Task Fusion: Specializing Common Representations in Multi-Task Learning

Ananth Agarwal
Google Inc., SCPD
ananthag@stanford.edu

Dennis Duan
Google Inc., SCPD
djd@stanford.edu

Ayush Singla
Stanford CS
ayushsn@stanford.edu

Abstract

Normalization layers such as BatchNorm and LayerNorm provide significant improvements in many single-task machine learning workflows by helping to combat internal covariate shift, and they often can be plugged into common model architectures to provide performance boosts.

In multi-task learning (MTL), models aim to learn multiple tasks jointly. However, traditional hard parameter sharing paradigms can lead to problems like negative transfer and inter-task interference. Thus, ensuring that the model leverages the multi-task inductive bias towards extraction of shared cross-task features without compromising individual task performance is a major challenge for MTL approaches.

Our project seeks to address this challenge for computer vision tasks by introducing a general-purpose neural network layer called "task fusion" that augments the shared layers of a multi-task network with task-specific parameters. Ideally placed between two shared layers, task fusion consists of a task-specific modulation and a shared bottleneck, which together learn task-specific features and then promptly fuse them back to the original input shape.

From the activations output by the preceding shared layer, the task-specific modulation learns both shared parameters and task-specific parameters through a chosen function applied directly onto the activations. The output from the task-specific modulation is passed into the shared bottleneck, which learns to distill salient features from all task-specific and shared activations back into the original shape using a chosen bottleneck module. By relaxing strict hard parameter sharing for shared layers in this fashion, the layer seeks to increase model expressivity. Gradient updates are performed on the shared parameters as per the norm, by backpropagating the average loss to all shared parameters. On the other hand, we update the task-specific parameters introduced in task fusion only with the loss of the corresponding task, aiming to allow those parameters to emphasize features relevant to that task.

We considered three different function choices for the task modulation layer and three different bottleneck module choices for the shared bottleneck layer. We tested our approach on three different MTL datasets, augmenting a commonly used model on each dataset with task fusion, and evaluating model performance using the respective metrics for each dataset. Namely, we evaluated LeNet-5 on MultiMNIST, two ResNet sizes on CIFAR-100, and three variants of SegNet on the NYUv2dataset.

On LeNet-5, we found that adding task fusion provided some improvements over the baseline, with one choice of bottleneck layer in particular providing consistent improvements. Our results on the Multi-Task CIFAR-100 dataset showed marginal effects when applying to ResNet18 but substantial performance boosts on ResNet50, suggesting that deeper models may be able to more effectively extract and fuse task-specific features. On the NYUv2 dataset, we saw relatively flat performance from including task fusion across all three sizes of SegNet that we evaluated.

To better understand these results, we conducted a series of ablative and qualitative studies. First, to measure the effect of our per-task gradient updates, we performed an ablation study where we retained the task fusion architecture, but instead updated all parameters (both shared and task-specific) with the average loss across tasks. Similar to the baseline experiments, these results varied across model architectures, with ResNet50 showing a clear benefit to task-specific gradient updates but marginal or no gaps for ResNet18 and SegNet models. We also varied the placement of task fusion in SegNet and ran experiments with task fusion applied only in the decoder layers, which showed little change in the resulting performance.

Finally, to qualitatively understand how much these per-task parameters learn, we examine the distribution of the per-task parameters in task fusion. We show that in LeNet-5, the per-task parameters show higher spread, suggesting that they learn throughout the course of training, while in ResNet50, this spread is smaller, and in SegNet (both with and without ablating the per-task backpropagation), the per-task parameters are almost unchanged from their initialization values. This aligns with our evaluation results with these corresponding models, with noticeable improvements in LeNet-5 and ResNet50, but not SegNet.

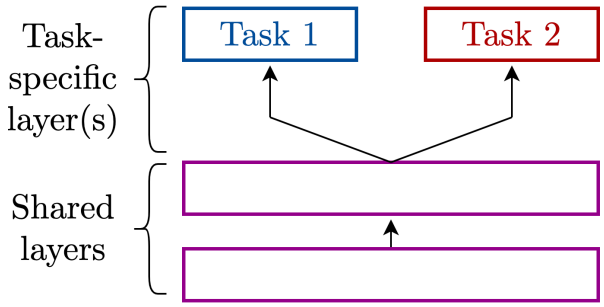

Figure 1. A two-task network with traditional hard parameter sharing

1. Introduction

Single-task learning in computer vision now outperforms human accuracies on many benchmarks, but ideally, a computer vision system should be able to perform multiple tasks simultaneously and efficiently. The research area of multi-task learning aims to accomplish this, where a model learns to perform multiple tasks at once by sharing its weights across multiple tasks. However, multitask networks are notoriously difficult to train — balancing each task's training objective to ensure convergence to commonly useful shared features while ensuring peak performance is a hard problem in multi-task learning [25].

This is especially evident in the hard parameter sharing paradigm in multi-task networks, which are perhaps the most ubiquitous form of multi-task learning architectures [22]. Splitting the model into shared layers and task-specific output heads, this paradigm is labeled "hard" parameter sharing because the same set of parameters is used for all tasks in the shared layers. There is no flexibility to learn task-specific parameters or adjust the sharing of parameters between tasks within these shared layers. Fig. 1 illustrates a two-task network that can be trained with hard parameter sharing.

While hard parameter sharing's rigid structure imposes a desirable inductive bias since disparate tasks training within the same network share the same representation up to a layer, this inflexibility that can help guard against overfitting may limit expressivity to a greater degree, and as a result, lower model performance. It can also lead to problems like negative transfer, where performance improvement of a specific task leads to performance degradation of other tasks, and inter-task interference, where training multiple tasks simultaneously can suffer from optimization challenges and limited representation capacity [20, 30].

Soft parameter sharing approaches, which allow for more fluid degrees of parameter sharing to solve some of the problems with negative transfer, are largely out of favor with the community because they add yet another set of design decisions and hyperparameters to the model and are much more memory intensive than hard parameter sharing models [22]. Thus, the aforementioned problem of ensuring the model leverages shared and task-specific features properly to ensure it converges to commonly useful shared features that yield peak performance is an open problem.

Our project addresses the challenge of task-interference when training multi-task models by introducing a general-purpose modulation layer called *task fusion* that can be added between the shared layers of a multi-task network. Task fusion introduces task-specific parameters into the otherwise shared backbone of common neural network architectures. Its input is the preceding layer's output activations. It runs these activations through task-specific modulation, which learns task-specific features and outputs a set of task-specific activations, and a shared bottleneck to fuse these task-specific activations back into the original layer input shape. We evaluate task fusion on three pairs of common computer vision models and multi-task datasets: LeNet-5 [15] on MultiMNIST [23], ResNet [10] on multi-task CIFAR-100 [14], and SegNet [1] on NYUv2 [24]. We integrate task fusion into each network by selectively modifying sequences of BatchNorm [12]-ReLU layers. While our results generally show promising signs of task fusion improving multi-task performance, we identify room for improving the ability of our new task-specific parameters to better specialize to their corresponding tasks.

2. Related Work

2.1. Task-specific Additions to Shared Architectures

Previous work has shown that task-specific additions to a shared architecture can improve model performance. Liu et al. [18] show that learning task-specific networks that compute attention scores on different layers of a shared convolutional backbone can outperform standard hard-parameter sharing models on the Cityscapes [5] and NYUv2 [24] datasets, where tasks include semantic segmentation and depth estimation. AdaShare [26] frames the parameter sharing problem as a second-layer optimization problem to learn which layers to share between tasks in a multi-task setup. Specifically, the work provides a method of adaptively determining which layers to share parameters and which to retain task-specific parameters. Wallingford et al. [29] also design a more general, architecture-agnostic method for parameter sharing in both transfer learning and multi-task settings.

Few-shot Learning with a Universal TEmplate (FLUTE) [27] implements the idea of task-specific parameters most closely to us, albeit for few-shot dataset generalization. FLUTE creates a universal template, i.e., a partially-parameterized model, that can be easily optimized on an unseen dataset using just a few steps of gradient descent from a task-dependent initialization. While training, FLUTE trains

a feature extractor with Feature-wise Linear Modulation (FiLM) [19] by sharing the parameters of the convolutional layers across datasets whilst allocating a separate set of batch normalization parameters for each dataset. At test time, FLUTE initializes the batch normalization parameters by combining per-dataset trained parameters using separately learned coefficients, which can be optimized further through a few gradient descent steps. However, these existing approaches only approach parameter sharing at the layer granularity, and do not consider sharing units of parameters smaller than a layer, through for instance, layers with mixed shared and non-shared parameters. In addition, none of these approaches consider backpropagating task-specific parameters using only the corresponding task's loss.

2.2. Multi-Task Optimization Algorithms (MTO)

Instead of supplementing networks with task-specific components for multi-task learning, another class of research has sought to counter task interference by tweaking the training process; such approaches are termed Multi-Task Optimization (MTO) algorithms [31]. Multiple MTO approaches have been proposed thus far in literature. PCGrad [32] performs "gradient surgery" to alter conflicting gradients from different tasks. GradNorm [3] automatically balances training in deep MTL models by dynamically tuning gradient magnitudes whilst adding only a single hyperparameter α. Multiple Gradient Descent Algorithm (MGDA) [7] approaches multi-task learning from a multi-objective optimization view point to efficiently find Pareto optimal solutions. Both [4, 17] inject randomization into the training pipeline to help the optimization trajectory of training process to escape poor local minima.

However, a recent benchmarking paper by Xin et al. [31] provocatively calls into question all reported gains in performance and reductions in interference reported by MTO algorithm papers. [31] shows that for neural machine translation and vision tasks including Cityscapes, MTO algorithms such as PCGrad do not outperform static weighting of tasks. Xin et al. call out some of the papers of the MTO algorithms for not sufficiently tuning hyperparameters like learning rate and weight decay for their baselines, and show that proper tuning of their scalarization baseline is able to mitigate their reported performance improvements.

2.3. FiLM

Rather than rigidly separating shared and task-specific parameters into their different layers, task fusion intends to elegantly modulate shared layers with task-specific weights. We take inspiration from the Feature-wise Linear Modulation (FiLM) paper [19]. In short, considering feature maps F_C output by a 2D convolutional layer with C channels, FiLM applies a per-channel scale $\gamma \in \mathbb{R}^C$ and shift $\beta \in \mathbb{R}^C$:

$$F_{C,FiLM} = \gamma F_C + \beta \tag{1}$$

While the authors used FiLM in multi-input settings where γ and β are output from a neural network [19], in our work, one of our task-specific modulation functions adopts FiLM's scale and shift parameters to build an affine transformation that learns parameters for each task separately.

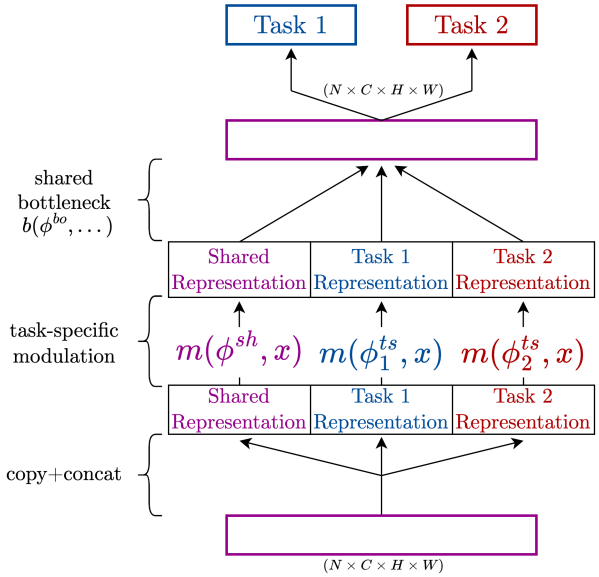

Figure 2. A two-task network with a task fusion layer between two shared layers

3. Methods

3.1. Task Fusion Design

The motivation behind the task fusion layer is to add capacity within shared layers for learning task-specific features from input activations without necessitating alterations to the existing network. We design task fusion as a self-contained black box that can be plugged into convolutional neural networks. The two main components are 1) *task-specific modulation* and 2) *shared bottleneck*. Fig. 2 illustrates how task fusion can be inserted into the shared backbone of a multi-task network.

3.1.1 Task-specific Modulation

Instead of constraining task-specific parameters to only the output heads, task fusion's novelty is adding parameters for cleaner differentiation between tasks in shared layers. Our task-specific modulation submodule m learns parameters ϕ_t^{ts} for each task $t \in T$ by applying function $m(\phi_t^{ts}, x)$ on (N, C, H, W) input activations x, in addition to a shared modulation $m(\phi^{sh}, x)$. The intent is for ϕ_t^{ts} to specialize to task t, thereby allowing each task to learn its own

representations earlier in the network without interference from other tasks, while also enabling shared features to flow through with ϕ^{sh}. We consider three choices for m:

1. FiLM-inspired scale/shift parameters:

$$m_{film}(\phi_t^{ts}, x) = \gamma_t x + \beta_t$$
$$\phi_t^{ts} = \{\gamma_t, \beta_t\}$$

   Each γ is initialized to 1 and each β is initialized to 0.

2. Task-specific PReLU [11] (MultiPReLU) parameters:

$$m_{prelu}(\phi_t^{ts}, x) = \begin{cases} x & \text{if } x \geq 0 \\ \alpha_t x & \text{if } x < 0 \end{cases}$$
$$\phi_t^{ts} = \{\alpha_t\}$$

   Each α is initialized to 0.25.

3. FiLM - MultiPReLU combination:

$$m(\phi_t^{ts}, x) = m_{prelu}(m_{film}(\phi_t^{ts}, x))$$
$$\phi_t^{ts} = \{\alpha_t\, \gamma_t, \beta_t\}$$

The output of the modulation layer is the concatenation of each modulation with shape $(N, C \times (T+1), H, W)$:

$$m_{out} = m(\phi^{sh}, x) \big|\big| m(\phi_1^{ts}, x) \big|\big| \dots \big|\big| m(\phi_T^{ts}, x) \quad (2)$$

3.1.2 Shared Bottleneck

The bottleneck module b contains shared parameters ϕ^{bo} that reduce the $C \times (T+1)$ filter dimension from m_{out} back to C:

$$b_{out} = b(\phi^{bo}, m_{out}) \quad (3)$$

Conceptually, it learns how to extract and fuse salient features from each of the task-specific and shared activations. We consider three choices for b:

1. 2D convolution (kernel size 1) and batch norm, modeled after the residual connection implementation in ResNet [10]
2. Fully-connected (linear)
3. Self-attention modified to reduce the filter dimension through the query, key and value matrices. This submodule in turn has two variants:
   (a) Single-headed self-attention, treating each activation as a token like in [28]
   (b) Patch-based multi-headed self attention in the style of [8], where each activation is split into p patches on which multi-headed self-attention is performed

3.2. Backpropagation

As in traditional multi-task networks, we update the network's shared parameters and task output head parameters by backpropagating the average loss across all tasks. The task-specific parameters introduced in task fusion are updated using only the corresponding task's loss.

$$\theta_t \leftarrow \nabla_{\theta_t} L_t \quad \forall t \in T \quad (4)$$
$$\theta_{sh} \leftarrow \nabla_{\theta_{sh}} \frac{1}{T} \sum_t L_t \quad (5)$$

$\theta_t =$ parameters specific to task t $\quad L_t =$ loss for task t
$\theta_{sh} =$ shared parameters $\quad T =$ number of tasks

Note from Eq. (5) that each task's loss is weighted equally. θ_t is the union of all $\{\phi_t^{ts}\}$ across all task fusion layers.

4. Dataset and Features

We evaluate task fusion's performance on three common vision datasets that encompass a variety of vision tasks. We use a different model architecture on each dataset.

4.1. MultiMNIST

MultiMNIST [23] is a multi-task variant of the traditional MNIST [16] dataset where two digits are overlaid with a small offset to create an image with a left-positioned digit and a right-positioned digit. There are two image classification tasks, one for classifying each digit. The evaluation metrics are each task's classification accuracy. The dataset consists of 60,000 training examples and 10,000 test examples [16]. We split the training data into 48,000 training examples and 12,000 validation examples for hyperparameter tuning.

The model is LeNet-5 [15] with two classification output heads, one for each task. LeNet-5 is a shallow and narrow model. Coupled with the simplicity of MultiMNIST, this is a basic setup to validate the basis for task fusion.

4.2. Multi-Task CIFAR-100

The standard single task multiclass classification dataset CIFAR-100 [14] can instead be broken into a set of multi-task binary predictions [21]. First, we replicate the approach in [2] of dividing the 100 "fine" labels into 20 equal size "coarse" groups of labels. For each image, our model outputs 20 binary predictions for classification into each of these coarse groups, which is 20 tasks. Due to time and compute constraints with the dimension expansion core to task-specific modulation (Sec. 3.1.1), we further manually group the 20 coarse tasks into 4 groups of different sizes: Animal, Plant, Man-made, and Landscape. The composition of each group is shown in Tab. 8 in Appendix A.1.

These 4 groups only apply within task-specific modulation function m and for backpropagation; there are still 20 tasks and 20 classification output heads trained with binary cross-entropy loss. Denoting $h(t)$ as the mapping from task t to one of the 4 groups, we apply a slight modification to the task-specific backpropagation from Eq. (4):

$$\theta_{h(t)} \leftarrow \nabla_{\theta_{h(t)}} \frac{1}{|h(t)|} L_t \quad \forall t \in T \tag{6}$$

A unique insight we gain from having groups of varying sizes (the largest has size 10 while the smallest has size 1) is seeing how task-specific modulation's performance changes based on how much it needs to generalize, *i.e.*, if it starts to suffer from the same task interference as shared parameters do if the group is large.

The original dataset consists of 50,000 training examples and 10,000 test examples. We split the training data into 40,000 training examples and 10,000 validation examples. For inference, we are essentially performing standard multiclass classification: run softmax on the 20 output predictions and predict the class with highest probability.

The models we assess are ResNet18 and ResNet50 [10]. These two models enable us to infer how model depth affects task fusion's performance.

4.3. NYUv2

Compared to MultiMNIST and Multi-Task CIFAR-100, the NYUv2 [24] dataset presents the most difficult and most varied tasks. It consists of 3-channel images of indoor scenes, annotated with labels for three tasks: (1) semantic segmentation across 13 classes as defined in [6], (2) depth data obtained with a Microsoft Kinect, and (3) surface normals as provided in [9]. We use these images as they are preprocessed in [18], with images rescaled to 288×384 pixels, with 795 training and 654 validation images, and during training we use the same loss and accuracy metrics as defined in [18].

The model we use for NYUv2 tasks is SegNet [1], a convolutional encoder-decoder model designed for scene understanding. We largely adapt the implementation from [18] and also evaluate task fusion on three variants of SegNet: (1) the standard architecture, (2) a wide variant, with an increased number of filters in the later layers, and (3) a deep variant, with double the number of convolution layers.

5. Experiments

In general, we use task fusion to modify the common BatchNorm-ReLU pattern in existing vision networks. Specifically, we replace the learned BatchNorm scale/shift parameters (when using a FiLM modulation layer), the ReLU layer (when using a MultiPReLU modulation layer), or both (when using a FiLM-MultiPReLU modulation layer).

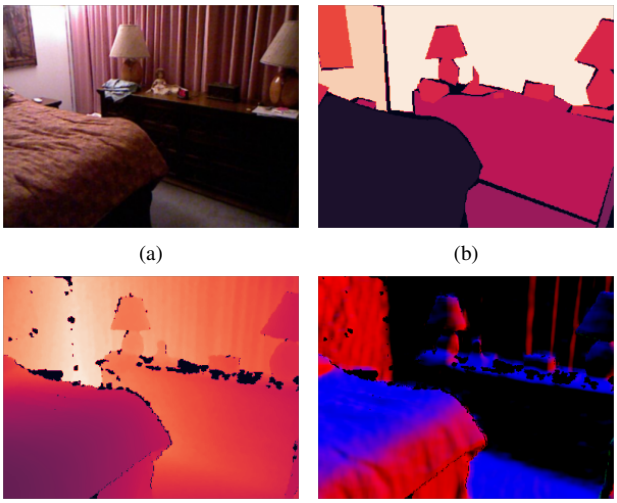

Figure 3. An example image (a) from the NYUv2dataset, with its corresponding semantic segmentation label (b), depth data (c), and surface normal data (d).

We use SGD with momentum of 0.9, and anneal the learning rate with cosine annealing for all training runs. For the MultiMNIST and Multi-Task CIFAR-100 runs, we swept over learning rates $[0.001, 0.005, 0.01, 0.05, 0.1, 0.5]$. Furthermore, all experiments were run on a single NVIDIA A100 GPU.

5.1. MultiMNIST

We train LeNet-5 for a maximum of 100 epochs with early stopping with a patience of 10 epochs for the two-task average validation loss and batch size 256.

All task-specific modulation and shared bottleneck function combinations are reported. The baseline model we train is an un-augmented LeNet-5 model. Note that the vanilla LeNet-5 does not include BatchNorm, so instead we are experimenting with replacing the ReLU that follows each of the two convolutional layers with task fusion:

Base:	Conv - ReLU
FiLM:	Conv - FiLM - Bottleneck - ReLU
MPReLU:	Conv - MPReLU - Bottleneck
FiLM+MPReLU:	Conv - FiLM - MPReLU - Bottleneck

5.2. Multi-Task CIFAR-100

Similar to the MultiMNIST training configuration, we train each of ResNet18 and ResNet50 for a maximum of 100 epochs with early stopping for the 20-task average validation loss and batch size 512.

Given the larger size of ResNet18 and especially ResNet50 compared to LeNet-5, we selected a subset of task-specific modulation - bottleneck combinations for final training runs. For ResNet18, we exclude the MultiPReLU-

only modulation and report all other combinations. For ResNet50, we additionally only consider the convolutional bottleneck since early runs suggested it to be the most performant of the bottleneck functions.

The baseline models are the standard un-augmented versions. ResNet18 consists of four sets of two "Basic Block"s each. We replace the BatchNorm-ReLU layers following the first convolutional layer (out of two) in each Basic Block in the second and fourth sets. ResNet50 consists of four sets of 3, 4, 6, and 3 "Bottleneck Block"s each. Here, we replace the BatchNorm-ReLU layers following the second convolutional layer (out of three) in each BottleNeck Block, also in the second and fourth sets.

5.3. NYUv2

Training is performed over 100 epochs with a batch size of 4, resulting in 20k training iterations, with an initial learning rate of 0.001. Our baselines are the un-augmented versions of each SegNet model we evaluate, and run training runs for each combination of modulation layer and bottleneck layer; however, in early testing we found the MultiPReLU-only modulation and the linear bottleneck to be less performant across all other combinations, so we exclude them in our final results. In addition, we exclude the attention bottleneck layer due to memory and runtime issues. The task fusion layer is used to augment all BatchNorm-ReLU structures in each networks, except in the deep SegNet variant, where we only apply task fusion to the last BatchNorm-ReLU pair in each convolution block (effectively, every other BatchNorm-ReLU instance).

5.4. Ablation Studies

5.4.1 Removing Task-Specific Backprop

Task fusion introduces new task-specific parameters to the network in the task-specific modulation (Sec. 3.1.1) submodule. Our primary approach is to update these parameters by backpropagating the loss for the corresponding task as depicted in Eq. (4). To determine if there is benefit from having these parameters be task-specific, we run an ablation where we still create m_{out} with $C \times (T+1)$ filters as shown in Eq. (2), but all parameters are shared. Specifically, we do not apply Eq. (4) and instead backprop the average loss across all tasks to all parameters (an extension of Eq. (5)). In this setting, there are no task-specific parameters in task fusion, and m can instead be interpreted as applying a single large shared modulation. The purpose of this experiment is to test if any performance change achieved by task fusion is due to its task-specific design or just added capacity from adding more parameters to the network.

	Test Acc Left Digit (%)	Test Acc Right Digit (%)
Single-headed	88.2	86.04
Multi-headed	87.87	85.42

Table 1. Comparison of attention bottlenecks on MultiMNIST.

5.4.2 Decoder-Only SegNet

In addition, we run another study to test whether layer selection when applying task fusion affects performance in SegNet. Specifically, since SegNet utilizes an encoder-decoder architecture, we run additional training runs where only the decoder blocks include task fusion layers. Our hypothesis is that this may strike a balance between allowing the model to efficiently learn shared visual features that may be useful across tasks and allowing each task to learn custom representations in earlier layers. In addition, applying task fusion more sparsely helps reduce the memory and computational footprint of training.

6. Results and Analysis

6.1. MultiMNIST

Results are shown in Tab. 2 across the three specified modulation layers and three bottleneck layers; the attention layer used in these experiments is single-headed, non-patch-based attention.

We saw the most improvement overall from the FiLM-MultiPReLU+Conv2D task fusion architecture. In terms of bottleneck layers, Conv2D bottlenecks provided the largest improvement, followed by linear bottlenecks and attention. Most notably, the attention bottlenecks provided the lowest performance boost, took the longest to converge, and had a much higher memory and computation footprint than the other bottleneck types. In addition, we ran an early experiment using both attention bottleneck variants described above, with results in Sec. 6.1. Both attention bottlenecks were trained with FiLM+MultiPReLU task-specific modulation function using an LR of 0.001 for 200 epochs optimized by SGD with a momentum of 0.9.

6.2. Multi-Task CIFAR-100

Results are displayed in Tab. 3 (the attention configuration used here is also single-headed self-attention). Compellingly, there is a stark difference in the performance boost afforded by task fusion between ResNet18 and ResNet50. While task fusion has a marginal effect on ResNet18, the FiLM-MultiPReLU modulation improves ResNet50 test accuracy by 2.5 percentage points over the baseline. This is a mild but notable improvement considering ResNet18's performance. Recall from Sec. 4.2 that loss is the binary cross-entropy loss (reported as the aver-

Modulation	Bottleneck	Test Acc Left Digit (%)	Test Acc Right Digit (%)	Average Test Loss	Early Stop Epoch
Baseline		96.08	94.46	0.1474	8
FiLM	Conv2D	96.26	94.81	0.1374	20
MultiPReLU	Conv2D	96.42	94.96	0.1342	29
FiLM-MultiPReLU	Conv2D	**96.48**	**95.20**	**0.1305**	19
FiLM	Linear	**96.03**	**94.54**	0.1536	22
MultiPReLU	Linear	95.96	94.37	**0.1501**	11
FiLM-MultiPReLU	Linear	95.79	94.30	0.1586	21
FiLM	Attention	94.45	92.79	0.2025	59
MultiPReLU	Attention	94.85	93.72	0.1753	47
FiLM-MultiPReLU	Attention	**95.69**	**94.11**	**0.1615**	31

Table 2. Results of applying task fusion to MultiMNIST

age over all 20 tasks in the table), while inference accuracy is analogous to single task multiclass classification where the highest probability class is selected. Although the difference is small, for ResNet18 it is important to note that both task-modulation functions with the Conv2D bottleneck have lower loss than the baseline, suggesting that task fusion does slightly improve the network's ability to handle multiple tasks. For ResNet18, the models with task fusion may be improving individually at each task (*i.e.*, reducing probability of incorrect classes within each task) even though the cross-task performance of assigning the highest probability to the correct class is generally unchanged. Despite the longer convergence time, ResNet50 boasts an accuracy improvement that implies deeper models may be better able to extract task-specific features, and most importantly, effectively fuse them in subsequent layers before the per-task output heads. Surprisingly, across both models there is no consistent positive difference between the FiLM and FiLM-MultiPReLU modulations, suggesting that the MultiPReLU nonlinearity is not particularly effective.

Fig. 4 plots the 20 task test losses for the baseline and the Conv2D bottleneck. Tasks in the largest and most diverse groups, Animal and Man-made, have higher loss than tasks in smaller groups. The most notable difference is the closer clustering of the task losses for the FiLM-MultiPReLU modulation, which hints that backpropagating to task fusion by group as illustrated in Eq. (6) somewhat "levels the playing field" between the tasks in the group, perhaps by more evenly distributing task interference within a group than if there are no group-specific parameters.

6.3. NYUv2

The main results are found in Tab. 5, and all metrics reported are averages from two training runs. In general, we saw flat to slightly improved results from the models with task fusion. Specifically, the FiLM-MPReLU+Conv2D treatment showed small improvements on all three tasks over the baseline in the standard and wide SegNet architectures. On the deep SegNet architecture, we saw more mixed results, with no model outperforming the others on all tasks. In these cases, this suggests that the addition of the task fusion simply traded performance in one task for another, rather than contributing meaningful improvements or declines. Overall, we saw no clear difference between the FiLM-MPReLU and FiLM-only runs, and more samples would be required to determine a statistically significant difference, if one exists.

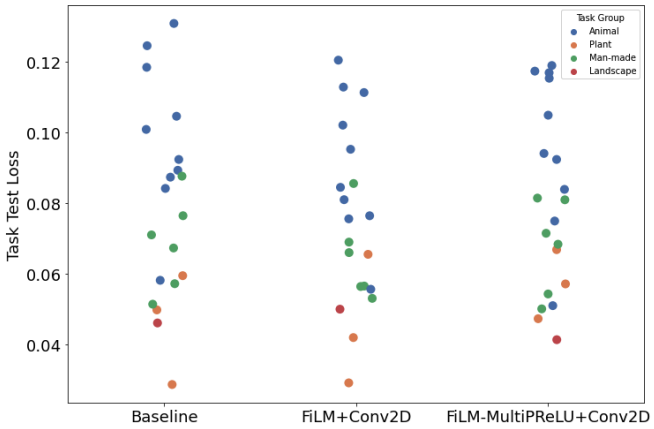

Figure 4. ResNet18 Task Test Binary Cross-Entropy Losses

6.4. Ablation Studies

6.4.1 Removing Task-Specific Backprop

We ran ablative studies on the ResNet and SegNet architectures, with results shown in Tab. 4 and Tab. 6.

As in the main results, ResNet size affects performance. For ResNet50, there is a clear advantage to task-specific backprop with FiLM-MultiPReLU+Conv2D; it outperforms having all parameters shared by more than three percentage points in test accuracy and has lower test loss. A similar accuracy boost is observed for FiLM+Conv2D, al-

Model	Modulation	Bottleneck	Test Acc (%)	Average Test Loss	Early Stop Epoch
ResNet18	Baseline		73.32	0.0793	32
	FiLM	Conv2D	73.01	**0.0745**	31
	FiLM-MultiPReLU	Conv2D	**73.19**	0.0758	40
	FiLM	Linear	**73.39**	**0.0776**	40
	FiLM-MultiPReLU	Linear	72.44	0.0805	41
	FiLM	Attention	68.27	0.0815	31
	FiLM-MultiPReLU	Attention	**69.36**	**0.081**	42
ResNet50	Baseline		71.93	0.0769	30
	FiLM	Conv2D	74.3	0.0761	40
	FiLM-MultiPReLU	Conv2D	**74.48**	**0.0735**	39

Table 3. Results of applying task fusion to multi-task CIFAR-100

Model	Architecture	Test Acc (%)	Average Test Loss	Early Stop Epoch
ResNet18	FiLM+Conv2D	73.01	**0.0745**	31
	- no per-task backprop	**73.9**	0.0771	39
	FiLM-MultiPReLU+Conv2D	**73.19**	**0.0758**	40
	- no per-task backprop	72.92	0.0777	39
	FiLM+Linear	**73.39**	**0.0776**	40
	- no per-task backprop	72.34	0.0782	36
	FiLM-MultiPReLU+Linear	**72.44**	0.0805	41
	- no per-task backprop	70.72	**0.08**	33
	FiLM+Attention	68.27	0.0815	31
	- no per-task backprop	**69.7**	**0.0789**	39
	FiLM-MultiPReLU+Attention	**69.36**	0.081	42
	- no per-task backprop	68.61	**0.0809**	35
ResNet50	FiLM+Conv2D	**74.3**	0.0761	40
	- no per-task backprop	72.72	**0.0745**	32
	FiLM-MultiPReLU+Conv2D	**74.48**	**0.0735**	39
	- no per-task backprop	71.39	0.0795	33

Table 4. Backprop ablation results on multi-task CIFAR-100

though the loss difference is much smaller. Notably, task-specific task fusion takes longer to converge than if the parameters are shared, demonstrating that task-specificity has an increased computational cost, but it pays off with performance improvement. Margins are much smaller for ResNet18. This is somewhat expected given that the main results also show little difference between the baseline and task fusion models.

On the SegNet runs, again there was no clear benefit or harm in replacing the per-task backpropagation with standard backpropagation through the network - of all the architectures, only the FiLM-MPReLU+Conv2D setting on standard SegNet showed a decline in performance across all tasks when ablating our per-task backprop. In other architectures, often the ablated model would perform better in some tasks and worse than others, suggesting that it was simply trading off performance in one task for another, rather than providing meaningful improvements or declines. Furthermore, differences in metrics across the basic and ablated task fusion runs were extremely slim and would likely need several more runs to gain a sense of any clear difference.

6.4.2 Decoder-Only SegNet

The decoder-only task fusion results on SegNet are shown in Tab. 7. Once again, we see a very slim difference, if any, between the base task fusion architectures and the decoder-

	Normal					Depth		Segmentation	
	Angle within t° (↑)			Angle error (↓)		Depth error (↓)		mIoU/Pixel Acc. (↑)	
	11.25	22.5	30	Mean	Med.	Abs.	Rel.	mIoU	Pixel Acc
Segnet-standard									
Baseline	0.1168	0.2625	0.3642	41.06	39.40	0.7269	0.2968	0.4306	0.1382
FiLM+Conv2D	0.1102	0.2619	0.3678	**40.57**	38.87	**0.6965**	**0.2801**	**0.4575**	**0.169**
FiLM-MPReLU+Conv2D	**0.1194**	**0.2712**	**0.375**	40.65	**38.66**	0.7097	0.2904	0.4451	0.1411
Segnet-wide									
Baseline	0.1153	0.2666	0.3706	40.70	38.89	0.7120	0.2857	0.4424	0.1477
FiLM+Conv2D	0.1108	0.2636	0.3716	**39.85**	**38.44**	**0.6960**	**0.2747**	**0.4468**	**0.1624**
FiLM-MPReLU+Conv2D	**0.1168**	**0.2669**	**0.3723**	40.64	38.65	0.7086	0.2937	0.4451	0.1484
Segnet-deep									
Baseline	0.1075	0.2599	**0.3652**	**40.94**	**39.17**	**0.7088**	0.2961	0.4332	0.1333
FiLM+Conv2D	0.1024	0.2523	0.3586	41.12	39.45	0.7108	**0.2848**	**0.4345**	**0.1434**
FiLM-MPReLU+Conv2D	**0.1139**	**0.2613**	0.3643	41.15	39.36	0.7526	0.3066	0.4258	0.1215

Table 5. Results of applying task fusion on NYUv2. For each SegNet architecture, the best result in each metric is in bold.

only task fusion architectures.

6.5. Task-Specific Parameter Distributions

To further investigate the seeming absence of an effect of our layer in several cases, both against the non-task fusion baselines and against the ablative studies, we plotted the distributions of the task-specific γ, β, and α parameters from the best-performing FiLM-MPReLU models in several architectures (Fig. 5). These distributions are computed from the final checkpoint during model training. The task-specific parameters in LeNet-5 show the highest variance, while in ResNet50 most of the task-specific parameters are clustered closely around their initialization values (1, 0, and 0.25 for γ, β, and α, respecitvely). Both SegNet models show the most extreme concentration of parameters, with virtually no spread, both in the ablative and base task fusion models.

These distributions align with the overall performance results we report across the models. In the LeNet-5 experiments, we saw that task fusion provided consistent improvements, and likewise, we see a wider spread in its per-task parameters, suggesting that task fusion is able to learn more effectively. On the contrary, we see more mixed results in the SegNet performances, with very minimal (if any) performance gaps from the baseline; in these experiments, task fusion's task-specific parameters shift very little from their initialization value, suggesting little learning occuring. ResNet50's accuracy results with task fusion showed a mild improvement, again aligning with the qualitative assessment of its per-task parameter distributions, which show more spread than that of SegNet, but less than LeNet-5.

7. Conclusion and Future Work

Perhaps most notably, this work begs the question of how to improve per-task parameter learning. Sec. 6.5 shows that in the larger models, task fusion per-task parameters struggle to learn effectively, possibly pointing toward a vanishing gradient problem. Future work should consider how to address this problem to encourage per-task learning and potentially reap more general performance benefits.

Furthermore, there are several architectural extensions of our layer that would be interesting to explore. For one, rather than including a shared modulation block, task fusion could reincorporate the input features with the bottleneck output via a skip connection, allowing the modulation and bottlenecks to just learn a residual. In addition, the modulation layers themselves could also learn a downprojection of the features, *e.g.* allowing the tasks to each learn a lower-dimension representation of the input features rather than simply an affine scale and shift.

We weighed each task's loss evenly when backpropagating gradients to shared parameters, but there are other loss weighting approaches like uncertainty weights [13] which learn loss weights in order to improve overall performance. We were able to implement it, but due to time constraints we did not include it in our final runs.

While the original intention was for the task fusion layer to be computationally lightweight, there is a noticeable increase in training time with task fusion than without it. The culprit is the $C * (T + 1)$ dimension expansion of task-specific modulation. Improving the runtime of this could entail implementation tweaks to use less memory or re-

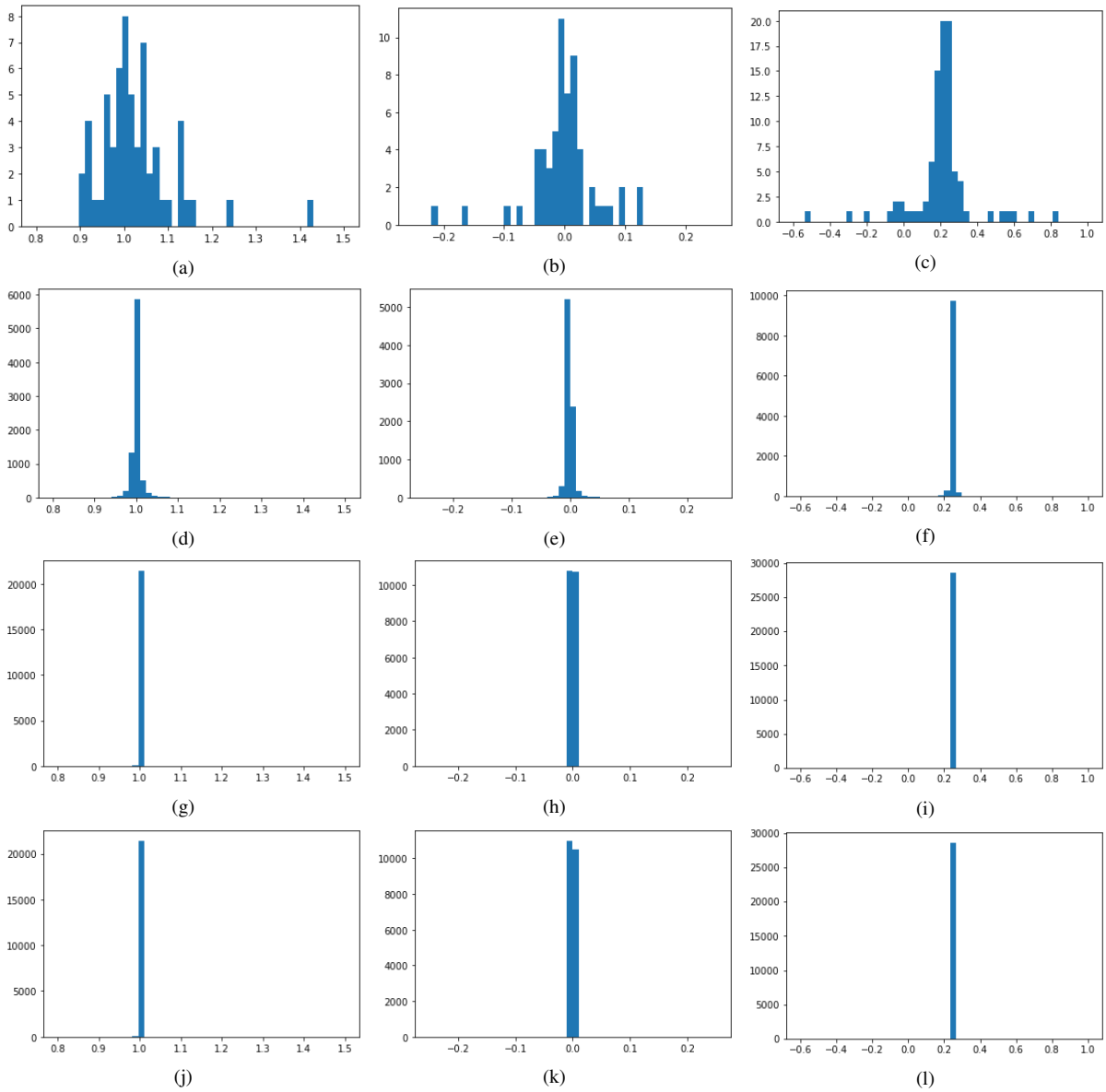

Figure 5. Parameter distributions of task fusion at the end of training across different architectures. The left and center columns show FiLM's γ and β parameters, respectively. The right column shows MultiPReLU's α parameter. The model architectures are, row-wise top to bottom, LeNet-5, ResNet50, SegNet-Standard, and SegNet-Standard without per-task backpropagation.

designing it to reduce the dimension expansion. On the bottleneck side, using single-headed self-attention as the shared bottleneck layer significantly increased the number of parameters of the model and generally took a greater number of epochs to converge to a lower accuracy. In the early MultiMNIST experiment reported in Sec. 6.1, single-headed self-attention still took less time to converge than the patch-based multi-headed attention, indicating that multiple attention heads may be unsuitable for our purposes. With more compute resources, we could more thoroughly explore attention designs on our three datasets with hyperparameter sweeps to more decisively conclude if it is worth the increased parameter count.

8. Acknowledgements

We thank Christopher Fifty, one of Professor Finn's PhD students, for his mentorship throughout this project. His guidance and unreserved assistance in developing the research idea through weekly meetings with the team over the quarter was critical to our success.

We would also like to thank Derrick Xin, Behrooz Ghor-

	Normal					Depth		Segmentation	
	Angle within t° (↑)			Angle error (↓)		Depth error (↓)		mIoU/Pixel Acc. (↑)	
	11.25	22.5	30	Mean	Med.	Abs.	Rel.	mIoU	Pixel Acc
Segnet-standard									
FiLM+Conv2D	0.1102	0.2619	0.3678	40.57	38.87	0.6965	**0.2801**	0.4575	**0.1690**
- no per-task backprop	**0.1115**	**0.2626**	**0.3685**	**40.31**	**38.79**	**0.6987**	0.2813	**0.4623**	0.1687
FiLM-MPReLU+Conv2D	**0.1194**	**0.2712**	**0.3750**	**40.65**	**38.66**	**0.7097**	**0.2904**	**0.4451**	**0.1411**
- no per-task backprop	0.1145	0.2642	0.3693	40.84	38.91	0.7106	0.2962	0.4419	0.1398
Segnet-wide									
FiLM+Conv2D	0.1108	0.2636	**0.3716**	**39.85**	**38.44**	0.6960	**0.2747**	**0.4468**	0.1624
- no per-task backprop	**0.1144**	**0.2653**	0.3698	40.42	38.78	**0.6856**	0.2760	0.4609	**0.1687**
FiLM-MPReLU+Conv2D	0.1168	0.2669	0.3723	40.64	38.65	**0.7086**	0.2937	**0.4451**	**0.1484**
- no per-task backprop	**0.1213**	**0.2717**	**0.3770**	**40.47**	**38.50**	0.7091	**0.2839**	0.4441	0.1436
Segnet-deep									
FiLM+Conv2D	0.1024	0.2523	0.3586	41.12	39.45	0.7108	**0.2848**	**0.4345**	**0.1434**
- no per-task backprop	**0.1130**	**0.2649**	**0.3706**	**40.73**	**38.82**	**0.7089**	0.2880	0.4341	0.1342
FiLM-MPReLU+Conv2D	**0.1139**	**0.2613**	**0.3643**	**41.15**	**39.36**	0.7526	0.3066	**0.4258**	0.1215
- no per-task backprop	0.1078	0.2561	0.3614	41.15	39.42	**0.7501**	**0.2928**	0.4254	**0.1237**

Table 6. Backprop ablation results on NYUv2. For each task fusion architecture, the best result between the base model and the ablation experiment is in bold.

	Normal					Depth		Segmentation	
	Angle within t° (↑)			Angle error (↓)		Depth error (↓)		mIoU/Pixel Acc. (↑)	
	11.25	22.5	30	Mean	Med.	Abs.	Rel.	mIoU	Pixel Acc
Segnet-standard									
FiLM+Conv2D	**0.1102**	0.2619	0.3678	40.57	**38.87**	**0.6965**	**0.2801**	**0.4575**	**0.1690**
- decoder-only	0.1096	**0.2631**	**0.3689**	**40.52**	38.93	0.7036	0.2883	0.4599	0.1596
FiLM-MPReLU+Conv2D	**0.1194**	**0.2712**	**0.3750**	**40.65**	**38.66**	**0.7097**	0.2904	0.4451	0.1411
- decoder-only	0.1154	0.2673	0.3718	40.68	38.76	0.7099	**0.2890**	**0.4565**	**0.1509**
Segnet-wide									
FiLM+Conv2D	**0.1108**	**0.2636**	**0.3716**	**39.85**	**38.44**	**0.6960**	**0.2747**	0.4468	**0.1624**
- decoder-only	0.1095	0.2617	0.3659	40.62	39.10	0.7085	0.2957	**0.4474**	0.1596
FiLM-MPReLU+Conv2D	0.1168	0.2669	**0.3723**	**40.64**	**38.65**	**0.7086**	**0.2937**	**0.4451**	**0.1484**
- decoder-only	**0.1186**	**0.2670**	0.3711	40.87	38.88	0.7331	0.2969	0.4371	0.1381
Segnet-deep									
FiLM+Conv2D	0.1024	0.2523	0.3586	41.12	39.45	**0.7108**	**0.2848**	0.4345	**0.1434**
- decoder-only	**0.1107**	**0.2644**	**0.3697**	**40.87**	**39.03**	0.7192	0.2925	**0.4360**	0.1424
FiLM-MPReLU+Conv2D	**0.1139**	**0.2613**	**0.3643**	**41.15**	**39.36**	**0.7526**	**0.3066**	**0.4258**	0.1215
- decoder-only	0.1057	0.2551	0.3587	41.42	39.77	0.7681	0.3164	0.4145	**0.1282**

Table 7. Decoder-only task fusion results on NYUv2. For each task fusion architecture, the best result between the base model and the decoder-only version is in bold.

bani, and Justin Gilmer from Google Research for meeting with us to discuss their MTO paper [31] and sharing their code.

9. Contributions

Ananth built the per-task FiLM layer and the integrations with ResNet, CIFAR-100, and MultiMNIST. Dennis built the per-task MultiPReLU layer, the single-headed attention bottleneck, and integrations with SegNet and NYUv2. Den-

nis and Ananth equally contributed to performing the experiments, ablation and qualitative studies, and writing the training code. Ayush wrote and evaluated the multi-task attention bottleneck, implemented uncertainty weighting, and experimented with floating point 16 precision for faster model training. All contributed equally to the writing of this final report.

References

[1] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495, 2017. 2, 5

[2] Ryan Chan. Cifar100 coarse. https://github.com/ryanchankh/cifar100coarse, 2020. 4

[3] 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. PMLR, 2018. 3

[4] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *Advances in Neural Information Processing Systems*, 33:2039–2050, 2020. 3

[5] 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. *CoRR*, abs/1604.01685, 2016. 2

[6] Camille Couprie, Clément Farabet, Laurent Najman, and Yann LeCun. Indoor semantic segmentation using depth information. *arXiv preprint arXiv:1301.3572*, 2013. 5

[7] Jean-Antoine Désidéri. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318, 2012. 3

[8] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020. 4

[9] David Eigen and Rob Fergus. Predicting depth, surface normals and semantic labels with a common multi-scale convolutional architecture. In *Proceedings of the IEEE international conference on computer vision*, pages 2650–2658, 2015. 5

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 2, 4, 5

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015. 4

[12] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. 2

[13] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics, 2017. 9

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. 2, 4

[15] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. 2, 4

[16] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010. 4

[17] Baijiong Lin, Feiyang Ye, and Yu Zhang. A closer look at loss weighting in multi-task learning. *arXiv preprint arXiv:2111.10603*, 2021. 3

[18] Shikun Liu, Edward Johns, and Andrew J Davison. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1871–1880, 2019. 2, 5

[19] Ethan Perez, Florian Strub, Harm de Vries, Vincent Dumoulin, and Aaron C. Courville. Film: Visual reasoning with a general conditioning layer. *CoRR*, abs/1709.07871, 2017. 3

[20] Amelia Pollard. *Multitask Learning, Biased Competition, and Inter-Task Interference*. The University of Manchester (United Kingdom), 2021. 2

[21] Clemens Rosenbaum, Tim Klinger, and Matthew Riemer. Routing networks: Adaptive selection of non-linear functions for multi-task learning. *CoRR*, abs/1711.01239, 2017. 4

[22] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017. 2

[23] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. Dynamic routing between capsules. *CoRR*, abs/1710.09829, 2017. 2, 4

[24] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European conference on computer vision*, pages 746–760. Springer, 2012. 2, 5

[25] Trevor Standley, Amir R. Zamir, Dawn Chen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Which tasks should be learned together in multi-task learning?, 2019. 2

[26] Ximeng Sun, Rameswar Panda, Rogerio Feris, and Kate Saenko. Adashare: Learning what to share for efficient deep multi-task learning. *Advances in Neural Information Processing Systems*, 33:8728–8740, 2020. 2

[27] Eleni Triantafillou, Hugo Larochelle, Richard Zemel, and Vincent Dumoulin. Learning a universal template for few-shot dataset generalization. In *International Conference on Machine Learning*, pages 10424–10433. PMLR, 2021. 2

[28] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 4

[29] Matthew Wallingford, Hao Li, Alessandro Achille, Avinash Ravichandran, Charless Fowlkes, Rahul Bhotika, and Stefano Soatto. Task adaptive parameter sharing for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7561–7570, 2022. 2

[30] Zirui Wang, Zihang Dai, Barnabás Póczos, and Jaime Carbonell. Characterizing and avoiding negative transfer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11293–11302, 2019. 2

[31] Derrick Xin, Behrooz Ghorbani, Justin Gilmer, Ankush Garg, and Orhan Firat. Do current multi-task optimization methods in deep learning even help? In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022. 3, 11

[32] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836, 2020. 3

A. Appendix

A.1. Multi-task CIFAR-100 Task Groups

Plant (3)
orchid, poppy, rose, sunflower, tulip
maple_tree, oak_tree, palm_tree, pine_tree, willow_tree
apple, mushroom, orange, pear, sweet_pepper
Animal (10)
aquarium_fish, flatfish, ray, shark, trout
beaver, dolphin, otter, seal, whale
bee, beetle, butterfly, caterpillar, cockroach
bear, leopard, lion, tiger, wolf
camel, cattle, chimpanzee, elephant, kangaroo
fox, porcupine, possum, raccoon, skunk
crab, lobster, snail, spider, worm
baby, boy, girl, man, woman
crocodile, dinosaur, lizard, snake, turtle
hamster, mouse, rabbit, shrew, squirrel
Man-made (6)
lawn_mower, rocket, streetcar, tank, tractor
bottle, bowl, can, cup, plate
clock, keyboard, lamp, telephone, television
bed, chair, couch, table, wardrobe
bridge, castle, house, road, skyscraper
bicycle, bus, motorcycle, pickup_truck, train
Landscape (1)
cloud, forest, mountain, plain, sea

Table 8. Multi-task CIFAR-100 task groups