Supplementary material for
"AdvisIL - A Class-Incremental Learning Advisor"

Eva Feillet[1], Grégoire Petit[1,2] Adrian Popescu[1], Marina Reyboz[3], Céline Hudelot[4]
[1]Université Paris-Saclay, CEA, LIST, F-91120, Palaiseau, France
[2]LIGM, Ecole des Ponts, Univ Gustave Eiffel, CNRS, Marne-la-Vallée, France
[3]Université Grenoble Alpes, CEA, LIST, F-38000 Grenoble, France
[4]Université Paris-Saclay, CentraleSupélec, MICS, France
{eva.feillet, gregoire.petit, adrian.popescu, marina.reyboz}@cea.fr,
celine.hudelot@centralesupelec.fr

1. Implementation details

All our experiments are run with PyTorch. For each algorithm, we point to the original repository on which our implementation is based. Except otherwise stated, we used the method-specific hyperparameters presented in the original articles (e.g. the temperature scalar in LUCIR). Our complete code containing image lists, algorithms, hyperparameters and network architectures will be published in a dedicated repository.

1.1. Class-incremental learning algorithms

• **SIW.** Our implementation is based on the original repository of [2][1].

• **DeeSIL.** Our implementation is based on the original repository of [1][2].

• **LUCIR.** Our implementation is based on the original repository of [4][3]. LUCIR has initially been proposed as a learning algorithm with memory of past examples. In practice, as we focus on examplar-free class-incremental learning, we set the size of LUCIR's memory buffer to zero.

• **SPB.** We use the SPB-M version [7], which uses a data augmentation procedure based on image rotations. Our implementation is based on LUCIR's implementation, with a modified loss function and SPB-M's data augmentation procedure.

• **DSLDA.** Our implementation is based on the original repository of [3][4].

• **FeTrIL.** Our implementation of FetrIL [5] is available on a public repository [5].

To choose the learning hyperparameters of each neural network, we used the values listed in Table 1 for a hyperparameter search.

Hyperparameter	Range of values
nbr. epochs	$\{70, 100\}$
learning rate (lr)	$\{0.005, 0.01, 0.05, 0.1\}$
momentum	$\{0.75, 0.80, 0.85, 0.90, 0.95\}$
weight decay	$\{0.0001, 0.0005\}$
lr stratifier factor	0.1
lr stratifier steps	$\{30, (30, 50), 40, (40, 60), 50\}$
batch size	$\{32, 64\}$

Table 1: Range of values for hyperparameter tuning

2. Datasets

We summarize in Table 2 key information about each of the datasets used in our experiments. Note that all our datasets contain 100 classes. Regardless of their initial resolution, all images are resized to $224x224$ RGB pixels when processed by a neural network.

2.1. Backbone networks

Width and depth multipliers. Neural networks are commonly represented as a sequence of layers, with repetitions of specific layer subsequences, which are called *blocks*. We use convolutional neural networks of various sizes and control their size using two hyperparameters.

- A depth multiplier d determines the number of blocks and therefore the depth of the network.
- A width multiplier w determines the number of convolutional filters of each block and therefore the width of the

[1]https://github.com/EdenBelouadah/class-incremental-learning/tree/master/siw
[2]https://github.com/EdenBelouadah/class-incremental-learning/tree/master/deesil
[3]https://github.com/hshustc/CVPR19_Incremental_Learning
[4]https://github.com/tyler-hayes/Deep_SLDA
[5]https://github.com/GregoirePetit/FeTrIL

	Name	Source	Nbr train images	Nbr test images	Example classes
Reference datasets	INFauna	ImageNet	340	60	creepy-crawly, lemon shark, sparrow
	INFlora	ImageNet	340	60	wood anemone, clove-pink, fescue grass
	INFood	ImageNet	340	60	yolk, lentil soup, aqua vitae
	INRand_0	ImageNet	340	60	hand-held computer, hush puppy, satchel
	INRand_1	ImageNet	340	60	poteen (strong Irish alcohol), carriageway
Test datasets	INRand_2	ImageNet	340	60	Exmoor (English moorland region), mango
	LAND100	Google Landmarks v1	375 in average	20	New York stack exchange (USA), Colosseum (Italy)
	INAT100	iNaturalist	300	10	mushroom species, insect species, orcal
	FOOD100	FOOD101	750	250	beef tartare, waffles, sashimi

Table 2: Key information about the datasets used in our experiments.

network.

We implemented neural networks based on the PyTorch implementation of ResNet18 [6], that of Mobilenetv2 (x1.0) [7] and that of Shufflenetv2 (x1.0) [8]. In the code, each of these neural networks is an instance of a dedicated class, to which we added a width multiplier argument and a depth multiplier argument to control its size. We round the result of multiplications by d to obtain an integer greater or equal to 1 (no deletion of a type of block). We round the result of multiplications by w to obtain a number of convolutional filters divisible by 8.

• **ResNet18.** The architecture of ResNet18 is illustrated in Table 3. It is based on the repetition of blocks containing convolutional layers, called *BasicBlocks* in its PyTorch implementation. We introduce a depth multiplier which controls the number of times this building unit is repeated, and a width multiplier w which controls the number of convolutional filters contained in each BasicBlock. By applying the multipliers $w = 0.5$ and $d = 0.5$ to the initial architecture of ResNet18, we obtain a smaller network composed of one BasicBlock with 32 convolutional filters instead of two BasicBlocks with 64 convolutional filters in conv_2, one BasicBlock with 64 convolutional filters instead of two BasicBlocks with 128 convolutional filters in conv_3, etc.

• **MobileNetv2.** This architecture has been proposed with a parameter to control its width [6]. It uses inverted residual blocks and bottleneck layers. A network is defined by a quadruplet (t, c, n, s) where t is called the *expansion factor* and controls the size of a bottleneck layer, c is the number of output channels for convolutional layers, n is the number of repetitions of building blocks and s is the stride. In practice, we apply our depth multiplier to n and our width multiplier to c. The original architecture of MobileNetV2 is described in Table 4 and contains 3.5M parameters. In our experiments, we use for example $w = 1.4$ and $d = 0.2$, which corresponds to a network containing 4.0M parameters distributed across blocks as described in Table 4.

• **ShuffleNetv2.** The original ShuffleNetv2 contains 2.3M parameters and is implemented as follows. A list, called `stages_repeats`, controls the number of repetitions for each type of block. An argument called `stages_out_channels` defines the number of output channels. In practice, we apply the depth multiplier to the list `stages_repeats`, and the width multiplier to `stages_out_channels`. For example, with $w = 3.0$ and $d = 0.1$, we obtain a network containing 4.3M parameters, distributed across layers as described in Table 5.

3. Scaling experiments

We consider the scaling of ResNet18, MobileNetv2 and ShuffleNetv2, for two examples of small memory budgets (2.5M and 5.0M parameters). In Figures 1, 2 and 3 we report the average incremental accuracy of scaled models trained using a memory-free version of the LUCIR algorithm. The classification problem consists in learning one hundred classes from the INRand_0 dataset, equally distributed over ten states. For a given backbone network and the two memory budgets, the performances of different configurations (w, d), including the initial architecture (in red) and the one recommended by our heuristic (in green) are presented.

4. Experiments using reference configurations

On average across all scenarios, the classification performances of datasets INFood, INFlora and INFauna, that correspond to fine-grained classifications, are lower than the classification performances of datasets INRand_0 and INRand_1, that correspond to coarser classifications. This is illustrated in Figure 4 where classification performances corresponding to experiments corresponding to the set of reference configurations are displayed in a boxplot by dataset.

[6] https://pytorch.org/vision/0.8/_modules/torchvision/models/resnet.html#resnet18

[7] https://pytorch.org/vision/0.8/_modules/torchvision/models/mobilenet.html#mobilenet_v2

[8] https://pytorch.org/vision/0.8/_modules/torchvision/models/shufflenetv2.html#shufflenet_v2_x1_0

layer name	output size	ResNet18 ($w = 1.0$, $d = 1.0$)	Example ($w = 0.5$, $d = 0.5$)
Image	224 x 224	-	-
conv_1	112x112	7x7, 64, stride 2	7x7, 64, stride 2
conv_2	56x56	3x3, maxpool, stride 2 $\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix}$ x2	3x3, maxpool, stride 2 $\begin{bmatrix} 3 \times 3, 32 \\ 3 \times 3, 32 \end{bmatrix}$ x1
conv_3	28x28	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix}$ x2	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix}$ x1
conv_4	14x14	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix}$ x2	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix}$ x1
conv_5	1x1	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix}$ x2	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix}$ x1
		avgpoof, fc, softmax	avgpoof, fc, softmax
Nbr. params.	100 classes	11.7M	1.5M

Table 3: Architectures of ResNet18 and of scaled version of it. The building blocks are indicated in brackets. Each block is indicated with its number of repetitions in the network. The depth multiplier d is applied to the number of blocks (pink). The width multiplier w is applied to the number of convolutional filters in each block (blue). Down-sampling is performed by conv_3, conv_4, and conv_5 with a stride of 2. The number of parameters is computed for 100 classes.

Network		MobileNetV2 ($w = 1.0$, $d = 1.0$)				Example ($w = 1.4$, $d = 0.2$)			
layer name	output size	t	c	n	s	t	c	n	s
Image	224 x 224	-	3	-	-	-	3	-	-
conv2d	112 x 112	-	32	1	2	-	32	1	2
bottleneck	112 x 112	1	16	1	1	1	24	1	1
bottleneck	56 x 56	6	24	2	2	6	32	1	2
bottleneck	28 x 28	6	32	3	2	6	48	1	2
bottleneck	14 x 14	6	64	4	2	6	88	1	2
bottleneck	14 x 14	6	96	3	1	6	136	1	1
bottleneck	7 x 7	6	160	3	2	6	224	1	2
bottleneck	7 x 7	6	320	1	1	6	448	1	1
conv2d 1x1	7 x 7	-	1280	1	1	-	1280	1	1
conv2d 7x7	1x1x1280	-	-	1	-	-	-	-	-
conv2d 1x1	100	-	100	-	-	-	100	-	-
Nbr. params.	100 classes	3.5M				4.0M			

Table 4: Architecture of MobileNetV2 and of its scaled version. The depth multiplier d is applied to the number of layers n of bottleneck layers (pink). The width multiplier w is applied to the number of conv. filters c of bottleneck layers (blue).

Network		ShuffleNetv2 ($w = 1.0$, $d = 1.0$)		Example ($w = 3.0$, $d = 0.1$)	
layer name	output size	repeat	output channels	repeat	output channels
Image	224 x 224	-	3	-	3
Conv1 MaxPool	112 x 112 56 x 56	1	24	1	72
Stage2	28 x 28 28 x 28	1 3	116	1 1	352
Stage3	14 x 14 14 x 14	2 1	232	1 1	696
Stage4	7 x 7 7 x 7	1 3	464	1 1	1392
Conv5	7 x 7	1	1024	1	1024
GlobalPool	1 x 1	-	-	-	-
FC	-	-	100	-	100
Nbr. params.	100 classes	2.3M		4.3M	

Table 5: Architecture of ShuffleNetv2 and of its scaled version. The depth multiplier d is applied to the number of repetition of the main layers (pink). The width multiplier w is applied to the number of convolutional filters c of these layers (blue).

As one could expect, for a given dataset, scenario, algorithm and backbone network, performance increases as the memory budget associated to the model increases. Averaged values are presented in Figure 5.

With regard to the parameters of the scenarios, we observe the following trend : the higher the number of classes in the initial step, the higher the final average incremental accuracy. On the contrary, a greater number of steps and/or a fewer number of classes per incremental step implies a lower average incremental accuracy. This is illustrated by Figure 6 where we see that on average across all datasets, memory budgets, backbone networks and algorithms, the greater the number of classes in the initial step, the higher the classification performance of the final model.

5. Experiments using test configurations

Additional baselines We presented in the article three *baseline pairs*, which are three fixed combinations: b_1: (FetrIL, ResNet), b_2: (DSLDA, ShuffleNet) and b_3: (SPB, MobileNet). We give three supplementary baselines in Table 7, namely b_4: (DeeSIL, ResNet), b_5: (LUCIR, MobileNet) and b_6: (SIW, ResNet). Their corresponding models are called *baseline models*. These pairs were selected according to their aggregated rank on reference datasets i.e. for each algorithm, we selected the backbone corresponding to the algorithm-backbone pair with the accuracy ranking the highest on all reference experiments. In practice, we present in Table 7 the classification performance of the six baseline models and of the recommended model for eighteen scenarios. Table 7 also shows how the recommended algorithm and backbone network correlate with the memory budget and the other settings of an incremental learning scenario.

Additional ablation experiments In Table 6 we show the results obtained when only a single algorithm is available. Each column corresponds to the average classification performance obtained with AdvisIL recommendations when the set of reference configurations is reduced to consider a single algorithm and multiple backbone networks.

Avg. Incr. Acc. — Fixed algorithms

AdvisIL	Δ_{FT}	Δ_{DS}	Δ_{SPB}	Δ_{Dee}	Δ_{LU}	Δ_{SIW}
50.88	-0.45	0.42	-9.74	-12.12	-16.79	-25,38

Table 6: Classification performances of AdvisIL and of its variants which use a single possible algorithm, either FeTrIL (FT), DSDLA (DS), SPB, DeeSIL (Dee), LUCIR (LU) or SIW. Results are averaged over all test scenarios and test datasets.

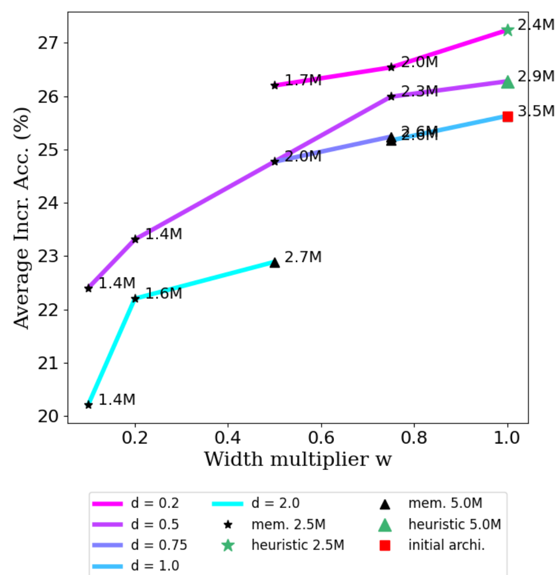

Figure 1: Scaling experiments using MobileNetv2 as initial architecture (red marker). Green markers correspond to the architecture given by our scaling heuristic for two different memory budgets (2.5M parameters and 5.0M parameters).

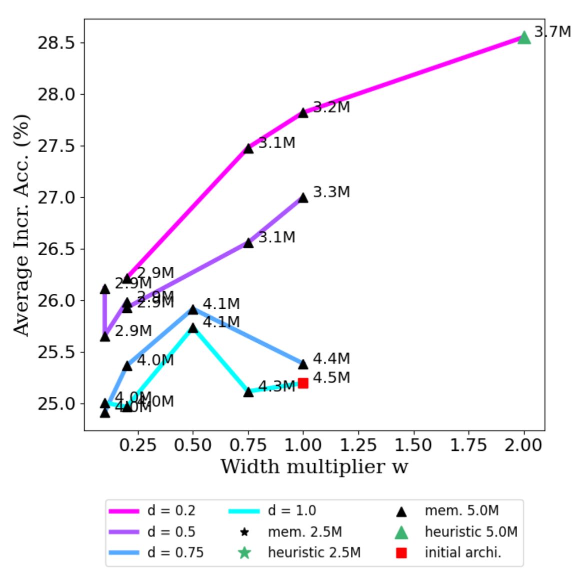

Figure 2: Scaling experiments using ShuffleNetv2 as initial architecture (red marker). Green markers correspond to the architecture given by our scaling heuristic for two different memory budgets (2.5M parameters and 5.0M parameters).

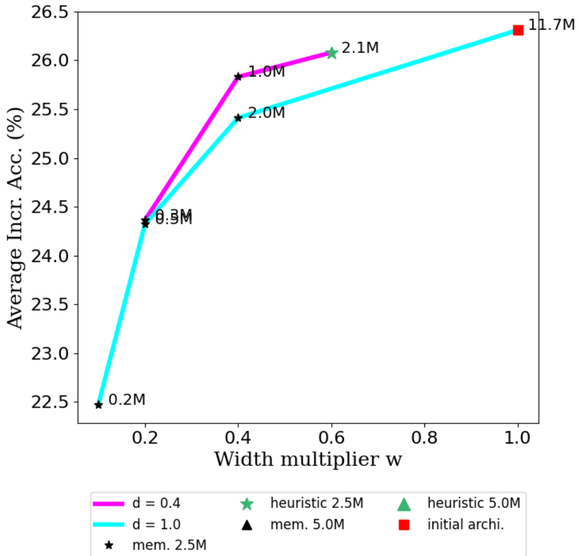

Figure 3: Scaling experiments using ResNet18 as initial architecture (red marker). Green markers correspond to the architecture given by our scaling heuristic for two different memory budgets (2.5M parameters and 5.0M parameters).

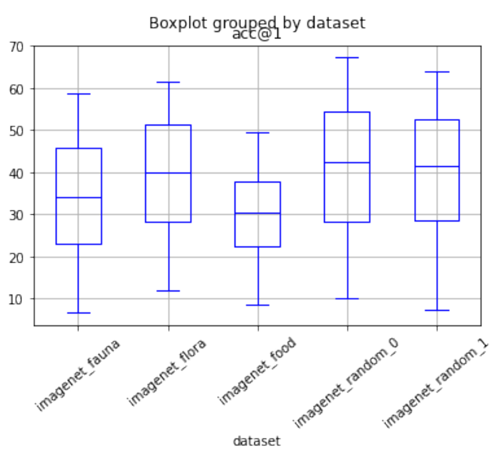

Figure 4: Boxplot grouping by dataset the average incremental accuracy (in percent) of all experiments computed using the set of reference configurations.

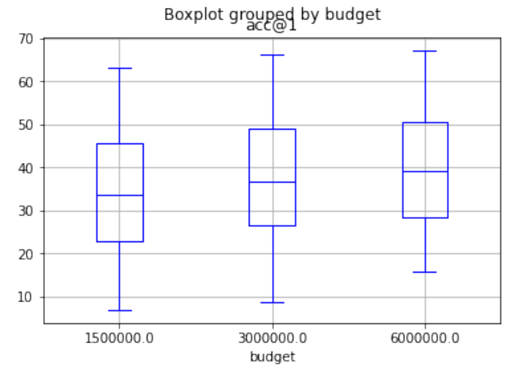

Figure 5: Boxplot grouping by memory budget the average incremental accuracy (in percent) of all experiments computed using the set of reference configurations.

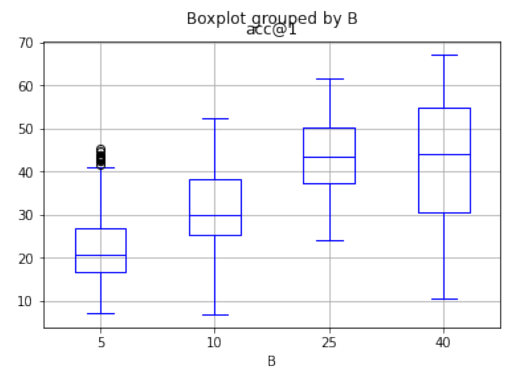

Figure 6: Boxplot grouping by initial step size the average incremental accuracy (in percent) of all experiments computed using the set of reference configurations.

References

[1] Eden Belouadah and Adrian Popescu. Deesil: Deep-shallow incremental learning. *TaskCV Workshop @ ECCV 2018.*, 2018.

[2] Eden Belouadah, Adrian Popescu, and Ioannis Kanellos. Initial classifier weights replay for memoryless class incremental learning. In *British Machine Vision Conference (BMVC)*, 2020.

[3] Tyler L Hayes and Christopher Kanan. Lifelong machine learning with deep streaming linear discriminant analysis. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 220–221, 2020.

[4] Saihui Hou, Xinyu Pan, Chen Change Loy, Zilei Wang, and Dahua Lin. Learning a unified classifier incrementally via rebalancing. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pages 831–839, 2019.

[5] Grégoire Petit, Adrian Popescu, Hugo Schindler, David Picard, and Bertrand Delezoide. Fetril: Feature translation for exemplar-free class-incremental learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2023.

[6] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[7] Guile Wu, Shaogang Gong, and Pan Li. Striking a balance between stability and plasticity for class-incremental learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1124–1133, 2021.

Combination (a, b)	Mem. budget m	CIL setting (k, α, β)						avg
		(50, 2, 2)	(25, 4, 4)	(5, 20, 20)	(13, 40, 5)	(11, 50, 5)	(6, 50, 10)	
(SPB, MobileNet)	1.5M	12.46	19.87	44.60	48.80	52.46	55.71	38.98
(DSLDA, ShuffleNet)		**12.77**	18.12	45.43	**59.72**	**61.84**	61.24	43.19
(FetrIL, ResNet)		10.62	27.75	**53.00**	58.20	61.60	**62.30**	45.58
(DeeSIL, ResNet)		6.51	17.97	50.27	41.83	44.07	53.98	35.77
(LUCIR, MobileNet)		9.76	16.26	46.00	30.63	34.65	45.58	30.48
(SIW, ResNet)		6.81	13.20	34.40	24.05	22.02	32.23	22.12
AdvisIL's pair (a, b)		11.34 (DS, Res)	**28.52** (DS, Res)	**53.00** (FT, Res)	**59.72** (DS, Shu)	**61.84** (DS, Shu)	61.24 (DS, Shu)	**45.94**
(SPB, MobileNet)	3.0M	13.78	22.09	48.57	51.47	55.10	58.45	41.58
(DSLDA, ShuffleNet)		22.81	**35.18**	55.93	**63.71**	**65.47**	64.88	51.33
(FetrIL, ResNet)		22.08	34.58	55.42	60.70	61.85	62.50	49.52
(DeeSIL, ResNet)		10.38	22.31	52.50	44.24	45.16	54.70	38.21
(LUCIR, MobileNet)		10.73	18.30	50.09	33.27	38.34	49.85	33.43
(SIW, ResNet)		9.09	15.18	35.59	25.05	30.66	32.19	24.63
AdvisIL's pair (a, b)		**24.37** (DS, Res)	34.69 (DS, Res)	**57.05** (DS,Shu)	**63.71** (DS, Shu)	**65.47** (DS, Shu)	**65.81** (DS, Mob)	**51.85**
(SPB, MobileNet)	6.0M	14.86	22.94	51.7	50.89	55.68	58.25	42.39
(DSLDA, ShuffleNet)		32.24	38.86	56.38	64.07	65.21	64.59	53.56
(FetrIL, ResNet)		**34.55**	**42.00**	55.88	61.80	63.68	64.18	53.68
(DeeSIL, ResNet)		13.99	27.03	52.74	45.33	46.14	56.60	40.30
(LUCIR, MobileNet)		12.95	21.20	52.81	38.28	41.81	52.96	36.67
(SIW, ResNet)		10.83	16.62	37.14	35.20	37.86	36.10	28.96
AdvisIL's pair (a, b)		**34.55** (FT, Res)	**42.00** (FT, Res)	**57.18** (FT, Mob)	**64.25** (DS, Mob)	**65.89** (DS, Mob)	**65.29** (DS, Mob)	**54.86**

Table 7: Classification performance for the six baseline models and for the recommended model. Performance is averaged over the four test datasets. For each scenario (m, k, α, β), the best result is in bold and the combination of algorithm (either DSLDA (DS) or FetrIL (FT)) and backbone (MobileNet (Mob) , ResNet (Res) or Shufflenet (Shu)) recommended by AdvisIL is provided.