Joint-Task Regularization for Partially Labeled Multi-Task Learning

Supplementary Material

Here, we present additional material which could not be included in the main paper due to space constraints.

6.1. JTR Pseudo-Code

Algorithm 1: Pseudo-code for JTR.

for $e = 1$ **to** num_epochs **do**
  From $\mathcal{D}$, draw a mini-batch $\mathcal{B} = ((x_0, y_0), \cdots, (x_B, y_B))$
  for $(x, y) \in \mathcal{B}$ **do**
    // get model predictions
    $\hat{y} = h_\psi(f_\phi(x))$
    // supervised loss for labeled tasks
    $\mathcal{L} = 0$
    for $t \in \mathcal{T}_x$ **do**
      $\mathcal{L} = \mathcal{L} + \frac{1}{|\mathcal{T}_x|}\mathcal{L}^t(\hat{y}^t, y^t)$
    // stack predictions and labels
    $\hat{Y} = [\hat{y}_x^1 \ \cdots \ \hat{y}_x^K]$
    $Y = [\delta(x, 1) \ \cdots \ \delta(x, K)]$
    // apply distance loss
    $\mathcal{L} = \mathcal{L} + c_{dist} \cdot \mathcal{L}_{Dist}(g_{\theta_1}(\hat{Y}), g_{\theta_1}(Y))$
    // apply reconstruction loss
    $\mathcal{L} = \mathcal{L} + c_{recon} \cdot (\mathcal{L}_{Recon}(g(\langle\hat{Y}\rangle), \langle\hat{Y}\rangle) + \mathcal{L}_{Recon}(g(\langle Y\rangle), \langle Y\rangle))$
    // update parameters
    for $\zeta \in \{\phi, \psi, \theta_1, \theta_2\}$ **do**
      $\zeta = \mathrm{SGD}(\mathcal{L}, \zeta)$

In the above pseudo-code, $\delta(x, t) = \begin{cases} y_x^t & t \in \mathcal{T}_x \\ \hat{y}_x^t & t \notin \mathcal{T}_x \end{cases}$, and $\langle\cdot\rangle$ detaches a variable from the graph to prevent backpropagation through the variable.

6.2. Implementation Details

Hyperparameters Across all experiments, we hold all hyperparameters constant for each method for all label scenarios. We use the default hyperparameter values provided by the official MTPSL source code repository for baseline methods [23, 33]. For JTR, we keep hyperparameter adjustments to a minimum with the only new/modified parameters being: $c_{dist} = 4$, $c_{recon} = 2$, and the batch size (4 for NYU-v2, 16 for Cityscapes, 8 for Taskonomy). For additional information, please refer to our code repository at github.com/KentoNishi/JTR-CVPR-2024.

Auto-Encoder We use the same SegNet architecture as the model itself. The latent dimensions are $512 \times 9 \times 12$, and we did not tune this parameter. For an ablation study of this parameter, please see the "Bottleneck Size" ablation study in Sec. 6.4. In terms of interpretation, the JTR auto-encoder captures task relations which efficiently encode predictions and labels across multiple tasks in unison. The joint-task embeddings for Y_x and $\hat{Y}_x$ can be thought of as compressed representations of predictions and targets in a shared space. The compression is facilitated by the reconstruction component of JTR—since the JTR encoder needs to compress its inputs to a reduced-dimension space, the encoder exploits commonalities and patterns across all tasks to create a compact and non-trivial encoding. This means that regularization applied in this feature space is not per-task; rather, all tasks are regularized at once.

Complexities of MTPSL MTPSL relies on FiLM to condition on task-pairs. This involves adding parameters of shapes $(2C) \times K \times (K-1)$ and $1 \times C$ for *every* Conv2D layer in MTPSL (C channels, K tasks), as well as implementing affine transformations for each layer. FiLM parameters also require a separate optimizer, LR, scheduler, etc. This makes using off-the-shelf architectures (*i.e.* ResNet) challenging, since the model's inner code must be modified. In contrast, JTR has no per-layer intricacies and is easier to implement.

Batching JTR and all baselines use simple random batching for nearly all experiments. Two notable exceptions are NYU-v2 "extended" and "Partially (Un)labeled MTL," for which we under/over-sample and append the additional data to each batch by a fixed quantity. The number of additional data examples added per batch are 4 for NYU-v2 "extended" and 1 for "Partially (Un)labeled MTL."

6.3. Explaining Results

Copied Baselines For every experiment we ran, MTPSL and JTR use the same augmentation technique and dataloader. This ensures that any performance gain is solely due to JTR's advantages over MTPSL. However, for results for non-MTPSL baselines marked with * in some tables, we present scores directly sourced from [33] to maintain consistency, since some baselines (*i.e.* Direct / Perceptual Map) were not reproducible with RandAugment. For further clarity, we supplement our results with comparisons of MTPSL and JTR relative to Supervised MTL reproduced with RandAugment in Tab. 10.

Method	Seg. ↑	Depth ↓	Norm. ↓	$\overline{\Delta\%}$ ↑
NYU-v2 "onelabel"				
Supervised MTL	22.52	0.6920	35.27	+0.000
MTPSL	30.40	0.5926	31.68	+19.84
Ours (JTR)	31.96	0.5919	30.80	+23.02
NYU-v2 "randomlabels"				
Supervised MTL	31.87	0.5957	31.64	+0.000
MTPSL	35.60	0.5576	29.70	+8.077
Ours (JTR)	37.08	0.5541	29.44	+10.09
Cityscapes "onelabel"				
Supervised MTL	68.35	0.0178	—	+0.000
MTPSL	72.09	0.0168	—	+5.545
Ours (JTR)	72.33	0.0163	—	+7.125

Table 10. Results for MTPSL and JTR relative to Supervised MTL results reproduced with RandAugment.

Dataset	Scenario	Seg. ↑	Depth ↓	Norm. ↓
NYU-v2	*onelabel*	23.65	0.7513	30.56
NYU-v2	*randomlabels*	28.68	0.7530	29.47
Cityscapes	*onelabel*	70.32	0.0134	—

Table 11. Per-task Single-Task Learning (STL) results.

Method	NYU-v2		Cityscapes
	onelabel	*randomlabels*	*onelabel*
Supervised MTL	–4.100	+8.216	–17.82
MTPSL	+15.33	+16.43	–11.43
Ours (JTR)	+18.52	+18.60	–9.392

Table 12. $\overline{\Delta\%}$ scores relative to per-task STL results on NYU-v2 and Cityscapes. All results were obtained with RandAugment.

Method	*onelabel*	*randomlabels*	*halflabels*
Supervised MTL	+2.594	–2.817	–1.859
Consistency Reg.	+3.622	+0.592	+2.025
MTPSL	+5.852	+3.248	+2.026
Ours (JTR)	+5.482	+3.919	+2.785

Table 13. $\overline{\Delta\%}$ scores relative to STL on Taskonomy.

Statistical Significance MTL training is expensive; therefore, we followed previous works [33, 38, 58, 71] in running each method only once using the same seed. For additional clarity, we also ran JTR and MTPSL three times with the same seed on NYUv2 "randomlabels." The 95% confidence intervals in Tab. 14 show that JTR's $\overline{\Delta\%}$ gap is statistically significant.

Single-Task Learning Baseline In our experiments, we report the $\overline{\Delta\%}$ scores relative to *Supervised MTL* because we believe it is the most relevant and fair comparison baseline. Alternatively, comparisons can be made against scores achieved with Single-Task Learning (STL). However, STL takes at least N-times more parameters, VRAM, and computation than an N-task MTL model, and is thus not a representative baseline. Nonetheless, we present comparisons with STL in Tab. 11, Tab. 12, and Tab. 13.

Method	Seg. ↑	Depth ↓	Norm. ↓	$\overline{\Delta\%}$ ↑
MTPSL	35.45±0.247	0.5636±0.0059	29.95±0.344	+18.92±0.88
JTR	36.86±0.574	0.5517±0.0024	29.39±0.088	+21.82±0.58

Table 14. Mean and SD on NYU-v2 *randomlabels*.

Method	Seg. ↑	Depth ↓	Norm. ↓	$\overline{\Delta\%}$ ↑
Supervised MTL	48.79	0.4528	25.35	+0.000
Consistency Reg.	46.02	0.4855	27.44	–7.048
MTPSL [33]	48.37	0.4228	25.39	+1.869
Ours (JTR)	**49.07**	**0.4129**	**25.05**	**+3.523**

Table 15. Partially labeled multi-task learning results on NYU-v2 "randomlabels" using DeepLabV3+ with ResNet-50.

Task Performance Tradeoffs In some experiments, JTR performs marginally lower than MTPSL in one task while scoring significantly higher in another (Tab. 1, Tab. 3, Tab. 5). Our understanding is that this is due to the tradeoff between tasks in MTL (*i.e.* due to conflicting gradients [55, 71] and negative transfer [31, 39]). Crucially, JTR's lowest-scoring tasks are still competitive with MTPSL, while its highest-scoring tasks significantly outperform MTPSL, yielding better overall $\overline{\Delta\%}$ performance.

6.4. Ablation Studies

Model-Agnosticism Because JTR is a model-agnostic method, it can be applied to any dense multi-task prediction architecture. We demonstrate this general agnosticism by benchmarking "Supervised MTL," "Consistency Regularization,", MTPSL [33], and JTR using a DeepLabV3+ model with the ResNet-50 architecture as the backbone. This study was carried out under the NYU-v2 "randomlabels" scenario using a batch size of 4. We present the results in Tab. 15. In comparison to the earlier SegNet experiments in Tab. 1, all of the methods perform better; however, the model trained under JTR still obtains the best performance in all tasks, as well as the highest $\overline{\Delta\%}$ score. These results showcase the general applicability of JTR regardless of the model architecture. This setting corresponds to the ResNet-50 resource usage benchmarks in Tab. 6 in the main paper.

Joint-Task Latent Distance Metric In our experiments, we primarily used cosine distance (dot product of normalized vectors) as our joint-task latent embedding distance metric. However, other loss functions such as the L1/L2 distance can also be used as alternatives. We present our comparisons between L1 norm, L2 norm, and cosine distance loss functions in Tab. 16. These results show that co-

Function	Seg. ↑	Depth ↓	Norm. ↓	$\overline{\Delta\%}$ ↑
Supervised MTL	31.87	0.5957	31.64	+0.000
L1 Norm.	35.21	0.5736	33.48	+2.792
L2 Norm.	35.11	0.5608	32.94	+3.972
Cosine Distance	**37.08**	**0.5541**	**29.44**	**+10.09**

Table 16. Ablation study of loss functions for $\mathcal{L}_{Dist}$ with NYU-v2 "randomlabels." All results were obtained with RandAugment.

Dimensions	Seg. ↑	Depth ↓	Norm. ↓	$\overline{\Delta\%}$ ↑
Supervised MTL	31.87	0.5957	31.64	+0.000
$256 \times 9 \times 12$	35.93	0.5583	29.91	+8.162
$512 \times 9 \times 12$	**37.08**	**0.5541**	**29.44**	**+10.09**
$1024 \times 9 \times 12$	35.69	0.5641	29.94	+7.555

Table 17. Ablation study of the JTR auto-encoder bottleneck dimensions with NYU-v2 "randomlabels." All results were obtained with RandAugment.

sine similarity is a favorable distance metric for the joint-task latent space; hence, we use cosine similarity as $\mathcal{L}_{Dist}$ in all of our experiments.

Bottleneck Size The latent dimensions of our SegNet auto-encoder in JTR are $512 \times 9 \times 12$ (the same dimensions as the MTL model). In Tab. 17, we perform an ablation study by varying the dimensions of the JTR auto-encoder bottleneck across $(256 \times 9 \times 12)$, $(512 \times 9 \times 12)$, and $(1024 \times 9 \times 12)$. We find that $(512 \times 9 \times 12)$ is the most effective bottleneck size. This is consistent with the dimensions of the SegNet instances used in the MTPSL codebase (for both the MTL model and the auxiliary modules) [23, 33].

6.5. Additional Resource Usage Comparisons

In Tab. 6, we present training time and VRAM usage comparisons for MTPSL [33] and JTR. For completeness, we also benchmark the "Supervised MTL" and "Consistency Regularization" baselines in Tab. 18 and Tab. 19. The results show that both MTPSL and JTR are more resource-intensive than naïve baselines. However, this tradeoff is worthwhile given the high prediction performance attained by the two methods in exchange. Furthermore, comparing the JTR training time to the supervised baseline on Taskonomy, JTR only takes 8 additional hours to train. Meanwhile, MTPSL takes about 126 additional hours. This shows that JTR indeed scales more favorably with more tasks.

7. Visualizations

We provide comparisons of model predictions and labels for examples in the NYU-v2 test set in Fig. 3. Additionally, we provide visualizations of JTR auto-encoder inputs and reconstructions for examples in the NYU-v2 "randomlabels" training set in Fig. 4.

Method	SegNet		ResNet-50	
	Time ↓	VRAM ↓	Time ↓	VRAM ↓
Supervised MTL	4h 25m	4GiB	4h 10m	3.5GiB
Consistency Reg.	4h 30m	4GiB	4h 10m	3.5GiB
MTPSL [33]	8h 20m	17.5GiB	17h 40m	21.5GiB
Ours (JTR)	9h 00m	17.5GiB	10h 50m	16.2GiB

Table 18. Time and VRAM requirements for training on NYU-v2 using a fixed batch size of 4 on an NVIDIA A100 for 300 epochs.

Method	Cityscapes		Taskonomy	
	Time ↓	VRAM ↓	Time ↓	VRAM ↓
Cityscapes (2 tasks), onelabel, SegNet				
Supervised MTL	4h 10m	10.9GiB	97h 00m	9.9GiB
Consistency Reg.	4h 15m	10.9GiB	97h 00m	10.2GiB
MTPSL [33]	22h 10m	14.4GiB	223h 10m	34.4GiB
Ours (JTR)	23h 45m	19.2GiB	105h 00m	23.9GiB

Table 19. Time and VRAM requirements for training on Cityscapes and Taskonomy using the SegNet architecture on an NVIDIA A100, with 300 and 20 epochs respectively and batch sizes 16 and 8 respectively.

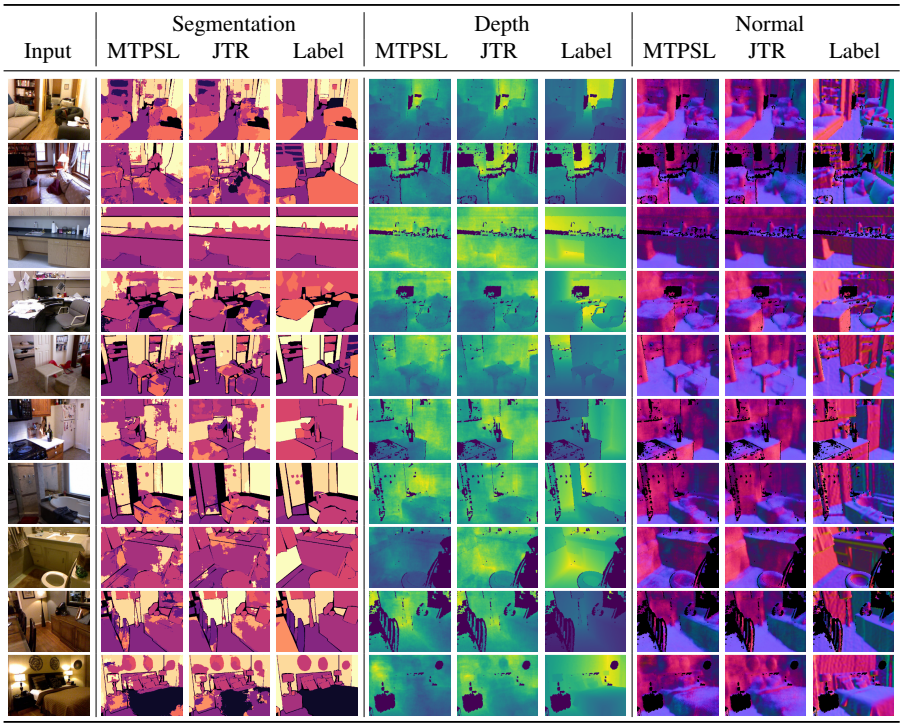

Figure 3. Comparison of dense prediction outputs from a SegNet trained with MTPSL and JTR on NYU-v2 "randomlabels" alongside the corresponding input image and ground-truth labels. Examples are sampled randomly from the test set without cherry-picking. Visually, the predictions of the JTR model are slightly more accurate than those of the MTPSL model.

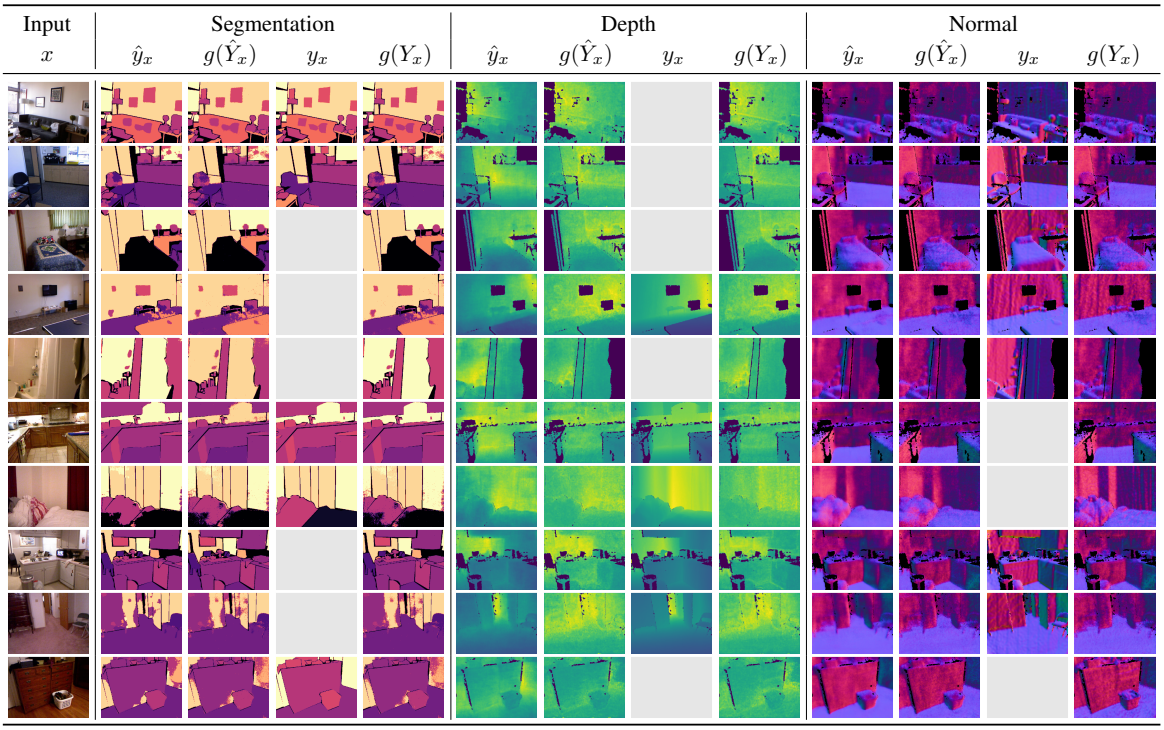

Figure 4. Visualization of input images x, dense prediction outputs $\hat{y}$, labels y, JTR reconstructions of the noisy prediction tensor $g(\hat{Y}_x)$, and JTR reconstructions of the reliable target tensor $g(Y_x)$ for a SegNet trained with JTR on NYU-v2 "randomlabels." Examples are randomly sampled from the training set without cherry-picking. Missing y_x entries indicate unlabeled tasks under "randomlabels."