Taskology: Utilizing Task Relations at Scale

Yao Lu[1,2], Sören Pirk[1,2], Jan Dlabal[2], Anthony Brohan[1,2], Ankita Pasad[3,*]

Zhao Chen[4], Vincent Casser[4], Anelia Angelova[1,2], Ariel Gordon[1,2]

[1]Robotics at Google, [2]Google Research, [3]Toyota Technological Institute at Chicago, [4]Waymo LLC

{yaolug, pirk, dlabal, brohan}@google.com, ankitap@ttic.edu,
{zhaoch, casser}@waymo.com, {anelia, gariel}@google.com

Abstract

Many computer vision tasks address the problem of scene understanding and are naturally interrelated e.g. object classification, detection, scene segmentation, depth estimation, etc. We show that we can leverage the inherent relationships among collections of tasks, as they are trained jointly, supervising each other through their known relationships via consistency losses. *Furthermore, explicitly utilizing the relationships between tasks allows improving their performance while dramatically reducing the need for labeled data, and allows training with additional unsupervised or simulated data. We demonstrate a distributed joint training algorithm with task-level parallelism, which affords a high degree of asynchronicity and robustness. This allows learning across multiple tasks, or with large amounts of input data, at scale. We demonstrate our framework on subsets of the following collection of tasks: depth and normal prediction, semantic segmentation, 3D motion and egomotion estimation, and object tracking and 3D detection in point clouds. We observe improved performance across these tasks, especially in the low-label regime.*

1. Introduction

Many tasks in computer vision, such as depth and surface normal estimation, flow prediction, pose estimation, semantic segmentation, or classification, are inherently related as they describe the surrounding scene along with its dynamics. While solving for each of these tasks may require specialized methods, most tasks are connected by the underlying physics observed in the real world. A considerable amount of research aims to reveal the relationships between tasks [60, 5, 15, 56, 13, 54, 55], but only a few methods exploit these fundamental relationships. Some approaches rely on the unparalleled performance of deep networks to learn explicit mappings between tasks [55, 54]. However, while training tasks pairs leverages their relationships, it
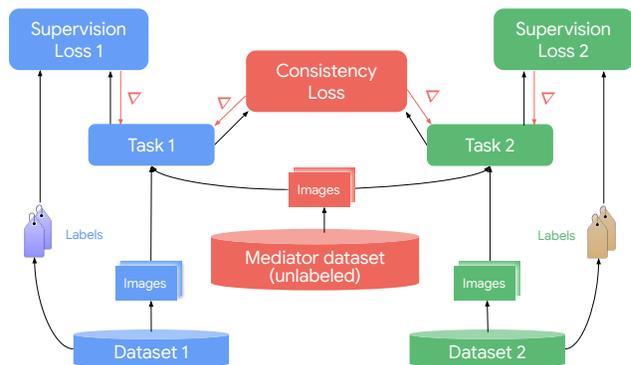
Figure 1. Illustration of our framework for the collective training of multiple tasks with a *consistency loss* (two tasks are shown). Each task is performed by a separate network, and trained on a its own dataset and a shared unlabeled *mediator* dataset. The consistency loss is imposed for samples from the mediator dataset.

may lead to inconsistencies across multiple tasks, e.g. [55], and points to the alternative of training tasks jointly.

Multi-task learning targets the problem of training multiple tasks jointly. Common to many approaches is a shared feature-extractor component with multiple "heads" that perform separate tasks [15, 56, 13]. Training multiple tasks together increases the coherency between them and – in some setups – also enables their self-supervision [60, 5]. However, the joint training also has a few disadvantages. For one, a single model for multiple tasks is difficult to design, maintain and improve, as any changes in the training data, losses, or hyperparameters associated with one of the tasks, also affects all others. Secondly, different modalities come with different architectures, which are difficult to merge into a single model. For example, point clouds require sparse processing [43], while tensored images use CNNs. Thirdly, it can become intractable to process a single model – built to perform multiple tasks – on a single compute node.

In this paper we introduce a novel approach for distributed collective training that explicitly leverages the inherent connections between multiple tasks (Fig. 1). *Con-*

sistency losses are designed for related tasks, intended to enforce their logical or geometric structure. For example, given the two tasks of predicting surface normals and depth from RGB images, the consistency loss between them is based on the analytical relation between them – normals can be computed from the derivatives of a depth map. We show here that explicitly enforcing consistency between tasks can improve their individual performance, while their collective training also establishes the correspondence among the tasks, which – in turn – leads to a more sound visual understanding of the whole scene. We term the framework 'Taskology', as it connects tasks by their physical and logical constraints.

Using consistency losses to collectively train tasks enables a modular design for training neural networks, which offers three major advantages: We train structurally different tasks with entirely separate networks that are better suited for each individual task. This is also advantageous from a design, development, and maintainability point of view; each component can be replaced or improved separately from all others. Secondly, we benefit from *unsupervised or partially labeled data*. For example, many datasets are labeled for either segmentation or scene depth; with consistency losses, we can use partially labeled datasets for training both tasks, where the consistency losses are active for the unlabeled portion of the data (Fig. 1). Finally, we train multiple complex models jointly and asynchronously in a *distributed manner*, on different compute nodes. Each network is processed on a separate machine, while their training is tied together through consistency losses. The communication between collectively trained networks – through their predictions – is asynchronous. Our experiments show that networks for different tasks can be trained with stale predictions from their peers; we do not observe a decrease in performance for up to 20 minute ($\sim$ 2000 steps) old predictions. Unlike existing methods for distributed training [37] that mostly rely on data- or model parallelism to split training across multiple compute nodes, our framework separates training at task level; each model is trained independently and asynchronously from all other models, while all models are coherently trained together. Distributed training allows scalability in multi-tasks learning, both in the number of tasks and datasets sizes.

To summarize, the contributions are: (1) we present a framework that enables a modular design for training neural networks by separating tasks into modules that can be combined and then trained collectively; (2) we propose consistency losses for coherently training multiple tasks jointly, which allows improving their overall performance; (3) we demonstrate distributed training of multiple tasks, which allows for scalability; (4) we show that collectively trained tasks supervise themselves, which reduces the need for labeled data, and can leverage unsupervised or simulated data.

2. Related Work

Exploiting the structure of – and between – tasks has a long tradition in computer science [51, 53] and computer vision [36]. It has been recognized that knowing about the structure of a task and how it is related to other tasks can be used as a powerful learning scheme [3, 55, 54, 3, 60, 5, 22]. In our work we are interested in making use of these relations more explicitly through the joint co-training of multiple tasks based on consistency losses. Therefore, our method is connected to related work on self- and unsupervised learning, multi-task learning, domain adaptation, and distributed training. While this spans a breadth of related work that we cannot comprehensively discuss, we aim to provide an overview of approaches closest to ours and with a focus on computer vision.

Methods based on self-supervision aim to autonomously generate labels for training data based on exploiting the inherent structure of related tasks. As a prominent example, Doersch et al. [14] use unlabeled image collections to learn a representation for recognizing objects. Similarly, many other approaches use proxy or surrogate tasks to learn rich representations for visual data [40, 42, 58, 56, 41]. Self-supervised multi-view learning [60] is closely related to our approach as it aims to train tasks by establishing geometric consistency between them. By designing the consistency losses we directly make use of the known relations between tasks and thereby shape the tasks space, which is similar to common self-supervised training schemes.

The goal of multi-task learning is training models for tasks so as to obtain multiple outputs for a given input, while jointly improving the performance of each individual tasks [4, 59, 46]. Many approaches exist that extract features through a shared backbone and then train multiple heads for different objectives [15, 56]. These approaches are often restricted to tuning the loss function to balance contributions between different tasks [8, 30, 48].

While our framework is not limited to any specific task domain or combination of tasks, in this work we focus on the collective training of computer vision models. To this end, we are interested in using established methods for learning depth [22, 18, 32, 60] also together with egomotion [60, 5], surface normals [57, 13, 25, 19, 44, 52], segmentation [49, 35, 29], optical flow [16, 28, 45, 9], or point cloud tracking [24, 1]. While we do not aim to change the model architectures for any of these tasks, our goal is to improve their performance by designing consistency losses for subsets of these tasks and by jointly training them.

Finally, the distributed version of our framework is closely related to the concept of distillation [26] and online distillation [2], where one network is trained toward an objective with the goal to guide the training of another network. Federated learning [39] is another technique where learning is distributed among multiple instances of the same

model. However, unlike them, we explicitly leverage the structure between tasks and by training multiple complex vision tasks simultaneously.

3. Method

Our main goal is to enable the distributed and collective training of individual network architectures for computer vision tasks that are inherently related. The main idea is to connect tasks via shared consistency losses, which represent functional or logical relationship. In particular, we aim to exploit consistency constraints relating the tasks of predicting depth, surface normals, egomotion, semantic segmentation, object motion, and object tracking with 2D and 3D sensor data. For some tasks it is possible to directly formulate their relationship as an analytical differentiable expression – e. g. normals can be computed from the derivatives of depth values [27], other tasks are related in more intricate ways, e. g. depth and egomotion, or segmentation and optical flow [23, 5, 9]. To train models collectively, we use existing model architectures for specific tasks (e. g. such as for predicting depth or segmentation) and define consistency losses between them. In this section we describe our framework, the collective training of two or multiple tasks (Sec. 3.1), the motivation for and examples of consistency losses (Sec. 3.2), as well as the distributed training of multiple tasks (Sec 3.3).

3.1. Collective Training of Tasks

Given a set of tasks $T = \{t_1, \ldots, t_n\}$ we define losses for supervising each task individually, which we denote as $\mathcal{L}_i^{sup}$, as well as consistency losses for the collective training of sets of tasks, defined as $\mathcal{L}^{con}$. In the following, tasks are referred to by their index i. We then define the overall loss as follows:

$$
\begin{aligned}
\mathcal{L} \;=\; & \sum_{i=1}^{n} \mathcal{L}_i^{sup}\bigg(\hat{y}_i(w_i, x), y_i(x) \bigg) \\
& + \; \mathcal{L}^{con}\bigg(\hat{y}_1(w_1, x), \hat{y}_2(w_2, x), \ldots, \hat{y}_n(w_n, x) \bigg), (1)
\end{aligned}
$$

where $\hat{y}_i(\cdot)$ denotes the generated prediction based on the weights w_i of a task i, $y_i(\cdot)$ denotes the groudtruth label of a task i and x is a data sample.

We assume that each task is performed by a separate deep network accompanied by a labeled dataset, which we refer to as its *dedicated dataset*. Furthermore, we use the standard supervision loss for each model as if we wanted to train it in isolation. For collective training we then use a separate (unlabeled) dataset, which we refer to as *mediator dataset*, to enforce consistency between the tasks. During training, both tasks receive samples from the mediator dataset and the results of this forward pass are used to compute the consistency loss. The training loop of each task

alternates by drawing samples from the dedicated and the mediator dataset. The setup for the collective training of two tasks is illustrated in Fig. 1.

The setup described above can have a few special cases: for one, a dedicated dataset for either task, or all, can be empty. In this case the setup reduces to unsupervised training; the unsupervised learning of depth and egomotion [60] exemplifies this case. Datasets can overlap, i. e. a dataset can have labels for multiple tasks (e. g. for semantic segmentation and depth), or datasets can be only partially labeled, i. e. one dataset has labels for depth only and another one has labels for segmentation only. In the latter case the consistency loss will be applied to both datasets, whereas supervised losses are applied to individual datasets only. The setup naturally generalizes to the collective training of N tasks, where the consistency loss is generally a function of the predictions of all participating tasks.

3.2. Task Consistency Constraints

We rely on the established knowledge [27, 47] in computer vision of tasks and their relationships to identify consistency constraints. Consistency constraints ensure the coherency between different tasks and are derived from laws of geometry and physics. Our goal is to leverage the consistency constraints to define *consistency losses* for task combinations. Any constraint that can be written as differentiable analytic expression can be used within our framework. While this work focuses on already existing relations between tasks via e. g. analytical loss relationships, future work can focus on potentially learning these losses. Sections 3.2.1, 3.2.2, 3.2.3 below describe specific task relations considered in this work.

3.2.1 Scene depth, segmentation and ego-motion

We first exploit consistency constraints between predicting depth, ego-motion and semantic segmentation. We establish consistency between these three tasks by considering the relations in image pixels and scene geometry between two consecutive frames during training. More specifically, we can 'deconstruct' a scene at a time-step t, estimating its depth and potentially moving objects; at time $t + 1$ we can 're-construct' the scene as a function of scene geometry (depth) and moving objects observed at the previous time step and considering the potential ego-motion and objects' motion. Within this setup we impose both geometric and semantics consistencies to reflect the relations between these tasks. More specifically we consider training several tasks:

Motion Prediction Networks: Given two consecutive RGB frames, $I_1(i, j)$ and $I_2(i, j)$, predict the ego-motion between these frames, i. e. the transformation of the camera between frame 2 and frame 1. This can be subdivided into a translation vector $T_{1 \rightarrow 2}$ and rotation matrix $R_{1 \rightarrow 2}$. To model independently moving objects, for every pixel (i, j),

another task can predict the movement of the point visible at the pixel of frame 1, relative to the scene, which occurred between frame 1 and frame 2, denoted as $\delta t_{1\rightarrow2}(i,j)$.

Depth Prediction Network: The depth prediction network predicts a depth map, $z(i,j)$, for an image $I(i,j)$.

Semantic Segmentation Network: From an RGB frame, the semantic segmentation network predicts a logit map $l_c(i,j)$ for each class c. For each pixel (i,j), the class is given by $c(i,j) = \text{argmax}_c\ l_c(i,j)$.

The tasks are interrelated as follows: the 3D translation fields can deviate from their background value (due to camera motion) only at pixels that belong to possibly-moving objects (e.g. vehicles, pedestrians). Therefore, semantic segmentation informs 3D motion prediction. Conversely, given a depth map and a 3D motion field, optical flow fields can be derived and then used to assert the consistency of segmentation masks in pairs of adjacent frames. The flow fields can then be used to inform training of a semantic segmentation module.

Let us define $m(i,j)$ to be the *movable mask*:

$$m(i,j) = \begin{cases} 1 & c(i,j) \in \mathcal{M} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

$\mathcal{M}$ is the collection of all classes that represent movable objects, e.g. persons, cars. For each pixel (i,j), $m(i,j)$ equals 1 if the pixel belongs to one of the movable object classes, and 0 otherwise.

We can now compute the *warping* of the first frame onto the second as a result of the scene motion and object motion. More specifically, given two adjacent video frames, 1 and 2, a depth map of frame 1, $z_1(i,j)$, the camera matrix K, and a pixel position in homogeneous coordinates $p_1(i,j) = (j,i,1)^T$, one can write the shift in p resulting from the rotation and translation and object motion δt that occurred between the two frames and obtain new values $z'_1(i,j)$ and $p'_1(i,j)$, which are a function of z, p, δt and m and egomotion network predictions for R and T (see the supp. material) $z'_1(i,j)p'_1(i,j) = KR_{1\rightarrow2}K^{-1}z_1(i,j)p_1(i,j) + K(m_1(i,j)\delta t_{1\rightarrow2}(i,j) + T_{1\rightarrow2})$.

Here p'_1 and z'_1 are respectively the new homogeneous coordinates of the pixel and the new depth, projected onto frame 2. From them we can obtain new estimated values for the image $I'_1(i,j)$, via back projection to the image space. The movable mask $m_1(i,j)$ determines the motion of objects relative to the scene to occur only at pixels that belong to movable objects, i. e. δt is applied for these objects only.

Consistency Losses: The first type of the consistency constraint is **photometric consistency** across adjacent frames, imposing that RGB values will be preserved after warping. To formulate this constraint, we sample $I_2(i,j)$ at the pixel positions $p'_1(i,j)$, and using bilinear interpolation,
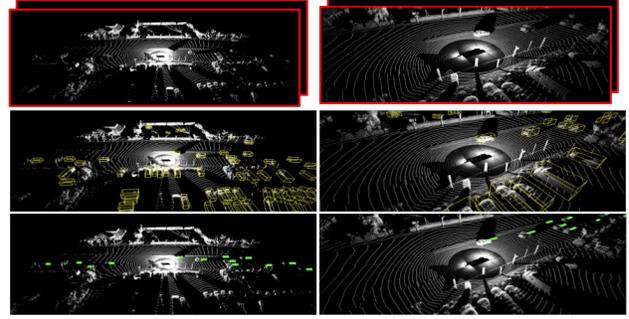
Figure 2. 3D Object Detection setup: from a sequence of point clouds (top), we predict bounding boxes per individual point cloud (middle) and the optical flow, denoted as green arrows (bottom).

we obtain $I'_1(i,j)$, frame 2's RGB image warped onto frame 1. The photometric loss can then be written generally as:

$$\mathcal{L}_{\text{ph}} = \sum_{i,j}\mathcal{L}_{\text{p}}(I'_1(i,j),I_1(i,j)) + \sum_{i,j}\mathcal{L}_{\text{p}}(I'_2(i,j),I_2(i,j)),$$
$$(3)$$

where I'_2 is defined analogously to I'_1, just with 1 and 2 swapped everywhere. $\mathcal{L}_{\text{p}}$ stands for a pixelwise photometric loss, such as an L1 penalty on the difference in RGB space and structural similarity (SSIM), each weighed by a coefficient. In our experiments we used the same photometric loss as described in [23]. The depth prediction and motion prediction networks were taken from therefrom as well. The segmentation network was taken from [21].

The second type of the consistency constraint is **segmentation logits consistency** across adjacent frames. To formulate this constraint, we sample $l_{c2}(i,j)$ at the pixel positions $p'_1(i,j)$, and using bilinear interpolation, we obtain $l'_{c1}(i,j)$. The segmentation consistency loss can then be written generally as:

$$\begin{aligned} \mathcal{L}_{\text{seg}} &= \sum_{i,j,c}\mathcal{L}_2\left(l'_{c1}(i,j),l_{c1}(i,j)\right) + \\ &+ \sum_{i,j,c}\mathcal{L}_2\left(l'_{c2}(i,j),l_{c2}(i,j)\right), \quad (4) \end{aligned}$$

where l'_{c2} is defined analogously to l'_{c1}, just with 1 and 2 swapped everywhere. $\mathcal{L}_2$ stands for a L2 loss squared.

Overall, the final consistency loss becomes

$$\mathcal{L}_{2D\_sem}^{Con} = \mathcal{L}_{\text{ph}} + \mathcal{L}_{\text{seg}}. \quad (5)$$

3.2.2 3D Object Detection in Point Clouds in Time

In this section, we focus on object detection from 3D point clouds, a difficult task which also is crucial in many practical applications such as for autonomous vehicles. We show that, when we simultaneously train an object flow network, which predicts motion of objects through time,

we can apply a motion-consistency loss to significantly boost the single-frame 3D object detector performance, especially in the low-label regime. Similarly to establishing consistencies between dynamic scenes in 2D space as in Section 3.2.1, one can establish consistencies in 3D. More specifically we can track moving objects in 3D across frames. Given a module that detects movable objects in a scene, in two consecutive frames, and another module that predicts rigid motion, we can assert that the motion-prediction module correctly estimates the motion of each object. We here directly enforced this in 3D space using 3D bounding box detections in Point Clouds and optical flow.

We address 3D Object Detection in Point Clouds using two models: one predicting 3D bounding boxes on a point cloud, and the other predicting 2D box flow on point cloud sequences (Figure 2). We use a PointPillar-based [33] network as our 3D detector, which allows us to work in a pseudo-2D top-down space for all our experiments. At the shared feature layer preceding the detector prediction head we attach a flow prediction head that, given n_f frames, outputs $3(n_f - 1)$ channels corresponding to flow. Our detection model operates on the grid-voxelized input point cloud of shape (n_x, n_y), where we choose $n_x = n_y = 468$ (the z dimension has been marginalized out as we are using a PointPillar [33] model). The grid size in $x - y$ corresponds to each grid point being of extent (0.32m, 0.32m) in real space. For each grid point, we predict:

1. $(7n_a, n_f)$ residual values pinned to n_a fixed anchors. The 7 values correspond to displacements $dx, dy, dz, dw, d\ell, dh, d\theta$ of the final predicted box from the values for the anchor boxes. Ground truth boxes are automatically corresponded to anchors at training time.

2. (n_a, n_f) class logit values denoting confidence that an object of the specified type exists for that anchor box.

3. $(n_a, 3(n_f - 1))$ values corresponding to the box flow $(flow_x, flow_y, flow_\theta)$ of a hypothetical box in the current frame to any $n_f - 1$ frames in the past.

As mentioned we use the same backbone to predict box flow. Namely, through an equivalent backbone we predict a three-channel map $(\text{flow}_x, \text{flow}_y, \text{flow}_\theta)$ of the same resolution of the detector predictions, corresponding to the flow of the boxes in a current frame to any of the previous frames in the sequence. The flow is only supervised at locations within the grid associated with positive object detections.

Consistency Loss: This set of detection and flow predictions induces natural consistency constraints. Namely, given a predicted flow that transforms an anchor point centered at (x, y) in the current frame to the closest anchor point $(x + \text{flow}_x, y + \text{flow}_y, \theta + \text{flow}_\theta) \approx (x', y', \theta')$ in another frame, we consider the following two loss functions

(in the following, all primed coordinates are coordinates after flow has been applied to the current frame):

1. $\mathcal{L}_{\text{class}}$: The class confidences at two points $(x, y, z) \mapsto (x + \text{flow}_x, y + \text{flow}_y, \theta + \text{flow}_\theta)$ connected by the predicted flow vector should have the same class confidence. $\mathcal{L}_{\text{class}} = (\text{classlogit}(x, y) - \text{classlogit}(x', y'))^2$

2. $\mathcal{L}_{\text{residual}}$: The predicted flow can be used to calculate consistent residual values for (x, y, θ) at two points connected by the predicted flow. The residual values for z, w, ℓ, and h should also remain constant between two detections connected by flow (for short time spans we assume near-constant elevation). $\mathcal{L}_{\text{residual}} = \sum_{i \in (x,y,\theta)} (di' - di + (\text{flow}_i - (i' - i))^2 + \sum_{j \in (z,\ell,w,h)} (dj' - dj)^2$

Overall, our 3D Point Cloud motion-consistency loss becomes $\mathcal{L}_{PC\_in\_time}^{Con} = \mathcal{L}_{\text{class}} + \mathcal{L}_{\text{residual}}$.

The class loss $\mathcal{L}_{\text{class}}$ ensures that class logits along predicted object tracks are equal, while the residual loss $\mathcal{L}_{\text{residual}}$ do the same for residual values along tracks. The first term in the residual consistency takes into account the predicted flow, which transforms (x, y, θ) to (x', y', θ'). Because the flow is continuous and not quantized like the voxel grid, we can normalize out the quantization noise exactly by adding a remainder term, $(\text{flow}_i - (i' - i))$ for $i \in (x, y, \theta)$. For the residual consistency, we further enforce that the bounding box residuals for z, ℓ, w, h do not change along object tracks. This reflects our assumption that the dimensions of the vehicle are preserved and there is no appreciable elevation change over the span of a second.

3.2.3 Depth and Surface Normals

Given a depth prediction module and a surface-normal prediction module, we can assert that the normals obtained from the spatial derivatives of the depth map are consistent with the predicted normals. More specifically, the depth model can produce continuous depth, from which one can analytically compute surface normals estimates per each location $\hat{n}_d$, which are a function of depth. On the other hand, a surface normals model, can be trained independently to produce surface normals predictions $\hat{n}_p$, and a consistency loss between these two predictions of surface normals can be imposed on the shared data source (Figure 3). The consistency is then computed as:

$$\mathcal{L}_{Normals}^{Con} = \text{cosine\_distance}(\hat{n}_d, \hat{n}_p), \qquad (6)$$

where $\hat{n}_d$ is the computed surface normals from the inferred depth and $\hat{n}_p$ is the normal map predicted from the normal prediction network (see the supp. material for derivation).

Interestingly, that can also be done by combining simulated and real data sources. Figure 3 visualizes the setup
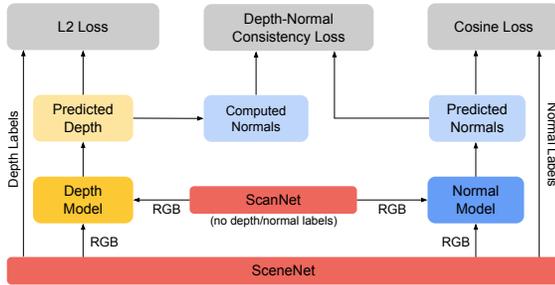
Figure 3. Depth and normals joint training for domain adaptation: we use SceneNet (simulated) data to supervise the training of separate models for depth and surface normal prediction and apply a consistency loss to jointly train both models for images from ScanNet, where we do not use any ground truth normals labels.

used in our experiments later, where the data source for surface normals (supervised) training is simulated, whereas a real data source can be used by both (it is used in unsupervised manner for the normals model).

3.3. Distributed Training

Collective training of multiple networks eventually requires distributing the computation across multiple compute nodes, to speed up the training, or simply because a large enough collection of models cannot be processed on a single machine. In distributed training it is often the communication between the nodes that sets the limitations [37]. Our framework provides the advantage of training tasks independently, with communication via consistency losses only. Furthermore, the modules share, potentially vast amounts of unsupervised, data, which allows for data-parallelism.

To reduce the communication load, distributed training schemes often aim to be asynchronous, which means that model parameters or their gradient updates develop some degree of 'staleness', which denotes the interval between updates to each model. It is easy to observe that stale predictions are less harmful than stale gradient updates [2], since predictions are expected to converge as the training progresses. Therefore, modules will query each other's predictions to compute shared losses, but propagate gradients locally, within their own module.

Our distributed implementation is based on this principle and takes advantage of shared losses. Each module is training on a separate machine ("trainer"), as illustrated in Fig. 4. The consistency loss depends on the outputs of all co-training tasks, which means that its computation requires evaluating a forward pass through all of them. Each trainer evaluates the forward pass of its own module and, to evaluate the forward passes of the other modules, it queries servers that host stale copies of the peer modules. At each training step, the trainer then pushes gradients to its own module and updates its weights.

One advantage of our distributed method is that each
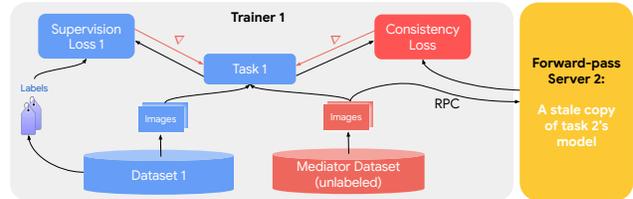
Figure 4. Illustration of our distributed setup for collective training. Each task-module is training on its own machine (its "trainer", only Trainer 1 is shown in the figure). In order to compute the consistency loss, Trainer 1 reaches out to a server that hosts a stale copy of Task 2's model and performs the forward pass (and vice versa). Each trainer pushes gradient updates to its respective model, and every so often, the stale copies on both forward-pass servers are refreshes with fresh copies form the respective trainer.

module can train with its own hyperparameters, including optimizer, regularizers, and learning rate schedules. Typically per-task modules are published together with these hyper-parameters, and our method allows using them as necessary for each respective model. Moreover, more computationally expensive modules can be allocated with more computational resources, to approximately equalize the training times among the modules. Finally, since the modules communicate through predictions, and predictions are typically much more lightweight than network weights, the communication overhead is significantly lower compared to other distributed training techniques.

4. Experiments

In the following sections we report results of experiments on using consistency losses, for the co-training of multiple models as described in Sections 3.2.1, 3.2.2, 3.2.3. We observe improved performance across tasks (Section 4.1), successful training with unlabeled data, where our approach is more helpful in lower label regimes (Section 4.2), and successful domain adaptation (Section 4.3). The experiments in Sections 4.1, 4.2, and 4.4 were run distributed, whereas the rest of the experiments were run on a single machine (e. g. as in Fig. 1).

4.1. Scene Depth, Segmentation and Ego-motion

For this experiment we show results for the distributed collective training of three tasks. The first task is semantic segmentation based on NAS-FPN [21]. The other two tasks are depth prediction and motion estimation (for both camera and objects), for which we rely on existing models [23]. Segmentation masks were used to regularize the 3D motion fields [23] (Section 3.2.1). Semantic segmentation was trained on COCO [34] 2017 as its dedicated dataset and Cityscapes [10] was used as the unlabeled mediator dataset – no Cityscapes labels were used at training. Each of the three models was trained on a separate machine, as outlined in Sec. 3.3. The segmentation module received a greater

allocation of compute resources than the others, since it is significantly more computationally expensive. The hardware configuration is described in the supp. material. The batch size, the optimizer, the learning rate, and other hyperparameters varied across the tasks and based on the respective published values for each model. During training, each of the three models queries its peers via RPC to obtain their predictions, which were up to *one minute* stale.

Configuration	Depth Error (Abs. Rel.)	Segmentation MIOU
A. Depth & motion only	0.165	-
B. Segmentation only	-	0.455
C. Frozen segmentation model B with depth & motion	0.129	-
D. Frozen depth & motion model C & segmentation	-	0.471
E. Depth, motion and segmentation training jointly	**0.125**	**0.478**

Table 1. Results of the distributed collective training of three models: Depth prediction, 3D motion prediction, and semantic segmentation. COCO was the dedicated dataset for semantic segmentation, and Cityscapes served as an unlabeled mediator dataset. Both depth prediction and segmentation were evaluated on Cityscapes, with segmentation evaluated only for predictions associated with pedestrians and vehicles (details of the evaluation protocol are given in the supp. material).

The effect of collective training on the performance of the participating models is shown in Tab. 1. Experiments A and B are the baselines, where the depth and motion models were trained jointly, but separately from segmentation. Experiment E shows the improvement in performance when all three tasks train jointly with consistency constraints. Rows C and D are ablations that demonstrate the changes in performance when consistency constraints are turned on progressively. In C the depth and motion models are supervised by the segmentation model from experiment B. C achieves the same depth error as a similar configuration trained on a single machine [23], where segmentation masks were precomputed. In experiment D, segmentation was concistency supervised by the improved depth and motion model from experiment C, but the latter two models remained frozen. The progression in quality demonstrates the effect of consistency supervision on all tasks.

While consistency contributes to correctness, it does not guarantee the latter. This is reflected in the failure cases of the method. Some illustrative examples are shown in the supp. material.

4.2. 3D Object Detection in Point Clouds in Time

We perform all experiments on the vehicle class of the Waymo Open Dataset [50] which provides complete tightly-fitting 3D bounding box annotations along with tracks for each vehicle and use a sequence length $n_f = 3$ point clouds ($\Delta = 0.5s$). Our backbone architecture is based on a PointPillar detector [33]. Given a sequence of input point clouds, we quantize the points for each frame into a grid in the x-y plane and then use our single-frame

Method	Labels	3D mAP/mAPH (%)	BEV mAP/mAPH (%)
No Consistency	5%	17.6/9.6	44.3/24.3
Adding $\mathcal{L}^{con}$	5%	**23.5/12.0**	**51.1/26.5**
No Consistency	20%	30.8/16.4	63.0/34.1
Adding $\mathcal{L}^{con}$	20%	**31.6/19.1**	**65.7/39.2**
No Consistency	100%	53.0/47.6	**75.0/66.8**
Adding $\mathcal{L}^{con}$	100%	**54.2/49.6**	**75.0/68.5**

Table 2. 3D detection and 2D (BEV) metrics on the Waymo Open Dataset, given various degrees of dataset labels provided for training. We can see consistent improvements when applying our motion-based consistency loss, especially with fewer labels.

detection model to produce a confidence value at each grid point for the presence of an object box as well as residual values $(x, y, z, w, \ell, h, \theta)$ to refine the final box coordinates (Section 3.2.2). For all our reported experiments, $n_f = 3$ and $n_a = 2$. We follow the original PointPillar network settings in choosing all class thresholds. We perform experiments with partial labeling in which only 5% or 20% of the box labels are available. Our baseline (100%) is trained in isolation, whereas all partial label studies are performed in the distributed framework.

Our results are shown in Table 2. We can see the three sets of experiments, in which we stripped the dataset of its labels to various degrees. Our metrics are based on the standard mean average precision (mAP) metrics for 2D and 3D detection. We also use the mAPH metric introduced in [50], which takes into account object heading. mAPH is calculated similarly to mAP, but all true positives are scaled by err_θ/π, with err_θ being the absolute angle error of the prediction in radians. We also report results on both 3D detection and 2D Bird-Eye-View (BEV) detection.

We can see that joint training with consistency losses is very beneficial. The consistency loss improves the object detector performance in all three settings, with more significant improvements when labels are scarce. This also holds for both 3D detection and 2D BEV detection. Furthermore, the consistency loss has a beneficial effect on mAPH, i.e. is able to correct errors in heading, as it enforces rotational consistency along each object track.

4.3. Depth and Surface Normals with Domain Shift

Since training in our framework involves multiple datasets, it is interesting to explore what happens when there is a large domain disparity between them. To this end, we select the extreme case of domain disparity between the dedicated datasets and the mediator dataset. As tasks for this experiment we selected depth estimation and surface normal prediction. We use SceneNet [38] (simulated data) as the dedicated dataset, and ScanNet [11] (real data) as the unlabeled mediator dataset (Figure 3). We use simulated SceneNet to train depth and normal estimation models and evaluate them on the real ScanNet data as our baseline. The strong domain disparity is evident from the fact that a model trained on simulated data performs poorly

on the real dataset (Table 3). For the baseline both models are trained separately to predict depth and surface normals, with a mean squared error loss for the depth model and a cosine loss for the surface normal model. The trained models are then used to predict surface normals for samples from ScanNet. Accuracy is measured by using the ground truth data of ScanNet for depth and surface normals generated by the method of [25].

We then train the models with consistency loss on Scan-Net. The consistency loss can then be computed as cosine similarity of the computed surface normals and those predicted by the normal prediction network. The consistency is based on the fact that a normal map can be analytically computed from a depth map [31] (Section 3.2.3).

Table 3 shows the results of individual training of depth and surface normals prediction on SceneNet (simulated) and tested on ScanNet (real), and when training in the same transfer setting but with loss consistency. We observe that training with loss consistency improves the performance on both tasks on this challenging sim-to-real transfer task.

	Normals Accuracy (in %)			Depth Error (in %)
Method	$< 11.25°$	$< 22.50°$	$< 30.00°$	Abs. Rel
SceneNet $\rightarrow$ $ScanNet$	9.2	30.8	46.3	28.2
SceneNet $\rightarrow$ $ScanNet$ (with Consistency)	**13.6**	**34.9**	**46.7**	**24.9**

Table 3. Surface normal prediction transfer from SceneNet (simulated) to ScanNet (real).

4.4. Tolerance to Staleness

As discussed in Sec. 3.3, in our setup, individual modules communicate with each other through their predictions. This is motivated by the increased resilience to staleness that predictions exhibit compared to weights and gradients [2]. To study the amount of staleness our setup can afford, we train depth and egomotion [5] on the KITTI dataset [20], each on a separate machine. This experiment is particularly challenging because it is fully unsupervised: each of the modules is only supervised by the predictions produced by its peer. Since both modules are randomly initialized, each model initially receives a *random* and *stale* peer-supervision signal.

In Fig. 5 we show the results of this experiment, which is the depth prediction error as function of time for various values of staleness. While greater staleness values initially hinder the training, all experiments converge to approximately the same result, and approximately at the same time. Staleness of up to *20 minutes* – or *2000 training steps* – is shown to have no adverse effect on the convergence time or the test metric.

The negative effects of staleness on convergence time and on result metrics have been studied for various distributed training methods [12, 7, 6, 17]. While the tolerance
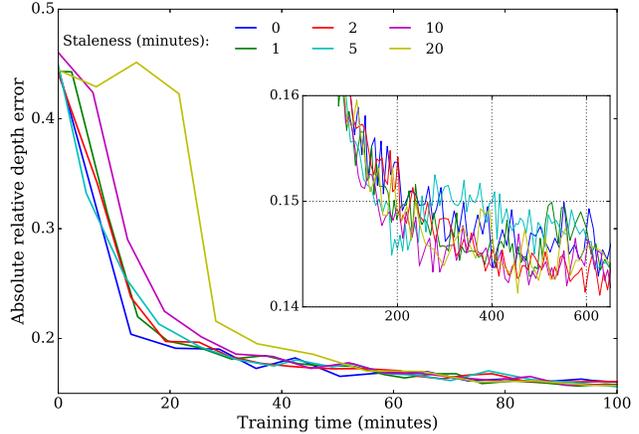
Figure 5. Depth prediction error as function of time for different values of staleness for distributed collective training on depth and egomotion on KITTI. Staleness of 20 minutes means that the depth trainer receives egomotion labels from an egomotion model that refreshes every 20 minutes, and vice versa. The depth trainer performs about 100 training steps per minute, so *20 minutes* translates to *2000 steps*. A value of 0 staleness denotes a configuration where all networks were placed on the same machine and trained synchronously. The graphs in the inset show the long-time progression of training. All experiments achieve the same absolute relative depth prediction error of about 0.143 (which on is par with the state-of-the-art [5] for models that disregard object motion), at about the same time, irrespectively of the staleness.

to staleness varies widely, due to the diversity of methods, most studies only report the sensitivity of methods to staleness of up to a few tens of steps. Unlike these findings, our distributed training setup is much more robust and thereby enables training with staleness of up to thousands of steps.

5. Conclusions

We have introduced a novel framework, 'Taskology', for the collective training of multiple models of different computer vision tasks. Our main contribution is that our framework enables a modular design for training neural networks by separating tasks into modules that can be combined and trained collectively. Furthermore, we employ consistency losses so as to exploit the structure between tasks. By jointly training multiple tasks, we have shown that consistency losses help to improve the performance, and can take advantage of unlabeled and simulated data. Our approach achieves better results from joint training, especially when a large portion of the dataset is not labeled. We also demonstrated a distributed version of the framework, which trains models on separate machines and is robust to staleness.

References

[1] Eman Ahmed, Alexandre Saint, Abd El Rahman Shabayek, Kseniya Cherenkova, Rig Das, Gleb Gusev, Djamila

Aouada, and Björn E. Ottersten. Deep learning advances on different 3d data representations: A survey. *ArXiv*, 2018.

[2] Rohan Anil, Gabriel Pereyra, Alexandre Passos, Robert Ormandi, George E. Dahl, and Geoffrey E. Hinton. Large scale distributed neural network training through online distillation. In *ICLR*, 2018.

[3] Arunkumar Byravan and Dieter Fox. Se3-nets: Learning rigid body motion using deep neural networks. 2017.

[4] Rich Caruana. Multitask learning: A knowledge-based source of inductive bias. In *ICML*, page 41–48, 1993.

[5] Vincent Casser, Soeren Pirk, Reza Mahjourian, and Anelia Angelova. Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos. In *AAAI*, volume 33, pages 8001–8008, 2019.

[6] Chi-Chung Chen, Chia-Lin Yang, and Hsiang-Yun Cheng. Efficient and robust parallel dnn training through model parallelism on multi-gpu platform, 2018.

[7] Jianmin Chen, Xinghao Pan, Rajat Monga, Samy Bengio, and Rafal Jozefowicz. Revisiting distributed synchronous sgd, 2016.

[8] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International Conference on Machine Learning*, pages 794–803, 2018.

[9] Jingchun Cheng, Yi-Hsuan Tsai, Shengjin Wang, and Ming-Hsuan Yang. Segflow: Joint learning for video object segmentation and optical flow. In *ICCV*, pages 686–695, 2017.

[10] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *CVPR*, 2016.

[11] Angela Dai, Angel X. Chang, Manolis Savva, Maciej Halber, Thomas Funkhouser, and Matthias Nießner. Scannet: Richly-annotated 3d reconstructions of indoor scenes. In *CVPR*, 2017.

[12] Wei Dai, Yi Zhou, Nanqing Dong, Hao Zhang, and Eric P. Xing. Toward understanding the impact of staleness in distributed machine learning, 2018.

[13] Thanuja Dharmasiri, Andrew Spek, and Tom Drummond. Joint prediction of depths, normals and surface curvature from RGB images using cnns. *CoRR*, 2017.

[14] Carl Doersch, Abhinav Gupta, and Alexei A. Efros. Unsupervised visual representation learning by context prediction. In *ICCV*, page 1422–1430, 2015.

[15] Carl Doersch and Andrew Zisserman. Multi-task self-supervised visual learning. *CoRR*, 2017.

[16] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *ICCV*, pages 2758–2766, 2015.

[17] Sanghamitra Dutta, Gauri Joshi, Soumyadip Ghosh, Parijat Dube, and Priya Nagpurkar. Slow and stale gradients can win the race: Error-runtime trade-offs in distributed sgd, 2018.

[18] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. In *NIPS*, pages 2366–2374. 2014.

[19] David F. Fouhey, Abhinav Gupta, and Martial Hebert. Data-driven 3d primitives for single image understanding. In *ICCV*, pages 3392–3399, 2013.

[20] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[21] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*, June 2019.

[22] Clément Godard, Oisin Mac Aodha, and Gabriel J. Brostow. Unsupervised monocular depth estimation with left-right consistency. *CVPR*, 2017.

[23] Ariel Gordon, Hanhan Li, Rico Jonschkowski, and Anelia Angelova. Depth from videos in the wild: Unsupervised monocular depth learning from unknown cameras. In *ICCV*, 2019.

[24] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun. Deep learning for 3d point clouds: A survey. *arXiv*, 2019.

[25] Steven Hickson, Karthik Raveendran, Alireza Fathi, Kevin Murphy, and Irfan A. Essa. Floors are flat: Leveraging semantics for real-time surface normal prediction. *CoRR*, 2019.

[26] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv*, 2015.

[27] S. Holzer, R. B. Rusu, M. Dixon, S. Gedikli, and N. Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2684–2689, 2012.

[28] Tak-Wai Hui, Xiaoou Tang, and Chen Change Loy. Liteflownet: A lightweight convolutional neural network for optical flow estimation. In *CVPR*, 2018.

[29] X. Jin, X. Li, H. Xiao, X. Shen, Z. Lin, J. Yang, Y. Chen, J. Dong, L. Liu, Z. Jie, J. Feng, and S. Yan. Video scene parsing with predictive feature learning. In *ICCV*, pages 5581–5589, 2017.

[30] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *CVPR*, pages 7482–7491, 2018.

[31] K. Klasing, D. Althoff, D. Wollherr, and M. Buss. Comparison of surface normal estimation methods for range sensing applications. In *ICRA*, pages 3206–3211, 2009.

[32] Arun CS Kumar, Suchendra M. Bhandarkarand, and Mukta Prasad. Depthnet: A recurrent neural network architecture for monocular depth prediction. In *CVPRW*, pages 396–3968, 2018.

[33] Alex H Lang, Sourabh Vora, Holger Caesar, Lubing Zhou, Jiong Yang, and Oscar Beijbom. Pointpillars: Fast encoders for object detection from point clouds. In *CVPR*, pages 12697–12705, 2019.

[34] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, pages 740–755, 2014.

[35] Pauline Luc, Natalia Neverova, Camille Couprie, Jacob Verbeek, and Yann LeCun. Predicting deeper into the future of semantic segmentation. *ICCV*, 2017.

[36] Jitendra Malik, Pablo Arbeláez, João Carreira, Katerina Fragkiadaki, Ross Girshick, Georgia Gkioxari, Saurabh Gupta, Bharath Hariharan, Abhishek Kar, and Shubham Tulsiani. The three r's of computer vision: Recognition, reconstruction and reorganization. *Pattern Recognition Letters*, pages 4 – 14, 2016.

[37] Ruben Mayer and Hans-arno Jacobsen. Scalable deep learning on distributed infrastructures: Challenges, techniques, and tools. *ACM Computing Surveys (CSUR)*, pages 1–37, 2020.

[38] John McCormac, Ankur Handa, Stefan Leutenegger, and Andrew J.Davison. Scenenet rgb-d: Can 5m synthetic images beat generic imagenet pre-training on indoor segmentation? 2017.

[39] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282, Fort Lauderdale, FL, USA, 20–22 Apr 2017. PMLR.

[40] Mehdi Noroozi and Paolo Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *ECCV*, pages 69–84, 2016.

[41] Mehdi Noroozi, Hamed Pirsiavash, and Paolo Favaro. Representation learning by learning to count. *CoRR*, 2017.

[42] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. *CVPR*, 2016.

[43] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. *CVPR*, 2017.

[44] Xiaojuan Qi, Renjie Liao, Zhengzhe Liu, Raquel Urtasun, and Jiaya Jia. Geonet: Geometric neural network for joint depth and surface normal estimation. In *CVPR*, 2018.

[45] Zhe Ren, Junchi Yan, Bingbing Ni, Bin Liu, Xiaokang Yang, and Hongyuan Zha. Unsupervised deep learning for optical flow estimation. In *AAAI*, page 1495–1501, 2017.

[46] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *CoRR*, 2017.

[47] Johannes L. Schonberger and Jan-Michael Frahm. Structure-from-motion revisited. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[48] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. In *NeurIPS*, pages 527–538, 2018.

[49] Shai Shalev-Shwartz and Amnon Shashua. On the sample complexity of end-to-end training vs. semantic abstraction training. *CoRR*, 2016.

[50] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *CVPR*, 2020.

[51] Alan M. Turing. Computing machinery and intelligence. (236):433–460, 1950.

[52] Peng Wang, Xiaohui Shen, Bryan Russell, Scott Cohen, Brian Price, and Alan L Yuille. Surge: Surface regularized geometry estimation from a single image. *NIPS*, pages 172–180, 2016.

[53] Terry Winograd. *Thinking Machines: Can There Be? Are We?*, page 167–189. Cambridge University Press, USA, 1990.

[54] Amir Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. pages 3712–3722, 06 2018.

[55] Amir R. Zamir, Alexander Sax, Nikhil Cheerla, Rohan Suri, Zhangjie Cao, Jitendra Malik, and Leonidas J. Guibas. Robust learning through cross-task consistency. June 2020.

[56] Amir Roshan Zamir, Tilman Wekel, Pulkit Agrawal, Colin Wei, Jitendra Malik, and Silvio Savarese. Generic 3d representation via pose estimation and matching. In *ECCV*, 2016.

[57] Huangying Zhan, Chamara Saroj Weerasekera, Ravi Garg, and Ian D. Reid. Self-supervised learning for single view depth and surface normal estimation. *CoRR*, 2019.

[58] Richard Zhang, Phillip Isola, and Alexei A. Efros. Colorful image colorization. In *ECCV*, pages 649–666, 2016.

[59] Yu Zhang and Qiang Yang. A survey on multi-task learning. *CoRR*, 2017.

[60] Tinghui Zhou, Matthew Brown, Noah Snavely, and David G Lowe. Unsupervised learning of depth and ego-motion from video. In *CVPR*, pages 1851–1858, 2017.