Learning Multiple Pixelwise Tasks Based on Loss Scale Balancing

Jae-Han Lee
Korea University
jaehanlee@mcl.korea.ac.kr

Chul Lee
Dongguk University
chullee@dongguk.edu

Chang-Su Kim
Korea University
changsukim@korea.ac.kr

Abstract

We propose a novel loss weighting algorithm, called loss scale balancing (LSB), for multi-task learning (MTL) of pixelwise vision tasks. An MTL model is trained to estimate multiple pixelwise predictions using an overall loss, which is a linear combination of individual task losses. The proposed algorithm dynamically adjusts the linear weights to learn all tasks effectively. Instead of controlling the trend of each loss value directly, we balance the loss scale — the product of the loss value and its weight — periodically. In addition, by evaluating the difficulty of each task based on the previous loss record, the proposed algorithm focuses more on difficult tasks during training. Experimental results show that the proposed algorithm outperforms conventional weighting algorithms for MTL of various pixelwise tasks. Codes are available at https://github.com/jaehanlee-mcl/LSB-MTL.

1. Introduction

Multi-task learning (MTL) is a machine learning technique to solve multiple learning tasks simultaneously while exploiting commonalities and differences across tasks. MTL can improve the learning efficiency and prediction accuracy for related tasks using an integrated model, as compared to separate models trained for the multiple tasks independently. The effectiveness of MTL has been proven both theoretically and experimentally [1–4, 11]. With the advances in deep learning, MTL has been employed in a wide range of applications, such as computer vision [14, 19, 32], natural language processing (NLP) [7, 10, 28], reinforcement learning [12,17,35], and speech recognition [8,16,39].

In computer vision, MTL models using convolutional neural networks (CNNs) have been proposed mainly. For example, MTL models have been developed to jointly perform three pixelwise tasks of depth estimation, surface normal estimation, and semantic segmentation from an input image [25, 46]. Also, the joint depth and motion estimation from a video [6, 44] and the classification of an image for multiple attributes [30, 37] have been studied. These researches have advanced the learning of closely related tasks using a single MTL model. Since a large part of the network is shared between tasks, an MTL model is advantageous in terms of complexity, inference time, and learning efficiency.

To develop an effective deep MTL algorithm, two factors should be considered: architecture and training scheme. First, the architecture should be designed to learn both task-across and task-specific representations by allocating network parameters appropriately for shared and task-specific purposes. Many architectures have been developed by considering various factors, such as the CNN capacity, data type, and relationship between tasks [12, 29, 34]. Second, a training scheme should discourage any bias toward a specific task [6,18,25]. Because an MTL model generates multiple estimates, the corresponding losses should be defined and then combined to form an overall loss. Since each loss has a different scale during training, the overall loss may be dominated by a specific loss. Moreover, the losses may vary in different directions or even fluctuate during training. It is hence important to balance loss contributions and thus enforce that MTL learns all tasks effectively.

In this paper, we propose a novel loss weighting algorithm, called loss scale balancing (LSB), for MTL of pixelwise computer vision tasks (*e.g.* depth estimation and semantic segmentation), which can dynamically adjust weights to learn all tasks effectively. Based on the observation that balancing the *loss scale*, which is the product of a loss value and its weight, is more efficient than controlling the trend of each loss value directly, we adjust the loss scales to be balanced periodically. Besides, by evaluating the difficulty of each task using the previous loss record, the proposed algorithm can focus more on difficult tasks. Experimental results demonstrate that the proposed algorithm can successfully balance loss scales to improve the performance of each task. The proposed LSB algorithm outperforms conventional weighting algorithms.

The main contributions of this paper are as follows:

- We propose balancing loss scales, instead of losses themselves, to improve the performance of MTL of pixelwise vision tasks.
- We further improve the performance by tuning loss

scales through the assessment of task difficulties.

- It is shown experimentally that the proposed algorithm outperforms conventional ones consistently, regardless of MTL architecture, dataset, and encoder backbone.

2. Related Work

MTL architectures: A single MTL model should provide an estimate for each of the multiple tasks by considering both general and task-specific representations of input data. Thus, an MTL architecture should be designed to allocate limited parameters for shared and task-specific purposes appropriately.

Figure 1 shows three typical MTL architectures, in comparison with single-task learning (STL). The simplest one is to add multiple estimation layers at the end of an STL network, as shown in Figure 1(b). Because of its simplicity and efficiency in computation and memory space, the full-sharing architecture has been widely employed for multilingual recognition [16], pose estimation and action detection [13], facial landmark detection [47], multiple classification [37], as well as for pixelwise visions. However, it may fail to learn task-specific representations effectively because all hidden layers are shared across tasks.

Figure 1(c) is the multi-decoder architecture, composed of a shared encoder and multiple task-specific decoders. This architecture has been employed for computer vision [5, 18, 31] and NLP [7, 26] applications. Some models adopt feature selection modules to transfer the representations from shared encoder output selectively to task-specific decoders. Various techniques for selecting features for each task have been developed: linear combination of intermediate features [9], multi-scale fusion [19], connection of layers at different depths [14], and attention modules [25].

Finally, the multi-column architecture is in Figure 1(d), in which an encoder-decoder network is designed for each task and modules for sharing features across the networks are added. Thus, the encoder parameters are soft-shared across tasks. The main issue in this architecture is how to share features between tasks effectively; the linear combination of features [29, 33], one-way transfer between tasks [35], and distillation modules [43] have been tried.

In addition to the MTL architectures in Figure 1, several others have been developed as well. In [12], previous approaches were extended by exploiting multiple input and output layers for different tasks and adopting many convolutional paths connecting them. In [48], a single-encoder-multi-decoder architecture was designed, but, similarly to the multi-column approach, features were transferred between layers for different tasks.

MTL model training: When training an MTL model, losses for different tasks interact intricately with one another via backpropagation. The weighting of these losses has a significant impact on the performance of each task [18]. These weights can be manually determined [13, 19, 39, 41], which, however, requires expert knowledge on individual loss functions and extensive trial-and-errors.

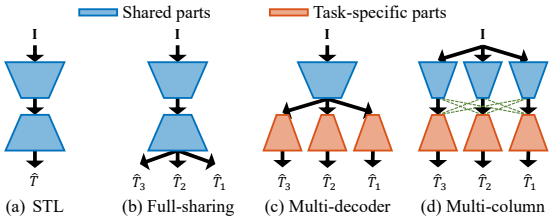


Figure 1. Typical architectures for MTL in the case of triple tasks.

Dynamic weighting, which adjusts weights of losses during training, can overcome this problem. Recent dynamic weighting algorithms update weights periodically based on the uncertainty of losses [18], the rates of change of losses [25], and the weighted geometric mean of losses [6]. In [40], an attempt was made to achieve the Pareto optimality between losses. GradNorm [5] also adjusts loss weights by monitoring loss reduction rates over time. However, unlike the proposed algorithm, it needs to access the gradients of losses with respect to shared network parameters. Thus, when tasks are connected in a complicated way, such as in the multi-column architecture in Figure 1(d), GradNorm may demand a high computational cost. Therefore, it is applied only to the multi-decoder architecture in Figure 1(c) in [5].

Curriculum learning [14, 23, 30, 37] is another effective approach to MTL that starts with easy tasks and gradually moves on to more difficult tasks. In [30, 37], the learning sequence of tasks was determined according to the correlation between tasks. In [23], instance difficulties, as well as task difficulties, were considered. In [14], difficulties of tasks were measured and used for learning schedule similarly to the proposed algorithm. However, the proposed algorithm differs from the training of other MTL algorithms in that it adjusts weights based on loss scales. In [22], loss scales were considered for training networks, but it only focused on the monocular depth estimation [15, 20, 21].

3. Proposed Algorithm

We train a neural network performing multiple tasks in a supervised manner. Let f be the network and θ be its parameters. We aim at determining optimal parameters

$$\theta^* = \arg\min_{\theta} \sum_{(I,J)\in\mathcal{D}} \ell(f(I;\theta), J) \tag{1}$$

where $\mathcal{D}$ is a training dataset composed of (I, J) pairs. I is an input signal and J represents the ground-truth labels for multiple tasks. Also, ℓ is a loss function between the estimate $\widehat{J} \triangleq f(I;\theta)$ and the ground-truth J.

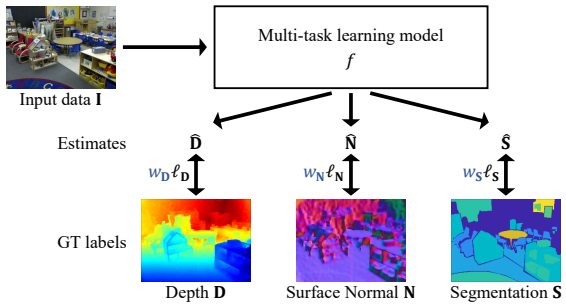


Figure 2. A schematic diagram of an MTL model for three tasks.

Figure 2 shows a schematic diagram of f for three tasks involving depth, normal, and segmentation losses. In general, let J_k, $\widehat{J}_k$, and ℓ_k denote a ground-truth label, its estimate, and a loss function for the kth task, respectively. The overall loss function ℓ in (1) is defined as a weighted sum of the loss functions for individual tasks, given by

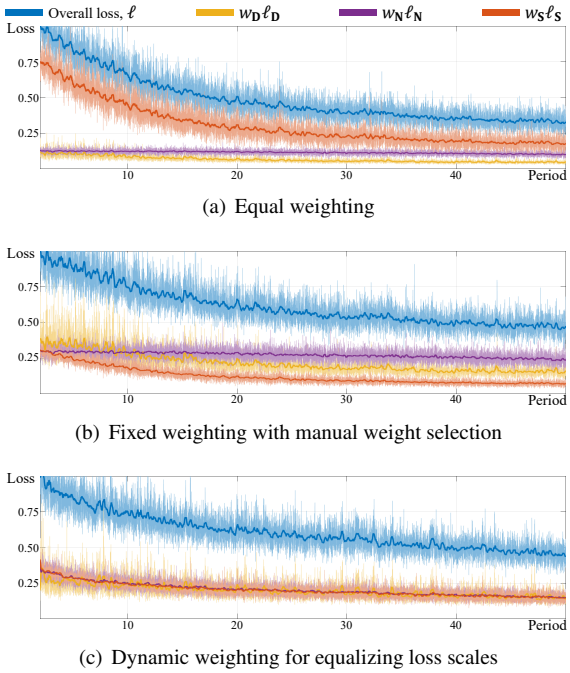
$$\ell(f(I;\theta),J) = \sum_{k=1}^{n} w_k \ell_k(\widehat{J}_k, J_k) \tag{2}$$

where w_k is a weight for the kth task, and n is the number of tasks. Either fixed or dynamic weighting can be adopted to determine w_k in MTL.

In fixed weighting, weights w_k are fixed throughout the training. The simplest scheme, called equal weighting, is to fix all weights identically. In such a case, the overall loss may be dominated by a specific loss, since each loss may have a different scale. Figure 3(a) shows an example of equal weighting, in which depth and normal errors may generate too small gradients as compared with dominant segmentation errors. This imbalance between losses can be reduced by setting weights manually as in Figure 3(b), so that the contribution of each loss is equalized at the beginning. However, it requires trial and error to find such weights. Besides, it cannot reflect loss variations during training. In Figure 3(b), the segmentation loss decreases faster than the others, resulting in an imbalance in the end.

Dynamic weighting can overcome these problems. By adjusting weights after each period, the contributions of losses can be equalized throughout the training, as in Figure 3(c). Let us define a *loss scale* as the product of a loss value and its weight. In Figure 3(c), the weights are determined adaptively to equalize the loss scales of all tasks.

We propose a sophisticated dynamic weighting algorithm, which adjusts weights to balance loss scales and to make the MTL network learn all tasks effectively. Notice that the proposed algorithm requires neither extra network parameters nor expert knowledge of individual loss functions. It simply monitors the progress of loss scales during training and adjusts weights periodically with only a slight computational overhead, so that the trained network can undertake all tasks more effectively. In this work, a training



(a) Equal weighting

(b) Fixed weighting with manual weight selection

(c) Dynamic weighting for equalizing loss scales

Figure 3. Loss function graphs of three weighting schemes.

epoch is defined as the period.

The proposed LSB algorithm has three phases. In the first period, without prior information about losses, weights are set manually. In the second period, loss scales are balanced using the previous loss record. From the third period onward, the difficulty of each task is evaluated using the loss record of the previous two periods. Then, the training focuses on difficult tasks.

Let $\{\pi_1, \pi_2, ..., \pi_n\}$ be the set of priority factors for losses. Specifically, the priority factor π_k represents the target contribution ratio of the kth loss function ℓ_k to the overall loss function ℓ. Thus, $\sum_{k=1}^{n} \pi_k = 1$. In applications, these priority factors can be provided explicitly. If there is no such provision, we set $\pi_k = \frac{1}{n}$ for every k. Also, let L_k^t be the average loss of ℓ_k over the tth period. Then, the average loss L^t of ℓ over period t is given by

$$L^t = \sum_{k=1}^{n} w_k^t L_k^t \tag{3}$$

where w_k^t is the weight for ℓ_k in period t. It is updated from w_k^{t-1} to w_k^t at the start of period t.

1st period: There is no record of previous losses. Thus, equal weighting can be adopted as in Figure 4(a). In general, we set the weights using the priority factors by

$$w_k^1 = \pi_k, \quad 1 \le k \le n. \tag{4}$$

2nd period: We attempt to make the loss scale of the kth task proportional to the priority factor π_k. Suppose that

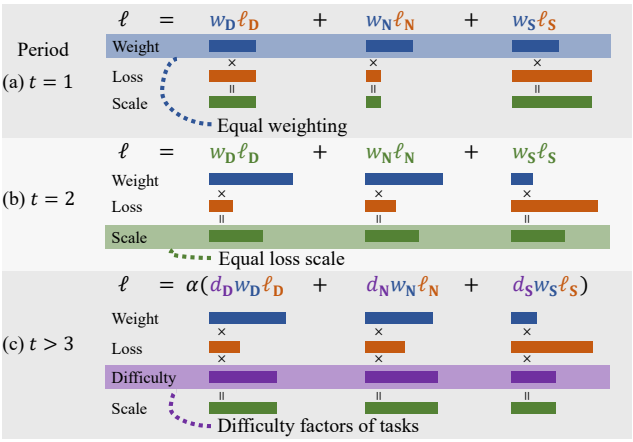


Figure 4. Three phases of the proposed LSB algorithm.

$\pi_k = \frac{1}{n}$ for every k. Then, as shown in Figure 4(b), the loss scales can be equalized by adjusting each weight to be inversely proportional to the corresponding loss. Notice that, at period t (≥ 2), the loss record of the previous period $t-1$ is available: L^{t-1} and L_k^{t-1} for $1 \leq k \leq n$.

At the start of period t, we aim to balance the loss scales $w_k^t L_k^t$ by adjusting the weights w_k^t. However, we do not know the losses L_k^t yet. To address this problem, we assume that

$$\frac{L^t}{L^{t-1}} = \frac{L_1^t}{L_1^{t-1}} = \frac{L_2^t}{L_2^{t-1}} = \cdots = \frac{L_n^t}{L_n^{t-1}}. \tag{5}$$

It is a reasonable assumption if all tasks are learned at a similar pace during period t. We seek to train the MTL network to satisfy this assumption as much as possible, and it holds to some extent if the period is short enough.

The loss scale $w_k^t L_k^t$ of task k should contribute to the overall loss L^t according to the priority factor π_k;

$$w_k^t L_k^t = \pi_k L^t. \tag{6}$$

The weight w_k^t is hence given by $w_k^t = \pi_k L^t / L_k^t$. By the assumption in (5), we have $L^t / L_k^t = L^{t-1} / L_k^{t-1}$. Thus, the proposed algorithm determines the weight w_k^t using the previous loss record by

$$w_k^t = \pi_k \frac{L^{t-1}}{L_k^{t-1}} \tag{7}$$

for $t \geq 2$ and $1 \leq k \leq n$.

The weighting rule in (7) has the following property.

Proposition 1. For $t \geq 2$, we have

$$\sum_{k=1}^{n} w_k^t L_k^{t-1} = \sum_{k=1}^{n} w_k^{t-1} L_k^{t-1}. \tag{8}$$

Proof. From (7), $w_k^t L_k^{t-1} = \pi_k L^{t-1}$. Since $\sum_{k=1}^n \pi_k = 1$, $\sum_{k=1}^n w_k^t L_k^{t-1} = L^{t-1}$. Then, (8) is established by (3). □

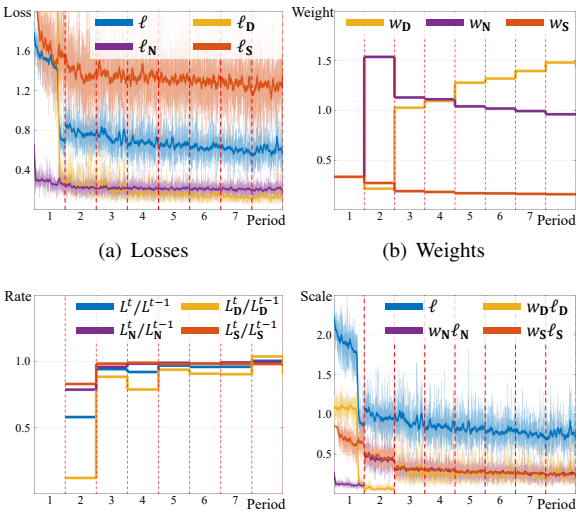


Figure 5. Illustration of the weighting rule in (7).

This proposition means that the overall loss L^{t-1} is unchanged when the weights w_k^{t-1} are updated to w_k^t. Thus, the update of the weights does not change the magnitude of the overall loss, which makes the monitoring of the overall losses over periods easier.

Figure 5 shows an example of the loss trends when the weighting rule in (7) is adopted. Three loss functions $\ell_{\mathbf{D}}$, $\ell_{\mathbf{N}}$, and $\ell_{\mathbf{S}}$ yield different values in Figure 5(a), but the corresponding loss scales are equalized after the first two periods in Figure 5(d) using the weights in Figure 5(b). Also, in Figure 5(c), loss reduction rates of the three tasks are similar to one another, as assumed in (5), except for the second period. Finally, after the first two periods, the overall loss ℓ in Figure 5(d) does not have big discontinuities between periods as predicted by *Proposition* 1.

We use the weighting rule in (7) for all $t \geq 2$. However, from the third period onward, more information is available about the characteristics of loss functions ℓ_k. To exploit such characteristics, we modify the overall loss function ℓ using the following scheme when $t \geq 3$.

3rd period onward: Using the loss record of the previous two periods, we quantify the difficulty of each task. Then, we focus on more difficult tasks by assigning bigger weights to them. Let d_k^t denote the difficulty factor for task k at period t, which is computed by

$$d_k^t = \left(\frac{L_k^{t-1} / L_k^{t-2}}{L^{t-1} / L^{t-2}} \right)^{\beta} \tag{9}$$

for $t \geq 3$ and $1 \leq k \leq n$. Here, β is a hyper-parameter. Suppose that $\beta = 1$. Then, d_k^t indicates whether the kth loss ℓ_k decreases faster or slower than the overall loss ℓ. If $d_k^t > 1$, ℓ_k decreases slower, implying that task k is difficult. On the contrary, $d_k^t < 1$ implies that task k is relatively easy.

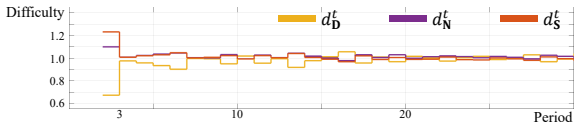


Figure 6. Trends of difficulty factors, when $\beta = 1$ in (9).

When $t \geq 3$, by incorporating the difficulty factors, we define the overall loss function ℓ as

$$\ell(f(I;\theta), J) = \alpha \sum_{k=1}^{n} d_k w_k \ell_k(\widehat{J}_k, J_k) \tag{10}$$

where α is a parameter to make the overall loss unchanged by the update of weights. Then, the average loss L^t of ℓ over period t is

$$L^t = \alpha^t \sum_{k=1}^{n} d_k^t w_k^t L_k^t \tag{11}$$

where α^t is the control parameter in period t, given by

$$\alpha^t = \frac{1}{\sum_{k=1}^{n} \pi_k d_k^t}. \tag{12}$$

Figure 4(c) illustrates this third phase.

Similar to *Proposition* 1, we have the following property.

Proposition 2. For $t \geq 3$, we have

$$\alpha^t \sum_{k=1}^{n} d_k^t w_k^t L_k^{t-1} = \alpha^{t-1} \sum_{k=1}^{n} d_k^{t-1} w_k^{t-1} L_k^{t-1} \tag{13}$$

Proof. From (7), $w_k^t L_k^{t-1} = \pi_k L^{t-1}$. Also, by (12),

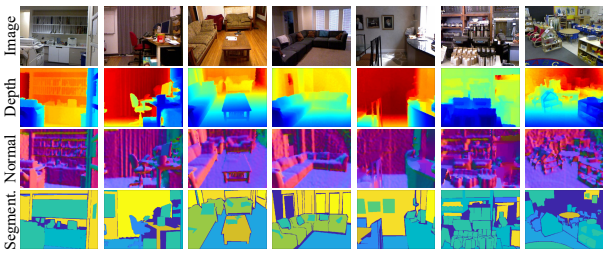
$$\alpha^t \sum_{k} d_k^t w_k^t L_k^{t-1} = \frac{\sum_k d_k^t \pi_k L^{t-1}}{\sum_{k=1}^{n} \pi_k d_k^t} = L^{t-1}. \tag{14}$$

The proposition is then established by (11). □

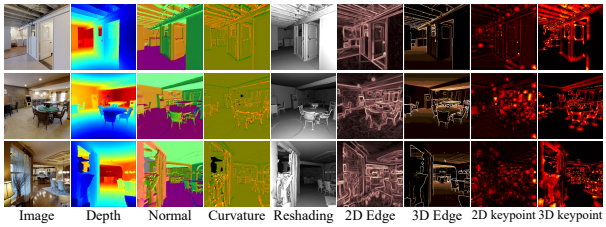
Difficulty factors d_k^t vary dynamically during training. Figure 6 illustrates difficulty trends during training when β in (9) is fixed to 1. Two observations can be made:

- Difficulty factors oscillate. A difficult task can become an easy one and vice versa, as the training goes on.
- The difficulty gaps between tasks are large at the start of training, but they get smaller gradually. Eventually, all difficulty levels become near 1.

Because of these two properties, even though the loss function in (10) is used instead of (2), each loss scale converges to be proportional to its priority factor as the training progresses. However, large variations of d_k in initial periods can affect the training adversely. Therefore, we adopt a curriculum learning scheme of smoothly increasing β in (9) according to t. Specifically, β is initialized to 0 and incremented gradually. Thus, initially, the loss function in (10) is reduced to (2), but difficulty factors play more important roles in later periods.



(a) NYUv2



(b) Taskonomy

Figure 7. Examples of images and labels in the (a) NYUv2 [42] and (b) Taskonomy [45] datasets.

Table 1. Summary of the network structures in experiments.

Full-sharing	An input image is processed by a single encoder and then a single decoder, which generates multi-channel output for each pixel representing the estimates of multiple tasks.
Multi-decoder	Multiple identically structured decoders are linked to an encoder backbone. Each decoder outputs an estimate for the corresponding task.
Multi-column	The cross-stich network [29] is used. It consists of multiple encoder-decoder columns for as many tasks and cross-stitch units connecting them.

4. Experimental Results

4.1. Dataset

The proposed LSB algorithm is applied to joint learning of multiple pixelwise prediction tasks. We test MTL networks, which process an RGB image $\mathbf{I} \in \mathbb{R}^{w \times h \times 3}$ to estimate multiple maps of spatial resolution $w \times h$. A different kind of map is generated according to the task, as illustrated in Figure 7. We employ two widely used datasets for MTL: the NYUv2 [42] and Taskonomy [45] datasets.

The NYUv2 dataset consists of 795 training and 654 test RGB images with labels for depth, normal, and segmentation. Using NYUv2, we train networks to learn the three tasks jointly from input images, resized to 384×288.

The Taskonomy dataset provides labels for a wide range of tasks. From the tiny split, we build a mini dataset of 2,762 training images and 548 test images with a $\frac{1}{100}$ sampling ratio. We test two combinations of tasks: 4-task for depth, normal, curvature, and reshading and 8-task for depth, normal, curvature, reshading, 2D edge, 3D edge, 2D keypoint, and 3D keypoint. Images are resized to 256×256.

Table 2. Performance comparison of the proposed algorithm with the conventional algorithms on the NYUv2 dataset using various architectures with the MobileNet.v2 backbone. In each test, the performance rank is provided within parentheses. The best average rank is boldfaced, while the second-best one is underlined.

	Depth		Normal		Segmentation		Rank		
	δ_1	RMSE	δ_{30°	$\angle^{\text{mean}}$	mIoU	Acc.	Min	Max	Avg.
(a) Single-task learning									
Depth	57.6%	0.824	-	-	-	-	-	-	-
Normal	-	-	43.2%	39.9	-	-	-	-	-
Segmentation	-	-	-	-	18.9%	54.3%	-	-	-
(b) Full-sharing									
Equal weighting	57.1% (3)	0.841 (3)	42.9% (4.5)	40.8 (4)	18.7% (4.5)	53.4% (5)	3	5	4.00
Uncert [18]	56.1% (4.5)	0.844 (4)	42.9% (4.5)	40.8 (4)	19.3% (3)	54.3% (3)	3	4.5	3.83
DWA [25]	57.7% (1)	0.825 (1)	43.0% (3)	40.8 (4)	18.7% (4.5)	54.0% (4)	1	4.5	2.92
GLS [6]	56.1% (4.5)	0.847 (5)	43.1% (2)	40.4 (1)	19.6% (2)	54.4% (2)	1	4.5	<u>2.75</u>
Proposed	57.6% (2)	0.829 (2)	43.3% (1)	40.5 (2)	19.8% (1)	54.7% (1)	1	2	**1.50**
(c) Multi-decoder									
Equal weighting	58.4% (4)	0.821 (5)	42.7% (3)	40.6 (1)	19.2% (3)	54.6% (1.5)	1	5	2.92
GradNorm [5]	57.0% (6)	0.829 (6)	41.6% (6)	40.9 (2.5)	20.5% (1)	53.3% (6)	1	6	4.58
Uncert [18]	58.1% (5)	0.813 (3)	42.0% (5)	41.7 (6)	19.0% (5)	53.8% (5)	3	6	4.83
DWA [25]	58.9% (3)	0.814 (4)	42.5% (4)	40.9 (2.5)	19.1% (4)	54.1% (4)	2.5	4	3.58
GLS [6]	59.6% (1)	0.797 (1)	42.8% (2)	41.1 (5)	18.7% (6)	54.3% (3)	1	6	<u>3.00</u>
Proposed	59.2% (2)	0.800 (2)	43.0% (1)	41.0 (4)	19.5% (2)	54.6% (1.5)	1	4	**2.25**
(d) Multi-column (cross-stitch [29])									
Equal weighting	59.5% (5)	0.801 (5)	44.3% (2.5)	40.0 (2)	19.9% (4)	55.7% (5)	2	5	3.92
Uncert [18]	60.9% (2)	0.775 (2)	44.1% (4)	40.1 (3)	20.8% (2)	56.3% (3)	2	4	<u>2.67</u>
DWA [25]	59.9% (4)	0.792 (4)	43.7% (5)	40.2 (4)	20.5% (3)	56.2% (2)	2	5	3.67
GLS [6]	60.6% (3)	0.778 (3)	44.3% (2.5)	40.5 (5)	19.6% (5)	55.9% (4)	2.5	5	3.75
Proposed	61.6% (1)	0.772 (1)	44.8% (1)	39.5 (1)	21.0% (1)	56.5% (1)	1	1	**1.00**

4.2. Implementation

As a weighting algorithm for multi-task losses, the proposed LSB algorithm can be applied to various MTL networks. Table 1 summarizes three architectures for verifying the effectiveness of the LSB algorithm. Each structure corresponds to a category in Figure 1. In all architectures, the encoders down-sample an input image 5 times and extract features of spatial resolution $\frac{w}{32} \times \frac{h}{32}$, while the decoders up-sample the encoder output 5 times. Thus, the output has the same resolution as the input. As the backbone networks, a lightweight network MobileNet.v2 [36] and a high capacity network PNASNet [24] are used, respectively.

In training, we adopt a single loss function for each task. For example, the depth loss is defined as the absolute difference between a logarithmic ground-truth depth and its estimate, and the normal loss is computed from the inner product of ground-truth and estimated normal vectors. More detailed implementation including network architecture, loss functions, and training details are specified in Sections 1, 2, and 3, respectively, of the supplemental document.

4.3. Performance Comparison

We compare the proposed LSB algorithm with conventional MTL algorithms: equal weighting, Uncert [18], DWA [25], GLS [6], and GradNorm [5]. These algorithms are recent ones, and most of them can be applied to various MTL architectures. Thus, the comparison is possible under the same experimental conditions. To verify the versatility and reliability of the proposed algorithm, we adopt the two datasets [42, 45], the three networks in Table 1 and the two encoder backbones [24, 36] and conduct experiments on their various combinations.

NYUv2: Table 2 summarizes the comparison results on NYUv2 using the MobileNet.v2 backbone. Six evaluation metrics (two for each task) in Table 3 are employed. GradNorm is tested for the multi-decoder architecture only, as done in [5]. Also, as a benchmark, we measure the performance of STL for each task.

There is a trade-off between multiple tasks in MTL. By focusing on a specific task only, an algorithm can easily outperform the other algorithms on that task, but it may perform poorly on the other tasks. Thus, it is hard to compare MTL algorithms using a single metric. It is also unreasonable to average the scores of multiple metrics, which represent different physical quantities. Therefore, we adopt a rank-based evaluation protocol [38]. In Table 2, we provide the rank of each algorithm in each metric within parentheses and present the average rank in the rightmost column.

The proposed algorithm consistently yields the best average ranks for all architectures. Also, the proposed algorithm ranks 1st or 2nd in most metrics. Especially, for the multi-column architecture, the proposed algorithm ranks 1st in all metrics and thus has the best average rank ($= 1.00$), which is significantly better than the second-best one ($= 2.67$) of Uncert. Also, it even outperforms the STL networks in all metrics. These results indicate that the proposed LSB algorithm is an effective approach to train MTL networks, without demanding additional network parameters or expert knowledge on loss functions. The average performances

Table 3. Evaluation metrics: d_i, n_i, and s_i are the ground-truth depth, normal vector, and segmentation class of pixel i, while $\widehat{d}_i$, $\widehat{n}_i$, and $\widehat{s}_i$ are their estimates. N is the number of pixels in an image.

Depth	δ_n	% of d_i such that $\max\left\{\frac{\widehat{d}_i}{d_i}, \frac{d_i}{\widehat{d}_i}\right\} < 1.25^n$
	RMSE	$\left(\frac{1}{N}\sum_i(\widehat{d}_i - d_i)^2\right)^{\frac{1}{2}}$
Normal	δ_{t°	% of n_i such that $\angle(\widehat{n}_i, n_i) < t^\circ$
	$\angle^{\text{mean}}$	Mean of $\angle(\widehat{n}_i, n_i)$
Segmentation	mIoU	Average IoU ratio over classes
	Accuracy	% of s_i such that $\widehat{s}_i = s_i$

Table 4. Performance comparison of the proposed algorithm with the conventional algorithms on the NYUv2 dataset using various architectures with the PNASNet backbone.

(a) Full-sharing

	Depth δ_1	Normal δ_{30°	Segmentation Acc.	Rank Avg.
Equal weighting	74.8% (3)	44.8% (4)	64.4% (2.5)	3.17
Uncert [18]	71.9% (5)	44.6% (5)	64.3% (4)	4.67
DWA [25]	73.3% (4)	44.9% (3)	64.4% (2.5)	3.17
GLS [6]	78.3% (1)	46.8% (2)	58.1% (5)	2.67
Proposed	77.5% (2)	50.5% (1)	65.0% (1)	**1.33**

(b) Multi-decoder

Equal weighting	77.6% (1.5)	51.8% (4)	65.6% (4)	3.17
Uncert [18]	77.1% (3)	53.7% (3)	65.6% (4)	3.33
DWA [25]	74.2% (4)	51.6% (5)	65.6% (4)	4.33
GLS [6]	73.0% (5)	57.2% (1)	66.3% (2)	2.67
Proposed	77.6% (1.5)	56.6% (2)	66.9% (1)	**1.50**

(c) Multi-column (cross-stitch [29])

Equal weighting	77.7% (5)	60.5% (5)	65.9% (2)	4.00
Uncert [18]	78.9% (3)	61.4% (3)	65.5% (4)	3.33
DWA [25]	78.6% (4)	61.5% (2)	65.1% (5)	3.67
GLS [6]	80.2% (1)	61.1% (4)	65.6% (3)	2.67
Proposed	79.0% (2)	64.7% (1)	66.4% (1)	**1.33**

over multiple trials are compared in Section 4 of the supplemental document, which confirm that the superiority of the proposed algorithm is statistically significant.

Table 4 summarizes the comparison results using the PNASNet encoders, which have a much higher learning capacity than the MobileNet.v2 encoders. Regardless of encoder backbones, the proposed algorithm consistently outperforms the conventional algorithms. Figure 8 qualitatively compares depth, normal, and segmentation results of the proposed algorithm with those of the equal weighting scheme. The proposed algorithm provides more accurate prediction results with smaller errors.

Taskonomy: Table 5 compares the results of 4-task learning and 8-task learning on the Taskonomy dataset using the multi-column structure with the MobileNet.v2 backbone. The evaluation protocols for these eight tasks are specified in Section 2 of the supplemental document. In both settings, the proposed algorithm yields the best average ranks 1.75 and 1.88. In the 8-task learning, the proposed algorithm provides the best results in four out of the eight tasks. Also, in terms of the 2D edge loss, the proposed algorithm yields 0.114, which is meaningfully lower than 0.119 of GLS.

Table 5. Performance comparison of 4-task learning and 8-task learning settings on the Taskonomy dataset using the multi-column architecture with the MobileNet.v2 backbone.

(a) 4-task learning

	Equal weighting	Uncert	DWA	GLS	Proposed
Depth	0.298 (5)	0.291 (3)	0.292 (4)	0.290 (2)	0.289 (1)
Normal	0.276 (1)	0.280 (4)	0.282 (5)	0.278 (3)	0.277 (2)
Curvature	0.267 (5)	0.264 (3)	0.265 (4)	0.263 (1.5)	0.263 (1.5)
Reshading	0.247 (5)	0.244 (4)	0.242 (2.5)	0.239 (1)	0.242 (2.5)
Rank Avg.	4.00	3.50	3.88	1.88	**1.75**

(b) 8-task learning

Depth	0.290 (4)	0.286 (1)	0.288 (3)	0.299 (5)	0.287 (2)
Normal	0.301 (3)	0.316 (5)	0.300 (2)	0.299 (1)	0.309 (4)
Curvature	0.269 (4)	0.274 (5)	0.268 (3)	0.262 (1)	0.267 (2)
Reshading	0.238 (2.5)	0.238 (2.5)	0.239 (4)	0.241 (5)	0.235 (1)
2D edge	0.145 (5)	0.135 (3)	0.144 (4)	0.119 (2)	0.114 (1)
3D edge	0.552 (1)	0.560 (5)	0.558 (4)	0.555 (2)	0.557 (3)
2D keypoint	0.126 (4)	0.121 (3)	0.129 (5)	0.103 (2)	0.101 (1)
3D keypoint	0.106 (5)	0.105 (3)	0.105 (3)	0.105 (3)	0.104 (1)
Rank Avg.	3.56	3.44	3.50	2.63	**1.88**

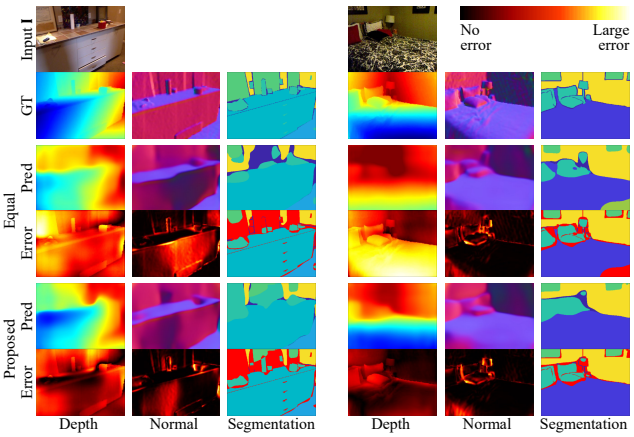


Figure 8. Qualitative comparison of the proposed algorithm with the equal weighting scheme on the NYUv2 dataset with the PNASNet backbone. For easier comparison, prediction error maps, as well as predicted results, are provided. In segmentation error maps, erroneously predicted regions are in red.

4.4. Analysis

Comparison of loss scale trends: Figure 9 compares the weight and loss scale trends on the NYUv2 dataset over periods. In equal weighting, the segmentation loss is dominant throughout the training. Although Uncert and DWA assign bigger weights to the depth and normal losses than the segmentation loss, the overall loss is still dominated by the segmentation loss. In contrast, the proposed algorithm equalizes the loss scales by adjusting the weights more effectively. Loss scale trends for more experiments are also available in Section 5 of the supplemental document.

Hyper-parameter β: In the default mode, we initialize the hyper-parameter β in (9) to 0 and increase it each period. Table 6 compares different settings of β. Note that a negative β makes the proposed algorithm focus on easy tasks,

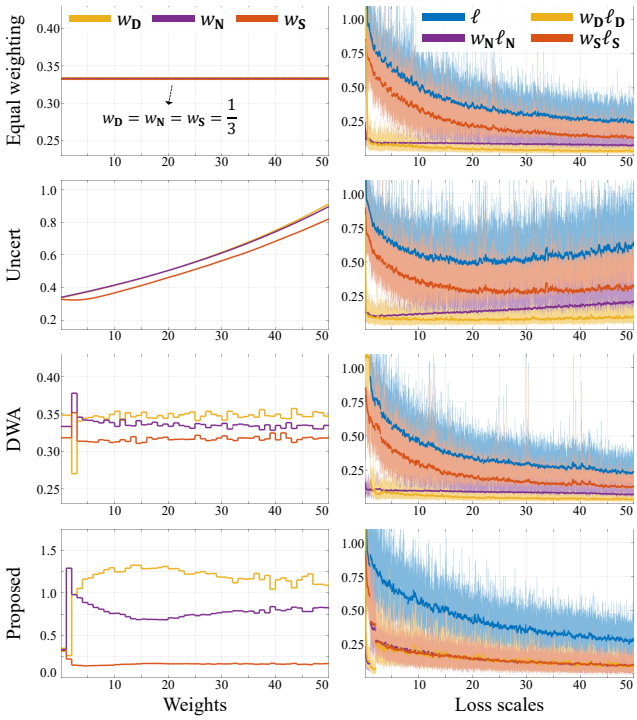


Figure 9. Illustration of weights and loss scales for equal weighting, Uncert [18], DWA [25], and the proposed algorithm.

Table 6. Comparison of various settings of β.

	Depth		Normal		Segmentation	
β	δ_1	RMSE	δ_{30°	$\angle^{\text{mean}}$	mIoU	Acc.
-3	76.3%	0.538	44.8%	36.1	25.0%	56.5%
-1	78.5%	0.519	45.3%	36.1	25.5%	56.4%
0	78.6%	0.528	46.6%	36.2	27.5%	58.4%
1	**79.2%**	**0.510**	48.0%	35.6	28.1%	61.6%
3	77.5%	0.539	47.6%	36.0	27.1%	60.3%
10	77.9%	0.537	45.1%	36.0	21.8%	53.5%
Default	77.5%	0.522	**50.5%**	**34.1**	**32.6%**	**65.0%**

while a positive β on difficult tasks. At $\beta = 0$, the difficulty is not taken into account. From the three settings $\beta = -1$, $\beta = 0$, and $\beta = 1$, it is observed that focusing on difficult tasks is more effective. However, as indicated by $\beta = 3$ and $\beta = 10$, too large a β degrades the results, since it causes loss scales to fluctuate excessively in early periods. Figure 10 shows difficulty trends in the default mode. At later periods, with bigger β's, difficulty factors experience larger variations. However, those factors help to balance loss scales more accurately in Figure 9.

Further analysis and qualitative results are provided in Sections 6, 7, and 8 of the supplemental document.

Non-pixelwise tasks: The proposed LSB algorithm can be applied to non-pixelwise tasks as well. If the design of appropriate network architecture is preceded, LSB can be applied to a wider range of tasks. We verify this extensibility by learning face landmark localization and face attribute classification using CelebA dataset [27]. Given a face image, the attribute classification performs binary classification of each of 40 different attributes, which is a non-pixelwise task. The multi-column architecture is adopted for training. For classification, convolutional layers of decoders are substituted with fully connected layers and sigmoid operations. Table 7 and Figure 11 show that LSB can be applied effectively to these tasks as well.

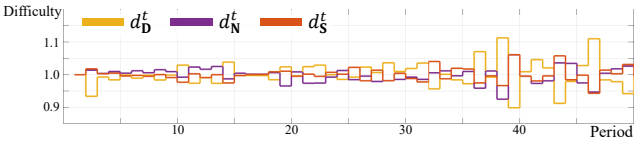


Figure 10. Trends of difficulty factors of the proposed algorithm, when β gradually increases starting from 0 in the default mode.

Table 7. Quantitative results on the CelebA dataset. Pixel distance and accuracy are used as the evaluation metrics for the localization and the classification, respectively. Compared with the equal weighting scheme, the proposed algorithm performs better in the localization, while also slightly better in the classification.

	Landmark localization			Attribute
	Eye	Nose	Mouth	classification
Equal weighting	0.859	2.044	1.290	90.6%
Proposed	**0.641**	**1.172**	**0.963**	**90.8%**

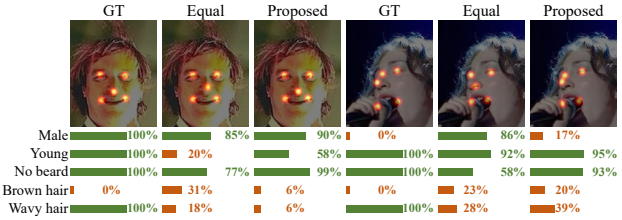


Figure 11. Qualitative comparison of the proposed algorithm with the equal weighting scheme on the CelebA dataset. The estimated landmark locations are highlighted, and some attribute classification results are provided.

5. Conclusions

We proposed the LSB algorithm for MTL that adjusts weights to make the network learn all tasks effectively. Instead of controlling the weight of each loss directly, the proposed algorithm balances loss scales periodically. Also, by assessing the difficulty factors of tasks in each period, the proposed algorithm focuses on learning more difficult tasks. Extensive experiments on various pixelwise vision tasks demonstrated that the proposed algorithm outperforms conventional algorithms regardless of architecture, dataset, and encoder backbone.

Acknowledgments

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (No. NRF-2018R1A2B3003896 and No. NRF-2021R1A4A1031864).

References

[1] R. K. Ando and T. Zhang. A framework for learning predictive structures from multiple tasks and unlabeled data. *J. Mach. Learn. Res.*, 6:1817–1853, Nov. 2005. 1

[2] A. Argyriou, T. Evgeniou, and M. Pontil. Multi-task feature learning. In *NIPS*, 2007. 1

[3] S. Ben-David and R. Schuller. Exploiting task relatedness for multiple task learning. In *Learning Theory and Kernel Machines*, 2003. 1

[4] R. Caruana. Multitask learning. *Machine Learning*, 28(1):41–75, 1997. 1

[5] Z. Chen, V. Badrinarayanan, C. Y. Lee, and A. Rabinovich. GradNorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *ICML*, 2018. 2, 6

[6] S. Chennupati, G. Sistu, S. Yogamani, and S. A Rawashdeh. MultiNet++: Multi-stream feature aggregation and geometric loss strategy for multi-task learning. In *CVPR Workshops*, 2019. 1, 2, 6, 7

[7] R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, 2008. 1, 2

[8] L. Deng, G. Hinton, and B. Kingsbury. New types of deep neural network learning for speech recognition and related applications: An overview. In *ICASSP*, 2013. 1

[9] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *ICCV*, 2017. 2

[10] D. Dong, H. Wu, W. He, D. Yu, and H. Wang. Multi-task learning for multiple language translation. In *Proc. ACL and IJCNLP*, 2015. 1

[11] T. Evgeniou and M. Pontil. Regularized multi-task learning. In *Proc. KDD*, 2004. 1

[12] C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. PathNet: Evolution channels gradient descent in super neural networks. *arXiv preprint arXiv:1701.08734*, 2017. 1, 2

[13] G. Gkioxari, B. Hariharan, R. Girshick, and J. Malik. R-CNNs for pose estimation and action detection. *arXiv preprint arXiv:1406.5212*, 2014. 2

[14] M. Guo, A. Haque, D. A. Huang, S. Yeung, and L. Fei-Fei. Dynamic task prioritization for multitask learning. In *ECCV*, 2018. 1, 2

[15] M. Heo, J. Lee, K.-R. Kim, and C.-S. Kim. Monocular depth estimation using whole strip masking and reliability-based refinement. In *ECCV*, 2018. 2

[16] J. T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *ICASSP*, 2013. 1, 2

[17] M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2016. 1

[18] A. Kendall, Y. Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *CVPR*, 2018. 1, 2, 6, 7, 8

[19] 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, 2

[20] J. H. Lee, M. Heo, K.-R. Kim, and C.-S. Kim. Single-image depth estimation based on fourier domain analysis. In *CVPR*, 2018. 2

[21] J. H. Lee and C.-S. Kim. Monocular depth estimation using relative depth maps. In *CVPR*, 2019. 2

[22] J.-H. Lee and C.-S. Kim. Multi-loss rebalancing algorithm for monocular depth estimation. In *ECCV*, 2020. 2

[23] C. Li, J. Yan, F. Wei, W. Dong, Q. Liu, and H. Zha. Self-paced multi-task learning. In *AAAI*, 2017. 2

[24] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L. J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy. Progressive neural architecture search. In *ECCV*, 2018. 6

[25] S. Liu, E. Johns, and A. J. Davison. End-to-end multi-task learning with attention. In *CVPR*, 2019. 1, 2, 6, 7, 8

[26] X. Liu, P. He, W. Chen, and J. Gao. Multi-task deep neural networks for natural language understanding. In *Proc. ACL*, 2019. 2

[27] Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *ICCV*, 2015. 8

[28] M. T. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. In *ICLR*, 2016. 1

[29] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, 2016. 1, 2, 5, 6, 7

[30] A. Pentina, V. Sharmanska, and C. H. Lampert. Curriculum learning of multiple tasks. In *CVPR*, 2015. 1, 2

[31] P. Rajeev R. Ranjan, V. M. Patel, and R. Chellappa. HyperFace: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(1):121–135, Dec. 2017. 2

[32] S. A. Rebuffi, H. Bilen, and A. Vedaldi. Learning multiple visual domains with residual adapters. In *NIPS*, 2017. 1

[33] S. Ruder, J. Bingel, I. Augenstein, and Anders A. Søgaard. Learning what to share between loosely related tasks. *arXiv preprint arXiv:1705.08142*, 2017. 2

[34] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard. Latent multi-task architecture learning. In *AAAI*, 2019. 1

[35] A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu, and R. Hadsell. Progressive neural networks. *arXiv preprint arXiv:1606.04671*, 2016. 1, 2

[36] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. C. Chen. MobileNetV2: Inverted residuals and linear bottlenecks. In *CVPR*, 2018. 6

[37] N. Sarafianos, T. Giannakopoulos, C. Nikou, and I. A. Kakadiaris. Curriculum learning for multi-task classification of visual attributes. In *ICCV Workshops*, 2017. 1, 2

[38] D. Scharstein and R. Szeliski. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vis.*, 47:7–42, Apr. 2002. 6

[39] M. L. Seltzer and J. Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *ICASSP*, 2013. 1, 2

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

[41] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. OverFeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013. 2

[42] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus. Indoor segmentation and support inference from RGBD images. In *ECCV*, 2012. 5, 6

[43] D. Xu, W. Ouyang, X. Wang, and N. Sebe. PAD-Net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. In *CVPR*, 2018. 2

[44] Z. Yin and J. Shi. GeoNet: Unsupervised learning of dense depth, optical flow and camera pose. In *CVPR*, 2018. 1

[45] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. In *CVPR*, 2018. 5, 6

[46] Z. Zhang, Z. Cui, and C. Xu. Pattern-affinitive propagation across depth, surface normal and semantic segmentation. In *CVPR*, 2019. 1

[47] Z. Zhang, P. Luo, C. C. Loy, and X. Tang. Facial landmark detection by deep multi-task learning. In *ECCV*, 2014. 2

[48] L. Zhou, Z. Cui, C. Xu, Z. Zhang, C. Wang, T. Zhang, and J. Yang. Pattern-structure diffusion for multi-task learning. In *CVPR*, 2020. 2