Domain Adversarial Training

신경망응용및실습 두번째 프로젝트

Daehyun Cho
Cognitive System Lab
A.I. Dept, Korea Univ.

Index

Domain Generalization

Train classification task of different datasets of styles(=domain).

There are several approaches to this task.

10 categories, 4 domains

2/3 training set, 1/6 validation set, 1/6 test set

[Fig. 1] OfficeHome dataset samples and splits from proposal.

✓ https://www.youtube.com/watch?v=uN8z4pyL2N4

DANN Implementation with PyTorch Adapt

Simple framework to implement several approaches for domain adaptation.

I also used **DANN** from the proposal.

```
G = timm.create_model(self.args.model_name_or_path, pretrained=True).to(self.device)
G.classifier = nn.Identity()

C = Classifier(in_size=self.args.embed_dim, num_classes=10).to(self.device)
D = Discriminator(in_size=self.args.embed_dim, h=256).to(self.device)

self.models = Models({"G": G, "C": C, "D": D})

optimizers = Optimizers((torch.optim.Adam, {"lr": 1e-4}))
optimizers.create_with(self.models)
self.optimizers = list(optimizers.values())

self.hook = DANNHook(self.optimizers)
```

[Fig. 2] PyTorch adapt example.

- ✓ https://github.com/KevinMusgrave/pytorch-adapt
- ✓ https://medium.com/@tkm45/deep-domain-adaptation-using-pytorch-adapt-c020ae596e25

Models

Since there was a hardware limit, I used MobileNetV3 and ResMLP-distilled.

Both models were implemented with the help of timm.

- ✓ Adam
- ✓ LR=1e-4
- ✓ Epochs=20

Tests

From baseline, I got this result.

I have also tested in reverse way.

Classification accuracy on target domain (%)					
	R -> P	P -> C	C -> A	A -> R	AVG.
Baseline	35.10	27.47	27.65	27.10	29.33
Yours					

[Table. 1] Baseline performance from proposal.

DANN Results (Accuracy/AUROC)

Since there was a hardware limit, I used MobileNetV3 and ResMLP

Models	R→P	P→C	C→A	A→R	Average.
Baseline	35.10	27.47	27.65	27.10	**29.33**
MobileNetV3	87.76 98.21	41.24 85.07	66.04 96.15	80.53 97.85	**68.89** **94.32**
ResMLP12 Distilled	52.04 88.01	22.68 79.04	45.28 88.68	70.8 96.27	**47.7** **88.0**

[Table. 1] Baseline comparison with my model.
Depicted values are accuracy and auroc in %.

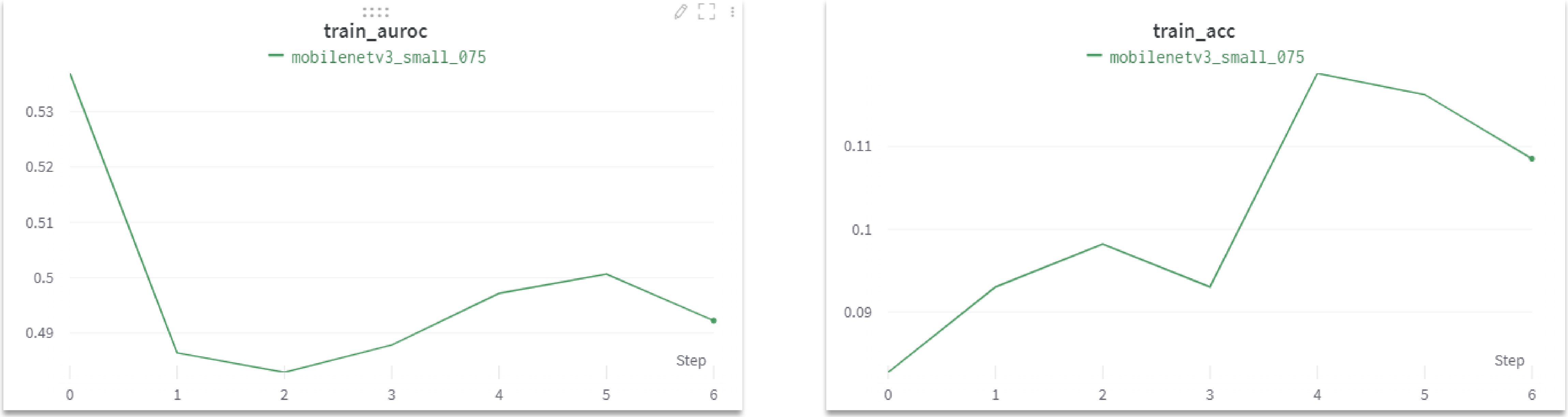
Models	P→R	C→P	A→C	R→A	Average.
MobileNetV3	79.91 95.37	72.98 94.90	58.66 89.45	76.81 96.73	72.05 94.075
ResMLP12 Distilled	9.37 50.0	8.83 53.49	9.30 48.69	60.87 85.18	22.05 59.34

[Table. 2] Reversed training of source and target.

Training: Non-pretrained Models

Since there was small portion of data only, it was inevitable to use pretrained models.

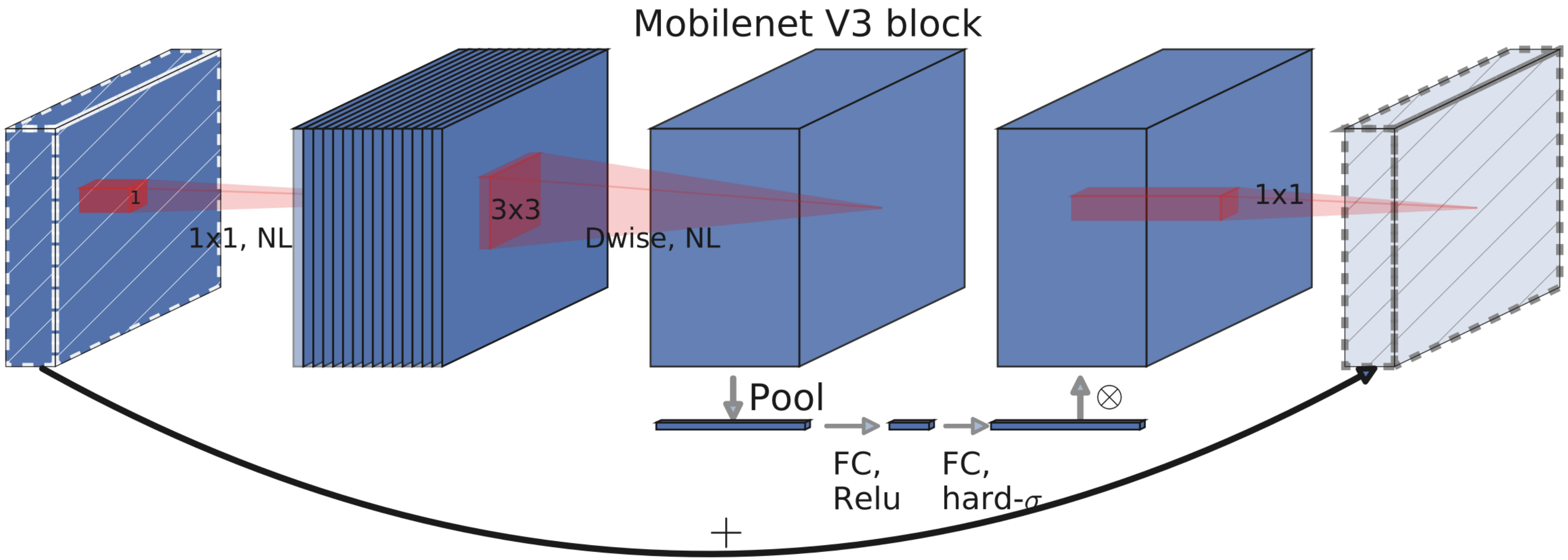
Non-pretrained models fail to optimize in random initialization state.



[Fig. x] Non-pretrained model training curve for source data (Art → Clipart).

Training: MobileNetV3

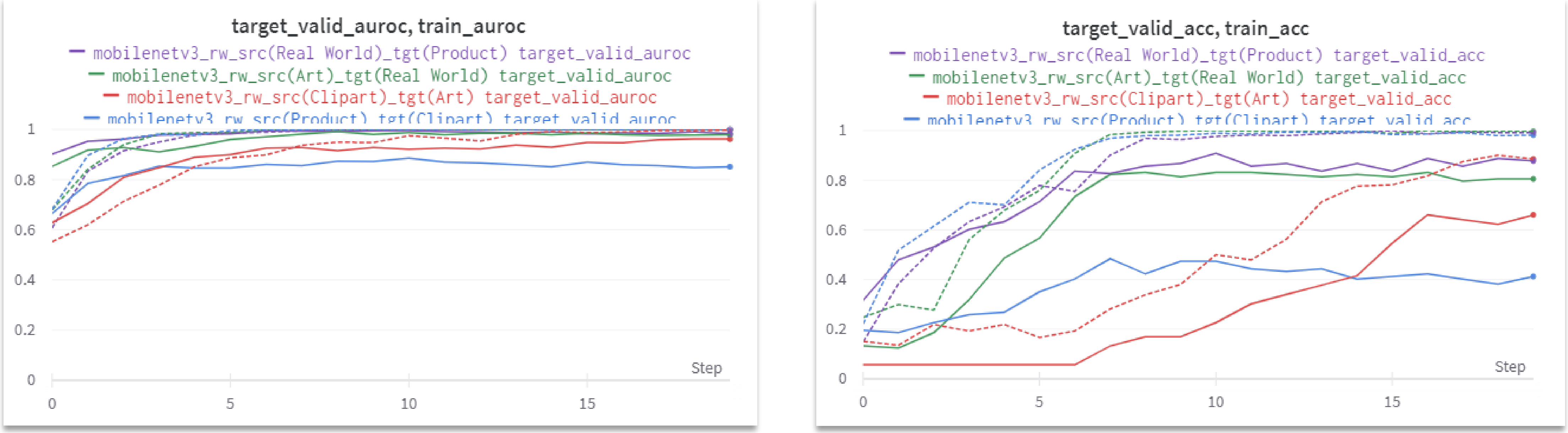
- ✓ MobileNet V3 is NetAdapt Algorithm applied NAS searched architecture.
- ✓ Efficiency of the model was highly increased and reduced latency compared to the preceeding version.



[Fig. x] MobileNet V3 Basic Block Description.

Training: MobileNetV3

✓ Pretrained version optimizes very well and shows high performance.



[Fig. x] AUROC, ACC of source(dashed) and target(plain) line of MobileNet V3 Pretrained.

Training: ResMLP-12-distilled

- ✓ Mixed-MLP was recently proposed multi-layer perceptron architecture for computer vision.
- ✓ Here I used 12 layers model.

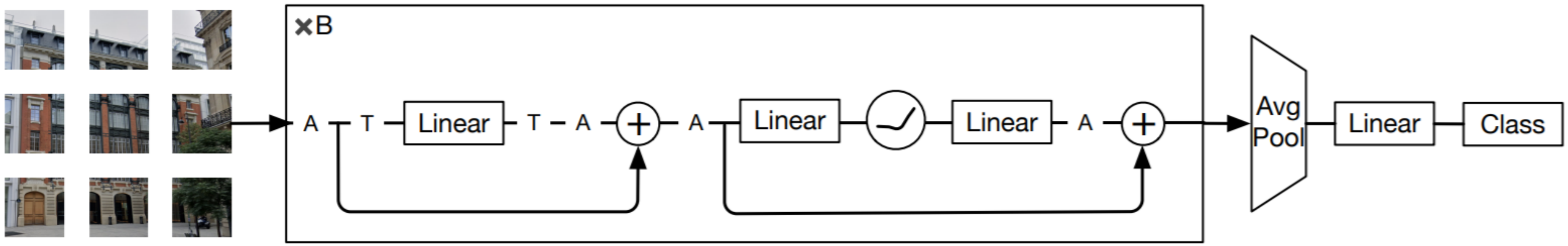


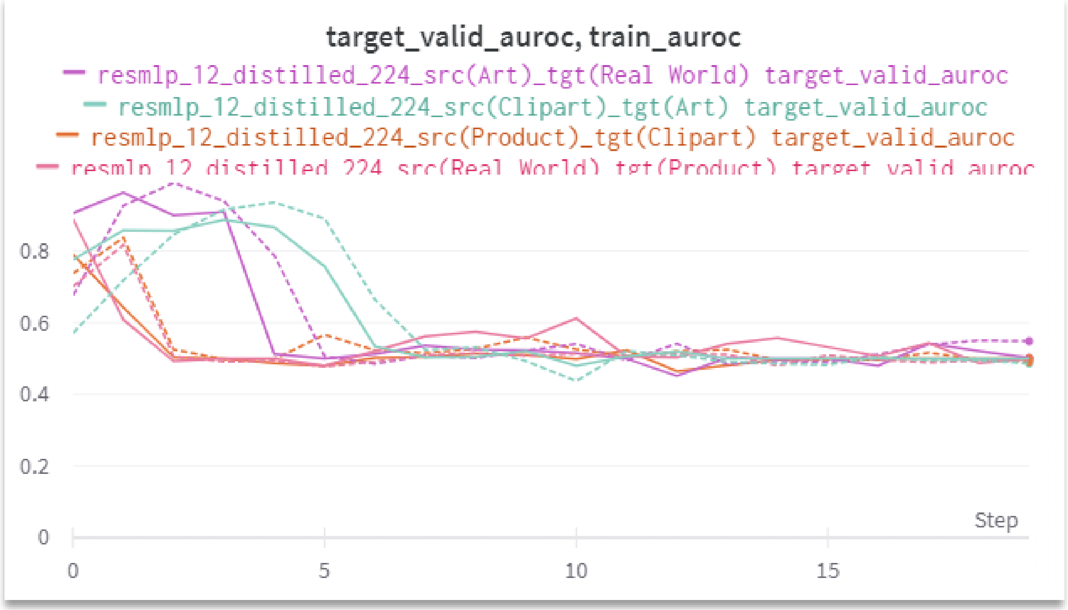
Figure 1: The ResMLP architecture: After flattening the patch into vectors, our network alternately processes them by (1) a communication layer between vectors implemented as a linear layer; (2) a two-layer residual perceptron. We denote by A the operator Aff, and by T the transposition.

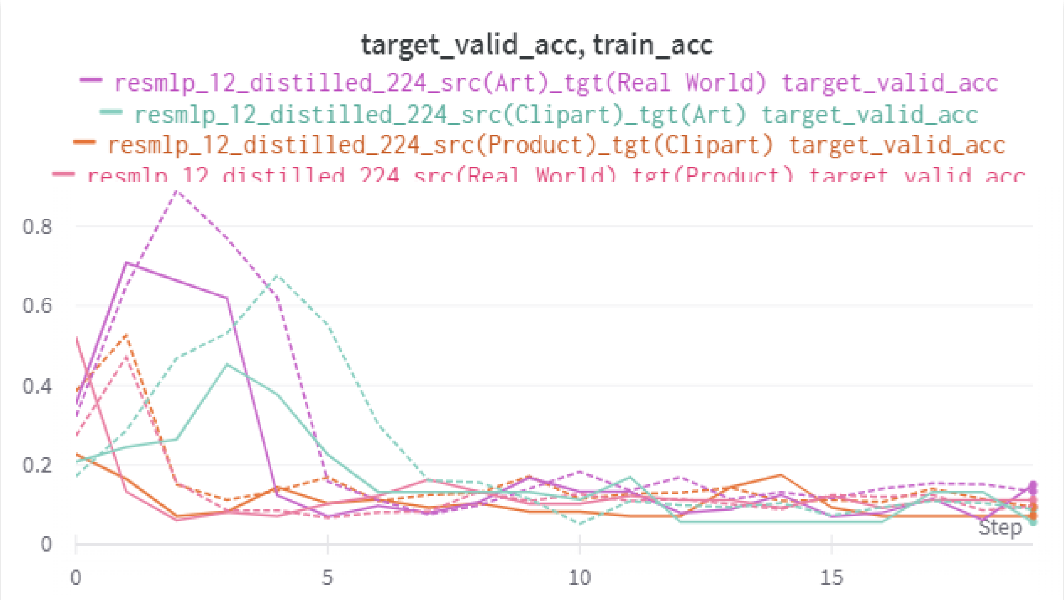
[Fig. x] ResMLP Architecture and description.

- ✓ https://arxiv.org/abs/2105.03404
- ✓ https://github.com/rishikksh20/ResMLP-pytorch

Training: ResMLP-12-Distilled

- ✓ Pretrained model shows relatively high performance in early stages, but fails to optimize.
- ✓ This is due to sensitive configurations for attention networks.

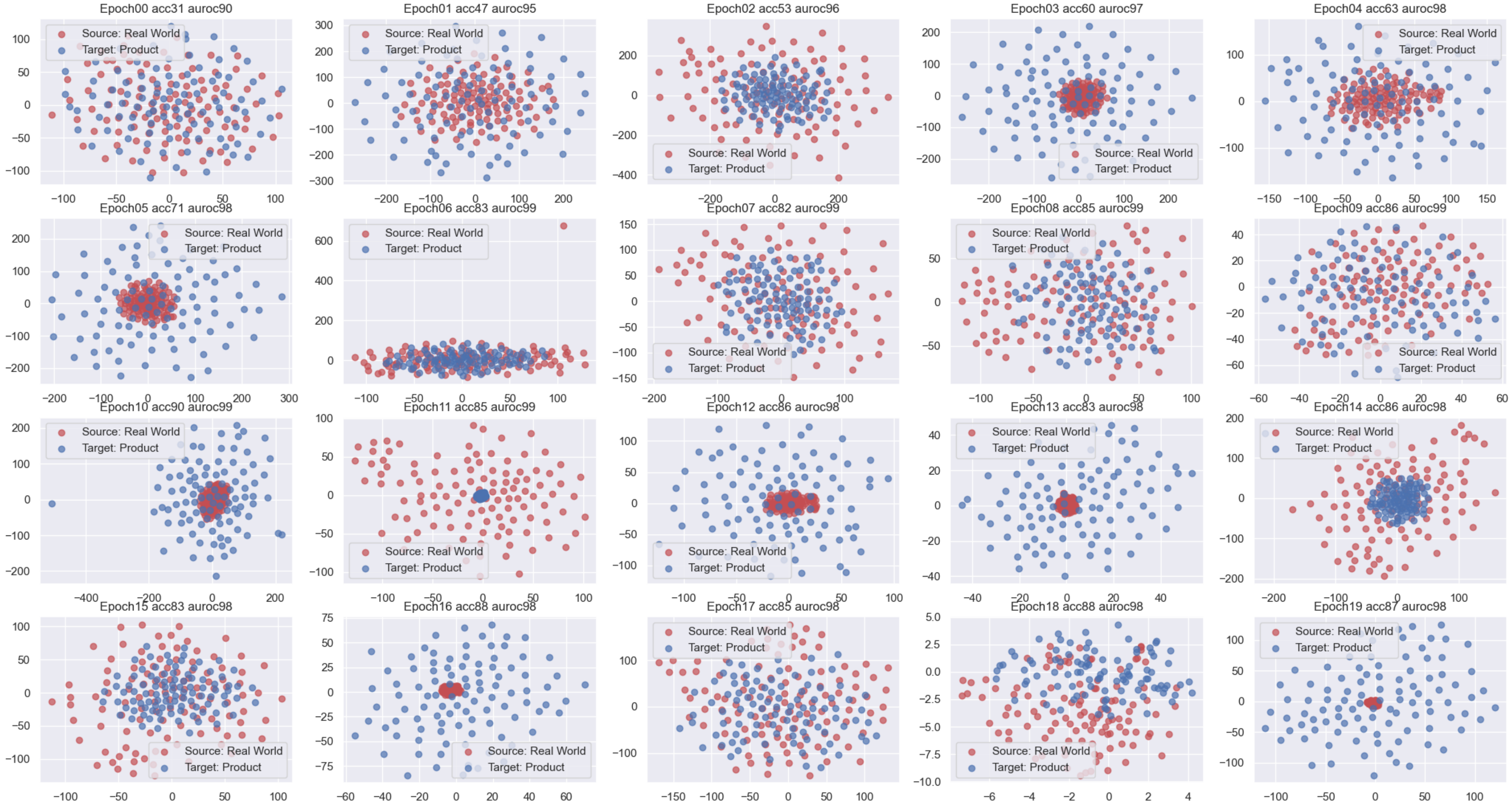




[Fig. x] AUROC, ACC of source(dashed) and target(plain) line of ResMLP-12-distilled

Real World to Product TSNE result on valid dataset. MobileNetV3 (87.76% acc)

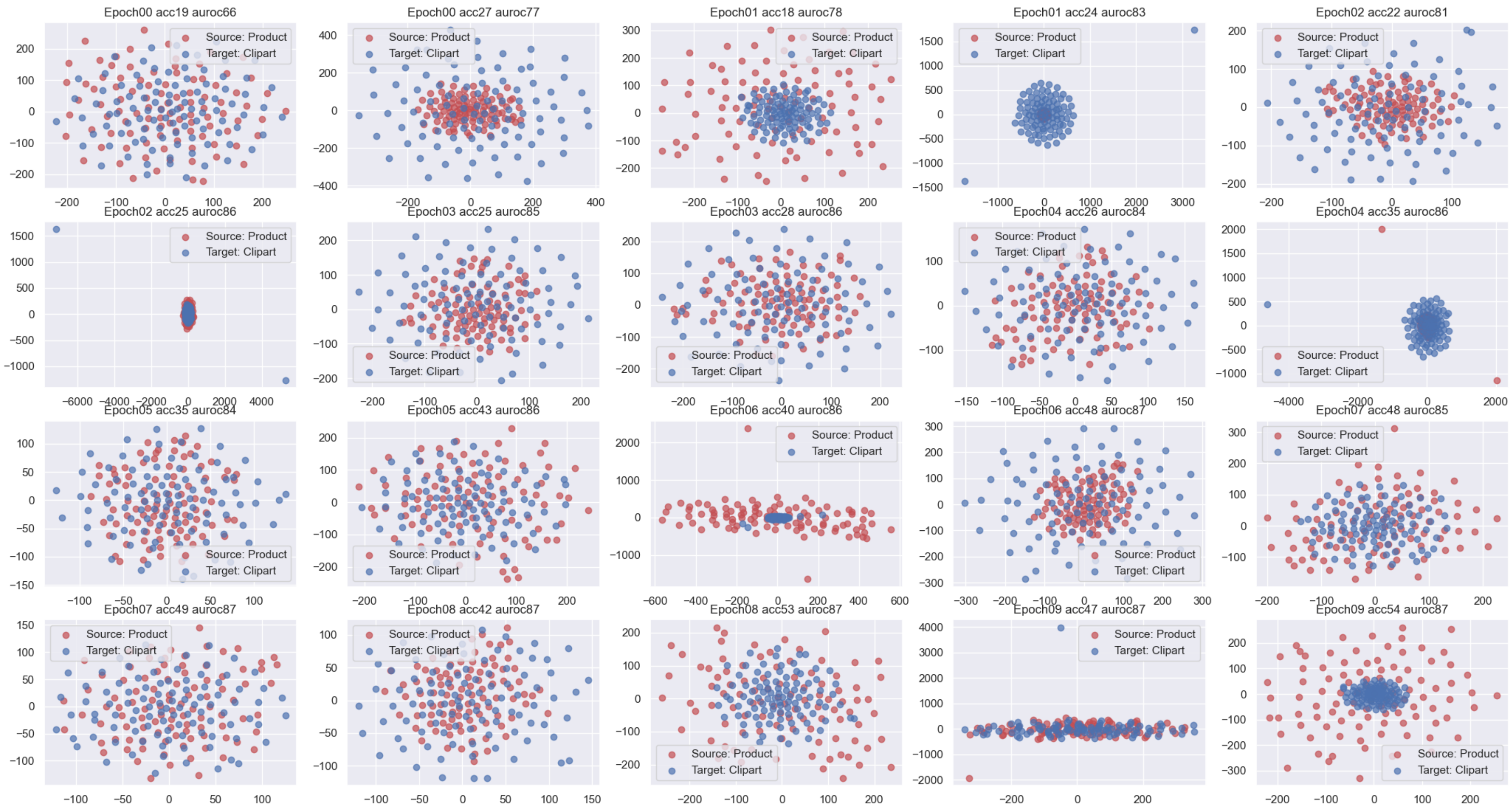
✓ Source is embedded cohesively while target is not.



[Fig. x] TSNE result by MobileNetV3 during the training.

Product to Clipart TSNE result on valid dataset. MobileNetV3 (41.24% acc)

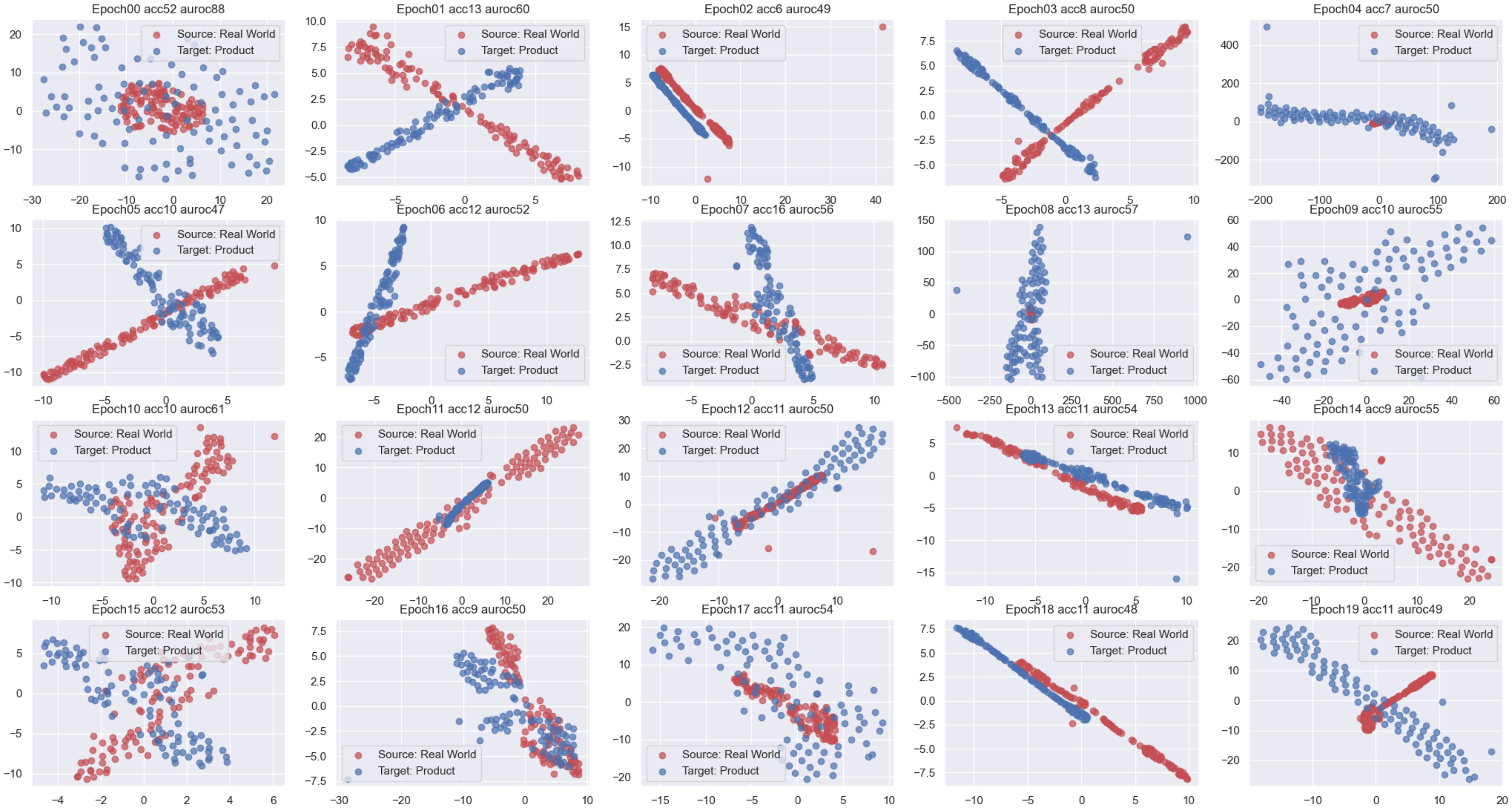
✓ Interestingly, target was cohesive too.



[Fig. x] TSNE result by MobileNetV3 during the training.

Real World to Product TSNE result on valid dataset. ResMLP (52.04% acc)

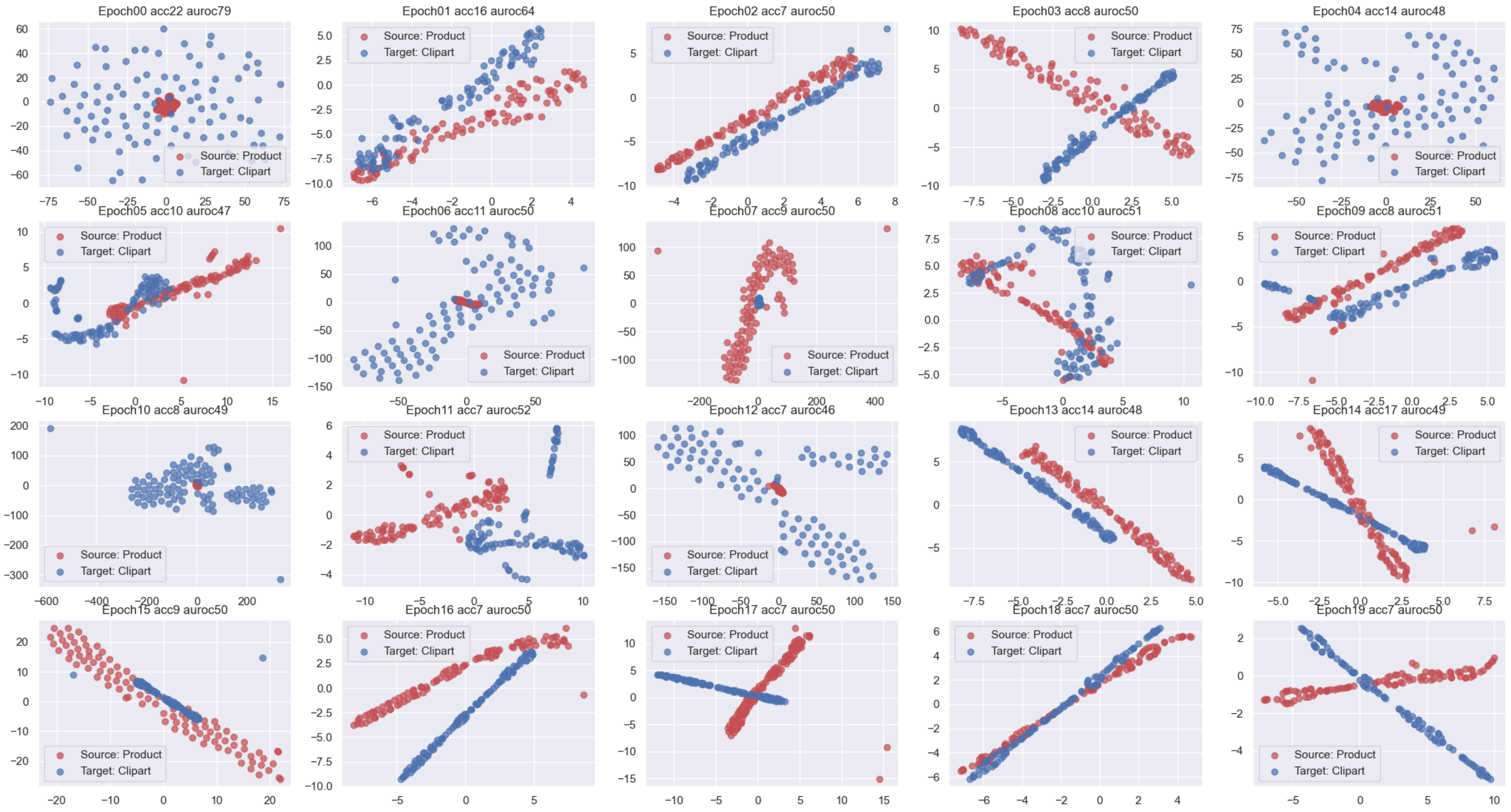
- ✓ ResMLP tends to embed everything in linear way.



[Fig. x] TSNE result by ResMLP during the training.

Product to Clipart TSNE result on valid dataset. ResMLP (22.68% acc)

✓ ResMLP tends to embed everything in linear way.



[Fig. x] TSNE result by ResMLP during the training.

Take-aways

- ✓ Through PyTorch-Adapt, we can easily implement several methods of domain adaptation.
- ✓ However, we still need to think of how to generalize all 4 datasets, and extend to general images.
- ✓ There was a shortage of data – 1,800 data in total (including all datasets)
- ✓ Cohesiveness was not significantly observable and fluctuates very much during the training.

- ✓ MobileNetV3 and ResMLP show different tendency in TSNE embeddings
- ✓ Would be interesting to see how unsupervised/semi-supervised image models would work here.

Thank you 🙏

https://github.com/1pha/DomainGeneralization

Daehyun Cho
Cogsys Lab, AI Dept
1phantasmas@korea.ac.kr