Computer Vision

Lecture 08: Large-scale training

Contents

- Knowledge Distillation
- Self-training and pseudo labeling
- Transfer Learning
- Continual Learning

Simple PyTorch code to train CNNs

```
import torch
import torch.nn

class MyCNN(nn.Module):
  def __init__(self):
    super().__init__()
    self.layer = nn.Sequential(
        nn.Conv2d(1, 16, 5),
        nn.ReLU(),
        nn.Conv2d(16, 32, 5),
        nn.MaxPool2d(2, 2)
        nn.Conv2d(32, 64, 5)
        nn.ReLU()
        nn.MaxPool2d(2,2)
    )
    self.fc_layer = nn.Sequential(
        nn.Linear(64*3*3, 10)
        nn.ReLu()
        nn.Linear(100, 10)
    )

  def forward(self, x):
    out = self.layer(x)
    out = out.view(batch_size, -1)
    out = self.fc_layer(out)

    return out
```

```
import torch

net = MyCNN()

loss_func = torch.nn.MSELoss()
optimizer = torch.optim.SGD(net.parameters(), lr=0.01)

losses = []

for i in range(num_epoch):
  optimizer.zero_grad()

  output = net(x)

  loss = loss_func(output, y)
  loss.backward()

  optimizer.step()

  losses.append(loss.item())
```

Define different networks.
Use different losses and optimizers.
Use different (x,y) pairs.

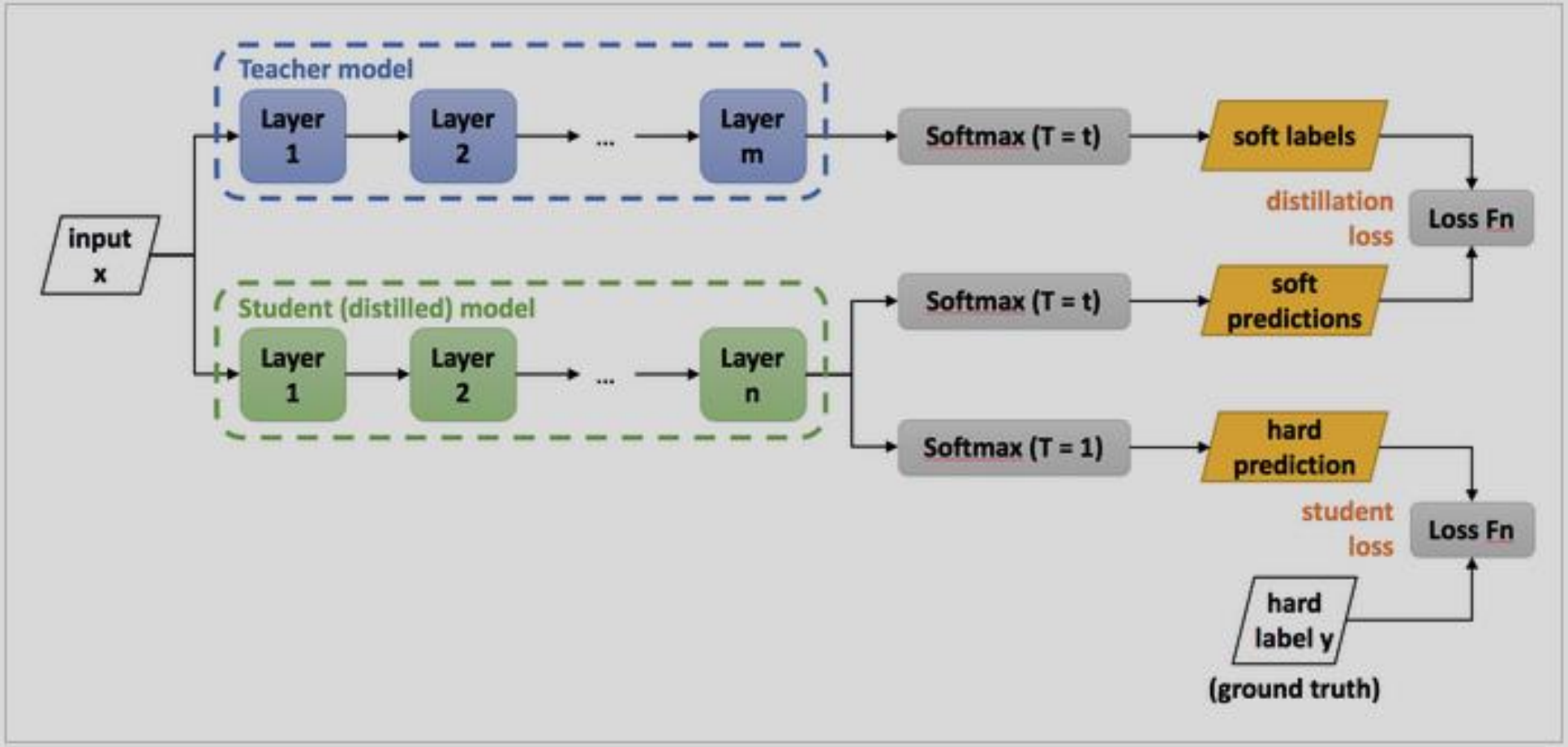
Knowledge distillation

Knowledge distillation

- The teacher network provides a richer supervisory signal than the data supervision.

- KD guides the training of a student network by encouraging it to mimic some aspect of a teacher network.

Knowledge distillation



Knowledge distillation

cow	dog	cat	car
0	1	0	0

cow	dog	cat	car
10^{-6}	.9	.1	10^{-9}

cow	dog	cat	car
.05	.3	.2	.005

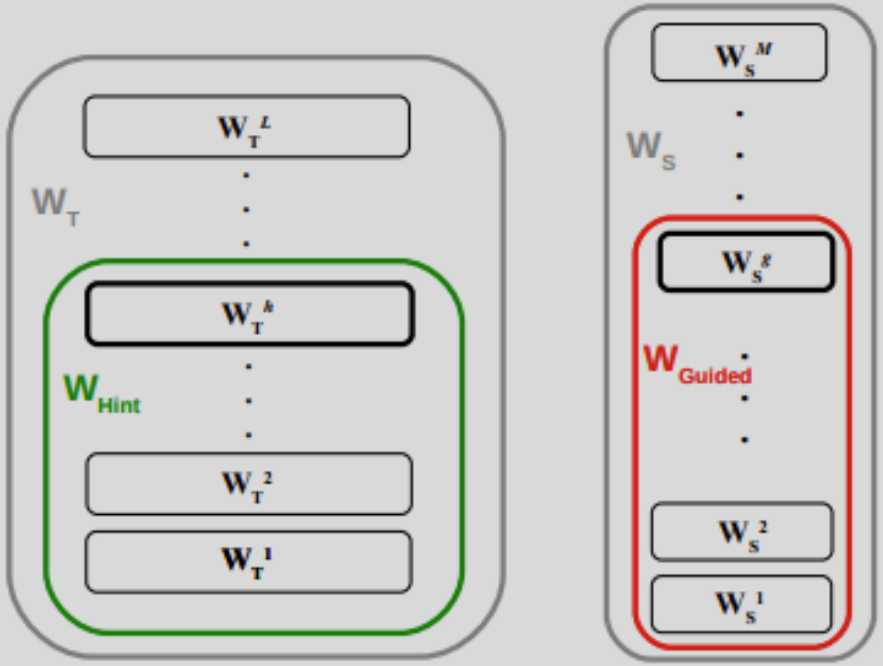
`Dark knowledge'

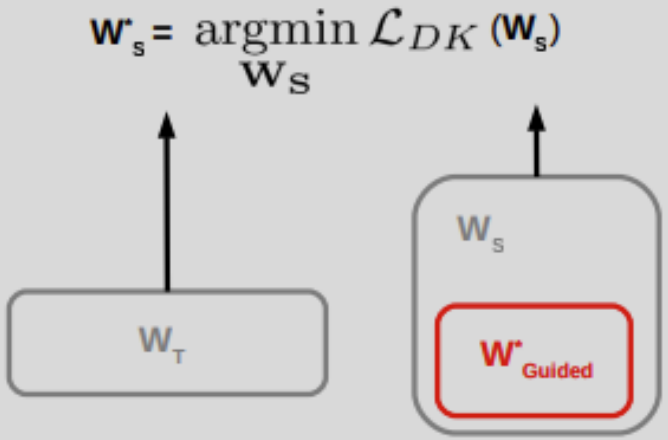
Which classes the teacher found more similar to the predicted class.

$$q_i = \frac{exp(z_i/T)}{\sum_j exp(z_j/T)}$$

q : probability
T : temperature

Knowledge distillation





$$\mathrm{P}_{\mathrm{T}}^{\tau} = \mathrm{softmax}\left(\frac{\mathbf{a}_T}{\tau}\right), \quad \mathrm{P}_{\mathrm{S}}^{\tau} = \mathrm{softmax}\left(\frac{\mathbf{a}_S}{\tau}\right) \quad \mathcal{L}_{KD}(\mathbf{W_S}) = \mathcal{H}(\mathbf{y_{true}}, \mathrm{P_S}) + \lambda \mathcal{H}(\mathrm{P}_{\mathrm{T}}^{\tau}, \mathrm{P}_{\mathrm{S}}^{\tau})$$

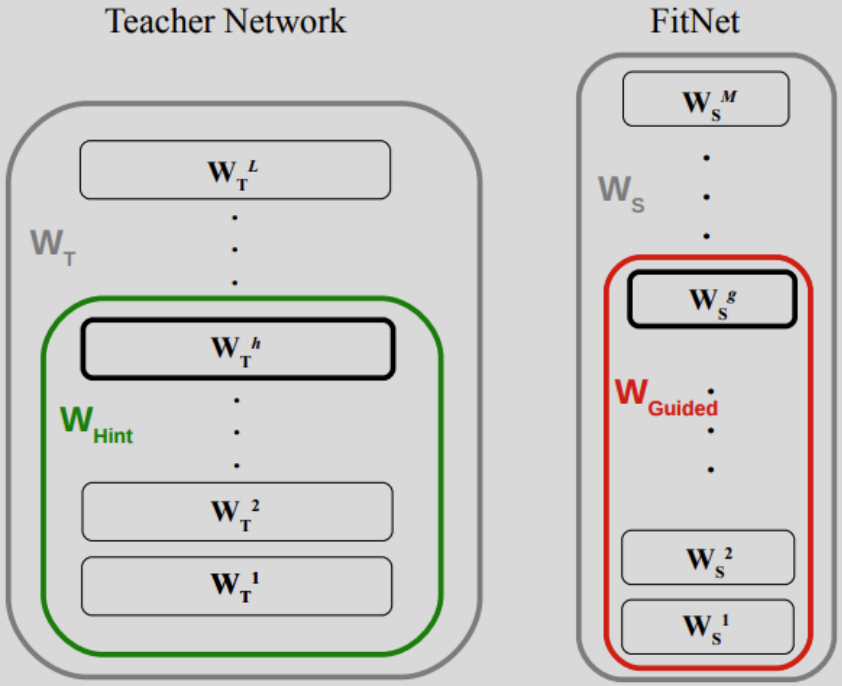
Distilling the Knowledge in a Neural Network, NeurIPS'14

Hints

- KD fails when the depth of the student network getting deeper.

- *Hint* is defined as the output of a teacher's hidden layer.

Learning hints



$$W^*_{Guided} = \underset{W_{Guided}}{\text{argmin}}\ \mathcal{L}_{HT}(W_{Guided}, W_r)$$

$$\mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r}) = \frac{1}{2}||u_h(\mathbf{x}; \mathbf{W_{Hint}}) - r(v_g(\mathbf{x}; \mathbf{W_{Guided}}); \mathbf{W_r})||^2$$

FitNets: hints for thin deep nets, ICLR'15.

Learning hints

Input: $\mathbf{W_S}, \mathbf{W_T}, g, h$
Output: $\mathbf{W_S^*}$

1: $\mathbf{W_{Hint}} \leftarrow \{\mathbf{W_T}^1, \ldots, \mathbf{W_T}^h\}$
2: $\mathbf{W_{Guided}} \leftarrow \{\mathbf{W_S}^1, \ldots, \mathbf{W_S}^g\}$
3: Intialize $\mathbf{W_r}$ to small random values
4: $\mathbf{W^*_{Guided}} \leftarrow \underset{\mathbf{W_{Guided}}}{\text{argmin}}\ \mathcal{L}_{HT}(\mathbf{W_{Guided}}, \mathbf{W_r})$
5: $\{\mathbf{W_S}^1, \ldots, \mathbf{W_S}^g\} \leftarrow \{\mathbf{W_{Guided}}^{*1}, \ldots, \mathbf{W_{Guided}}^{*g}\}$
6: $\mathbf{W_S^*} \leftarrow \underset{\mathbf{W_S}}{\text{argmin}}\, \mathcal{L}_{KD}(\mathbf{W_S})$

Intermediate representation

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**91.61%**
Teacher	~9M	90.18%
Mimic single	~54M	84.6%
Mimic single	~70M	84.9%
Mimic ensemble	~70M	85.8%
State-of-the-art methods		
Maxout		90.65%
Network in Network		91.2%
Deeply-Supervised Networks		**91.78%**
Deeply-Supervised Networks (19)		88.2%

Table 1: Accuracy on CIFAR-10

Algorithm	# params	Accuracy
Compression		
FitNet	~2.5M	**64.96%**
Teacher	~9M	63.54%
State-of-the-art methods		
Maxout		61.43%
Network in Network		64.32%
Deeply-Supervised Networks		**65.43%**

Table 2: Accuracy on CIFAR-100

Intermediate representation

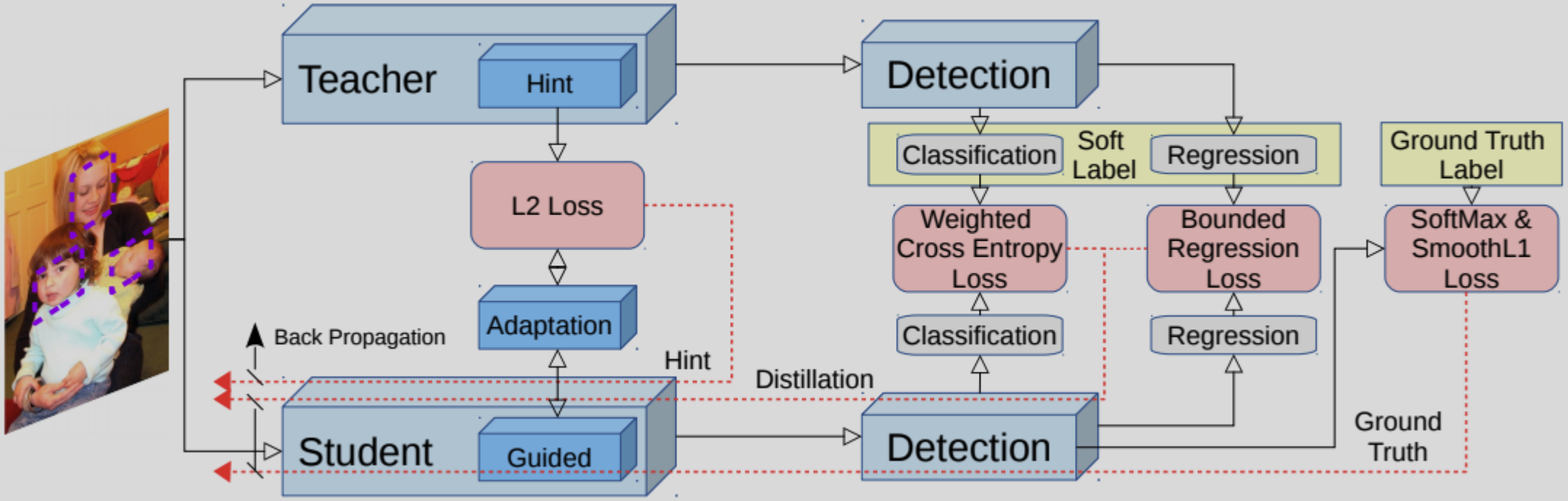
Algorithm	# params	Misclass
Compression		
FitNet	~1.5M	2.42%
Teacher	~4.9M	**2.38%**
State-of-the-art methods		
Maxout		2.47%
Network in Network		2.35%
Deeply-Supervised Networks		**1.92%**

Table 3: SVHN error

Algorithm	# params	Misclass
Compression		
Teacher	~361K	0.55%
Standard backprop	~30K	1.9%
KD	~30K	0.65%
FitNet	~30K	**0.51%**
State-of-the-art methods		
Maxout		0.45%
Network in Network		0.47%
Deeply-Supervised Networks		**0.39%**

Table 4: MNIST error

KD for object detection



Learning Efficient Object Detection Models with Knowledge Distillation, NIPS'17.

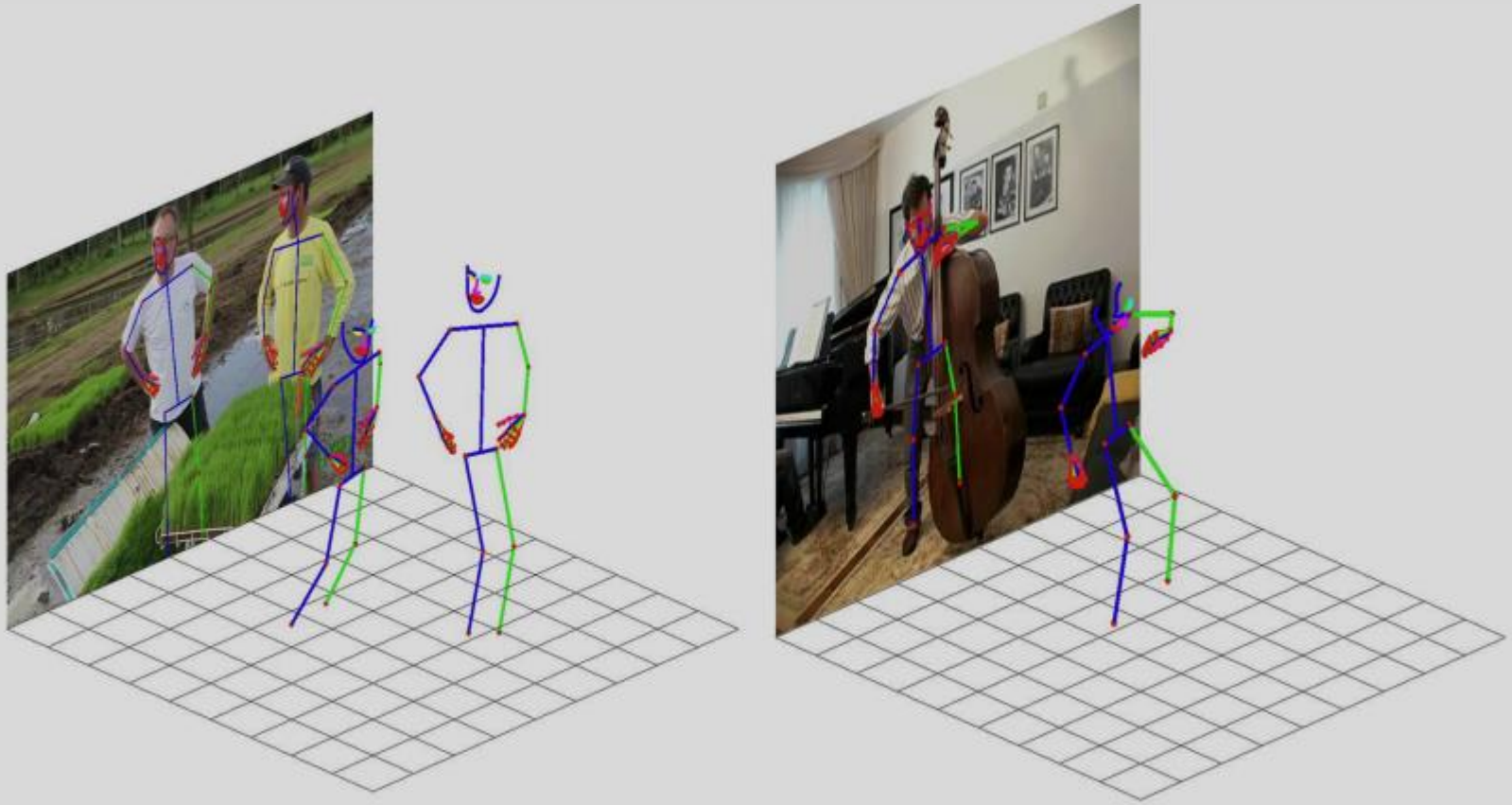
KD for object detection

		Baseline	Distillation	Hint	Distillation + Hint
PASCAL	Trainval	79.6	78.3	80.9	83.5
	Test	54.7	58.4	58	59.4
COCO	Train	45.3	45.4	47.1	49.6
	Val	25.4	26.1	27.8	28.3

learning on different datasets with Tucker and VGG16 pair.

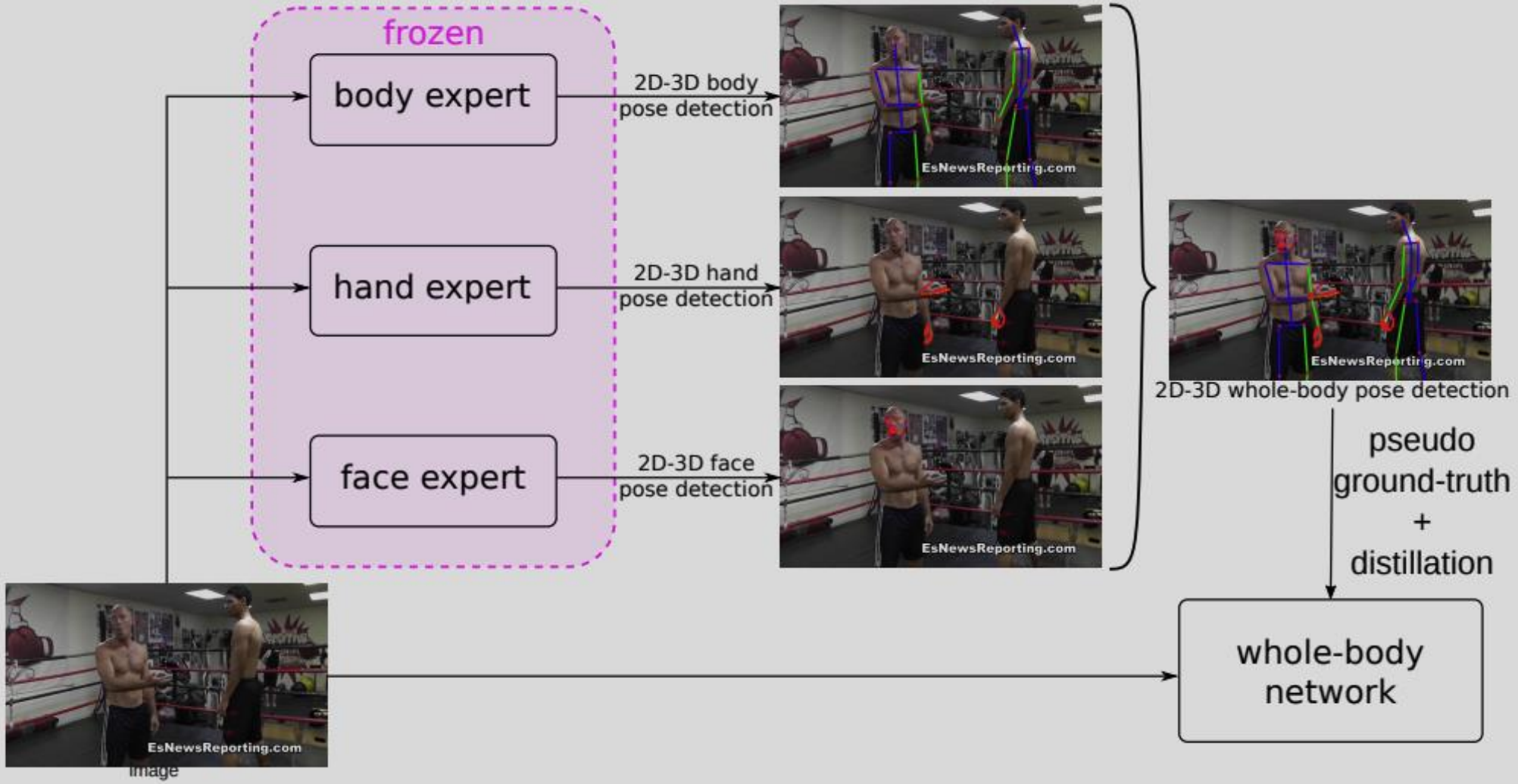
Learning Efficient Object Detection Models with Knowledge Distillation, NIPS'17.

Distillation of part experts

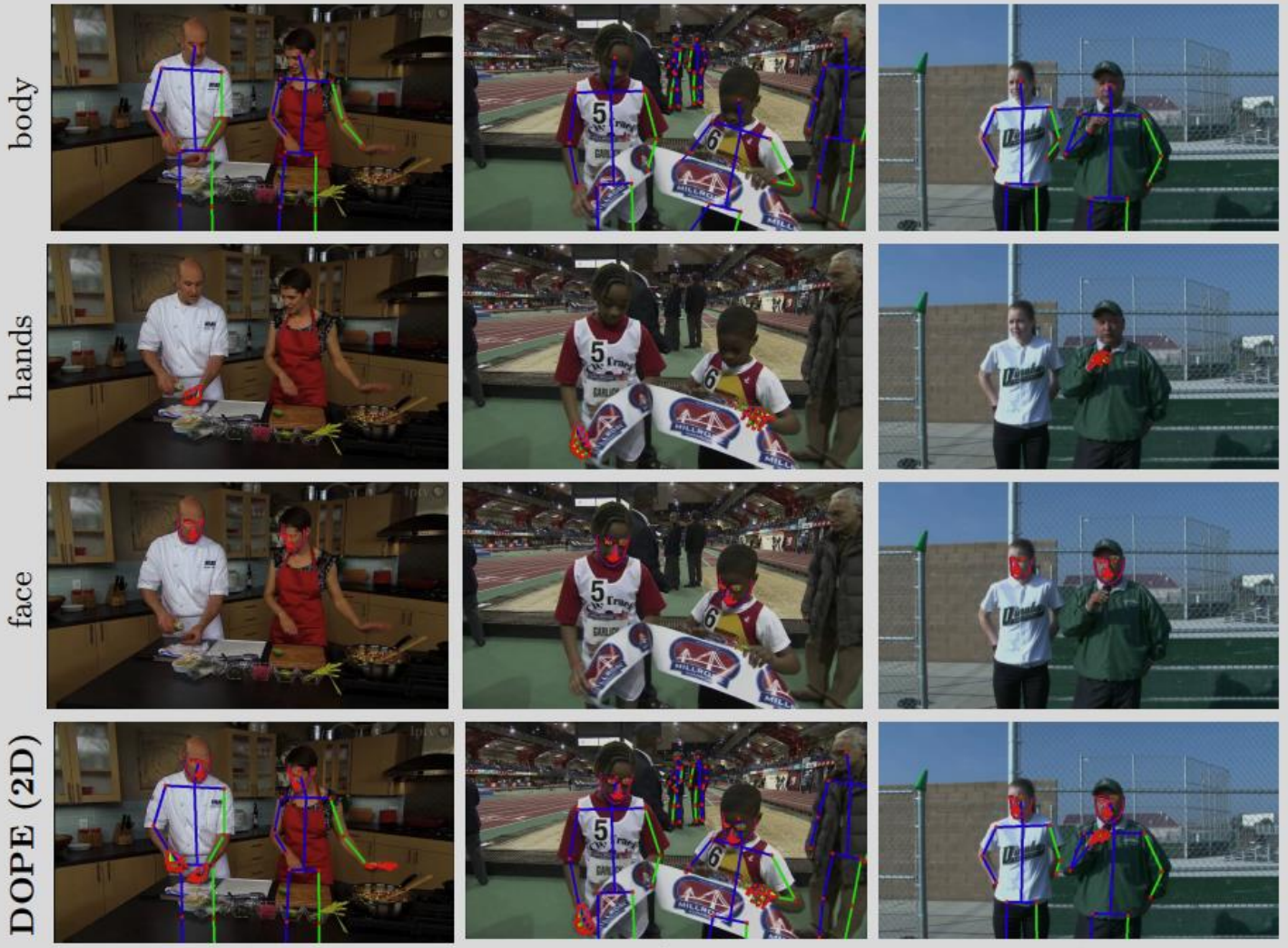


DOPE: Distillation Of Part Experts for whole-body 3D pose estimation in the wild, ECCV'20.

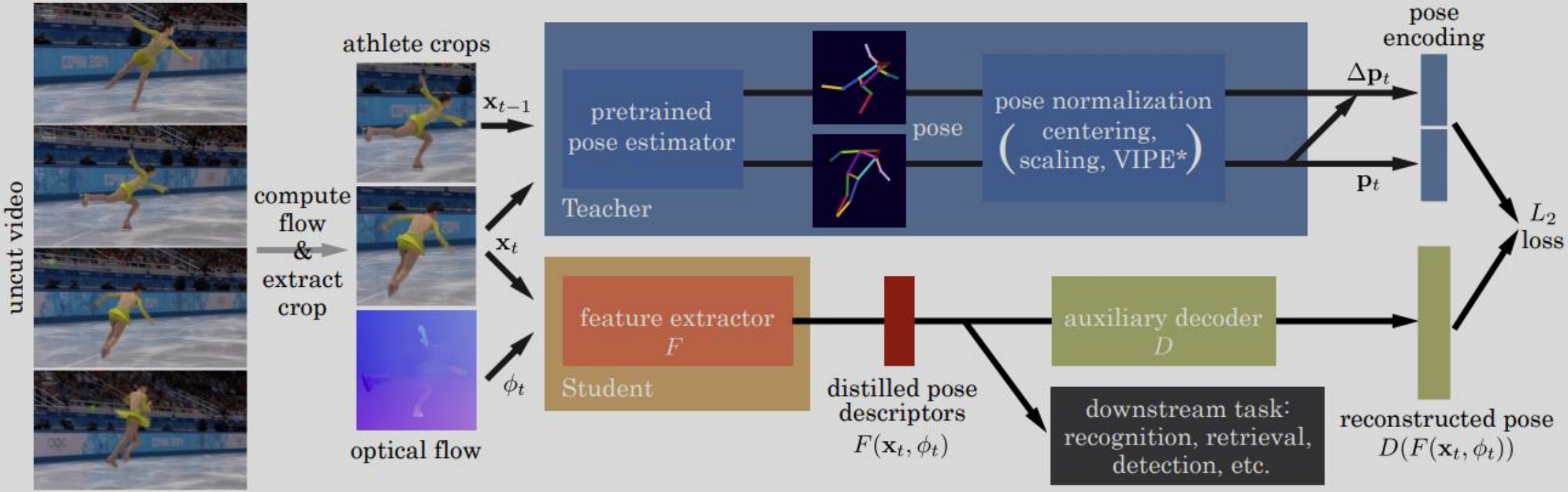
Distillation of part experts



Distillation of part experts



Video pose distillation



Video Pose Distillation for Few-Shot, Fine-Grained Sports Action Recognition, ICCV'21.

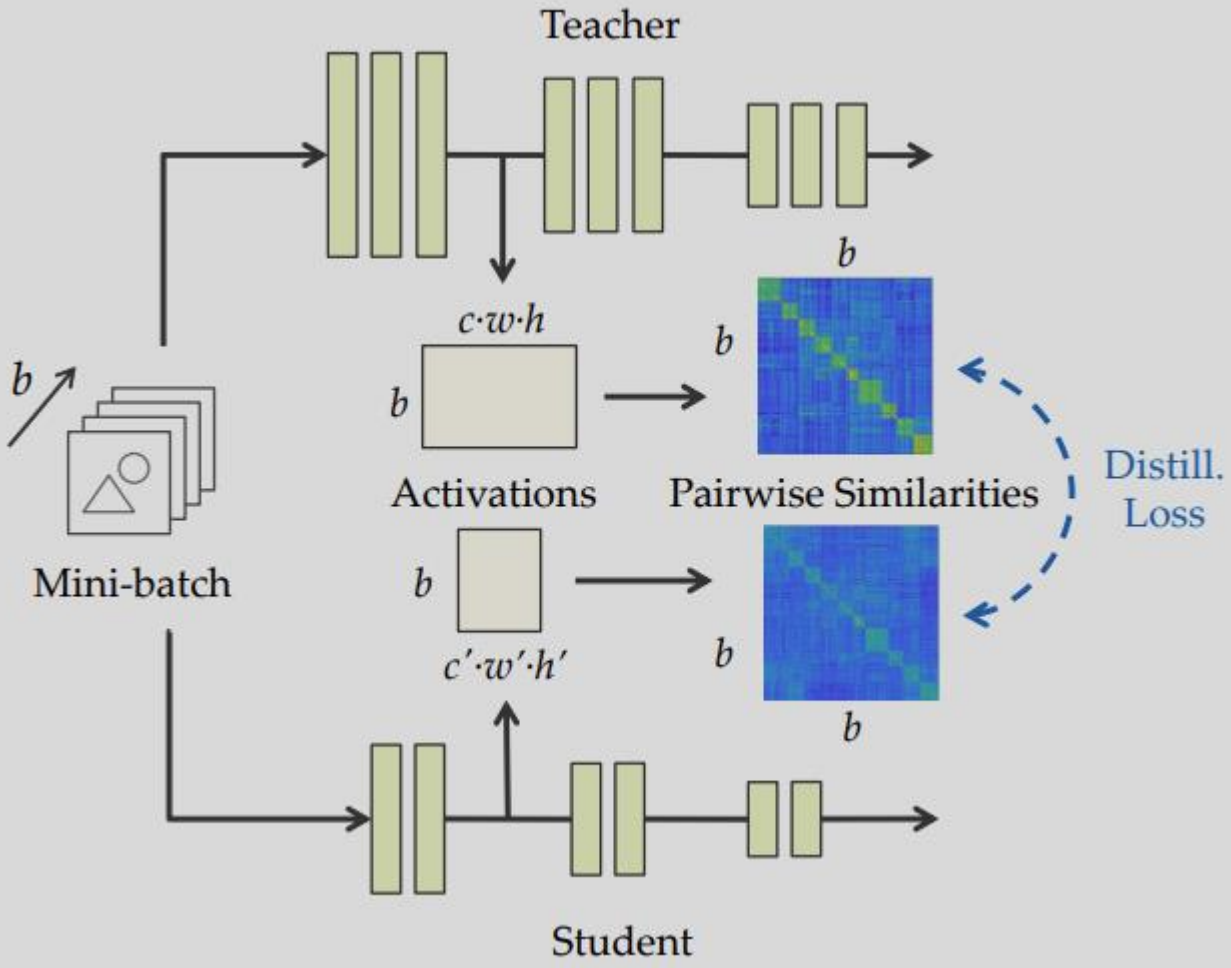
Video pose distillation

$$\Delta \mathbf{p}_t := \mathbf{p}_t - \mathbf{p}_{t-1}$$

$$\underset{F,D}{\text{minimize}} \sum_{t=1}^{N} \left\| D\left(F\left(\mathbf{x}_t, \phi_t\right)\right) - \begin{bmatrix} \mathbf{p}_t \\ \Delta \mathbf{p}_t \end{bmatrix} \right\|_2^2$$

Video Pose Distillation for Few-Shot, Fine-Grained Sports Action Recognition, ICCV'21.

Similarity-preserving KD

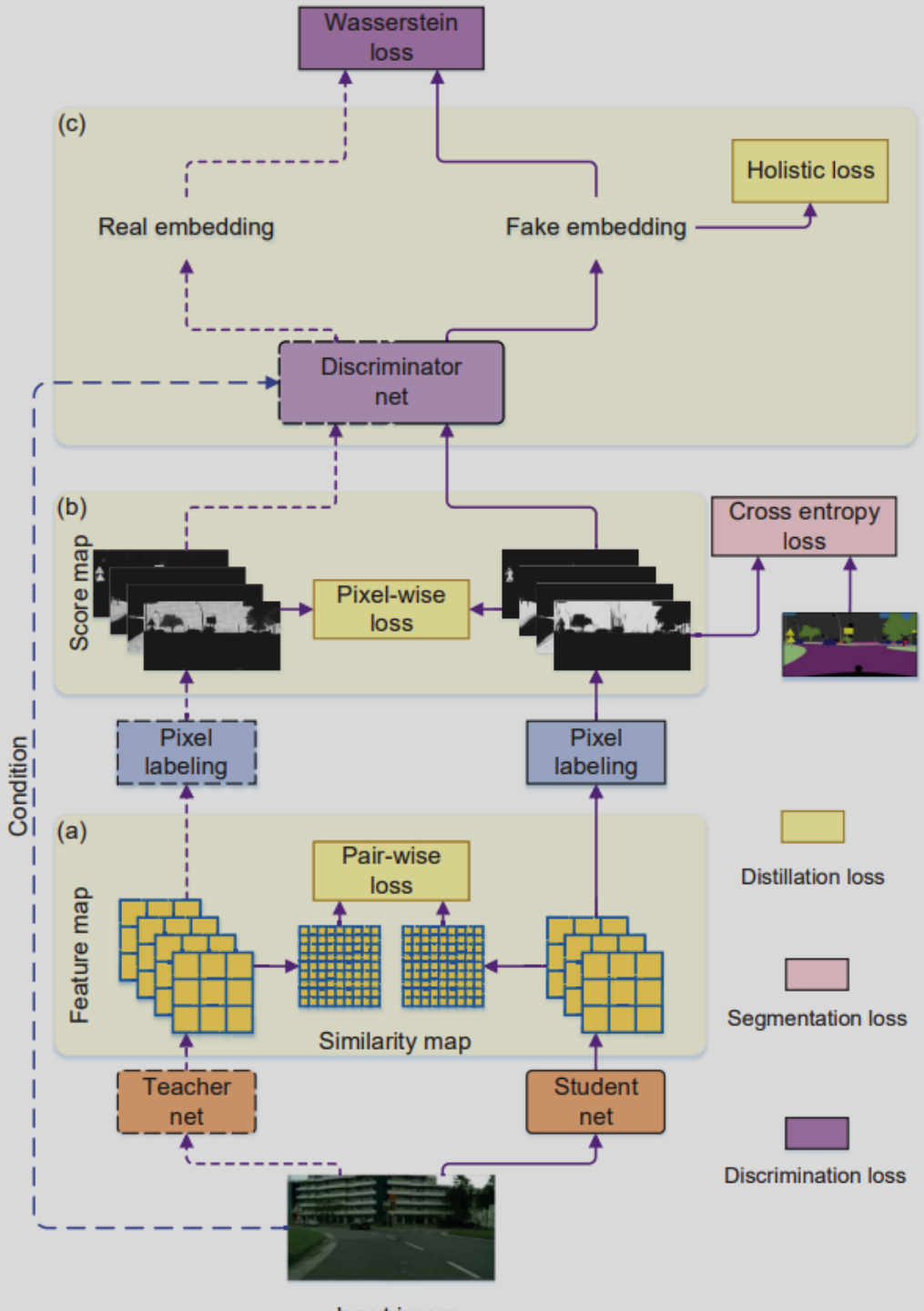


Semantically similar inputs tend to elicit similar activation patterns in a trained neural network.

Input pairs that produce similar (dissimilar) activations in the trained teacher network produce similar (dissimilar) activations in the student network.

Similarity-Preserving Knowledge Distillation, ICCV'19

preserving KD



Structured Knowledge Distillation for Semantic Segmentation, CVPR'19.

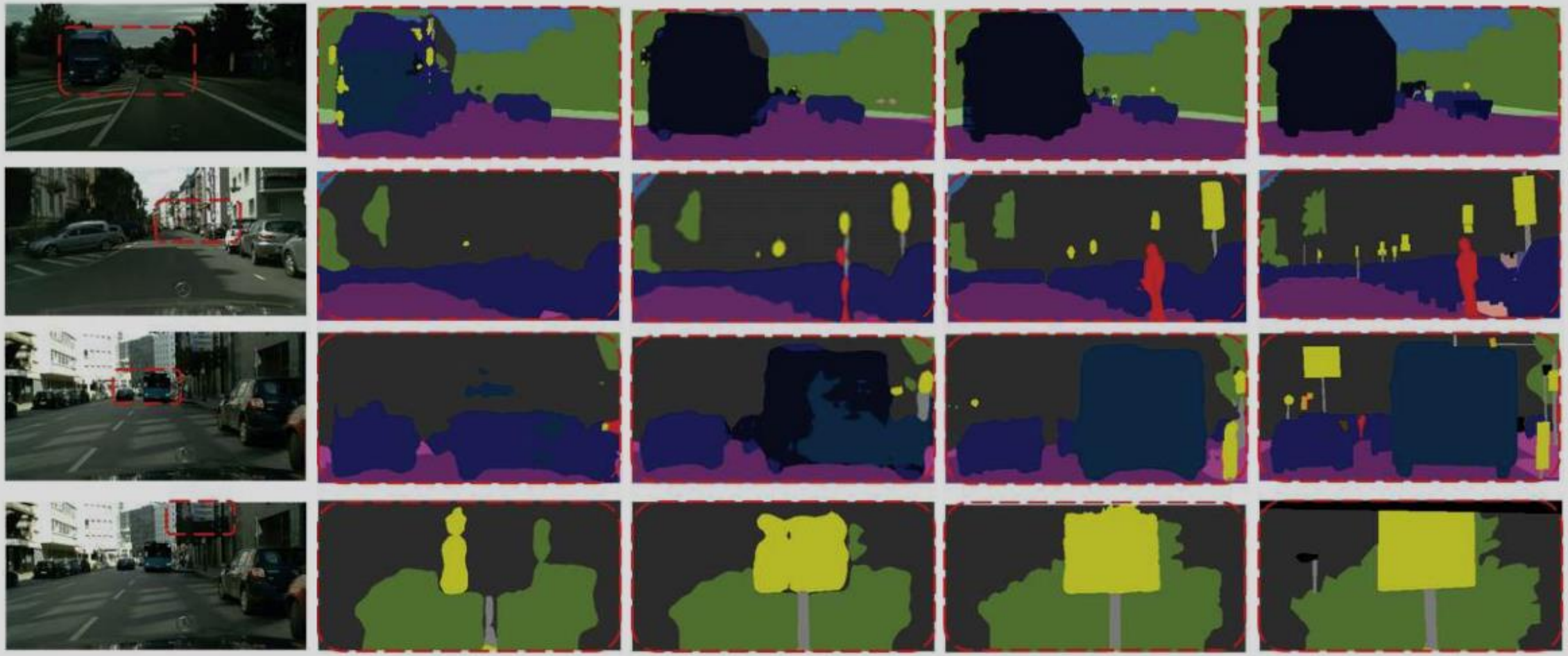
Similarity-preserving KD

Table 1: The effect of different components of the loss in the proposed method. PI = pixel-wise distillation, PA = pair-wise distillation, HO = holistic distillation, ImN = initial from the pretrain weight on the ImageNet.

Method	Validation mIoU (%)	Training mIoU (%)
Teacher	78.56	86.09
ResNet18 (0.5)	55.37 ± 0.25	60.67 ± 0.37
+ PI	57.07 ± 0.69	62.33 ± 0.66
+ PI + PA	61.03 ± 0.49	65.73 ± 0.38
+ PI + PA + HO	$\mathbf{61.63 \pm 0.99}$	$\mathbf{66.13 \pm 0.70}$
ResNet18 (1.0)	57.50 ± 0.49	62.98 ± 0.45
+ PI	58.63 ± 0.31	64.32 ± 0.32
+ PI + PA	62.48 ± 0.23	68.77 ± 0.37
+ PI + PA + HO	$\mathbf{63.24 \pm 0.74}$	$\mathbf{69.93 \pm 0.86}$
+ ImN	69.10 ± 0.21	74.12 ± 0.19
+ PI + ImN	70.51 ± 0.37	75.10 ± 0.37
+ PI + PA + ImN	71.37 ± 0.12	76.42 ± 0.20
+ PI + PA + HO + ImN	$\mathbf{72.67 \pm 0.57}$	$\mathbf{78.03 \pm 0.51}$

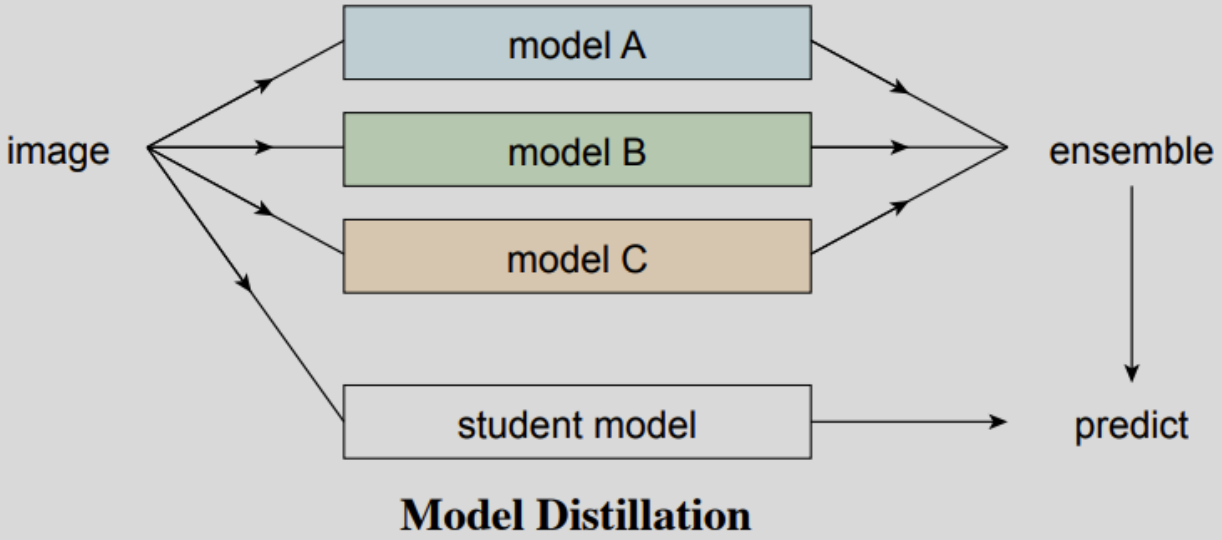
Teacher: ResNet-101
Student: ResNet-18

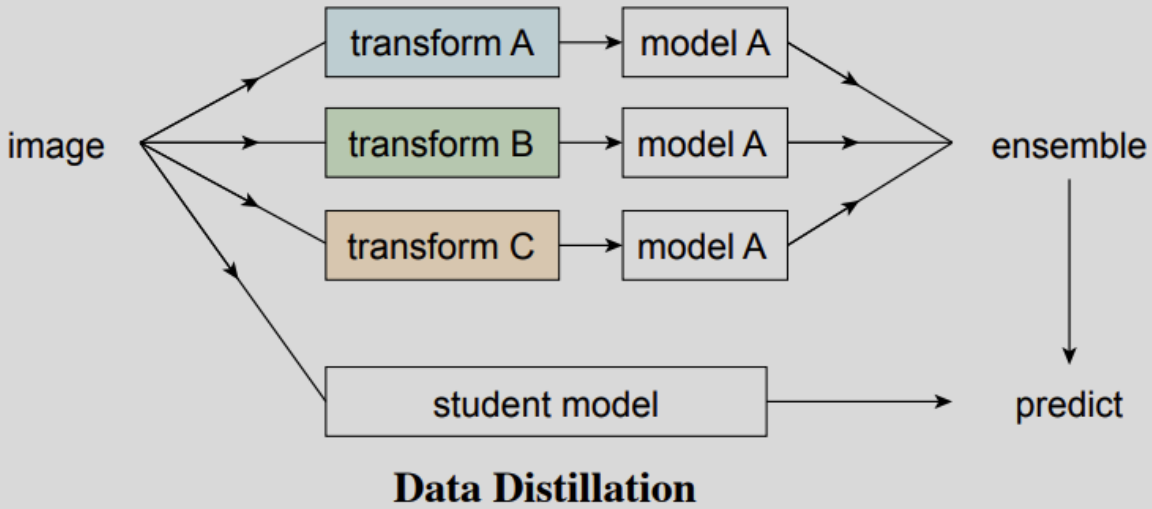
Similarity-preserving KD



(a) Image (b) W/o distillation (c) Pixel-wise distillation (d) Our method (e) Ground truth

Data distillation





Data Distillation: Towards Omni-Supervised Learning, CVPR'18.

Data distillation

Figure 2. **Ensembling keypoint predictions from multiple data transformations can yield a single superior (automatic) annotation.** For visualization purposes all images and keypoint predictions are transformed back to their original coordinate frame.

Data Distillation: Towards Omni-Supervised Learning, CVPR'18.

Data distillation

backbone	DD	AP	AP_{50}	AP_{75}	AP_M	AP_L
ResNet-50		65.1	86.6	70.9	59.9	73.6
ResNet-50	✓	**67.1**	**87.9**	**73.4**	**62.2**	**75.1**
ResNet-101		66.1	87.7	71.7	60.5	75.0
ResNet-101	✓	**67.8**	**88.2**	**73.8**	**62.8**	**76.0**
ResNeXt-101-32×4		66.8	87.5	73.0	61.6	75.2
ResNeXt-101-32×4	✓	**68.7**	**88.9**	**75.1**	**63.9**	**76.7**
ResNeXt-101-64×4		67.3	88.0	73.3	62.2	75.6
ResNeXt-101-64×4	✓	**69.1**	**88.9**	**75.3**	**64.1**	**77.1**

	#iter	AP	AP_{50}	AP_{75}	AP_M	AP_L
fully-supervised	90k	64.2	86.4	69.2	59.1	72.6
	130k	**65.1**	**86.6**	**70.9**	**59.9**	**73.6**
	270k	64.7	**86.6**	70.4	59.7	73.0
data distillation	90k	63.6	85.9	69.2	58.8	71.7
	180k	65.8	87.3	71.6	60.8	74.2
	270k	66.5	**88.0**	72.2	61.5	74.6
	360k	**66.6**	87.3	**72.6**	**61.6**	**75.0**

Data Distillation: Towards Omni-Supervised Learning, CVPR'18.

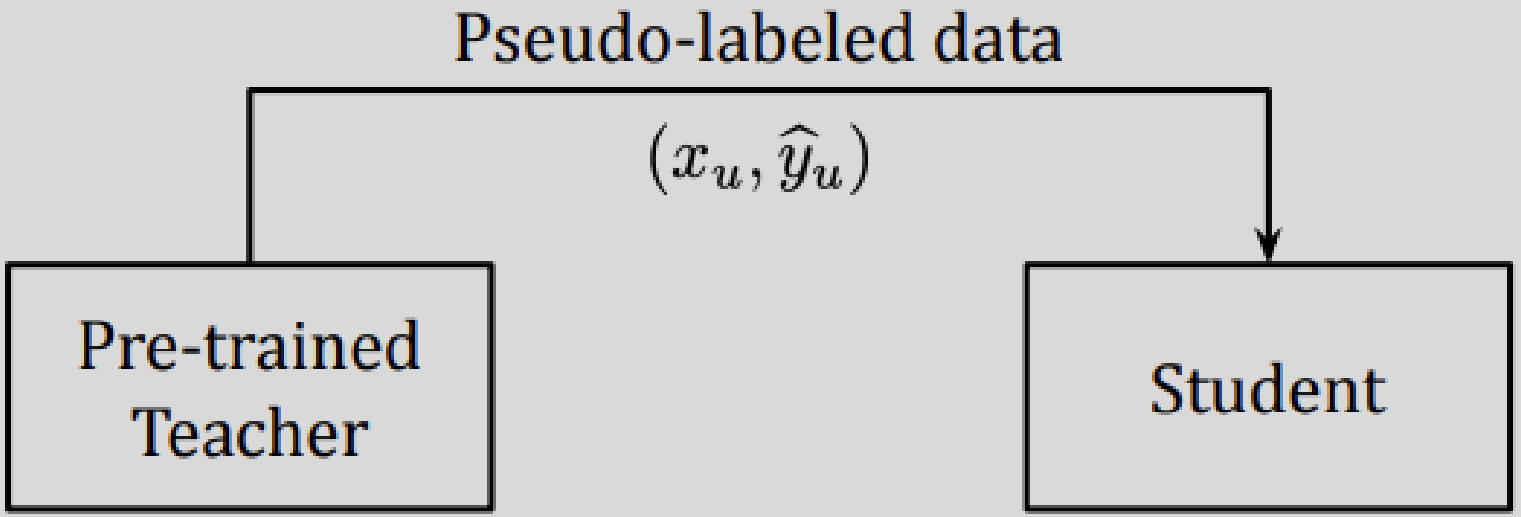
Pseudo labeling

Pseudo labels

- Networks are trained in a supervised fashion jointly with labeled and unlabeled data.

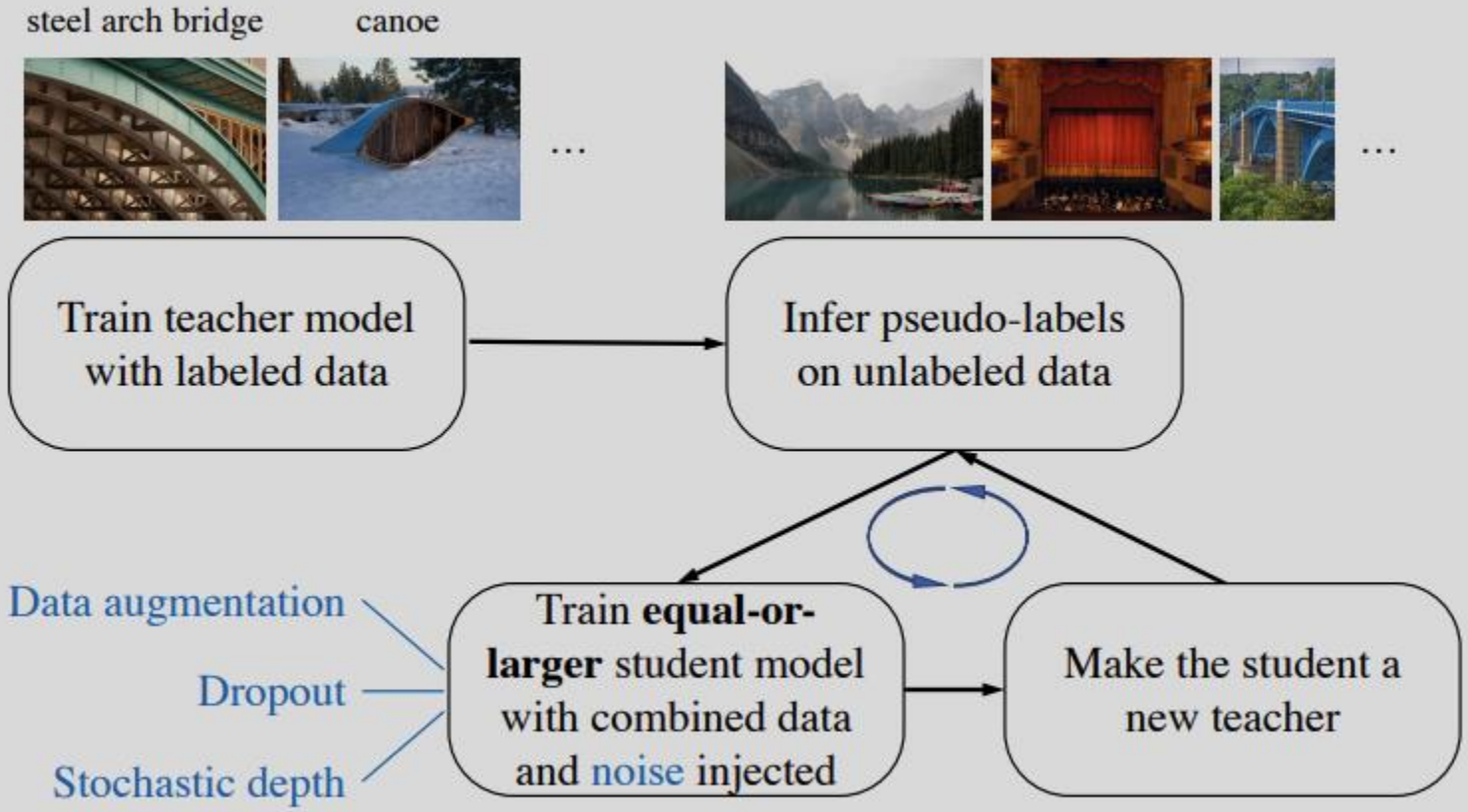
- Pseudo-Labels are target classes for unlabeled data predicted from another network as if they were true labels.

Pseudo labels



$$\theta_S^{\mathrm{PL}} = \underset{\theta_S}{\operatorname{argmin}} \underbrace{\mathbb{E}_{x_u}\Big[\mathrm{CE}\big(T(x_u;\theta_T), S(x_u;\theta_S)\big)\Big]}_{:=\mathcal{L}_u\left(\theta_T,\theta_S\right)}$$

Self-training



Self-training with noisy student improves imagenet classification, CVPR'20.

Self-training

Require: Labeled images $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ and unlabeled images $\{\tilde{x}_1, \tilde{x}_2, ..., \tilde{x}_m\}$.

1: Learn teacher model θ_*^t which minimizes the cross entropy loss on labeled images

$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f^{noised}(x_i, \theta^t))$$

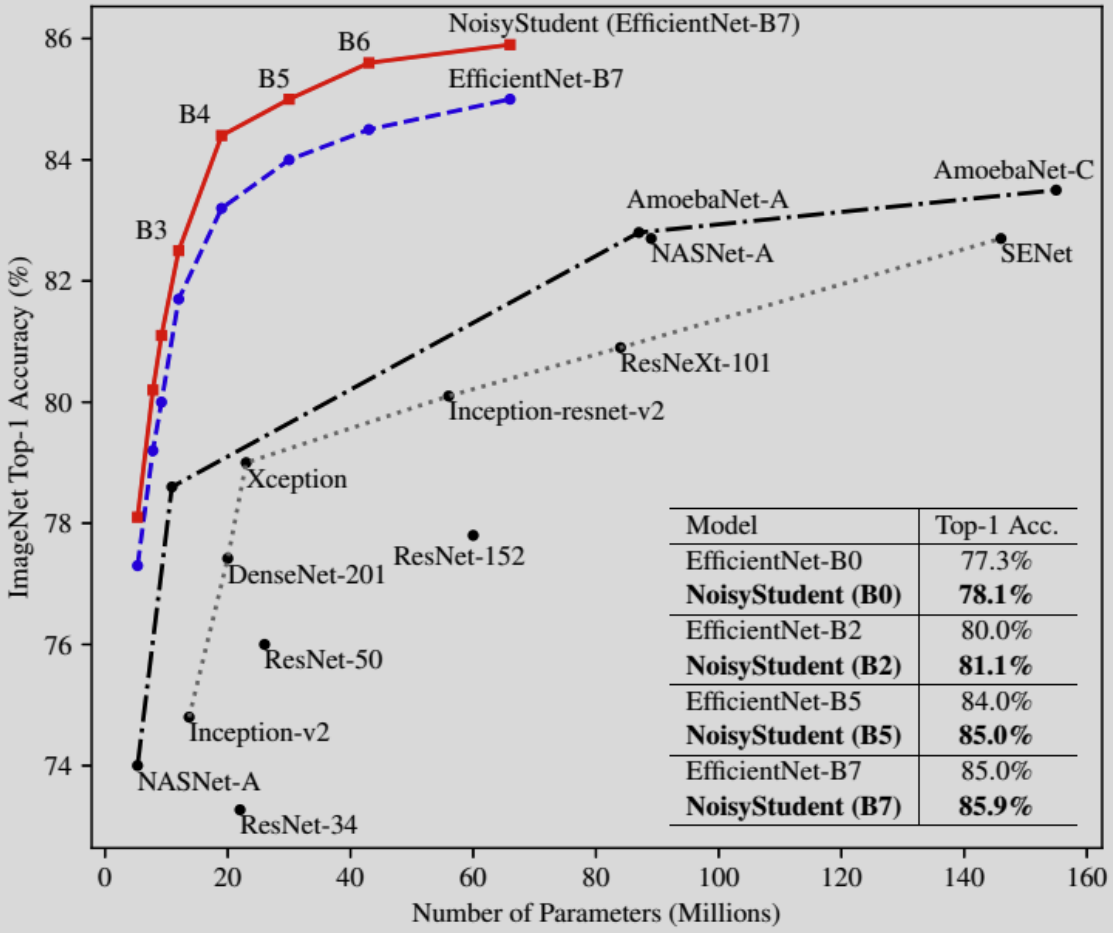
2: Use an unnoised teacher model to generate soft or hard pseudo labels for unlabeled images

$$\tilde{y}_i = f(\tilde{x}_i, \theta_*^t), \forall i = 1, \cdots, m$$

3: Learn an **equal-or-larger** student model θ_*^s which minimizes the cross entropy loss on labeled images and unlabeled images with **noise** added to the student model

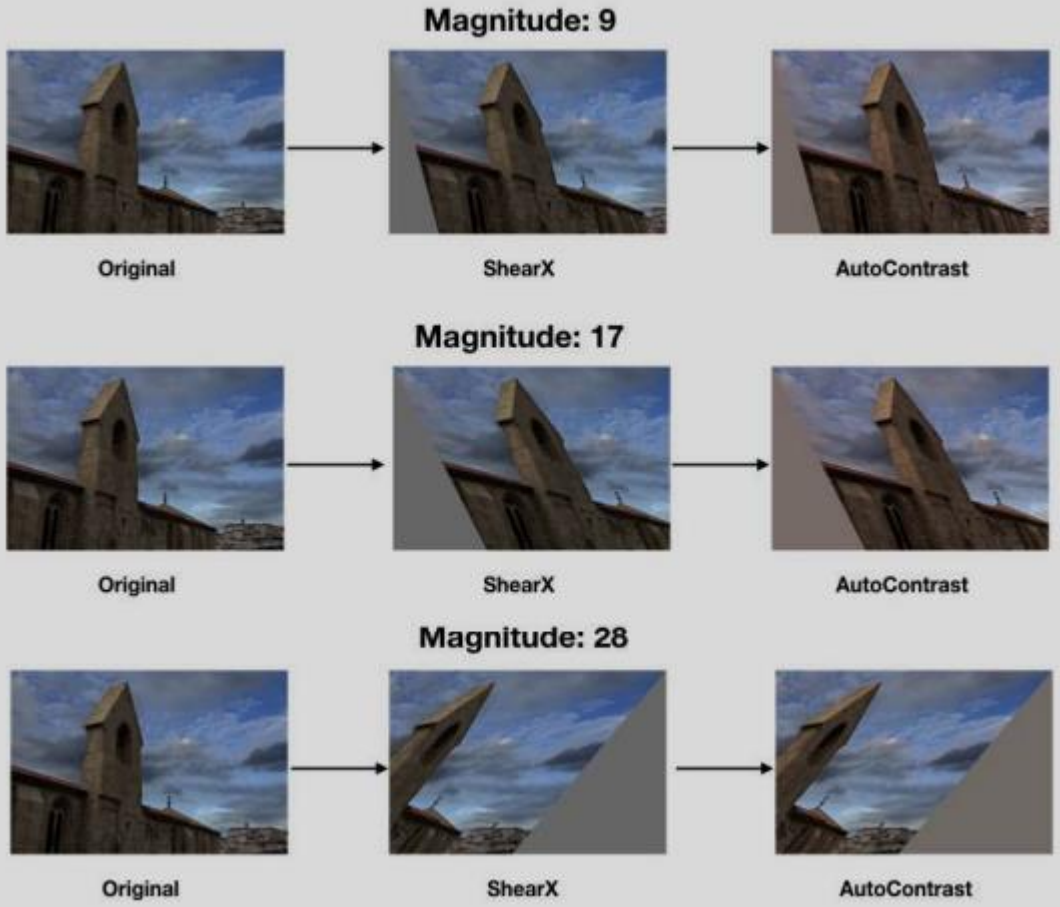
$$\frac{1}{n}\sum_{i=1}^{n} \ell(y_i, f^{noised}(x_i, \theta^s)) + \frac{1}{m}\sum_{i=1}^{m} \ell(\tilde{y}_i, f^{noised}(\tilde{x}_i, \theta^s))$$

4: Iterative training: Use the student as a teacher and go back to step 2.



Model	Top-1 Acc.
EfficientNet-B0	77.3%
NoisyStudent (B0)	**78.1%**
EfficientNet-B2	80.0%
NoisyStudent (B2)	**81.1%**
EfficientNet-B5	84.0%
NoisyStudent (B5)	**85.0%**
EfficientNet-B7	85.0%
NoisyStudent (B7)	**85.9%**

Self-training (Noise)



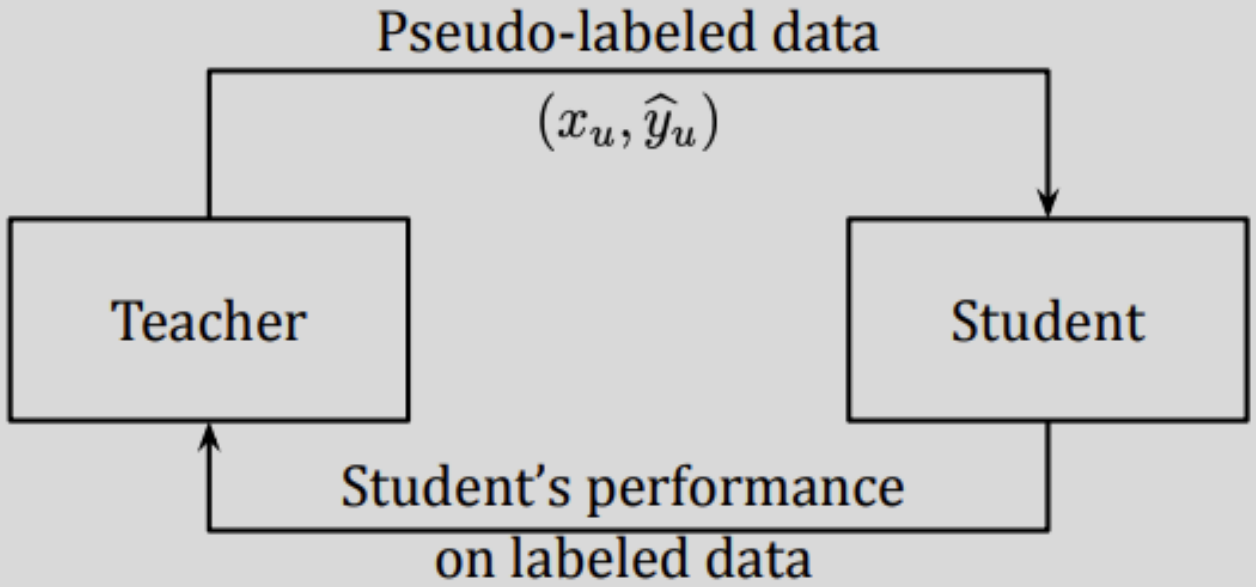

```
transforms = [
'Identity', 'AutoContrast', 'Equalize', 'Rotate',
    'Solarize', 'Color',
'Posterize', 'Contrast', 'Brightness', 'Sharpness',
    'ShearX', 'ShearY',
'TranslateX', 'TranslateY']

def randaugment(N, M):
"""Generate a set of distortions.

  Args:
    N: Number of augmentation transformations to apply
        sequentially.
    M: Magnitude for all the transformations.
"""

  sampled_ops = np.random.choice(transforms, N)
  return [(op, M) for op in sampled_ops]
```

Meta pseudo labels



$$\min_{\theta_T} \quad \mathcal{L}_l\left(\theta_S^{\mathrm{PL}}(\theta_T)\right),$$
$$\text{where} \quad \theta_S^{\mathrm{PL}}(\theta_T) = \operatorname*{argmin}_{\theta_S} \mathcal{L}_u\left(\theta_T, \theta_S\right).$$

Meta Pseudo Labels, CVPR'21

Become bad if the pseudo labels are inaccurate.
Systematic mechanism for the teacher to correct the bias.

Meta pseudo labels

$$\min_{\theta_T} \quad \mathcal{L}_l\left(\theta_S^{\mathrm{PL}}(\theta_T)\right),$$
$$\text{where} \quad \theta_S^{\mathrm{PL}}(\theta_T) = \operatorname*{argmin}_{\theta_S} \mathcal{L}_u\left(\theta_T, \theta_S\right).$$

Approximation:

$$\theta_S^{\mathrm{PL}}(\theta_T) \approx \theta_S - \eta_S \cdot \nabla_{\theta_S} \mathcal{L}_u\left(\theta_T, \theta_S\right)$$

$$\min_{\theta_T} \quad \mathcal{L}_l\Big(\theta_S - \eta_S \cdot \nabla_{\theta_S} \mathcal{L}_u\left(\theta_T, \theta_S\right)\Big)$$

Meta pseudo labels

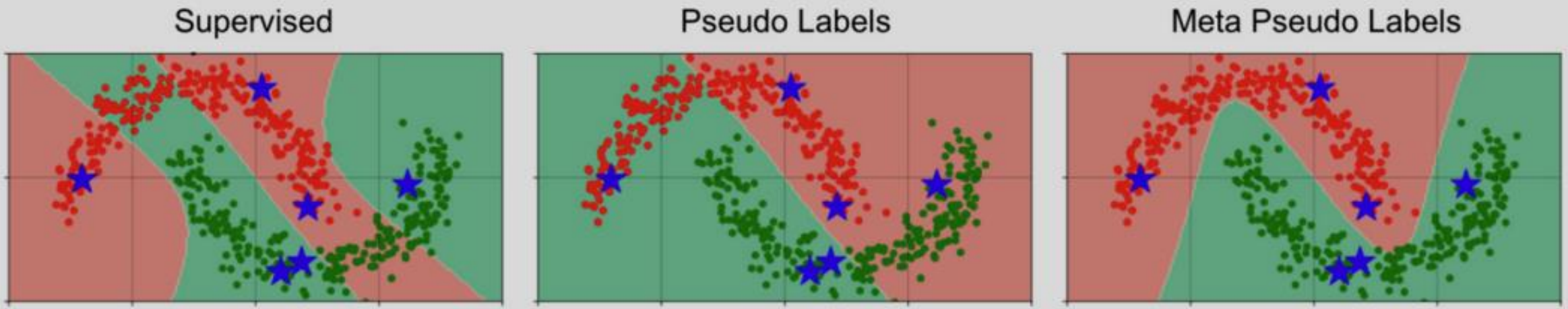
- Student: draw a batch of unlabeled data x_u, then sample $T(x_u; \theta_T)$ from teacher's prediction, and optimize objective 1 with SGD: $\theta'_S = \theta_S - \eta_S \nabla_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S)$,

- Teacher: draw a batch of labeled data (x_l, y_l), and "reuse" the student's update to optimize objective 3 with SGD: $\theta'_T = \theta_T - \eta_T \nabla_{\theta_T} \mathcal{L}_l\big(\underbrace{\theta_S - \nabla_{\theta_S} \mathcal{L}_u(\theta_T, \theta_S)}_{= \theta'_S \text{ reused from student's update}}\big)$.

Meta pseudo labels

	Method	CIFAR-10-4K (mean ± std)	SVHN-1K (mean ± std)	ImageNet-10% Top-1	ImageNet-10% Top-5
Label Propagation Methods	Temporal Ensemble [35]	83.63 ± 0.63	92.81 ± 0.27	–	–
	Mean Teacher [64]	84.13 ± 0.28	94.35 ± 0.47	–	–
	VAT + EntMin [44]	86.87 ± 0.39	94.65 ± 0.19	–	83.39
	LGA + VAT [30]	87.94 ± 0.19	93.42 ± 0.36	–	–
	ICT [71]	92.71 ± 0.02	96.11 ± 0.04	–	–
	MixMatch [5]	93.76 ± 0.06	96.73 ± 0.31	–	–
	ReMixMatch [4]	94.86 ± 0.04	97.17 ± 0.30	–	–
	EnAET [72]	94.65	97.08	–	–
	FixMatch [58]	95.74 ± 0.05	97.72 ± 0.38	71.5	89.1
	UDA* [76]	94.53 ± 0.18	97.11 ± 0.17	68.07	88.19
Self-Supervised Methods	SimCLR [8, 9]	–	–	71.7	90.4
	MOCOv2 [10]	–	–	71.1	–
	PCL [38]	–	–	–	85.6
	PIRL [43]	–	–	–	84.9
	BYOL [21]	–	–	68.8	89.0
	Meta Pseudo Labels	**96.11 ± 0.07**	**98.01 ± 0.07**	**73.89**	**91.38**
	Supervised Learning with full dataset*	94.92 ± 0.17	97.41 ± 0.16	76.89	93.27

Meta pseudo labels



Even works better than the supervised method for specific cases.

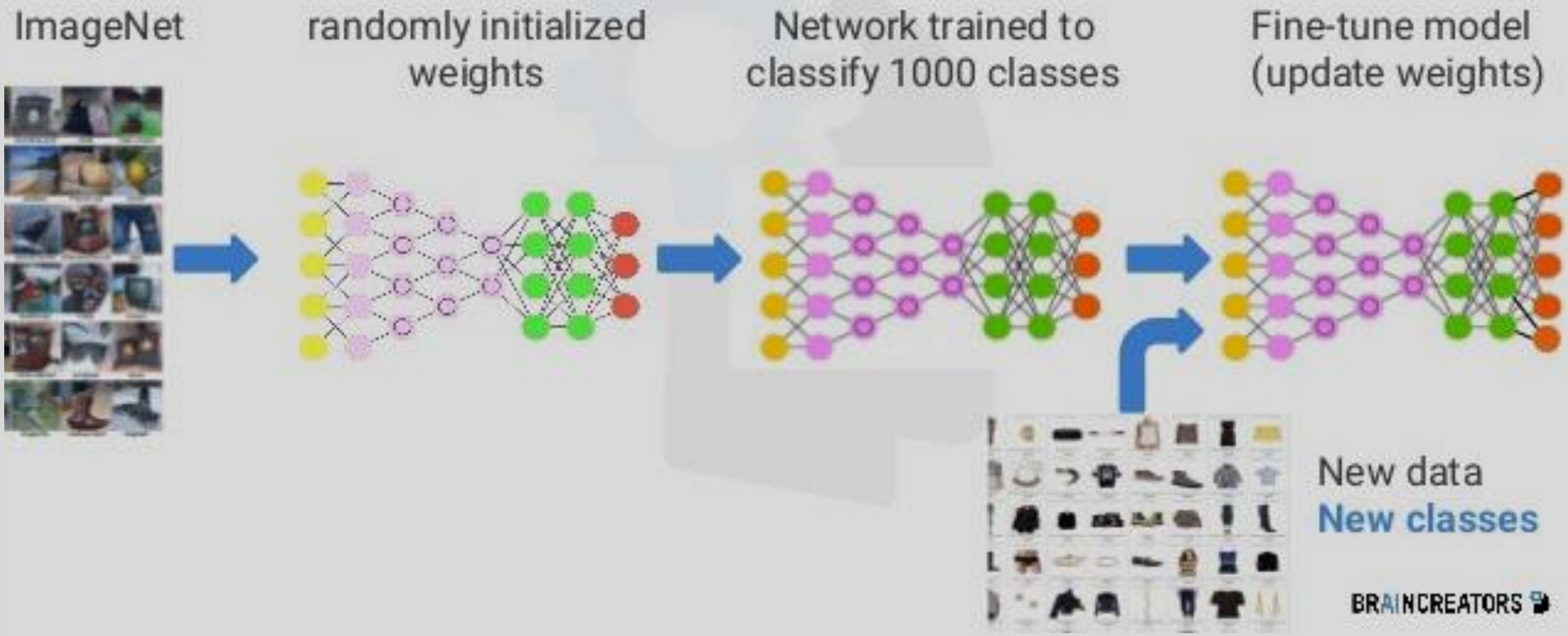
Transfer learning

Transfer learning

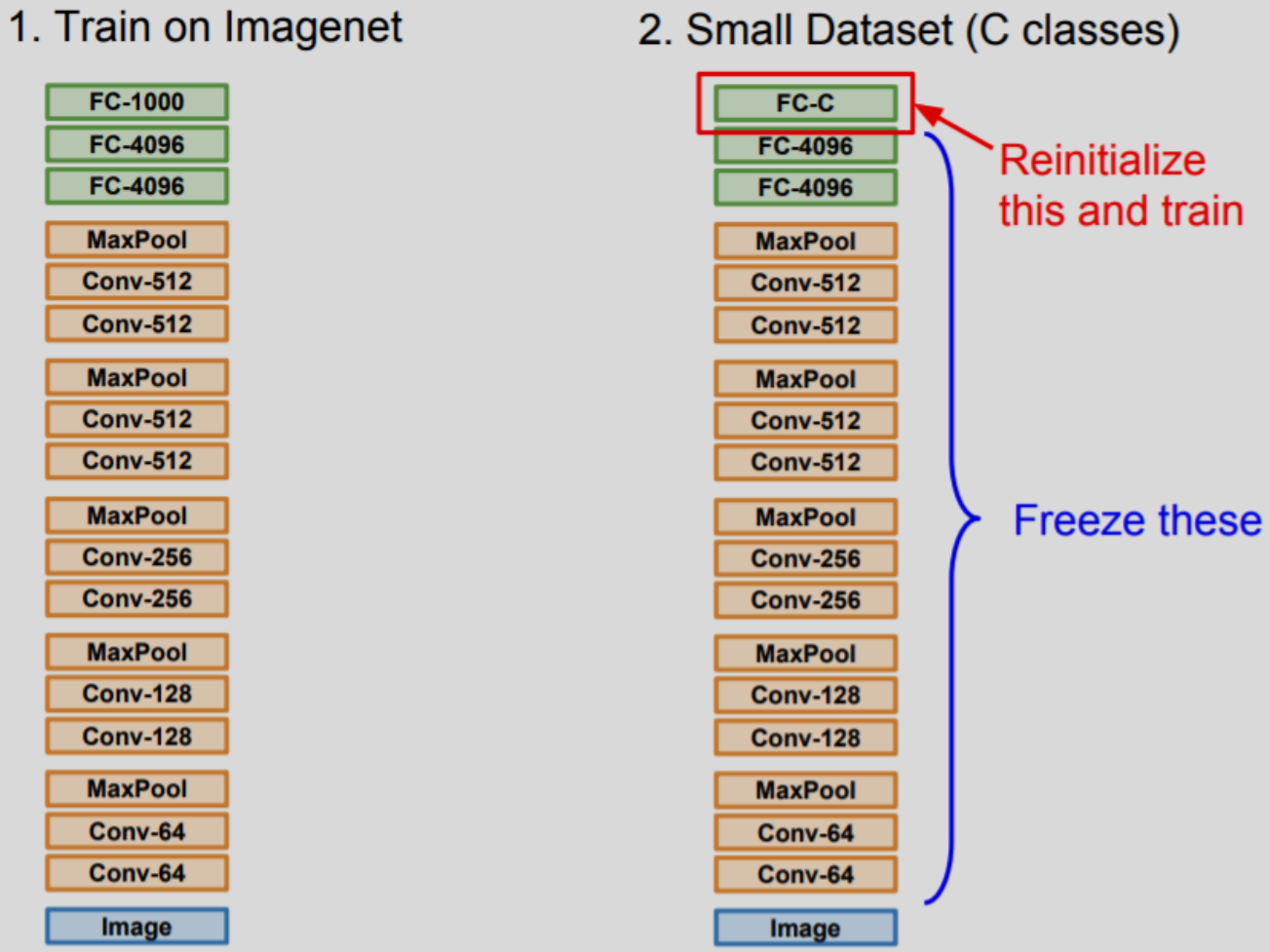
- Storing knowledge gained while solving one problem and applying it to a different but related problem.

- [cf] The goal of knowledge distillation is to provide smaller models that solve the same task as larger models; whereas, the goal of transfer learning is to reduce training epochs of models that solve a task similar to the task solved by some other model.

Transfer learning



Transfer learning



Transfer learning

https://pytorch.org/docs/stable/torchvision/models.html

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

You can construct a model with random weights by calling its constructor:

```
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
shufflenet = models.shufflenet_v2_x1_0()
mobilenet = models.mobilenet_v2()
resnext50_32x4d = models.resnext50_32x4d()
wide_resnet50_2 = models.wide_resnet50_2()
mnasnet = models.mnasnet1_0()
```

Transfer learning

```
from torchvision import models

net = models.alexnet(pretrained=True)

print(net)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Transfer learning

```
from torchvision import models

net = models.alexnet(pretrained=True)

for p in net.parameters():
  p.requires_grad = False

net.classifier[6] = torch.nn.Linear(in_features=4096, out_features=10,
 bias=True)

print(net)
```

```
AlexNet(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
    (1): ReLU(inplace=True)
    (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
    (4): ReLU(inplace=True)
    (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): ReLU(inplace=True)
    (8): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): ReLU(inplace=True)
    (10): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=9216, out_features=4096, bias=True)
    (2): ReLU(inplace=True)
    (3): Dropout(p=0.5, inplace=False)
    (4): Linear(in_features=4096, out_features=4096, bias=True)
    (5): ReLU(inplace=True)
    (6): Linear(in_features=4096, out_features=10, bias=True)
  )
)
```

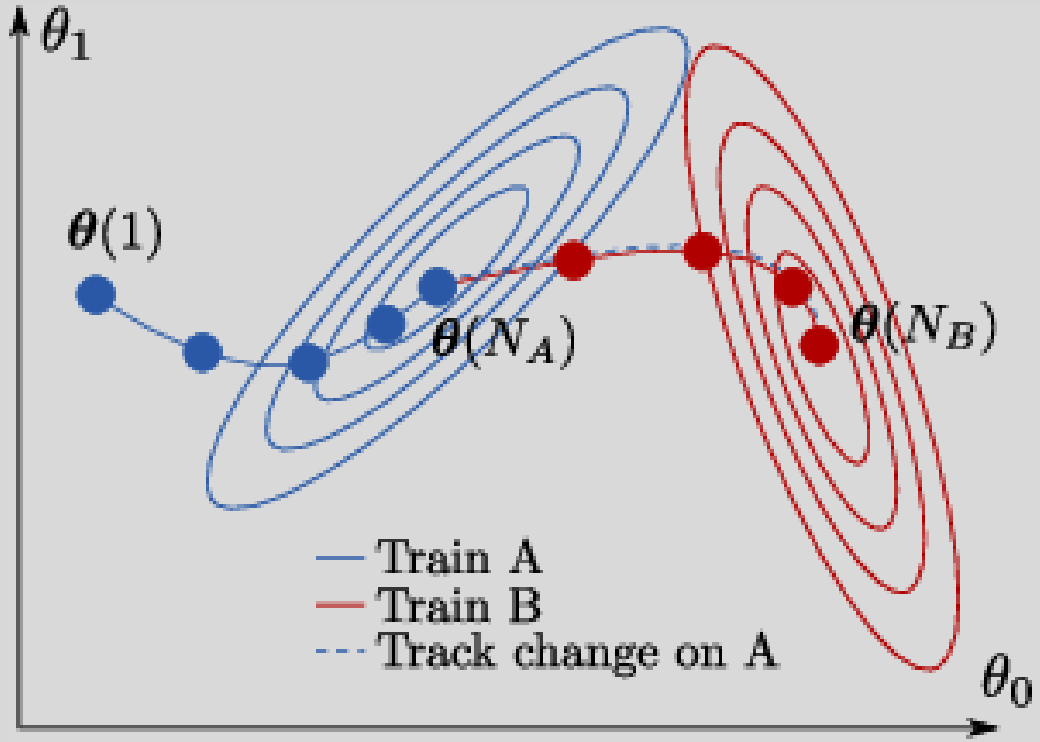
Continual learning

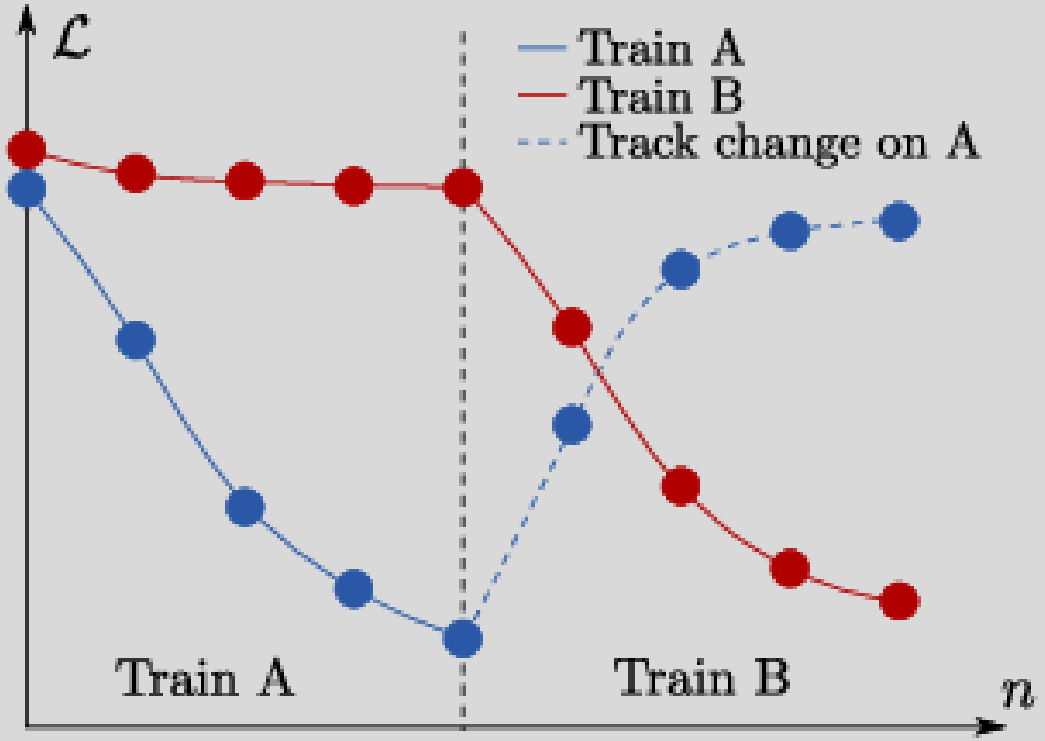
Continual learning

- When trained on one task, then trained on a second task, many machine learning models "*forget*" how to perform the first task.

- A.K.A Lifelong learning, incremental learning

Catastrophic forgetting





Performance for task A drops, as training proceed with task B.

Effectiveness of Dropout

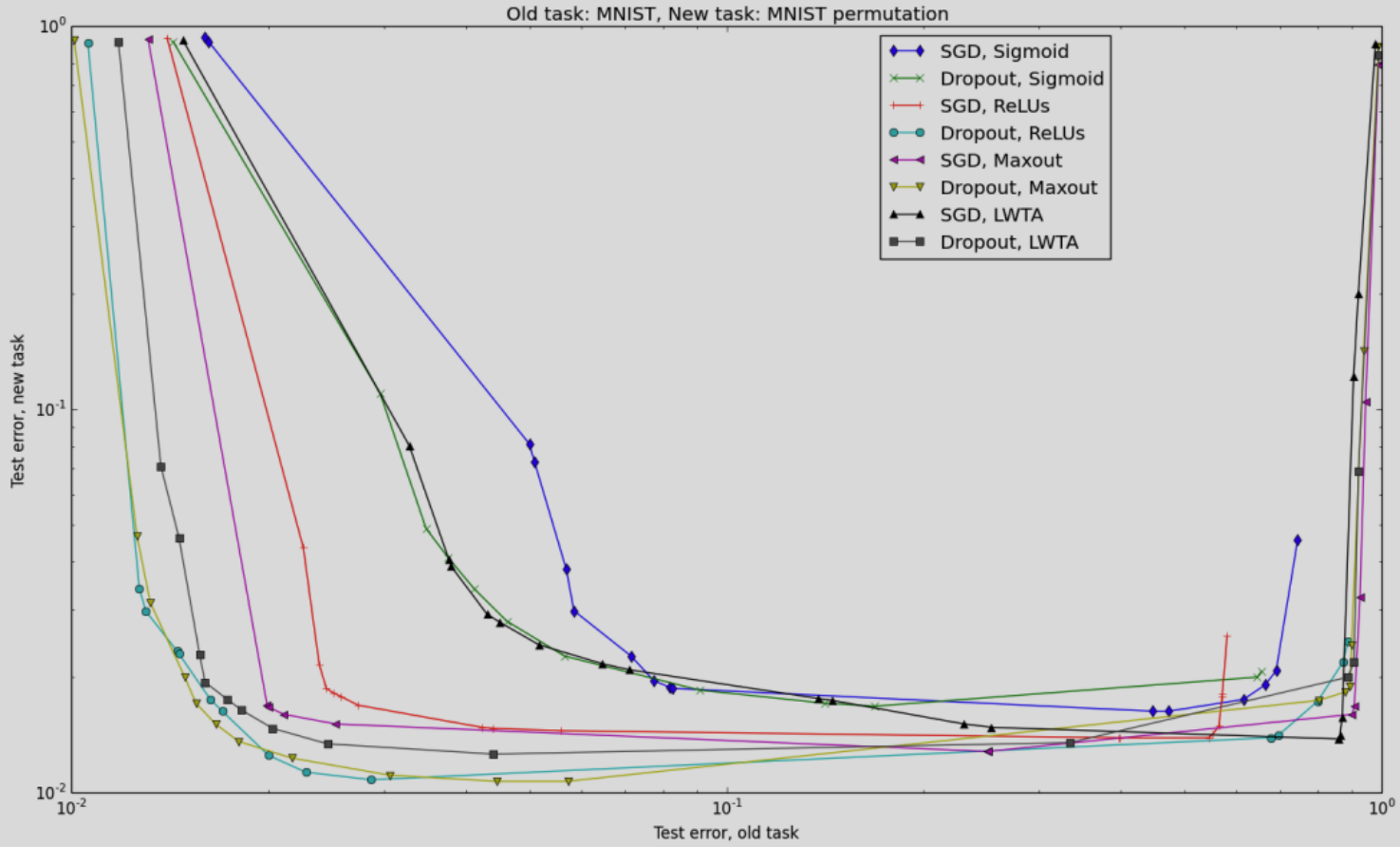
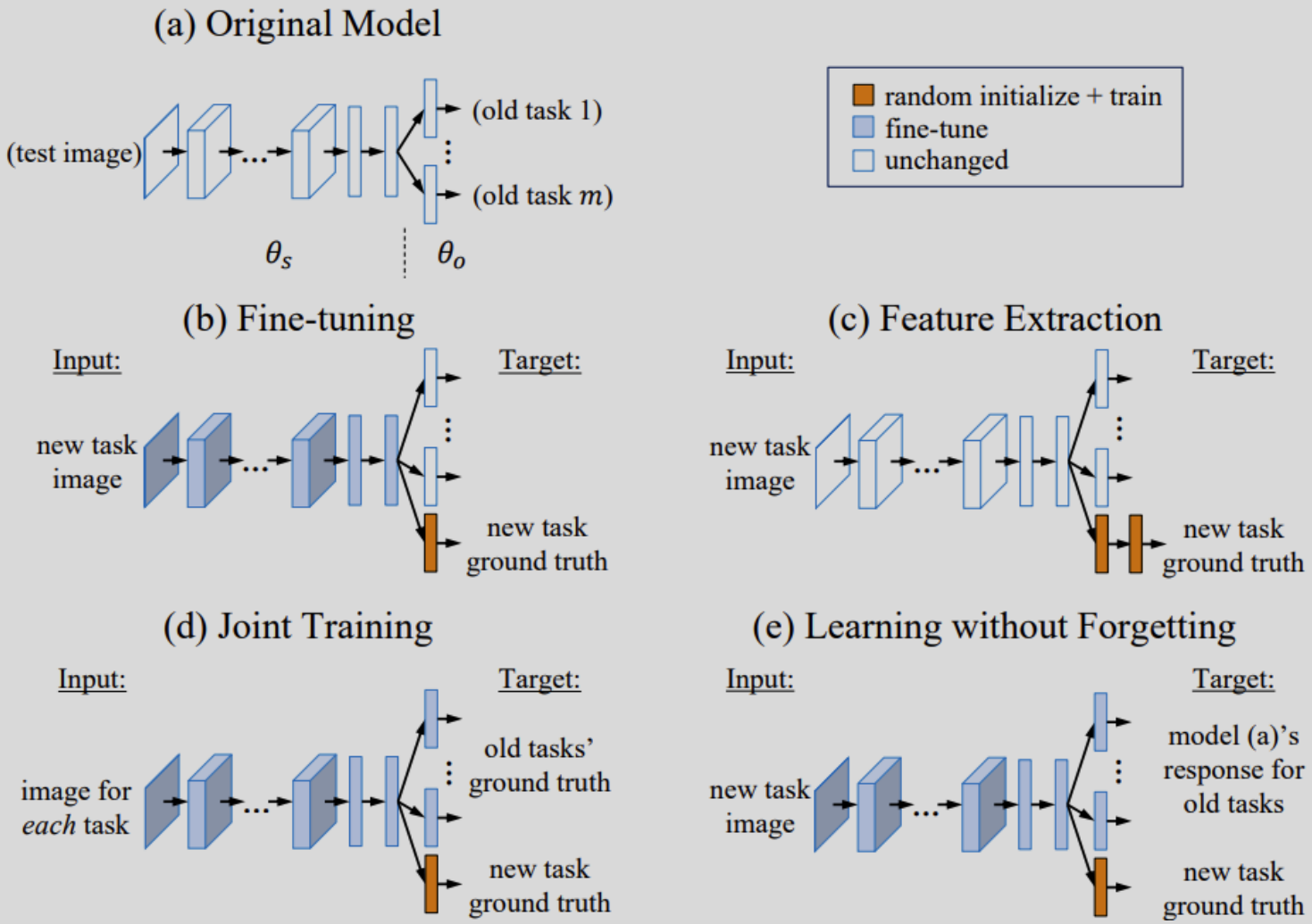


Figure 1. Possibilities frontiers for the input reformatting experiment.

An Empirical Investigation of Catastrophic Forgetting in Gradient-Based Neural Networks, ArXiv'13.

Distillation-based (LwF)



Learning without Forgetting,TPAMI'17

Distillation-based (LwF)

	Fine Tuning	Duplicating and Fine Tuning	Feature Extraction	Joint Training	Learning without Forgetting
new task performance	good	good	X medium	best	✓best
original task performance	X bad	good	good	good	✓good
training efficiency	fast	fast	fast	X slow	✓fast
testing efficiency	fast	X slow	fast	fast	✓fast
storage requirement	medium	X large	medium	X large	✓medium
requires previous task data	no	no	no	X yes	✓no

Joint-training is best for accuracy; while not best for efficiency.

Learning without Forgetting,TPAMI'17

Distillation-based (LwF)

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

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

$$y_o'^{(i)} = \frac{(y_o^{(i)})^{1/T}}{\sum_j (y_o^{(j)})^{1/T}}, \quad \hat{y}_o'^{(i)} = \frac{(\hat{y}_o^{(i)})^{1/T}}{\sum_j (\hat{y}_o^{(j)})^{1/T}}$$

Learning without Forgetting,TPAMI'17

Distillation-based (LwF)

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*

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

Learning without Forgetting,TPAMI'17

Distillation-based (LwF)

(a) Using AlexNet structure (validation performance for ImageNet/Places365/VOC)

	ImageNet→VOC		ImageNet→CUB		ImageNet→Scenes		Places365→VOC		Places365→CUB		Places365→Scenes		ImageNet→MNIST	
	old	new	old	new	old	new	old	new	old	new	old	new	old	new
LwF (ours)	56.2	76.1	54.7	57.7	55.9	64.5	50.6	70.2	47.9	34.8	50.9	75.2	49.8	99.3
Fine-tuning	-0.9	-0.3	-3.8	-0.7	-2.0	-0.8	-2.2	0.1	-4.6	1.0	-2.1	-1.7	-2.8	0.0
LFL	0.0	-0.4	-1.9	-2.6	-0.3	-0.9	0.2	-0.7	0.7	-1.7	-0.2	-0.5	-2.9	-0.6
Fine-tune FC	0.5	-0.7	0.2	-3.9	0.6	-2.1	0.5	-1.3	1.8	-4.9	0.3	-1.1	7.0	-0.2
Feat. Extraction	0.8	-0.5	2.3	-5.2	1.2	-3.3	1.1	-1.4	3.8	-12.3	0.8	-1.7	7.3	-0.8
Joint Training	0.7	-0.2	0.6	-1.1	0.5	-0.6	0.7	-0.0	2.3	1.5	0.3	-0.3	7.2	-0.0

Learning without Forgetting,TPAMI'17

Distillation-based (LwF)

(b) Test set performance

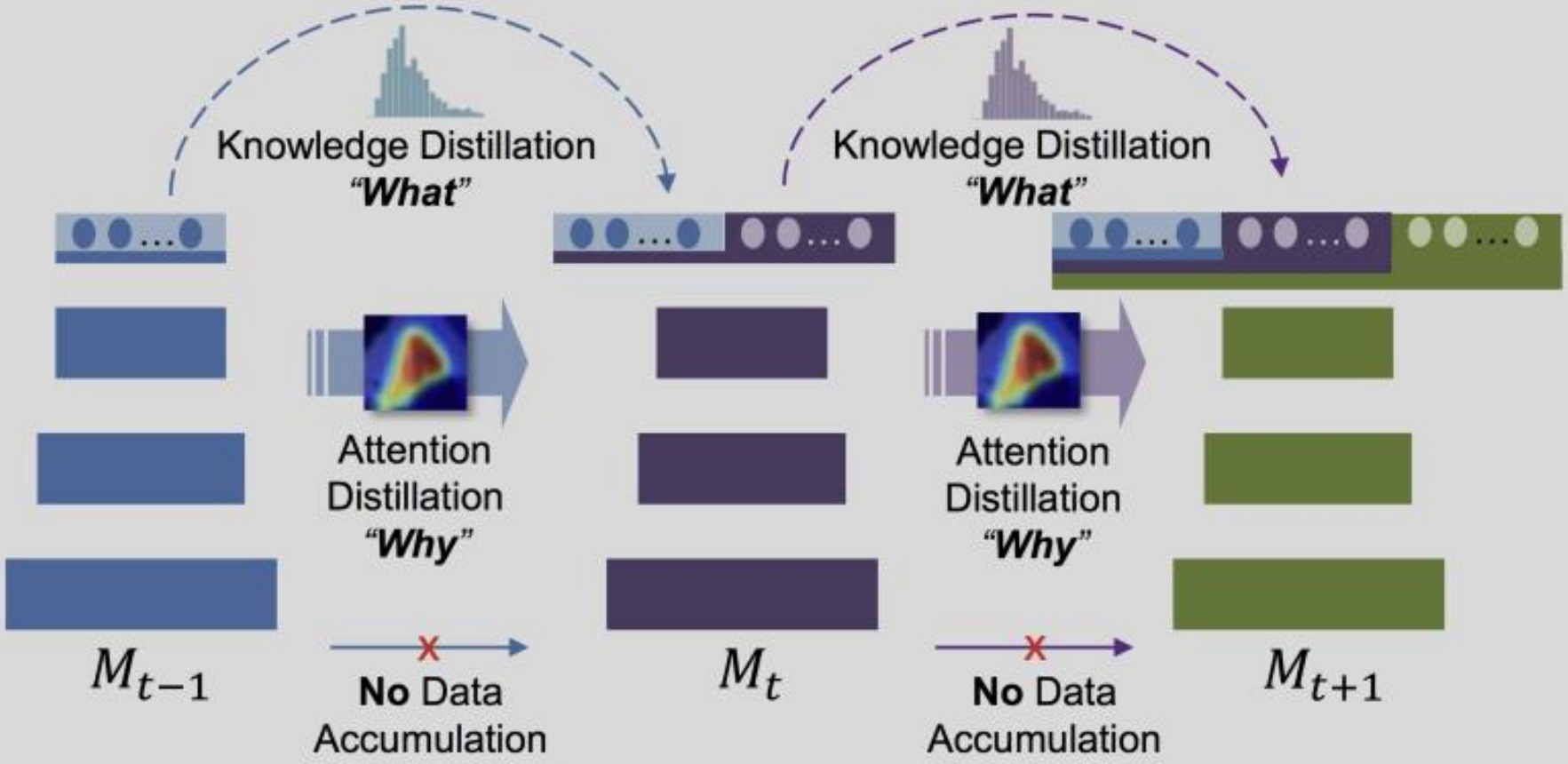
	Places365→VOC	
	old	new
LwF (ours)	50.6	73.7
Fine-tuning	-2.1	0.1
Feat. Extraction	1.3	-2.3
Joint Training	0.9	-0.1

(c) Using VGGnet structure

	ImageNet→CUB		ImageNet→Scenes	
	old	new	old	new
LwF (ours)	60.6	72.5	66.8	74.9
Fine-tuning	-9.9	0.6	-4.1	-0.3
LFL	0.3	-2.8	-0.0	-2.1
Fine-tune FC	3.2	-6.7	1.4	-2.4
Feat. Extraction	8.2	-8.6	1.9	-5.1
Joint Training	8.0	2.5	4.1	1.5

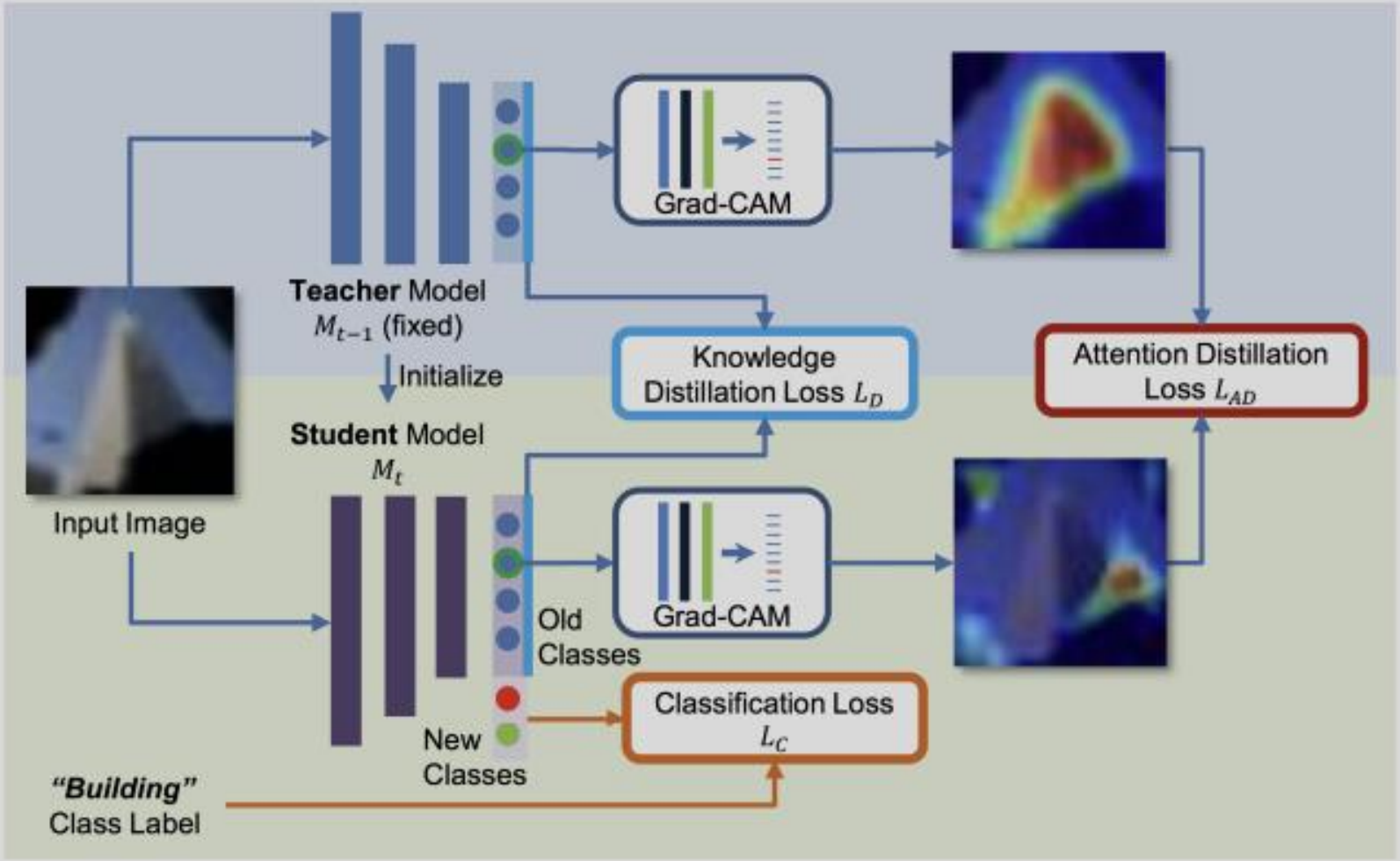
Learning without Forgetting,TPAMI'17

Distillation-based (LwM)



Learning without Memorizing, CVPR'19

Distillation-based (LwM)



$$L_{LwM} = L_C + \beta L_D + \gamma L_{AD}$$

$$L_{AD} = \sum_{j=1}^{l} \| \frac{Q_{t-1,j}^{I_n,b}}{\|Q_{t-1}^{I_n,b}\|_2} - \frac{Q_{t,j}^{I_n,b}}{\|Q_t^{I_n,b}\|_2} \|_1$$

$$Q_{t-1}^{i,c} = vector(\text{Grad-CAM}(i, M_{t-1}, c))$$

$$Q_t^{i,c} = vector(\text{Grad-CAM}(i, M_t, c))$$

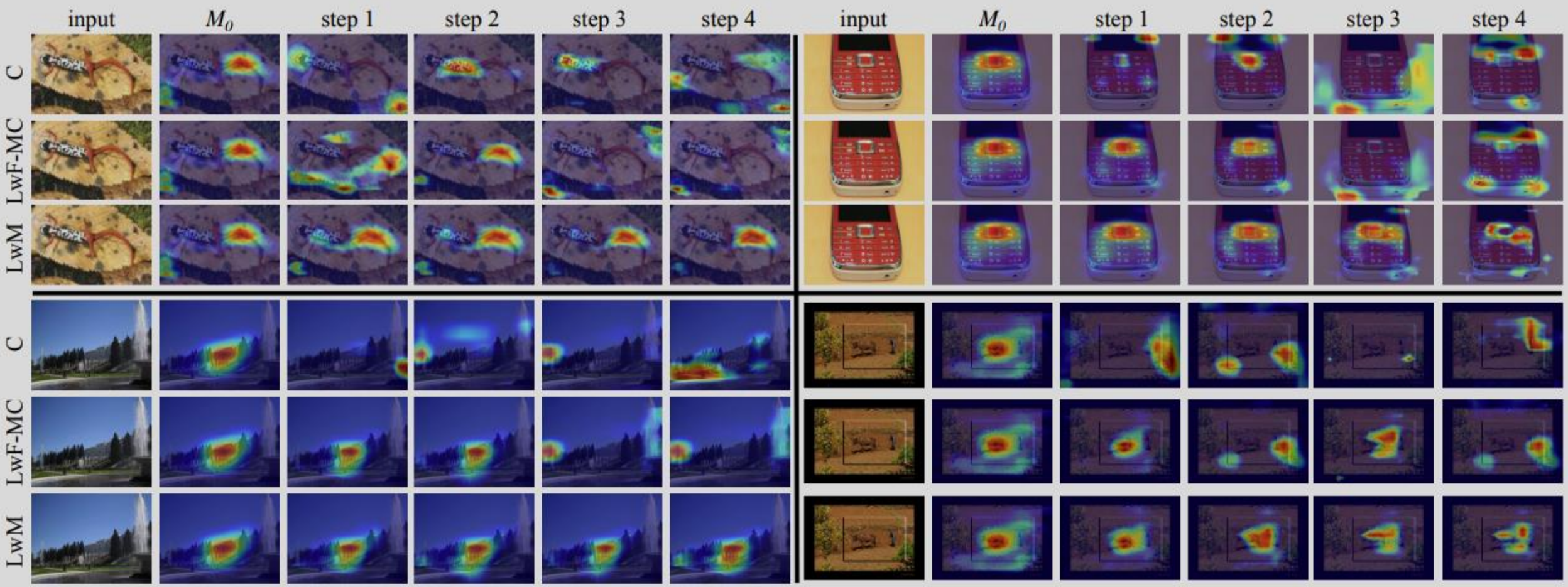
LwM = LwF + Attention Distillation Loss (L_AD)

Learning without Memorizing, CVPR'19

Distillation-based (LwM)

Learning without Memorizing, CVPR'19

Distillation-based (LwM)



Regularization (EWC)

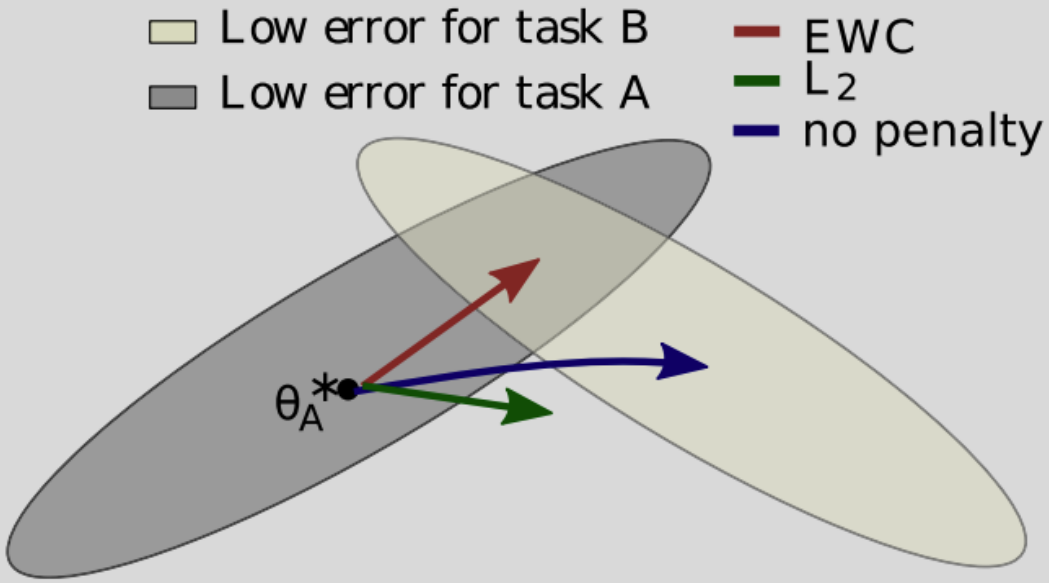


Fig. 1. EWC ensures task *A* is remembered while training on task *B*. Training trajectories are illustrated in a schematic parameter space, with parameter regions leading to good performance on task *A* (gray) and on task *B* (cream color). After learning the first task, the parameters are at θ_A^*. If we take gradient steps according to task *B* alone (blue arrow), we will minimize the loss of task *B* but destroy what we have learned for task *A*. On the other hand, if we constrain each weight with the same coefficient (green arrow), the restriction imposed is too severe and we can remember task *A* only at the expense of not learning task *B*. EWC, conversely, finds a solution for task *B* without incurring a significant loss on task *A* (red arrow) by explicitly computing how important weights are for task *A*.

Overcoming catastrophic forgetting in neural networks, PNAS'17

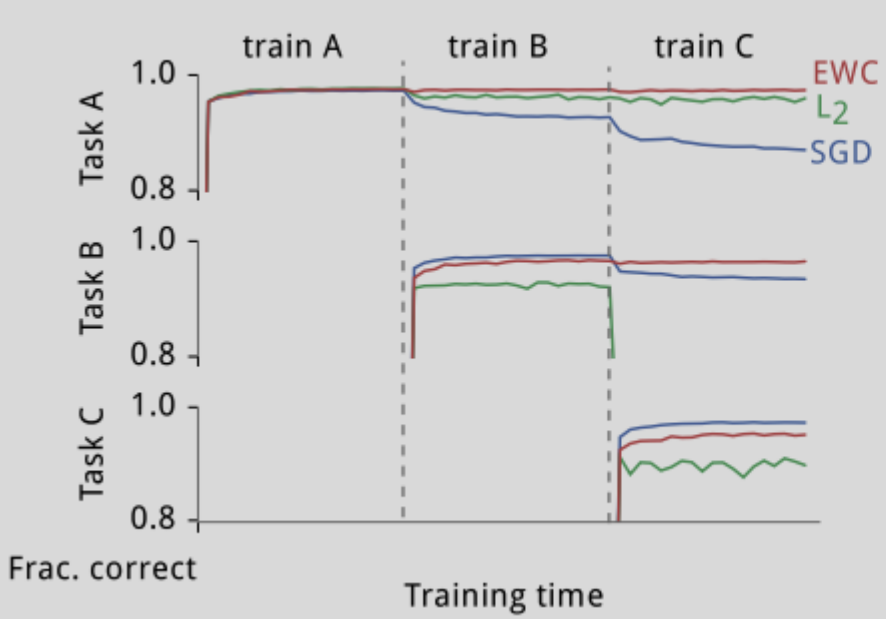
Regularization (EWC)

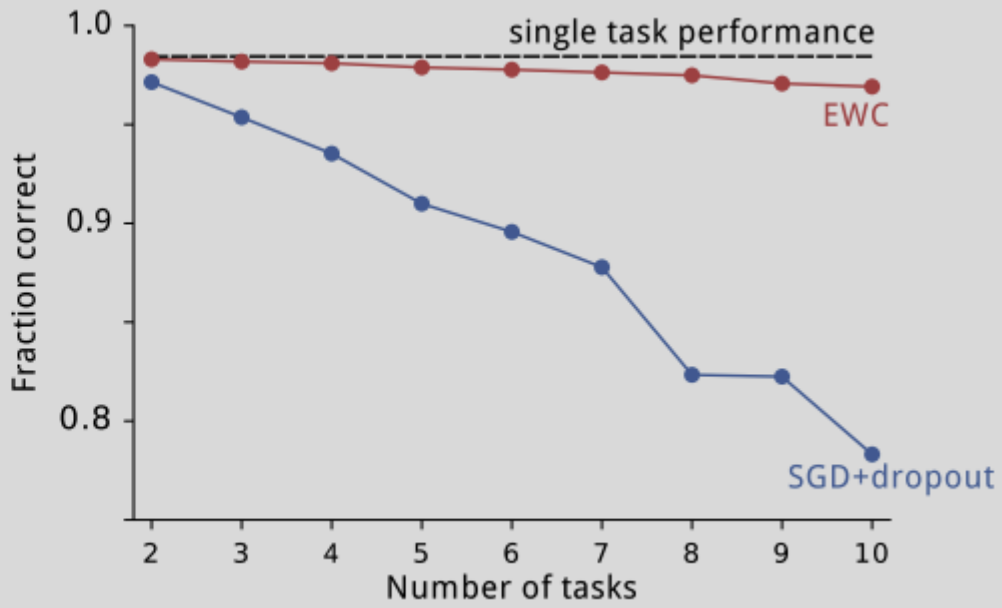
$$\mathcal{L}(\theta) = \mathcal{L}_B(\theta) + \sum_i \frac{\lambda}{2} F_i (\theta_i - \theta^*_{A,i})^2$$

$\mathcal{L}_B(\theta)$ is the loss for task B only

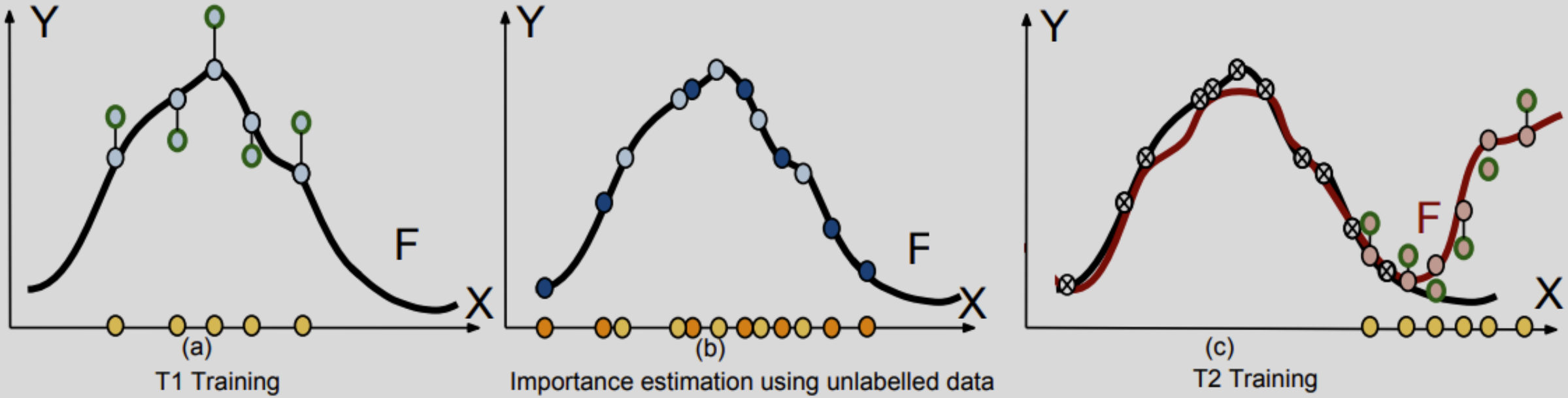
diagonal of the Fisher information matrix F

Regularization (EWC)





Regularization (MAS)



Changes to important parameters are penalized.

Memory Aware Synapses: Learning what (not) to forget , ECCV'18

Regularization (MAS)

$$F(x_k;\theta+\delta)-F(x_k;\theta)\approx\sum_{i,j}g_{ij}(x_k)\delta_{ij}$$

$$g_{ij}(x_k)=\frac{\partial(F(x_k;\theta))}{\partial\theta_{ij}}$$

$$\Omega_{ij}=\frac{1}{N}\sum_{k=1}^{N}\|\,g_{ij}(x_k)\,\|$$

Outputs would be much affected according to certain parameters, if they are important.

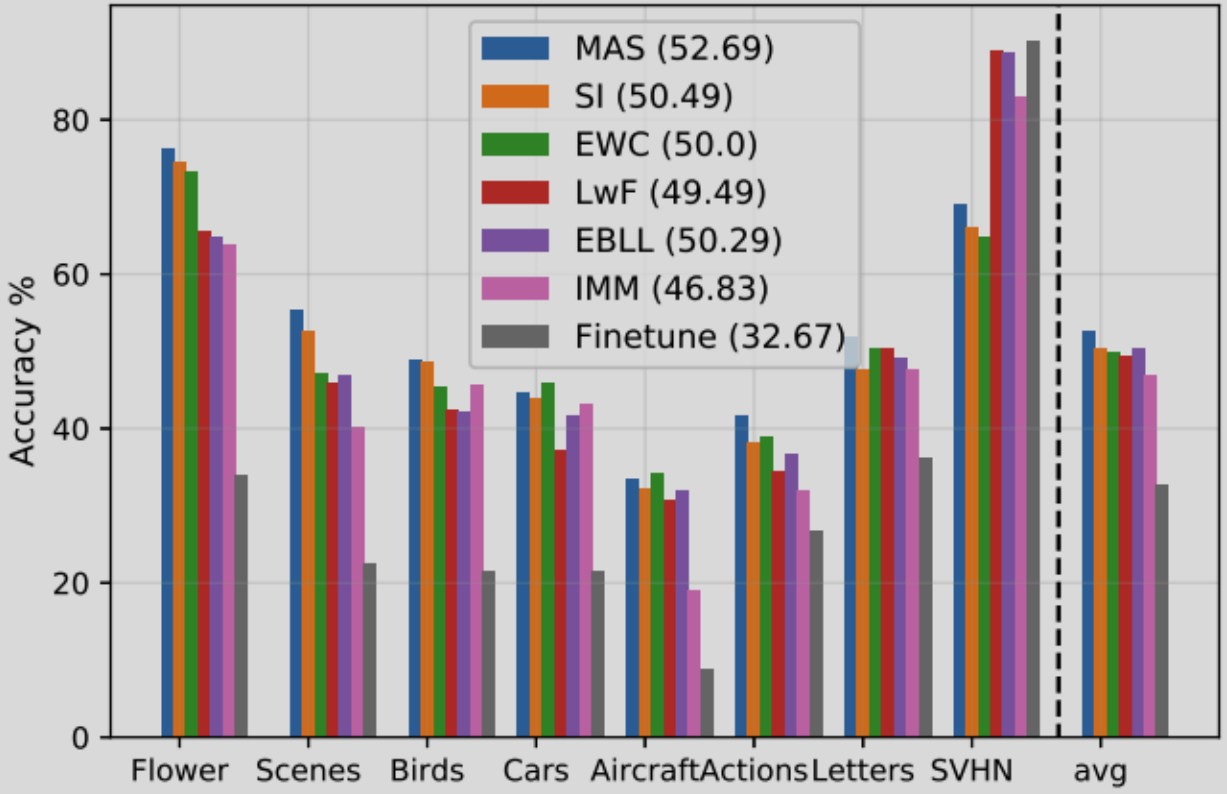
Regularization (MAS)

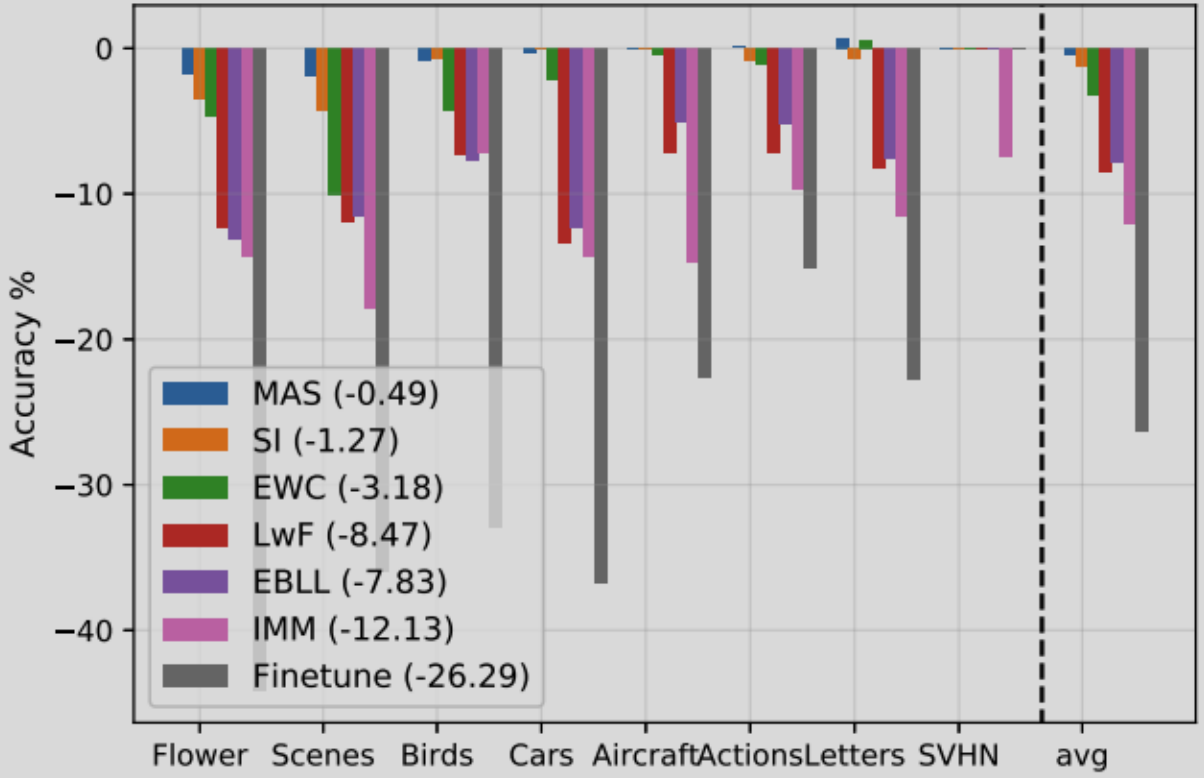
$$L(\theta) = L_n(\theta) + \lambda \sum_{i,j} \Omega_{ij} (\theta_{ij} - \theta^*_{ij})^2$$

θ^*_{ij} the "old" network parameters

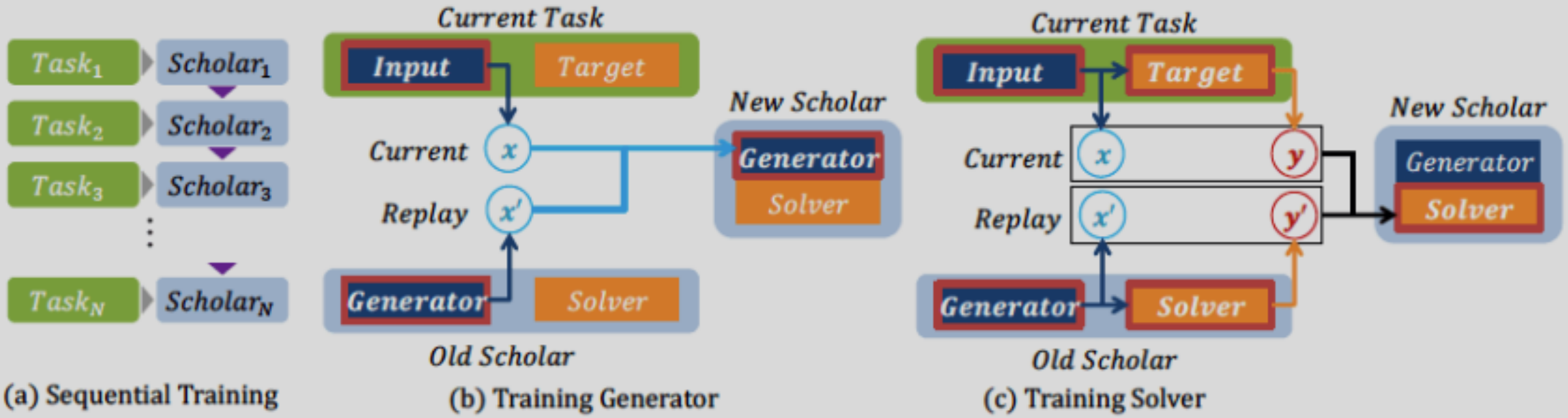
$L_n(\theta)$ new task loss

Regularization (MAS)



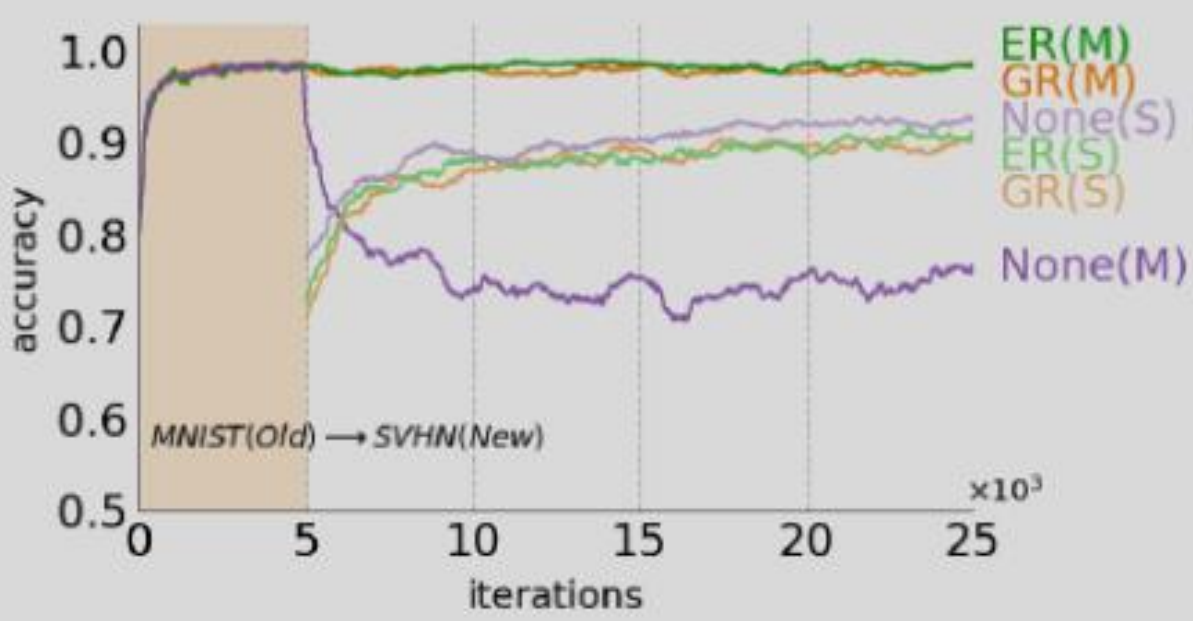


GAN-based

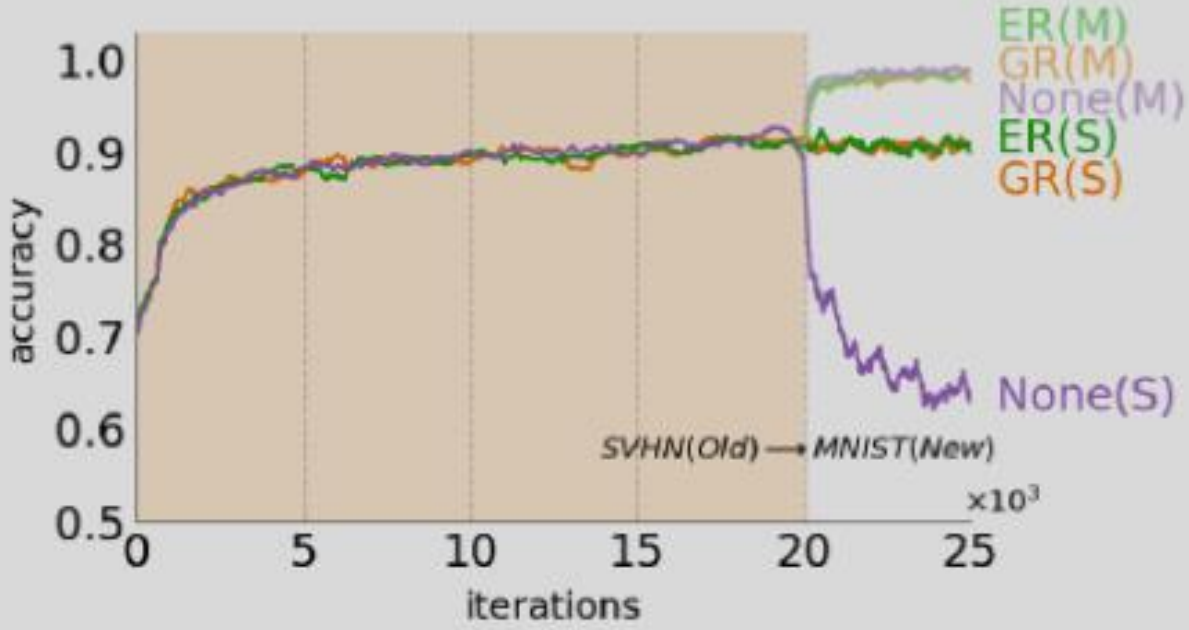


Continual learning with deep generative replay, NIPS'17

GAN-based

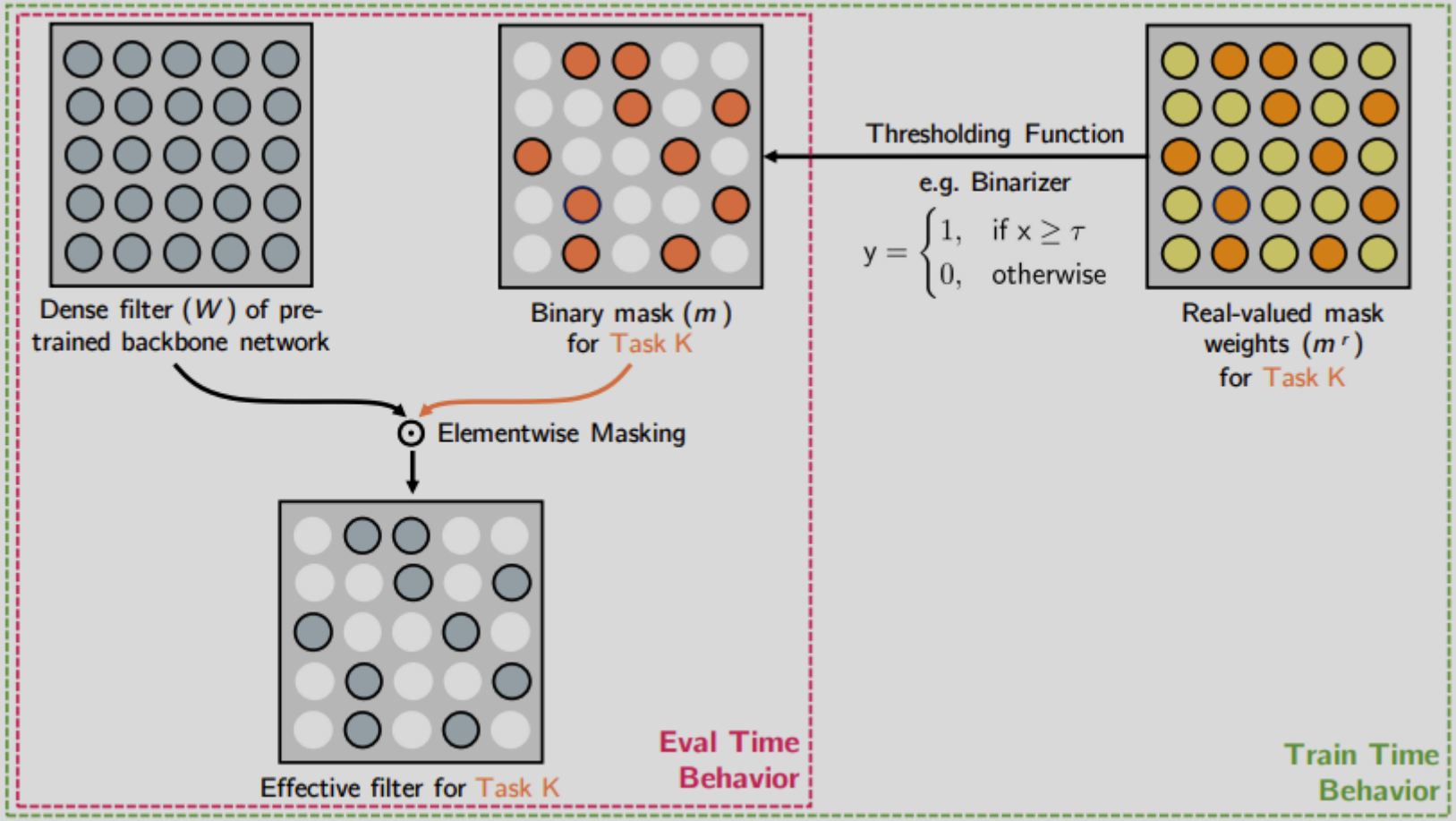


(a) MNIST to SVHN



(b) SVHN to MNIST

Structural change



Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights, ECCV'18

Structural change

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

$$m_{ji} = \begin{cases} 1, & \text{if } m^r{}_{ji} \geq \tau \\ 0, & \text{otherwise} \end{cases}$$

$$\mathbf{y} = (\mathbf{W} \odot \mathbf{m})\,\mathbf{x}$$

$$\delta w_{ji} \triangleq \frac{\partial E}{\partial w_{ji}} = \left(\frac{\partial E}{\partial y_j}\right) \cdot \left(\frac{\partial y_j}{\partial w_{ji}}\right)$$

$$\delta m_{ji} \triangleq \frac{\partial E}{\partial m_{ji}} = \left(\frac{\partial E}{\partial y_j}\right) \cdot \left(\frac{\partial y_j}{\partial m_{ji}}\right)$$

Structural change

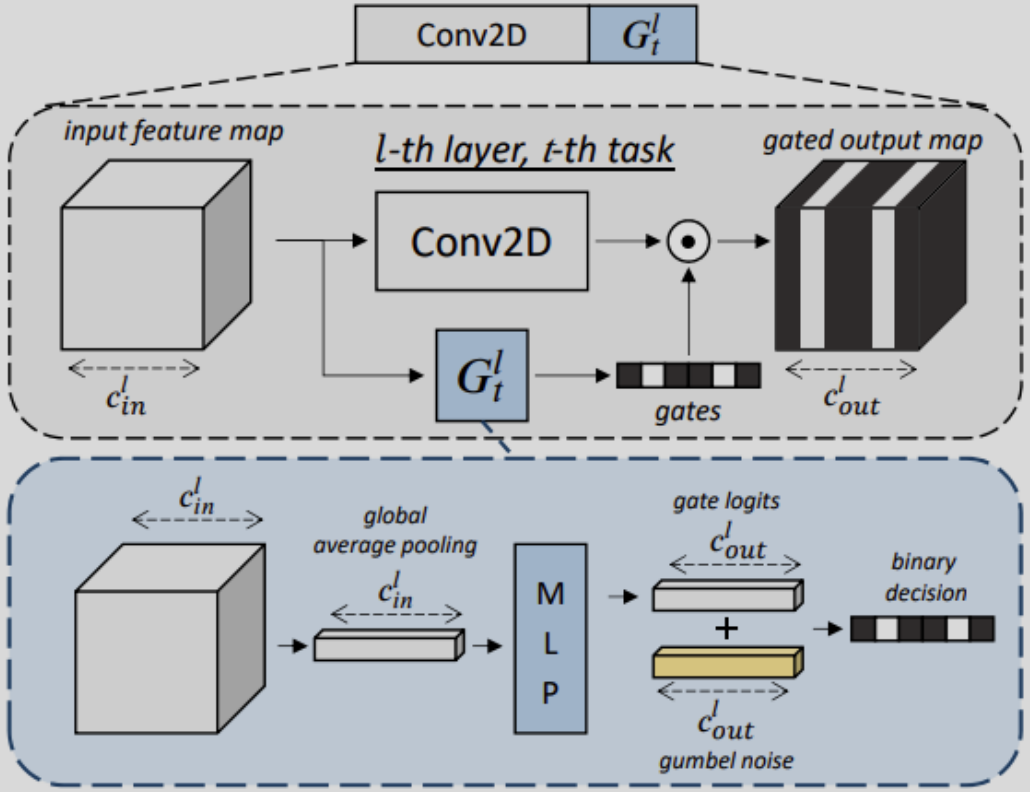


Figure 1: The proposed gating scheme for a convolution layer. Depending on the input feature map, the gating module G_t^l decides which kernels should be used.

$$\hat{\mathbf{h}}^{l+1} = G_t^l(\mathbf{h}^l) \odot \mathbf{h}^{l+1}$$

$$G_t^l(\mathbf{h}^l) = [g_1^l, \ldots, g_{c_{out}^l}^l], \; g_i^l \in \{0, 1\}$$

Conditional Channel Gated Networks for Task-Aware Continual Learning, CVPR'20

Structural change

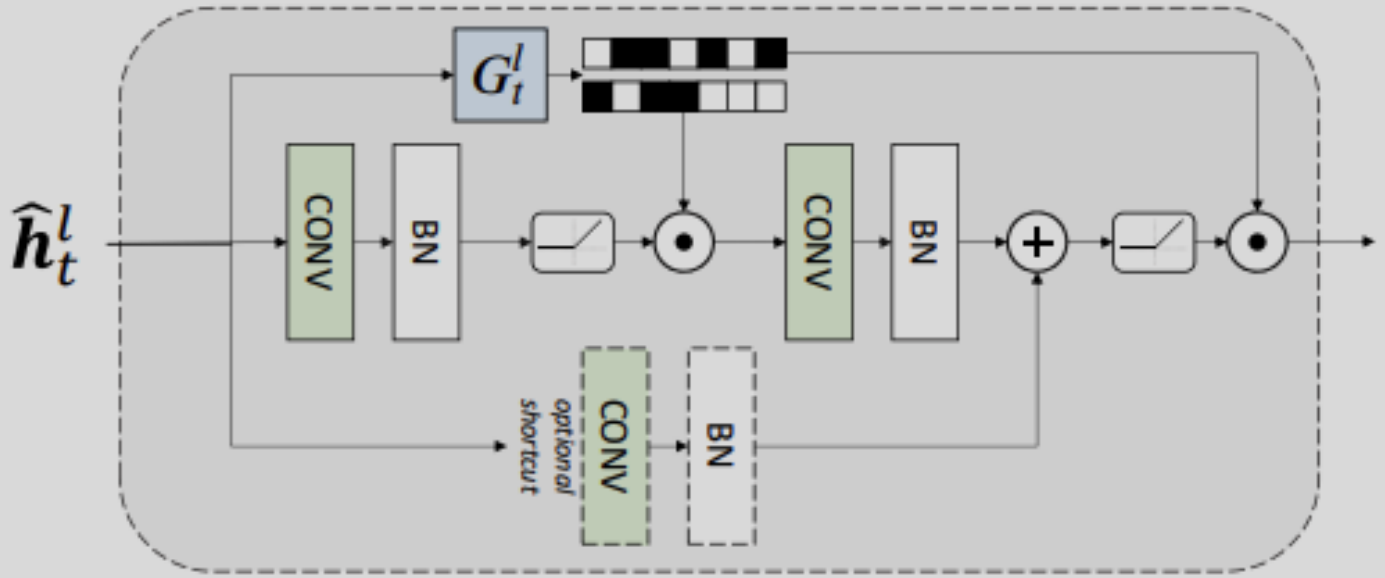


Figure 3: The gating scheme applied to ResNet-18 blocks. Gating on the *shortcut* is only applied when downsampling.

Conditional Channel Gated Networks for Task-Aware Continual Learning, CVPR'20

Structural change

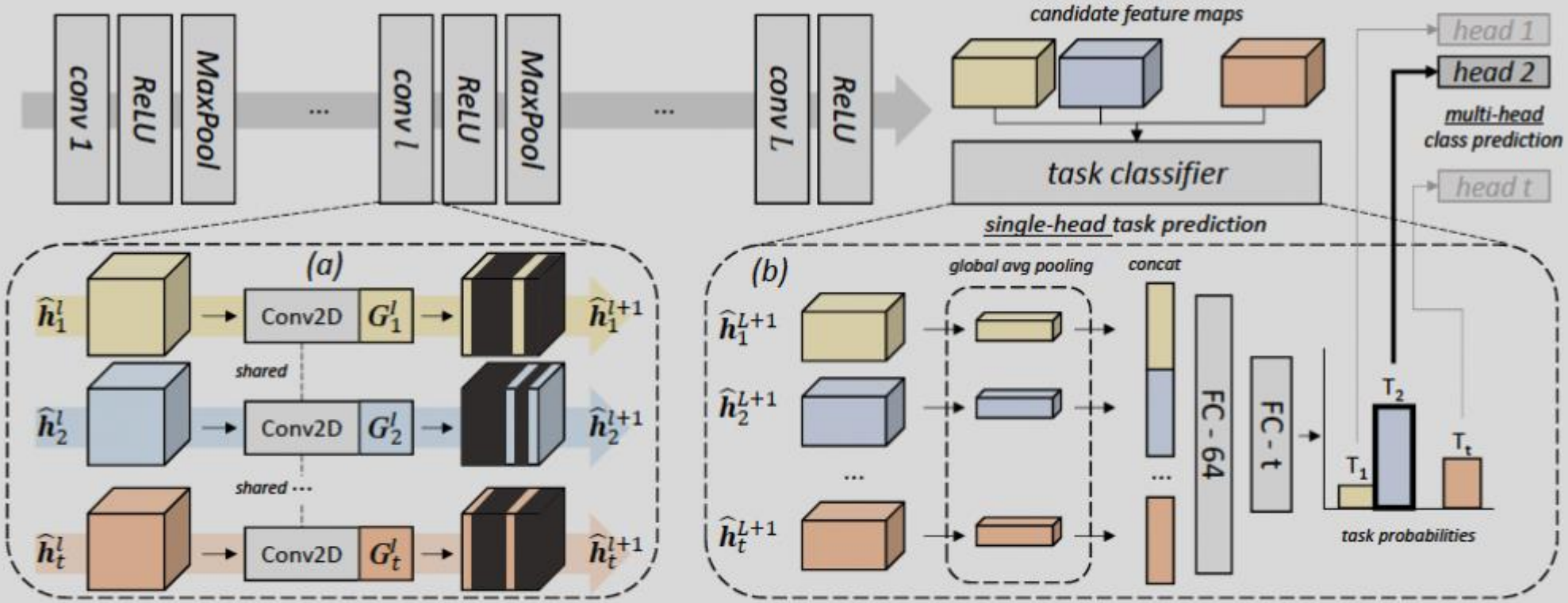


Figure 2: Illustration of the task prediction mechanism for a generic backbone architecture. First (block 'a'), the l-th convolutional layer is fed with multiple gated feature maps, each of which is relevant for a specific task. Every feature map is then convolved with kernels selected by the corresponding gating module G_x^l, and forwarded to the next module. At the end of the network the task classifier (block 'b') takes as input candidate feature maps and decides which task to solve.

Thank you!