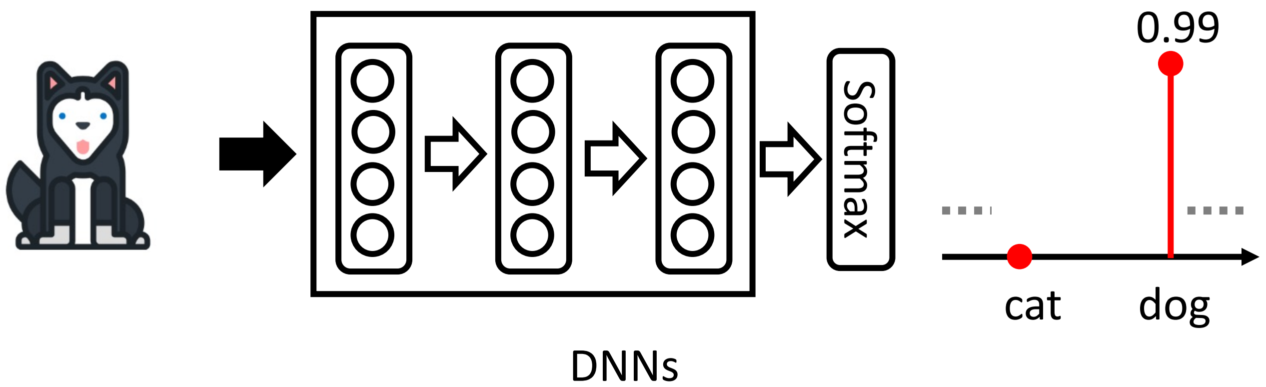
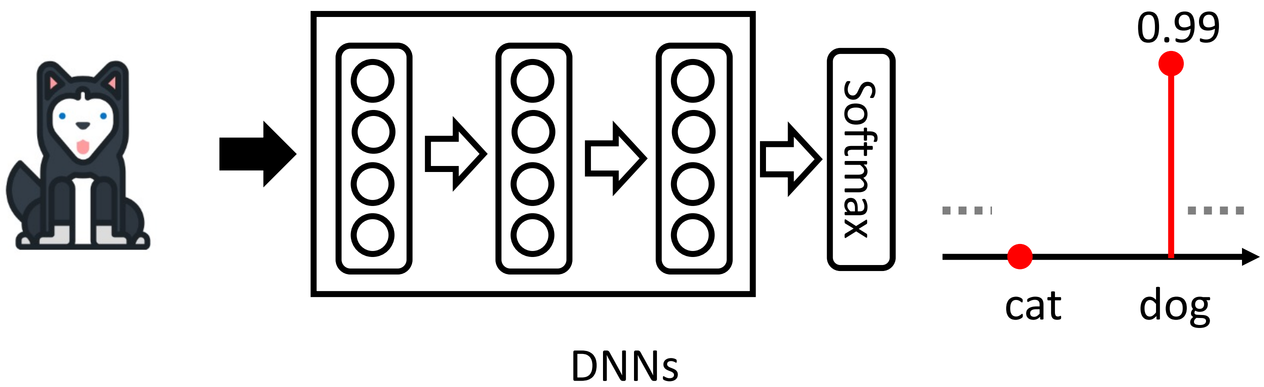


Table of Contents

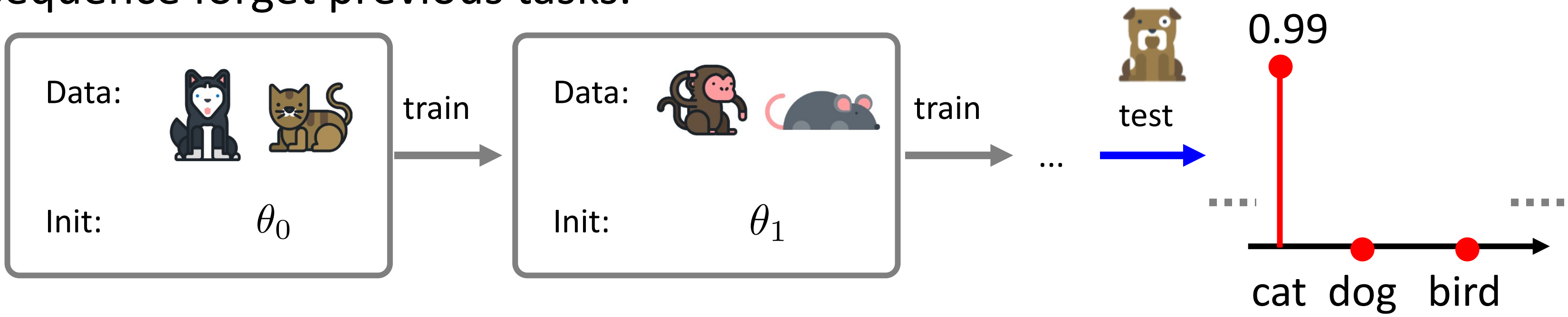
- Deep neural networks (DNNs) can be trained well on a given individual task.
  - E.g., image classifier



- Deep neural networks (DNNs) can be trained well on a given individual task.
    - E.g., image classifier



- **Catastrophic Forgetting/Inference**: DNNs which trained on multiple tasks in sequence forget previous tasks.

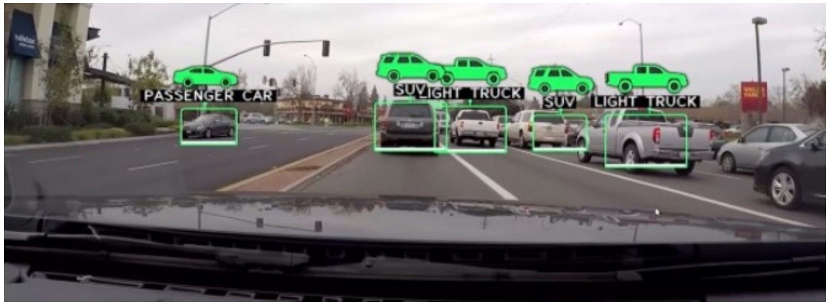


What is Continual Learning?

- Train from scratch with all data of tasks can mitigate forgetting
  - However, it takes too much time to training.
  - Data of the past task may be unavailable.

- Train from scratch with all data of tasks can mitigate forgetting
  - However, it takes too much time to training.
  - Data of the past task may be unavailable.

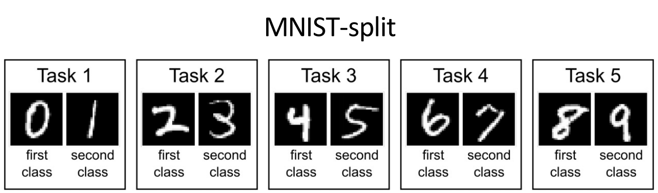
- **Continual Learning**
  - Learn from a non-iid stream of data without catastrophically forgetting the previously learned knowledge.
  - Humans can learn incrementally throughout their lifetime.

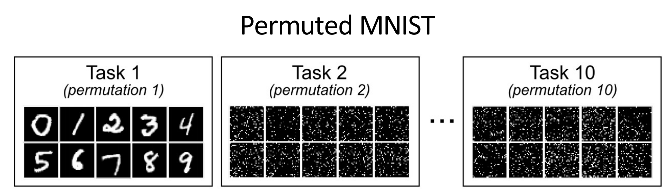


Autonomous drive

Logistics

- Preliminary
  - Common benchmark
    - Split MNIST : the original MNIST is split into disjoint subset(task), where each set consists of two digit classes (a two-way classification).
    - Split CIFAR-10/100: the original CIFAR-10/100 is split into disjoint subset(task), where each set consists of two classes (a two-way classification).
    - Permuted MNIST: MNIST with different random permutation in pixel level, where each task is a ten-way classification.



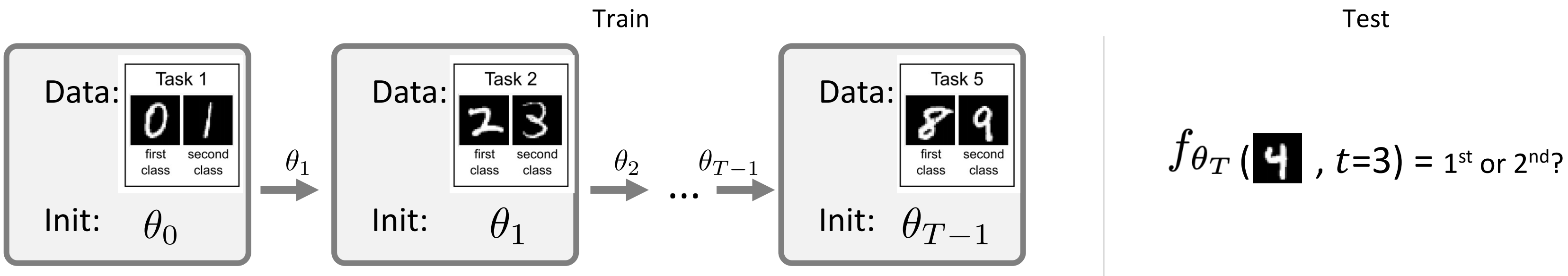


  - Baseline model
    - Fine-tune: trains a model incrementally based on the model parameters learned in the previous stage.

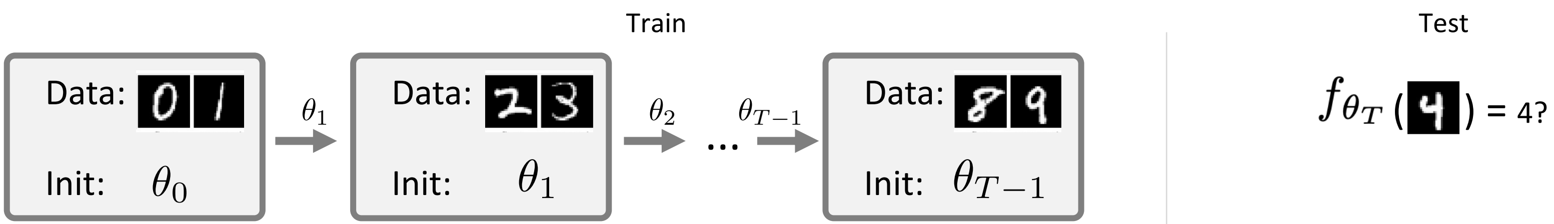
Data: D_1 Init: θ_0	→	Data: D_2 Init: θ_1	→	Data: D_3 Init: θ_2	→ ... →	Data: D_T Init: θ_{T-1}

What is Continual Learning?

- Preliminary
  - Basic continual learning setup
    - Classification tasks are given with the task description t.
    - E.g., for MNIST-Split dataset, is it the 1st or 2nd class with given task description?



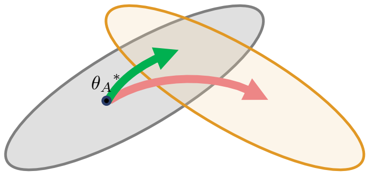
  - Advanced continual learning setup (task-free)
    - No explicit task identifier/boundary information at train/test time.
    - Assume input stream is infinite and non-iid.
    - The data domain may gradually shift without any clear task boundary.
    - Such setups are recently proposed to assume more realistic/practical situation.



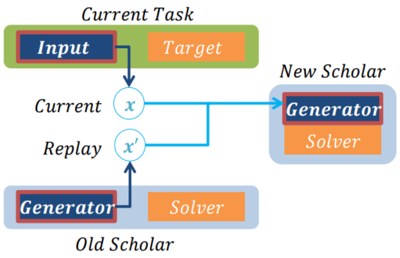
What is Continual Learning?

- How to solve this problem?

- How to solve this problem?
  - **Part 2: Regularization-based Approach**
    - Elastic Weight Consolidation (EWC) [Kirkpatrick, J.,Pascanu, R., et al., 2017]
    - Learning without Forgetting (LwF) [Li et al., 2016]



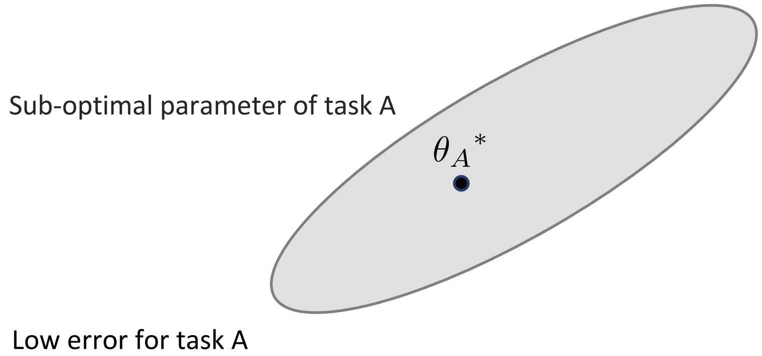
  - **Part 3: Replay-based Approach**
    - ER-Reservoir sampling [Chaudhry et al., 2019]
    - Gradient Episodic Memory (GEM) [Lopez-Paz and Ranzato, 2017]
    - Dark Experience Replay (DER) [Buzzega, 2020]
    - Deep Generative Replay [Shin et al., 2017]



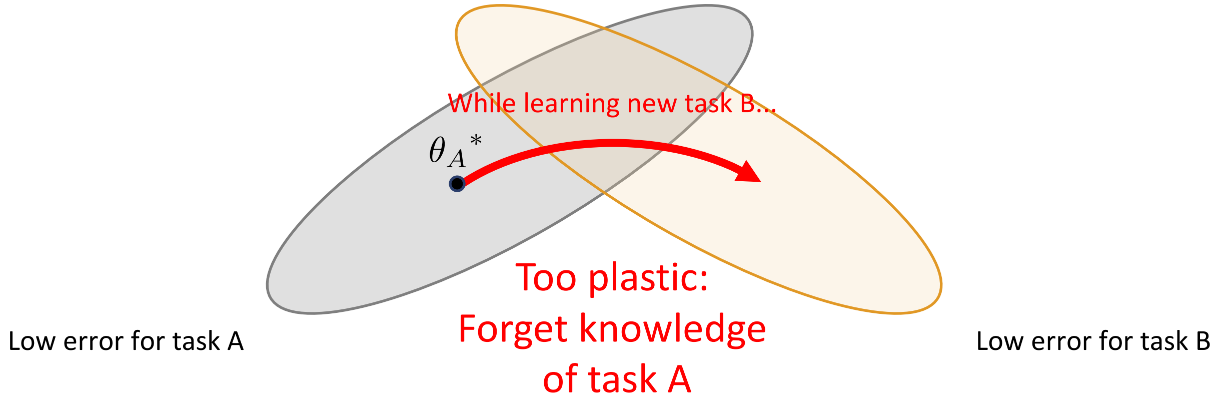
  - **Part 4: Expansion-based Approach**
    - Progressive Neural Network [Rusu and Rabinowitz et al., 2016]
    - Dynamically Expandable Networks (DEN) [Yoon, et a., 2018]

Table of Contents

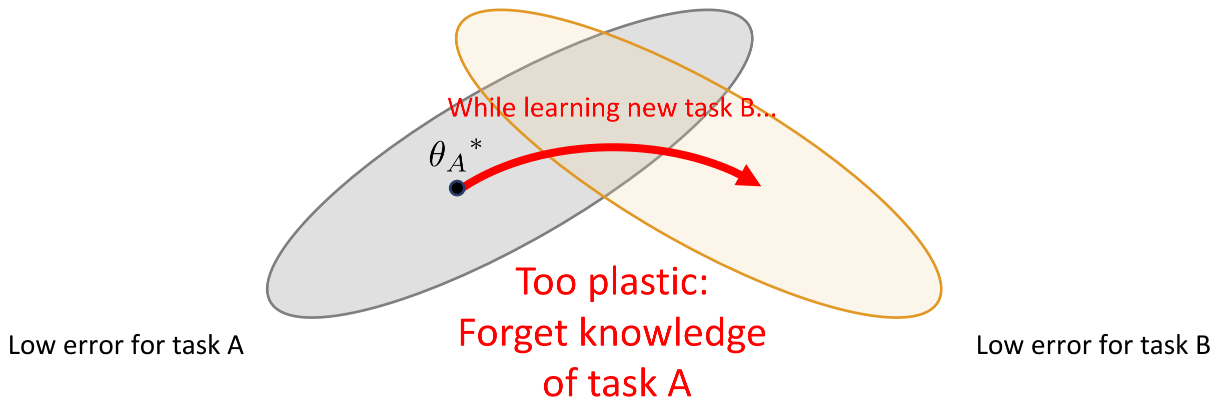
- Continual Learning basically aims to overcome **Plasticity-Stability dilemma.**
  - Balance between network stability (to preserve past knowledge) and plasticity (to rapidly learn the current experience).



- Continual Learning basically aims to overcome **Plasticity-Stability dilemma.**
  - Balance between network stability (to preserve past knowledge) and plasticity (to rapidly learn the current experience).

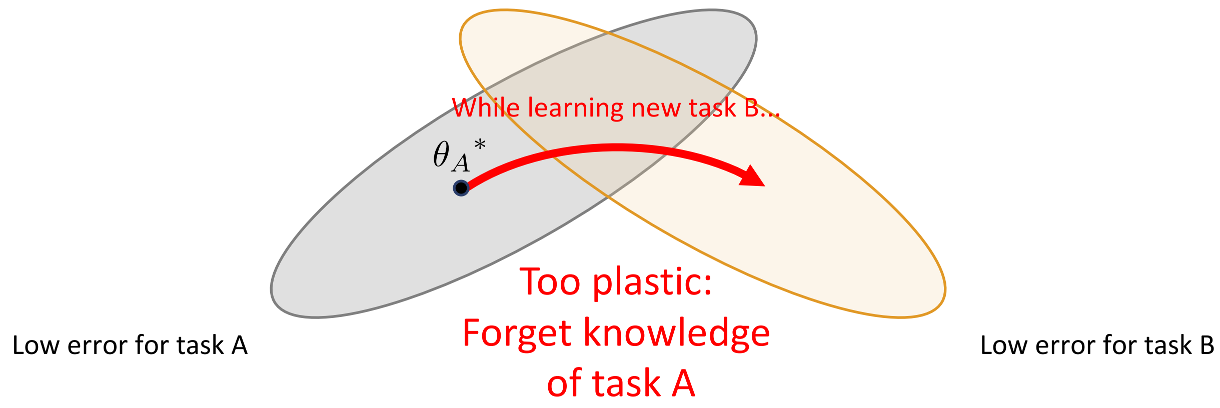


- Continual Learning basically aims to overcome **Plasticity-Stability dilemma.**
  - Balance between network stability (to preserve past knowledge) and plasticity (to rapidly learn the current experience).



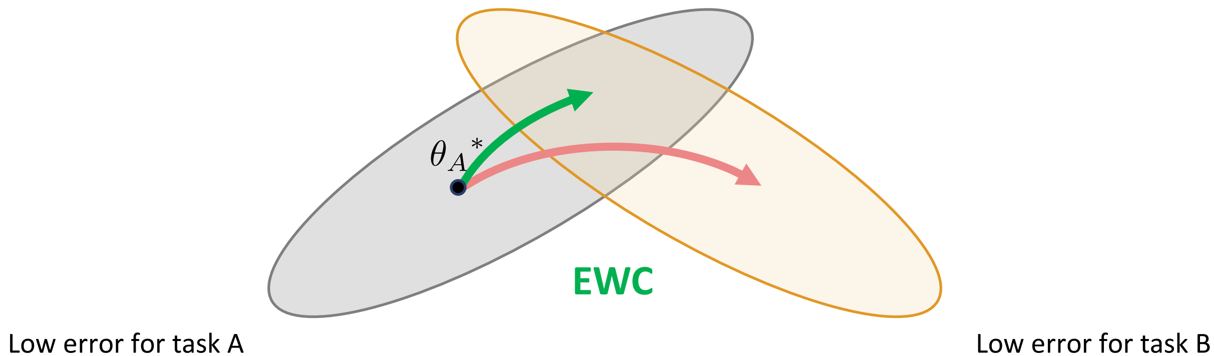
- How to stabilize important parameters for previous tasks and plasticize other parameters to learn new tasks?

- Continual Learning basically aims to overcome **Plasticity-Stability dilemma.**
  - Balance between network stability (to preserve past knowledge) and plasticity (to rapidly learn the current experience).



- How to stabilize important parameters for previous tasks and plasticize other parameters to learn new tasks?
  - **Fisher information** roughly measures the sensitivity of the model's output distribution to small changes in the parameters.
  - *Def (Fisher Information). The negative second derivative of the log likelihood function.* $I(\theta) = -\frac{d^2 l(\theta)}{d\theta^2}$

- [Krikpatrick et al., 2017] **Elastic Weight Consolidation (EWC)**
- Limiting the learning of parameters critical to the performance of past tasks, as measured by the Fisher information matrix (FIM).



Balance Plasticity-Stability

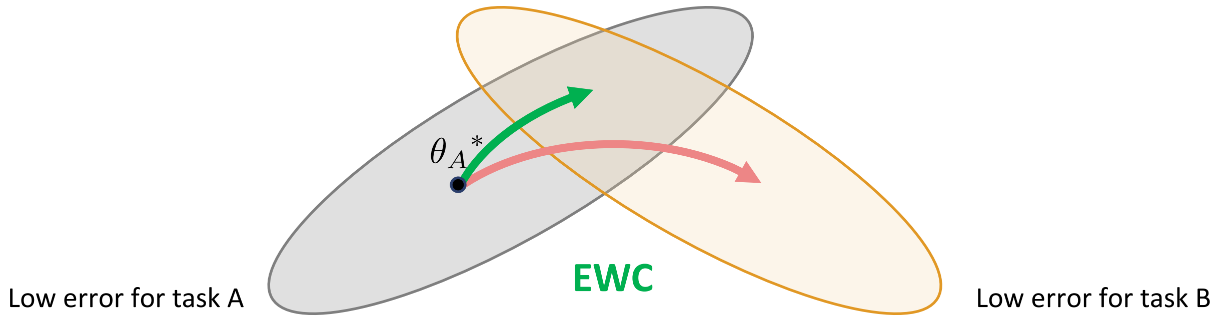
$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta^*_{A,i})^2$$

Current task loss

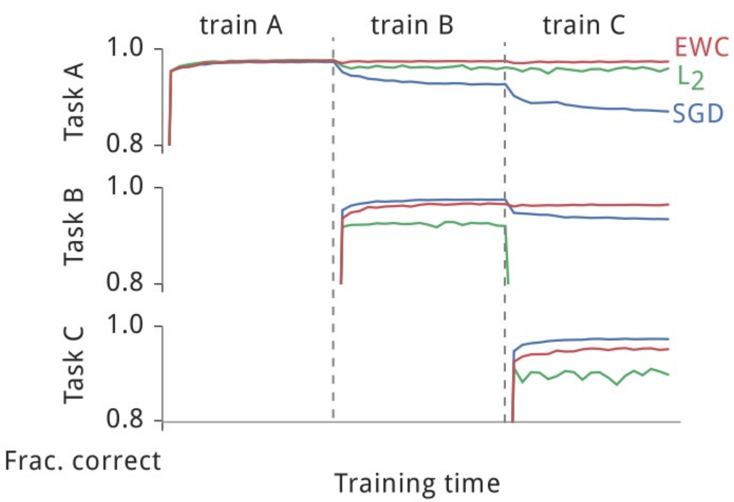
Fisher information matrix (diagonal values)

- Penalizing output changes of model from changes in model parameter can be used as a regularizer!

- [Krikpatrick et al., 2017] **Elastic Weight Consolidation (EWC)**
  - Limiting the learning of parameters critical to the performance of past tasks, as measured by the Fisher information matrix (FIM).

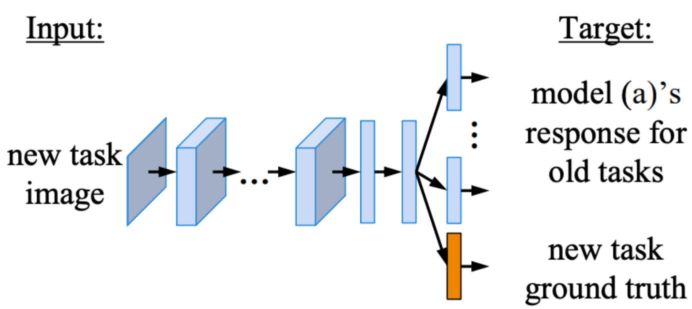


- Results on the permuted MNIST task
  - EWC retains previous tasks' performance.
  - L2 regularized scheme more tends to stabilize on previous task.
  - SGD is too plastic, which results in forgetting previous tasks.



- [Li et al., 2016] **Learning without Forgetting (LwF)**
- Preserve output logit (LwF-logit) of current task samples with the model trained on previous task: **regularize output**

- [Li et al., 2016] **Learning without Forgetting (LwF)**
- Preserve output logit (LwF-logit) of current task samples with the model trained on previous task: **regularize output**
  - Use both {sample, target} and {sample, LwF-logit} pairs to train (multi head).



LEARNINGWITHOUTFORGETTING:

Start with:

θ_s: shared parameters

θ_o: task specific parameters for each old task

X_n, Y_n: training data and ground truth on the new task

Initialize:

$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ *// compute output of old tasks for new data*

$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ *// randomly initialize new parameters*

Train:

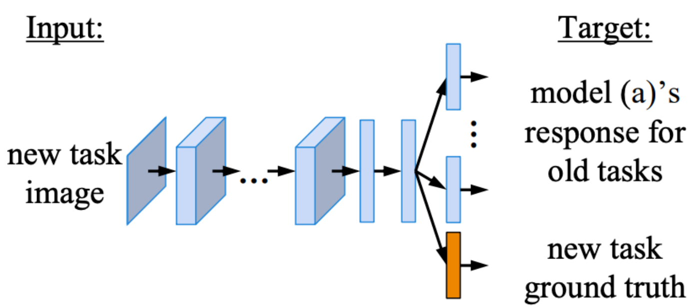
Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ *// old task output*

Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ *// new task output*

$$\theta_s^*,\ \theta_o^*,\ \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$$

Balance Plasticity-Stability

- [Li et al., 2016] **Learning without Forgetting (LwF)**
- Preserve output logit (LwF-logit) of current task samples with the model trained on previous task: **regularize output**
  - Use both {sample, target} and {sample, LwF-logit} pairs to train (multi head).



LEARNINGWITHOUTFORGETTING:
Start with:
θ_s: shared parameters
θ_o: task specific parameters for each old task
X_n, Y_n: training data and ground truth on the new task
Initialize:
$Y_o \leftarrow \text{CNN}(X_n, \theta_s, \theta_o)$ // *compute output of old tasks for new data*
$\theta_n \leftarrow \text{RANDINIT}(|\theta_n|)$ // *randomly initialize new parameters*
Train:
Define $\hat{Y}_o \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_o)$ // *old task output*
Define $\hat{Y}_n \equiv \text{CNN}(X_n, \hat{\theta}_s, \hat{\theta}_n)$ // *new task output*
$\theta_s^*, \theta_o^*, \theta_n^* \leftarrow \underset{\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n}{\operatorname{argmin}} \left(\lambda_o \mathcal{L}_{old}(Y_o, \hat{Y}_o) + \mathcal{L}_{new}(Y_n, \hat{Y}_n) + \mathcal{R}(\hat{\theta}_s, \hat{\theta}_o, \hat{\theta}_n) \right)$

Balance Plasticity-Stability

- Both EWC and LwF regularize changes in trained model's parameters or outputs rather than storing previous tasks' samples to preserve learned knowledge.
  - Control plasticity-stability using a hyperparameter.

Table of Contents

- Continual Learning assumes a particular situation where access to previous data is limited to the current task.

- Continual Learning assumes a particular situation where access to previous data is limited to the current task.
  - What if we can replay some of the previously observed samples?

- **Memory replay**
  - Episodic memory that stores a subset of data can alleviate forgetting.
  - Which samples should be stored in replay memory?
  - How to prevent forgetting while learning new task via utilizing episodic memory?

- Continual Learning assumes a particular situation where access to previous data is limited to the current task.
  - What if we can replay some of the previously observed samples?

- **Memory replay**
  - Episodic memory that stores a subset of data can alleviate forgetting.
  - Which samples should be stored in replay memory?
  - How to prevent forgetting while learning new task via utilizing episodic memory?

- **Generative replay**
  - Pseudo-rehearsal technique: Pseudo-inputs and pseudo-targets produced by a memory network can be fed into the task network.
  - How to generate fake inputs learned from past input distribution and train current task simultaneously?

- Continual Learning assumes a particular situation where access to previous data is limited to the current task.
  - What if we can replay some of the previously observed samples?

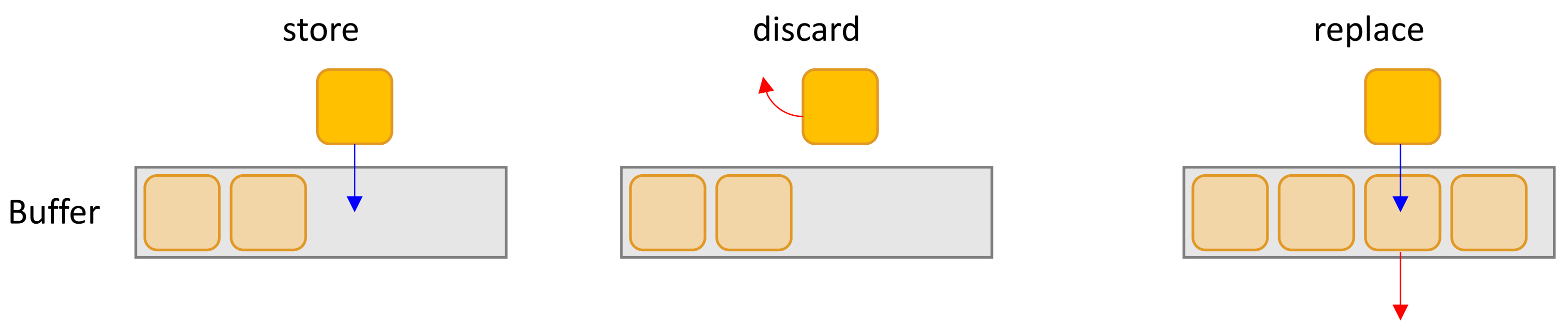
- **Memory replay**
  - Episodic memory that stores a subset of data can alleviate forgetting.
  - **Which samples should be stored in replay memory?**
  - How to prevent forgetting while learning new task via utilizing episodic memory?

- **Generative replay**
  - Pseudo-rehearsal technique: Pseudo-inputs and pseudo-targets produced by a memory network can be fed into the task network.
  - How to generate fake inputs learned from past input distribution and train current task simultaneously?

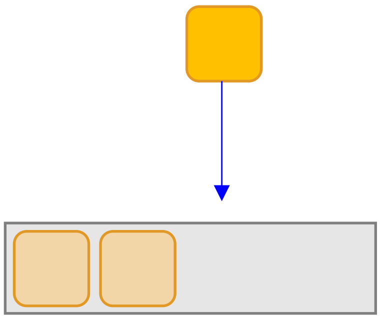
- If one can store samples representative to input distribution, the replayed samples enable us to partially retrieve previous task.
  - Possibly effective to prevent forgetting!

- **Sampling strategy:** How we keep a fixed buffer of size M to be used as a representative of the previous samples?

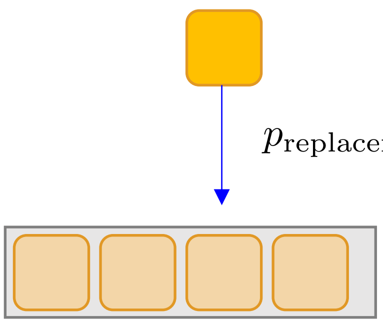
- [Chaudhry et al., 2019] **Reservoir sampling** attempts to keep memory to be representative.
  - Basic operations: samples can be **stored**, **discarded** or **replaced** at every update step.



- [Chaudhry et al., 2019] **Reservoir sampling**
- In the beginning when the buffer is not full, add incoming samples.

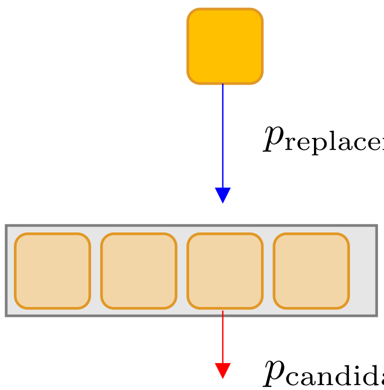


- [Chaudhry et al., 2019] **Reservoir sampling**
- In the beginning when the buffer is not full, add incoming samples.
  - Once the buffer is full, replace current sample with a probability (replay buffer size)/(# examples encountered so far).



$$p_{\text{replacement}} = \frac{\text{(replay buffer size)}}{\text{(\# examples encountered so far)}}$$

- [Chaudhry et al., 2019] **Reservoir sampling**
- In the beginning when the buffer is not full, add incoming samples.
  - Once the buffer is full, replace current sample with a probability (replay buffer size)/(# examples encountered so far).
  - The sample to be replaced in the replay buffer is selected with a uniform distribution.

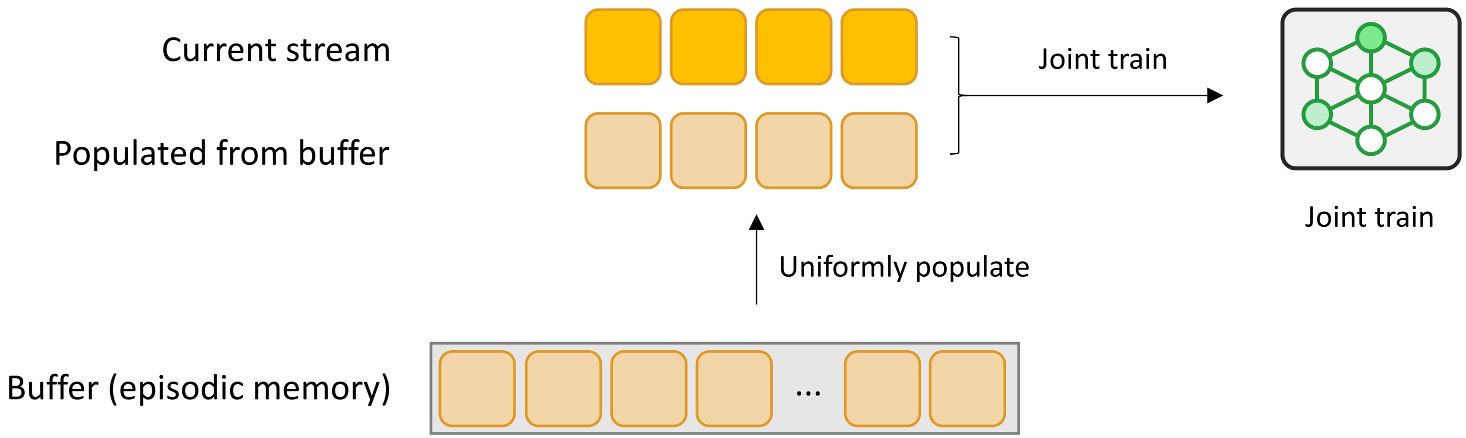


$$p_{\text{replacement}} = \frac{\text{(replay buffer size)}}{\text{(\# examples encountered so far)}}$$

$$p_{\text{candidate}} = \frac{1}{\text{(replay buffer size)}}$$

  - Reservoir sampled instances in replay buffer are representative of the inputs.
    - It also works in the infinite non-iid input stream settings

- [Chaudhry et al., 2019] **Reservoir sampling**
- How to train with Reservoir sampled buffer?



- Populate samples from buffer same sized with batch size and jointly train model.

- Since experience replay with Reservoir sampling is simple yet effective, it is used as a strong baseline for replay-based continual learning studies.

- Continual Learning assumes a particular situation where access to previous data is limited to the current task.
  - What if we can replay some of the previously observed samples?

- **Memory replay**
  - Episodic memory that stores a subset of data can alleviate forgetting.
  - Which samples should be stored in replay memory?
  - **How to prevent forgetting while learning new task via utilizing episodic memory?**

- **Generative replay**
  - Pseudo-rehearsal technique: Pseudo-inputs and pseudo-targets produced by a memory network can be fed into the task network.
  - How to generate fake inputs learned from past input distribution and train current task simultaneously?

- [Lopez-Paz et al., 2017] **Gradient Episodic Memory (GEM)**
- Assume the continuum of data is locally iid.
  - We update parameters on observed triplet (x, t, y) where (x, y) is a pair of input-target and t is task identifier.
  - Prevent forgetting by optimizing networks on observed triplet only allowed to decrease loss on populated samples from memory.
    - We define average loss on samples from memory

$$\ell\left(f_\theta, \mathcal{M}_k\right) = \frac{1}{|\mathcal{M}_k|} \sum_{(x_i, k, y_i) \in \mathcal{M}_k} \ell\left(f_\theta\left(x_i, k\right), y_i\right)$$

k^th task memory

    - Then, we optimize parameters in what follows

the predictor state at the end of learning task t − 1

$$\begin{aligned}&\text{minimize}_\theta \ell\left(f_\theta(x, t), y\right) \\ &\text{subject to } \ell\left(f_\theta, \mathcal{M}_k\right) \leq \ell\left(f_\theta^{t-1}, \mathcal{M}_k\right) \text{ for all } k < t\end{aligned}$$

k^th task memory

  - We store trained triplets in fixed size memory in FIFO(first in first out) manner.

- [Lopez-Paz et al., 2017] **Gradient Episodic Memory (GEM)**
- Optimization rephrasing : **the gradients of past and current task should be aligned**.

the predictor state at the end of learning of task t – 1

$$\text{minimize}_\theta \ell\left(f_\theta(x,t),y\right)$$
$$\text{subject to } \ell\left(f_\theta,\mathcal{M}_k\right)\le\ell\left(f_\theta^{t-1},\mathcal{M}_k\right) \text{ for all } k<t$$

k^th task memory

Rephrased

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

Parameter update on observed triplet (x,t,y).

  - If satisfied, the gradient g is unlikely to increase the loss at previous tasks
  - If not satisfied, at least one previous task's loss is likely to increase after updating parameter on direction to g.
- If above products are negative, project g to the closest gradient $\tilde{g}$ satisfying positive transfer.

$$\text{minimize}_{\tilde{g}} \frac{1}{2}\|g-\tilde{g}\|_2^2$$
$$\text{subject to } \langle\tilde{g},g_k\rangle\ge 0 \text{ for all } k<t$$

- [Aljundi et al., 2019] **Gradient based Sample Selection (GSS)**
  - Maximize the variance of the stored memories with respect to the gradient direction of the model update they would generate.

- [Aljundi et al., 2019] **Gradient based Sample Selection (GSS)**
  - Maximize the variance of the stored memories with respect to the gradient direction of the model update they would generate.

  - The feasible set following constraints of GEM can decrease when the angles between each pair of gradients increase.

$$\text{minimize}_{\mathcal{M}} \sum_{i,j \in \mathcal{M}} \frac{\langle g_i, g_j \rangle}{\|g_i\| \, \|g_j\|}$$

- [Aljundi et al., 2019] **Gradient based Sample Selection (GSS)**
  - Maximize the variance of the stored memories with respect to the gradient direction of the model update they would generate.

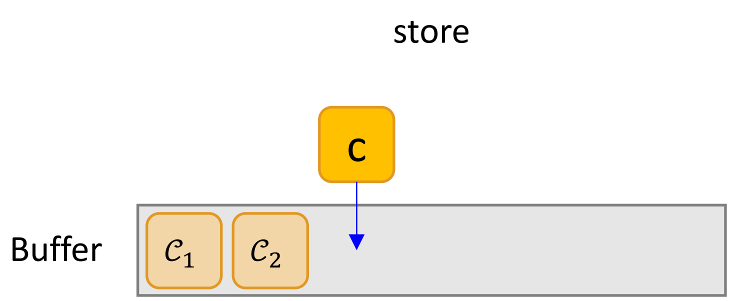
  - The feasible set following constraints of GEM can decrease when the angles between each pair of gradients increase.

$$\text{minimize}_{\mathcal{M}} \sum_{i,j \in \mathcal{M}} \frac{\langle g_i, g_j \rangle}{\|g_i\| \, \|g_j\|}$$

  - Such minimization is equivalent to maximization of the variance of the gradient direction.

$$\begin{aligned} \mathrm{Var}_{\mathcal{M}}\left[\frac{g}{\|g\|}\right] &= \frac{1}{M} \sum_{k \in \mathcal{M}} \left\| \frac{g}{\|g\|} \right\|^2 - \left\| \frac{1}{M} \sum_{k \in \mathcal{M}} \frac{g}{\|g\|} \right\|^2 \\ &= 1 - \frac{1}{M^2} \sum_{i,j \in \mathcal{M}} \frac{\langle g_i, g_j \rangle}{\|g_i\| \, \|g_j\|} \end{aligned}$$

- [Aljundi et al., 2019] **Greedy alternative for Gradient based Sample Selection (GSS-Greedy)**
  - In the beginning when the buffer is not full, add incoming samples and its *score(c)* to the replay buffer.

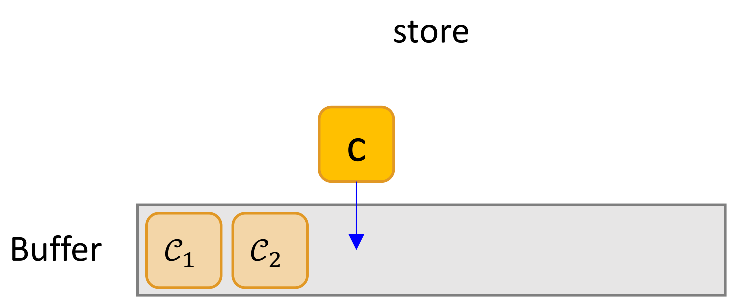


- [Aljundi et al., 2019] **Greedy alternative for Gradient based Sample Selection (GSS-Greedy)**
  - In the beginning when the buffer is not full, add incoming samples and its *score(c)* to the replay buffer.

$$g \leftarrow \nabla \ell_\theta(x, y); G \leftarrow \nabla_\theta \ell(X, Y)$$

$$c = \max_i\left(\frac{\langle g, G_i \rangle}{\|g\| \|G_i\|}\right) + 1$$

The maximal cosine similarity of the current sample with a fixed number of other random samples in the buffer.

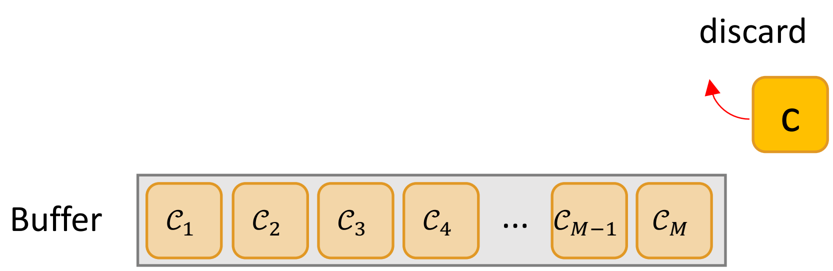


- [Aljundi et al., 2019] **Greedy alternative for Gradient based Sample Selection (GSS-Greedy)**
  - Once the buffer is full and the current sample's similarity is positive (score c >= 1), discard and receive incoming sample.

$$g \leftarrow \nabla \ell_\theta(x, y);\, G \leftarrow \nabla_\theta \ell(X, Y)$$

$$c = \max_i\left(\frac{\langle g, G_i \rangle}{\|g\|\|G_i\|}\right) + 1$$

The maximal cosine similarity of the current sample with a fixed number of other random samples in the buffer.



$$c = \max_i\left(\frac{\langle g, G_i \rangle}{\|g\|\|G_i\|}\right) + 1$$

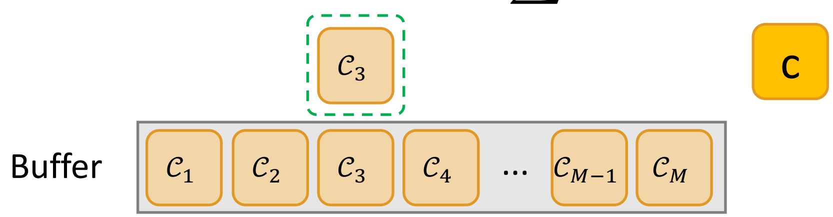
- [Aljundi et al., 2019] **Greedy alternative for Gradient based Sample Selection (GSS-Greedy)**
  - Once the buffer is full and the current sample's similarity is negative (score c < 1), randomly select a sample(i) with the normalized score as the probability to be replaced.

$$g \leftarrow \nabla \ell_\theta(x, y);\, G \leftarrow \nabla_\theta \ell(X, Y)$$

$$c = \max_i\left(\frac{\langle g, G_i\rangle}{\|g\|\|G_i\|}\right) + 1$$

The maximal cosine similarity of the current sample with a fixed number of other random samples in the buffer.

The drawn sample $i \sim P(i) = \mathcal{C}_i / \sum \mathcal{C}_j$



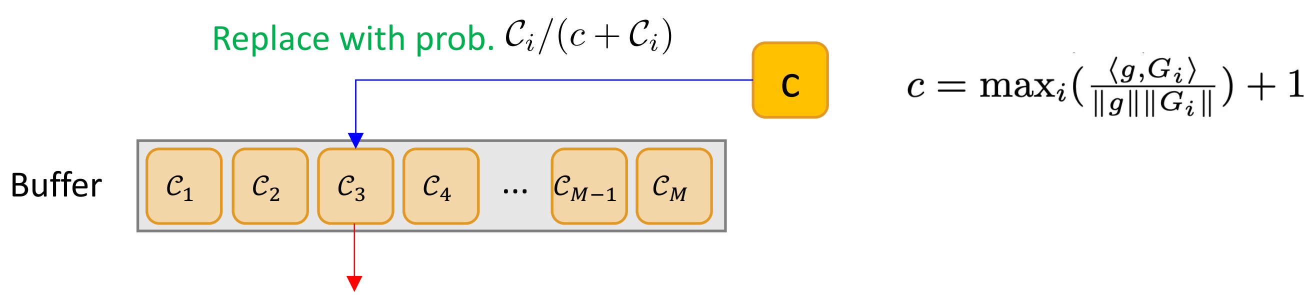
$$c = \max_i\left(\frac{\langle g, G_i\rangle}{\|g\|\|G_i\|}\right) + 1$$

- [Aljundi et al., 2019] **Greedy alternative for Gradient based Sample Selection (GSS-Greedy)**
  - Once the buffer is full and the current sample's similarity is negative (score c < 1), randomly select a sample(i) with the normalized score as the probability to be replaced.
  - **Replace** the drawn sample(i) from buffer with probability $\mathcal{C}_i/(c+\mathcal{C}_i)$.

$$g \leftarrow \nabla \ell_\theta(x, y);\ G \leftarrow \nabla_\theta \ell(X, Y)$$

$$c = \max_i\left(\frac{\langle g, G_i\rangle}{\|g\|\|G_i\|}\right) + 1$$

The maximal cosine similarity of the current sample with a fixed number of other random samples in the buffer.



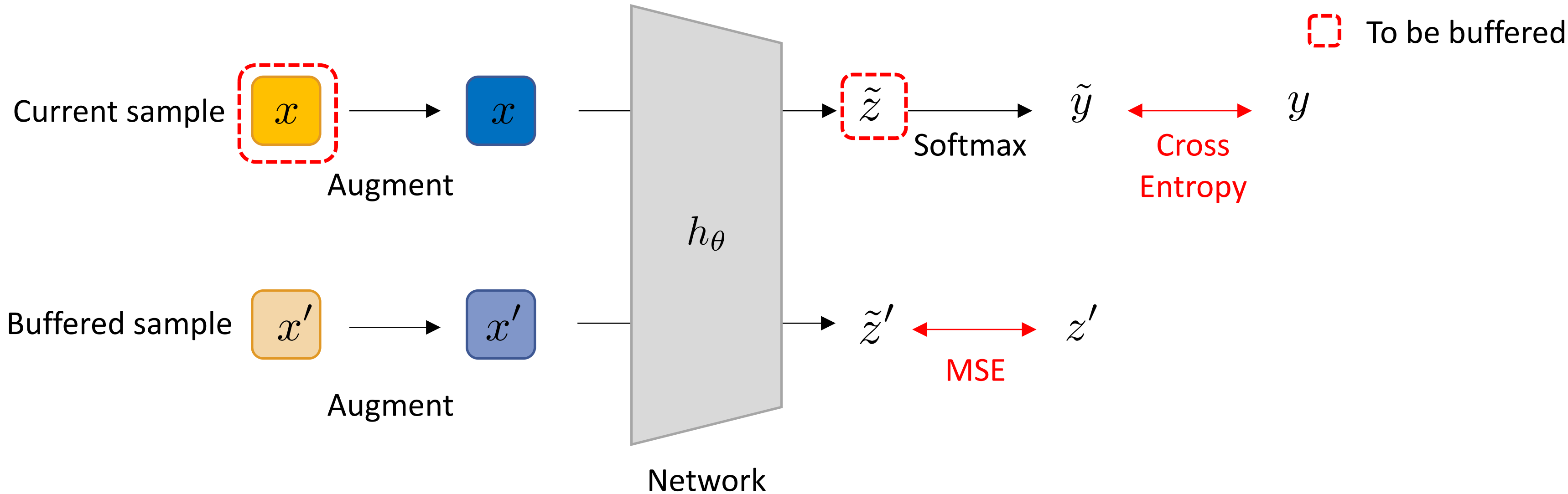
- Experimental results
  - Baselines
    - Rand: randomly select M samples from incoming and stored memories.
    - GSS-Clust / FSS-Clust: M centroid cluster-based selection either in the feature space (FSS) or the gradient space (GSS).
    - GSS-IQP: an integer quadratic programming solver for GSS (MNIST only).
    - GSS-Greedy: a greedy alternative for GSS.
  - MNIST divided into 5 tasks with two labels in each.

Method \ Buffer Size	300	400	500
Rand	37.5 ± 1.3	45.9 ± 4.8	57.9 ± 4.1
GSS-IQP(ours)	75.9 ± 2.5	82.1 ± 0.6	84.1 ± 2.4
GSS-Clust	75.7 ± 2.2	81.4 ± 4.4	83.9 ± 1.6
FSS-Clust	75.8 ± 1.7	80.6 ± 2.7	83.4 ± 2.6
GSS-Greedy(ours)	$\mathbf{82.6 \pm 2.9}$	$\mathbf{84.6 \pm 0.9}$	$\mathbf{84.8 \pm 1.8}$

  - CIFAR10 divided into 5 tasks with two labels in each.

Method	T1	T2	T3	T4	T5	Avg
Rand	0 ± 0.0	0.49 ± 0.4	5.68 ± 4.4	52.18 ± 0.8	84.96 ± 4.4	28.6 ± 1.2
GSS-Clust	0.35 ± 0.5	15.27 ± 8.3	7.96 ± 6.3	9.97 ± 2.1	77.83 ± 0.7	22.5 ± 0.4
FSS-Clust	0.2 ± 0.2	0.8 ± 0.5	5.4 ± 0.7	38.12 ± 5.2	87.90 ± 3.1	26.7 ± 1.5
GSS-Greedy(ours)	42.36 ± 12.1	14.61 ± 2.7	13.60 ± 4.5	19.30 ± 2.7	$77.83 \pm 4.\ 2$	$\mathbf{33.56 \pm 1.7}$

- [Buzzega et al., 2020] **Dark Experience Replay (DER)**
- Encourage the network to mimic its original responses for past samples.
  - Logit matching: retain the network's logits instead of the ground truth labels.
  - Similar to previous replay-based methods, DER also looks for parameters that fit the current task well while approximating the behavior observed in the old ones.
  - However, DER does not approximate past behaviors in gradient spaces.



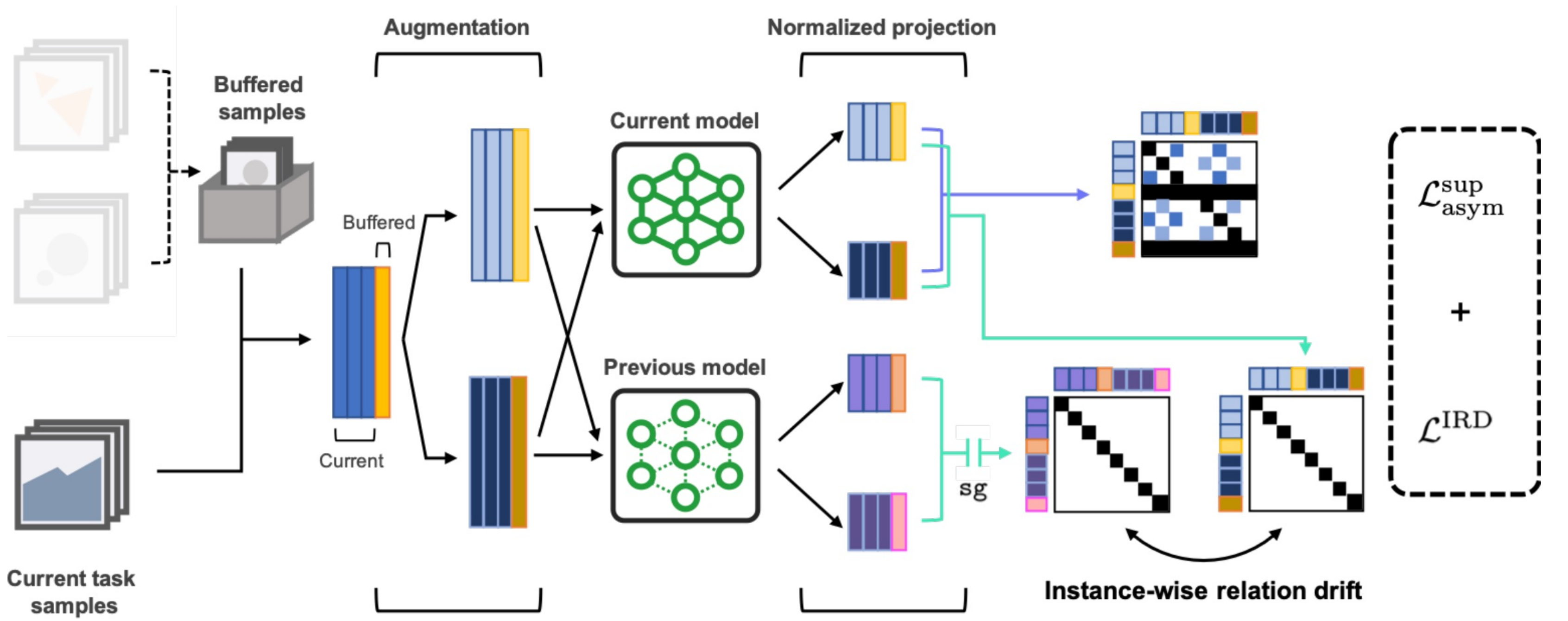
$$\mathcal{L}_{t_c} + \alpha \, \mathbb{E}_{(x,z)\sim\mathcal{M}}\left[\|z - h_\theta(x)\|_2^2 \right]$$

- [Buzzega et al., 2020] **Dark Experience Replay (DER)**

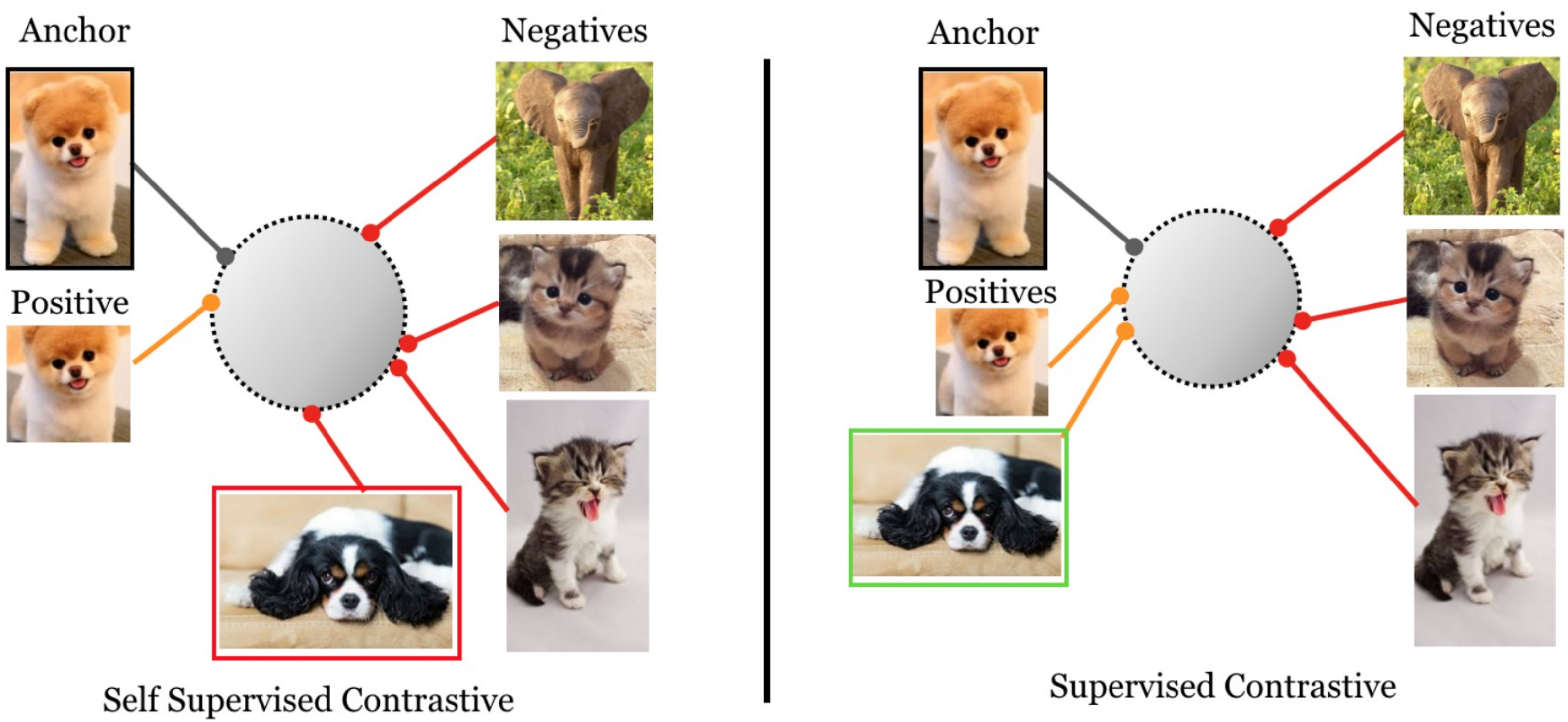
Buffer	Method	S-CIFAR-10 Class-IL	S-CIFAR-10 Task-IL	S-Tiny-ImageNet Class-IL	S-Tiny-ImageNet Task-IL	P-MNIST Domain-IL	R-MNIST Domain-IL
–	JOINT	92.20±0.15	98.31±0.12	59.99±0.19	82.04±0.10	94.33±0.17	95.76±0.04
	SGD	19.62±0.05	61.02±3.33	7.92±0.26	18.31±0.68	40.70±2.33	67.66±8.53
–	oEWC [36]	19.49±0.12	68.29±3.92	7.58±0.10	19.20±0.31	**75.79±2.25**	**77.35±5.77**
	SI [42]	19.48±0.17	68.05±5.91	6.58±0.31	36.32±0.13	65.86±1.57	71.91±5.83
	LwF [24]	**19.61±0.05**	63.29±2.35	**8.46±0.22**	15.85±0.58	-	-
	PNN [35]	-	**95.13±0.72**	-	**67.84±0.29**	-	-
200	ER [33]	44.79±1.86	91.19±0.94	8.49±0.16	38.17±2.00	72.37±0.87	85.01±1.90
	GEM [27]	25.54±0.76	90.44±0.94	-	-	66.93±1.25	80.80±1.15
	A-GEM [9]	20.04±0.34	83.88±1.49	8.07±0.08	22.77±0.03	66.42±4.00	81.91±0.76
	iCaRL [32]	49.02±3.20	88.99±2.13	7.53±0.79	28.19±1.47	-	-
	FDR [4]	30.91±2.74	91.01±0.68	8.70±0.19	40.36±0.68	74.77±0.83	85.22±3.35
	GSS [1]	39.07±5.59	88.80±2.89	-	-	63.72±0.70	79.50±0.41
	HAL [8]	32.36±2.70	82.51±3.20	-	-	74.15±1.65	84.02±0.98
	DER (ours)	61.93±1.79	91.40±0.92	**11.87±0.78**	40.22±0.67	81.74±1.07	90.04±2.61
	DER++ (ours)	**64.88±1.17**	**91.92±0.60**	10.96±1.17	**40.87±1.16**	**83.58±0.59**	**90.43±1.87**
500	ER [33]	57.74±0.27	93.61±0.27	9.99±0.29	48.64±0.46	80.60±0.86	88.91±1.44
	GEM [27]	26.20±1.26	92.16±0.69	-	-	76.88±0.52	81.15±1.98
	A-GEM [9]	22.67±0.57	89.48±1.45	8.06±0.04	25.33±0.49	67.56±1.28	80.31±6.29
	iCaRL [32]	47.55±3.95	88.22±2.62	9.38±1.53	31.55±3.27	-	-
	FDR [4]	28.71±3.23	93.29±0.59	10.54±0.21	49.88±0.71	83.18±0.53	89.67±1.63
	GSS [1]	49.73±4.78	91.02±1.57	-	-	76.00±0.87	81.58±0.58
	HAL [8]	41.79±4.46	84.54±2.36	-	-	80.13±0.49	85.00±0.96
	DER (ours)	70.51±1.67	93.40±0.39	17.75±1.14	51.78±0.88	87.29±0.46	92.24±1.12
	DER++ (ours)	**72.70±1.36**	**93.88±0.50**	**19.38±1.41**	**51.91±0.68**	**88.21±0.39**	**92.77±1.05**
5120	ER [33]	82.47±0.52	**96.98±0.17**	27.40±0.31	67.29±0.23	89.90±0.13	93.45±0.56
	GEM [27]	25.26±3.46	95.55±0.02	-	-	87.42±0.95	88.57±0.40
	A-GEM [9]	21.99±2.29	90.10±2.09	7.96±0.13	26.22±0.65	73.32±1.12	80.18±5.52
	iCaRL [32]	55.07±1.55	92.23±0.84	14.08±1.92	40.83±3.11	-	-
	FDR [4]	19.70±0.07	94.32±0.97	28.97±0.41	68.01±0.42	90.87±0.16	94.19±0.44
	GSS [1]	67.27±4.27	94.19±1.15	-	-	82.22±1.14	85.24±0.59
	HAL [8]	59.12±4.41	88.51±3.32	-	-	89.20±0.14	91.17±0.31
	DER (ours)	83.81±0.33	95.43±0.33	36.73±0.64	69.50±0.26	91.66±0.11	94.14±0.31
	DER++ (ours)	**85.24±0.49**	96.12±0.21	**39.02±0.97**	**69.84±0.63**	**92.26±0.17**	**94.65±0.33**

- Despite its simplicity, DER/DER++ outperform most of CL baselines in various scenarios.
  - DER++ additionally populates and utilizes ground truth labels (y).
- The source of its greatness is not fully explained yet.
  - There is still much room for improvement!

- Recent progress in **contrastive representation learning** shows superior performance in various downstream tasks and domains
- [Cha et al., 2021] **Contrastive Continual Learning (Co^2L)** leverages this learning scheme for continual learning
  - CO^2L *learns* the representations with supervised contrastive loss
  - CO^2L *preserves* learned representations using self-supervised distillation

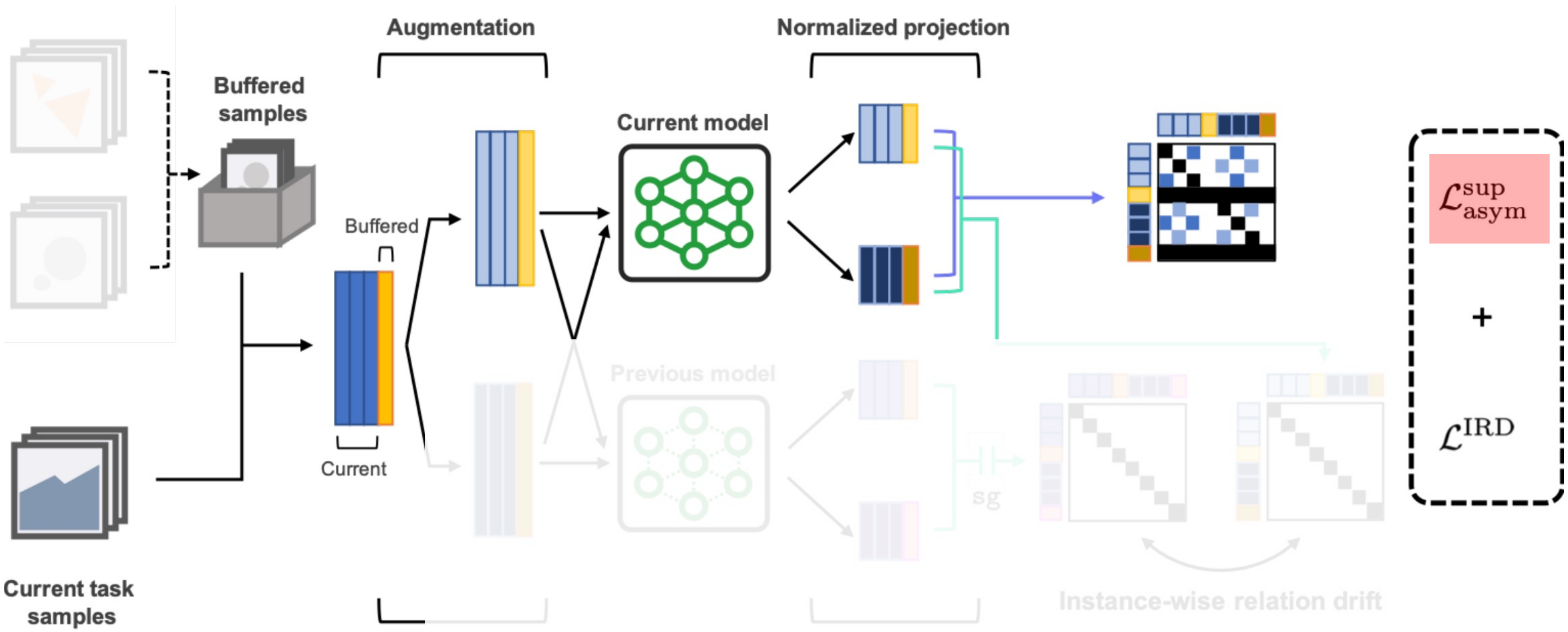


- **Fast recap of contrastive learning**
  - The **self-supervised contrastive loss** (left) contrasts a single positive for each anchor (i.e., an augmented version of the same image) against a set of negatives consisting of the entire remainder of the batch
  - The **supervised contrastive loss** (right) contrasts the set of all samples from the same class as positives against the negatives from the remainder of the batch



- [Cha et al., 2021] **Contrastive Continual Learning (Co²L)**
  - CO²L *learns* the representations with **supervised contrastive loss**
    - The loss is modified to the **asymmetric version**, to prevent a model from overfitting to a small number of past task samples.
      - current task samples as anchors; past samples from the buffer as negative

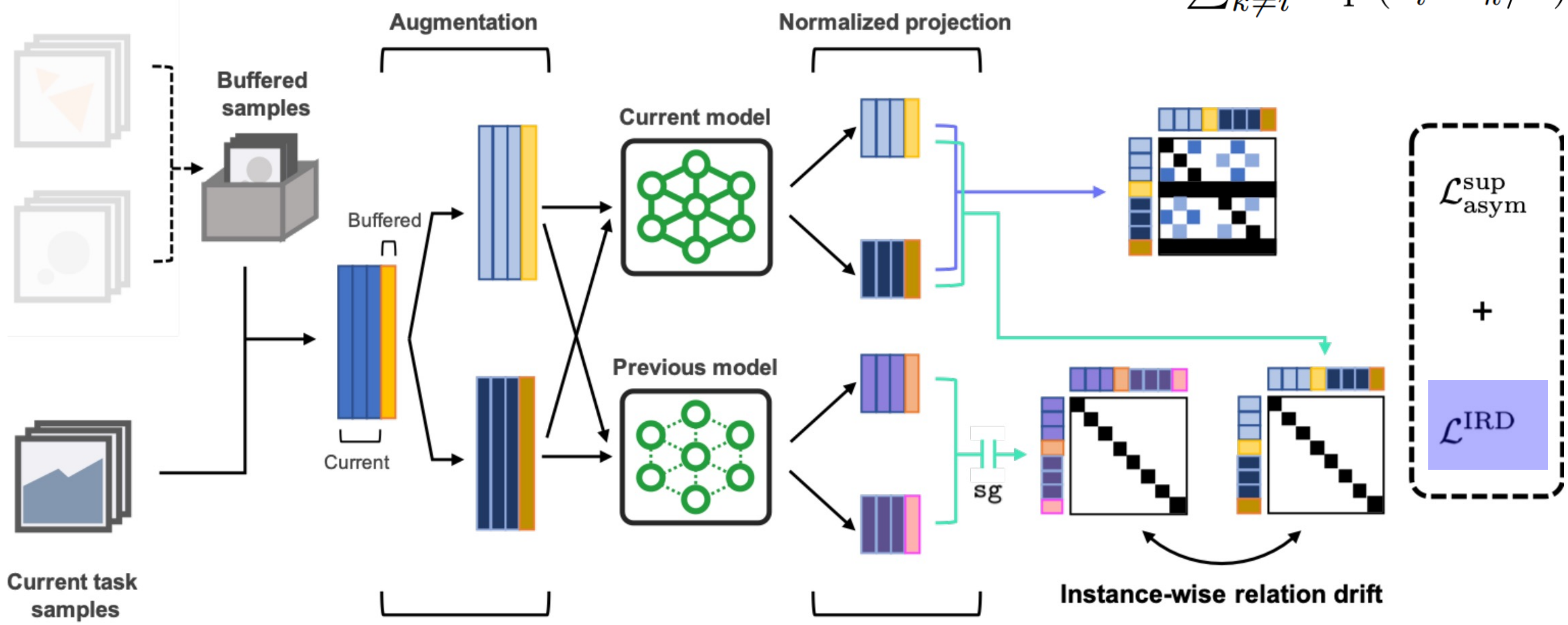
$$\mathcal{L}_{\mathrm{asym}}^{\mathrm{sup}} = \sum_{i \in S} \frac{-1}{|\mathfrak{p}_i|} \sum_{p \in \mathfrak{p}_i} \log \frac{\exp\left(\mathbf{z}_i \cdot \mathbf{z}_p / \tau\right)}{\sum_{k \neq i} \exp\left(\mathbf{z}_i \cdot \mathbf{z}_k / \tau\right)}$$



- [Cha et al., 2021] **Contrastive Continual Learning (Co²L)**
  - CO²L *preserves* learned representations using **self-supervised distillation**
    - **Instance-wise Relation Distillation (IRD)** loss regulates the changes in feature relation between batch samples via self-distillation

$$\mathcal{L}^{\mathrm{IRD}} = \sum_{i=1}^{2N} -\mathbf{p}\left(\tilde{\mathbf{x}}_i; \psi^{\mathrm{past}}, \kappa^*\right) \cdot \log \mathbf{p}\left(\tilde{\mathbf{x}}_i; \psi, \kappa\right)$$

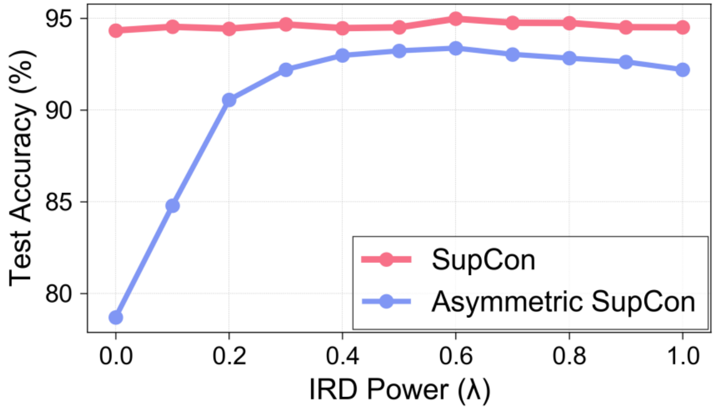
where $\mathbf{p}\left(\tilde{\mathbf{x}}_i; \psi, \kappa\right) = \left[p_{i,1}, \ldots, p_{i,i-1}, p_{i,i+1}, \ldots, p_{i,2N}\right], \; p_{i,j} = \dfrac{\exp\left(\mathbf{z}_i \cdot \mathbf{z}_j / \kappa\right)}{\sum_{k \neq i}^{2N} \exp\left(\mathbf{z}_i \cdot \mathbf{z}_k / \kappa\right)}$



- CO^2L consistently outperforms all memory replay-based continual learning baselines in various scenarios

Buffer	Dataset	Seq-CIFAR-10		Seq-Tiny-ImageNet		R-MNIST
	Scenario	Class-IL	Task-IL	Class-IL	Task-IL	Domain-IL
200	ER [32]	44.79±1.86	91.19±0.94	8.49±0.16	38.17±2.00	93.53±1.15
	GEM [28]	25.54±0.76	90.44±0.94	-	-	89.86±1.23
	A-GEM [8]	20.04±0.34	83.88±1.49	8.07±0.08	22.77±0.03	89.03±2.76
	iCaRL [31]	49.02±3.20	88.99±2.13	7.53±0.79	28.19±1.47	-
	FDR [4]	30.91±2.74	91.01±0.68	8.70±0.19	40.36±0.68	93.71±1.51
	GSS [2]	39.07±5.59	88.80±2.89	-	-	87.10±7.23
	HAL [7]	32.36±2.70	82.51±3.20	-	-	89.40±2.50
	DER [5]	61.93±1.79	91.40±0.92	11.87±0.78	40.22±0.67	96.43±0.59
	DER++ [5]	64.88±1.17	91.92±0.60	10.96±1.17	40.87±1.16	95.98±1.06
	Co^2L (ours)	**65.57**±1.37	**93.43**±0.78	**13.88**±0.40	**42.37**±0.74	**97.90**±1.92

- The proposed asymmetric contrastive loss is more effective than the original SupCon loss



	Seq-CIFAR-10		Seq-Tiny-ImageNet	
Buffer	200	500	200	500
$\mathcal{L}^{sup}$	60.49±0.72	68.66±0.68	13.51±0.48	19.68±0.62
$\mathcal{L}^{sup}_{asym}$	**65.57**±1.37	**74.26**±0.77	**13.88**±0.40	**20.12**±0.42

- Continual Learning assumes a particular situation where access to previous data is limited to the current task.
    - What if we can replay some of the previously observed samples?

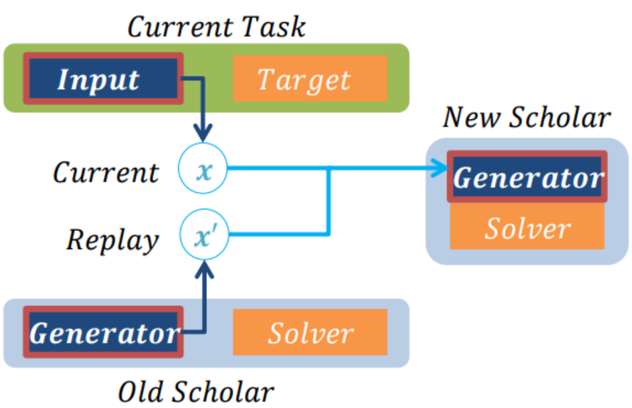
- **Memory replay**
    - Episodic memory that stores a subset of data can alleviate forgetting.
    - How to utilize episodic memory to prevent forgetting while learning new task?
    - Which samples should be stored in replay memory?

- **Generative replay**
    - Pseudo-rehearsal technique: Pseudo-inputs and pseudo-targets produced by a memory network can be fed into the task network.
    - How to generate fake inputs learned from past input distribution and train current task simultaneously?

- [Shin et al., 2017] **Deep Generative Replay**
- A cooperative **dual model architecture** consisting of a deep generative model ("generator") and a task solving model ("solver") to retains the knowledge without revisiting actual past data.

- [Shin et al., 2017] **Deep Generative Replay**
- A cooperative **dual model architecture** consisting of a deep generative model ("generator") and a task solving model ("solver") to retains the knowledge without revisiting actual past data.

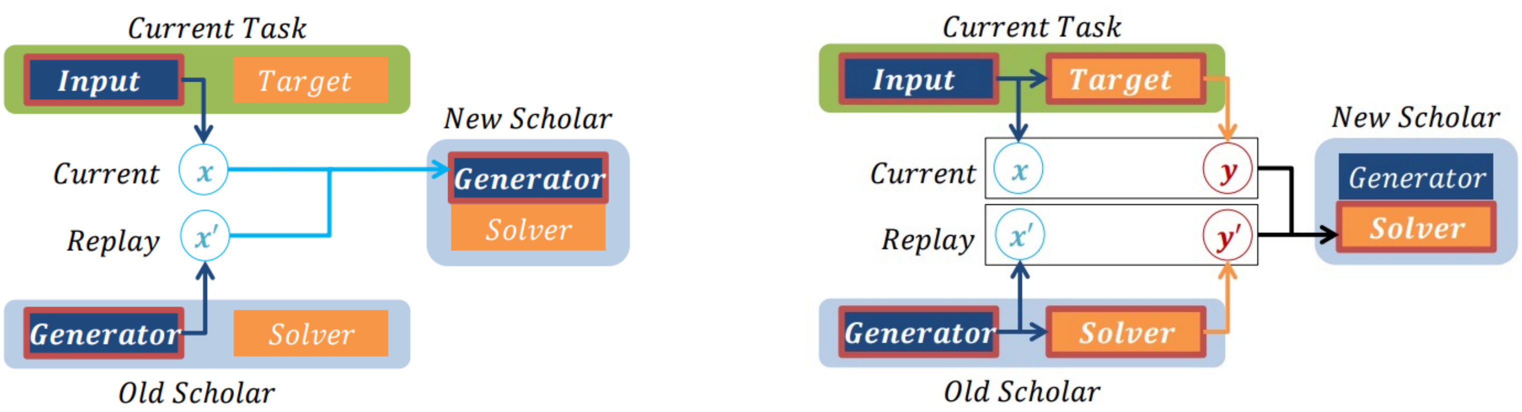
  - **Generator** is sequentially trained to generate pseudo-input from current task inputs and generated inputs from *old scholar*'s generator.



- [Shin et al., 2017] **Deep Generative Replay**
- A cooperative **dual model architecture** consisting of a deep generative model ("generator") and a task solving model ("solver") to retains the knowledge without revisiting actual past data.

  - **Generator** is sequentially trained to generate pseudo-input from current task inputs and generated inputs from *old scholar*'s generator.

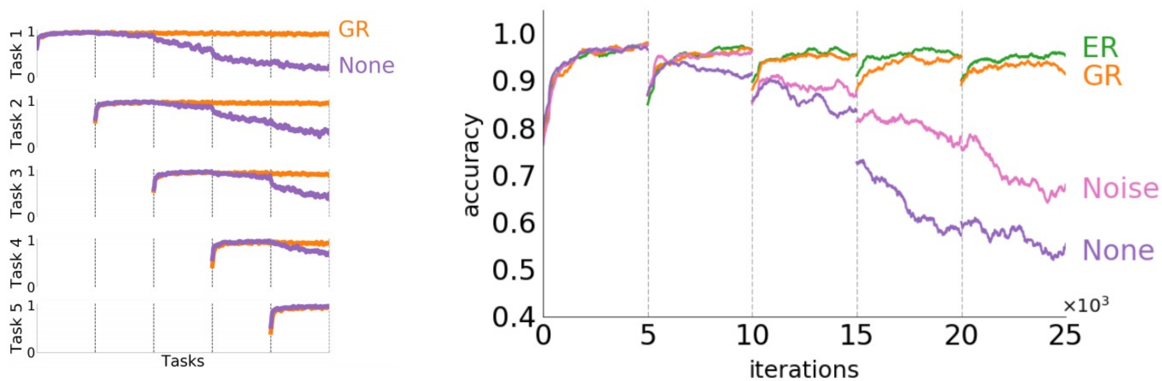
  - **Solver** is sequentially trained on both current input-target pairs and pairs of generated input-target from *old scholar*'s solver.



- [Shin et al., 2017] **Deep Generative Replay**
- Experimental results
  - Test accuracy of sequentially learned solver measured on full test data from MNIST.
  - The first solver learned from real data, and subsequent solvers learned from previous scholar networks.

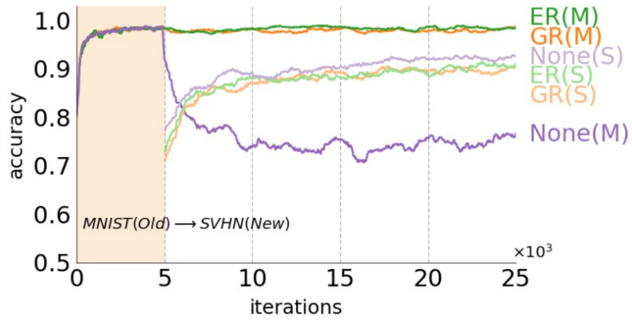
	$Solver_1$	$\rightarrow$	$Solver_2$	$\rightarrow$	$Solver_3$	$\rightarrow$	$Solver_4$	$\rightarrow$	$Solver_5$
Accuracy(%)	98.81%		98.64%		98.58%		98.53%		98.56%

  - Permuted MNIST experiment
    - Generative replay(GR) trains well as much as Exact replay (ER).
    - Sequential training on a solver alone suffer forgetting (None).
    - Replaying gaussian noise does not help tempering performance loss (Noise).

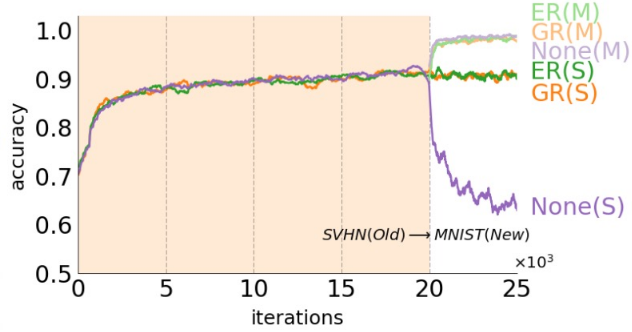


- [Shin et al., 2017] **Deep Generative Replay**
  - Learning new domains

the original task (thick curves)
the new task (dim curves)



(a) MNIST to SVHN

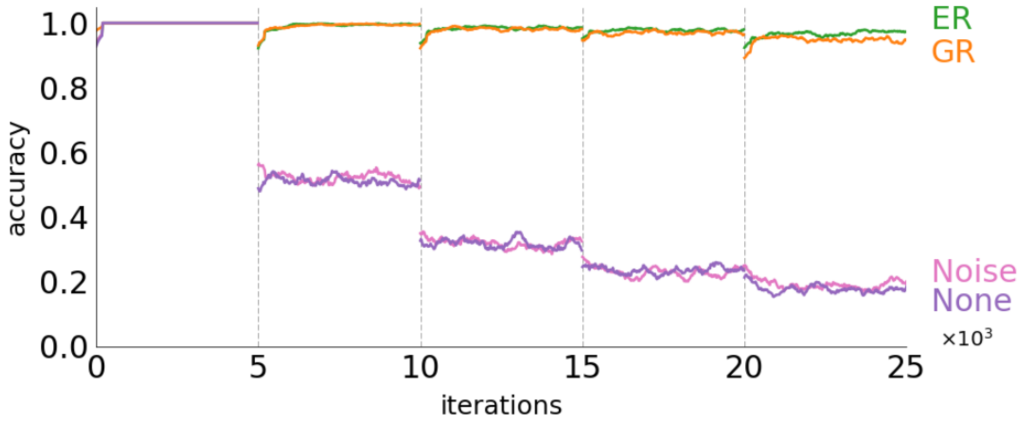


(b) SVHN to MNIST

1000 iterations 2000 iterations 5000 iterations 10000 iterations 20000 iterations

- MNIST→SVHN, SVHN→MNIST
- Generative replay learns new domains sequentially without forgetting.

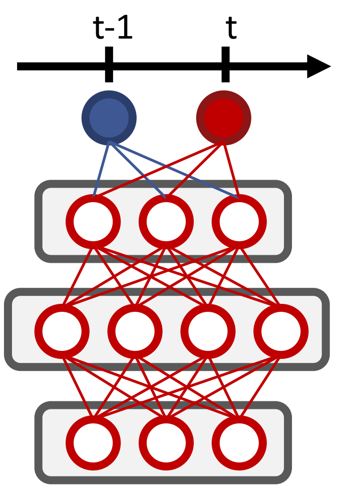
- [Shin et al., 2017] **Deep Generative Replay**
  - Learning new classes



  - MNIST divided into 5 tasks with two labels in each.
  - Generative replay learns new classes sequentially without forgetting.

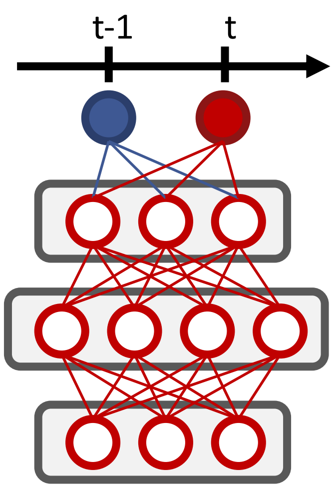
Table of Contents

- Regularization-based approaches prevent forgetting by regularizing the change of a specific set of weights (e.g. EWC).
  - Making the current weights closer to the previous ones may not always ensure that the predictions on the past tasks also remain unchanged.

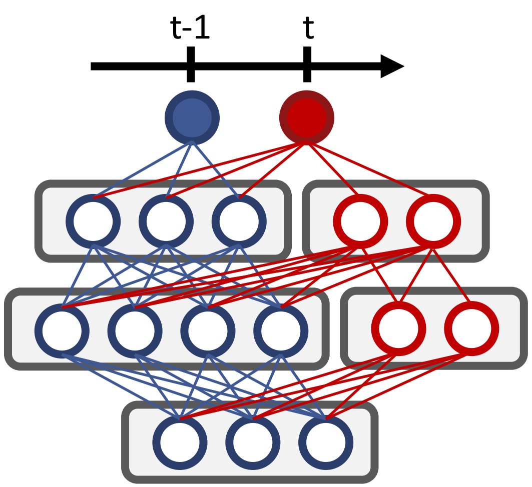


Without fixing model architecture

- Regularization-based approaches prevent forgetting by regularizing the change of a specific set of weights (e.g. EWC).
  - Making the current weights closer to the previous ones may not always ensure that the predictions on the past tasks also remain unchanged.
- **Expansion-based Continual Learning**
  - **Expand** the model architecture to **accommodate new data** instead of fixing it beforehand.
  - Prevent pre-existing components from being overwritten by the new information.

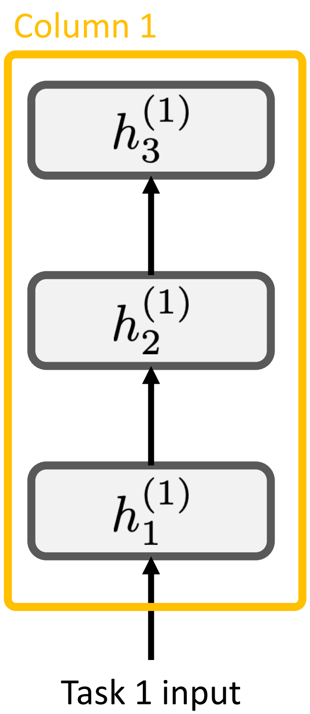


Without fixing model architecture

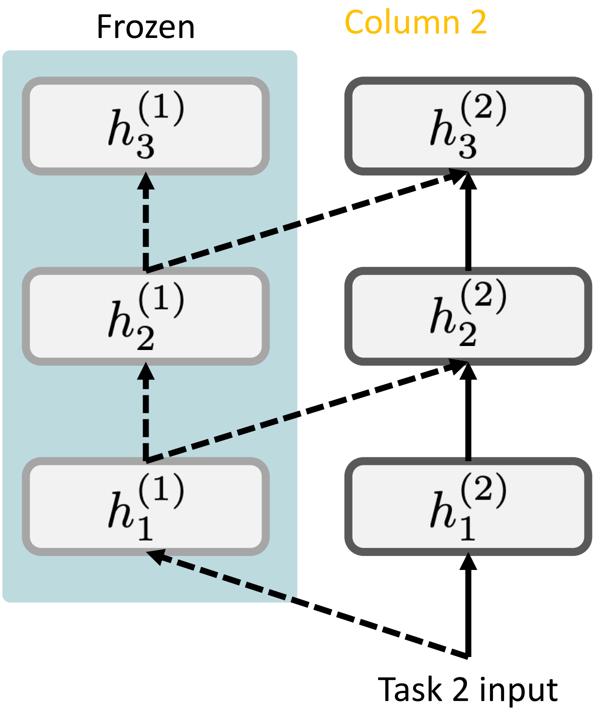


Expanding model architecture

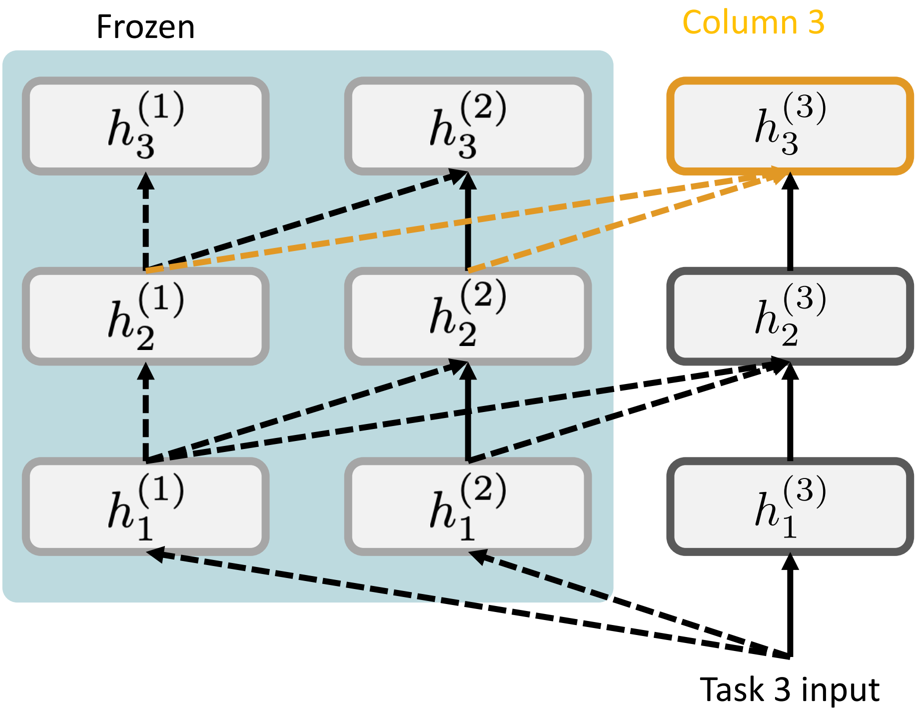
- [Rusu et al., 2016] Progressive Neural Networks
  - Begin with just a single column NN with an initial task.



- [Rusu et al., 2016] Progressive Neural Networks
  - Begin with just a single column NN with an initial task.
  - Allocating a new column for each new task, whose weights are initialized randomly.



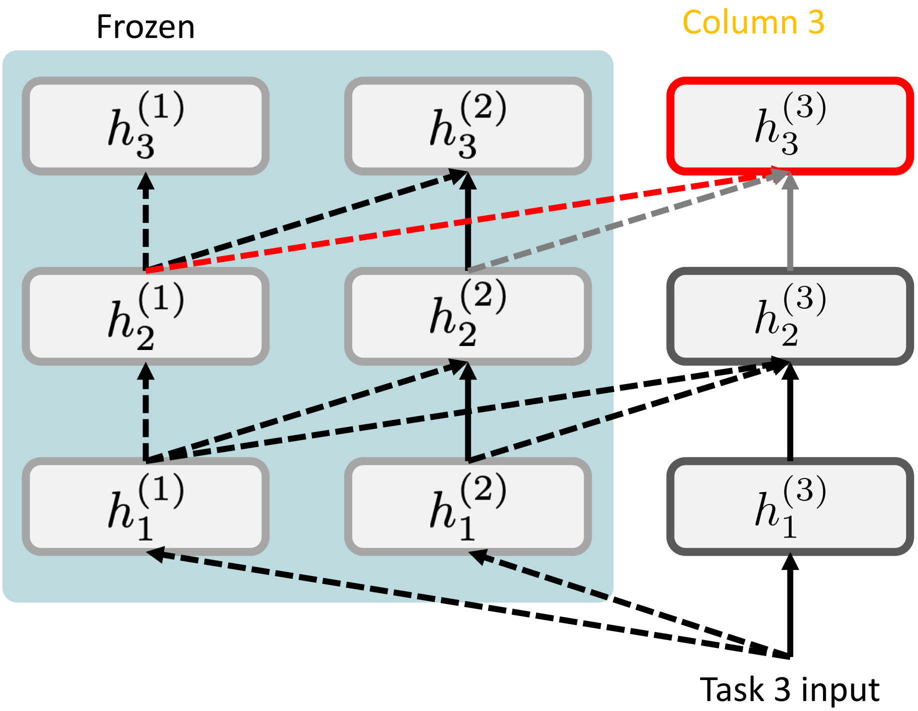
- [Rusu et al., 2016] Progressive Neural Networks
  - Begin with just a single column NN with an initial task.
  - Allocating a new column for each new task, whose weights are initialized randomly.



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

Connections from layer i-1 of column j, to layer i of column k

- [Rusu et al., 2016] Progressive Neural Networks
  - Begin with just a single column NN with an initial task.
  - Allocating a new column for each new task, whose weights are initialized randomly.

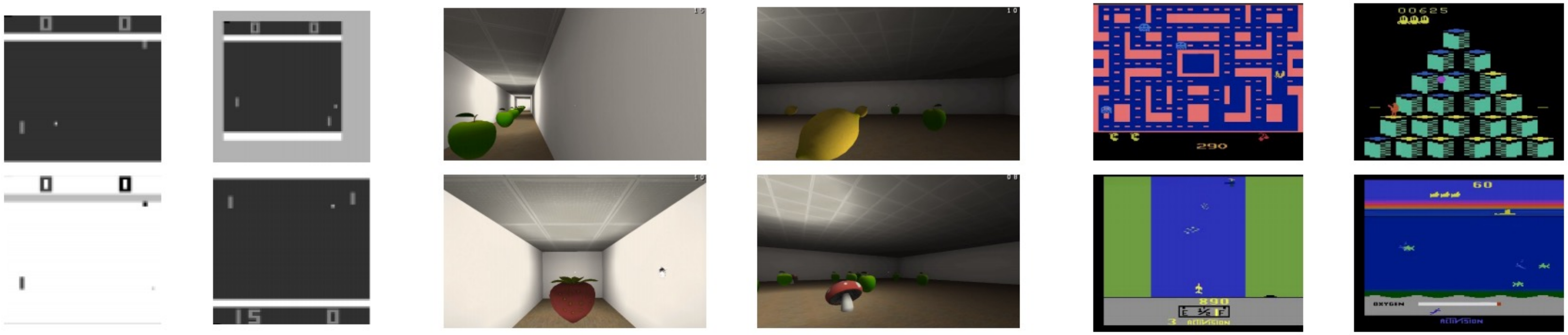


$$h_i^{(k)} = f(W_i^{(k)} h_{i-1}^{(k)} + \sum_{j<k} U_i^{(k:j)} h_{i-1}^{(j)})$$

Connections from layer i-1 of column j, to layer i of column k

- For example, if $h_2^{(1)}$ has enough information(=transferrable) to perform task 3 at layer 3, $h_3^{(3)}$ can ignore inputs other than $h_2^{(1)}$.

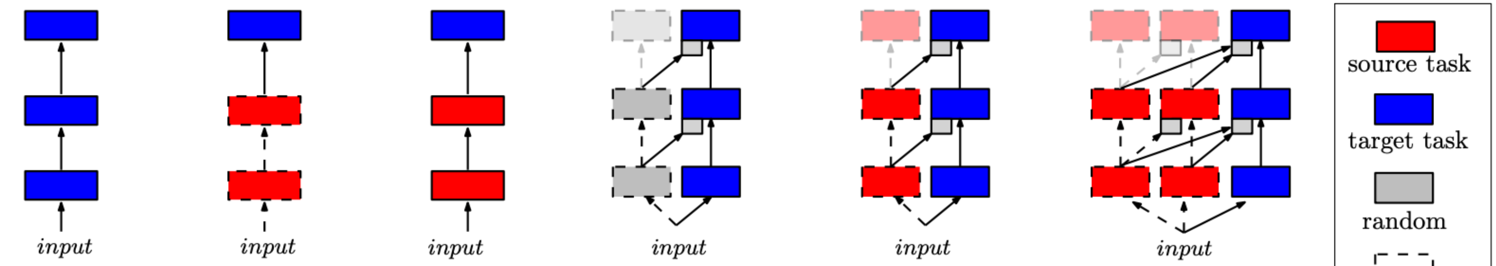
- Experiments
  - Setup



(a) Pong variants (b) Labyrinth games (c) Atari games

- Evaluate across three different RL domains
  - Pong variants: synthetic version of Pong including flipped, noisy, scaled and recolored transforms.
  - Labyrinth games: a set of 3D maze games
  - Atari games: random sequences of Atari games
- New column is linearly added when new task(domain) is given.

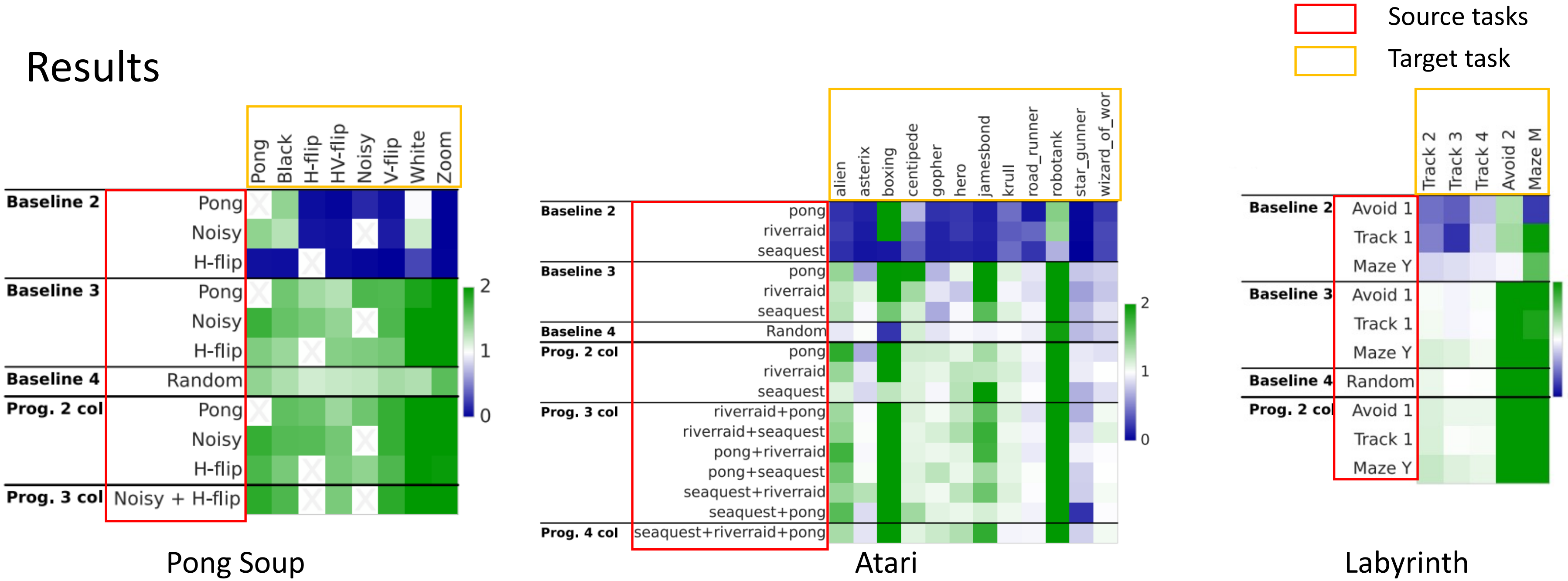
- Experiments
  - Baselines



(1) **Baseline 1** (2) **Baseline 2** (3) **Baseline 3** (4) **Baseline 4** (5) **Progressive Net 2 columns** (6) **Progressive Net 3 columns**

    - Baseline 2: a quite standard in supervised learning with finetuning only output layer of pretrained networks.
    - Baseline 3: support full finetuning of pretrained network
    - Baseline 4: Does progressive NN take advantage of positive transfer from previously learned columns?
  - Metrics
    - Transfer score: the relative performance of an architecture compared with baseline1 (high is better). Clipped in range [0,2].
    - Provide mean and median transfer scores.

- Results



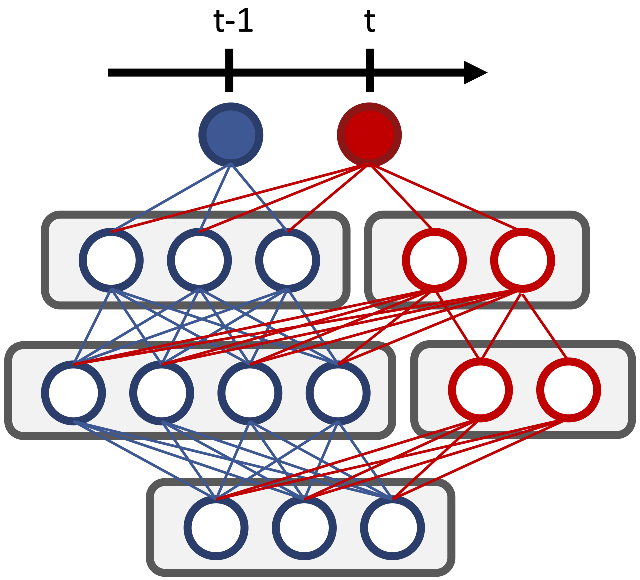
- Baseline2 (Single column, only output layer is finetuned) fails to learn the target task (negative transfer).
- Progressive NNs (with 2 or more columns) show greater **transferability** from source task domains to target domain.

- Results

	Pong Soup		Atari		Labyrinth	
	Mean (%)	Median (%)	Mean (%)	Median (%)	Mean (%)	Median (%)
Baseline 1	100	100	100	100	100	100
Baseline 2	35	7	41	21	88	85
Baseline 3	181	160	133	110	235	112
Baseline 4	134	131	96	95	185	108
Progressive 2 col	209	169	132	112	**491**	**115**
Progressive 3 col	**222**	**183**	140	111	—	—
Progressive 4 col	—	—	**141**	**116**	—	—

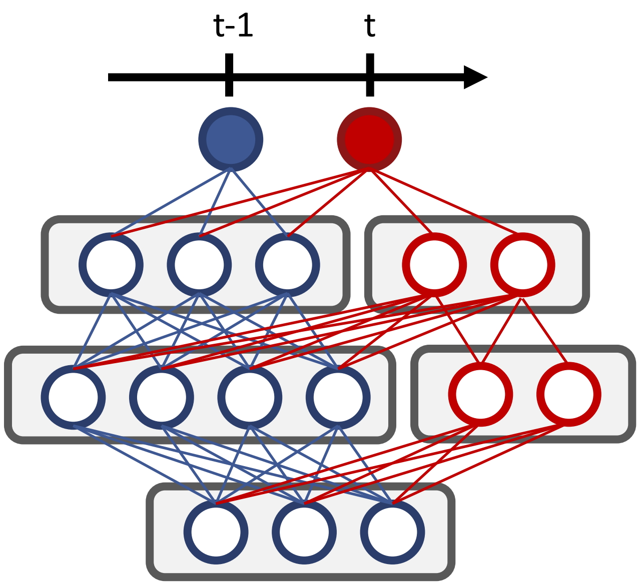
- Baseline 3 shows high positive transfer but progressive NN shows much higher performance in terms of **mean** and **median** score.
  - This suggests progressive NN is better to **exploit transfer** when source and target domains are compatible.
  - Also, since baseline3 learns target domain without preserving features of source task domains, it might suffers **catastrophic forgetting** while progressive NN does not.

- [Yoon et al., 2018] Dynamically Expandable Networks (DEN)
  - Progressive NN-like approaches increase model size linearly on the number of tasks.

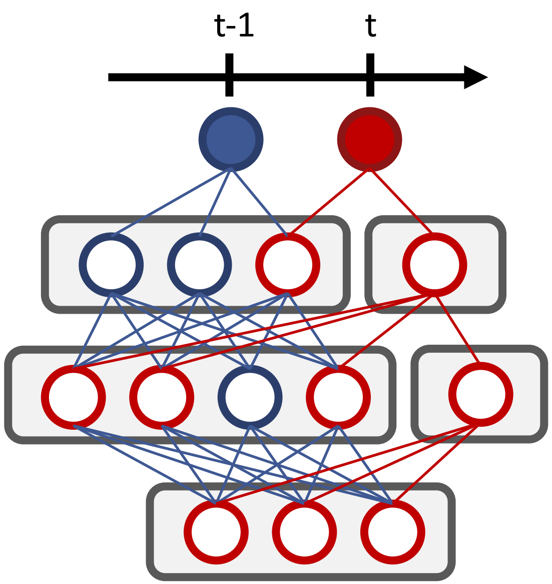


Expanding model architecture

- [Yoon et al., 2018] Dynamically Expandable Networks (DEN)
  - Progressive NN-like approaches increase model size linearly on the number of tasks.
  - **DEN** selectively retrains the old network, expanding its capacity when necessary.

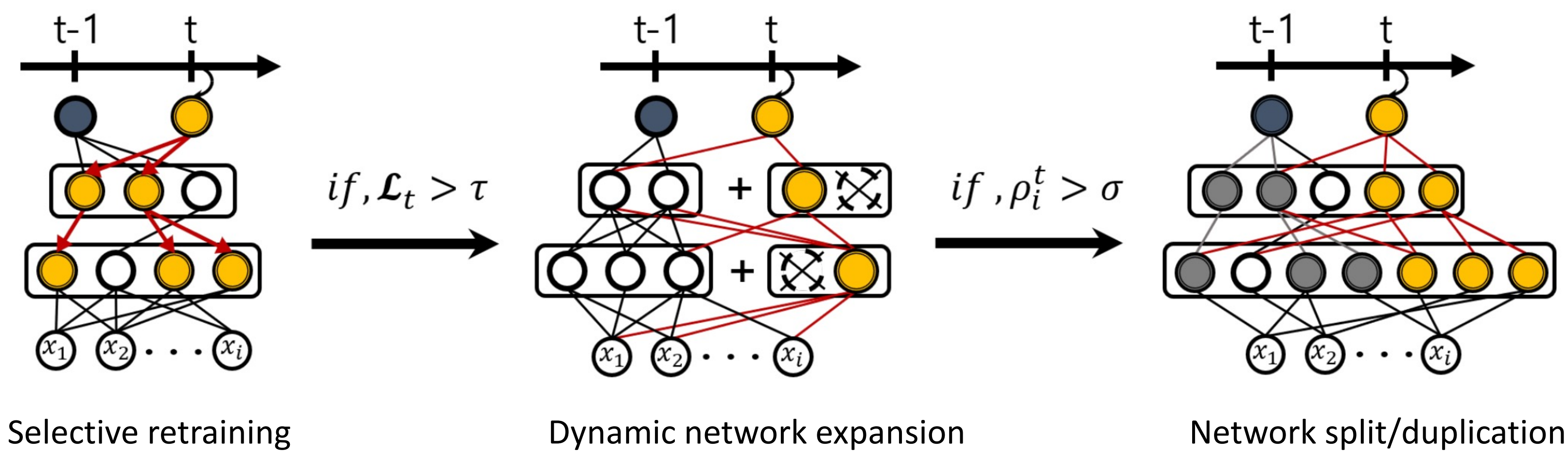


Expanding model architecture

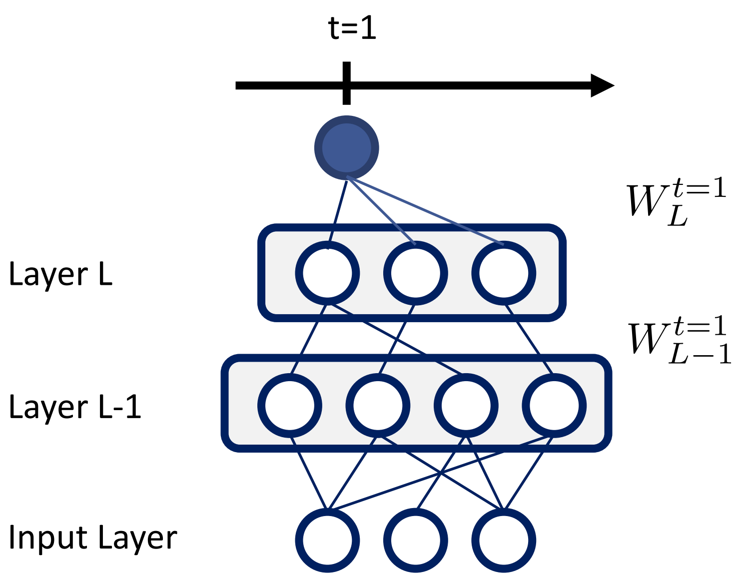


Dynamically Expanding model architecture

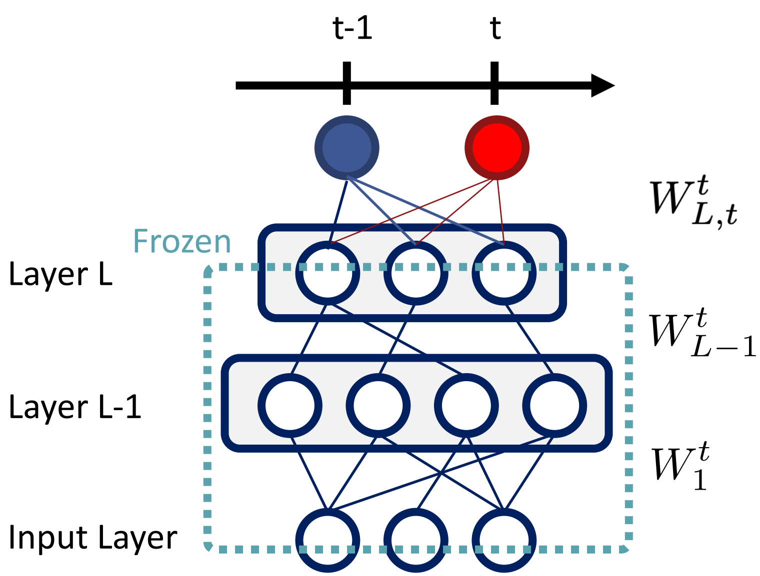
- [Yoon et al., 2018] Dynamically Expandable Networks (DEN)
  - Selective retraining
  - Dynamic network expansion
  - Network split/duplication



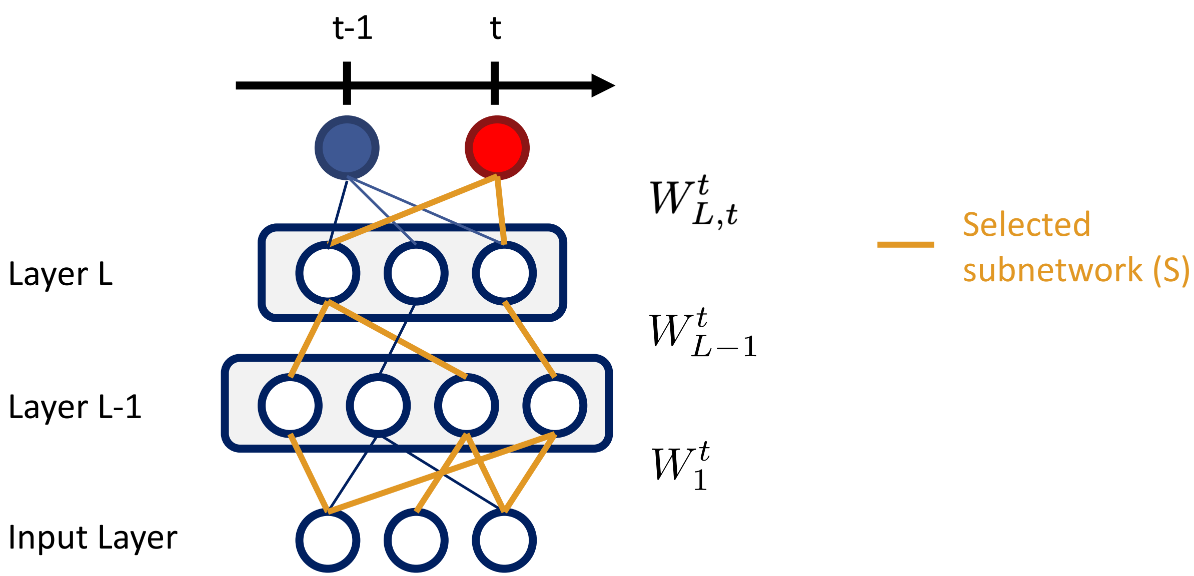
- **Selective Retraining**
  - At the initial task (t=1), train with L1-regularization (sparse network)



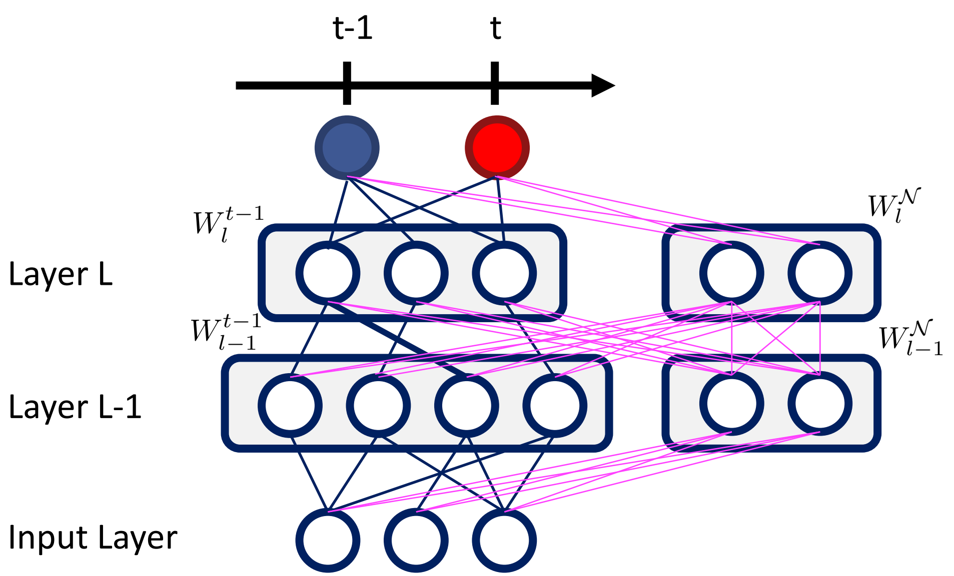
- **Selective Retraining**
  - At every Incoming new task,
    - Train only $W_{L,t}^{t}$ weights (L1-reg).



- **Selective Retraining**
  - At every Incoming new task,
    - Train only $W_{L,t}^{t}$ weights (L1-reg).
    - Then, the non-zero values of $W_{L,t}^{t}$ weights is related to t-task (parameter selection).
    - Selected subnetworks (S): A set of neurons adjacent to selected parameters
    - Train subnetwork S with L2-reg.



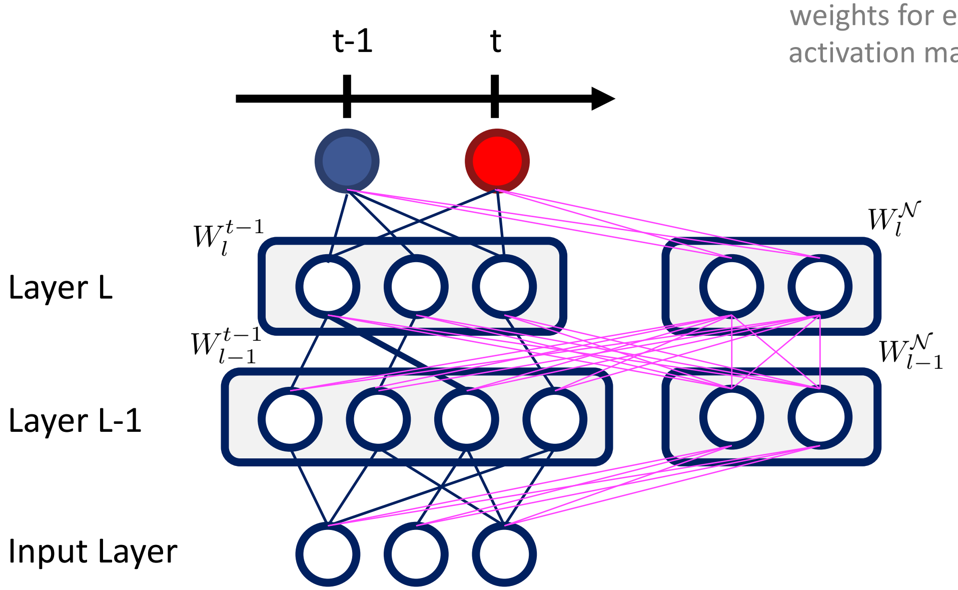
- **Dynamic network expansion**
  - Does selective retrained model perform well on task t?
  - If $(\mathcal{L}_t > \tau)$, expand network.



- **Dynamic network expansion**
  - Does selective retrained model perform well on task t?
  - If ($\mathcal{L}_t > \tau$), expand network.
  - Using group sparse regularization to dynamically decide how many neurons to add at which layer

$$\underset{\boldsymbol{W}_l^{\mathcal{N}}}{\text{minimize}} \mathcal{L}\left(\boldsymbol{W}_l^{\mathcal{N}}; \boldsymbol{W}_l^{t-1}, \mathcal{D}_t\right) + \mu \left\|\boldsymbol{W}_l^{\mathcal{N}}\right\|_1 + \gamma \sum_g \left\|\boldsymbol{W}_{l,g}^{\mathcal{N}}\right\|_2$$

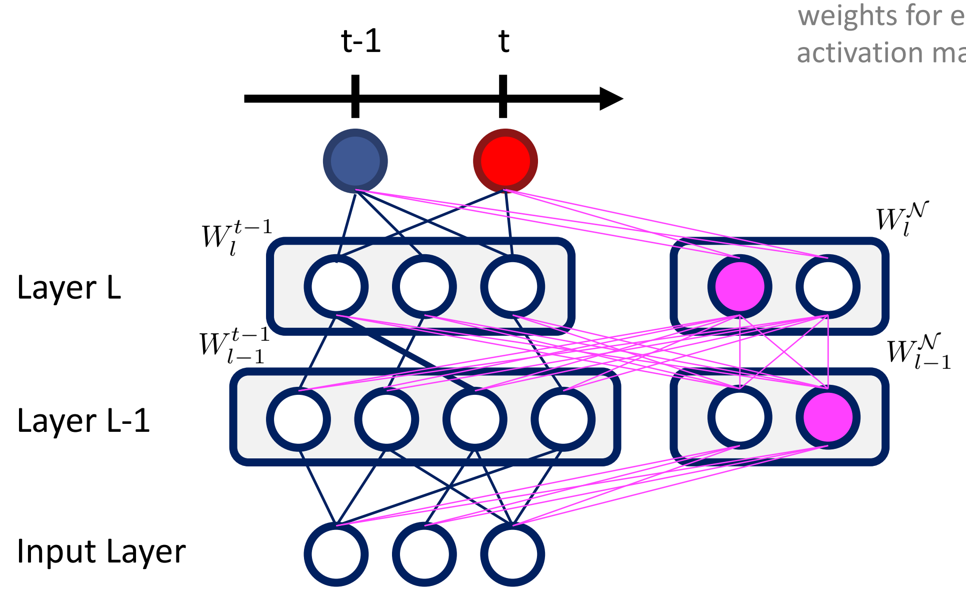
Group defined on the incoming weights for each neuron (e.g., activation map for CNN filter).



- **Dynamic network expansion**
  - Does selective retrained model perform well on task t?
  - If $(\mathcal{L}_t > \tau)$, expand network.
  - Using group sparse regularization to dynamically decide how many neurons to add at which layer

$$\underset{\boldsymbol{W}_l^{\mathcal{N}}}{\text{minimize}} \mathcal{L}\left(\boldsymbol{W}_l^{\mathcal{N}}; \boldsymbol{W}_l^{t-1}, \mathcal{D}_t\right) + \mu \left\|\boldsymbol{W}_l^{\mathcal{N}}\right\|_1 + \gamma \sum_{g} \left\|\boldsymbol{W}_{l,g}^{\mathcal{N}}\right\|_2$$

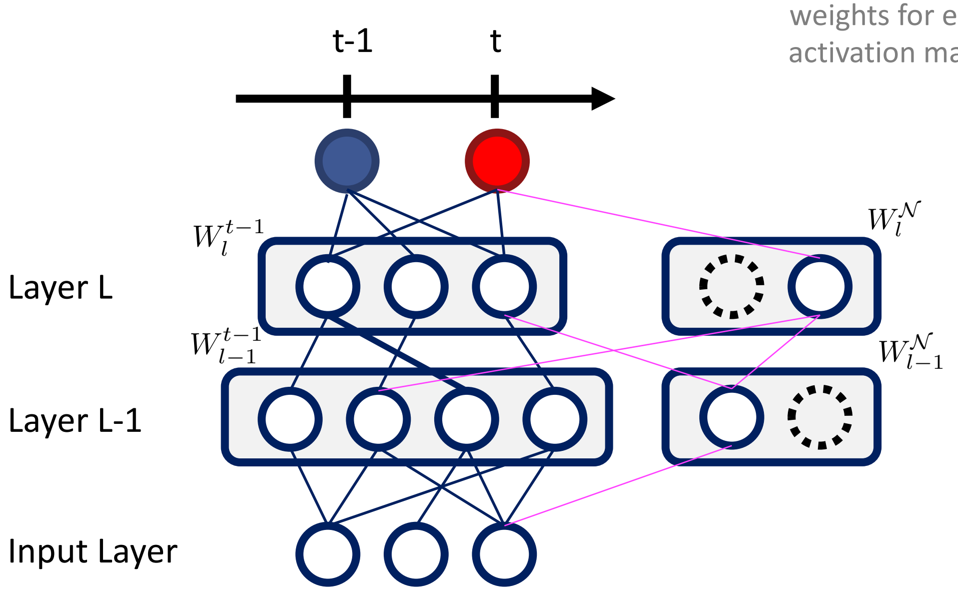
Group defined on the incoming weights for each neuron (e.g., activation map for CNN filter).



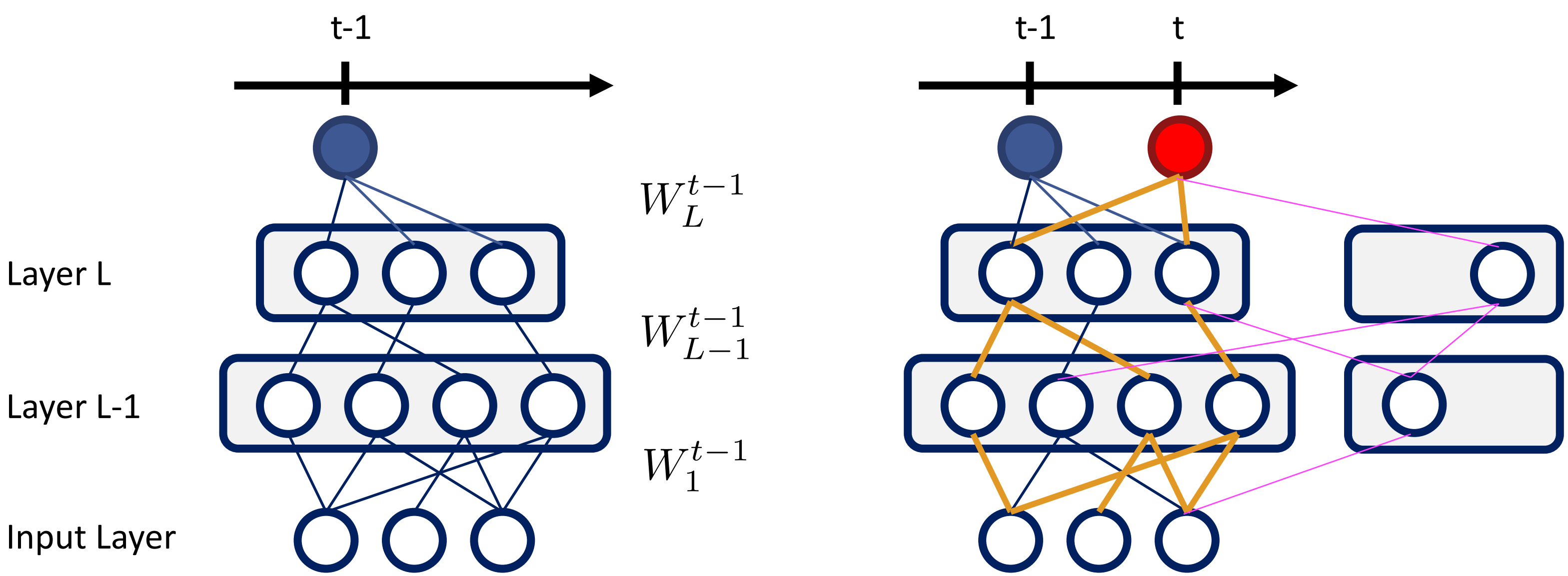
- **Dynamic network expansion**
  - Does selective retrained model perform well on task t?
  - If ($\mathcal{L}_t > \tau$), expand network.
  - Using group sparse regularization to dynamically decide how many neurons to add at which layer

$$\underset{\boldsymbol{W}_l^{\mathcal{N}}}{\text{minimize}} \mathcal{L}\left(\boldsymbol{W}_l^{\mathcal{N}}; \boldsymbol{W}_l^{t-1}, \mathcal{D}_t\right) + \mu \left\|\boldsymbol{W}_l^{\mathcal{N}}\right\|_1 + \gamma \sum_{g} \left\|\boldsymbol{W}_{l,g}^{\mathcal{N}}\right\|_2$$

Group defined on the incoming weights for each neuron (e.g., activation map for CNN filter).

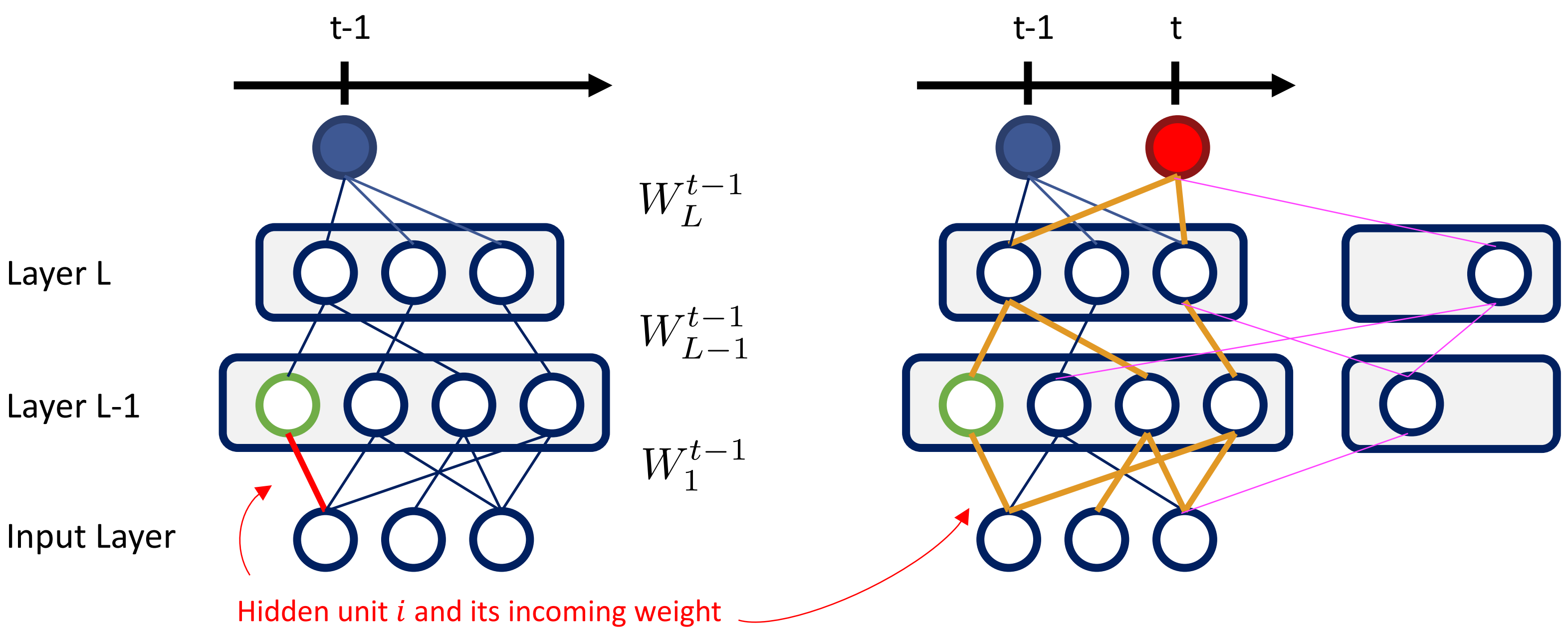


- **Network split/duplication**
  - Prevent forgetting after training with dynamic expanded networks



- **Network split/duplication**
  - Prevent forgetting after training with dynamic expanded networks
  - Measure the amount of semantic drift for each hidden unit i:

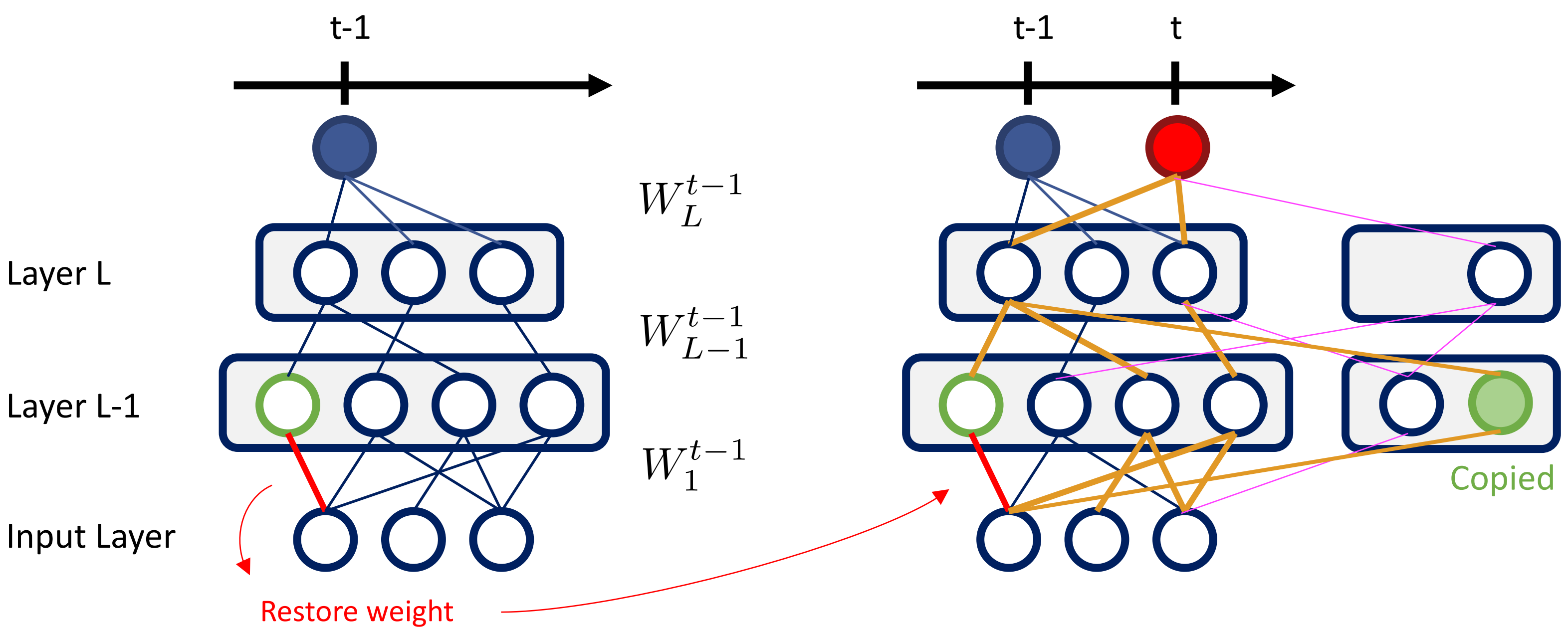
$$\rho_i^t = \|w_i^t - w_i^{t-1}\|_2$$



- **Network split/duplication**
  - Prevent forgetting after training with dynamic expanded networks
  - Measure the amount of semantic drift for each hidden unit i:

$$\rho_i^t = \|w_i^t - w_i^{t-1}\|_2$$

  - If such semantic drift is too large, copy neuron and adjacent weights.
    - The original neuron and weights are restored to the time stamp at t-1th task trained.

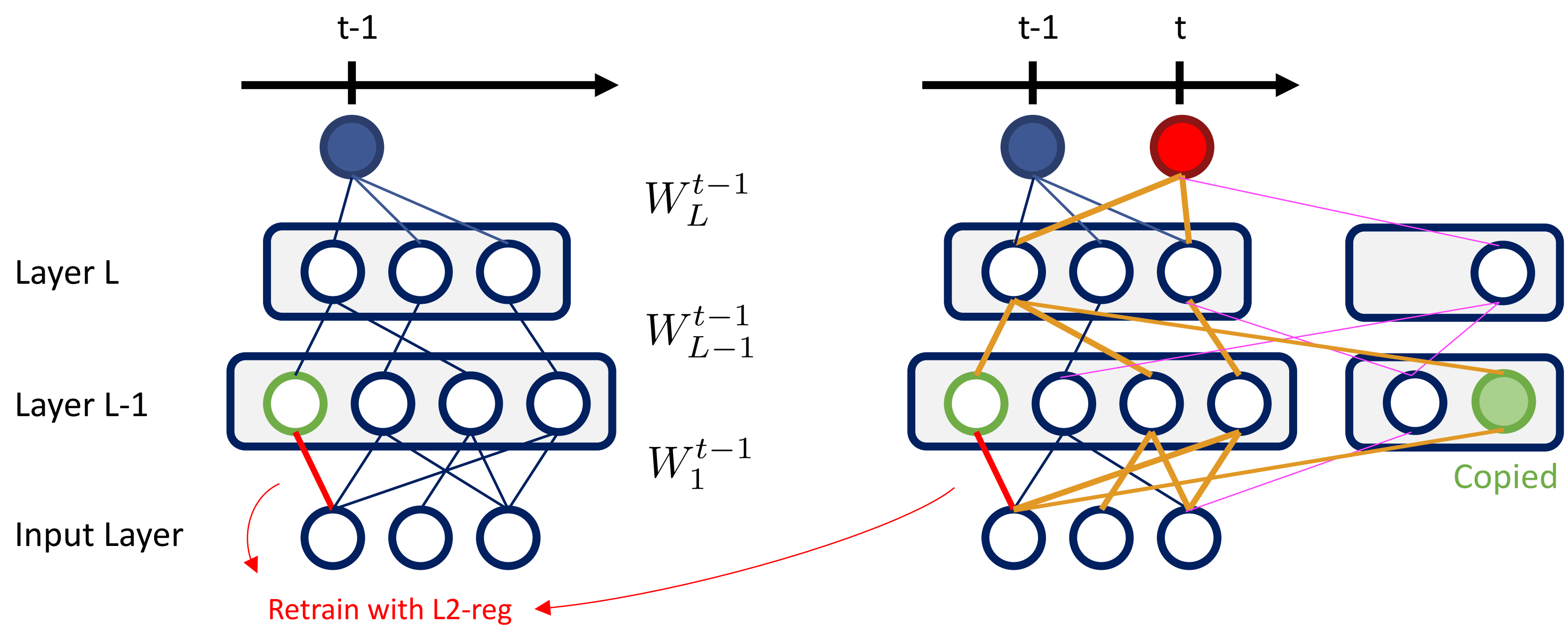


- **Network split/duplication**
  - Prevent forgetting after training with dynamic expanded networks
  - Measure the amount of semantic drift for each hidden unit i:

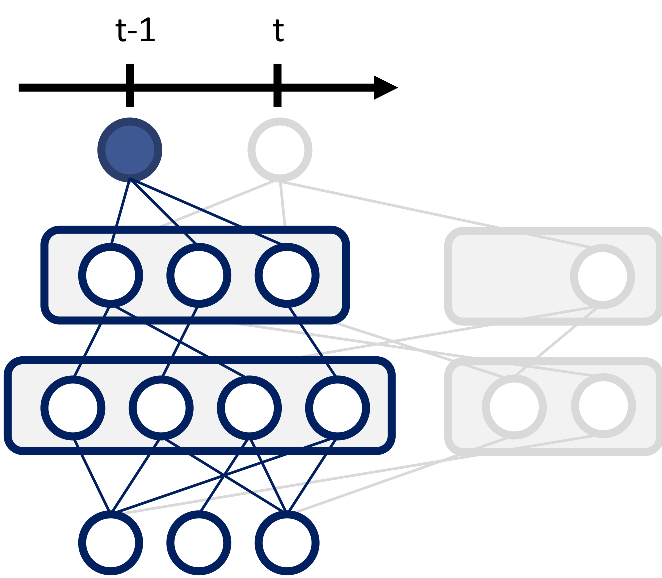
$$\rho_i^t = \|w_i^t - w_i^{t-1}\|_2$$

  - If such semantic drift is too large, copy neuron and adjacent weights.
    - The original neuron and weights are restored to the time stamp at t-1th task trained.
  - Then, retrain network on tth task while retaining t-1th task learned weights.

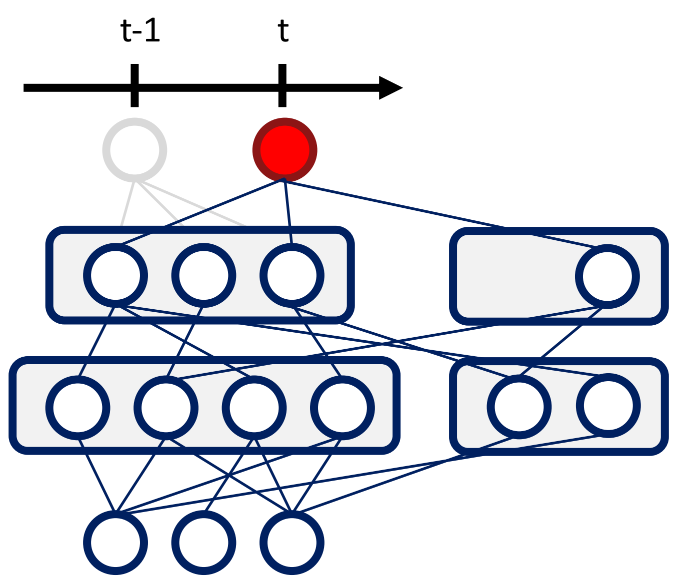
$$\text{minimize}_{\boldsymbol{W}^t} \mathcal{L}\left(\boldsymbol{W}^t; \mathcal{D}_t\right) + \lambda \left\|\boldsymbol{W}^t - \boldsymbol{W}^{t-1}\right\|_2^2$$



- **Timestamped Inference**
  - While network expansion and split procedures, DEN timestamps each newly added unit j by setting $\{z\}_j = t$ to record the training stage t.
  - At inference time, each task will only use the parameters that were introduced up to state t.

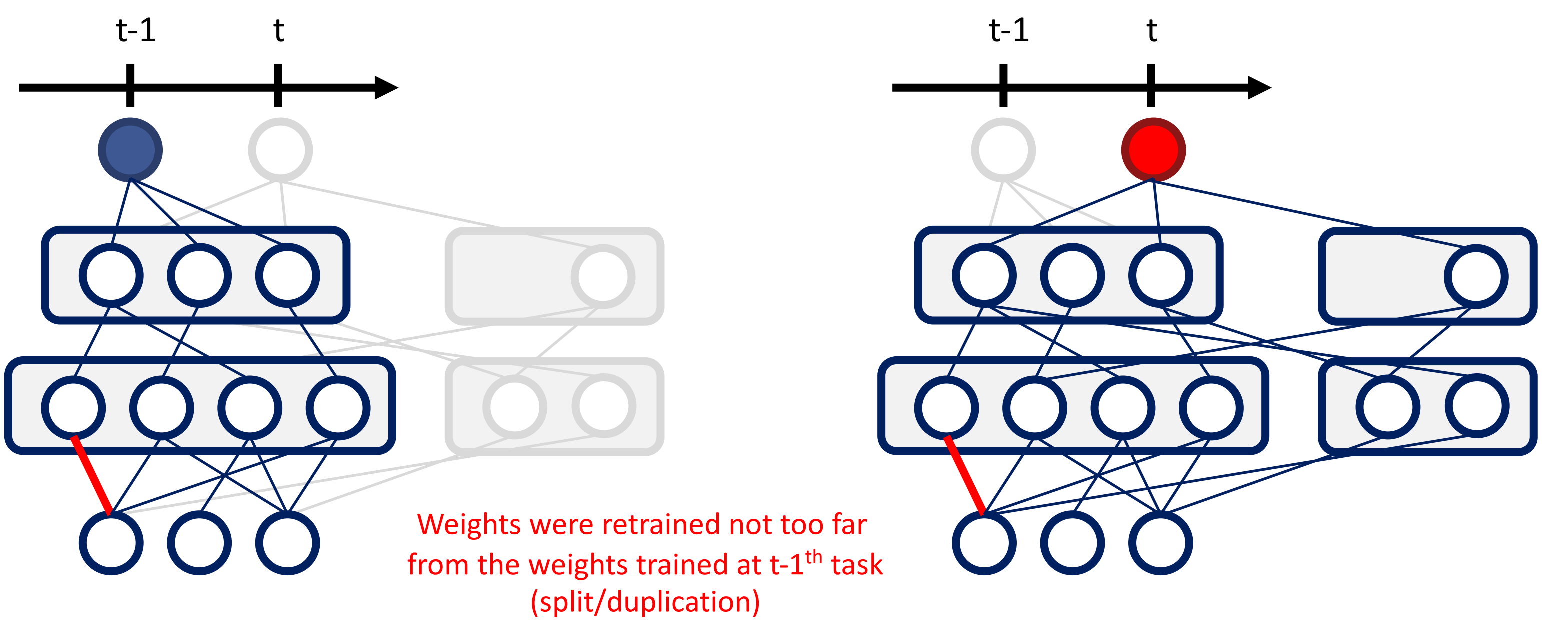


Task identifier is given as t-1



Task identifier is given as t

- **Timestamped Inference**
  - While network expansion and split procedures, DEN timestamps each newly added unit j by setting $\{z\}_j = t$ to record the training stage t.
  - At inference time, each task will only use the parameters that were introduced up to state t.
    - This is why the network split/duplication works for preventing forgetting.

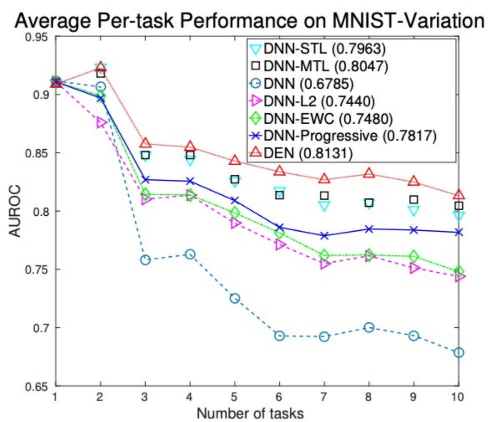


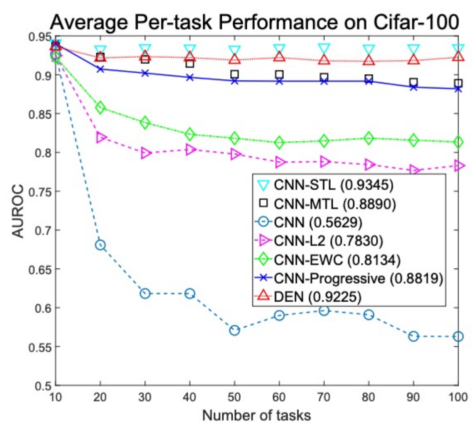
Task identifier is given as t-1 **Task identifier is given as t**

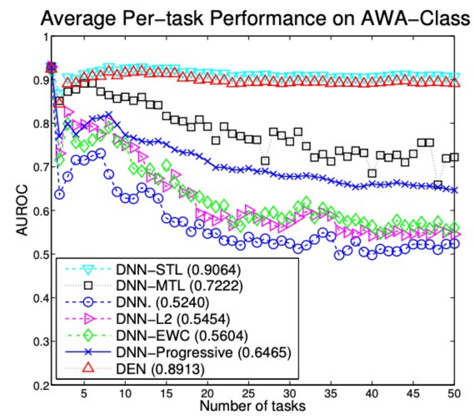
- Baselines
  - DNN-STL: Base DNN, trained for each task separately Offline learning
  - DNN-MTL: Base DNN, trained for all tasks at once
  - DNN: Base DNN, using l2-regularizations Reg-based CL
  - DNN-L2: Base DNN, using l2-regularizations b/w weights of previous/current tasks
  - DNN-EWC: DNN trained with elastic weight consolidation for regularization
  - DNN-Progressive: DNN trained with progressive nets Expansion-based CL
  - DEN

- Datasets
  - MNIST-Variation: rotated / noised MNIST images. One-versus-rest binary task
  - CIFAR-100: 100 classes. Binary task on each class.
  - AwA (Animal with attributes): 50 classes. Binary task on each class.

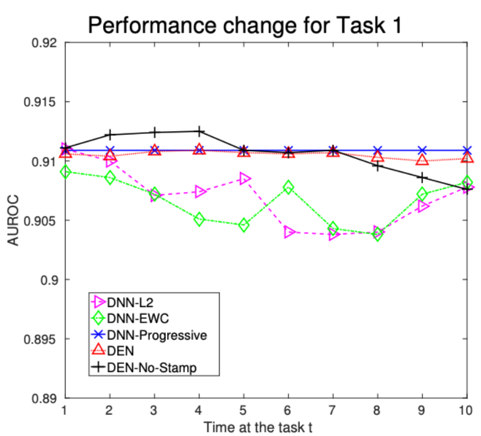
- Average Per-task performance
  - DEN outperforms all online-trained baselines

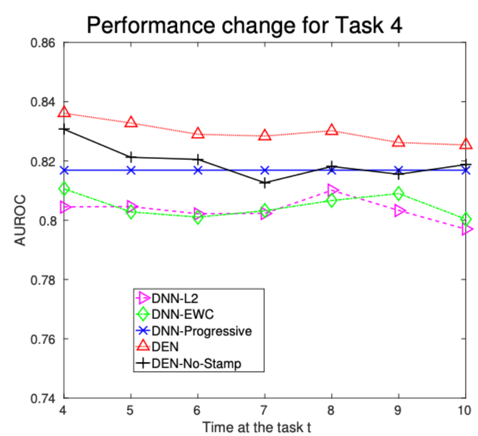


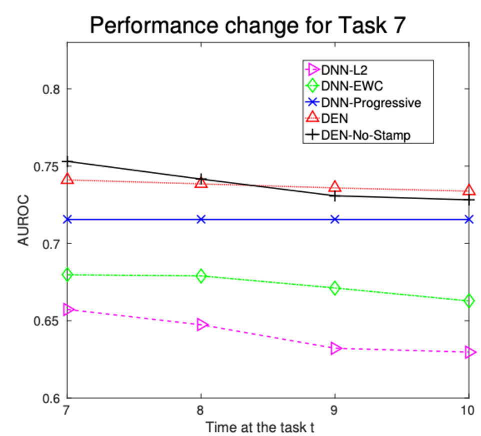




- Performance retention over time (MNIST-Variation)
  - DEN steadily retains learned performance at any time (prevent forgetting)





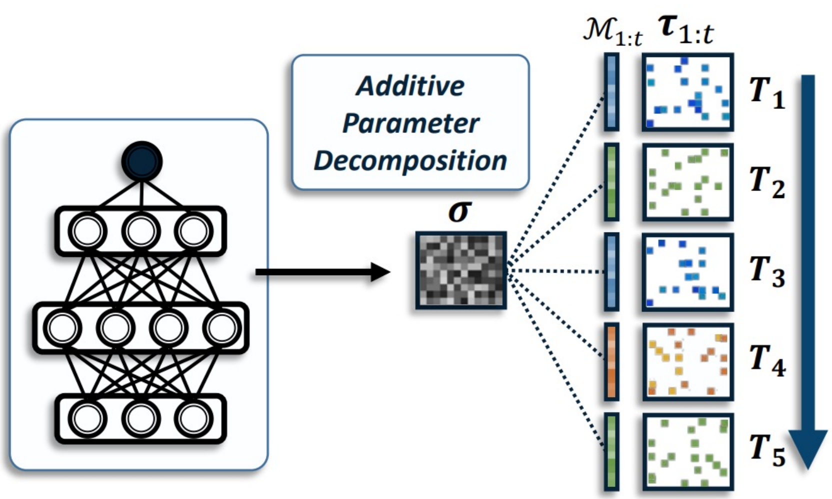


- [Yoon et al., 2020] **Additive Parameter Decomposition (APD)**
  - To effectively mitigate both catastrophic-forgetting and order-sensitivity, APD decompose model parameters into a task-shared parameter σ and a task-adaptive parameter τ_t, M_t:

$$\boldsymbol{\theta}_t = \boldsymbol{\sigma} \otimes \boxed{\mathcal{M}_t} + \boldsymbol{\tau}_t$$

Mask / Attention

  - It can decompose the *generic* / *task-specific* knowledge



- [Yoon et al., 2020] **Additive Parameter Decomposition (APD)**
  - For sparsity, APD decompose model parameters into a task-shared parameter σ and a task-adaptive parameter τ_t, M_t:

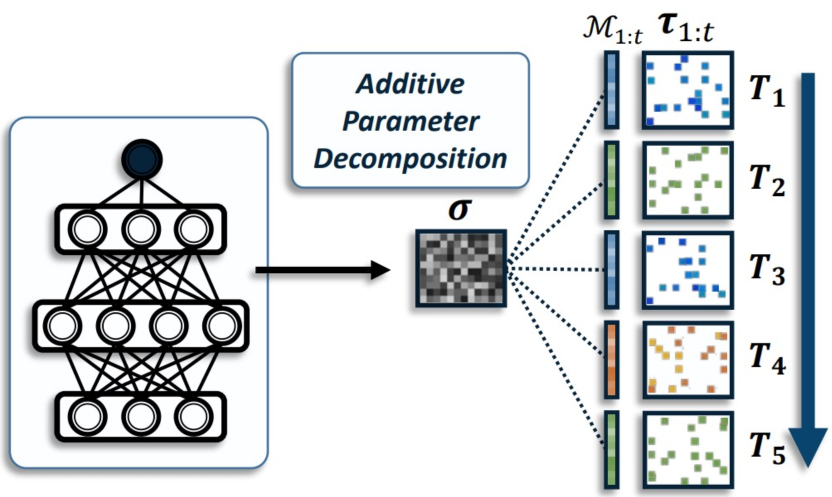
$$\boldsymbol{\theta}_t = \boldsymbol{\sigma} \otimes \mathcal{M}_t + \boldsymbol{\tau}_t$$

  - When a new task t arrives, APD minimizes the task loss with two additional **regularization** terms:

$$\underset{\boldsymbol{\sigma},\boldsymbol{\tau}_t,\boxed{\mathbf{v}_t}}{\text{minimize}} \quad \mathcal{L}\left(\{\boldsymbol{\sigma} \otimes \boxed{\mathcal{M}_t} + \boldsymbol{\tau}_t\}; \mathcal{D}_t\right) + \lambda_1 \|\boldsymbol{\tau}_t\|_1 + \lambda_2 \|\boldsymbol{\sigma} - \boldsymbol{\sigma}^{(t-1)}\|_2^2,$$

sigmoid

For sparsity For transfer



- [Yoon et al., 2020] **Additive Parameter Decomposition (APD)**
  - For order robustness, APD *retroactively* update task adaptive parameters of the past tasks, so that all previous tasks can maintain their original solutions
  - When a new task t arrives,
    - First recover all previous parameters for tasks $i < k$:

$$\boldsymbol{\theta}_i^* = \boldsymbol{\sigma}^{(t-1)} \otimes \mathcal{M}_i^{(t-1)} + \boldsymbol{\tau}_i^{(t-1)}$$

    - Then update previous parameters by constraining them to be close to the current parameters θ_i

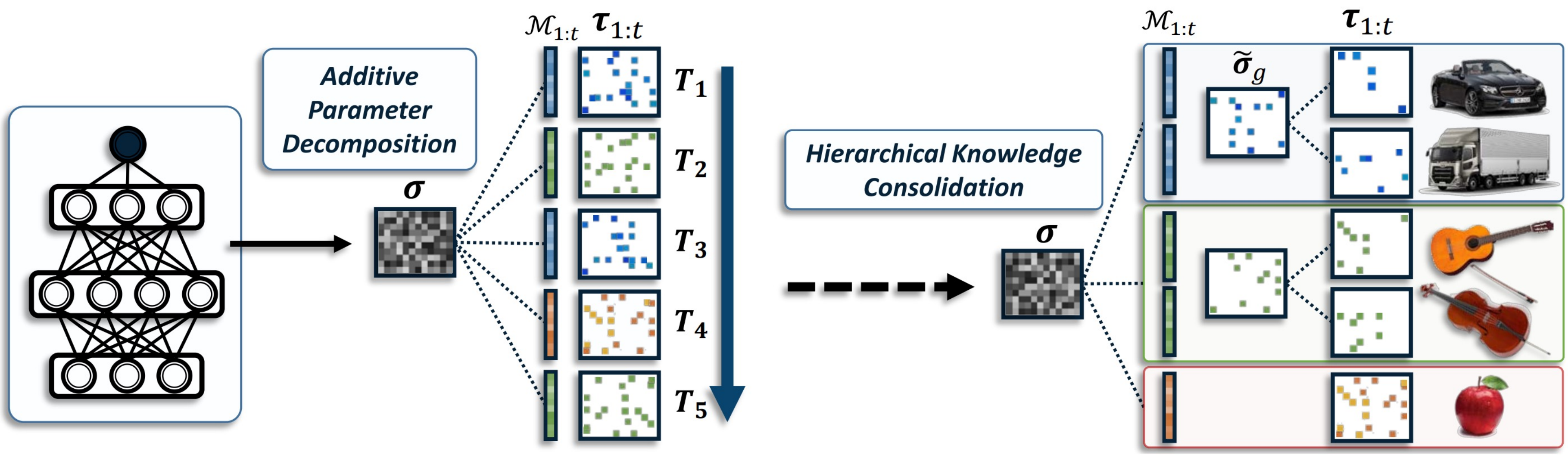
$$\underset{\boldsymbol{\sigma},\boldsymbol{\tau}_{1:t},\mathbf{v}_{1:t}}{\text{minimize}} \; \mathcal{L}\left(\{\boldsymbol{\sigma} \otimes \mathcal{M}_t + \boldsymbol{\tau}_t\}; \mathcal{D}_t\right) + \lambda_1 \sum_{i=1}^{t} \|\boldsymbol{\tau}_i\|_1 + \lambda_2 \sum_{i=1}^{t-1} \|\underbrace{\boldsymbol{\theta}_i^*}_{\text{Past } (t-1) \text{ parameters for } i\text{-th task}} - \underbrace{(\boldsymbol{\sigma} \otimes \mathcal{M}_i + \boldsymbol{\tau}_i)}_{\text{Current } (t) \text{ parameters for } i\text{-th task}}\|_2^2$$

- [Yoon et al., 2020] **Additive Parameter Decomposition (APD)**
  - For further sparsification, APD clusters task adaptive parameters $\{\tau_i\}_{i=1}^{t}$, and decompose them into the locally-shared parameters (center of the cluster) σ_g and sparse parameters $\tilde{\tau}_i$ (i.e., hierarchical knowledge consolidation)

$$\widetilde{\boldsymbol{\tau}}_i = \boldsymbol{\tau}_i + \widetilde{\boldsymbol{\sigma}}_g \text{ for } i \in \mathcal{G}_g$$

  - Now, the final loss is as follows:

$$\underset{\boldsymbol{\sigma},\boldsymbol{\tau}_{1:t},\mathbf{v}_{1:t}}{\text{minimize}} \; \mathcal{L}\left(\{\boldsymbol{\sigma} \otimes \mathcal{M}_t + \boldsymbol{\tau}_t\}; \mathcal{D}_t\right) + \lambda_1 \sum_{i=1}^{t} \|\boldsymbol{\tau}_i\|_1 + \lambda_2 \sum_{i=1}^{t-1} \|\boldsymbol{\theta}_i^* - (\boldsymbol{\sigma} \otimes \mathcal{M}_i + \widetilde{\boldsymbol{\tau}}_i)\|_2^2$$



- APD outperforms baseline continual learning methods in terms of task-average performance (“Accuracy”, ↑) and order robustness (“AOPD”, “MOPD”, ↓)
  - Also, it shows high memory efficiency (“Capacity”, ↓)
- **Each component improves the performance consistently**
  - APD-Fixed + retroactive update = APD(1)
  - APD(1) + hierarchical knowledge consolidation = APD(2)

Single task learning for all tasks (upper bound)

	CIFAR-100 Split				CIFAR-100 Superclass			
Methods	Capacity	Accuracy	AOPD	MOPD	Capacity	Accuracy	AOPD	MOPD
STL	1,000%	63.75%	0.98%	2.23%	2,000%	61.00%	2.31%	3.33%
L2T	100%	48.73%	8.62%	17.77%	100%	41.40%	8.59%	20.08%
EWC	100%	53.72%	7.06%	15.37%	100%	47.78%	9.83%	16.87%
P&C	100%	53.54%	6.59%	11.80%	100%	48.42%	9.05%	20.93%
PGN	171%	54.90%	8.08%	14.63%	271%	50.76%	8.69%	16.80%
DEN	181%	57.38%	8.33%	13.67%	191%	51.10%	5.35%	10.33%
RCL	181%	55.26%	5.90%	11.50%	184%	51.99%	4.98%	14.13%
APD-Fixed	132%	59.32%	2.43%	4.03%	128%	55.75%	3.16%	6.80%
APD(1)	134%	**59.93%**	**2.12%**	**3.43%**	133%	**56.76%**	**3.02%**	**6.20%**
APD(2)	135%	**60.74%**	**1.79%**	**3.43%**	130%	**56.81%**	**2.85%**	**5.73%**

- Many attempts exist to better transfer the knowledge from source to target domains.

- Mainly, two branches of approaches exist.
- [1] Training **universal feature extractor** from data rich source domain
  - e.g.) Big Transfer, self-supervised learning methods
- [2] Consideration of "**what should be transferred**" while doing transfer
  - e.g.) Jacobian matching

- Fine-tuning/transferring knowledge from backbone network is getting very common:
  - e.g.) Most NLP works fine-tune BERT (or GPT), rather than training from scratch

Summary

- Continual learning aims to prevent catastrophic forgetting while learning sequential tasks.

- To prevent forgetting, previous works try to preserve learned knowledge by

😀 😞

  - **[1] Regulating** parameter changes,
    - Elastic Weight Consolidation (EWC)
    - Learning without Forgetting (LwF)

    Resource efficient

    Low performance
    Require task identity

  - **[2] Storing/replaying** past task-specific samples,
    - Experience Replay (ER)
    - Deep Generative Replay

    High performance
    Easy to implement

    Resource expensive
    Privacy

  - **[3] Expanding** model to separate knowledge physically.
    - Progressive Neural Network(PNN)
    - Dynamically Expandable Networks(DEN)

    High performance

    Not practical

- Recent works aim to overcome practical limitations of various types of continual learning settings.
  - Online streamed data (online learning)
  - Task-free (training without task identifier/boundary information)

References

Fine-tuning

- [Pan et al., 2010] A survey on transfer learning, *IEEE Transactions on knowledge and data engineering*, 2010.
  link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5288526
- [Wang et al., 2017] Growing a brain: Fine-tuning by increasing model capacity, *CVPR*, 2017.
  link: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8099806
- [Hendrycks et al., 2019] Using Pre-training Can Improve Model Robustness and Uncertainty , *ICML*, 2019.
  link: https://arxiv.org/pdf/1901.09960.pdf
- [Kolesnikov et al.,2020] Big Transfer (BiT): General Visual Representation Learning, 2020.
  link: https://arxiv.org/pdf/1912.11370.pdf
- [Kumar et al., 2022] Fine-Tuning can Distort Pretrained Features and Underperform Out-of-Distribution, 2022.
  link: https://arxiv.org/pdf/2202.10054.pdf

References

Knowledge distillation

- [Hinton et al., 2015] Distilling the knowledge in a neural network, *NIPS workshops*, 2015.
  link: https://arxiv.org/pdf/1503.02531.pdf
- [Zagoruyko et al., 2017] Paying more attention to attention: Improving the performance of convolutional neural networks via attention transfer, *ICLR*, 2017.
  link: https://arxiv.org/pdf/1612.03928.pdf
- [Srinivas et al., 2018] Knowledge Transfer with Jacobian Matching, *ICML*, 2018.
  link: https://arxiv.org/pdf/1803.00443.pdf
- [Ahn et al., 2019] Variational Information Distillation for Knowledge Transfer, *CVPR*, 2019.
  link: https://arxiv.org/abs/1904.05835
- [Park et al., 2019] Relational Knowledge Distillation, *CVPR*, 2019.
  link: https://arxiv.org/abs/1904.05068
- [Jang et al., 2019] Learning What and Where to Transfer, ICML, 2019.
  link: https://arxiv.org/pdf/1905.05901
- [Tian et al., 2020] Contrastive Representation Distillation, *ICLR*, 2020.
  link: https://arxiv.org/abs/1910.10699

References

Domain Adaptation

- [Grandvalet & Bengio, 2004] Semi-supervised Learning by Entropy Minimization. NIPS 2004.
  link : https://papers.nips.cc/paper/2740-semi-supervised-learning-by-entropy-minimization
- [Ganin et al., 2015] Unsupervised Domain Adaptation by Backpropagation. ICML 2015.
  link : http://proceedings.mlr.press/v37/ganin15.html
- [Tzeng et al., 2017] Adversarial Discriminative Domain Adaptation. CVPR 2017.
  link : https://arxiv.org/abs/1702.05464
- [Bousmalis et al., 2016] Domain Separation Networks. NIPS 2016.
  link : https://arxiv.org/abs/1608.06019
- [Long et al., 2016] Unsupervised Domain Adaptation with Residual Transfer Networks. NIPS 2016.
  link : https://arxiv.org/abs/1602.04433
- [Tobin et al., 2017] Domain Randomization for Transferring Deep Neural Networks from Simulation... IROS 2017.
  link : https://arxiv.org/abs/1703.06907
- [Shrivastava et al., 2017] Learning from Simulated and Unsupervised Images through Adversarial... CVPR 2017.
  link : https://arxiv.org/abs/1612.07828
- [Bousmalis et al., 2017] Unsupervised Pixel-Level Domain Adaptation with Generative Adversarial... CVPR 2017.
  link : https://arxiv.org/abs/1612.05424
- [Hoffman et al., 2018] CyCADA: Cycle-Consistent Adversarial Domain Adaptation. ICML 2018.
  link : https://arxiv.org/abs/1711.03213

References

Multi-task learning

- [Zhang et al., 2017] A Survey on Multi-Task Learning, arxiv, 2017.
  link: https://arxiv.org/abs/1707.08114
- [Misra et al., 2016] Cross-stitch Networks for Multi-task Learning, CVPR, 2016.
  link: https://arxiv.org/abs/1604.03539
- [Mudrakarta et al., 2019] K for the Price of 1: Parameter-efficient Multi-task and Transfer Learning, ICLR, 2019.
  link: https://arxiv.org/abs/1810.10703
- [Kendall et al., 2018] Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics, *CVPR*, 2018.
  link: https://arxiv.org/pdf/1705.07115.pdf
- [Chen et al., 2018] GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks, ICML, 2018.
  link: https://arxiv.org/abs/1711.02257
- [Sener et al., 2018] Multi-task Learning as Multi-objective Optimization, NIPS, 2018.
  link: https://arxiv.org/abs/1810.04650
- [Barber et al., 2003] The IM Algorithm: A Variational Approach to Information Maximization, NIPS, 2003.
  link: http://papers.nips.cc/paper/2410-information-maximization-in-noisy-channels-a-variational-approach.pdf
- [Andrychowicz et al., 2016] Learning to learn by gradient descent by gradient descent, NIPS, 2016.
  link: https://arxiv.org/abs/1606.04474
- [Finn et al., 2017] Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks, ICML, 2017.
  link: https://arxiv.org/abs/1703.03400

References

Continual Learning

[Kirkpatrick, J., Pascanu, R., et al., 2017] Overcoming catastrophic forgetting in neural networks, In PNAS, 2017.
https://arxiv.org/pdf/1612.00796.pdf

[Li et al., 2016] Learning without forgetting, In ECCV, 2016.
https://arxiv.org/abs/1606.09282

[Chaudhry et al., 2019] On Tiny Episodic Memories in Continual Learning. *arXiv preprint*, 2019.
https://arxiv.org/pdf/1902.10486.pdf

[Lopez-Paz and Ranzato, 2017] Gradient Episodic Memory for Continual Learning. In NeurIPS, 2017.
https://arxiv.org/pdf/1706.08840.pdf

[Aljundi et al., 2019] Gradient based sample selection for online continual learning. In NeurIPS 2019.
https://arxiv.org/pdf/1903.08671.pdf

[Buzzega et al., 2020] Dark experience for general continual learning: a strong, simple baseline, In NeurIPS, 2020.
https://papers.nips.cc/paper/2020/hash/b704ea2c39778f07c617f6b7ce480e9e-Abstract.html

[Cha et al., 2021] Co2l: Contrastive continual learning, In ICCV 2021.
https://openaccess.thecvf.com/content/ICCV2021/papers/Cha_Co2L_Contrastive_Continual_Learning_ICCV_2021_paper.pdf

[Shin et al., 2017] Continual Learning with Deep Generative Replay, In NIPS 2017.
https://arxiv.org/pdf/1705.08690.pdf

[Rusu et al., 2016] Progressive Neural Networks, *arXiv preprint, 2016.*
https://arxiv.org/pdf/1606.04671.pdf

[Yoon, et al., 2018] Lifelong Learning with Dynamically Expandable Networks, In ICLR, 2018.
https://arxiv.org/pdf/1708.01547.pdf

[Yoon et al., 2020] Scalable and Order-robust Continual Learning with Additive Parameter Decomposition, In ICLR 2020.
https://arxiv.org/pdf/1902.09432.pdf

References

Other useful references

- [Weiss et al., 2016] A survey of transfer learning, *Journal of Big Data*, 2016.
  link: https://journalofbigdata.springeropen.com/track/pdf/10.1186/s40537-016-0043-6
- [Tan et al., 2018] A Survey on Deep Transfer Learning, *arXiv preprint arXiv:1808.01974*, 2018.
  link: https://arxiv.org/pdf/1808.01974.pdf
- [Yaniv et al., 2014] DeepFace: Closing the Gap to Human-Level Performance in Face Verification, *CVPR*, 2014.
  link: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6909616
- [Razavian et al., 2014] CNN features off-the-shelf: an astounding baseline for recognition, *CVPR Workshops*, 2014.
  link: https://arxiv.org/pdf/1403.6382.pdf
- [Ge et al., 2017] Borrowing treasures from the wealthy: Deep transfer learning through selective joint fine-tuning, *CVPR*, 2017.
  link: https://arxiv.org/pdf/1702.08690.pdf
- [Springenberg et al., 2015] Striving for Simplicity: The All Convolutional Net, *ICLR*, 2015.
  link: https://arxiv.org/pdf/1412.6806.pdf
- [Selvaraju et al., 2016] Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization, *ICCV*, 2017.
  link: https://arxiv.org/pdf/1610.02391.pdf
- [Czarnecki et al., 2017] Sobolev raining for neural networks, *NIPS*, 2017.
  link: https://arxiv.org/pdf/1706.04859.pdf
- [Cui et al., 2018] Large scale fine-grained categorization and domain specific transfer learning, *CVPR*, 2018.
  link: https://arxiv.org/abs/1806.06193