Maximum Roaming Multi-Task Learning

Lucas Pascal, Pietro Michiardi, Xavier Bost, Benoit Huet, Maria A Zuluaga

Maximum Roaming Multi-Task Learning

Lucas Pascal[1,2], **Pietro Michiardi**[1], **Xavier Bost**[2], **Benoit Huet**[3], **Maria A. Zuluaga**[1]
[1]EURECOM, France, {Pascal,Michiardi,Zuluaga}@eurecom.fr
[2]Orkis, France, Xbost@orkis.com
[3]Median Technologies, France, Benoit.Huet@mediantechnologies.com

Abstract

Multi-task learning has gained popularity due to the advantages it provides with respect to resource usage and performance. Nonetheless, the joint optimization of parameters with respect to multiple tasks remains an active research topic. Sub-partitioning the parameters between different tasks has proven to be an efficient way to relax the optimization constraints over the shared weights, may the partitions be disjoint or overlapping. However, one drawback of this approach is that it can weaken the inductive bias generally set up by the joint task optimization. In this work, we present a novel way to partition the parameter space without weakening the inductive bias. Specifically, we propose Maximum Roaming, a method inspired by dropout that randomly varies the parameter partitioning, while forcing them to visit as many tasks as possible at a regulated frequency, so that the network fully adapts to each update. We study the properties of our method through experiments on a variety of visual multi-task data sets. Experimental results suggest that the regularization brought by roaming has more impact on performance than usual partitioning optimization strategies. The overall method is flexible, easily applicable, provides superior regularization and consistently achieves improved performances compared to recent multi-task learning formulations.

1 Introduction

Multi-task learning (MTL) consists in jointly learning different tasks, rather than treating them individually, to improve generalization performance. This is done by jointly training tasks while using a shared representation (Caruana 1997). This approach has gained much popularity in recent years with the breakthrough of deep networks in many vision tasks. Deep networks are quite demanding in terms of data, memory and speed, thus making sharing strategies between tasks attractive.

MTL exploits the plurality of the domain-specific information contained in training signals issued from different related tasks. The plurality of signals serves as an inductive bias (Baxter 2000) and has a regularizing effect during training, similar to the one observed in transfer learning (Yosinski et al. 2014). This allows us to build task-specific models that generalize better within their specific domains. However, the plurality of tasks optimizing the same set of parameters can lead to cases where the improvement imposed by one task is to the detriment of another task. This phenomenon is called task interference, and can be explained by the fact that different tasks need a certain degree of specificity in their representation to avoid under-fitting.

To address this problem, several works have proposed to enlarge deep networks with task specific parameters (Gao et al. 2019; He et al. 2017; Kokkinos 2017; Liu, Johns, and Davison 2019; Lu et al. 2017; Misra et al. 2016; Mordan et al. 2018), giving tasks more room for specialization, and thus achieving better results. Other works adopt architectural adaptations to fit a specific set of tasks (Xu et al. 2018; Zhang, Wei, and Yang 2018; Zhang et al. 2019; Vandenhende, Georgoulis, and Van Gool 2020). These approaches, however, do not solve the problem of task interference in the shared portions of the networks. Furthermore, they generally do not scale well with the number of tasks. A more recent stream of works address task interference by constructing task-specific partitioning of the parameters (Bragman et al. 2019; Maninis, Radosavovic, and Kokkinos 2019; Strezoski, Noord, and Worring 2019), allowing a given parameter to be constrained by fewer tasks. As such, these methods sacrifice inductive bias to better handle the problem of task interference.

In this work, we introduce Maximum Roaming, a dynamic partitioning scheme that sequentially creates the inductive bias, while keeping task interference under control. Inspired by the dropout technique (Srivastava et al. 2014), our method allows each parameter to *roam* across several task-specific sub-networks, thus giving them the ability to learn from a *maximum* number of tasks and build representations more robust to variations in the input domain. It can therefore be considered as a regularization method in the context of multi-task learning. Differently from other recent partitioning methods that aim at optimizing (Bragman et al. 2019; Maninis, Radosavovic, and Kokkinos 2019) or fixing (Strezoski, Noord, and Worring 2019) a specific partitioning, ours privileges continuous random partition and assignment of parameters to tasks allowing them to learn from each task. Experimental results show consistent improvements over the state of the art methods.

The remaining of this document is organized as follows. Section 2 discusses related work. Section 3 sets out some preliminary elements and notations before the details of Max-

imum Roaming are presented in Section 4. Extensive experiments are conducted in Section 5 to, first, study the properties of the proposed method and to demonstrate its superior performance with respect to that one of other state-of-the-art MTL approaches. Finally, conclusions and perspectives are discussed in Section 6.

2 Related Work

Several prior works have pointed out the problems incurred by task interference in multi-task learning (Chen et al. 2018; Kendall, Gal, and Cipolla 2018; Liu, Johns, and Davison 2019; Maninis, Radosavovic, and Kokkinos 2019; Sener and Koltun 2018; Strezoski, Noord, and Worring 2019). We refer here to the three main categories of methods.

Loss weighting. A common countermeasure to task interference is to correctly balance the influence of the different task losses in the main optimization objective, usually a weighted sum of the different task losses. The goal is to prevent a task objective variations to be absorbed by some other tasks objectives of higher magnitude. In (Kendall, Gal, and Cipolla 2018) each task loss coefficient is expressed as a function of some task-dependent uncertainty to make them trainable. In (Liu, Johns, and Davison 2019) these coefficients are modulated considering the rate of loss change for each task. GradNorm (Chen et al. 2018) adjusts the weights to control the gradients norms with respect to the learning dynamics of the tasks. More recently, (Sinha et al. 2018) proposed a similar scheme using adversarial training. These methods, however, do not aim at addressing task interference, their main goal being to allow each task objective to have more or less magnitude in the main objective according to its learning dynamics. Maximum Roaming, instead, is explicitly designed to control task interference during optimization.

Multi-objective optimization. Other works have formulated multi-task learning as a multi-objective optimization problem. Under this formulation, (Sener and Koltun 2018) proposed MGDA-UB, a multi-gradient descent algorithm (Désidéri 2012) addressing task interference as the problem of optimizing multiple conflicting objectives. MGDA-UB learns a scaling factor for each task gradient to avoid conflicts. This has been extended by (Lin et al. 2019) to obtain a set of solutions with different trade-offs among tasks. These methods ensure, under reasonable assumptions, to converge into a Pareto optimal solution, from which no improvement is possible for one task without deteriorating another task. They keep the parameters in a fully shared configuration and try to determine a consensual update direction at every iteration, assuming that such consensual update direction exists. In cases with strongly interfering tasks, this can lead to stagnation of the parameters. Our method avoids this stagnation by reducing the amount of task interference, and by applying discrete updates in the parameters space, which ensures a broader exploration of this latter.

Parameter partitioning. Attention mechanisms are often used in vision tasks to make a network focus on different feature map regions (Liu, Johns, and Davison 2019). Recently, some works have shown that these mechanisms can be used at the convolutional filter level allowing each task to select, i.e. partition, a subset of parameters to use at every layer. The more the partitioning is selective, the less tasks are likely to use a given parameter, thus reducing task interference. This approach has also been used on top of pre-trained frozen networks, to better adapt the pre-trained representation to every single task (Mancini et al. 2018; Mallya, Davis, and Lazebnik 2018), but without joint parameter optimization. Authors in (Strezoski, Noord, and Worring 2019) randomly initialize hard binary tasks partitions with a hyper-parameter controlling their selectivity.(Bragman et al. 2019) sets task specific binary partitions along with a shared one, and trains them with the use of a Gumbel-Softmax distribution (Maddison, Mnih, and Teh 2017; Jang, Gu, and Poole 2017) to avoid the discontinuities created by binary assignments. Finally, (Maninis, Radosavovic, and Kokkinos 2019) uses task specific Squeeze and Excitation (SE) modules (Hu, Shen, and Sun 2018) to optimize soft parameter partitions. Despite the promising results, these methods may reduce the inductive bias usually produced by the plurality of tasks: (Strezoski, Noord, and Worring 2019) uses a rigid partitioning, assigning each parameter to a fixed subset of tasks, whereas (Bragman et al. 2019) and (Maninis, Radosavovic, and Kokkinos 2019) focus on obtaining an optimal partitioning, without taking into account the contribution of each task to the learning process of each parameter. Our work contributes to address this issue by pushing each parameter to learn sequentially from every task.

3 Preliminaries

Let us define a training set $\mathcal{T} = \{(\mathbf{x}_n, \mathbf{y}_{n,t})\}_{n\in[N],t\in[T]}$, where T is the number of tasks and N the number of data points. The set $\mathcal{T}$ is used to learn the T tasks with a standard shared convolutional network of depth D having the final prediction layer different for each task t. Under this setup, we refer to the convolutional filters of the network as *parameters*. We denote $S^{(d)}$ the number of parameters of the d^{th} layer and use $i \in \left\{1, \ldots, S^{(d)}\right\}$ to index them. Finally, $S_{max} = \max_d \left\{S^{(d)}\right\}$ represents the maximum number of parameters contained by a network layer.

In standard MTL, with fully shared parameters, the output of the d^{th} layer for task t is computed as:

$$f_t^{(d)}(H) = \sigma\left(H * K^{(d)}\right), \tag{1}$$

where $\sigma(.)$ is a non-linear function (e.g. ReLU), H a hidden input, and $K^{(d)}$ the convolutional kernel composed of the $S^{(d)}$ parameters of layer d.

Parameter Partitioning

Let us now introduce

$$\mathcal{M} = \left\{\left(\mathbf{m}_1^{(d)}, \ldots, \mathbf{m}_T^{(d)}\right)\right\}_{d\in[D]},$$

the binary parameter partitioning matrix, with $\mathbf{m}_t^{(d)} \in \{0,1\}^{S^{(d)}}$ a column vector associated to task t in the d^{th} layer, and $m_{i,t}^{(d)}$ an element on such vector associated to the i^{th} parameter. As $\mathcal{M}$ allows to select a subset of parameters for every t, the output of the d^{th} layer for task t (Eq. 1) is now computed as:

$$f_t^{(d)}(H_t) = \sigma\left(\left(H_t * K^{(d)}\right) \odot \mathbf{m}_t^{(d)}\right), \tag{2}$$

with $\odot$ the channel-wise product. This notation is consistent with the formalization of the dropout (e.g. (Gomez et al. 2019)). By introducing $\mathcal{M}$, the hidden inputs are now also task-dependent: each task requires an independent forward pass, like in (Maninis, Radosavovic, and Kokkinos 2019; Strezoski, Noord, and Worring 2019). In other words, given a training point $(\mathbf{x}_n, \{\mathbf{y}_{n,t}\}_{t=1}^T)$, for each task t we compute an independent forward pass $F_t(x) = f_t^{(D)} \circ ... \circ f_t^{(1)}(\mathbf{x})$ and then back-propagate the associated task-specific losses $\mathcal{L}_t(F_t(\mathbf{x}), \mathbf{y}_t)$. Each parameter i receives independent training gradient signals from the tasks using it, i.e. $m_{i,t}^{(d)} = 1$. If the parameter is not used, i.e. $m_{i,t}^{(d)} = 0$, the received training gradient signals from those tasks account to zero.

For the sake of simplicity in the notation and without loss of generality, in the remaining of this document we will omit the use of the index d to indicate a given layer.

Parameter Partitioning Initialization

Every element of $\mathcal{M}$ follows a Bernoulli distribution of parameter p:

$$P(m_{i,t} = 1) \sim \mathcal{B}(p).$$

We denote p the sharing ratio (Strezoski, Noord, and Worring 2019). We use the same value p for every layer of the network. The sharing ratio controls the overlap between task partitions, i.e. the number of different gradient signals a given parameter i will receive through training. Reducing the number of training gradient signals reduces task interference, by reducing the probability of having conflicting signals, and eases optimization. However, reducing the number of task gradient signals received by i also reduces the amount and the quality of inductive bias that different task gradient signals provide, which is one of the main motivations and benefits of multi-task learning (Caruana 1997).

To guarantee the full capacity use of the network, we impose

$$\sum_{t=1}^{T} m_{i,t} \geq 1. \tag{3}$$

Parameters not satisfying this constraint are attributed to a unique uniformly sampled task. The case $p = 0$, thus corresponds to a fully disjoint parameter partitioning, i.e. $\sum_{t=1}^{T} m_{i,t} = 1, \forall\, i$, whereas $p = 1$ is a fully shared network, i.e. $\sum_{t=1}^{T} m_{i,t} = T, \forall\, i$, equivalent to Eq. 1.

Following a strategy similar to dropout (Srivastava et al. 2014), which forces parameters to successively learn efficient representations in many different randomly sampled subnetworks, we aim to make every parameter i learn from every possible task by regularly updating the parameter partitioning $\mathcal{M}$, i.e. make parameters *roam* among tasks to sequentially build the inductive bias, while still taking advantage of the "simpler" optimization setup regulated by p. For this we introduce Maximum Roaming Multi-Task Learning, a learning strategy consisting of two core elements: 1) a parameter partitioning update plan that establishes how to introduce changes in $\mathcal{M}$, and 2) a parameter selection process to identify the elements of $\mathcal{M}$ to be modified.

4 Maximum Roaming Multi-Task Learning

In this section we formalize the core of our contribution. We start with an assumption that relaxes what can be considered as inductive bias.

Assumption 1. *The benefits of the inductive bias provided by the simultaneous optimization of parameters with respect to several tasks can be obtained by a sequential optimization with respect to different subgroups of these tasks.*

This assumption is in line with (Yosinski et al. 2014), where the authors state that initializing the parameters with transferred weights can improve generalization performance, and with other works showing the performance gain achieved by inductive transfer (see (He et al. 2017; Singh 1992; Tajbakhsh et al. 2016; Zamir et al. 2018)).

Assumption 1 allows to introduce the concept of evolution in time of the parameters partitioning $\mathcal{M}$, by indexing over time as $\mathcal{M}(c)$, where $c \in \mathbb{N}$ indexes update time-steps, and $\mathcal{M}(0)$ is the partitioning initialization from Section 3. At every step c, the values of $\mathcal{M}(c)$ are updated, under constraint (3), allowing parameters to roam across the different tasks.

Definition 1. *Let $A_t(c) = \{i \,|\, m_{i,t}(c) = 1\}$ be the set of parameter indices used by task t, at update step c, and $B_t(c) = \cup_{l=1}^{c} A_t(l)$ the set of parameter indices that have been visited by t, at least once, after c update steps. At step $c+1$, the binary parameter partitioning matrix $\mathcal{M}(c)$ is updated according to the following update rules:*

$$\begin{cases} \mathbf{m}_{i_-,t}(c+1) = 0, & i_- \in A_t(c) \\ \mathbf{m}_{i_+,t}(c+1) = 1, & i_+ \in \{1, ..., S\} \backslash B_t(c) \\ \mathbf{m}_{i,t}(c+1) = \mathbf{m}_{i,t}(c), & \forall\, i \notin \{i_-, i_+\} \end{cases} \tag{4}$$

with i_+ and i_- unique, uniformly sampled in their respective sets at each update step.

The frequency at which $\mathcal{M}(c)$ is updated is governed by Δ, where $c = \left\lfloor \frac{E}{\Delta} \right\rfloor$ and E denotes the training epochs. This allows parameters to learn from a fixed partitioning over Δ training iterations in a given partitioning configuration. Δ has to be significantly large (we express it in terms of training epochs), so the network can fully adapt to each new configuration, while a too low value could reintroduce more task interference by alternating too frequently different task signals on the parameters. Considering we apply discrete updates in the parameter space, which has an impact in model performance, we only update one parameter by update step to minimize the short-term impact.

Lemma 1. *Any update plan as in Def.1, with update frequency Δ has the following properties:*

1. *The update plan finishes in $\Delta(1-p)S_{max}$ training steps.*
2. *At completion, every parameter has been trained by each task for at least Δ training epochs.*
3. *The number of parameters attributed to each task remains constant over the whole duration of update plan.*

Proof: The first property comes from the fact that $B_t(c)$grows by 1 at every step c, until all possible parameters in a given layer d are included, thus no new i_+ can be

sampled. At initialization, $|B_t(c)| = pS$, and it increases by one every Δ training iterations, which gives the indicated result, upper bounded by the layer containing the most parameters. The proof of the second property is straightforward, since each new parameter partition remains frozen for at least Δ training epochs. The third property is also straightforward since every update consists in the exchange of parameters i_- and i_+ □

Definition 1 requires to select update candidate parameters i_+ and i_- from their respective subsets (Eq 4). We select both i_+, i_- under a uniform distribution (without replacement), a lightweight solution to guarantee a constant overlap between the parameter partitions of the different tasks.

Lemma 2. *The overlap between parameter partitions of different tasks remains constant, on average, when the candidate parameters i_- and i_+, at every update step $c + 1$, are sampled without replacement under a uniform distribution from $A_t(c)$ and $\{1, ..., S\} \backslash B_t(c)$, respectively.*

Proof: We prove by induction that $P(m_{i,t}(c) = 1)$ remains constant over c, i and t, which ensures a constant overlap between the parameter partitions of the different tasks. The detailed proof is provided in Appendix A □

We now formulate the probability of a parameter i to have been used by task t, after c update steps as:

$$P(i \in B_t(c)) = p + (1 - p)\, r(c), \tag{5}$$

where

$$r(c) = \left(\frac{c}{(1-p)S} \right), \quad c \leq (1-p)S \tag{6}$$

is the *update ratio*, which indicates the completion rate of the update process within a layer. The condition $c \leq (1 - p)S$ refers to the fact that there cannot be more updates than the number of available parameters. It is also a necessary condition for $P(i \in B_t(c)) \in [0, 1]$. The increase of this probability represents the increase in the number of visited tasks for a given parameter, which is what creates inductive bias, following Assumption 1.

We formalize the benefits of Maximum Roaming in the following theorem:

Proposition 1. *Starting from a random binary parameter partitioning $\mathcal{M}(0)$ controlled by the sharing ratio p, Maximum Roaming maximizes the inductive bias across tasks, while controlling task interference.*

Proof: Under Assumption 1, the inductive bias is correlated to the averaged number of tasks having optimized any given the parameter, which is expressed by Eq. 5. $P(i \in B_t(c))$ is maximized with the increase of the number of updates c, to compensate the initial loss imposed by $p \leq 1$. The control over task interference cases is guaranteed by Lemma 2 □

5 Experiments

This section first describes the datasets and the baselines used for comparison. We then evaluate the presented Maximum Roaming MTL method on several problems. First we study its properties such as the effects the sharing ratio p, the impact of the interval between two updates Δ and the completion rate of the update process $r(c)$ and the importance of having a random selection process of parameters for update. Finally, we present a benchmark comparing MR with the different baseline methods. All code, data and experiments are available at https://github.com/lucaspascal/Maximum-Roaming-Mutli-Task-Learning.

Datasets

We use three publicly available datasets in our experiments:

Celeb-A. We use the official release, which consists of more than 200k celebrities images, annotated with 40 different facial attributes. To reduce the computational burden and allow for faster experimentation, we cast it into a multi-task problem by grouping the 40 attributes into eight groups of spatially or semantically related attributes (*e.g.* eyes attributes, hair attributes, accessories..) and creating one attribute prediction task for each group. Details on the pre-processing procedure are provided in Appendix B.

CityScapes. The Cityscapes dataset (Cordts et al. 2016) contains 5000 annotated street-view images with pixel-level annotations from a car point of view. We consider the seven main semantic segmentation tasks, along with a depth-estimation regression task, for a total of 8 tasks.

NYUv2. The NYUv2 dataset (Silberman et al. 2012) is a challenging dataset containing 1449 indoor images recorded over 464 different scenes from Microsoft Kinect camera. It provides 13 semantic segmentation tasks, depth estimation and surfaces normals estimation tasks, for a total of 15 tasks. As with CityScapes, we use the pre-processed data provided by (Liu, Johns, and Davison 2019).

Baselines

We compare our method with several alternatives, including two parameter partitioning approaches (Maninis, Radosavovic, and Kokkinos 2019; Strezoski, Noord, and Worring 2019). Among these, we have not included (Bragman et al. 2019) as we were not able to correctly replicate the method with the available resources. Specifically, we evaluate: **i) MTL**, a standard fully shared network with uniform task weighting; **ii) GradNorm** (Chen et al. 2018), a fully shared network with trainable task weighting method ; **iii) MGDA-UB** (Sener and Koltun 2018), a fully shared network which formulates the MTL as a multi-objective optimization problem; **iv) Task Routing (TR)** (Strezoski, Noord, and Worring 2019), a parameter partitioning method with fixed binary masks; **v) SE-MTL** (Maninis, Radosavovic, and Kokkinos 2019) a parameters partitioning method, with trainable real-valued masks; and **vi) STL**, the single-task learning baselines, using one model per task.

Note that SE-MTL (Maninis, Radosavovic, and Kokkinos 2019) consists of a more complex framework which comprises several other contributions. For a fair comparison with the other baselines, we only consider the parameter partitioning and not the other elements of their work

Facial attributes detection

In these first experiments, we study in detail the properties of our method using the Celeb-A dataset (Liu et al. 2015). Being a small dataset it allows for fast experimentation.

For the sake of fairness in comparison, all methods use the same network, a ResNet-18 (He et al. 2016), as the backbone. All models are optimized with Adam optimizer (Kingma and Ba 2017) with learning rate $10\mathrm{e}{-4}$. The reported results are averaged over five seeds.

Effect of Roaming. In a first experiment, we study the effects of the roaming imposed to parameters in MTL performance as a function of the sharing ratio p and compare these with a fixed partitioning setup. Figure 1 reports achieved F-scores as p varies, with $\Delta = 0.1$ and $r(c) = 100\%$. Let us remark that as all models scores are averaged over 5 seeds, this means that the fixed partitioning scores are the average of 5 different (fixed) partitionings.

Results show that for the same network capacity Maximum Roaming provides improved performance w.r.t. a fixed partitioning approach. Moreover, as the values of p are smaller, and for the same network capacity, Maximum Roaming does not suffer from a dramatic drop in performance as it occurs using a fixed partitioning. This behaviour suggests that parameter partitioning does have an unwanted effect on the inductive bias that is, thus, reflected in poorer generalization performance. However, these negative effects can be compensated by parameter roaming across tasks.

The fixed partitioning scheme (blue bars) achieves its best performance at $p = 0.9$ (F-score$= 0.6552$). This is explained by the fact that the dataset is not originally made for multi-task learning: all its classes are closely related, so they naturally have a lot to share with few task interference. Maximum Roaming achieves higher performance than this nearly full shared configuration (the overlap between task partitions is close to its maximum) for every p in the range $[0.3, 0.9]$. In this range, the smaller p is, the greater the gain in performance: it can be profitable to partially separate tasks even when they are very similar (i.e. multi-class, multi-attribute datasets) while allowing parameters to roam.

Effect of Δ and $r(c)$. Here we study the impact of the interval between two updates Δ and the completion rate of the update process $r(c)$ (Eq. 6). Using a fixed sharing ratio, $p = 0.5$, we report the obtained F-score values of our method over a grid search over these two hyper-parameters in Figure 1(center).

Results show that the models performance increase with Δ. Improved performance in terms if the F-score can observed for $\Delta \geq 0.05$ epochs. This suggests that Δ needs to be large enough so that the network can correctly adapt its weights to the changing configuration. A rough knowledge of the overall learning behaviour on the training dataset or a coarse grid search is enough to set it. Regarding the completion percentage, as it would be expected, the F-score increases with r. The performance improvement becomes substantial beyond $r = 25\%$. Furthermore, the performance saturates after 75%, suggesting that, at this point, the parameters have built robust representations, thus additional parameter roaming does not contribute to further improvements.

Role of random selection. Finally, we assess the importance of choosing candidate parameters for updates under a uniform distribution. To this end, we here define a deterministic selection process to systematically choose i_- and i_+ within the update plan of Def. 1. New candidate parameters are selected to minimize the average cosine similarity in the task parameter partition. The intuition behind this update plan is to select parameters which are the most likely to provide additional information for a task, while discarding the more redundant ones based on their weights. The candidate parameters i_- and i_+ are thus respectively selected such that:

$$i_- = \arg\min_{u \in A_t(c)} \left(\sum_{v \in (A_t(c) \setminus \{u\})} \frac{K_u \cdot K_v}{||K_u|| ||K_v||} \right)$$
$$i_+ = \arg\max_{u \in \{1,..,S\} \setminus B_t(c)} \left(\sum_{v \in A_t(c)} \frac{K_u \cdot K_v}{||K_u|| ||K_v||} \right)$$

with K_u, K_v the parameters u, v of the convolutional kernel K. Figure 1 (right) compares this deterministic selection process with Maximum Roaming by reporting the best F-scores achieved by the fully converged models for different completion rates $r(c)$ of the update process.

Results show that, while both selection methods perform about equally at low values of r, Maximum Roaming progressively improves as r grows. We attribute this to the varying overlapping induced by the deterministic selection. With a deterministic selection, outliers in the parameter space have more chances than others to be quickly selected as update candidates, which slightly favours a specific update order, common to every task. This has the effect of increasing the overlap between the different task partitions, along with the cases of task interference.

It should be noted that the deterministic selection method still provides a significant improvement compared to a fixed partitioning ($r = 0$). This highlights the primary importance of making the parameters learn from a maximum number of tasks, which is guaranteed by the update plan (Def. 1), i.e. the *roaming*, used by both selection methods.

Benchmark. Finally, we benchmark of our method with the different baselines. We report precision, recall and f-score metrics averaged over the 40 facial attributes, along with the average ranking of each MTL model over the reported performance measures; and the ratio #P of trainable parameters w.r.t. the MTL baseline (Table 1). The partitioning methods (TR, SE-MTL and MR) achieve the three best results, and our method perform substantially better than the two others.

Scene understanding

This experiment compares the performance of MR with the baseline methods in two well-established scene-understanding benchmarks: CityScapes (Cordts et al. 2016) and NYUv2 (Silberman et al. 2012).

For the sake of this study, we consider each segmentation task as an independent task, although it is a common approach to consider all of them as a unique segmentation task. As with the Celeb dataset, for the sake of fairness in

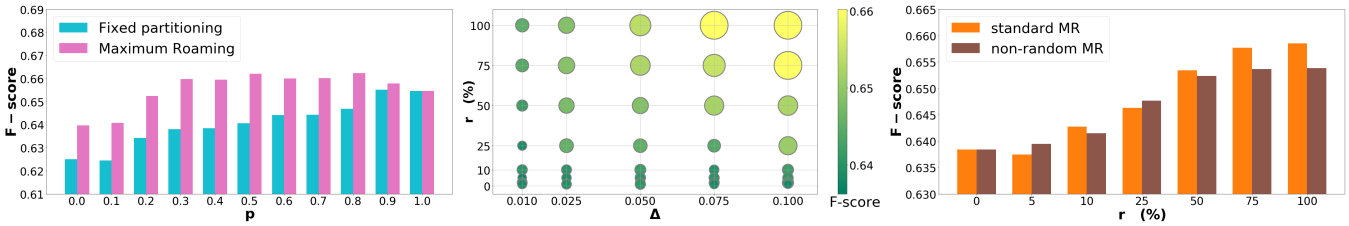

Figure 1: **(left)** Contribution of Maximum Roaming depending on the parameter partitioning selectivity p. **(middle)** F-score of our method reported for different values of the update interval Δ and the update completion rate r. **(right)** Comparison of Maximum Roaming with random and non-random selection process of parameter candidates for updates.

		Multi-Attribute Classification			
	#P	Precision	Recall	F-Score	Avg. Rank
STL	7.9	67.10 ± 0.37	61.99 ± 0.49	64.07 ± 0.21	-
MTL	1.0	68.67 ± 0.69	59.54 ± 0.52	62.95 ± 0.21	5.33
GradNorm ($\alpha = 0.5$)	1.0	70.36 ± 0.07	59.49 ± 0.58	63.55 ± 0.49	5.00
MGDA-UB	1.0	68.64 ± 0.12	60.21 ± 0.33	63.56 ± 0.27	4.66
SE-MTL	1.1	71.10 ± 0.28	62.64 ± 0.51	65.85 ± 0.17	2.33
TR ($p = 0.9$)	1.0	$\underline{71.71 \pm 0.06}$	61.75 ± 0.47	65.51 ± 0.32	2.33
MR ($p = 0.8$)	1.0	71.24 ± 0.35	$\underline{63.04 \pm 0.56}$	$\underline{66.23 \pm 0.20}$	$\underline{1.33}$

Table 1: Celeb-A results (Average over 40 facial attributes). The best per column score of an MTL method is underlined.

comparison, all approaches use the same base network. We use a SegNet (Badrinarayanan, Kendall, and Cipolla 2017), split after the last convolution, with independent outputs for each task, on top of which we build the different methods to compare. All models are trained with Adam (learning rate of $10e{-}4$). We report Intersection over Union (mIoU) and pixel accuracy (Pix. Acc.) averaged over all segmentation tasks, average absolute (Abs. Err.) and relative error (Rel. Err.) for depth estimation tasks, mean (Mean Err.) and median errors (Med. Err.) for the normals estimation task, the ratio #P of trainable parameters w.r.t. MTL, and the average rank of the MTL methods over the measures. STL is not included in the ranking, as we consider it of a different nature, but reported as a baseline reference.

Tables 2 and 3 report the results on CityScapes and NYUv2, respectively. The averange rankings for NYUv2 are presented in Table 4. The reported results are the best achieved with each method on the validation set, averaged over 3 seeds, after a grid-search on the hyper-parameters.

Maximum Roaming reaches the best scores on segmentation and normals estimation tasks, and ranks second on depth estimation tasks. In particular, it outperforms other methods on the segmentation tasks: our method restores the inductive bias decreased by parameter partitioning, so the tasks benefiting the most from it are the ones most similar to each other, which are here the segmentation tasks. Furthermore, MR uses the same number of trainable weights than the MTL baseline, plus a few binary partitions masks (negligible), which means it scales almost optimally to the number of tasks. This is also the case for the other presented baselines, which sets them apart from heavier models in the literature, which add task-specific branches in their networks to improve performance at the cost of scalability.

Regarding other MTL baselines, we first observe that GradNorm (Chen et al. 2018) fails on the regression tasks (depth and normals estimation). This is due to the equalization of the task respective gradient magnitudes. Specifically, since the multi-class segmentation task is divided into independent segmentation tasks (7 for CityScapes and 13 for NYUv2), GradNorm attributes to the depth estimation task of CityScapes only one eighth of the total gradient magnitude, which gives it a systematically low importance compared to the segmentation tasks which are more likely to agree on a common gradient direction, thus diminishing the depth estimation task. Instead in MTL the gradient's magnitude is not constrained, having more or less importance depending on the loss obtained on a given task, which explains why the regression tasks are better handled by this simpler model in this configuration. For instance, in a configuration with the CityScape segmentation classes addressed as one task (for 2 tasks in total), GradNorm keeps its good sgementation performance (mIoU: 56.83 ± 0.09, Pix. Acc.: 97.37 ± 0.01) and improves at regression tasks (Abs.Err.: 0.0157 ± 0.0001, Rel.Err: 36.92 ± 2.22), thus confirming our hypothesis. We also observe that MGDA-UB (Sener and Koltun 2018) reaches pretty low performance on the NYUv2 dataset, especially on segmentation tasks, while being one of the best performing ones on CityScapes. It appears that during training, the loss computed for the shared weights quickly converges to zero, leaving task-specific prediction layers to learn their task independently from an almost frozen shared representation. This could also explain why it still achieves good results at the regression tasks, these being easier tasks. We hypothesize that the solver fails at finding good directions improving all tasks, leaving the model stuck

		Segmentation		Depth estimation		
		(Higher Better)		(Lower Better)		
	#P	mIoU	Pix. Acc.	Abs. Err.	Rel. Err.	Avg. Rank
STL	7.9	58.57 ± 0.49	97.46 ± 0.03	0.0141 ± 0.0002	22.59 ± 1.15	-
MTL	1.0	56.57 ± 0.22	97.36 ± 0.02	0.0170 ± 0.0006	43.99 ± 5.53	3.75
GradNorm ($\alpha = 0.6$)	1.0	56.77 ± 0.08	$\underline{97.37 \pm 0.02}$	0.0199 ± 0.0004	68.13 ± 4.48	3.87
MGDA-UB	1.0	56.19 ± 0.24	97.33 ± 0.01	$\underline{0.0130 \pm 0.0001}$	$\underline{25.09 \pm 0.28}$	2.50
SE-MTL	1.1	55.45 ± 1.03	97.24 ± 0.10	0.0160 ± 0.0006	35.72 ± 1.62	4.87
TR ($p = 0.6$)	1.0	56.52 ± 0.41	97.24 ± 0.04	0.0155 ± 0.0003	31.47 ± 0.55	3.87
MR ($p = 0.6$)	1.0	$\underline{57.93 \pm 0.20}$	$\underline{97.37 \pm 0.02}$	0.0143 ± 0.0001	29.38 ± 1.66	$\underline{1.62}$

Table 2: Cityscape results. The best per column score of an MTL method is underlined.

		Segmentation		Depth estimation		Normals estimation	
		(Higher Better)		(Lower Better)		(Lower Better)	
	#P	mIoU	Pix. Acc.	Abs. Err.	Rel. Err.	Mean Err.	Med. Err.
STL	14.9	13.12 ± 1.06	94.58 ± 0.14	67.46 ± 2.64	28.79 ± 1.18	29.77 ± 0.22	23.93 ± 0.15
MTL	1.0	15.98 ± 0.56	94.22 ± 0.25	60.95 ± 0.41	$\underline{25.54 \pm 0.07}$	32.43 ± 0.19	27.43 ± 0.35
GradNorm	1.0	16.13 ± 0.23	94.43 ± 0.07	76.26 ± 0.34	32.08 ± 0.50	34.45 ± 0.52	30.98 ± 0.80
MGDA-UB	1.0	2.96 ± 0.35	82.87 ± 0.23	186.95 ± 15.33	98.74 ± 5.34	46.96 ± 0.37	45.15 ± 0.70
SE-MTL	1.2	16.02 ± 0.12	94.56 ± 0.01	$\underline{59.88 \pm 1.12}$	26.30 ± 0.58	32.22 ± 0.02	26.12 ± 0.02
TR ($p = 0.8$)	1.0	16.54 ± 0.02	94.58 ± 0.11	63.54 ± 0.85	27.86 ± 0.90	30.93 ± 0.19	25.51 ± 0.28
MR ($p = 0.8$)	1.0	$\underline{17.40 \pm 0.31}$	$\underline{94.86 \pm 0.06}$	60.82 ± 0.23	27.50 ± 0.15	$\underline{30.58 \pm 0.04}$	$\underline{24.67 \pm 0.08}$

Table 3: NYUv2 results. The best per column score of an MTL method is underlined.

	Avg. Rank
MTL	3.66
GradNorm	4.50
MGDA-UB	6.00
SE-MTL	2.66
TR ($p = 0.8$)	2.66
MR ($p = 0.8$)	$\underline{1.50}$

Table 4: NYUv2 results

in a Pareto-stationary point.

When comparing to the single task learners counterpart, we observe that on CityScapes STL achieves slightly better segmentation performances than the other approaches, and competitive results on depth estimation. On NYUv2 (and Celeb-A), its results are far from the best MTL models. These shows that complex setups proposing numerous tasks, as in our setup (8, 8 and 15), are challenging for the different MTL baselines, resulting in losses in performance as the number of tasks increase. This is not a problem with STL, which uses an independent model for each task. However, the associated increase in terms of training time and parameters (15 more parameters for NYUv2, which is equivalent to 375M parameters) makes it inefficient in practice, while its results are not even guaranteed to be better than the multi-task approaches, as demonstrated by the obtained results.

6 Conclusion

In this paper, we introduced Maximum Roaming, a dynamic parameter partitioning method that reduces the task interference phenomenon while taking full advantage of the latent inductive bias represented by the plurality of tasks. Our approach makes each parameter learn successively from all possible tasks, with a simple yet effective parameter selection process. The proposed algorithm achieves it in a minimal time, without additional costs compared to other partitioning methods, nor additional parameter to be trained on top of the base network. Experimental results show a substantially improved performance on all reported datasets, regardless of the type of convolutional network it applies on, which suggests this work could form a basis for the optimization of the shared parameters of future Multi-Task Learning works.

Maximum Roaming relies on a binary partitioning scheme that is applied at every layer independently of the layer's depth. However, it is well-known that the parameters in the lower layers of deep networks are generally less subject to task interference. Furthermore, it fixes an update interval, and show that the update process can in some cases be stopped prematurely. We encourage any future work to apply Maximum Roaming or similar strategies to more complex partitioning methods, and to allow the different hyper-parameters to be automatically tuned during training. As an example, one could eventually find a way to include a term favoring *roaming* within the loss of the network.

References

Badrinarayanan, V.; Kendall, A.; and Cipolla, R. 2017. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(12): 2481–2495.

Baxter, J. 2000. A Model of Inductive Bias Learning. *Journal of Artificial Intelligence Research* 12: 149–198.

Bragman, F. J.; Tanno, R.; Ourselin, S.; Alexander, D. C.; and Cardoso, J. 2019. Stochastic Filter Groups for Multi-Task CNNs: Learning Specialist and Generalist Convolution Kernels. In *The IEEE International Conference on Computer Vision (ICCV)*, 1385–1394.

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

Chen, Z.; Badrinarayanan, V.; Lee, C.-Y.; and Rabinovich, A. 2018. GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80, 794–803.

Cordts, M.; Omran, M.; Ramos, S.; Rehfeld, T.; Enzweiler, M.; Benenson, R.; Franke, U.; Roth, S.; and Schiele, B. 2016. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Désidéri, J.-A. 2012. Multiple-gradient Descent Algorithm (MGDA) for Multiobjective Optimization. *Comptes Rendus Mathematique* 350(5): 313–318.

Gao, Y.; Ma, J.; Zhao, M.; Liu, W.; and Yuille, A. L. 2019. NDDR-CNN: Layerwise Feature Fusing in Multi-Task CNNs by Neural Discriminative Dimensionality Reduction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3200–3209.

Gomez, A. N.; Zhang, I.; Kamalakara, S. R.; Madaan, D.; Swersky, K.; Gal, Y.; and Hinton, G. E. 2019. Learning Sparse Networks Using Targeted Dropout. *arXiv:1905.13678 [cs, stat]* URL http://arxiv.org/abs/1905.13678. ArXiv: 1905.13678.

He, K.; Gkioxari, G.; Dollar, P.; and Girshick, R. 2017. Mask R-CNN. In *The IEEE International Conference on Computer Vision (ICCV)*, 2961–2969.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep Residual Learning for Image Recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 770–778.

Hu, J.; Shen, L.; and Sun, G. 2018. Squeeze-and-Excitation Networks. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7132–7141.

Jang, E.; Gu, S.; and Poole, B. 2017. Categorical Reparameterization with Gumbel-Softmax. *arXiv:1611.01144 [cs, stat]* URL http://arxiv.org/abs/1611.01144. ArXiv: 1611.01144.

Kendall, A.; Gal, Y.; and Cipolla, R. 2018. Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 7482–7491.

Kingma, D. P.; and Ba, J. 2017. Adam: A Method for Stochastic Optimization. *arXiv:1412.6980 [cs]* URL http://arxiv.org/abs/1412.6980. ArXiv: 1412.6980.

Kokkinos, I. 2017. Ubernet: Training a Universal Convolutional Neural Network for Low-, Mid-, and High-Level Vision Using Diverse Datasets and Limited Memory. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 6129–6138.

Lin, X.; Zhen, H.-L.; Li, Z.; Zhang, Q.-F.; and Kwong, S. 2019. Pareto Multi-Task Learning. In *Advances in Neural Information Processing Systems 32*, 12060–12070.

Liu, S.; Johns, E.; and Davison, A. J. 2019. End-To-End Multi-Task Learning With Attention. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1871–1880.

Liu, Z.; Luo, P.; Wang, X.; and Tang, X. 2015. Deep Learning Face Attributes in the Wild. In *The IEEE International Conference on Computer Vision (ICCV)*, 3730–3738.

Lu, Y.; Kumar, A.; Zhai, S.; Cheng, Y.; Javidi, T.; and Feris, R. 2017. Fully-Adaptive Feature Sharing in Multi-Task Networks With Applications in Person Attribute Classification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 5334–5343.

Maddison, C. J.; Mnih, A.; and Teh, Y. W. 2017. The Concrete Distribution: A Continuous Relaxation of Discrete Random Variables. *arXiv:1611.00712 [cs, stat]* URL http://arxiv.org/abs/1611.00712. ArXiv: 1611.00712.

Mallya, A.; Davis, D.; and Lazebnik, S. 2018. Piggyback: Adapting a Single Network to Multiple Tasks by Learning to Mask Weights. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Mancini, M.; Ricci, E.; Caputo, B.; and Rota Bulo, S. 2018. Adding New Tasks to a Single Network with Weight Transformations using Binary Masks. In *Proceedings of the European Conference on Computer Vision (ECCV) Workshops*.

Maninis, K.-K.; Radosavovic, I.; and Kokkinos, I. 2019. Attentive Single-Tasking of Multiple Tasks. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1851–1860. Long Beach, CA, USA: IEEE.

Misra, I.; Shrivastava, A.; Gupta, A.; and Hebert, M. 2016. Cross-Stitch Networks for Multi-Task Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3994–4003.

Mordan, T.; Thome, N.; Henaff, G.; and Cord, M. 2018. Revisiting Multi-Task Learning with ROCK: a Deep Residual Auxiliary Block for Visual Detection. In *Advances in Neural Information Processing Systems 31*, 1310–1322.

Sener, O.; and Koltun, V. 2018. Multi-Task Learning as Multi-Objective Optimization. In *Advances in Neural Information Processing Systems 31*, 527–538.

Silberman, N.; Hoiem, D.; Kohli, P.; and Fergus, R. 2012. Indoor Segmentation and Support Inference from RGBD Images. In *European Conference on Computer Vision (ECCV) 2012*, Lecture Notes in Computer Science, 746–760.

Singh, S. P. 1992. Transfer of Learning by Composing Solutions of Elemental Sequential Tasks. *Machine Learning* 8(3-4): 323–339.

Sinha, A.; Chen, Z.; Badrinarayanan, V.; and Rabinovich, A. 2018. Gradient Adversarial Training of Neural Networks. *arXiv:1806.08028 [cs, stat]* URL http://arxiv.org/abs/1806.08028. ArXiv: 1806.08028.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15(56): 1929–1958.

Strezoski, G.; Noord, N. v.; and Worring, M. 2019. Many Task Learning With Task Routing. In *The IEEE International Conference on Computer Vision (ICCV)*, 1375–1384.

Tajbakhsh, N.; Shin, J. Y.; Gurudu, S. R.; Hurst, R. T.; Kendall, C. B.; Gotway, M. B.; and Liang, J. 2016. Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? *IEEE Transactions on Medical Imaging* 35(5): 1299–1312.

Vandenhende, S.; Georgoulis, S.; and Van Gool, L. 2020. MTI-Net: Multi-Scale Task Interaction Networks for Multi-Task Learning. *arXiv:2001.06902 [cs]* URL http://arxiv.org/abs/2001.06902. ArXiv: 2001.06902.

Xu, D.; Ouyang, W.; Wang, X.; and Sebe, N. 2018. PAD-Net: Multi-Tasks Guided Prediction-and-Distillation Network for Simultaneous Depth Estimation and Scene Parsing. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 675–684.

Yosinski, J.; Clune, J.; Bengio, Y.; and Lipson, H. 2014. How Transferable are Features in Deep Neural Networks? In *Advances in Neural Information Processing Systems 27*, 3320–3328.

Zamir, A. R.; Sax, A.; Shen, W.; Guibas, L. J.; Malik, J.; and Savarese, S. 2018. Taskonomy: Disentangling Task Transfer Learning. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Zhang, Y.; Wei, Y.; and Yang, Q. 2018. Learning to Multitask. In *Advances in Neural Information Processing Systems 31*, 5771–5782.

Zhang, Z.; Cui, Z.; Xu, C.; Yan, Y.; Sebe, N.; and Yang, J. 2019. Pattern-Affinitive Propagation Across Depth, Surface Normal and Semantic Segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 4106–4115.

A Proof of Lemma 2

At $c = 0$, every element of $\mathcal{M}(0)$ follows a Bernoulli distribution:

$$P(m_{i,t} = 1) \sim \mathcal{B}(p).$$

We assume $P\left(m_{i,t}(c) = 1\right) = p, \ \forall c \in \{1, ..., (1-p)S - 1\}$ and prove it holds for $c + 1$.

The probability $P\left(m_{i,t}(c+1) = 1\right)$ can be written as:

$$\begin{aligned} &P(m_{i,t}(c+1) = 1) = \\ &\quad P(m_{i,t}(c+1) = 1 \mid m_{i,t}(c) = 1) P(m_{i,t}(c) = 1) \\ &+ P(m_{i,t}(c+1) = 1 \mid m_{i,t}(c) = 0) P(m_{i,t}(c) = 0). \end{aligned} \quad (7)$$

Since $P\left(m_{i,t}(c) = 1\right) = P\left(i \in A_t(c)\right)$, Eq. 7 can be reformulated as:

$$\begin{aligned} &P(i \in A_t(c+1)) = \\ &\quad P(i \in A_t(c+1) \mid i \in A_t(c)) P(i \in A_t(c)) \\ &\quad + P(i \in A_t(c+1) \mid i \notin A_t(c)) P\left(i \notin A_t(c)\right). \end{aligned} \quad (8)$$

As i_- is uniformly sampled from $A_t(c)$, the first term in Eq. 8 can be reformulated as

$$\begin{aligned} &P(i \in A_t(c+1) \mid i \in A_t(c)) P(i \in A_t(c)) = \\ &\qquad \left(1 - \frac{1}{pS}\right) p = p - \frac{1}{S}. \end{aligned} \quad (9)$$

Let us now expand the second term in Eq. 8 by considering whether $i \in B_t(c)$ or not:

$$\begin{aligned} &P(i \in A_t(c+1) \mid i \notin A_t(c)) P(i \notin A_t(c)) = \\ &\quad P(i \in A_t(c+1) \mid i \notin A_t(c), i \notin B_t(c)) \\ &\quad \times P(i \notin A_t(c) \mid i \notin B_t(c)) P(i \notin B_t(c)) \\ &\quad + P(i \in A_t(c+1) \mid i \notin A_t(c), i \in B_t(c)) \\ &\quad \times P(i \notin A_t(c) \mid i \in B_t(c)) P(i \in B_t(c)). \end{aligned} \quad (10)$$

From Def. 1, $P(i \in A_t(c+1) \mid i \notin A_t(c), i \in B_t(c)) = 0$ and $A_t(c) \subset B_t(c)$, thus (10) becomes:

$$\begin{aligned} &P(i \in A_t(c+1) \mid i \notin A_t(c)) P\left(i \notin A_t(c)\right) = \\ &\quad P(i \in A_t(c+1) \mid i \notin B_t(c)) P\left(i \notin B_t(c)\right). \end{aligned}$$

Given that i_+ is uniformly sampled from $\{1, ..., S\} \backslash B_t(c)$:

$$\begin{aligned} &P(i \in A_t(c+1) \mid i \notin A_t(c)) P(i \notin A_t(c)) = \\ &\qquad \frac{1}{(1-p)S - c} \cdot \frac{(1-p)S - c}{S} = \frac{1}{S}. \end{aligned} \quad (11)$$

By replacing (9) and (11) in Eq. 8 we obtain

$$\begin{aligned} P\left(m_{i,t}(c+1) = 1\right) &= P\left(i \in A_t(c+1)\right) \\ &= p - \frac{1}{S} + \frac{1}{S} \\ &= p, \end{aligned}$$

which demonstrates that $P(m_{i,t}(c) = 1)$ remains constant over c, given a uniform sampling of i_- and i_+ from $A_t(c)$ and $\{1, ..., S\} \backslash B_t(c)$, respectively □

Tasks	Classes
Global	Attractive, Blurry, Chubby, Double Chin, Heavy Makeup, Male, Oval Face, Pale Skin, Young
Eyes	Bags Under Eyes, Eyeglasses, Narrow Eyes, Arched Eyebrows, Bushy Eyebrows
Hair	Bald, Bangs, Black Hair, Blond Hair, Brown Hair, Gray Hair, Receding Hairline, Straight Hair, Wavy Hair
Mouth	Big Lips, Mouth Slightly Open, Smiling, Wearing Lipstick
Nose	Big Nose, Pointy Nose
Beard	5 o' Clock Shadow, Goatee, Mustache, No Beard, Sideburns
Cheeks	High Cheekbones, Rosy Cheeks
Wearings	Wearing Earrings, Wearing Hat, Wearing Necklace, Wearing Necktie

Table 5: Class composition of each the tasks for the Celeb-A dataset.

B Experimental Setup

In this section we provide a detailed description of the experimental setup used for the experiments on each of the considered datasets.

Celeb-A

Table 5 provides details on the distribution of the 40 facial attributes between the 8 created tasks. Every attribute in a task uses the same parameter partition. During training, the losses of all the attributes of the same task are averaged to form a task-specific loss. All baselines use a ResNet-18 (He et al. 2016) truncated after the last average pooling as a shared network. We then add 8 fully connected layers of input size 512, one per task, with the appropriate number of outputs, i.e. the number of facial attributes in the task. The partitioning methods ((Maninis, Radosavovic, and Kokkinos 2019), (Strezoski, Noord, and Worring 2019) and Maximum Roaming) are applied to every shared convolutional layer in the network. The parameter α in GradNorm (Chen et al. 2018) has been optimized in the set of values $\{0.5, 1, 1.5\}$. All models were trained with an Adam optimizer (Kingma and Ba 2017) and a learning rate of $1\mathrm{e}{-4}$, until convergence, using a binary cross-entropy loss function, averaged over the different attributes of a given task. We use a batch size of 256, and all input images are resized to $(64 \times 64 \times 3)$. The reported results are evaluated validation split provided in the official release of the dataset (Liu et al. 2015).

CityScapes

All baselines use a SegNet (Badrinarayanan, Kendall, and Cipolla 2017) outputting 64 feature maps of same height and width as the inputs. For each of the 8 tasks, we add one

prediction head, composed of one $(3 \times 3 \times 64 \times 64)$ and one $(1 \times 1 \times 64 \times 1)$ convolutions. A sigmoid function is applied on the output of the segmentation tasks. The partitioning methods ((Maninis, Radosavovic, and Kokkinos 2019), (Strezoski, Noord, and Worring 2019) and Maximum Roaming) are applied to every shared convolutional layer in the network. This excludes those in the task respective prediction heads. The parameter α in GradNorm (Chen et al. 2018) has been optimized in the set of values $\{0.5, 1, 1.5\}$. All models were trained with an Adam optimizer (Kingma and Ba 2017) and a learning rate of $1e{-4}$, until convergence. We use the binary cross-entropy as a loss function for each segmentation task, and the averaged absolute error for the depth estimation task. We use a batch size of 8, and the input samples are resized to 128×256, provided as such by (Liu, Johns, and Davison 2019). [1]. The reported results are evaluated on the validation split furnished by (Liu, Johns, and Davison 2019).

NYUv2

For both segmentation tasks and depth estimation task, we use the same configuration as for CityScapes. For the normals estimation task, the prediction head is made of one $(3 \times 3 \times 64 \times 64)$ and one $(1 \times 1 \times 64 \times 3)$ convolutions. Its loss is computed with an element-wise dot product between the normalized predictions and the ground-truth map. We use a batch size of 2, and the input samples are here resized to 288×384, provided as such by (Liu, Johns, and Davison 2019). The reported results are evaluated on the validation split furnished by (Liu, Johns, and Davison 2019).

C Celeb-A Dataset Benchmark

On top of the benchmark in the main document, figure 2 shows radar charts with the individual F-scores obtained by the different multi-task baselines for each of the 40 facial attributes. For improved readability, the scores have been plotted in two different charts, one for the 20 highest scores and one for the remaining 20 lowest.

Results confirm the superiority of our method (already shown in table 1), and show the consistency of our observations across the 40 classes, our method reaching the best performances on several individual facial attributes. Back on table 1, it is also important to remark that in (Sener and Koltun 2018) the authors report an error of 8.25% for MGDA-UB and 8.44% for GradNorm in the Celeb-A dataset. In our experimental setup, MGDA-UB reports an error of 10.53%, GradNorm reports 10.28% and Maximum Roaming 9.81%. These difference might be explained by factors linked to the different experimental setups. Firstly, (Sener and Koltun 2018) uses each facial attribute as an independent task, while we create 8 tasks out of different attribute groups. Secondly, both works use different reference metrics: we report performance at highest validation F-score, while they do it on accuracy.

[1] https://github.com/lorenmt/mtan

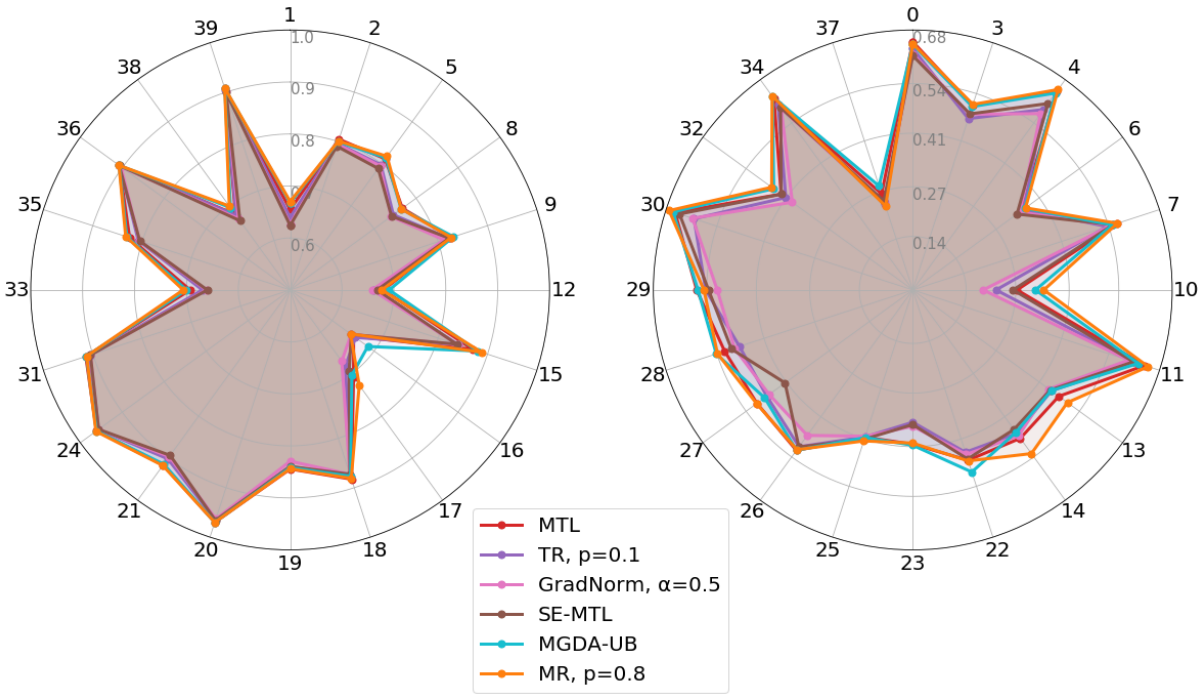

Figure 2: Radar chart comparing different baselines F-scores on every facial attribute of Celeb-A. **(left)** attributes with highest scores, **(right)** attributes with lowest scores. Each plot is displayed at a different scale.