Revisiting Multi-Task Learning in the Deep Learning Era

Simon Vandenhende, Stamatios Georgoulis, Marc Proesmans, Dengxin Dai and Luc Van Gool

Abstract—Despite the recent progress in deep learning, most approaches still go for a silo-like solution, focusing on learning each task in isolation: training a separate neural network for each individual task. Many real-world problems, however, call for a multi-modal approach and, therefore, for multi-tasking models. Multi-task learning (MTL) aims to leverage useful information across tasks to improve the generalization capability of a model. In this survey, we provide a well-rounded view on state-of-the-art MTL techniques within the context of deep neural networks. Our contributions concern the following. First, we consider MTL from a network architecture point-of-view. We include an extensive overview and discuss the advantages/disadvantages of recent popular MTL models. Second, we examine various optimization methods to tackle the joint learning of multiple tasks. We summarize the qualitative elements of these works and explore their commonalities and differences. Finally, we provide an extensive experimental evaluation across a variety of datasets to examine the pros and cons of different methods, including both architectural and optimization based strategies.

Index Terms—Multi-Task Learning, Optimization, Dense Prediction Tasks, Neural Networks.

1 Introduction

Over the last decade, neural networks have shown impressive results for a multitude of tasks, such as semantic segmentation [1], instance segmentation [2] and monocular depth estimation [3]. Traditionally, these tasks are tackled in isolation, i.e. a separate neural network is trained for each task. Yet, many real-world problems are inherently multi-modal. For example, an autonomous car should be able to detect all objects in the scene, localize them, understand what they are, estimate their distance and trajectory, etc., in order to safely navigate itself in its surroundings. Similarly, an intelligent advertisement system should be able to detect the presence of people in its viewpoint, understand their gender and age group, analyze their appearance, track where they are looking at, etc., in order to provide personalized content. At the same time, humans are remarkably good at solving many tasks concurrently. Biological data processing appears to follow a multi-tasking strategy too: instead of separating tasks and tackling them in isolation, different processes seem to share the same early processing layers in the brain (see V1 in macaques [4]). The aforementioned observations have motivated researchers to develop multi-task learning (MTL) models that given an input image can infer all desired task outputs.

Before the deep learning era, MTL works tried to model the common information among tasks in the hope that a joint task learning could result in better generalization performance. To achieve this, they placed assumptions on the task parameter space, such as: task parameters should lie close to each other w.r.t. some distance metric [5], [6], [7], [8], share a common probabilistic prior [9], [10], [11], [12], [13], or reside in a low dimensional subspace [14], [15], [16] or manifold [17]. These assumptions work well when all tasks are related [5], [14], [18], [19], but can lead to performance degradation if information sharing happens between unrelated tasks. The latter is a known problem in MTL, referred to as *negative transfer*. To mitigate this problem, some of these works opted to cluster tasks into groups based on prior beliefs about their similarity or relatedness.

In the deep learning era, MTL translates to designing networks capable of learning shared representations from multi-task supervisory signals. Compared to the single-task case, where each individual task is solved separately by its own network, such multi-task networks theoretically bring several advantages to the table. First, due to their inherent layer sharing, the resulting memory footprint is substantially reduced. Second, as they explicitly avoid to repeatedly calculate the features in the shared layers, once for every task, they show increased inference speeds. Most importantly, they have the potential for improved performance if the associated tasks share complementary information, or act as a regularizer for one another. Evidence for the former has been provided in the literature for certain pairs of tasks, e.g. detection and classification [20], [21], detection and segmentation [2], [22], segmentation and depth estimation [23], [24], while for the latter recent efforts point to that direction [25]. These remarks led to the development of the first deep multi-task networks that were historically classified into soft or hard parameter sharing techniques.

In soft parameter sharing, each task is assigned its own set of parameters (i.e. task-specific networks) and feature sharing mechanisms handle the cross-task talk. For example, cross-stitch networks [26] used a linear combination of the activations in every layer of the task-specific networks as a

- *Simon Vandenhende and Marc Proesmans are with the Center for Processing Speech and Images, Department Electrical Engineering, KU Leuven. E-mail: {simon.vandenhende,marc.proesmans}@kuleuven.be*
- *Stamatios Georgoulis and Dengxin Dai are with the Computer Vision Lab, Department Electrical Engineering, ETH Zurich. E-mail: {georgous,daid}@ee.ethz.ch*
- *Luc Van Gool is with both the Center for Processing Speech and Images, KU Leuven and the Computer Vision Lab, ETHZ. E-mail: vangool@vision.ee.ethz.ch*

means for soft feature fusion. Sluice networks [27] extended this idea by allowing to learn the selective sharing of layers, subspaces and skip connections. NDDR-CNN [28] also incorporated dimensionality reduction techniques into the feature fusion layers. Differently, MTAN [29] used an attention mechanism to share a general feature pool amongst the task-specific networks. A concern with soft parameter sharing approaches is scalability, as the size of the multi-task network tends to grow linearly with the number of tasks.

In hard parameter sharing, the parameter set is divided into shared and task-specific operations. UberNet [30] was one of the first such architectures that hosted a multi-head design across different network layers and scales. Still, the most characteristic hard parameter sharing design consists of a shared encoder that branches out into task-specific decoding heads [31], [32], [33], [34]. Multilinear relationship networks [35] extended this design by placing tensor normal priors on the parameter set of the fully connected layers. In these works the branching points in the network are determined ad hoc, which can lead to suboptimal task groupings. To alleviate this issue, tree-based approaches [36], [37] started from a thin network where tasks initially share all layers, but the final one, and dynamically branched the model by determining the optimal task groupings in a layer-by-layer fashion. Similarly, stochastic filter groups [38] re-purposed the convolution kernels in each layer to support shared or task-specific behaviour.

Despite the progress reported by these or similar works, the joint learning of multiple tasks is prone to negative transfer if the task dictionary contains unrelated tasks. The latter has been well documented in [30], where an improvement in estimating normals leads to a decline in object detection, or in [2] where the multi-task version underperforms the single-task ones. To remedy this situation, a group of methods carefully balanced the learning of the individual tasks in an attempt to find an equilibrium where no task declines significantly. For example, Kendall et al. [32] used the homoscedastic uncertainty of each individual task to re-weigh the losses. Gradient normalization [33] was proposed to balance the losses by adaptively normalizing the magnitude of each task's gradients. Similarly, Sinha et al. [39] tried to balance the losses by adapting the gradients magnitude, but differently, they employed adversarial training to this end. Dynamic task prioritization [40] proposed to dynamically sort the order of task learning, and prioritized 'difficult' tasks over 'easy' ones. Zhao et al. [41] introduced a modulation module to encourage feature sharing among 'relevant' tasks and disentangle the learning of 'irrelevant' tasks. Sener and Koltun [34] proposed to cast multi-task learning into a multi-objective optimization scheme, where the weighting of the different losses is adaptively changed such that a Pareto optimal solution is achieved.

Recently, Maninis et al. [42] followed a 'single-tasking' route for MTL, i.e. in a multi-tasking framework they performed separate forward passes, one for each task, that activate shared responses among all tasks, plus some residual responses that are task-specific. Furthermore, to suppress the negative transfer issue they applied adversarial training on the gradients level that enforces them to be statistically indistinguishable across tasks during the update step.

All works presented so far follow a common pattern: they *directly* predict all task outputs from the same input in one processing cycle (i.e. all predictions are generated once, in parallel or sequentially, and are not refined afterwards). In contrast, a few recent works first employed a multi-task network to make initial task predictions, and then leveraged features from these initial predictions to further improve each task output – in a one-off or recursive manner. PAD-Net [24] proposed to distil information from the initial predictions of other tasks, by means of spatial attention, before adding it as a residual to the task of interest. JTRL [43] opted for sequentially predicting each task, with the intention to utilize information from the past predictions of one task to refine the features of another task at each iteration. PAP-Net [44] extended upon this idea, and used a recursive procedure to propagate similar cross-task and task-specific patterns found in the initial task predictions. To do so, they operated on the affinity matrices of the initial predictions, and not on the features themselves, as was the case before [24], [43]. MTI-Net [45] adopted a multi-scale multi-modal distillation procedure to explicitly model the unique task interactions that happen at each individual scale.

Related work. MTL has been the subject of several surveys [47], [48], [49], [50]. In [47], Caruana showed that MTL can be beneficial as it allows for the acquisition of inductive bias through the inclusion of related additional tasks into the training pipeline. The author showcased the use of MTL in artificial neural networks, decision trees and k-nearest neighbors methods, but this study is placed in the very early days of neural networks rendering it outdated in the deep learning era. Ruder [48] gave an overview of recent MTL techniques (e.g. [26], [27], [32], [36]) applied in deep neural networks. In the same vein, Zhang and Yang [49] provided a survey that includes feature learning, low-rank, task clustering, task relation learning, and decomposition approaches for MTL. Yet, both works are rather literature review studies without an empirical evaluation or comparison of the presented techniques. Finally, Gong et al. [50] benchmarked several task balancing approaches (e.g. fixed, uncertainty [32], DWA [29]) using different hard parameter sharing networks (shared trunk, task-specific heads) across three MTL datasets. Still, the scope of this study is rather limited, and explicitly focuses on the task balancing aspect.

In this survey, we provide a well-rounded view on state-of-the-art MTL techniques within the context of deep neural networks. From a network architecture point-of-view, we analyze both encoder- and decoder-based approaches as well as other designs. From an optimization strategy point-of-view, we consider the majority of task balancing, adversarial and modulation techniques. We provide extensive experimental evaluation across different datasets both within the scope of each group of methods (e.g. encoder-based approaches) as well as across groups of methods (e.g. encoder- vs decoder-based approaches). Note that, in our experiments we emphasize on multiple dense prediction tasks, rather than multiple classification tasks, an arguably harder case[1] that has been mostly under-explored in MTL. To the best of our knowledge this is the first survey on MTL

1. As jointly learning multiple dense prediction tasks is governed by the use of different loss functions, unlike classification tasks that mostly use cross-entropy losses, additional consideration is required to avoid a scenario where some tasks overwhelm the others during training.

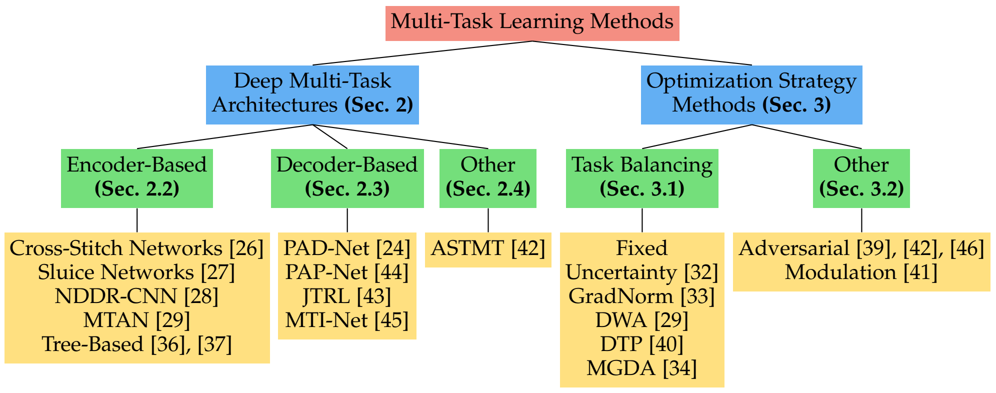
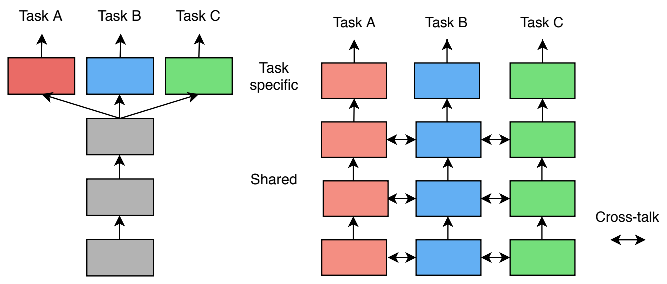


Fig. 1: A taxonomy of multi-task learning approaches in the deep learning era.



(a) Hard parameter sharing. (b) Soft parameter sharing.

Fig. 2: Multi-task learning using deep neural networks.

that explicitly explores such a wide range of subtopics and experimentally verifies their impact. Upon publication, all code will be made publicly available to ease the adoption of MTL techniques within deep neural networks.

Paper overview. Section 2 describes different deep multi-task architectures, subdivided into two main groups: encoder-based and decoder-based approaches. Section 3 surveys various optimization techniques for balancing the influence of the tasks when updating the network's weights. In Section 4 we provide an extensive experimental comparison between different MTL methods, including both architectural and optimization based techniques, across a variety of datasets. Section 5 discusses the relations of MTL with other fields. Section 6 concludes the paper.

2 Deep Multi-Task Architectures

In this section, we discuss deep multi-task architectures from different groups of works, and analyze their advantages and disadvantages from a theoretical point-of-view. An experimental comparison is also provided later in Section 4. Note that, as a detailed presentation of each architecture is beyond the scope of this survey, in each case we refer the reader to the corresponding paper for further details that complement the following descriptions.

2.1 A New Taxonomy of MTL Approaches

As explained in Section 1, multi-task networks have historically been classified into soft or hard parameter sharing techniques. In *hard parameter sharing*, the parameter set is divided into shared and task-specific parameters (see Figure 2a). MTL models using hard parameter sharing typically consist of a shared encoder that branches out into task-specific heads [31], [32], [33], [34]. In *soft parameter sharing*, each task is assigned its own set of parameters and a feature sharing mechanism handles the cross-talk (see Figure 2b).

Several recent MTL works took inspiration from both hard and soft parameter sharing schemes to jointly solve multiple dense prediction tasks. As a consequence, it is debatable whether the soft vs hard parameter sharing paradigm should still be used as the main framework for classifying MTL architectures. In this survey, we propose an alternative taxonomy that discriminates between different architectures on the basis of where the *task interactions* take place, i.e. the locations in the network where information or features are exchanged or shared between tasks. Given this criterion, we distinguish between two types of models: *encoder-based* and *decoder-based* MTL architectures.

In the former case, the tasks share or exchange features during the encoding stage, after which a set of task-specific heads predict the output for each task. Note that, there is no further interaction between the task-specific heads during the decoding stage. In the latter case, the tasks share or exchange features during the decoding stage (i.e. after the task-specific heads) to refine the initial predictions for every task. Figure 1 gives an overview of the proposed taxonomy, listing representative works in each case.

2.2 Encoder-Based Architectures

Encoder-based architectures share the task features in the encoding stage, before they process them with a set of independent task-specific heads. A number of works [31], [32], [33], [34], [51] followed an ad hoc strategy by sharing an off-the-shelf backbone network in combination with small task-specific heads (see Figure 2a). This model relies on the encoder (i.e. backbone network) to learn a generic representation of the scene. The features from the encoder are then used by the task-specific heads to get the predictions for every task. While this simple model shares the full encoder amongst all tasks, recent works have considered *where* and *how* the feature sharing should happen in the encoder. We discuss such sharing strategies in the following sections.

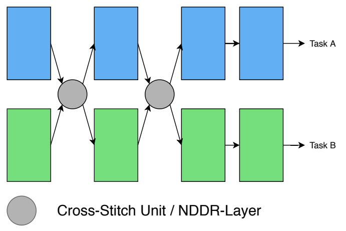


Fig. 3: The architecture of cross-stitch networks [26] and NDDR-CNNs [28]. The activations from all single-task networks are fused across several encoding layers. Different feature fusion mechanisms were used in each case.

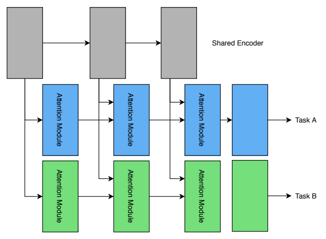


Fig. 4: The architecture of MTAN [29]. Task-specific attention modules select and refine features from several layers of a shared encoder.

2.2.1 *Cross-Stitch Networks*

Cross-stitch networks [26] shared the activations amongst all single-task networks in the encoder. Assume we are given two activation maps x_A, x_B at a particular layer, that belong to tasks A and B respectively. A learnable linear combination of these activation maps is applied, before feeding the transformed result $\tilde{x}_A, \tilde{x}_B$ to the next layer in the single-task networks. The transformation is parameterized by learnable weights α, and can be expressed as

$$\begin{bmatrix} \tilde{x}_A \\ \tilde{x}_B \end{bmatrix} = \begin{bmatrix} \alpha_{AA} & \alpha_{AB} \\ \alpha_{BA} & \alpha_{BB} \end{bmatrix} \begin{bmatrix} x_A \\ x_B \end{bmatrix}. \qquad (1)$$

As illustrated in Figure 3, this procedure is repeated at multiple locations in the encoder. By learning the weights α, the network can decide the degree to which the features are shared between tasks. In practice, we are required to pre-train the single-task networks, before stitching them together, in order to maximize the performance. A disadvantage of cross-stitch networks is that the size of the network increases linearly with the number of tasks. Furthermore, it is not clear where the cross-stitch units should be inserted in order to maximize their effectiveness. **Sluice networks** [27] extended this work by also supporting the selective sharing of subspaces and skip connections.

2.2.2 *Neural Discriminative Dimensionality Reduction*

Neural Discriminative Dimensionality Reduction CNNs (NDDR-CNNs) [28] used a similar architecture with cross-stitch networks (see Figure 3). However, instead of utilizing a linear combination to fuse the activations from all single-task networks, a dimensionality reduction mechanism is employed. First, features with the same spatial resolution in the single-task networks are concatenated channel-wise. Second, the number of channels is reduced by processing the features with a 1 by 1 convolutional layer, before feeding the result to the next layer. The convolutional layer allows to fuse activations across all channels. Differently, cross-stitch networks only allow to fuse activations from channels that share the same index. The NDDR-CNN behaves as a cross-stitch network when the non-diagonal elements in the weight matrix of the convolutional layer are zero.

Due to their similarity with cross-stitch networks, NDDR-CNNs are prone to the same problems. First, there is a scalability concern when dealing with a large number of tasks. Second, NDDR-CNNs involve additional design choices, since we need to decide where to include the NDDR layers. Finally, both cross-stitch networks and NDDR-CNNs only allow to use limited local information when fusing the activations from the different single-task networks. We hypothesize that this is suboptimal cause the use of sufficient context is very important during encoding – as already shown for the tasks of image classification [52] and semantic segmentation [53], [54], [55]. This is backed up by certain decoder-based architectures in Section 2.3 that overcome the limited receptive field by predicting the tasks at multiple scales and by sharing the features repeatedly at every scale.

2.2.3 *Multi-Task Attention Networks*

Multi-Task Attention Networks (MTAN) [29] used a shared backbone network in conjunction with task-specific attention modules in the encoder (see Figure 4). The shared backbone extracts a general pool of features. Then, each task-specific attention module selects features from the general pool by applying a soft attention mask. The attention mechanism is implemented using regular convolutional layers and a sigmoid non-linearity. Since the attention modules are small compared to the backbone network, the MTAN model does not suffer as severely from the scalability issues that are typically associated with cross-stitch networks and NDDR-CNNs. However, similar to the fusion mechanism in the latter works, the MTAN model can only use limited local information to produce the attention mask.

2.2.4 *Tree-Based Encoders*

The models presented in Sections 2.2.1-2.2.3 softly shared the features amongst tasks during the encoding stage. Differently, tree-based encoders followed a hard-parameter sharing scheme. Before presenting these methods, consider the following observation: deep neural networks tend to learn hierarchical image representations [56]. The early layers tend to focus on more general low-level image features, such as edges, corners, etc., while the deeper layers tend to extract high-level information that is more task-specific. Motivated by this observation, tree-based encoders opted to learn similar hierarchical encoding structures [36], [37], [57]. These ramified networks typically start with a number of shared layers, after which different (groups of) tasks branch out into their own sequence of layers. In doing so, the different branches gradually become more task-specific as we move to the deeper layers. This behaviour aligns well with the hierarchical representations learned by deep neural nets. However, as the number of possible network configurations are combinatorially large, deciding what layers to share and where to branch out becomes cumbersome. The following works tried to automate the procedure of hierarchically clustering the tasks to form tree-based encoders.

Fully-Adaptive Feature Sharing (FAFS) [36] started from a thin network, where tasks initially shared all layers, and dynamically grew the model in a greedy layer-by-layer fashion. The procedure for constructing the tree-based encoder is illustrated in Figure 5. The task groupings at each layer are decided based on the probability of concurrently ’simple’ or ’difficult’ examples across tasks. This strategy assumes that it is preferable to solve two tasks in an isolated manner

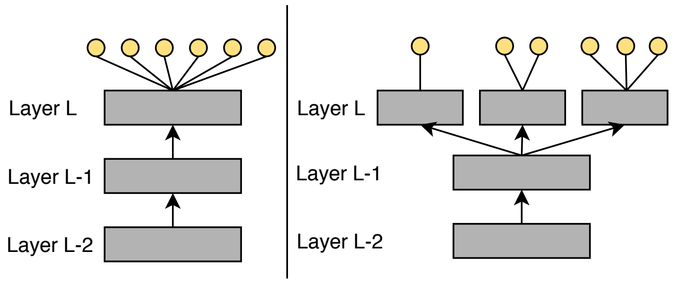


Fig. 5: Fully-adaptive feature sharing (FAFS) builds a tree-based encoder online during training [36]. *(Left)* The model initially shares all layers up to layer L, where we have a junction with 6 branches, one per task. Next, FAFS decides the task groupings at layer L. *(Right)* The 6 tasks are divided into 3 clusters. At layer $L-1$, we create 3 new branches, i.e. one per cluster. The weights at the newly created junction are initialized by replicating the current weights. In the next step, we determine how to group the tasks at layer $L-1$. The procedure finishes when no new branches are created.

(i.e. different branches) when the majority of examples are 'simple' for one task, but 'difficult' for the other[2]. Interestingly, during the tree construction phase the FAFS method allows flexibility by finding a trade-off between generating new branches for separating dissimilar (groups of) tasks, and minimizing the number of network parameters.

A considerable advantage of the FAFS method is that the computational overhead to determine the task groupings is negligible. However, the use of sample difficulty as a task affinity measure suffers from strong dataset dependency. Consider the example where we solve two separate tasks, monocular depth estimation and semantic segmentation, in an autonomous driving dataset. Estimating depth is known to be increasingly difficult at larger distances [31]. At the same time, there is typically an imbalance in the semantic labels. For example, only a limited number of scenes show examples of motorbikes, making it harder to predict their semantic label correctly. As a result, depending on where the motorbike examples are located in the scene, close by or far away, the semantic segmentation and monocular depth estimation tasks will be judged more or less related. The task affinity measure in FAFS is thus very sensitive to the bias that is present in the dataset, possibly ignoring the actual task relationships in the process.

Branched Multi-Task Networks [37] defined the task groupings based on feature affinity scores, rather than sample difficulty. The main assumption is that two tasks are strongly related, if their single-task models rely on a similar set of features. An efficient method, i.e. Representation Similarity Analysis [58], is used to quantify this property. In particular, the task affinity scores are found by computing correlations between image features that are extracted by a set of pretrained single-task networks.

Similarly to FAFS, it is possible to optimize a trade-off between network size and task similarity. Differently, the tree structure is determined offline using the pre-calculated affinities for the whole network, and not online in a greedy layer-by-layer fashion. This strategy promotes task groupings that are optimal in a global, rather than local, sense. Most importantly, as the task affinity scores are based on feature similarity, rather than sample difficulty, they do not suffer from strong dataset dependency as in [36].

Yet, a disadvantage of the approach is that a set of single-task networks need to be pretrained offline to determine the task groupings, which might render the tree construction phase time-consuming or computationally expensive. In contrast, FAFS determines the task groupings online during the tree construction phase.

2. In practice, the decision of whether an example is considered 'easy' or 'difficult' for a certain task is tied to a predetermined performance metric, e.g. a threshold w.r.t. to the task's loss function.

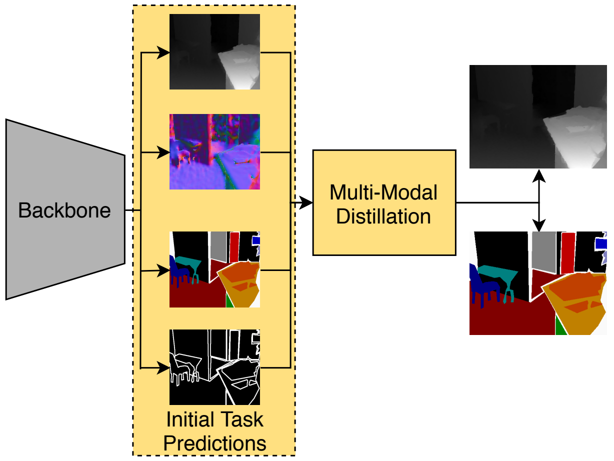


Fig. 6: The architecture in PAD-Net [24]. Features extracted by a backbone network are passed to task-specific heads to make initial task predictions. The task features from the different heads are then combined through a distillation unit to make the final predictions. Note that, auxiliary tasks can be used in this framework, i.e. tasks for which initial predictions are only generated, but not final ones.

2.3 Decoder-Based Architectures

The encoder-based architectures in Section 2.2 follow a common pattern: they *directly* predict all task outputs from the same input in one processing cycle (i.e. all predictions are generated once, in parallel or sequentially, and are not refined afterwards). By doing so, they fail to capture commonalities and differences among tasks, that are likely fruitful for one another (e.g. depth discontinuities are usually aligned with semantic edges). Arguably, this might be the reason for the moderate only performance improvements achieved by the encoder-based approaches to MTL (see Section 4.2). To alleviate this issue, a few recent works first employed a multi-task network to make initial task predictions, and then leveraged features from these initial predictions in order to further improve each task output – in an one-off or recursive manner. As these MTL approaches also share or exchange information during the decoding stage, we refer to them as decoder-based architectures.

2.3.1 PAD-Net

PAD-Net [24] was one of the first decoder-based architectures. The model itself is visualized in Figure 6. As can be seen, the input image is first processed by an off-the-shelf backbone network. The backbone features are further processed by a set of task-specific heads that produce an initial prediction for every task. These initial task predictions add deep supervision to the network, but they can also be used to exchange information between tasks, as will be explained next. The *task features* in the last layer of the task-specific heads contain a per-task feature representation of the scene. PAD-Net proposed to re-combine them via a multi-modal distillation unit, whose role is to extract cross-task information, before producing the final task predictions.

PAD-Net performs the multi-modal distillation by means of a spatial attention mechanism. Particularly, the output features F_k^o for task k are calculated as

$$F_k^o = F_k^i + \sum_{l \neq k} \sigma \left(W_{k,l} F_l^i \right) \odot F_l^i, \tag{2}$$

where $\sigma \left(W_{k,l} F_l^i \right)$ returns a spatial attention mask that is applied to the initial task features F_l^i from task l. The attention mask itself is found by applying a convolutional layer $W_{k,l}$ to extract local information from the initial task features. Equation 2 assumes that the task interactions are location dependent, i.e. tasks are not in a constant relationship across the entire image. This can be understood from a simple example. Consider two dense prediction tasks, e.g. monocular depth prediction and semantic segmentation. Depth discontinuities and semantic boundaries often coincide. However, when we segment a flat object, e.g. a magazine, from a flat surface, e.g. a table, we will still find a semantic boundary where the depth map is rather continuous. In this particular case, the depth features provide no additional information for the localization of the semantic boundaries. The use of spatial attention explicitly allows the network to select information from other tasks at locations where its useful.

The encoder-based approaches in Section 2.2 shared features amongst tasks using the intermediate representations in the encoder. Differently, PAD-Net models the task interactions by applying a spatial attention layer to the features in the task-specific heads. In contrast to the intermediate feature representations in the encoder, the task features used by PAD-Net are already disentangled according to the output task. We hypothesize that this makes it easier for other tasks to distill the relevant information. This multi-step decoding strategy from PAD-Net is applied and refined in other decoder-based approaches.

2.3.2 *Pattern-Affinitive Propagation Networks*

Pattern-Affinitive Propagation Networks (PAP-Net) [44] used an architecture similar to PAD-Net (see Figure 7), but the multi-modal distillation in this work is performed in a different manner. The authors argue that directly working on the task features space via the spatial attention mechanism, as done in PAD-Net, might be a suboptimal choice. As the optimization still happens at a different space, i.e. the task label space, there is no guarantee that the model will learn the desired task relationships. Instead, they statistically observed that pixel affinities tend to align well with common local structures on the task label space. Motivated by this observation, they proposed to leverage pixel affinities in order to perform multi-modal distillation.

To achieve this, the backbone features are first processed by a set of task-specific heads to get an initial prediction

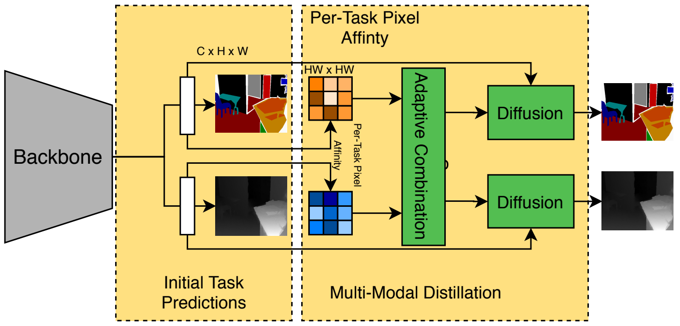


Fig. 7: The architecture in PAP-Net [44]. Features extracted by a backbone network are passed to task-specific heads to make initial task predictions. The task features from the different heads are used to calculate a per-task pixel affinity matrix. The affinity matrices are adaptively combined and diffused back into the task features space to spread the cross-task correlation information across the image. The refined features are used to make the final task predictions.

for every task. Second, a per-task pixel affinity matrix M_{T_j} is calculated by estimating pixel-wise correlations upon the task features coming from each head. Third, a cross-task information matrix $\hat{M}_{T_j}$ for every task T_j is learned by adaptively combining the affinity matrices M_{T_i} for tasks T_i with learnable weights $\alpha_i^{T_j}$

$$\hat{M}_{T_j} = \sum_{T_i} \alpha_i^{T_j} \cdot M_{T_i}. \tag{3}$$

Finally, the task features coming from each head j are refined using the cross-task information matrix $\hat{M}_{T_j}$. In particular, the cross-task information matrix is diffused into the task features space to spread the correlation information across the image. This effectively weakens or strengthens the pixel correlations for task T_j, based on the pixel affinities from other tasks T_i. The refined features are used to make the final predictions for every task.

All previously discussed methods only use limited local information when fusing features from different tasks. For example, cross-stitch networks and NDDR-CNNs combine the features in a channel-wise fashion, while PAD-Net only uses the information from within a 3 by 3 pixels window to construct the spatial attention mask. Differently, PAP-Net also models the non-local relationships through pixel affinities measured across the entire image.

2.3.3 *Joint Task-Recursive Learning*

Joint Task-Recursive Learning (JTRL) [43] recursively predicts two tasks at increasingly higher scales in order to gradually refine the results based on past states. The architecture is illustrated in Figure 8. Similarly to PAD-Net and PAP-Net, a multi-modal distillation mechanism is used to combine information from earlier task predictions, through which later predictions are refined. Differently, the JTRL model predicts two tasks sequentially, rather than in parallel, and in an intertwined manner. The main disadvantage of this approach is that it is not straightforward, or even possible, to extent this model to more than two tasks given the intertwined manner at which task predictions are refined.

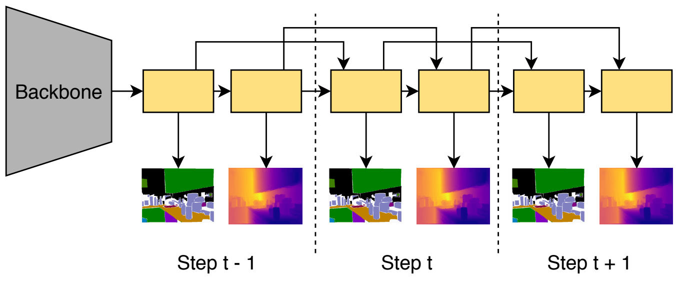


Fig. 8: The architecture in Joint Task-Recursive Learning [43]. The features of two tasks are progressively refined in an intertwined manner based on past states.

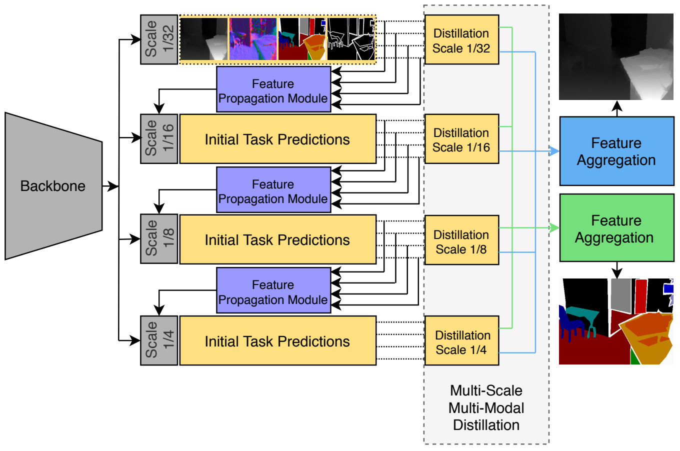


Fig. 9: The architecture in Multi-Scale Task Interaction Networks [45]. Starting from a backbone that extracts multi-scale features, initial task predictions are made at each scale. These task features are then distilled separately at every scale, allowing the model to capture unique task interactions at multiple scales, i.e. receptive fields. After distillation, the distilled task features from all scales are aggregated to make the final task predictions. To boost performance, a feature propagation module is included to pass information from lower resolution task features to higher ones.

2.3.4 *Multi-Scale Task Interaction Networks*

In the decoder-based architectures presented so far, the multi-modal distillation was performed at a fixed scale, i.e. the features of the backbone's last layer. This rests on the assumption that all relevant task interactions can solely be modeled through a single filter operation with specific receptive field. However, **Multi-Scale Task Interaction Networks** (MTI-Net) [45] showed that this is a rather strict assumption. In fact, tasks can influence each other differently at different receptive fields.

To account for this restriction, MTI-Net explicitly took into account task interactions at multiple scales. Its architecture is illustrated in Figure 9. First, an off-the-shelf backbone network extracts a multi-scale feature representation from the input image. From the multi-scale feature representation an initial prediction for every task is made at each scale. The task predictions at a particular scale are found by applying a task-specific head to the backbone features extracted at that scale. Similarly to PAD-Net, the features in the last layer of the task-specific heads are combined and refined to make the final predictions. Differently, in MTI-Net the per-task feature representations can be distilled at each scale separately. This allows to have multiple task interactions, each modeled within a specific receptive field. The distilled multi-scale features are upsampled to the highest scale and concatenated, resulting in a final feature representation for every task. The final task predictions are found by decoding these final feature representations in a task-specific manner again. The performance was further improved by also propagating information from the lower resolution task features to the higher ones using a Feature Propagation Module.

The experimental evaluation in [45] shows that distilling task information at multiple scales increases the multi-tasking performance compared to PAD-Net where such information is only distilled at a single scale. Furthermore, since MTI-Net distills the features at multiple scales, i.e. using different pixel dilations, it overcomes the issue of using only limited local information to fuse the features, which was already shown to be beneficial in PAP-Net.

2.4 Other Approaches

A number of approaches that fall outside the aforementioned categories have been proposed in the literature. For example, multilinear relationship networks [35] used tensor normal priors to the parameter set of the task-specific heads to allow interactions in the decoding stage. Yang et al. [59] generalized matrix factorisation approaches to MTL in order to learn cross-task sharing structures in every layer of the network. Routing networks [60] proposed a principled approach to determine the connectivity of a network's function blocks through routing. Piggyback [61] showed how to adapt a single, fixed neural network to a multi-task network by learning binary masks. Huang et al. [57] introduced a method rooted in neural architecture search for the automated construction of a tree-based multi-attribute learning network. Stochastic filter groups [38] re-purposed the convolution kernels in each layer of the network to support shared or task-specific behaviour. In a similar vein, feature partitioning [62] presented partitioning strategies to assign the convolution kernels in each layer of the network into different tasks. In general, these works have a different scope within MTL, e.g. automate the network architecture design. Moreover, they mostly focus on solving multiple (binary) classification tasks, rather than multiple dense predictions tasks. As a result, they fall outside the scope of this survey, with one notable exception that is discussed next.

Attentive Single-Tasking of Multiple Tasks (ASTMT) [42] proposed to take a 'single-tasking' route for the MTL problem. That is, within a multi-tasking framework they performed separate forward passes, one for each task, that activate shared responses among all tasks, plus some residual responses that are task-specific. Furthermore, to suppress the negative transfer issue they applied adversarial training on the gradients level that enforces them to be statistically indistinguishable across tasks. An advantage of this approach is that shared and task-specific information within the network can be naturally disentangled. On the negative side, however, the tasks can not be predicted altogether, but only one after the other, which significantly increases the inference speed and somehow defies the purpose of MTL.

3 Optimization in MTL

In the previous section, we discussed about the construction of network architectures able to learn multiple tasks concurrently. Still, a significant challenge in MTL stems from the optimization procedure itself. In particular, we need to carefully balance the joint learning of all tasks to avoid a scenario where one or more tasks have a dominant influence in the network weights. In this section, we discuss several methods that have considered this *task balancing* problem.

3.1 Task Balancing Approaches

Without loss of generality, the optimization objective in a MTL problem, assuming task-specific weights w_i and task-specific loss functions $\mathcal{L}_i$, can be formulated as

$$\mathcal{L}_{MTL} = \sum_i w_i \cdot \mathcal{L}_i. \tag{4}$$

When using stochastic gradient descent to minimize the objective from Equation 4, which is the standard approach in the deep learning era, the network weights in the shared layers W_{sh} are updated by the following rule

$$W_{sh} = W_{sh} - \gamma \sum_i w_i \frac{\partial \mathcal{L}_i}{\partial W_{sh}}. \tag{5}$$

From Equation 5 we can draw the following conclusions. First, the network weight update can be suboptimal when the task gradients conflict, or dominated by one task when its gradient magnitude is much higher w.r.t. the other tasks. This motivated researchers [29], [32], [33], [40] to balance the gradient magnitudes by setting the task-specific weights w_i in the loss. To this end, other works [34], [41], [63] have also considered the influence of the direction of the task gradients. Second, each task's influence on the network weight update can be controlled, either *indirectly* by adapting the task-specific weights w_i in the loss, or *directly* by operating on the task-specific gradients $\frac{\partial \mathcal{L}_i}{\partial W_{sh}}$. A number of methods that tried to address these problems are discussed next.

3.1.1 Uncertainty Weighting

Kendall et al. [32] used the **homoscedastic uncertainty** to balance the single-task losses. The homoscedastic uncertainty or *task-dependent uncertainty* is not an output of the model, but a quantity that remains constant for different input examples of the same task. The optimization procedure is carried out to maximise a Gaussian likelihood objective that accounts for the homoscedastic uncertainty. In particular, they optimize the model weights W and the noise parameters σ_1, σ_2 to minimize the following objective

$$\mathcal{L}(W, \sigma_1, \sigma_2) = \frac{1}{2\sigma_1^2}\mathcal{L}_1(W) + \frac{1}{2\sigma_2^2}\mathcal{L}_2(W) + \log \sigma_1 \sigma_2. \tag{6}$$

The loss functions $\mathcal{L}_1$, $\mathcal{L}_2$ belong to the first and second task respectively. By minimizing the loss $\mathcal{L}$ w.r.t. the noise parameters σ_1, σ_2, one can essentially balance the task-specific losses during training. The optimization objective in Equation 6 can easily be extended to account for more than two tasks too. The noise parameters are updated through standard backpropagation during training.

Note that, increasing the noise parameter σ_i reduces the weight for task i. Consequently, the effect of task i on the network weight update is smaller when the task's homoscedastic uncertainty is high. This is advantageous when dealing with noisy annotations since the task-specific weights will be lowered automatically for such tasks.

3.1.2 Gradient Normalization

Gradient normalization (GradNorm) [33] proposed to control the training of multi-task networks by stimulating the task-specific gradients to be of similar magnitude. By doing so, the network is encouraged to learn all tasks at an equal pace. Before presenting this approach, we introduce the necessary notations in the following paragraph.

We define the L2 norm of the gradient for the weighted single-task loss $w_i(t) \cdot L_i(t)$ at step t w.r.t. the weights W, as $G_i^W(t)$. We additionally define the following quantities,

- the mean task gradient w.r.t. the weights W at step t: $\bar{G}^W(t) = E_{task}\left[G_i^W(t)\right]$;
- the loss ratio for task i at step t, which can be viewed as a measure for the inverse training rate of task i: $\tilde{L}_i(t) = L_i(t) / L_i(0)$;
- the relative inverse training rate of task i at step t: $r_i(t) = \tilde{L}_i(t) / E_{task}\left[\tilde{L}_i(t)\right]$.

GradNorm aims to balance two properties during the training of a multi-task network. First, balancing the gradient magnitudes G_i^W. To achieve this, the mean gradient $\bar{G}^W$ is considered as a common basis from which the relative gradient sizes across tasks can be measured. Second, balancing the pace at which different tasks are learned. The inverse training rate $r_i(t)$ is used to this end. When the inverse training rate $r_i(t)$ increases, the gradient magnitude $G_i^W(t)$ for task i should increase as well to stimulate the task to train more quickly. GradNorm tackles both objectives by minimizing the following loss

$$\left| G_i^W(t) - \bar{G}^W(t) \cdot r_i(t) \right|. \tag{7}$$

Remember that, the gradient magnitude $G_i^W(t)$ for task i depends on the weighted single-task loss $w_i(t) \cdot L_i(t)$. As a result, the objective in Equation 7 can be minimized by adjusting the task-specific weights w_i. In practice, during training these task-specific weights are updated in every iteration using backpropagation. After every update, the task-specific weights $w_i(t)$ are re-normalized in order to decouple the learning rate from the task-specific weights.

Note that, calculating the gradient magnitude $G_i^W(t)$ requires a backward pass through the task-specific layers of every task i. However, savings on computation time can be achieved by considering the task gradient magnitudes only w.r.t. the weights in the last shared layer.

Different from uncertainty weighting, GradNorm does not take into account the task-dependent uncertainty to re-weight the task-specific losses. Rather, GradNorm tries to balance the pace at which tasks are learned, while avoiding gradients of different magnitude.

3.1.3 Dynamic Weight Averaging

Similarly to GradNorm, Liu et al. [29] proposed a technique, termed **Dynamic Weight Averaging** (DWA), to balance the pace at which tasks are learned. Differently, DWA only requires access to the task-specific loss values. This avoids having to perform separate backward passes during training in order to obtain the task-specific gradients. In DWA, the task-specific weight w_i for task i at step t is set as

$$w_i(t) = \frac{N \exp\left(r_i(t-1)/T\right)}{\sum_n \exp\left(r_n(t-1)/T\right)}, r_n(t-1) = \frac{L_n(t-1)}{L_n(t-2)}, \tag{8}$$

with N being the number of tasks. The scalars $r_n(\cdot)$ estimate the relative descending rate of the task-specific loss values L_n. The temperature T controls the softness of the task weighting in the softmax operator. When the loss of a task decreases at a slower rate compared to other tasks, the task-specific weight in the loss is increased.

Note that, the task-specific weights w_i are solely based on the rate at which the task-specific losses change. Such a strategy requires to balance the overall loss magnitudes beforehand, else some tasks could still overwhelm the others during training. GradNorm avoids this problem by balancing both the training rates and the gradient magnitudes through a single objective (see Equation 7).

3.1.4 Dynamic Task Prioritization

The task balancing techniques in Sections 3.1.1-3.1.3 opted to optimize the task-specific weights w_i as part of a Gaussian likelihood objective [32], or in order to balance the pace at which the different tasks are learned [29], [33]. In contrast, **Dynamic Task Prioritization** (DTP) [40] opted to prioritize the learning of 'difficult' tasks by assigning them a higher task-specific weight. The motivation is that the network should spend more effort to learn the 'difficult' tasks. Note that, this is opposed to uncertainty weighting, where a higher weight is assigned to the 'easy' tasks. We hypothesize that the two techniques do not necessarily conflict, but uncertainty weighting seems better suited when tasks have noisy labeled data, while DTP makes more sense when we have access to clean ground-truth annotations.

To measure the task difficulty, one could consider the progress on every task using the loss ratio $\tilde{L}_i(t)$ defined by GradNorm. However, since the loss ratio depends on the initial loss $L_i(0)$, its value can be rather noisy and initialization dependent. Furthermore, measuring the task progress using the loss ratio might not accurately reflect the progress on a task in terms of qualitative results. Therefore, DTP proposes the use of key performance indicators (KPIs) to quantify the difficulty of every task. In particular, a KPI κ_i is selected for every task i, with $0 < \kappa_i < 1$. The KPIs are picked to have an intuitive meaning, e.g. accuracy for classification tasks. For regression tasks, the prediction error can be thresholded to obtain a KPI that lies between 0 and 1. DTP sets the task-specific weight w_i for task i at step t as

$$w_i(t) = -\left(1 - \kappa_i(t)\right)^{\gamma_i} \log \kappa_i(t). \tag{9}$$

Note that, Equation 9 employs a focal loss expression [64] to down-weight the task-specific weights for the 'easy' tasks. In particular, as the value for the KPI κ_i increases, the weight w_i for task i is being reduced.

DTP requires to carefully select the KPIs. For example, consider choosing a threshold to measure the performance on a regression task. Depending on the threshold's value, the task-specific weight will be higher or lower during training. We conclude that the choice of the KPIs in DTP is not determined in a straightforward manner. Furthermore, similar to DWA, DTP requires to balance the overall magnitude of the loss values beforehand. After all, Equation 9 does not take into account the loss magnitudes to calculate the task-specific weights. As a result, DTP still involves manual tuning to set the task-specific weights.

3.1.5 MTL as Multi-Objective Optimization

A global optimum for the multi-task optimization objective in Equation 4 is hard to find. Due the complex nature of this problem, a certain choice that improves performance for one task could lead to performance degradation for another task. The task balancing methods discussed beforehand try to tackle this problem by setting the task-specific weights in the loss according to some heuristic. Differently, Sener and Koltun [34] view MTL as a multi-objective optimization problem, with the overall goal of finding a Pareto optimal solution among all tasks.

In MTL, a Pareto optimal solution is found when the following condition is satisfied: the loss for any task can be decreased without increasing the loss on any of the other tasks. A **multiple gradient descent algorithm** (MGDA) [65] was proposed in [34] to find a Pareto stationary point. In particular, the shared network weights are updated by finding a common direction among the task-specific gradients. As long as there is a common direction along which the task-specific losses can be decreased, we have not reached a Pareto optimal point yet. An advantage of this approach is that since the shared network weights are only updated along common directions of the task-specific gradients, conflicting gradients are avoided in the weight update step.

3.1.6 Discussion

In Section 3.1, we described several methods for balancing the influence of each task when training a multi-task network. Table 1 provides a qualitative comparison of the described methods. We summarize some conclusions below. (1) We find discrepancies between these methods, e.g. uncertainty weighting assigns a higher weight to the 'easy' tasks, while DTP advocates the opposite. The latter can be attributed to the experimental evaluation of the different task balancing strategies that was often done in the literature using different datasets or task dictionaries. We suspect that an appropriate task balancing strategy should be decided for each case individually. (2) We also find commonalities between the described methods, e.g. uncertainty weighting, GradNorm and MGDA opted to balance the loss magnitudes as part of their learning strategy. In Section 4.5, we provide extensive ablation studies under more common datasets or task dictionaries to verify what task balancing strategies are most useful to improve the multi-tasking performance, and under which circumstances. (3) A number of works (e.g. DWA, DTP) still require careful manual tuning of the initial hyperparameters, which can limit their applicability when dealing with a larger number of tasks.

3.2 Other Approaches

The task balancing works in Section 3.1 can be plugged in to most existing multi-task architectures to regulate the

TABLE 1: A qualitative comparison between task balancing techniques. First, we consider whether a method balances the loss magnitudes (**Balance Magnitudes**) and/or the pace at which tasks are learned (**Balance Learning**). Second, we show what tasks are prioritized during the training stage (**Prioritize**). Third, we show whether the method requires access to the task-specific gradients (**Grads Required**). Fourth, we consider whether the tasks gradients are enforced to be non-conflicting (**Non-Competing Grads**). Finally, we consider whether the proposed method requires additional tuning, e.g. manually choosing the weights, KPIs, etc. (**No Extra Tuning**). For a detailed description of each technique see Section 3.

Method	Balance Magnitudes	Balance Learning	Prioritize	Grads Required	Non-Competing Grads	No Extra Tuning	Motivation
Uncertainty [32]	✓		Low Noise			✓	Homoscedastic uncertainty
GradNorm [33]	✓	✓		✓		✓	Balance learning and magnitudes
DWA [29]		✓					Balance learning
DTP [40]			Difficult				Prioritize difficult tasks
MGDA [34]	✓	✓		✓	✓	✓	Pareto Optimum

task learning. Another group of works also tried to regulate the training of multi-task networks, albeit for more specific setups. We touch upon several of these approaches here. Note that, some of these concepts can be combined with task balancing strategies too.

Zhao et al. [41] empirically found that tasks with gradients pointing in the opposite direction can cause the destructive interference of the gradient. This observation is related to the update rule in Equation 5. They proposed to add a modulation unit to the network in order to alleviate the competing gradients issue during training.

Liu et al. [46] considered a specific multi-task architecture where the feature space is split into a shared and a task-specific part. They argue that the shared features should contain more common information, and no information that is specific to a particular task only. The network weights are regularized by enforcing this prior. More specifically, an adversarial approach is used to avoid task-specific features from creeping into the shared representation. Similarly, [39], [42] added an adversarial loss to the single-task gradients in order to make them statistically indistinguishable from each other in the shared parts of the network.

4 Experiments

This section provides an extensive comparison of the previously discussed methods. In particular, Section 4.1 introduces the employed datasets, followed by the evaluation criterion. Section 4.2 compares the encoder-based architectures, while the decoder-based ones are discussed in Section 4.3. Note that, we first opted for a separate analysis of the encoder-based and decoder-based approaches, as a direct joint comparison is impractical [3]. However, we provide a separate comparison between encoder-based and decoder-based approaches in Section 4.4. Finally, Section 4.5 considers the influence of the task balancing strategies.

4.1 Experimental Setup

Our experiments were conducted on a variety of datasets. As we need to compare a large number of works, including different MTL architectures and task balancing techniques, the datasets were selected separately for the encoder-based approaches, the decoder-based approaches and the task-balancing techniques. In each case, we opted for datasets that provide us with a rich and diverse group of settings and allow us to scrutinize the pros and cons of the methods under consideration. We additionally took into account what datasets were used in the original works. Note that, for every experiment we describe the most important elements of the experimental setup. However, further details to reproduce the reported results can be found in the supplementary materials. Next we discuss the employed datasets, followed by a presentation of the evaluation criterion.

4.1.1 Datasets

The **PASCAL** dataset [66] is a popular benchmark for dense prediction tasks. We use the split from PASCAL-Context [67] which has annotations for semantic segmentation, human part segmentation and semantic edge detection. Additionally, we consider the tasks of surface normals prediction and saliency detection. The annotations were distilled by [42] using pre-trained state-of-the-art models [68], [69].

The **Cityscapes** dataset [70] considers the scenario of urban scene understanding. The train, validation and test set contain 2975, 500 and 1525 images respectively, taken by driving a car in European cities. We consider a number of dense prediction tasks on the Cityscapes dataset: semantic segmentation, instance segmentation and monocular depth estimation. The depth maps were generated with SGM [71].

The **NYUD-v2** dataset [72] considers indoor scene understanding. The dataset contains 795 train and 654 test images annotated for semantic segmentation and monocular depth estimation. Other works have also considered surface normal prediction [24], [42], [44] and semantic edge detection [24], [42] on the NYUD-v2 dataset. The annotations for these tasks can be directly derived from the semantic and depth ground truth. In this work we focus on the semantic segmentation and depth estimation tasks.

The **Taskonomy** dataset [73] contains semi-real images of indoor scenes, annotated for 26 (dense prediction, classification, etc.) tasks. Out of all available tasks, we select scene categorization, semantic segmentation, edge detection, monocular depth estimation and keypoint detection for our experiments. The task dictionary was selected to be as diverse as possible, while still keeping the total number of tasks reasonably low for all required computations. We use the tiny split of the dataset, containing 275k train, 52k validation and 54k test images.

3. For example, some works rely on the use of particular backbones, or were designed for tackling specific task dictionaries, etc.

TABLE 2: MTL benchmarks used in the experiments section. Distilled task labels are marked with *.

Dataset	# Train	# Val	# Test	Seg.	Depth	Inst.	H. Parts	Normals	Saliency	Edges	Keypoints	Scene	Attr.
PASCAL	10,581	1,449	-	✓			✓	✓*	✓*	✓			
Cityscapes	2,975	500	500	✓	✓*	✓							
NYUD-v2	795	654	-	✓	✓			✓		✓			
Taskonomy	275k	52k	54k	✓*	✓					✓*	✓*	✓*	
CelebA	160k	20k	20k										✓

The **CelebA** dataset [74] contains over 200k images of celebrities labeled with 40 facial attribute categories. The training, validation and test set contain 160k, 20k and 20k images respectively. While this dataset is mainly suitable for multi-attribute learning, prior works [34], [36], [37] used it for MTL by treating the prediction of each facial attribute as a separate binary classification task. This universal use of a binary cross-entropy loss across all tasks makes CelebA an interesting setting to compare the loss balancing strategies, thus we chose to also include it in our experiments.

Table 2 gives an overview of the used datasets. We marked tasks for which the annotations were obtained through distillation with an asterik.

4.1.2 Evaluation Criterion

It is not straightforward to validate the overall performance of a model when optimizing for multiple tasks. This is particularly true when each task is evaluated through its own specific metric, e.g. semantic segmentation and depth are evaluated using *mIoU* and *rmse* respectively. Surprisingly, the vast majority of works on MTL provides little insight into how the best model is selected during the validation phase. In this paper, we measure the *multi-task performance* Δ_m as in [42]. For consistency reasons, we encourage future works to adopt the same protocol. In particular, the multi-task performance of model m is defined as the average per-task drop in performance w.r.t. the single-task baseline b:

$$\Delta_m = \frac{1}{T}\sum_{i=1}^{T}(-1)^{l_i}\left(M_{m,i} - M_{b,i}\right)/M_{b,i}, \tag{10}$$

where $l_i = 1$ if a lower value means better performance for metric M_i of task i, and 0 otherwise. The single-task performance is measured for a fully-converged model that uses the same backbone network only for that task.

To achieve a fair comparison, all results were obtained after performing a grid search on the hyperparameters. This ensures that every model is trained with comparable amounts of finetuning. In addition to a performance evaluation, we also include the model resources, i.e. number of parameters and FLOPS, when comparing the multi-task architectures. Finally, a good MTL model should be able to improve over a set of strong single-task baselines. Therefore, all experiments are performed using performant backbones in combination with strong task-specific decoders.

4.2 Encoder-Based Architectures

We first consider the encoder-based architectures [26], [28], [29], [36], [37]. Our experiments are performed on the NYUD-v2, Cityscapes and Taskonomy datasets which include two, three and five tasks respectively. On NYUD-v2, we consider the tasks of semantic segmentation and depth estimation. This pair of tasks is strongly related, since both semantic segmentation and depth estimation reveal similar characteristics about a scene, such as the layout and object shapes or boundaries. Differently, Cityscapes and Taskonomy include a larger and more diverse task dictionary.

TABLE 3: Comparison of the encoder-based architectures for MTL on the NYUD-v2 validation set. For more details about the experimental setup visit Section 4.2.1.

Method	S (IoU)↑	D (rmse)↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)↑
Single task	44.5	60.1	79	383	+ 0.00
MTL baseline	44.4	58.7	**56**	**267**	+ 1.11
MTAN	44.5	58.8	214	1064	+ 1.11
Cross-stitch	**44.6**	58.9	79	383	**+ 1.14**
NDDR-CNN	44.0	**58.5**	101	413	+ 0.80

4.2.1 NYUD-v2

Setup. The encoder is a ResNet-50 model [75] with dilated convolutions [76]. The task-specific heads use an Atrous Spatial Pyramid Pooling (ASPP) [69] module. The results are summarized in Table 3.

Single-Task vs Multi-Task. The MTL baseline improves over the set of single-task networks ($+1.11\%$), while it reduces the number of parameters and FLOPS. Similar results have been reported for other pairs of well-correlated tasks, e.g. depth and flow estimation [77], detection and classification [21], detection and segmentation [2], [22].

Multi-Task Models. Surprisingly, none of the encoder-based architectures shows significant improvements over the MTL baseline in this setup. This observation seems to contradict the results obtained in prior works [26], [28], [29]. However, looking closer we see that the experimental evaluation in those works was limited to the use of weaker backbones, e.g. AlexNet [78], VGG [79], etc. Sharing a ResNet backbone in combination with strong task-specific head units seems to provide a more viable alternative for solving a pair of well-correlated dense prediction tasks, like semantic segmentation and depth estimation.

Discussion. As documented in other works [2], [21], [22], [77], we observe that MTL proves an effective strategy for solving a pair of tasks, provided that they are well-correlated. In this setup, the MTL baseline obtains better results, while using fewer resources compared to the single-task models. Furthermore, we found that the MTL baseline with ResNet backbone and strong task-specific heads outperforms other encoder-based approaches. We encourage researchers to use this model as a baseline for future work.

TABLE 4: Comparison of the encoder-based architectures for MTL on the Cityscapes validation set. The Branched models refer to generating the task groupings with the method from [37]. We refer the reader to Section 4.2.2 for more details about the experimental setup.

Method	S (IoU)↑	I (px)↓	D (px)↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)↑
Single task	65.2	11.7	2.57	138	266	+ 0.00
MTL baseline	61.5	11.8	2.66	**92**	**166**	- 3.33
MTAN	63.1	11.9	2.66	330	676	- 2.67
Cross-stitch	65.1	**11.6**	2.55	140	266	+ 0.42
NDDR-CNN	**65.6**	**11.6**	**2.54**	190	372	**+ 0.89**
Branched-1	62.1	11.7	2.66	107	196	- 2.68
Branched-2	62.7	11.7	2.62	114	211	- 1.84
Branched-3	64.1	**11.6**	2.62	116	213	- 0.96

4.2.2 *Cityscapes*

Setup. As in prior works [32], [34], we use a ResNet-50 encoder with dilated convolutions, followed by a Pyramid Spatial Pooling (PSP) [52] head. Every input image is scaled to 512 x 256 pixels. For the instance segmentation task we reuse the approach from [32], i.e. we consider the proxy task of regressing each pixel to the center of the instance it belongs to. The results can be found in Table 4.

Single-Task vs Multi-Task. In contrast to the results on NYUD-v2 in Section 4.2.1, the MTL baseline reports lower performance when compared against the trio of single-task models (-3.33%). There are several possible explanations for this. First, we consider a larger task dictionary. Second, the semantic segmentation task requires a feature representation that is invariant for instances of the same class, while the instance segmentation task needs to disentangle instances of the same class. It has been shown [30], [42] that a diverse task dictionary is more prone to task interference. Third, the Cityscapes dataset contains more labeled examples compared to NYUD-v2. Prior work [50] observed that the multi-task performance gains can be lower when more annotations are available.

Tree-Based Encoders. First, we compare the performance of the tree-based encoders (i.e. task groupings) generated by Branched Multi-Task Networks against those generated by FAFS. In both cases, we trained all possible task groupings that can be derived from branching the ResNet-50 encoder in the last three ResNet blocks. Figure 10 visualizes the multi-task performance (y-axis) vs number of parameters (x-axis) for the trained architectures. Depending on the available parameter budget, both methods generate a specific task grouping. We visualize these generated groupings as a continuous path in Figure 10, when gradually increasing the parameter budget. Note that, a similar analysis can be made when increasing the number of available floating points operations. We find that, for a given parameter budget, the task groupings from Branched Multi-Task Networks achieve higher performance compared to the ones from FAFS. Furthermore, for a fixed parameter budget, in the majority of cases Branched Multi-Task Networks are capable of selecting the best task groupings w.r.t the 'multi-task performance vs number of parameters' metric. Given the results on the Cityscapes dataset, it seems that the task clustering method from FAFS is better suited for the multi-attribute learning problem, for which it was originally tested.

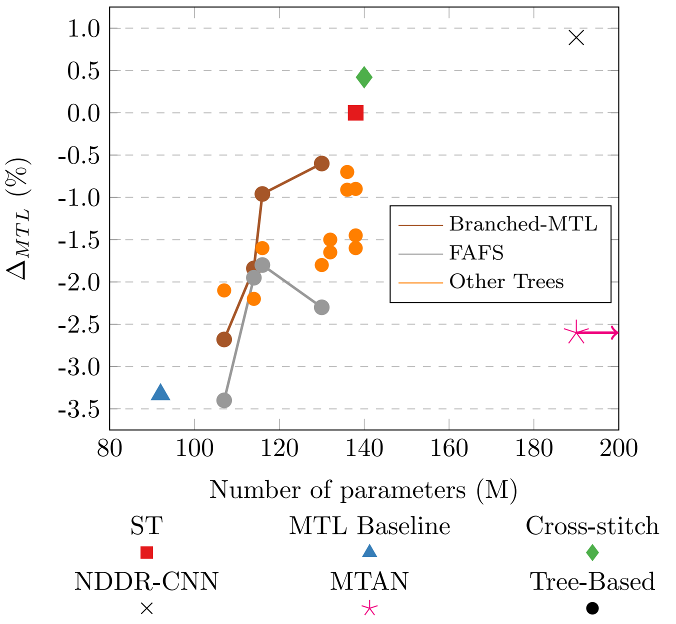


Fig. 10: We compare different encoder-based approaches for MTL on the Cityscapes validation set in terms of number of parameters and multi-task performance.

Second, we compare the tree-based encoders against the single-task networks. We observe that, even with a strong task grouping criterion [37], the tree-based encoders do not improve over the single-task networks. This is to be expected as tree-based encoders are not performance-oriented MTL approaches by design. They rather act as a proxy between a fully-shared MTL model (hard parameter sharing) and a task-specific MTL model (soft parameter sharing), conditioned on the model size. As a result, they are better geared towards resource-constrained environments, where the model size needs to fall within an upper threshold.

Multi-Task Models. Consider again Table 4. We observe that both cross-stitch networks ($+0.42\%$) and NDDR-CNNs ($+0.89\%$) improve over the single-task networks. However, the NDDR-CNN requires more parameters and FLOPS compared to the single-task networks. Differently, the MTAN model decreases the multi-task performance (-2.67%) despite requiring more computational resources compared to the single-task networks in this setup. Table 4 also includes tree-based encoders generated by Branched Multi-Task Networks. The used task groupings can be found in the supplementary materials. As discussed earlier, the performance of the Branched Multi-Task Networks improves when we increase the computational budget - parameters and FLOPS. We see that the Branched MTL-1 model performs on par with the MTAN model (-2.68%), but requires significantly less parameters and FLOPS. We observe that tree-based encoders can provide a viable alternative to other multi-task models when the computational resources are scarce.

Discussion. Depending on the available computational budget, different models can be deployed to maximize the multi-task performance (see Table 4). In general, the encoder-based approaches improve the multi-task performance over the MTL baseline at the cost of more parameters

TABLE 5: Comparison of the encoder-based architectures for MTL on the tiny Taskonomy test set. We use the following task abbreviations: depth estimation (D), semantic segmentation (S), scene categorization (C), edge detection (E) and keypoints detection (K). Note that, the results for edge and keypoints detection were multiplied by a factor of 100 for better readability.

Method	D (L1)↓	S (IoU)↑	C (Top-5)↑	E (L1)↓	K (L1)↓	#P (M)	FLOPS (G)↑	Δ_{MTL} (%)↑
Single task	**0.60**	43.5	**66.0**	0.99	**0.23**	224	136	**+ 0.00**
MTL baseline	0.75	47.8	56.0	1.37	0.34	**130**	**70**	- 22.50
MTAN	0.71	43.8	59.6	1.86	0.40	525	365	-37.36
Cross-stitch	0.61	44.0	58.2	1.35	0.50	224	135	- 32.29
NDDR-CNN	0.66	45.9	64.5	1.05	0.45	258	176	- 21.02
FAFS - 1	0.74	46.1	62.7	1.30	0.39	174	93	- 24.5
FAFS - 2	0.80	39.9	62.4	1.68	0.52	188	101	- 48.32
FAFS - 3	0.74	46.1	64.9	1.05	0.27	196	105	- 8.48
Branched - 1	0.76	47.6	63.3	1.12	0.29	174	93	- 11.88
Branched - 2	0.74	**48.0**	63.6	0.96	0.35	188	101	- 12.66
Branched - 3	0.74	47.9	64.5	**0.94**	0.26	196	105	- 4.93

and FLOPS. Currently, there is no encoder-based model that improves both in terms of computational resources and multi-task performance when compared against the single-task models on Cityscapes. This is different from the results on NYUD-v2, where we found that for a specific pair of well-correlated tasks, i.e. semantic segmentation and depth estimation, the MTL baseline improved the performance while consuming fewer resources at the same time.

4.2.3 Taskonomy

Setup. We reuse the architecture and training setup from [73]: the encoder is based on ResNet-50; a 15-layer fully-convolutional head is used for the pixel-to-pixel prediction tasks. The results can be found in Table 5.

Single-Task vs Multi-Task. Similar to the results reported on Cityscapes in Section 4.2.2, the MTL baseline obtains lower performance compared to the single-task networks, but with more noticeable drops (-22.50%). The reason is arguably the larger and more diverse task dictionary.

Tree-Based Encoders. We compare Branched Multi-Task Networks against FAFS. The candidate branching points are again located after every ResNet block. Since the number of tasks increased to five, it is very expensive to train all task groupings exhaustively, as done on Cityscapes. Instead, we limit ourselves to three architectures that are generated when gradually increasing the parameter budget. The generated task groupings are shown in the supplementary materials. Compared to the results reported on Cityscapes (cf. Figure 10), we find that in this setup the multi-task performance is more susceptible to the employed task groupings, possibly due to task interference. Overall, we observe that using the task grouping criterion from Branched Multi-Task Networks leads to better multi-task performance compared to the task groupings generated by the FAFS approach.

Multi-Task Models. We observe that cross-stitch networks, NDDR-CNNs and MTAN can not handle the larger, more diverse task dictionary: the multi-task performance decreases when using these models, while the number of parameters and FLOPS increases. Branched Multi-Task Networks seem to handle the diverse set of tasks rather positively, when compared against other encoder-based models. More specifically, they are able to strike a good compromise between the multi-tasking performance and the computational resources that are required. We argue that tree-based encoders, like Branched Multi-Task Networks, might be better suited for handling task interference as they explicitly assign dissimilar tasks to separate branches.

Discussion. Jointly tackling a task dictionary that contains five or more tasks proves challenging in practice, as also noted by [30], [42]. Existing encoder-based approaches do not seem to provide an efficient solution to this problem. In particular, a number of MTL models decreases the multi-task performance, while increasing the number of parameters and FLOPS compared to the single-task networks. Tree-based encoders can be more useful in this setup, since they strike a better trade-off between the amount of computational resources and the multi-task performance.

4.2.4 Discussion

We compared several encoder-based architectures across a variety of datasets. In most cases, the encoder-based architectures do not outperform the set of single-task networks. This is particularly the case when we consider a large or diverse task dictionary, e.g. Cityscapes or Taskonomy. Still, for specific pairs of tasks, e.g. depth estimation and semantic segmentation, we can boost the overall performance through MTL. Notably, there was no encoder-based model that consistently outperformed the other architectures. Instead, an appropriate MTL model should rather be decided on a per case basis, also taking into account the amount of available computational resources. Finally, we could not find a principled way to predict in advance the possible gains when using a particular architecture. Encoder-based architectures could benefit from further investigation.

4.3 Decoder-Based Architectures

The decoder-based architectures presented in Section 2.3 were designed to tackle dense prediction tasks. Therefore, we perform our comparison on two dense labeling datasets, NYUD-v2 and PASCAL. As before, the semantic segmentation and depth estimation tasks are considered on the NYUD-v2 dataset. On PASCAL we tackle a larger and more diverse task dictionary that contains semantic segmentation, human parts segmentation, surface normals estimation, semantic edge detection, and saliency estimation.

The following experiments provide a comparison between PAD-Net [24], PAP-Net [44], JTRL [43], ASTMT [42], and MTI-Net [45]. However, a direct comparison between

TABLE 6: Comparison of the decoder-based architectures for MTL on the NYUD-v2 validation set using a single-scale backbone network. Task abbreviations as before.

Method	S (IoU)↑	D (rmse)↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)↑
Single Task	44.5	60.1	79	383	+ 0.00
MTL Baseline	44.4	58.7	56	267	+ 1.11
JTRL	46.4	**50.1**	295	660	+ 10.45
PAP-Net	**50.4**	53.0	**52**	4800	**+ 12.53**
PAD-Net	50.2	58.2	**52**	**256**	+ 7.99

all models is not straightforward. There are several reasons behind this. First, MTI-Net operates on top of a multi-scale feature representation of the input image, which assumes a multi-scale backbone, unlike the other approaches that were originally designed with a single-scale backbone network in mind. Second, JTRL was strictly designed for a pair of tasks, without any obvious extension to the MTL setting. Finally, PAP-Net behaves similarly to PAD-Net, but operates on the pixel affinities to perform the multi-modal distillation through a recursive diffusion process.

Based on these observations, we organize the comparisons as follows. We compare PAD-Net, PAP-Net and JTRL on the NYUD-v2 dataset using a single-scale backbone. We draw a separate comparison between MTI-Net and PAD-Net on NYUD-v2 using a multi-scale backbone. Finally, we conduct a comparison between PAD-Net, ASTMT and MTI-Net on PASCAL using both single-scale and multi-scale backbones to verify how well the decoder-based approaches handle a larger and more diverse task dictionary.

4.3.1 NYUD-v2

Single-Scale Setup. We reuse the setup from Section 4.2.1. The single-tasking models are based on a ResNet-50 with dilated convolutions in combination with an ASPP head to decode the output. The results can be found in Table 6.

Single-Scale Architectures. All decoder-based architectures report significant gains over the single-task networks on NYUD-v2. PAP-Net achieves the highest multi-task performance (+12.53%), but consumes a large number of FLOPS. This is due to the use of task-affinity matrices, which require to calculate the feature correlation between every pair of pixels in the image. A similar improvement is seen for JTRL (+10.45%). JTRL recursively predicts the two tasks at increasingly higher scales. Therefore, a large number of filter operations is performed on high resolution feature maps, leading to an increase in the computational resources. Opposed to JTRL and PAP-Net, PAD-Net does not come with a significant increase in the number of computations. Yet, we still observe a relatively large improvement over the single-task networks (+7.99%). Arguably, PAD-Net shows more promising when it comes to tackling larger task dictionaries.

Multi-Scale Setup. We use an HRNet-18 backbone [80] to produce a multi-scale feature representation of the input image. The task-specific heads simply upsample and concatenate the multi-scale features before feeding them to two consecutive convolutional layers to get the task output. We use the different output scales of the HRNet backbone to perform multi-scale operations. This translates to four scales (1/4, 1/8, 1/16, 1/32). The results are reported in Table 7.

TABLE 7: Comparison of the decoder-based architectures for MTL on the NYUD-v2 validation set using a multi-scale backbone network. Task abbreviations as before.

Method	S (IoU)↑	D (rmse)↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)↑
Single task	33.2	66.7	8	22	+ 0.00
MTL Baseline	32.1	66.8	**4**	**12**	- 1.71
PAD-Net	33.9	65.8	7	44	+ 1.65
MTI-Net	**37.5**	**60.7**	13	20	**+ 10.91**

Multi-Scale Architectures. Different from the results obtained with a single-scale backbone in Table 6, the multi-scale performance decreases for the MTL baseline. We argue that this is due to the smaller task-specific heads that are used in conjunction with HRNet, while before we used the bulkier ASPP heads after ResNet-50 for the decoding. Again, PAD-Net outperforms the single-task networks (+1.65%), but MTI-Net further improves the performance (+10.91%). We conclude that it is beneficial to distill task information at a multitude of scales, instead of a single scale. We also compare the models in terms of the number of parameters and FLOPS. MTI-Net consumes fewer FLOPS compared to the single-task networks. However, the amount of parameters increases. This is due to the use of a rather shallow backbone, and a small number of tasks. As a consequence, the overhead introduced by adding the additional layers in MTI-Net is relatively large compared to the resources used by the backbone. PAD-Net reduces the number of parameters, but increases the number of FLOPS compared to MTI-Net. This is due to the fact that PAD-Net performs the multi-modal distillation at a single higher scale (1/4) with $4 \cdot C$ channels, C being the number of backbone channels at a single scale. Instead, MTI-Net performs most of the computations at smaller scales (1/32, 1/16, 1/8), while operating on only C channels at the higher scale (1/4).

4.3.2 PASCAL

The analysis on NYUD-v2 suggested that both PAD-Net and MTI-Net are promising options to tackle a larger and more diverse set of tasks. More specifically, both models significantly outperform their single-task counterparts with limited computational overhead. We provide an additional comparison on the PASCAL dataset to verify this hypothesis. We additionally include ASTMT into the comparison. Remember that, ASTMT uses a single-tasking approach to MTL that requires a separate forward pass to be performed for every task (see Section 2.4). Below, we present our results using both a single-scale and a multi-scale backbone.

Single-Scale Setup. We reuse the Deeplab-v3+ model from [42]. The backbone is a ResNet-26 model with dilated convolutions. Table 8 summarizes the results.

Single-Scale Architectures. We see that PAD-Net obtains lower multi-task performance compared to the single-task networks (−7.38%) and MTL baseline (−0.76%). Although it performs better or on par for four out of five tasks compared to the MTL baseline, it performs significantly worse on the semantic edge prediction task. This can be attributed to not using skip connections in our model. In contrast, ASTMT seems better suited to handle the larger task dictionary, and performs on par with the single-task networks (−0.11%), while requiring fewer parameters.

TABLE 8: Comparison of the decoder-based architectures for MTL on PASCAL using a single-scale backbone.

Method	Seg (mIoU) ↑	Parts (mIoU) ↑	Sal (mIoU) ↑	Edge (odsF) ↑	Norm (mErr) ↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)
Single Task	**64.9**	57.1	64.2	**71.3**	**14.9**	103	130	**+ 0.00**
MTL Baseline	60.2	54.1	62.1	69.2	17.0	**19**	**24**	- 6.62
ASTMT	64.6	**57.3**	**64.7**	71.0	15.0	31	132	- 0.11
PAD-Net	61.8	56.5	62.1	58.4	16.4	32	36	-7.38

TABLE 9: Comparison of the decoder-based architectures for MTL on PASCAL using a multi-scale backbone.

Method	Seg (mIoU) ↑	Parts (mIoU) ↑	Sal (mIoU) ↑	Edge (odsF) ↑	Norm (mErr) ↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)
Single Task	60.1	60.7	67.2	69.7	**14.6**	20	48	+ 0.00
MTL Baseline	53.6	58.5	65.1	70.6	15.1	**4**	**10**	- 3.70
PAD-Net	53.6	59.6	65.8	72.5	15.3	17	225	-3.08
MTI-Net	**64.3**	**62.1**	**68.0**	**73.4**	14.8	27	31	**+ 2.74**

Multi-Scale Setup. We reuse the HRNet-18 backbone from our earlier experiments on NYUD-v2. The results for this multi-scale backbone can be found in Table 9.

Multi-Scale Architectures. Similar to the results obtained with a single-scale backbone, PAD-Net lowers the performance compared to the single-task networks, but less (-3.08%). Differently, MTI-Net improves the performance ($+2.74\%$), while consuming fewer FLOPS and only slightly more parameters compared to the single-task networks. The consistent findings in the larger task dictionary backup our previous assumption about the importance of performing the multi-modal distillation at a multitude of scales.

4.3.3 Discussion

We compared several decoder-based architectures across two dense labeling datasets. On NYUD-v2, the performance on both semantic segmentation and depth estimation can be significantly improved by employing one of the decoder-based models. However, both PAP-Net and JTRL incur a large increase in the number of FLOPS. MTI-Net and PAD-Net provide more viable alternatives when we wish to limit the number of FLOPS. On PASCAL, a multi-scale approach, like MTI-Net, seems better fitted for increasing the multi-scale performance while retaining the computational resources low. We conclude that the decoder-based architectures obtain promising results on the MTL problem.

4.4 Encoder-Based vs Decoder-Based Architectures

So far, we considered the encoder-based and decoder-based architectures in isolation. In this section, we compare models from both paradigms against each other on NYUD-v2. This is followed by a qualitative discussion, also taking into account the experimental results from Sections 4.2 and 4.3.

4.4.1 NYUD-v2

Setup. We reuse the setup from our earlier experiments in Sections 4.2.1 and 4.3.1. We exclude the decoder-based approaches that were trained with a multi-scale backbone in order to enable a fair comparison. As a consequence, all models use an identical ResNet-50 backbone with dilated convolutions. The results can be found in Table 10.

Discussion. We find that the decoder-based architectures significantly outperform the encoder-based ones in terms of multi-tasking performance on NYUD-v2. We argue that each architecture paradigm serves different purposes. The encoder-based architectures aim to learn richer feature representations of the image by sharing information during the encoding process. Differently, the decoder-based ones focus on improving dense prediction tasks by repeatedly refining the predictions through cross-task interactions. Since the interactions take place near the output of the network, they allow for a better alignment of common cross-task patterns, which in turn, greatly boosts the performance.

TABLE 10: Comparison between encoder-based (*top*) and decoder-based (*bottom*) architectures for MTL on the NYUD-v2 validation set. Task abbreviations as before.

Method	S (IoU)↑	D (rmse)↓	#P (M)	FLOPS (G)	Δ_{MTL} (%)↑
Single task	44.5	60.1	79	383	+ 0.00
MTL baseline	44.4	58.7	56	267	+ 1.11
MTAN	44.5	58.8	214	1064	+ 1.11
Cross-stitch	44.6	58.9	79	383	+ 1.14
NDDR-CNN	44.0	58.5	101	413	+ 0.80
JTRL	46.4	**50.1**	295	660	+ 10.45
PAP-Net	**50.4**	53.0	**52**	4800	**+ 12.53**
PAD-Net	50.2	58.2	**52**	**256**	+ 7.99

4.4.2 Discussion

We saw that the decoder-based architectures can obtain better performance when solving a pair of well-correlated dense prediction tasks, i.e. depth estimation and semantic segmentation. Similarly, a state-of-the-art model, i.e. MTI-Net, was able to improve over its single-task counterparts when solving as much as five dense prediction tasks on the PASCAL dataset (see Table 9). As explained, this can be attributed to the alignment of common cross-task patterns that is key to dense prediction tasks. Differently, we did not observe improvements over the single-task networks when employing the encoder-based architectures to tackle five dense predictions tasks on the Taskonomy dataset (see Table 5). Nonetheless, we believe that the encoder-based architectures are valuable too, as they allow to learn richer image representations. The latter is arguably more important when dealing with multiple classification tasks, where the alignment of common cross-task patterns is obsolete. Finally, based on their complementary behavior, we hope to see both paradigms integrated in future works.

4.5 Task Balancing

We compare the task balancing techniques from Section 3.1. We analyze the use of fixed weights, uncertainty weight-

TABLE 11: Effect of applying different loss balancing strategies for MTL on three different datasets.

(a) CelebA. We learn 40 person attribute classification tasks.

Method	Accuracy (%) ↑
Single-task	91.52
Fixed	**91.91**
Uncertainty	91.62
GradNorm	91.76
DWA	91.84
MGDA	91.30

(b) NYUD-v2. We shift to dense prediction tasks, and learn the semantic segmentation (S) and depth estimation (D) tasks.

Method	S(IoU) ↑	D (rmse) ↓	Δ_{MTL} (%) ↑
Single-task	**44.6**	60.1	+ 0.00
Fixed	44.4	58.7	+ 1.11
Uncertainty	43.8	58.8	+ 0.38
GradNorm	44.2	**58.1**	**+ 1.45**
DWA	44.1	59.1	+ 0.43
MGDA	43.2	57.6	+ 0.71

(c) Cityscapes. We learn the semantic segmentation (S), instance segmentation (I) and depth estimation (D) tasks.

Method	S (IoU) ↑	I (px) ↓	D (px) ↓	Δ_{MTL} (%) ↑
Single-task	**65.2**	**11.7**	2.57	**+ 0.00**
Fixed	61.5	11.8	2.66	- 3.33
Uncertainty	63.3	11.7	2.59	- 1.40
GradNorm	63.4	12.1	**2.52**	- 1.63
DWA	62.8	11.9	2.66	- 2.29
MGDA	63.8	12.1	2.76	- 4.30

ing [32], GradNorm [33], DWA [29] and MGDA [34]. We did not include DTP [40], as this technique requires to define additional key performance indicators (see Section 3.1.4).

The experiments are performed on three datasets: CelebA, NYUD-v2 and Cityscapes. The CelebA dataset requires to jointly solve 40 person attribute classification tasks. All attributes are learned through a binary cross-entropy loss function. Differently, on the NYUD-v2 dataset we learn two dense prediction tasks - semantic segmentation and depth estimation - that use different loss functions. On Cityscapes, the task dictionary size is further increased to three dense prediction tasks, i.e. semantic segmentation, instance segmentation and depth estimation.

4.5.1 CelebA

Setup. We reuse the setup from [34]. A ResNet-18 backbone is used. We attach a separate linear head for every attribute. Different from the setup in [34], the input is rescaled to 224 by 224 pixels. The fixed weights scheme simply sums the task-specific losses together, i.e. the task-specific weights are set to one. The results are shown in Table 11a.

Discussion. The multi-task model trained with fixed weights outperforms the single-task networks (91.91% vs 91.52%). We argue that the additional supervisory signals from other attributes can act as a regularizer for the backbone features. Surprisingly, none of the proposed task balancing strategies improves over the model trained with fixed uniform weights on CelebA, although all methods are very close. We conclude that the uniform weighting scheme is already close to optimal in case of multi-attribute learning, where all individual losses are binary cross-entropy losses. Note that, a similar experiment was performed in [34], but different conclusions were reported. First, they found that the multi-task model trained with fixed weights performs worse when compared against the single-task networks. Second, the authors observed that the loss balancing strategies improve over the uniform weights. In this paper, we find that with sufficient and comparable amounts of hyperparameter tuning both conclusions no longer stand.

TABLE 12: Comparison of state-of-the-art techniques for multi-attribute classification on the CelebA dataset.

Method	Accuracy (%) ↑	#P (M)
LNet+ANet [81]	87	-
Walk and Learn [81]	88	-
MOON [82]	90.94	119.73
Independent Group [83]	91.06	-
MCNN [83]	91.26	-
MCNN-AUX [83]	91.29	-
VGG-16 [36]	91.44	134.41
FAFS [36]	90.79	2.09
GNAS [83]	91.63	7.73
Branched MTL [37]	91.73	7.73
ResNet18 + MGDA [34]	91.75	11.1
ResNet18 + Uniform (Ours)	**91.91**	11.1

We perform an additional comparison with prior work in Table 12. We report state-of-the-art performance on CelebA using an off-the-shelf ResNet-18 model trained with uniform weights. We encourage future works to use this simple, yet effective model as a baseline for multi-attribute learning.

4.5.2 NYUD-v2

Setup. The experiments are performed using the model from Section 4.2.1. The results are reported in Table 11b.

Discussion. The MTL baseline with fixed weights improves over the single-task networks ($+1.11\%$ for Δ_{MTL}, see also Section 4.2.1). GradNorm can further boost the performance by adjusting the task-specific weights in the loss during training ($+1.45\%$). We conclude that uniform weights are suboptimal for training a multi-task model when the tasks employ different loss functions. Similar to GradNorm, DWA tries to balance the pace at which tasks are learned, but does not equalize the gradient magnitudes. From Table 11b, we conclude that the latter is important as well ($+0.43\%$ with DWA vs $+1.45\%$ with GradNorm). Uncertainty weighting results in lower performance compared to GradNorm ($+0.38\%$ vs 1.45%). Uncertainty weighting assigns a smaller weight to noisy or difficult tasks. Since the annotations on NYUD-v2 were not distilled, the noise levels are rather small. When we have access to clean ground-truth annotations, it seems better to balance the learning of tasks rather than lower the weights for the difficult tasks. Finally, MGDA reports lower performance compared to GradNorm ($+0.71\%$ vs $+1.45\%$). MGDA only updates the weights along directions that are common among all task-specific gradients. The results suggest that it is better to allow some competition between the task-specific gradients in the shared layers, as this could help to avoid local minima.

4.5.3 Cityscapes

Setup. We reuse the model from our earlier experiments in Section 4.2.2. The results are shown in Table 11c.

Discussion. The MTL baseline with fixed weights reports lower performance when compared against the single-task networks (-3.33% for Δ_{MTL}, see also Section 4.2.2). Furthermore, methods that balance the loss magnitudes during training (GradNorm -1.63% and Uncertainty -1.40%) generally report higher performance compared to other optimization techniques (Fixed -3.33%, DWA -2.29%). However, this is not the case for MGDA (-4.30%). This is another indication (cf. NYUD-v2 results) that avoiding competing gradients by only back propagating along a common direction in the shared layers does not necessarily improve performance. Different from the results on NYUD-v2, uncertainty weighting achieves higher performance compared to GradNorm (-1.40% vs. -1.63%). We argue that the annotations on Cityscapes contain more noise, because the depth maps were labeled through SGM. Uncertainty weighting addresses this by assigning a smaller weight to tasks with higher homoscedastic uncertainty. Finally, the improvements when using an automated loss balancing scheme are larger on Cityscapes compared to NYUD-v2. We hypothesize that this is due to the larger and more diverse task dictionary. As a consequence, the fixed uniform weights are not well-suited to optimize the network weights. Task balancing methods that try to equalize the gradient or loss magnitudes provide a better alternative in this case.

4.5.4 Discussion

Several experiments were performed to compare the task balancing strategies under a number of different settings. We found that fixed uniform weights are sufficient to train a multi-attribute model. Differently, when the tasks are more diverse and use their own specific loss function, the MTL performance can be increased by using a technique that automatically balances the gradient magnitudes during training. Examples are GradNorm and uncertainty weighting. GradNorm proves the better choice when we have access to clean ground-truth annotations. However, when we are dealing with noisy annotations, uncertainty weighting provides a more viable alternative as it down-weighs the tasks with higher homoscedastic uncertainty. We conclude that there is not a single solution that fits all problems.

Note that, a number of techniques performed worse than anticipated. However, Gong et al. [50] obtained similar results with ours, albeit only for a few loss balancing strategies. Also, Maninis et al. [42] found that performing a grid search for the weights could outperform a state-of-the-art loss balancing scheme. Based on these works and our own findings, we argue that the optimization problem in MTL is still poorly understood (see also Section 3.1.6). We hope that our results stimulate further investigation into this problem.

5 Related Domains

So far, we focused on the application of MTL to jointly solve multiple vision tasks under a fully-supervised setting. In this section, we consider the MTL setup from a more general point-of-view, and analyze its connection with several related domains. The latter could potentially be combined with the MTL setup and improve it, or vice versa.

5.1 Transfer Learning

Transfer learning [84] makes use of the knowledge obtained when solving one task, and applies it to tackle another task. Different from MTL, transfer learning does not consider solving all tasks concurrently. An important question in both transfer learning and MTL is whether visual tasks have relationships, i.e. they are correlated. Ma et al. [85] modeled the task relationships in a MTL network through a Multi-gate Mixture-of-Experts model. Zamir et al. [73] provided a taxonomy for task transfer learning to quantify the task relationships. Similarly, Dwivedi and Roig [58] used Representation Similarity Analysis to obtain a measure of task affinity, by computing correlations between models pretrained on different tasks. Vandenhende et al. [37] then used these task relationships for the construction of tree-based encoders for MTL. Standley et al. [25] also relied on task relationships to define what tasks should be learned together in a MTL setup.

5.2 Neural Architecture Search

The experimental results in Section 4 showed that the success of MTL strongly depends on the use of a proper network architecture. Typically, such architectures are hand-crafted by human experts. However, given the size and complexity of the problem, this manual architecture exploration likely exceeds human design abilities. To automate the construction of the network architecture, **Neural Architecture Search** (NAS) [86] has been proposed in the literature. Yet, most existing NAS works are limited to task-specific models [87], [88], [89], [90], [91]. This is to be expected as using NAS for MTL assumes that layer sharing has to be jointly optimized with the layers types, their connectivity, etc., rendering the problem considerably expensive.

To alleviate the heavy computational burden associated with NAS, a recent work [92] implemented an evolutionary architecture search for MTL, while other researchers explored alternatives, like routing [60], stochastic filter grouping [38], and feature partitioning [62]. These methods do not build the architecture from scratch, but rather start from a predefined backbone network for which a cross-task layer sharing scheme is automatically determined. So far, NAS for MTL focused on how to share features across tasks in the encoder. We hypothesize that NAS could also be applied for the discovery of decoder-based MTL models.

5.3 Other

MTL has been applied to other problems too. This includes various domains, such as **language** [46], [93], [94], **robotics** [95], [96] and **video** [97], [98], as well as with different learning paradigms, such as **reinforcement learning** [99], [100], **self-supervised learning** [101], **semi-supervised learning** [102], [103] and **active learning** [104], [105]. Surprisingly, in the deep learning era, very few works have considered MTL under the semi-supervised or active learning setting. Nonetheless, we believe that these are interesting directions for future research.

For example, a major limitation of the fully-supervised MTL setting that we consider here is the requirement for all samples to be annotated for every task. Prior work [106] showed that the standard update rule from Equation 5 gives suboptimal results if we do not take precautions when annotations are missing. To alleviate this problem, Kim et al. [107] proposed to use an alternate learning scheme by updating the network weights for a single task at a time. A knowledge distillation term [108] is included, in order to avoid loosing relevant information about the other tasks. Differently, Nekrasov et al. [106] proposed to use the predictions from an expert model as synthetic ground-truth when annotations are missing. Although, these early attempts have shown encouraging results, we believe that this problem could benefit from further investigation.

6 Conclusion

In this paper, we reviewed recent methods for MTL within the scope of deep neural networks. First, we presented an extensive overview of both architectural and optimization based strategies for MTL. For each method, we described its key aspects, discussed the commonalities and differences with related works, and presented the possible advantages or disadvantages. Finally, we conducted an extensive experimental analysis of the described methods that led us to several key findings. We summarize some of our conclusions below, and present some possibilities for future work.

First, the performance of MTL strongly varies depending on the task dictionary. Its size, task types, label sources, etc., all affect the final outcome. As a result, an appropriate architecture and optimization strategy should better be selected on a per case basis. Although we provided concrete observations as to why some methods work better for specific setups, MTL could generally benefit from a deeper theoretical understanding to maximize the expected gains in every case. For example, these gains seem dependent on a plurality of factors, e.g. amount of data, task relationships, noise, etc. Future work should try to isolate and analyze the influence of these different elements.

Second, when it comes to tackling multiple dense prediction tasks using a single MTL model, decoder-based architectures currently offer more advantages in terms of multi-task performance, and with limited computational overhead compared to the encoder-based ones. As explained, this is due to an alignment of common cross-task patterns that decoder-based architectures promote, which naturally fits well with dense prediction tasks. Encoder-based architectures still offer certain advantages within the dense prediction task setting, but their inherent layer sharing seems better fitted to tackle multiple classification tasks.

Finally, we analyzed multiple task balancing strategies, and isolated the elements that are most effective for balancing task learning, e.g. down-weighing noisy tasks, balancing task gradients, etc. Yet, many optimization aspects still remain poorly understood. For example, opposed to recent works, our analysis indicates that avoiding gradient competition between tasks can hurt performance. Furthermore, our study revealed that some task balancing strategies still suffer from shortcomings, and highlighted several discrepancies between existing methods. We hope that this work stimulates further research efforts into this problem.

Acknowledgment. The authors acknowledge support by Toyota via the TRACE project and MACCHINA (KU Leuven, C14/18/065). This work is also sponsored by the Flemish Government under the Artificiele Intelligentie (AI) Vlaanderen programme.

Simon Vandenhende received his MSE degree in Electrical Engineering from the KU Leuven, Belgium, in 2018. He is currently studying towards a PhD degree at the Center for Processing Speech and Images at KU Leuven. His research focuses on multi-task learning and self-supervised learning. He won a best paper award at MVA'19. He is co-organizing the first Commands For Autonomous Vehicles workshop at ECCV'20.

Stamatios Georgoulis is a post-doctoral researcher at the CVL group of ETH Zurich. His current research interests include multi-task learning, unsupervised learning, and image generation. Before coming to Zurich, he was a doctoral student at the PSI group of KU Leuven, where he received his PhD under the supervision of Prof. Van Gool and Prof. Tuytelaars, focusing on extracting surface characteristics and lighting from images. Further back, he received his diploma in Electrical and Computer Engineering from the Aristotle University of Thessaloniki and participated in Microsoft's Imagine Cup Software Design Competition. He regularly serves as a reviewer in major Computer Vision and Machine Learning conferences (CVPR, NeurIPS, ICCV, ECCV, etc.) with distinctions.

Marc Proesmans is a research associate at the Center for Processing Speech and Images at KU Leuven, where he leads the TRACE team. He has an extensive research experience on various aspects in image processing, e.g. early vision processes, 3D reconstruction, optical flow, stereo, structure from motion, recognition. He has been involved in several European research projects, such as VANGUARD, IMPROOFS, MESH, MURALE, EUROPA, ROVINA. From 2000-2011, he has been CTO for a KU Leuven spin-off company specialized in special effects for the movie and game industry. He won several prizes, Golden Egg, Tech-Art, a European IST, tech. Oscar nomination, European seal of Excellence. From 2015 he s leading the research on automotive driving as Research expert, and co-founded TRACE vzw to drive the technology transfer from academia to the real R&D environment for autonomous driving in cooperation with Toyota.

Dengxin Dai is a senior scientist working with the Computer Vision Lab at ETH Zurich. In 2016, he obtained his PhD in Computer Vision at ETH Zurich. Since then he is the Team Leader of the TRACE-Zurich team, working on Autonomous Driving in collaboration with Toyota. His research interests lie in autonomous driving, robust perception in adverse weather and lighting conditions, automotive sensors, and learning under limited supervision.

Luc Van Gool is a full professor for Computer Vision at ETH Zurich and the KU Leuven. He leads research and teaches at both places. He has authored over 300 papers. Luc Van Gool has been a program committee member of several, major computer vision conferences (e.g. Program Chair ICCV'05, Beijing, General Chair of ICCV'11, Barcelona, and of ECCV'14, Zurich). His main interests include 3D reconstruction and modeling, object recognition, and autonomous driving. He received several Best Paper awards (eg. David Marr Prize '98, Best Paper CVPR'07). He received the Koenderink Award in 2016 and 'Distinguished Researcher' nomination by the IEEE Computer Society in 2017. In 2015 he also received they 5-yearly Excellence Prize by the Flemish Fund for Scientific Research. He was the holder of an ERC Advanced Grant (VarCity). Currently, he leads computer vision research for autonomous driving in the context of the Toyota TRACE labs in Leuven and at ETH, and has an extensive collaboration with Huawei on the issue of image and video enhancement.

REFERENCES

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *CVPR*, 2015, pp. 3431–3440.

[2] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *ICCV*, 2017, pp. 2961–2969.

[3] D. Eigen, C. Puhrsch, and R. Fergus, "Depth map prediction from a single image using a multi-scale deep network," in *NIPS*, 2014, pp. 2366–2374.

[4] M. Gur and D. M. Snodderly, "Direction selectivity in v1 of alert monkeys: evidence for parallel pathways for motion processing," *The Journal of physiology*, vol. 585, no. 2, pp. 383–400, 2007.

[5] T. Evgeniou and M. Pontil, "Regularized multi–task learning," in *KDD*, 2004.

[6] Y. Xue, X. Liao, L. Carin, and B. Krishnapuram, "Multi-task learning for classification with dirichlet process priors," *JMLR*, vol. 8, no. Jan, pp. 35–63, 2007.

[7] L. Jacob, J.-p. Vert, and F. R. Bach, "Clustered multi-task learning: A convex formulation," in *NIPS*, 2009, pp. 745–752.

[8] J. Zhou, J. Chen, and J. Ye, "Clustered multi-task learning via alternating structure optimization," in *NIPS*, 2011, pp. 702–710.

[9] B. Bakker and T. Heskes, "Task clustering and gating for bayesian multitask learning," *JMLR*, vol. 4, no. May, pp. 83–99, 2003.

[10] K. Yu, V. Tresp, and A. Schwaighofer, "Learning gaussian processes from multiple tasks," in *ICML*, 2005, pp. 1012–1019.

[11] S.-I. Lee, V. Chatalbashev, D. Vickrey, and D. Koller, "Learning a meta-level prior for feature relevance from multiple related tasks," in *ICML*, 2007, pp. 489–496.

[12] H. Daumé III, "Bayesian multitask learning with latent hierarchies," *arXiv preprint arXiv:0907.0783*, 2009.

[13] A. Kumar and H. Daume III, "Learning task grouping and overlap in multi-task learning," 2012.

[14] A. Argyriou, T. Evgeniou, and M. Pontil, "Convex multi-task feature learning," *Machine learning*, vol. 73, no. 3, pp. 243–272, 2008.

[15] J. Liu, S. Ji, and J. Ye, "Multi-task feature learning via efficient l2, 1-norm minimization," in *Uncertainty in Artificial Intelligence*, 2009, pp. 339–348.

[16] A. Jalali, S. Sanghavi, C. Ruan, and P. K. Ravikumar, "A dirty model for multi-task learning," in *NIPS*, 2010.

[17] A. Agarwal, S. Gerber, and H. Daume, "Learning multiple tasks using manifold regularization," in *NIPS*, 2010.

[18] R. K. Ando and T. Zhang, "A framework for learning predictive structures from multiple tasks and unlabeled data," *JMLR*, vol. 6, no. Nov, pp. 1817–1853, 2005.

[19] P. Rai and H. Daumé III, "Infinite predictor subspace models for multitask learning," in *AISTATS*, 2010, pp. 613–620.

[20] R. Girshick, "Fast r-cnn," in *ICCV*, 2015, pp. 1440–1448.

[21] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *NIPS*, 2015, pp. 91–99.

[22] N. Dvornik, K. Shmelkov, J. Mairal, and C. Schmid, "Blitznet: A real-time deep network for scene understanding," in *ICCV*, 2017, pp. 4154–4162.

[23] D. Eigen and R. Fergus, "Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture," in *ICCV*, 2015, pp. 2650–2658.

[24] 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, pp. 675–684.

[25] T. Standley, A. R. Zamir, D. Chen, L. Guibas, J. Malik, and S. Savarese, "Which tasks should be learned together in multi-task learning?" *arXiv preprint arXiv:1905.07553*, 2019.

[26] I. Misra, A. Shrivastava, A. Gupta, and M. Hebert, "Cross-stitch networks for multi-task learning," in *CVPR*, 2016.

[27] S. Ruder, J. Bingel, I. Augenstein, and A. Søgaard, "Latent multi-task architecture learning," in *AAAI*, 2019.

[28] Y. Gao, J. Ma, M. Zhao, W. Liu, and A. L. Yuille, "Nddr-cnn: Layerwise feature fusing in multi-task cnns by neural discriminative dimensionality reduction," in *CVPR*, 2019, pp. 3205–3214.

[29] S. Liu, E. Johns, and A. J. Davison, "End-to-end multi-task learning with attention," in *CVPR*, 2019.

[30] 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.

[31] D. Neven, B. De Brabandere, S. Georgoulis, M. Proesmans, and L. Van Gool, "Fast scene understanding for autonomous driving," in *IV Workshops*, 2017.

[32] A. Kendall, Y. Gal, and R. Cipolla, "Multi-task learning using uncertainty to weigh losses for scene geometry and semantics," in *CVPR*, 2018.

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

[34] O. Sener and V. Koltun, "Multi-task learning as multi-objective optimization," in *NIPS*, 2018.

[35] M. Long, Z. Cao, J. Wang, and S. Y. Philip, "Learning multiple tasks with multilinear relationship networks," in *NIPS*, 2017, pp. 1594–1603.

[36] Y. Lu, A. Kumar, S. Zhai, Y. Cheng, T. Javidi, and R. Feris, "Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification," in *CVPR*, 2017.

[37] S. Vandenhende, S. Georgoulis, B. De Brabandere, and L. Van Gool, "Branched multi-task networks: Deciding what layers to share," *arXiv preprint arXiv:1904.02920*, 2019.

[38] F. J. Bragman, R. Tanno, S. Ourselin, D. C. Alexander, and M. J. Cardoso, "Stochastic filter groups for multi-task cnns: Learning specialist and generalist convolution kernels," in *ICCV*, 2019.

[39] A. Sinha, Z. Chen, V. Badrinarayanan, and A. Rabinovich, "Gradient adversarial training of neural networks," *arXiv preprint arXiv:1806.08028*, 2018.

[40] M. Guo, A. Haque, D.-A. Huang, S. Yeung, and L. Fei-Fei, "Dynamic task prioritization for multitask learning," in *ECCV*, 2018.

[41] X. Zhao, H. Li, X. Shen, X. Liang, and Y. Wu, "A modulation module for multi-task learning with applications in image retrieval," in *ECCV*, 2018.

[42] K.-K. Maninis, I. Radosavovic, and I. Kokkinos, "Attentive single-tasking of multiple tasks," in *CVPR*, 2019, pp. 1851–1860.

[43] Z. Zhang, Z. Cui, C. Xu, Z. Jie, X. Li, and J. Yang, "Joint task-recursive learning for semantic segmentation and depth estimation," in *ECCV*, 2018, pp. 235–251.

[44] Z. Zhang, Z. Cui, C. Xu, Y. Yan, N. Sebe, and J. Yang, "Pattern-affinitive propagation across depth, surface normal and semantic segmentation," in *CVPR*, 2019, pp. 4106–4115.

[45] S. Vandenhende, S. Georgoulis, and L. Van Gool, "Mti-net: Multi-scale task interaction networks for multi-task learning," *arXiv preprint arXiv:2001.06902*, 2020.

[46] P. Liu, X. Qiu, and X. Huang, "Adversarial multi-task learning for text classification," in *ACL*, 2017, pp. 1–10.

[47] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[48] S. Ruder, "An overview of multi-task learning in deep neural networks," *arXiv preprint arXiv:1706.05098*, 2017.

[49] Y. Zhang and Q. Yang, "A survey on multi-task learning," *arXiv preprint arXiv:1707.08114*, 2017.

[50] T. Gong, T. Lee, C. Stephenson, V. Renduchintala, S. Padhy, A. Ndirango, G. Keskin, and O. H. Elibol, "A comparison of loss weighting strategies for multi task learning in deep neural networks," *IEEE Access*, vol. 7, pp. 141 627–141 632, 2019.

[51] M. Teichmann, M. Weber, M. Zoellner, R. Cipolla, and R. Urtasun, "Multinet: Real-time joint semantic reasoning for autonomous driving," in *IV*. IEEE, 2018, pp. 1013–1020.

[52] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *TPAMI*, vol. 37, no. 9, pp. 1904–1916, 2015.

[53] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *TPAMI*, vol. 40, no. 4, pp. 834–848, 2017.

[54] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *CVPR*, 2017, pp. 2881–2890.

[55] Y. Yuan and J. Wang, "Ocnet: Object context network for scene parsing," *arXiv preprint arXiv:1809.00916*, 2018.

[56] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?" in *NIPS*, 2014, pp. 3320–3328.

[57] S. Huang, X. Li, Z. Cheng, A. Hauptmann *et al.*, "Gnas: A greedy neural architecture search method for multi-attribute learning," 2018.

[58] K. Dwivedi and G. Roig, "Representation similarity analysis for efficient task taxonomy & transfer learning," in *CVPR*, 2019, pp. 12 387–12 396.

[59] Y. Yang and T. Hospedales, "Deep multi-task representation learning: A tensor factorisation approach," *arXiv preprint arXiv:1605.06391*, 2016.

[60] C. Rosenbaum, T. Klinger, and M. Riemer, "Routing networks: Adaptive selection of non-linear functions for multi-task learning," in *ICLR*, 2018.

[61] A. Mallya, D. Davis, and S. Lazebnik, "Piggyback: Adapting a single network to multiple tasks by learning to mask weights," in *ECCV*, 2018, pp. 67–82.

[62] A. Newell, L. Jiang, C. Wang, L.-J. Li, and J. Deng, "Feature partitioning for efficient multi-task architectures," *arXiv preprint arXiv:1908.04339*, 2019.

[63] M. Suteu and Y. Guo, "Regularizing deep multi-task networks using orthogonal gradients," *arXiv preprint arXiv:1912.06844*, 2019.

[64] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *ICCV*, 2017.

[65] J.-A. Désidéri, "Multiple-gradient descent algorithm (mgda) for multiobjective optimization," *Comptes Rendus Mathematique*, vol. 350, no. 5-6, pp. 313–318, 2012.

[66] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *IJCV*, vol. 88, no. 2, pp. 303–338, 2010.

[67] X. Chen, R. Mottaghi, X. Liu, S. Fidler, R. Urtasun, and A. Yuille, "Detect what you can: Detecting and representing objects using holistic models and body parts," in *CVPR*, 2014, pp. 1971–1978.

[68] A. Bansal, X. Chen, B. Russell, A. Gupta, and D. Ramanan, "Pixelnet: Representation of the pixels, by the pixels, and for the pixels," *arXiv preprint arXiv:1702.06506*, 2017.

[69] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *ECCV*, 2018, pp. 801–818.

[70] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," in *CVPR*, 2016.

[71] H. Hirschmuller, "Stereo processing by semiglobal matching and mutual information," *TPAMI*, vol. 30, no. 2, pp. 328–341, 2008.

[72] N. Silberman, D. Hoiem, P. Kohli, and R. Fergus, "Indoor segmentation and support inference from rgbd images," in *ECCV*. Springer, 2012, pp. 746–760.

[73] A. R. Zamir, A. Sax, W. Shen, L. J. Guibas, J. Malik, and S. Savarese, "Taskonomy: Disentangling task transfer learning," in *CVPR*, 2018.

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

[75] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016, pp. 770–778.

[76] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," *arXiv preprint arXiv:1511.07122*, 2015.

[77] Y. Zou, Z. Luo, and J.-B. Huang, "Df-net: Unsupervised joint learning of depth and flow using cross-task consistency," in *ECCV*, 2018, pp. 36–53.

[78] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *NIPS*, 2012, pp. 1097–1105.

[79] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[80] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *CVPR*, 2019, pp. 5693–5703.

[81] J. Wang, Y. Cheng, and R. Schmidt Feris, "Walk and learn: Facial attribute representation learning from egocentric video and contextual data," in *CVPR*, 2016.

[82] E. M. Rudd, M. Günther, and T. E. Boult, "Moon: A mixed objective optimization network for the recognition of facial attributes," in *ECCV*, 2016.

[83] E. M. Hand and R. Chellappa, "Attributes for improved attributes: A multi-task network utilizing implicit and explicit relationships for facial attribute classification." in *AAAI*, 2017.

[84] S. J. Pan, Q. Yang *et al.*, "A survey on transfer learning," *TKDE*, vol. 22, no. 10, pp. 1345–1359, 2010.

[85] J. Ma, Z. Zhao, X. Yi, J. Chen, L. Hong, and E. H. Chi, "Modeling task relationships in multi-task learning with multi-gate mixture-of-experts," in *KDD*, 2018, pp. 1930–1939.

[86] T. Elsken, J. H. Metzen, and F. Hutter, "Neural architecture search: A survey." *JMLR*, vol. 20, no. 55, pp. 1–21, 2019.

[87] B. Zoph and Q. V. Le, "Neural architecture search with reinforcement learning," in *ICLR*, 2017.

[88] 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.

[89] H. Pham, M. Y. Guan, B. Zoph, Q. V. Le, and J. Dean, "Efficient neural architecture search via parameter sharing," in *ICML*, 2018.

[90] H. Liu, K. Simonyan, and Y. Yang, "Darts: Differentiable architecture search," *arXiv preprint arXiv:1806.09055*, 2018.

[91] E. Real, A. Aggarwal, Y. Huang, and Q. V. Le, "Regularized evolution for image classifier architecture search," in *AAAI*, 2019.

[92] J. Liang, E. Meyerson, and R. Miikkulainen, "Evolutionary architecture search for deep multitask networks," in *GECCO*, 2018.

[93] D. Dong, H. Wu, W. He, D. Yu, and H. Wang, "Multi-task learning for multiple language translation," in *ACL*, 2015, pp. 1723–1732.

[94] B. McCann, N. S. Keskar, C. Xiong, and R. Socher, "The natural language decathlon: Multitask learning as question answering," *arXiv preprint arXiv:1806.08730*, 2018.

[95] M. Wulfmeier, A. Abdolmaleki, R. Hafner, J. T. Springenberg, M. Neunert, T. Hertweck, T. Lampe, N. Siegel, N. Heess, and M. Riedmiller, "Regularized hierarchical policies for compositional transfer in robotics," *arXiv preprint arXiv:1906.11228*, 2019.

[96] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller, "Learning an embedding space for transferable robot skills," 2018.

[97] A. Diba, M. Fayyaz, V. Sharma, M. Paluri, J. Gall, R. Stiefelhagen, and L. Van Gool, "Holistic large scale video understanding," *arXiv preprint arXiv:1904.11451*, 2019.

[98] R. Pasunuru and M. Bansal, "Multi-task video captioning with video and entailment generation," in *ACL*, 2017, pp. 1273–1283.

[99] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," in *ICML*, 2018.

[100] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: a hierarchical bayesian approach," in *ICML*, 2007, pp. 1015–1022.

[101] C. Doersch and A. Zisserman, "Multi-task self-supervised visual learning," in *ICCV*, 2017, pp. 2051–2060.

[102] Q. Liu, X. Liao, and L. Carin, "Semi-supervised multitask learning," in *NIPS*, 2008, pp. 937–944.

[103] Y. Zhang and D.-Y. Yeung, "Semi-supervised multi-task regression," in *KDD*. Springer, 2009, pp. 617–631.

[104] A. Acharya, R. J. Mooney, and J. Ghosh, "Active multitask learning using both latent and supervised shared topics," in *ICDM*. SIAM, 2014, pp. 190–198.

[105] R. Reichart, K. Tomanek, U. Hahn, and A. Rappoport, "Multitask active learning for linguistic annotations," in *ACL*, 2008, pp. 861–869.

[106] V. Nekrasov, T. Dharmasiri, A. Spek, T. Drummond, C. Shen, and I. Reid, "Real-time joint semantic segmentation and depth estimation using asymmetric annotations," in *ICRA*. IEEE, 2019, pp. 7101–7107.

[107] D.-J. Kim, J. Choi, T.-H. Oh, Y. Yoon, and I. S. Kweon, "Disjoint multi-task learning between heterogeneous human-centric tasks," in *WACV*. IEEE, 2018, pp. 1699–1708.

[108] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[109] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, “Pytorch: An imperative style, high-performance deep learning library,” in *NIPS*, 2019, pp. 8024–8035.

A Supplementary Materials - Introduction

The supplementary materials provide additional information to reproduce the experimental results reported in the paper. The majority of our results were obtained after performing a grid search on the hyperparameter space to ensure fair comparison. Furthermore, all models use pre-trained ImageNet weights. We used PyTorch [109] to implement the different methods. The supplementary materials follow the same structure as the experiments section in the main paper.

Hyperparameters Search As mentioned previously, the majority of our experiments was performed using a grid search on the hyperparameter space. In particular, we tested batches of three different sizes, poly learning rate decay vs step learning rate decay with decay factor 10, and Adam (initial learning rates 2e-4, 1e-4, 5e-5, 1e-5) vs stochastic gradient descent with momentum 0.9 (initial learning rates 5e-2, 1e-2, 5e-3, 1e-3). This results in 48 hyperparameter settings in total. The batch sizes were selected based on prior works and depending on the dataset. The hyperparameter search procedure was repeated for every experiment, unless noted otherwise. The remaining hyperparameters were set in accordance with the original works.

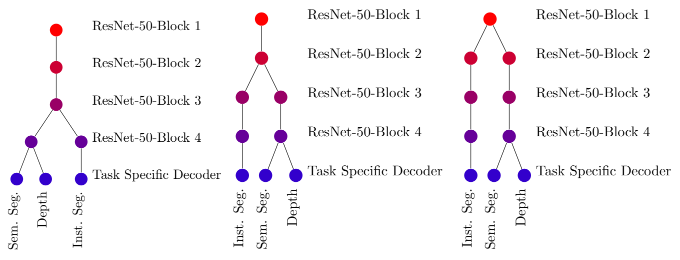
B Encoder-Based Approaches

B.1 NYUD-v2

A ResNet-50 [75] backbone with dilated convolutions [76] is used to encode the images. The task-specific heads use an Atrous Spatial Pyramid Pooling module [69]. The depth task is learned using an L1 loss. A standard cross-entropy loss is used for the semantic segmentation task. The RGB and depth images were randomly scaled with the selected ratio in $\{1, 1.2, 1.5\}$, and randomly horizontally flipped. The hyperparameters were optimized using the grid search procedure described before. We tested batches of size 4, 6 and 8. Note that, our experimental setup is essentially the same as in prior works [24], [44], [45].

B.2 Cityscapes

We followed the experimental setup from [37]. All models were trained following the hyperparameter search procedure described earlier. We show the tree-based encoders from Branched Multi-Task Networks in Figure S1 for completeness.

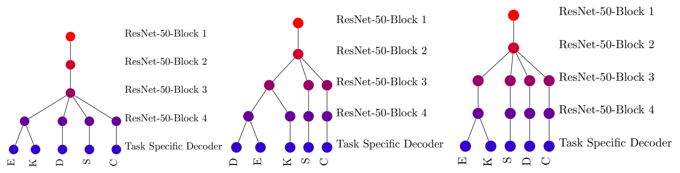


(a) Branched MTL-1 (b) Branched MTL-2 (c) Branched MTL-3

Fig. S1: Task groupings generated with the procedure from Branched Multi-Task Networks [37] on Cityscapes.

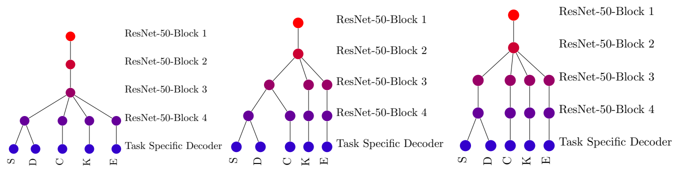
B.3 Taskonomy

We followed the experimental setup from [37]. The task groupings generated by FAFS are shown in Figure S2, while the ones obtained by Branched Multi-Task Networks [37] are shown in Figure S3.



(a) FA - 1 (b) FA - 2 (c) FA - 3

Fig. S2: Task groupings generated by FAFS [36] on Taskonomy.



(a) Branched MTL-1 (b) Branched MTL-2 (c) Branched MTL-3

Fig. S3: Task groupings generated with the procedure from Branched Multi-Task Networks [37] on Taskonomy.

C Decoder-Based Approaches

C.1 NYUD-v2

We reuse the single-task networks and MTL baseline from Section B.1. Note that, the experimental setting, i.e. backbone, loss functions, augmentation strategy, etc., in Section B.1 is essentially the same as the one employed by prior works on decoder-based architectures [24], [44], [45]. This facilitates a direct comparison between the models from Section B.1 and the results reported by the decoder-based architectures on NYUD-v2.

C.2 PASCAL

We reuse the results from ASTMT [42] and MTI-Net [45]. PAD-Net was re-implemented using the code base provided by [42]. In particular, we reuse the backbone, augmentation strategy, and loss weighting. Other hyperparameters were set in accordance with the original work [24].

D Task Balancing

D.1 CelebA

We used a standard ResNet-18 backbone. We attach a separate linear head for every attribute. The input images are rescaled to 224 by 224 pixels, and augmented with random crops and horizontal flips. We follow the hyperparameter search from Section A, including batches of size 32, 128 and 256. The attributes were learned using a binary cross-entropy loss.

D.2 NYUD-v2

We follow the procedure from Section B.1. Additional hyperparameters were set in accordance to the original works.

D.3 Cityscapes

We follow the procedure from Section B.2. Additional hyperparameters were set in accordance to the original works.