Response to Reviews of "Co-Tuning for Transfer Learning"

We thank all reviewers for their detailed reviews. Although transfer learning has many sub-areas (domain adaptation, fine-tuning, etc), fine-tuning is the most important one that widely affects CV/NLP communities and is a functionality shipped by PyTorch/TensorFlow. However, the major technique is still *feature* fine-tuning (*a.k.a.* **vanilla fine-tuning**), while a simple and more effective alternative is lacking for years. This paper aims to improve over vanilla fine-tuning with *feature and category* **Co-Tuning**, which simply needs *just a few lines of code* (see submitted code). Most reviewers find this paper simple, clearly written with significant results, and appreciate the potential impact on the NeurIPS community. In the following, we respond to common questions first and then to major concerns of each reviewer.

Common Responses: **Q1. Details of evaluation and train/val/test split.** Each dataset has a train/test split. For datasets without validation splits, 20% training data are used for validation and the rest 80% training data are used for training. This way, each dataset has a train/**val**/test split. Sampling rate further determines the actually used training data. For example, 30% sampling rate means to use 30% training data for training, leaving the rest 70% training data untouched (both labels and image/text). **Each method has access to the same set of training data.** In Alg. 1, Co-Tuning further splits the sampled training data into secondary train/val (Alg. 1, Line 4) to learn the category relationship. The empirical evaluation is **fair** because **Co-Tuning uses no extra data/labels compared to baselines**.

Q2. Adding new layers on top of source logits. Co-Tuning has two heads (Fig. 3): head-1 for target classification (same as vanilla fine-tuning) and head-2 for injecting additional supervision. Adding new layers on top of source logits means to use the output of head-2 for parametric classification, which is actually a misuse. Its accuracy is just **39.26%**, largely inferior to Co-Tuning (**52.58%**) and even inferior to vanilla fine-tuning (**45.25%**) in the setup of Table 5.

Q3. Does Co-Tuning help transfer to a dataset with more classes than the pre-trained dataset? We cannot directly answer this question because we find no target datasets with more classes than 1000 classes in the pre-trained ImageNet. However, co-tuning works across a wide spectrum of class configurations (COCO-70: 70 classes, Aircraft: 100 classes, Cars: 196 classes, CUB: 200 classes, and Places365 in R3/Q1: 365 classes). Note that these classes have **little overlap** with ImageNet classes. Therefore, we hold a positive answer to this question.

Response to R1: In fact, the vanilla fine-tuning still lacks a complete theory, not to mention Co-Tuning. Research of deep learning algorithms usually precedes the theories. We are actively seeking theoretical support for Co-Tuning.

Response to R2: Pre-train/fine-tune itself is two-stage. Co-Tuning is acceptable in this regard because it works.

Q1. Why is Co-Tuning simple? The core of Co-Tuning is to add supervision by the pre-trained task-specific classifier (**a few lines of code**). g's architecture can be fixed as one layer (*i.e.* Softmax regression), which is simple but works well for all datasets. While calibration is important as stated in [On Calibration of Modern Neural Networks, ICML 2017] (Citations 750), Co-Tuning without calibration (thereby without source data) already works well (Table 5). Even compared with re-weighting method [Learning to Transfer Learn: Reinforcement Learning-Based Selection for Adaptive Transfer Learning, ECCV 2020] which achieves 49.62% in Table 5 setup, Co-Tuning (52.58%) is better because it focuses on target data while the former may be biased by source data. The simplicity is agreed by the R1/3/4.

Q2. Hyper-parameter tuning details. All methods (both baselines and Co-Tuning) follow the same hyper-parameter tuning procedure. Each dataset has a validation split (common responses Q1), on which hyper-parameters are tuned. The learning rate ratio of new layers to pre-trained layers is fixed to 10 (a standard choice in fine-tuning), but the learning rate of pre-trained layers is tuned on validation data. We treat all methods the same and present a **fair comparison.**

Q3. Significance of results. We report standard deviations. Many gains exceed three times of standard deviation, and the rest reviewers all agree the results are significant. We will un-bold several insignificant results.

Response to R3: **Q1. Results on Places365 (semantically distinct dataset).** We sample 100 images per class for training. Fine-tuning achieves Top-1 accuracy **42.49%** while Co-Tuning improves it to **43.56%**. Although Places365 is distinct to object classification, categories learned in the latter can help scene recognition (*e.g.* existence of food indicates a "kitchen" scene). Co-Tuning may work in a wide spectrum of semantic gap between datasets.

Q2. Improvement of Co-Tuning + baseline over Co-Tuning is small. We will remove Table 4 and this argument. We will instead focus on improving Co-Tuning itself, bringing the community a strong alternative to the vanilla fine-tuning.

Q3. Use a calibrated version of Eq. (2) as the estimator. Even after calibrating the source logits $f0()$, its accuracy (**47.52%**) is inferior to Co-Tuning (**52.58%**) in the setup of Table 5. We will add these results in the paper.

Response to R4: Many thanks for your appreciation. Due to space limits, minor concerns will be addressed in a revision (more baselines, more tasks like NLI, etc). Implementation will be crystal clear since we will open-source.

Q1. Why Co-Tuning works when target classes are similar. *Even if target classes are similar, they have different source distributions provided that the pre-trained dataset is diverse enough.* Take two bird species "Crested Auklet" and "Parakeet Auklet" as example, top 3 similar ImageNet classes are in the right table to roughly represent their source distributions. Their distributions are similar (both have "black grouse" and "oystercatcher"), but still differ (one has "black swan" while the other has "junco"). In summary, Co-Tuning works for similar classes by supplementing their specific cues.

CUB Class	Top 3 Similar ImageNet Class		
`Crested Auklet`	black swan	oystercatcher	black grouse
`Parakeet Auklet`	black grouse	oystercatcher	junco

Q2. NLP experiments. Because NSP has binary outputs, transferring its categories makes less sense. The optimizer is Adam following the example code in Transformers library provided by Hugging Face. We will update Line 210.