Attentive Single-Tasking of Multiple Tasks

Appendix

The following additional information is provided in this Appendix:

- Ablated results using different backbone architectures for NYUD and FSV datasets in Section .
- Baselines using UberNet with and without the proposed modulation and adversarial training in Section .
- Results using MobileNet as the backbone architecture, in order to highlight potential mobile phone applications in Section .
- Technical details for training and testing in Section . Code will also be published.
- Qualitative results in Fig. 2 and Fig. 3

Appendix A: More results on NYUD and FSV

Table 1 illustrates the quantitative results obtained by our method on NYUD [11] and FSV [9], by changing the backbone architecture. Results are consistent among backbones, and by including modulation and adversarial training to the pipeline, we get improved results with respect to the multi-task and single-task baselines, irrespective of the network depth.

backbone	SEA	#T	Edge $\uparrow$	Seg $\uparrow$	Norm $\downarrow$	Depth $\downarrow$	$\Delta_m\%\downarrow$
R-26		1	72.9	29.87	24.34	0.650	0
R-26		4	72.4	27.74	24.83	0.729	5.50
R-26	✓	4	73.5	30.07	24.316	0.625	**-1.36**
R-50		1	74.4	32.82	23.3	0.610	
R-50		4	73.2	30.95	23.34	0.700	5.44
R-50	✓	4	74.5	32.16	23.18	0.570	**-1.22**
R-101		1	74.9	35.90	22.90	0.580	
R-101		4	73.8	31.20	23.07	0.650	6.63
R-101	✓	4	75.6	35.60	22.73	0.560	**-1.07**

(a) Our method on NYUD [11], for different backbones.

backbone	SEA	#T	Seg $\uparrow$	Albedo $\downarrow$	Depth $\downarrow$	$\Delta_m\%\downarrow$
R-26		1	69.77	0.087	0.065	0
R-26		3	66.71	0.090	0.073	6.41
R-26	✓	3	71.36	0.085	0.065	**-1.80**
R-50		1	71.14	0.086	0.063	0
R-50		3	66.90	0.093	0.078	7.04
R-50	✓	3	70.69	0.085	0.063	**-0.02**
R-101		1	72.10	0.086	0.063	0
R-101		3	68.12	0.091	0.072	8.75
R-101	✓	3	72.24	0.083	0.062	**-1.57**

(b) Our method on FSV [9], for different backbones.

Table 1. **Our method for NYUD, and FSV**: Results with different backbones: R-26, R-50, and R-101. Negative drop indicates improved performance with respect to the single-tasking baseline. Arrows indicate desired behaviour of each metric.

Appendix B: UberNet baseline

The architecture of [8] learns multiple tasks by using a common backbone and a light-weight decoder (skip connections and 1×1 convolutions) per task. We re-implement the UberNet architecture, and we substitute the VGG [15] backbone of the original work with the SE-ResNet [6] used in our work. The main difference with our architecture are the skip connections and 1×1 convolutions that comprise each task-specific head, instead of the powerful Deeplab-v3+ ASPP decoder [2] used throughout our work. Similarly to our work, and similarly to the observation of the original author, we observe a non-trivial drop in performance when learning to multi-task with a common, entirely shared backbone (Table 2). We plug-in SE and adversarial training and we recover most of the drop. We provide results for all 3 datasets that have been used throughout this work. Results obtained by our architecture are presented in the last row of each table. The relatively lower performance especially for semantic tasks compared to our method is due to the absence of a strong decoder. The observations regarding modulation and adversarial training are, nevertheless, consistent.

Appendix C: MobileNet-v2 backbone for multiple tasks

Our multi-tasking framework could find application in light-weight CNNs designed for mobile phone applications. For example, by using our framework, many different tasks can be executed with only a single and small set of parameters being shipped to the end user. To test this idea, we change our backbone to the light-weight MobileNet-v2 [13], an architecture specifically designed for mobile phones. We change the decoder accordingly: convolutions

backbone	SEA	#T	Edge ↑	Seg ↑	Parts ↑	Norm ↓	Sal ↑	$\Delta_m\%$ ↓
R-50-Uber		1	71.7	66.90	59.80	15.00	64.56	
R-50-Uber		5	70.3	60.90	57.00	16.65	62.15	7.10
R-50-Uber	✓	5	70.5	65.50	60.15	14.94	64.98	**0.43**
R-50	✓	5	72.4	68.00	61.10	14.80	65.70	n.a

(a) UberNet results in PASCAL.

backbone	SEA	#T	Edge ↑	Seg ↑	Norm ↓	Depth ↓	$\Delta_m\%$ ↓
R-50-Uber		1	73.9	32.50	22.90	0.669	0
R-50-Uber		4	72.3	29.48	24.16	0.716	6.00
R-50-Uber	✓	4	73.7	31.19	23.46	0.632	**0.30**
R-50	✓	4	74.5	32.20	23.20	0.570	n.a

(b) UberNet results in NYUD.

backbone	SEA	#T	Seg ↑	Albedo ↓	Depth ↓	$\Delta_m\%$ ↓
R-50-Uber		1	70.26	0.092	0.101	0
R-50-Uber		3	67.02	0.093	0.124	9.50
R-50-Uber	✓	3	69.45	0.091	0.111	**3.32**
R-50	✓	3	70.70	0.085	0.063	n.a

(c) UberNet results in FSV.

Table 2. **UberNet for PASCAL, NYUD, and FSV**: Standard multi-task learning results in a significant drop in performance, that is recovered with modulation and adversarial training. Last rows of each table present results obtained by our architecture.

backbone	enc	dec	#T	Edge ↑	Seg ↑	Parts ↑	Norm ↓	Sal ↑	$\Delta_m\%$ ↓
MNet			1	69.5	62.10	54.88	14.88	66.30	0
MNet			5	67.2	54.10	53.00	16.76	62.70	7.57
MNet		✓	5	67.5	57.40	54.50	16.55	63.00	5.47
MNet	✓	✓	5	69.2	61.60	55.17	15.21	65.60	**0.97**

Table 3. **Results using MobileNet in PASCAL**: Modulation with SE is able to recover the performance that is lost using standard multi-task learning.

are changed to depth-wise convolutions, and ReLU activations are changed to ReLU6. We pre-train a variant that uses Squeeze and Excitation modules on ImageNet (SE-MobileNet), and fine-tune for multi-task learning. We test standard multi-tasking against the variants of our method that use SE modulation. Table 3 summarizes our findings. Similarly to the experiments using SE-ResNet, disentangling the representations for each task also helps for MobileNet. By using SE per task both on the encoder and decoder, our method outperforms the single-tasking baseline. Figure 1 puts these results in perspective, comparing them to results obtained by SE-ResNet architecture. It is remarkable that by using modulation, MobileNet is no worse than the R-50 standard multi-tasking baseline using much less computational cost, and only 8% of its parameters.

Appendix D: Implementation Details

In this section we provide the technical details for our implementation.

Generic hyper-parameters: The entire hyper-parameter search was performed on the single-task baselines. For all tasks, we use synchronized SGD with momentum of 0.9 and weight decay of 1e-4. We set the initial learning rate to 0.005 and use the poly learning rate [1]. All our models are trained on a single GPU with batch size 8, and spatial input of 512 × 512. We used multi-GPU training with batch size 16 and synchronous batchnorm layers only for the sanity check experiments (Table 2 of main paper), for a fair comparison with competing methods. Standard flipping, rotations, and scaling was used for data augmentation. The number of total epochs is set to 60 for PASCAL [5], 200 for NYUD [11], and 3 for the large-scale FSV [9].

Weighting of the losses: Related work deals with automatically weighting of the losses for multi-task learning [3, 14]. We compared these methods to selecting the optimal weights by grid-search. In our setup, we found out that grid-search works better, probably because of the very imbalanced optimal parameters (optimal weighting of loss for edge detection is 50 times higher than semantic segmentation, for example). In particular, we re-implemented [14] that uses multi-objective optimization to re-weight the gradients from each task, in order to optimize the shared parts of the network towards a direction useful for all tasks. This lead to better results than uniform weights, however we obtained better results by simple grid-search.

For all our experiments, when training for multiple tasks, we divide the learning rate of the shared layers by the number of tasks (T), since T updates are happening for the same mini-batch on the shared part of the network.

During training, we used the following formula for the weights of the losses:

$$L = (1 - w_d) \cdot \sum_{t=1}^{T} w_t \cdot L_t + w_d \cdot L_d, \qquad (1)$$

where w_t weights the loss L_t of task t, and w_d the loss L_d of the discriminator. All losses are averaged to the number of samples that the prediction contains ($W \times H \times C \times N$).

Edge Detection: For edge detection, we use $w_t = 50$ and binary cross-entropy loss. As is common practice [16, 7, 10], the positive pixels are weighted more (0.95) than the negative ones (0.05), to account for the class imbalance. When training for a single task in BSDS (Table 2 of main paper), where there are more than a single annotators, we use the multi-instance learning (MIL) loss of [7]. No MIL is used when training in PASCAL or NYUD. We follow the common evaluation on those two datasets [10] by setting the maximum allowed mis-localization of edges (`maxDist` parameter) to 0.0075 and 0.011 for PASCAL and NYUD, respectively.

Semantic Segmentation: For semantic segmentation, we used $w_t = 1$, and cross-entropy loss. When training on VOC trainaug [2] (Table 2 of main paper), we did not finetune separately on VOC val.

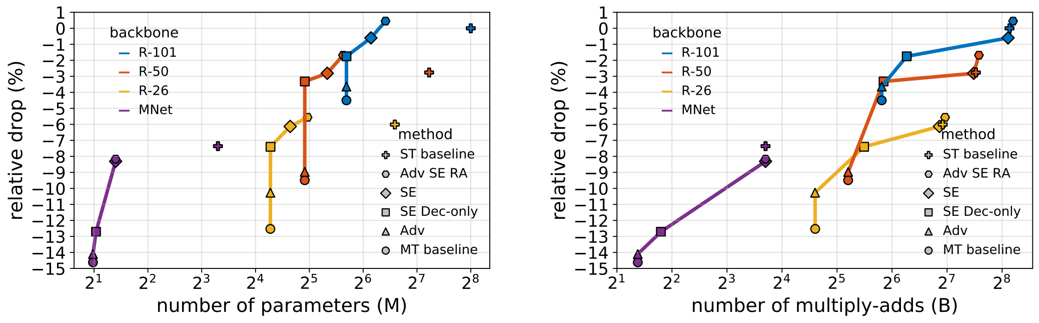

Figure 1. **Performance vs. Resources for MobileNet:** Average relative drop as a function of the number of parameters (left), and multiply-adds (right), for various points of operation of our method. Different backbones are indicated with different colors. MobileNet with our modulation is able to reach R-50 results for standard multi-tasking, by using much less parameters and computation.

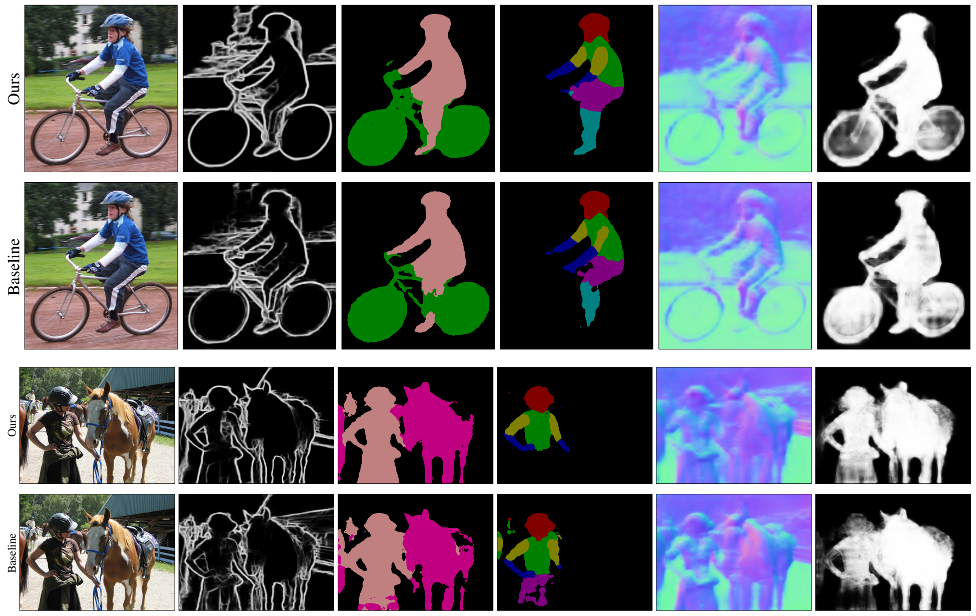

Figure 2. **More qualitative Results in PASCAL:** We compare our strongest model to results obtained by standard multi-tasking. The baseline suffers from mixing features (eg. edge features appear in saliency results), whereas our method results in smoother results.

Human Part Segmentation: For human part segmentation, we used $w_t = 2.$ and cross-entropy loss. Samples that do not contain any humans did not contribute to the loss.

Surface Normals: For surface normal estimation, we used $w_t = 10$ and $\mathcal{L}_1$ loss with unit-vector normalization. During rotation augmentation, we carefully rotate the unit vector of the surface normals accordingly, to point to a consistent direction.

Albedo: For albedo, we use $w_t = 10$, and standard $\mathcal{L}_1$ loss. We emphasize that our architecture is not optimal for this task, since the output is 4 times smaller than the input, and tiny details are not captured. Using an architecture suited for albedo is out of the scope of this work.

Monocular Depth: For monocular depth estimation we use $w_t = 1$, and the loss of [4], which is a combination of $\mathcal{L}_1$ loss and a smoothness term that enforces the spatial gradients of the prediction to be consistent with the ones of the ground truth. Our experiments showed that including the smoothness term to the loss leads to better results, quantitatively and qualitatively.

Discriminator: We use a fully-convolutional discriminator, which consists of two 1×1 conv layers and a ReLU activation. We did not observe improvements when using discriminators of larger depth. We normalize the gradient of the losses of the tasks by their norm before passing them through the discriminator. This practice makes training more stable because the norm of the gradients becomes smaller as training progresses. We use $w_d = 0.1$.

Further Technical Details: Our work is implemented in PyTorch [12]. During weight update PyTorch applies momentum and weight decay to all modules in the definition of a network. This behaviour is not desired When us-

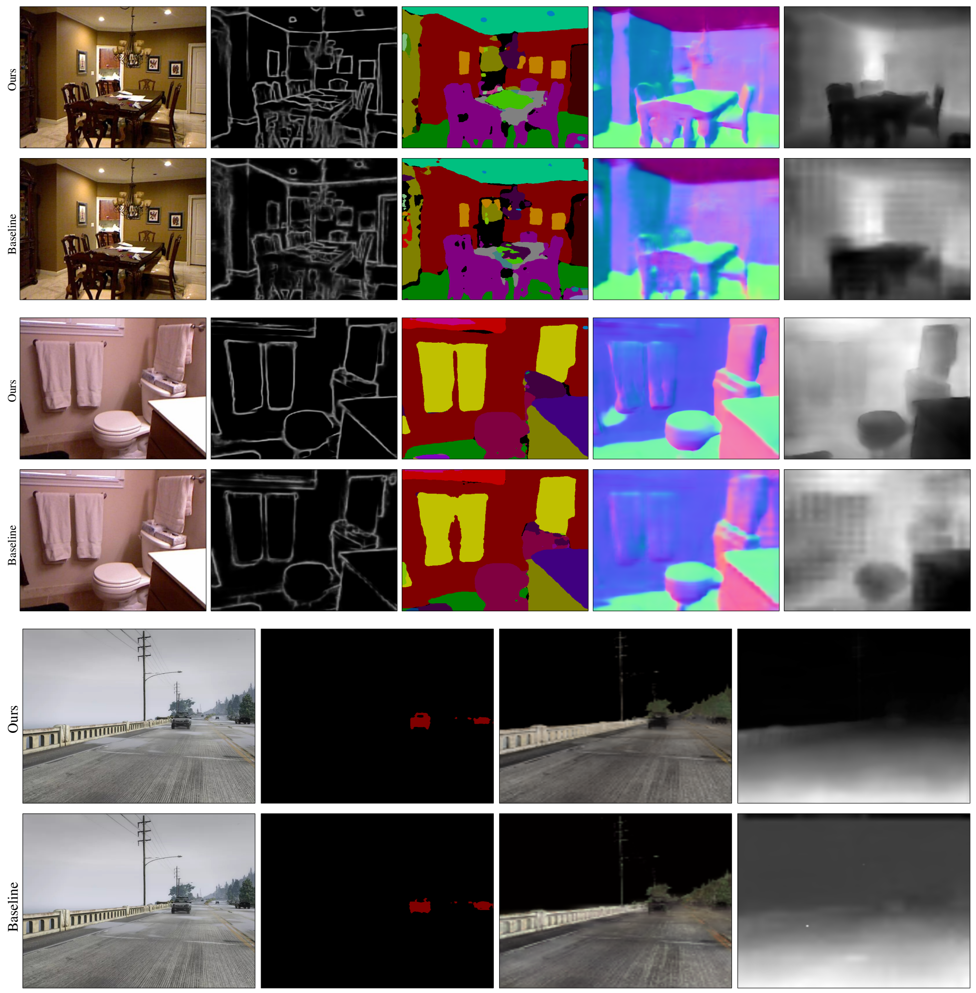

Figure 3. **More qualitative Results in NYUD and FSV**: We compare our strongest model to results obtained by standard multi-tasking in NYUD (rows 1-4), and FSV (rows 5-6). In contrast to the baseline, our method does not suffer from degraded performance coming from checkerboard effects in depth, blurry edges, or mixed semantic classes.

ing generic and task-specific weights, since the task-specific ones are only used in the forward pass of the particular task, which leads to $T - 1$ unwanted updates. This behaviour is avoided by tracing the graph of computation and updating only the weights that were used, which also translated into quantitative improvements.

References

[1] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs. *T-PAMI*, 2017. 2

[2] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 1, 2

[3] Z. Chen, V. Badrinarayanan, C.-Y. Lee, and A. Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv:1711.02257*, 2018. 2

[4] D. Eigen and R. Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *ICCV*, 2015. 3

[5] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. 2

[6] J. Hu, L. Shen, and G. Sun. Squeeze-and-excitation networks. In *CVPR*, 2018. 1

[7] I. Kokkinos. Pushing the boundaries of boundary detection using deep learning. In *ICLR*, 2016. 2

[8] I. Kokkinos. UberNet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, 2017. 1

[9] P. Krähenbühl. Free supervision from video games. In *CVPR*, 2018. 1, 2

[10] K. Maninis, J. Pont-Tuset, P. Arbeláez, and L. V. Gool. Convolutional oriented boundaries: From image segmentation to high-level tasks. *T-PAMI*, 40(4):819 – 833, 2018. 2

[11] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from RGBD images. In *ECCV*, 2012. 1, 2

[12] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. In *NIPS-W*, 2017. 3

[13] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 1

[14] O. Sener and V. Koltun. Multi-task learning as multi-objective optimization. In *NIPS*, 2018. 2

[15] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In *ICLR*, 2015. 1

[16] S. Xie and Z. Tu. Holistically-nested edge detection. *IJCV*, pages 1–16, 2017. 2