Task Switching Network for Multi-task Learning

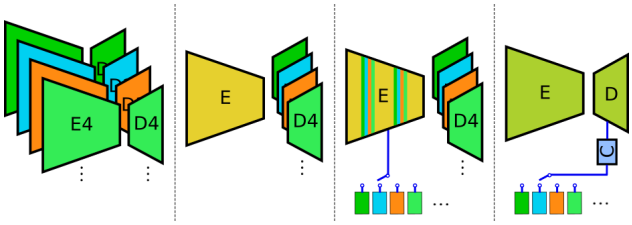
Guolei Sun[1], Thomas Probst[1], Danda Pani Paudel[1], Nikola Popovic[1], Menelaos Kanakis[1],
Jagruti Patel[1], Dengxin Dai[1,2], Luc Van Gool[1]
[1]Computer Vision Laboratory, ETH Zurich, Switzerland
[2]MPI for Informatics, Germany
guolei.sun@vision.ee.ethz.ch

Abstract

We introduce Task Switching Networks (TSNs), a task-conditioned architecture with a single unified encoder/decoder for efficient multi-task learning. Multiple tasks are performed by switching between them, performing one task at a time. TSNs have a constant number of parameters irrespective of the number of tasks. This scalable yet conceptually simple approach circumvents the overhead and intricacy of task-specific network components in existing works. In fact, we demonstrate for the first time that multi-tasking can be performed with a single task-conditioned decoder. We achieve this by learning task-specific conditioning parameters through a jointly trained task embedding network, encouraging constructive interaction between tasks. Experiments validate the effectiveness of our approach, achieving state-of-the-art results on two challenging multi-task benchmarks, PASCAL-Context and NYUD. Our analysis of the learned task embeddings further indicates a connection to task relationships studied in the recent literature.

1. Introduction

The very concept of computer vision is to automatically perform the tasks that a human visual system can do. Even artificial neural networks (ANNs) were also designed as an inspiration from the biological nervous system, such as the human brain. As opposed to the most successful ANNs, the brain and its visual cortex can perform multiple tasks – such as object, parts, and boundary detection or depth and orientation prediction – without any difficulty. Being able to perform multitude of such tasks has allowed humans to efficiently conduct complex activities. In the very spirit, real-world applications like autonomous driving, healthcare, agriculture, manufacturing, cannot be addressed by merely seeking for the perfection on solving individual tasks. It goes without saying that a system capable of performing multiple tasks not only has potential of being efficient in memory usage, computation, and learning speed, but it may also benefit from complementary tasks.



(a) Single-task (b) Multi-task (c) TC Multi-task (d) Our TSNs

Figure 1: **Solutions for multi-task learning.** (a) Every task is solved by training an individual network, *i.e.*, using an independent encoder-decoder pair for each task. (b) General multi-task solutions are built on sharing the encoder and maintaining separate decoders for each task. (c) Task-conditional (TC) multi-task solutions [15, 24] are built on sharing partial parameters of the encoder (task-specific modules also exist), and using separate decoders for each task. (d) In the proposed Task Switching Networks (TSNs), *all* parameters of a single encoder-decoder pair are shared, and a small task embedding network $\mathcal{C}$ facilitates switching between different tasks. *Best viewed in color.*

To tackle multi-task learning (MTL), different solutions have been proposed. Encoder-based methods [18, 27, 23] focus on the encoder, by enhancing the representation capability of architectures so that both shared and task-specific information can be encoded, while decoder-based approaches [47, 44] explore techniques mainly on the decoder part, to better refine the encoder features for specific tasks. Optimization-based methods [6, 17, 37] explicitly target on *task interference* or *negative transfer* issue from optimization perspective, by re-weighting the loss or re-ordering the task learning. In general, these methods follow the structure as Fig. 1 (b). Recently, another direction for MTL emerged, termed task-conditional (TC) multi-tasking [24, 15], shown in Fig. 1 (c). They perform separate pass within the MTL model and activate a set of task-specific modules for each task. The task-specific modules

are used to adapt the network for corresponding tasks. As this setting has many practical use cases [15, 24], our proposed TSNs also follow it and execute one task each time.

Despite that promising results are achieved, the existing methods [44, 24, 15] do not scale well with the number of tasks since they require a large number of task-specific parameters (modules). This can be seen in Fig. 1 (b) and (c), where task-specific decoders or modules (in the encoder) scale with the number of tasks. Additionally, even though task-specific modules minimize adverse interactions amongst tasks, they also minimize positive interactions, i.e., inductive bias [5]. Motivated by these, we propose Task Switching Networks (TSNs). TSNs share *all* parameters among all tasks and do not require *any* task-specific modules (parameters). Hence, our network is simple and has constant size independent of the number of tasks, while still enabling task interactions. We argue that our motivation is also consistent with the widely accepted view in neurobiology, that the visual cortex does not have *separate modules* for different tasks [20, 26].

More specifically, our task-switching networks solve multiple tasks by switching between them – performing one task at a time, and follows a task-conditional single-encoder-single-decoder architecture. As shown in Fig. 1 (d), the task switching is accomplished by employing a small network to learn task-specific embeddings from task encodings, and the behaviour of the decoder is adapted by conditioning it on those embeddings. In practice, we condition only the decoder (U-Net), in a hope that the encoder learns the concept of 'thought space' [36] as it is forced to be task-agnostic. This way, the encoder features can also be reused to efficiently perform multiple tasks in series, which is not possible in encoder-based conditioning [24, 15].

Interestingly, the task embedding network offers some insight into the relationships between tasks. During training, a latent embedding for each task is learned together with its corresponding mapping to the conditioning parameters in each decoder layer. Though there are still many open questions in the study of task relationships and multi-task learning [40], we observe that the structure of our task embeddings resemble the task relationships reported in [48].

To summarize, in this work we study multi-task networks without task-specific parameters, and investigate the behaviour with regards to efficiency, optimization, and accuracy. We state our key contributions as follows.

- We introduce Task-Switching Networks, an efficient yet simple architecture for multi-task learning.
- We demonstrate that conditioning a single shared decoder can outperform multi-decoder methods even on heterogeneous tasks such as segmentation and regression.
- We adopt a small embedding network to learn task conditioning, facilitating optimization and offering insights into relationships between tasks.

2. Related Works

Multi-task learning (MTL). MTL is concerned with learning multiple tasks simultaneously, while exerting shared influence on model parameters. The potential benefits are manifold, and include speed-up of training or inference, higher accuracy, better representations, as well as lower parameters or higher efficiency. A comprehensive survey on architectures, optimization and other aspects of MTL can be found in [7]. Many MTL methods perform multiple tasks by a single forward pass, using shared trunk [18, 3, 23, 43, 22, 8], cross talk [27], or prediction distillation [47, 50, 51, 44] architectures. A recent work of MTI-Net [44] follows this direction and proposes to utilize the task interactions between multi-scale features. Another stream of MTL methods are based on task-conditional networks [15, 24], which perform a separate forward pass and activate some task-specific modules as well as shared modules, for each task. As mentioned in [15], this setting is useful for many real-world setups. Hence, we follow this direction and propose TSNs. In stark contrast to conditioning the shared encoder, as done in [24, 15, 31, 2, 52, 41], we instead learn an unconditioned encoder, and condition only one single unified decoder for all tasks. To the best of our knowledge, our network is the first MTL method that does not require task-specific branching to multiple decoders, but rather shares *all* network parameters and a single conditional unified decoder for all tasks.

Conditioning strategy. In the context of MTL, [41] propose a heuristic masking of features to induce partially shared subnetworks for each task. On the other hand, features can be modulated by introducing task-specific projections [52], residual adapters [31, 32], attention mechanisms [24] or parametrized convolutions [15], while the original backbone network is shared among all tasks. Motivated by the successful application of adaptive normalization strategies in the context of domain adaptation [21], image generation [4, 16], style transfer [13], and super-resolution [46], we explore task-conditioned affine projections after instance normalization (IN) [42] for MTL. Inspired by the style-based generator of [16], the affine parameters are generated from a latent vector representing a desired task. Note that a concurrent work of [30] proposes a new task of CompositeTasking by fusing tasks spatially (pixel-wise), based on task-conditioned BatchNorm (BN) [14].

Task relationships and embedding. Knowledge on the relationships between tasks is crucial for many aspects of machine learning, including multi-task, transfer, or meta-learning. Including the seminal study by *Zamir et al.* [48], many recent works shed light on such task relationships for

transfer learning and multi-tasking [10, 38, 39, 9, 49] by means of computation. Based on the assumption that tasks can be meaningfully taxonomized, this structure can be represented using task embeddings in a high-dimensional space. Nevertheless, such embeddings are mostly explored in meta-learning literature [1, 35, 19], and very little from the multi and transfer learning perspective [49]. Note that the task embeddings [1, 35, 19] of meta-learning are indeed inspirational, but using the mechanisms of meta-learning in the context of MTL is however not straightforward, because they are based on fundamentally different assumptions. We believe it is worth studying the connection of task relationships, embeddings, and multi-task learning in more detail [7], and we see our work as a step in this direction.

3. Problem Formulation

We begin by formally introducing the multi-task learning problem from the perspective of network architecture. In our formulation we consider sequential multi-tasking, *i.e.* solving one task per forward pass, as done in recent MTL techniques [24, 15]. First, we are going to give a formal definition of multi-task networks and data sets.

Definition 3.1 (Multi-task network) *Given a set of tasks $\mathcal{T}$ to be performed on images $\mathcal{X} = [0,1]^{h\times w\times 3}$. For the sake of simplicity, let the output type $\mathcal{Y} = [0,1]^{h\times w\times c}$ be equal for all tasks. We define $f_{\boldsymbol{\theta}} : \mathcal{X}\times\mathcal{T} \rightarrow \mathcal{Y}$ to be a multi-tasking network with parameters $\boldsymbol{\theta}$, that performs one given task τ at a time on a given image. Further, let $\mathcal{F}_{\boldsymbol{\theta}}$ be the set of all multi-tasking networks with parameter set $\boldsymbol{\theta}$.*

Definition 3.2 (Multi-task data set) *Let us denote a multi-task data set as $\mathcal{D}_\tau = \{(\boldsymbol{I}_n, \boldsymbol{y}_n^\tau)\}_N$, with tasks $\tau \in \mathcal{T}$, and $\boldsymbol{y}_n^\tau$ as the ground-truth associated to task τ for image $\boldsymbol{I}_n$.*

Now we can define our goal for MTL as finding a multi-task network with a small set of parameters, that is able to solve all tasks with an accuracy that is close to or better than a single-task baseline. We measure the achievement of this goal by means of the relative performance drop.

Definition 3.3 (Relative performance drop) *Given a metric m_τ to evaluate the performance for task τ, we define the relative performance drop from a baseline m_τ^b as $\Delta_\tau(m_\tau, m_\tau^b) = s_\tau \frac{m_\tau - m_\tau^b}{m_\tau^b}$, where $s_\tau \in \{-1, 1\}$ indicates if higher values are better, or vice versa.*

We can now formalize our problem statement as follows.

Problem 3.4 (Parameter-efficient MTL) *Given a validation set of labelled images $\mathcal{D}_\mathcal{T}$ for tasks set $\mathcal{T}$, with corresponding loss functions ℓ_τ, and expected validation losses $\bar{\ell}_\tau^S$ of single-task baselines, we wish to find a multi-tasking network with the least number of parameters, while bounding the relative performance drop by Δ_τ, as follows.*

$$\begin{aligned} \min_{f\in\mathcal{F}_{\boldsymbol{\theta}},\boldsymbol{\theta}} \quad & |\boldsymbol{\theta}|, \\ \text{s.t.} \quad & \mathop{\mathbb{E}}_{(\boldsymbol{I},\boldsymbol{y}_\tau)\sim\mathcal{D}_\tau} [\ell_\tau(f(\boldsymbol{I},\tau),\boldsymbol{y}_\tau)] \leq (1+\Delta_\tau)\bar{\ell}_\tau^S, \\ & \forall \tau \in \mathcal{T}. \end{aligned} \tag{1}$$

While this is fundamentally a problem of architecture search that could be solved using neural architecture search [11] or combinatorial optimization [40], we instead develop our solution from an analysis of shared parameters.

With slight abuse of notation, we aim to learn a function $f(\boldsymbol{I}_n, \boldsymbol{\theta}_s, \boldsymbol{\theta}_\tau, \tau) = \boldsymbol{y}_n^\tau$, for all $n \in [1, N]$ and $\tau \in \mathcal{T}$. Here, $\boldsymbol{\theta}_s$ denotes the shared parameters across all tasks, while $\boldsymbol{\theta}_\tau$ represents the task-specific parameters. Let $\boldsymbol{\theta}$ denote the total parameters for T ($|\mathcal{T}|$) tasks, given by

$$\boldsymbol{\theta} = \boldsymbol{\theta}_s \cup \bigcup_{\tau\in\mathcal{T}} \boldsymbol{\theta}_\tau. \tag{2}$$

In addition to learning task-specific conditioning parameters, existing MTL methods [31, 2, 15, 24] also learn one or more task-specific convolution layer(s) after branching out to their respective output head. This is also necessary to cater for the output types of different tasks. For the simplicity of analysis, we assume that the number of parameters for all tasks is a constant, and we have the condition,

$$|\boldsymbol{\theta}_s \cup \boldsymbol{\theta}_\tau| \approx \boldsymbol{c}, \forall\, \tau \in \mathcal{T}. \tag{3}$$

Combining Eq. 2 and Eq. 3, we get

$$|\boldsymbol{\theta}| = T\boldsymbol{c} - (T-1)|\boldsymbol{\theta}_s|. \tag{4}$$

From Eq. 4, it is obvious that the total number of parameters for a MTL approach with T tasks is inversely correlated with the number of shared parameters $|\boldsymbol{\theta}_s|$. In the extreme case of single-task setting, where each task is solved by an individual network, without sharing any parameters across tasks, the total parameters are given by $|\boldsymbol{\theta}| = T\boldsymbol{c}$. Hence, existing approaches seek to increase the number of shared parameters $\boldsymbol{\theta}_s$, while reducing *task interference* [18, 52, 24] and maintaining the performance for all tasks. In this paper, we pursue a challenging goal: sharing as many parts of the network as possible across all tasks, up to the point that *all* parameters are shared, and the network becomes independent of the number of tasks T,

$$\boldsymbol{\theta}_\tau = \varnothing, \forall\, \tau \in \mathcal{T} \;\Rightarrow\; \boldsymbol{c} = |\boldsymbol{\theta}_s| = |\boldsymbol{\theta}|. \tag{5}$$

Next, we explain our solution to achieve this goal.

4. Method

In this section, we first present the proposed task switching networks in detail. We further explain task embedding learning using our network.

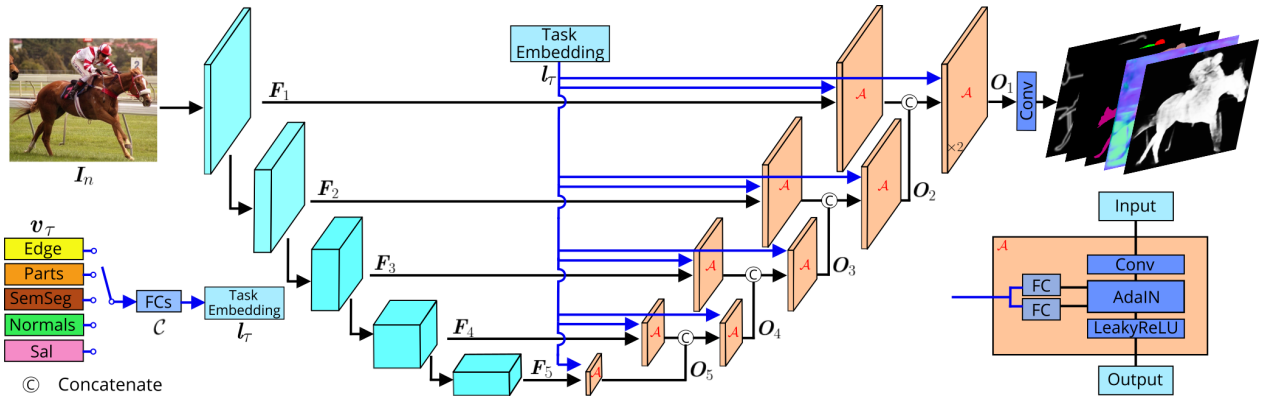


Figure 2: **Task Switching Network overview.** Our network performs multi-tasking by switching between tasks using a conditional decoder. Following U-Net [33], our encoder takes the image $\boldsymbol{I}_n$ and extracts features $\boldsymbol{F}_i$ at different layers. As second input, our network takes a task encoding vector $\boldsymbol{v}_\tau$, selecting task τ to be performed. A small task embedding network $\mathcal{C}$ maps each task to a latent embedding $\boldsymbol{l}_\tau$, that conditions the decoder layers along the blue paths, using module $\mathcal{A}$ [16]. The output is computed by conditioning and assembling encoder features $\boldsymbol{F}_i$ and decoder features $\boldsymbol{O}_i$ in a bottom up fashion.

4.1. Task Switching Network

As laid out in the introduction, we design our Task Switching Networks on the premise that all network parameters should be shared, in order to provide an efficient solution to Problem 3.4. However, as illustrated in Fig. 1, parameter sharing in MTL techniques in literature are limited to the encoder and parts of the decoder, and omit the potential of sharing the *complete* decoder. Moreover, state-of-the-art methods [24, 15] switch tasks by activating task-specific modules. To avoid such additional parameters for each task, we introduce task switching, by taking a task condition as an additional input to the network. To this end, we associate each task τ with a task-condition $\boldsymbol{v}_\tau \in \mathbb{R}^d$.

The input to our model therefore is a pair of image and task-encoding vector, *i.e.*, $(\boldsymbol{I}_n, \boldsymbol{v}_\tau)$, which represents conducting task τ on image $\boldsymbol{I}_n$. Our backbone encoder takes the image $\boldsymbol{I}_n$ and extracts features $\boldsymbol{F}_i$ at different layers. A small task embedding network $\mathcal{C}$ with a few fully connected layers maps each task to a latent embedding $\boldsymbol{l}_\tau$, that is used to condition the decoder layers using a module similar to StyleGAN [16]. The output is then computed by conditioning and assembling encoder features $\boldsymbol{F}_i$ and decoder features $\boldsymbol{O}_i$ bottom-up along the feature pyramid.

In our discussion, the dense prediction tasks (*i.e.*, *edge detection*, and *semantic segmentation*) are considered if not specifically stated, following [24, 15]. In the following, we describe the architectural details of our network.

4.1.1 Network Architecture

As shown in Fig. 2, our network is based on a simple U-Net architecture [33]. The encoder is a ResNet-based [12] backbone pre-trained on ImageNet [34], following existing MTL approaches [24, 15]. Let layer 1 denote the first convolution layer with kernel size of (7×7), and layer 2 to layer 5 represent conv2_x to conv5_x (notation in [12]) of the backbone, respectively. The outputs of layer 1 to layer 5 are $\boldsymbol{F}_1$ to $\boldsymbol{F}_5$. From layer j to layer $j+1$, the spatial resolution of the encoder feature maps is reduced by half.

For the decoder, we follow a similar structure as U-Net [33], collecting features from layer 5 to layer 1. Specifically, at layer j $(j \leq 4)$, the corresponding feature maps $\boldsymbol{F}_j$ from the encoder first pass through a conditional convolution module $\mathcal{A}$, then concatenate the features (after upsampling) from layer $j+1$, and finally pass through another instance of $\mathcal{A}$. For the highest layer $(j = 5)$, the feature maps go through module $\mathcal{A}$ only once since there is no higher layer. As shown in Fig. 2, module $\mathcal{A}$ transforms input features to new features based on the embedding vector $\boldsymbol{l}_\tau$, representing the specific task. Let $\boldsymbol{O}_j$ be the output of the decoder from layer j, which is given by

$$\boldsymbol{O}_j = \begin{cases} \mathcal{A}([\mathcal{U}(\boldsymbol{O}_{j+1}), \mathcal{A}(\boldsymbol{F}_j, \boldsymbol{l}_\tau)], \boldsymbol{l}_\tau), & \text{for } j \leq 4, \\ \mathcal{A}(\boldsymbol{F}_j, \boldsymbol{l}_\tau), & \text{for } j = 5 \end{cases}, \tag{6}$$

where $[\cdot, \cdot]$ denotes the concatenation of two features tensors along the channel dimension and $\mathcal{U}(\cdot)$ is a upsampling operation, which is omitted in Fig. 2 for simplicity.

The output feature $\boldsymbol{O}_1$ from decoder layer 1 has the same resolution as the original image and is passed to a convolution layer to make predictions for different tasks. As discussed previously, different tasks can either share a common convolution layer (*i.e.* a single head shared by all tasks), or have separate convolution layers (different heads for different tasks). In the interest of avoiding task-specific parameters, we opt for the choice that a single head is used by all tasks. To this end, we simply choose the number of output channels as the largest number channels needed for different tasks. Take PASCAL-Context [28] (the most

popular benchmark in MTL) as an example, the number of output channels needed among *edge detection*, *parts segmentation*, *semantic segmentation*, *normals*, and *saliency detection* are 1, 7, 21, 3, and 1, respectively. So we choose 21 output channels, which fits for *semantic segmentation*. For other tasks, we simply conduct adaptive average pooling along the channels to obtain the predictions matching the corresponding tasks. The elegance of sharing a head across tasks is that exactly a single and neat network is used to solve all tasks. Our experiments in fact support this approach, since we found that sharing one head performs competitively to using separate head for different tasks.

In the following, we describe the two key components of task switching networks, that facilitate the conditioning.

Conditional Convolution Module. The goal of this module (block $\mathcal{A}$ in Fig. 2) is to adjust feature representations from the encoder – that are shared by all tasks – to new features that serve the desired task. As mentioned above, to conduct task τ, the corresponding task-condition vector $\boldsymbol{v}_\tau$ is transformed by the embedding network $\mathcal{C}$ to obtain the task-specific latent vector $\boldsymbol{l}_\tau$, which is then passed to module $\mathcal{A}$, inspired by [16]. Let $\boldsymbol{x} \in \mathcal{R}^{1\times c_1\times h\times w}$ denote the input feature to module $\mathcal{A}$, where c_1, h and w represent number of channels, height and width of the feature map, respectively. Module $\mathcal{A}$ then works as follows. First, $\boldsymbol{x}$ is processed by a convolution layer $\hat{\boldsymbol{x}} = \boldsymbol{x} * \boldsymbol{W}$ with filter weights $\boldsymbol{W}$, generating $\hat{\boldsymbol{x}} \in \mathcal{R}^{1\times c_2\times h\times w}$. At the same time, $\boldsymbol{l}_\tau$ is transformed by two fully connected layers with weight matrices $\boldsymbol{W}_\gamma \in \mathbb{R}^{d\times c_2}$ and $\boldsymbol{W}_\beta \in \mathbb{R}^{d\times c_2}$, to form the normalization coefficients $\gamma \in \mathbb{R}^{1\times c_2}$ and $\beta \in \mathbb{R}^{1\times c_2}$, for the subsequent AdaIN. For feature $\hat{\boldsymbol{x}}$, AdaIN performs the normalization following,

$$\textbf{AdaIN}(\hat{\boldsymbol{x}}, \beta, \gamma) = \gamma\frac{(\hat{\boldsymbol{x}} - \mu)}{\sqrt{\sigma^2}} + \beta, \tag{7}$$

where β and σ^2 are the mean and variance of $\hat{\boldsymbol{x}}$, which are statistics computed according to instance normalization [42]. In summary, module $\mathcal{A}$ performs the operation

$$\mathcal{A}(\boldsymbol{x}, \boldsymbol{l}_\tau) = \boldsymbol{l}_\tau\boldsymbol{W}_\gamma\frac{(\boldsymbol{x} * \boldsymbol{W} - \mu)}{\sqrt{\sigma^2}} + \boldsymbol{l}_\tau\boldsymbol{W}_\beta. \tag{8}$$

Task embedding network. Recall that each task is associated with a unique task-condition vector $\boldsymbol{v}_\tau \in \mathbb{R}^d$, and the TSNs switch between tasks by feeding different $\boldsymbol{v}_\tau$ to the task embedding network $\mathcal{C}$, shown in the *left* of Fig. 2. The embedding network $\mathcal{C} : \mathbb{R}^d \to \mathbb{R}^d$ learns to embed the task τ in a latent space $\boldsymbol{l}_\tau = \mathcal{C}(\boldsymbol{v}_\tau)$, from which the AdaIN coefficients of Eq. 7 are generated for each module $\mathcal{A}$. In principle, there are many choices for the initialization of these vectors. Specifically, we investigate embedding dimensions d, with orthogonal $\boldsymbol{v}_\tau$ (binary vector) given by

Table 1: **Task switching performance.** Our TSNs perform competitively with single tasking and multi-tasking baselines, with substantially smaller model sizes. Optimal performance is observed when *all* parameters are shared via our task embedding module (INs+TE).

Method	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	Δ_m%↓	# params
Single-task	71.3	64.3	55.3	16.3	62.9	-	88.7M
Multi-decoder	72.2	55.4	55.5	16.8	59.1	4.32	43.9M
Ours (BNs)	71.6	55.9	54.1	16.7	60.0	4.38	**17.7M**
Ours (INs)	70.7	62.8	54.6	16.8	63.1	1.43	**17.7M**
Ours (INs+TE)	70.6	64.2	55.0	16.3	63.3	**0.30**	18.3M

$$\boldsymbol{v}_{\tau_1}^\intercal\boldsymbol{v}_{\tau_2} = \begin{cases} \frac{d}{T}, & \text{if } \tau_1 = \tau_2 \\ 0, & \text{otherwise} \end{cases}, \quad \boldsymbol{v}_{\tau_1,\tau_2} \in \mathbb{R}^d, \tag{9}$$

and Gaussian random vectors $\boldsymbol{v}_\tau \sim \mathcal{N}(\mathbf{0}_d, \text{diag}(\mathbf{1_d}))$ [16]. The results are reported in §5.1.

5. Experiments

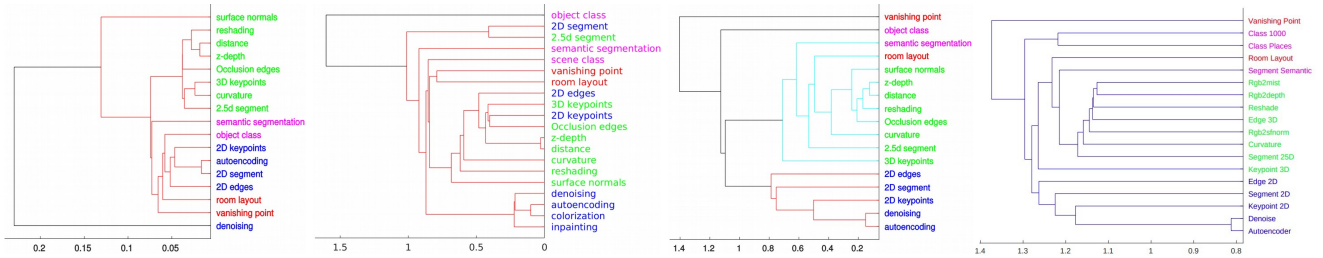
Overview . Following existing works [24, 15], we focus our MTL experiments on dense prediction tasks. In particular, we use the PASCAL-Context [28] dataset, which contains a total of 10,103 images, for the five tasks of *edge detection (Edge)*, *semantic segmentation (SemSeg)*, *human parts segmentation (Parts)*, *surface normals (Normals)*, and *saliency detection (Sal)*. We further evaluate and compare our approach on the NYUD dataset [37], which is comprised of 1,449 images of indoor scenes and comes with annotations for the four tasks of *edge detection*, *semantic segmentation*, *surface normals*, and *depth estimation (Depth)*.

Evaluation metric. We use standard evaluation metrics, following [24, 15, 45]. Specifically, to evaluate the predictive performance for each task, we use the optimal dataset F-measure (odsF) [25] for *edge detection*, mean intersection over union (mIoU) for *semantic segmentation*, *human parts segmentation*, and *saliency*, mean error (Error) for *surface normals*, and root mean square error (RMSE) for *depth*. In order to compare to a multi-task approach m, we average the relative performance drop (see Definition 3.3), with respect to the single-task baseline b over all tasks: $\Delta_m = \frac{1}{T}\sum_{\tau\in\mathcal{T}}\Delta_\tau(p_{m,\tau}, p_{b,\tau})$, where $p_{m,\tau}$ and $p_{b,\tau}$ are the metrics for task τ for the multi-task method m and for single-task baseline b, respectively.

Network configuration. We employ the ResNet-18 backbone with the architecture introduced in §4.1 for all of our experiments, unless stated otherwise. The task embedding network $\mathcal{C}$ contains 8 fully connected layers of width d. Our method is implemented in *PyTorch* [29] and experiments were conducted on NVIDIA GPUs.

5.1. Ablation study

Study on module sharing. We compare our method with various baselines in Table 1. All approaches use the same network architecture and are trained with the same hyperparameters, to ensure fair comparisons. The details of all



(a) *Zamir et al.* [48] (b) Ours (c) *Dwivedi et al.* [10] (d) *Song et al.* [39]

Figure 3: **Task embedding relationships.** We analyse the similarity of our task embeddings after training our network with 20 tasks on a small subset of the Taskonomy dataset [48]. The hierarchical clustering of task affinities from our learned embeddings (b), reveals an interesting similarity to the relationships found by the compared methods (a,c,d).

Table 2: **Impact of task embedding strategy.** The designed task embedding is robust against different choices for the task encoding $\boldsymbol{v}_\tau$, as well as for the dimensionality d of the embedding network $\mathcal{C}$.

Type	d	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	$\Delta_m\%$↓
Orthogonal	50	70.8	63.6	55.2	16.3	63.4	0.32
	100	70.6	64.2	55.0	16.3	63.3	**0.30**
	150	70.5	64.3	54.9	16.3	63.2	0.38
	250	70.5	63.8	54.8	16.4	63.1	0.75
Gaussian	50	70.8	64.1	55.1	16.3	63.0	**0.30**
	100	70.3	63.2	54.4	16.5	63.1	1.22
	150	70.7	63.6	54.8	16.3	63.4	0.44

Table 3: **Impact of network architecture.** The designed task embedding is robust against various backbones.

Backbone	Method	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	$\Delta_m\%$↓
ResNet-18	Single-task	71.3	64.3	55.5	16.3	62.9	-
	Ours	70.6	64.2	55.0	16.3	63.3	0.30
ResNet-34	Single-task	72.7	68.6	58.7	16.0	64.4	-
	Ours	71.8	67.6	58.0	16.1	64.3	0.99
ResNet-101	Single-task	74.2	70.7	62.1	15.8	65.0	-
	Ours	73.3	70.9	61.0	15.9	64.5	0.93

considered baselines are as follows. Single-task means that each task is trained with an individual network, shown in Fig. 1 (a). Multi-decoder represents a simple multi-task solution where encoder is shared but decoders are task-specific. We further compare to our architecture without the task embedding (TE) network, by using task-specific batch-(BNs), and instance (INs) normalizations. We see that the Multi-decoder model, sharing a common encoder but using different decoders does not perform well, which is consistent with MTL literature [15]. Moreover, it has a large number of parameters (43.9 millions). Task-specific BNs on the other hand performs only slightly worse than Multi-decoder, with a substantially smaller model size. Interestingly, Task-specific INs performs much better than Task-specific BNs. The results for Task-specific BNs and INs show that simply adapting features to different tasks by affine transformation in the decoder is able to give reasonable performance for multi-task learning. Our method, with the task embedding network to jointly learn the (affine transformation) coefficients for AdaIN, outperforms task-specific INs by 1.13% in terms of average performance drop Δ_m. It demonstrates that learning the normalization coefficients jointly through the task embedding is better than learning them separately for each task. We also observed that during training, our method converges much faster than Task-specific INs and BNs. Moreover, our method only increases the size of the model by a small margin, because the task embedding network $\mathcal{C}$ in our model is very small.

Task embedding network. We study the impact of two different choices for the task-condition vector $\boldsymbol{v}_\tau$, as described in §4. The results are shown in Table 2. For orthogonal encodings, we observe that the performance of our method is robust towards the embedding dimensionality d, while perfoming best at $d = 100$. Gaussian encodings perform equally well as the orthogonal counterpart for dimensionality below 100, and tends to be slightly worse above. We conjecture that under Gaussian encoding, the distance between task-condition vectors for two tasks is random (close or far), which is not desirable. However, this study demonstrates that our conditioning is robust towards these hyper-parameters. In our experiments, we choose orthogonal encoding with a dimension of 100 for PASCAL-Context dataset (5 tasks). For NYUD dataset (4 tasks), we use dimension of 120 (divisible by 4).

More network architecture. Following [24, 15], we study the robustness of our method against more network architectures (ResNet-34 and ResNet-101). The results are shown in Table 3. As expected, the absolute performance on all tasks improves when using larger networks. Furthermore, our method performs closely to the corresponding single-task baselines for different backbones. Note that single-task baselines have 5 times more parameters than ours. The fact that our approach achieves similarly low average performance drop ($\Delta_m\%$) across various networks, demonstrates its robustness and effectiveness in reducing negative interference between tasks.

5.2. Comparison to state-of-the-arts

The state-of-the-art comparisons for PASCAL-Context are shown in Table 4. We compare our method to

Table 4: **Comparison with state-of-the-art.** Our TSNs outperform different multi-decoder methods on PASCAL-Context, with only a single decoder and substantially fewer parameters.

Method	Edge↑	SemSeg↑	Parts↑	Normals↓	Sal↑	$\Delta_m\%\downarrow$	# params
Single-task	71.3	64.3	55.5	16.3	62.9	-	88.7M
Series RA [31]	72.0	55.1	54.6	17.0	58.7	5.21	51.7M
Parallel RA [32]	72.1	55.9	55.0	17.0	58.6	4.81	50.8M
RCM [15]	72.3	56.6	55.8	16.7	59.3	3.62	51.7M
Ours	70.6	64.2	55.0	16.3	63.3	**0.30**	**18.3M**

Task-conditional (TC) multi-task methods: Series Residual Adapter (Series RA) [31], Parallel Residual Adapter (Parallel RA) [32], and RCM [15], since these approaches follow the same direction of MTL as mentioned previously. For a fair comparison, both Series RA and Parallel RA are implemented in our setting. Performance RCM is obtained by adapting the official implementation to our setting (U-Net architecture). We observe that our method achieves the best performance among those existing methods in terms of average performance drop, with respect to our single-task baseline. We report the number of parameters for each method, and show that our method uses the least parameters among the compared methods. Specifically, our method outperforms RCM by 3.32% and only uses 18.3M parameters, compared to 51.7M of RCM.

In fact, our main motivation in §3 is driven by efficient parameter utilization. In Fig. 4, we can see how the number of parameters $|\boldsymbol{\theta}_m|$ of each method scales with the number of tasks T. By design, our TSNs have constant parameters irrespective of T. On the other hand, other methods (RCM, Multi-decoder, etc.) scale linearly with T. For instance, when $T = 9$, our method still has 18.3M parameters, whereas the 84.0M parameters for RCM and 159.6M for Single-task make it obvious that these methods are not applicable in practical cases where many tasks are required and resources are limited.

We further validate our method in NYUD dataset. The results are shown in Table 5. Similarly, we observe that our method outperforms existing approaches by clear margins, which further demonstrates the general effectiveness of the proposed task-switching network. The qualitative comparisons are shown in Fig. 6.

Table 5: **Comparison with state-of-the-art.** On the four tasks of NYUD dataset, our TSNs outperform the compared methods, with superior parameter efficiency.

Method	Edge↑	SemSeg↑	Normals↓	Depth↓	$\Delta_m\%\downarrow$
Single-task	67.7	26.6	26.2	74.0	-
Series RA [31]	68.5	18.9	29.0	84.1	10.39
Parallel RA [32]	68.5	23.1	29.0	84.2	7.25
RCM [15]	68.7	23.2	28.4	82.1	6.14
Ours	67.9	25.9	26.1	72.7	**0.03**

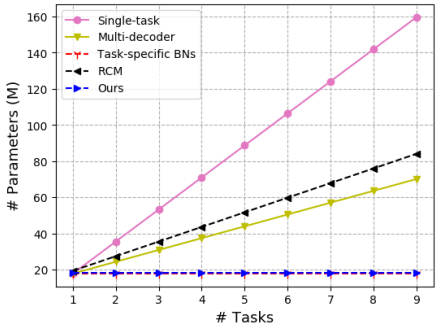
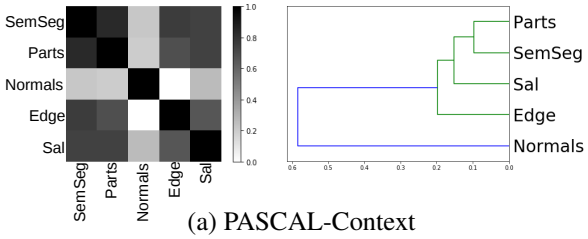
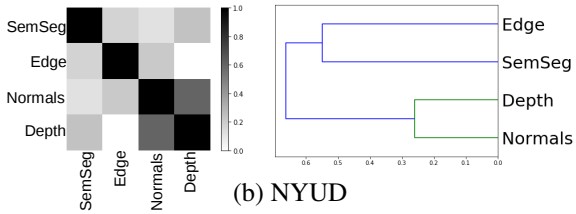


Figure 4: **Model parameters scaling.** While the number of parameters in TSNs is independent of T, it scales in a linear fashion for the compared MTL methods.



(a) PASCAL-Context



(b) NYUD

Figure 5: **Task embedding similarity.** We observe that similar tasks – such as body parts and semantic segmentation in PASCAL-Context (a), or depth and normals in NYUD (b) – cluster together in the learned embedding space, while 3D tasks are separated from 2D tasks.

5.3. Task relationships

Learning a task embedding network jointly with the MTL objective naturally raises the question, if the learned task embeddings carry some meaningful information for task relationships. To analyse this question empirically, we compute task affinity between two tasks as follows,

$$A(\tau_i, \tau_j) = 1 - \frac{\boldsymbol{l}_{\tau_i}^{\intercal} \boldsymbol{l}_{\tau_j}}{\|\boldsymbol{l}_{\tau_i}\| \|\boldsymbol{l}_{\tau_j}\|}. \tag{10}$$

We visualize our found affinities for PASCAL-Context and NYUD datasets in Fig. 5, together with an illustration of the hierarchical clustering. We make two interesting observations. First, there seems to be a clear distinction between 2D and 3D tasks in the embedding, with depth and normals being close in NYUD, as well as normals getting separated from segmentation tasks in both datasets. Second, in PASCAL, the cluster hierarchy appears to be correlated

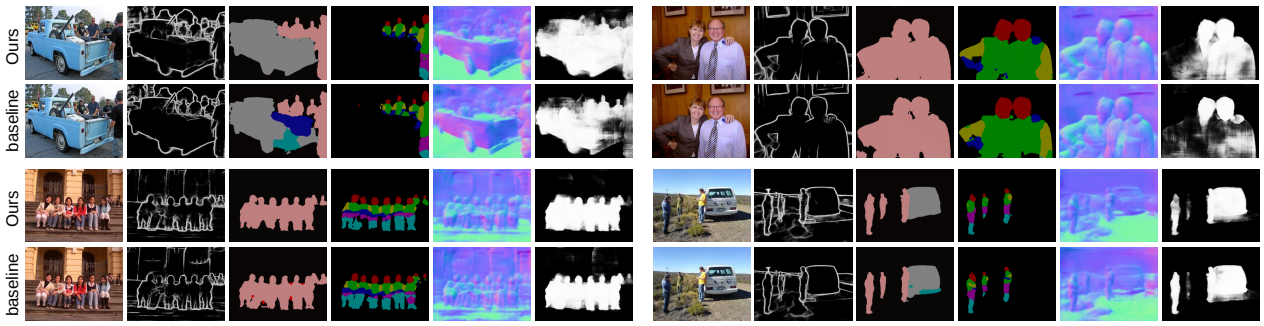


Figure 6: **Qualitative results.** We compare our model with baseline (Task-specific INs) visually. Task interference is observed in baseline where detected edges could exist in saliency predictions. Our method resolves this and outperforms the baseline in high-level tasks such as *semantic segmentation*, *parts*, and *saliency detection*. *Best viewed with zooming*.

with "semanticness" – connecting first parts and semantic segmentation, then saliency, edges and finally normals.

We further investigate the task embeddings on the 20 tasks of the Taskonomy [48] dataset, which is intended for finding task relationships. In Fig. 3, we compare our found task relationships with the ones established by *Zamir et al.*, as well as with two recent methods [39, 10]. Interestingly, there appears to be a striking similarity between the found "taskonomies". Although not perfect, we roughly observe the trend of 2D and 3D tasks clustering together, as well as a separation of low level (e.g. denoising, inpainting) from high level (semantic segmentation, scene classification) tasks. Note that our method establishes task relationships much more efficiently than the compared methods [48, 39, 10]. Specifically, these approaches need to have separate models trained for individual tasks (*i.e.*, 20 separate models for 20 tasks). Then the method proposed in Taskonomy [48] does transfer learning between different tasks to find the task similarities, while both RSA [10] and DEPARA [39] conduct pairwise comparisons among deep features extracted from a certain number of images. However, our approach only uses a single unified model and obtains the task similarities by simply computing the affinities between task embeddings.

We hypothesize that our embeddings implicitly transfer knowledge between tasks in the embedding space, in order to provide the impressive results of Table 1 and 4. If two tasks require similar features, it is favorable to share certain patterns in the conditioning, and therefore be localized closer together in the embedding space. From experimental results, we can see that this behaviour is further encouraged by the limited capacity of the embedding network.

5.4. Discussion

Test-time parameters. In Table 4 and Table 5, we report the number of parameters of our TSNs. When it comes to maximizing memory and computational efficency, we can however convert our task-switching network into a task-conditioned network, by computing the AdaIN parameters once and storing them with the model. In that case, the number of parameters drops from 18.3M to 17.7M, corresponding to the size of our IN baseline in Table 1. From this perspective, our task embedding can interpreted as a additional inductive bias for the MTL.

Architecture. We chose the U-Net architecture for simplicity, together with the ResNet-18 backbone to demonstrate the idea of task switching networks, and its behaviour and performance with regards to recent MTL methods. The application of TSNs principle using other more powerful or more efficent architectures, backbones, decoders, or conditioning strategies leaves for future work.

6. Conclusion

In this paper, we introduce the first approach for multi-task learning that uses only a single encoder and decoder architecture. By design, our Task Switching Networks offer a substantial advantage in terms of simplicity and parameter efficiency. This is achieved by sharing the complete set of network parameters among all tasks and using a conditioning network to learn task-specific latent vectors (embeddings) which then adapt the decoder for corresponding tasks. As demonstrated in our experiments, our proposed task switching strategy improves MTL performance by learning the task embeddings jointly with all tasks, and offers a new perspective on multi-task learning through the lens of task embeddings. Our experiments further validate the utility and efficiency of the proposed framework, which outperforms state-of-the-art multi-decoder methods on standard benchmark datasets with much less parameters, under fair comparisons. We also show interesting findings on task relationships using the learnt task embeddings. To conclude, we believe that further investigation into the concept of task embeddings for multi-task learning will be an interesting topic for future work.

Acknowledgements

This work was partly supported by Specta AI.

References

[1] A. Achille, Michael Lam, Rahul Tewari, A. Ravichandran, Subhransu Maji, Charless C. Fowlkes, Stefano Soatto, and P. Perona. Task2vec: Task embedding for meta-learning. *ICCV*, 2019. 3

[2] Hakan Bilen and A. Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *arXiv*, 2017. 2, 3

[3] Felix J. S. Bragman, Ryutaro Tanno, Sébastien Ourselin, D. Alexander, and M. Cardoso. Stochastic filter groups for multi-task cnns: Learning specialist and generalist convolution kernels. *ICCV*, 2019. 2

[4] A. Brock, J. Donahue, and K. Simonyan. Large scale gan training for high fidelity natural image synthesis. *arXiv*, 2019. 2

[5] Rich Caruana. Multitask learning. *Machine learning*, 28(1):41–75, 1997. 2

[6] Z. Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. *arXiv*, 2018. 1

[7] M. Crawshaw. Multi-task learning with deep neural networks: A survey. *arXiv*, 2020. 2, 3

[8] C. Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. *ICCV*, 2017. 2

[9] Kshitij Dwivedi, Jiahui Huang, Radoslaw Martin Cichy, and Gemma Roig. Duality diagram similarity: a generic framework for initialization selection in task transfer learning. *arXiv*, 2020. 3

[10] Kshitij Dwivedi and Gemma Roig. Representation similarity analysis for efficient task taxonomy & transfer learning. *CVPR*, 2019. 3, 6, 8

[11] Yuan Gao, Haoping Bai, Zequn Jie, Jiayi Ma, Kui Jia, and Wei Liu. Mtl-nas: Task-agnostic neural architecture search towards general-purpose multi-task learning. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11540–11549, 2020. 3

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 4

[13] X. Huang and Serge J. Belongie. Arbitrary style transfer in real-time with adaptive instance normalization. *ICCV*, 2017. 2

[14] S. Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv*, 2015. 2

[15] Menelaos Kanakis, David Bruggemann, Suman Saha, Stamatios Georgoulis, Anton Obukhov, and Luc Van Gool. Reparameterizing convolutions for incremental multi-task learning without task interference. *ECCV*, 2020. 1, 2, 3, 4, 5, 6, 7

[16] Tero Karras, Samuli Laine, and Timo Aila. A style-based generator architecture for generative adversarial networks. In *CVPR*, 2019. 2, 4, 5

[17] Alex Kendall, Yarin Gal, and R. Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. *CVPR*, 2018. 1

[18] Iasonas Kokkinos. Ubernet: Training a universal convolutional neural network for low-, mid-, and high-level vision using diverse datasets and limited memory. In *CVPR*, 2017. 1, 2, 3

[19] L. Lan, Zhenguo Li, X. Guan, and P. Wang. Meta reinforcement learning with task embedding and shared policy. *arXiv*, 2019. 3

[20] Wu Li, V. Piëch, and C. Gilbert. Perceptual learning and top-down influences in primary visual cortex. *Nature Neuroscience*, 7:651–657, 2004. 2

[21] Yanghao Li, Naiyan Wang, J. Shi, Xiaodi Hou, and Jiaying Liu. Adaptive batch normalization for practical domain adaptation. *Pattern Recognit.*, 80:109–117, 2018. 2

[22] Shikun Liu, Edward Johns, and A. Davison. End-to-end multi-task learning with attention. *CVPR*, 2019. 2

[23] Y. Lu, Abhishek Kumar, Shuangfei Zhai, Yu Cheng, T. Javidi, and R. Feris. Fully-adaptive feature sharing in multi-task networks with applications in person attribute classification. *CVPR*, 2017. 1, 2

[24] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. In *CVPR*, 2019. 1, 2, 3, 4, 5, 6

[25] David R Martin, Charless C Fowlkes, and Jitendra Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *TPAMI*, 26(5):530–549, 2004. 5

[26] Justin N. J. McManus, W. Li, and C. Gilbert. Adaptive shape processing in primary visual cortex. *Proceedings of the National Academy of Sciences*, 108:9739 – 9746, 2011. 2

[27] I. Misra, Abhinav Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. *CVPR*, 2016. 1, 2

[28] Roozbeh Mottaghi, Xianjie Chen, Xiaobai Liu, Nam-Gyu Cho, Seong-Whan Lee, Sanja Fidler, Raquel Urtasun, and Alan Yuille. The role of context for object detection and semantic segmentation in the wild. In *CVPR*, 2014. 4, 5

[29] Adam Paszke, S. Gross, Francisco Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, Alban Desmaison, Andreas Köpf, E. Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, B. Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. *ArXiv*, abs/1912.01703, 2019. 5

[30] Nikola Popovic, Danda Pani Paudel, Thomas Probst, Guolei Sun, and Luc Van Gool. Compositetasking: Understanding images by spatial composition of tasks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6870–6880, 2021. 2

[31] Sylvestre-Alvise Rebuffi, Hakan Bilen, and Andrea Vedaldi. Learning multiple visual domains with residual adapters. In *NeurIPS*, 2017. 2, 3, 7

[32] Sylvestre-Alvise Rebuffi, Hakan Bilen, and A. Vedaldi. Efficient parametrization of multi-domain deep neural networks. *CVPR*, 2018. 2, 7

[33] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICAI*, 2015. 4

[34] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 115(3):211–252, 2015. 4

[35] Andrei A. Rusu, D. Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv*, 2019. 3

[36] Holger Schwenk and M. Douze. Learning joint multilingual sentence representations with neural machine translation. In *Rep4NLP@ACL*, 2017. 2

[37] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012. 1, 5

[38] J. Song, Yixin Chen, X. Wang, Chengchao Shen, and Mingli Song. Deep model transferability from attribution maps. *NeurIPS*, 2019. 3

[39] J. Song, Yixin Chen, Jingwen Ye, X. Wang, Chengchao Shen, Feng Mao, and Mingli Song. Depara: Deep attribution graph for deep knowledge transferability. *CVPR*, 2020. 3, 6, 8

[40] Trevor Scott Standley, A. Zamir, Dawn Chen, L. Guibas, Jitendra Malik, and S. Savarese. Which tasks should be learned together in multi-task learning? *ICML*, 2020. 2, 3

[41] Gjorgji Strezoski, Nanne van Noord, and M. Worring. Many task learning with task routing. *ICCV*, 2019. 2

[42] Dmitry Ulyanov, Andrea Vedaldi, and Victor Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv*, 2016. 2, 5

[43] Simon Vandenhende, Bert De Brabandere, and L. Gool. Branched multi-task networks: Deciding what layers to share. *BMVC*, 2019. 2

[44] Simon Vandenhende, S. Georgoulis, and L. Gool. Mti-net: Multi-scale task interaction networks for multi-task learning. In *ECCV*, 2020. 1, 2

[45] Simon Vandenhende, Stamatios Georgoulis, and Luc Van Gool. Mti-net: Multi-scale task interaction networks for multi-task learning. *ECCV*, 2020. 5

[46] Xintao Wang, K. Yu, C. Dong, and Chen Change Loy. Recovering realistic texture in image super-resolution by deep spatial feature transform. *CVPR*, 2018. 2

[47] D. Xu, Wanli Ouyang, X. Wang, and N. Sebe. Pad-net: Multi-tasks guided prediction-and-distillation network for simultaneous depth estimation and scene parsing. *CVPR*, 2018. 1, 2

[48] A. Zamir, Alexander Sax, William Bokui Shen, L. Guibas, Jitendra Malik, and S. Savarese. Taskonomy: Disentangling task transfer learning. *CVPR*, 2018. 2, 6, 8

[49] Y. Zhang, Y. Wei, and Qiang Yang. Learning to multitask. *arXiv*, 2018. 3

[50] Z. Zhang, Zhen Cui, Chunyan Xu, Zequn Jie, Xiang Li, and Jian Yang. Joint task-recursive learning for semantic segmentation and depth estimation. In *ECCV*, 2018. 2

[51] Z. Zhang, Zhen Cui, Chunyan Xu, Yan Yan, N. Sebe, and J. Yang. Pattern-affinitive propagation across depth, surface normal and semantic segmentation. *CVPR*, 2019. 2

[52] Xiangyun Zhao, Haoxiang Li, Xiaohui Shen, Xiaodan Liang, and Ying Wu. A modulation module for multi-task learning with applications in image retrieval. In *ECCV*, 2018. 2, 3