BEYOND SHARED HIERARCHIES: DEEP MULTITASK LEARNING THROUGH SOFT LAYER ORDERING

Anonymous authors
Paper under double-blind review

ABSTRACT

Existing deep multitask learning (MTL) approaches align layers shared between tasks in a *parallel ordering*. Such an organization significantly constricts the types of shared structure that can be learned. The necessity of parallel ordering for deep MTL is first tested by comparing it with *permuted ordering* of shared layers. The results indicate that a flexible ordering can enable more effective sharing, thus motivating the development of a *soft ordering* approach, which learns *how* shared layers are applied in different ways for different tasks. Deep MTL with soft ordering outperforms parallel ordering methods across a series of domains. These results suggest that the power of deep MTL comes from learning highly general building blocks that can be assembled to meet the demands of each task.

1 INTRODUCTION

In multitask learning (MTL) (Caruana, 1998), auxiliary data sets are harnessed to improve overall performance by exploiting regularities present across tasks. As deep learning has yielded state-of-the-art systems across a range of domains, there has been increased focus on developing deep MTL techniques. Such techniques have been applied across settings such as vision (Bilen and Vedaldi, 2016; 2017; Jou and Chang, 2016; Lu et al., 2017; Misra et al., 2016; Ranjan et al., 2016; Yang and Hospedales, 2017; Zhang et al., 2014), natural language (Collobert and Weston, 2008; Dong et al., 2015; Hashimoto et al., 2016; Liu et al., 2015a; Luong et al., 2016), speech (Huang et al., 2013; 2015; Seltzer and Droppo, 2013; Wu et al., 2015), and reinforcement learning (Devin et al., 2016; Fernando et al., 2017; Jaderberg et al., 2017; Rusu et al., 2016). Although they improve performance over single-task learning in these settings, these approaches have generally been constrained to joint training of relatively few and/or closely-related tasks.

On the other hand, from a perspective of Kolmogorov complexity, "transfer should always be useful"; any pair of distributions underlying a pair of tasks must have *something* in common (Mahmud, 2009; Mahmud and Ray, 2008). In principle, even tasks that are "superficially unrelated" such as those in vision and NLP can benefit from sharing (even without an *adaptor* task, such as image captioning). In other words, for a sufficiently expressive class of models, the inductive bias of requiring a model to fit multiple tasks simultaneously should encourage learning to converge to more *realistic* representations. The expressivity and success of deep models suggest they are ideal candidates for improvement via MTL. So, why have existing approaches to deep MTL been so restricted in scope?

MTL is based on the assumption that learned transformations can be shared across tasks. This paper identifies an additional implicit assumption underlying existing approaches to deep MTL: this sharing takes place through *parallel ordering* of layers. That is, sharing between tasks occurs only at aligned levels (layers) in the feature hierarchy implied by the model architecture. This constraint limits the kind of sharing that can occur between tasks. It requires subsequences of task feature hierarchies to match, which may be difficult to establish as tasks become plentiful and diverse.

This paper investigates whether parallel ordering of layers is necessary for deep MTL. As an alternative, it introduces methods that make deep MTL more flexible. First, existing approaches are reviewed in the context of their reliance on parallel ordering. Then, as a foil to parallel ordering, *permuted ordering* is introduced, in which shared layers are applied in different orders for different tasks. The increased ability of permuted ordering to support integration of information across tasks is analyzed, and the results are used to develop a *soft ordering* approach to deep MTL. In this approach, a joint model learns *how* to apply shared layers in different ways at different depths for

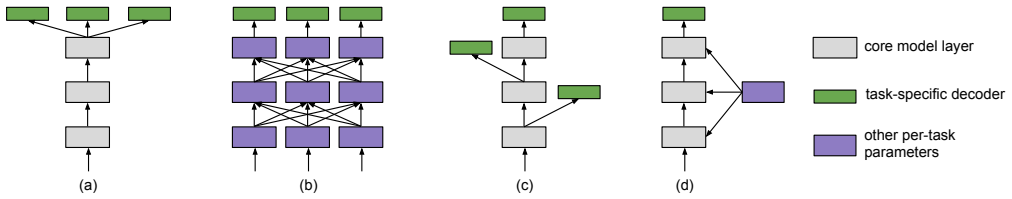


Figure 1: **Classes of existing deep multitask learning architectures.** (a) *Classical approaches* add a task-specific decoder to the output of the core single-task model for each task; (b) *Column-based approaches* include a network column for each task, and define a mechanism for sharing between columns; (c) *Supervision at custom depths* adds output decoders at depths based on a task hierarchy; (d) *Universal representations* adapts each layer with a small number of task-specific scaling parameters. Underlying each of these approaches is the assumption of parallel ordering of shared layers (Section 2.2): each one requires aligned sequences of feature extractors across tasks.

different tasks as it simultaneously learns the parameters of the layers themselves. In a suite of experiments, soft ordering is shown to improve performance over single-task learning as well as over fixed order deep MTL methods.

Importantly, soft ordering is not simply a technical improvement, but a new way of thinking about deep MTL. Learning a different soft ordering of layers for each task amounts to discovering a set of *generalizable modules that are assembled in different ways for different tasks*. This perspective points to future approaches that train a collection of layers on a set of training tasks, which can then be assembled in novel ways for future unseen tasks. Some of the most striking structural regularities observed in the natural, technological and sociological worlds are those that are repeatedly observed across settings and scales; they are ubiquitous and universal. By forcing shared transformations to occur at matching depths in hierarchical feature extraction, deep MTL falls short of capturing this sort of functional regularity. Soft ordering is thus a step towards enabling deep MTL to realize the diverse array of structural regularities found across complex tasks drawn from the real world.

2 Parallel Ordering of Layers in Deep MTL

This section presents a high-level classification of existing deep MTL approaches (Sec. 2.1) that is sufficient to expose the reliance of these approaches on the *parallel ordering assumption* (Sec. 2.2).

2.1 A classification of existing approaches to deep multitask learning

Designing a deep MTL system requires answering the key question: *How should learned parameters be shared across tasks?* The landscape of existing deep MTL approaches can be organized based on how they answer this question at the joint network architecture level (Figure 1).

Classical approaches. Neural network MTL was first introduced in the case of shallow networks (Caruana, 1998), before deep networks were prevalent. The key idea was to add output neurons to predict auxiliary labels for related tasks, which would act as regularizers for the hidden representation. Many deep learning extensions remain close in nature to this approach, learning a shared representation at a high-level layer, followed by task-specific (i.e., unshared) decoders that extract labels for each task (Devin et al., 2016; Dong et al., 2015; Huang et al., 2013; 2015; Jaderberg et al., 2017; Liu et al., 2015a; Ranjan et al., 2016; Wu et al., 2015; Zhang et al., 2014) (Figure 1a). This approach can be extended to task-specific input encoders (Devin et al., 2016; Luong et al., 2016), and the underlying single-task model may be adapted to ease task integration (Ranjan et al., 2016; Wu et al., 2015), but the core network is still shared in its entirety.

Column-based approaches. Column-based approaches (Jou and Chang, 2016; Misra et al., 2016; Rusu et al., 2016; Yang and Hospedales, 2017), assign each task its own layer of task-specific parameters at each shared depth (Figure 1b). They then define a mechanism for sharing parameters between tasks at each shared depth, e.g., by having a shared tensor factor across tasks (Yang and Hospedales, 2017), or allowing some form of communication between columns (Jou and Chang, 2016; Misra et al., 2016; Rusu et al., 2016). Observations of negative effects of sharing in column-

based methods (Rusu et al., 2016) can be attributed to mismatches between the features required at the same depth between tasks that are too dissimilar.

Supervision at custom depths. There may be an intuitive hierarchy describing how a set of tasks are related. Several approaches integrate supervised feedback from each task at levels consistent with such a hierarchy (Hashimoto et al., 2016; Toshniwal et al., 2017; Zhang and Weiss, 2016) (Figure 1c). This method can be sensitive to the design of the hierarchy (Toshniwal et al., 2017), and to which tasks are included therein (Hashimoto et al., 2016). One approach learns a task-relationship hierarchy during training (Lu et al., 2017), though learned parameters are still only shared across matching depths. Supervision at custom depths has also been extended to include explicit recurrence that reintegrates information from earlier predictions (Bilen and Vedaldi, 2016; Zamir et al., 2016). Although these recurrent methods still rely on pre-defined hierarchical relationships between tasks, they provide evidence of the potential of learning transformations that have a different function for different tasks at different depths, i.e., in this case, at different depths unrolled in time.

Universal representations. One approach shares all core model parameters except batch normalization scaling factors (Bilen and Vedaldi, 2017) (Figure 1d). When the number of classes is equal across tasks, even output layers can be shared, and the small number of task-specific parameters enables strong performance to be maintained. This method was applied to a diverse array of vision tasks, demonstrating the power of a small number of scaling parameters in adapting layer functionality for different tasks. This observation helps to motivate the method developed in Section 3.

2.2 The parallel ordering assumption

A common interpretation of deep learning is that layers extract progressively higher level features at later depths (Lecun et al., 2015). A natural assumption is then that the learned transformations that extract these features are also tied to the depth at which they are learned. The core assumption motivating MTL is that regularities across tasks will result in learned transformations that can be leveraged to improve generalization. However, the methods reviewed in Section 2.1 add the further assumption that *subsequences of the feature hierarchy align across tasks and sharing between tasks occurs only at aligned depths* (Figure 1); we call this the *parallel ordering assumption*.

Consider T tasks $t_1, \ldots, t_T$ to be learned jointly, with each t_i associated with a model $y_i = \mathcal{F}_i(x_i)$. Suppose sharing across tasks occurs at D consecutive depths. Let $\mathcal{E}_i$ ($\mathcal{D}_i$) be t_i's task-specific encoder (decoder) to (from) the core sharable portion of the network from its inputs (to its outputs). Let W_k^i be the layer of learned weights (e.g., affine or convolutional) for task i at shared depth k, with ϕ_k an optional nonlinearity. The parallel ordering assumption implies

$$y_i = (\mathcal{D}_i \circ \phi_D \circ W_D^i \circ \phi_{D-1} \circ W_{D-1}^i \circ \ldots \circ \phi_1 \circ W_1^i \circ \mathcal{E}_i)(x_i), \text{ with } W_k^i \approx W_k^j \; \forall \, (i, j, k). \quad (1)$$

The approximate equality "$\approx$" means that at each shared depth the applied weight tensors for each task are similar and compatible for sharing. For example, learned parameters may be shared across all W_k^i for a given k, but not between W_k^i and W_l^j for any $k \neq l$. For closely-related tasks, this assumption may be a reasonable constraint. However, as more tasks are added to a joint model, it may be more difficult for each layer to represent features of its given depth for all tasks. Furthermore, for very distant tasks, it may be unreasonable to expect that task feature hierarchies match up at all, even if the tasks are related intuitively. The conjecture explored in this paper is that parallel ordering limits the potential of deep MTL by the strong constraint it enforces on the use of each layer.

3 Deep Multitask Learning with Soft Ordering of Layers

Now that parallel ordering has been identified as a constricting feature of deep MTL approaches, its necessity can be tested, and the resulting observations can be used to develop more flexible methods.

3.1 A foil for the parallel ordering assumption: Permuting shared layers

Consider the most common deep MTL setting: hard-sharing of layers, where each layer in $\{W_k\}_{k=1}^D$ is shared in its entirety across all tasks. The baseline deep MTL model for each task t_i is given by

$$y_i = (\mathcal{D}_i \circ \phi_D \circ W_D \circ \phi_{D-1} \circ W_{D-1} \circ \ldots \circ \phi_1 \circ W_1 \circ \mathcal{E}_i)(x_i). \quad (2)$$

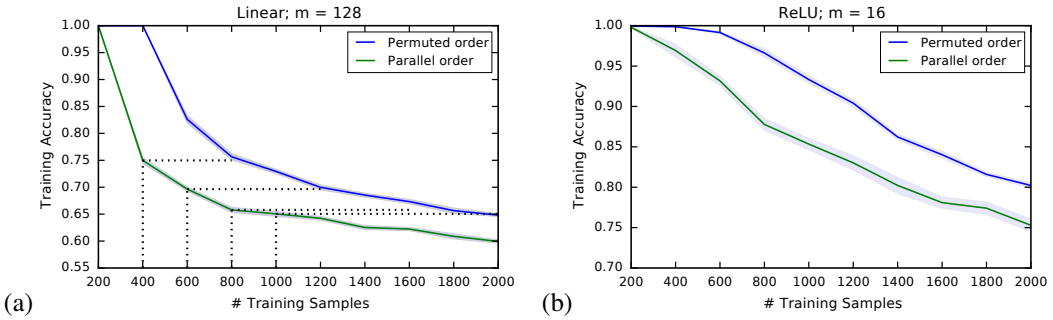


Figure 2: **Fitting two random tasks.** (a) The dotted lines show that permuted ordering fits n samples as well as parallel fits $n/2$ for linear networks; (b) For ReLU networks, permuted ordering enjoys a similar advantage. Thus, permuted ordering of shared layers eases integration of information across disparate tasks.

This setup satisfies the parallel ordering assumption. Consider now an alternative scheme, equivalent to the above, except with learned layers applied in different orders for different task. That is,

$$y_i = (\mathcal{D}_i \circ \phi_D \circ W_{\sigma_i(D)} \circ \phi_{D-1} \circ W_{\sigma_i(D-1)} \circ \ldots \circ \phi_1 \circ W_{\sigma_i(1)} \circ \mathcal{E}_i)(x_i), \tag{3}$$

where σ_i is a task-specific permutation of size D, and σ_i is fixed before training. If there are sets of tasks for which joint training of the model defined by Eq. 3 achieves similar or improved performance over Eq. 2, then parallel ordering is not a necessary requirement for deep MTL. Of course, in this formulation, it is required that the W_k can be applied in any order. See Section 6 for examples of possible generalizations.

Note that this multitask permuted ordering differs from an approach of training layers in multiple orders for a single task. The single-task case results in a model with increased commutativity between layers, a behavior that has also been observed in residual networks (Veit et al., 2016), whereas here the result is *a set of layers that are assembled in different ways for different tasks*.

3.2 The increased expressivity of permuted ordering

Fitting tasks of random patterns. Permuted ordering is evaluated by comparing it to parallel ordering on a set of tasks. Randomly generated tasks (similar to (Kirkpatrick et al., 2017)) are the most disparate possible tasks, in that they share minimal information, and thus help build intuition for how permuting layers could help integrate information in broad settings. The following experiments investigate how accurately a model can jointly fit two tasks of n samples. The data set for task t_i is $\{(x_{ij}, y_{ij})\}_{j=1}^{n}$, with each x_{ij} drawn uniformly from $[0, 1]^m$, and each y_{ij} drawn uniformly from $\{0, 1\}$. There are two shared learned affine layers $W_k : \mathbb{R}^m \to \mathbb{R}^m$. The models with permuted ordering (Eq. 3) are given by

$$y_1 = (O \circ \phi \circ W_2 \circ \phi \circ W_1)(x_1) \text{ and } y_2 = (O \circ \phi \circ W_1 \circ \phi \circ W_2)(x_2), \tag{4}$$

where O is a final shared classification layer. The reference parallel ordering models are defined identically, but with W_k in the same order for both tasks. Note that fitting the parallel model with n samples is equivalent to a single-task model with $2n$. In the first experiment, $m = 128$ and $\phi = I$. Although adding depth does not add expressivity in the single-task linear case, it is useful for examining the effects of permuted ordering, and deep linear networks are known to share properties with nonlinear networks (Saxe et al., 2013). In the second experiment, $m = 16$ and $\phi = \text{ReLU}$.

The results are shown in Figure 2. Remarkably, in the linear case, permuted ordering of shared layers does not lose accuracy compared to the single-task case. A similar gap in performance is seen in the nonlinear case, indicating that this behavior extends to more powerful models. Thus, the learned permuted layers are able to successfully adapt to their different orderings in different tasks.

Looking at conditions that make this result possible can shed further light on this behavior. For instance, consider T tasks $t_1, \ldots, t_T$, with input and output size both m, and optimal linear solutions $F_1, \ldots, F_T$, respectively. Let $F_1, \ldots, F_T$ be $m \times m$ matrices, and suppose there exist matrices

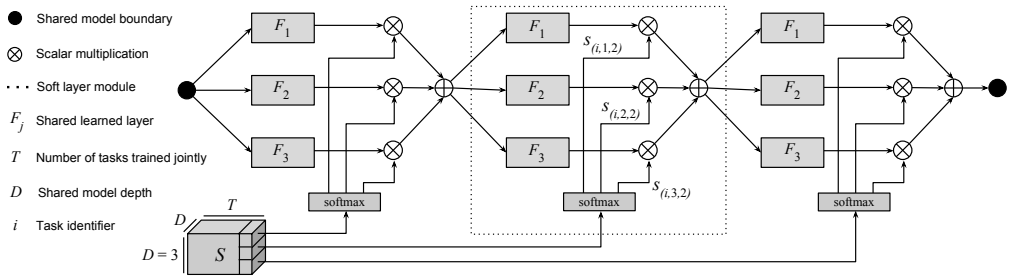


Figure 3: **Soft ordering of shared layers.** Sample soft ordering network with three shared layers. Soft ordering (Eq. 7) generalizes Eqs. 2 and 3, by learning a tensor S of task-specific scaling parameters. S is learned jointly with the F_j, to allow flexible sharing across tasks and depths. The F_j in this figure each include a shared weight layer and any nonlinearity. This architecture enables the learning of layers that are used in different ways at different depths for different tasks.

$G_1, \ldots, G_T$ such that $F_i = G_i G_{(i+1 \bmod T)} \ldots G_{(i-1 \bmod T)} \; \forall \, i$. Then, because the matrix trace is invariant under cyclic permutations, the constraint arises that

$$\mathrm{tr}(F_1) = \mathrm{tr}(F_2) = \ldots = \mathrm{tr}(F_T). \tag{5}$$

In the case of random matrices induced by the random tasks above, the traces of the F_i are all equal in expectation and concentrate well as their dimensionality increases. So, the restrictive effect of Eq. 5 on the expressivity of permuted ordering here is negligible.

Adding a small number of task-specific scaling parameters. Of course, real world tasks are generally much more structured than random ones, so such reliable expressivity of permuted ordering might not always be expected. However, adding a small number of task-specific scaling parameters can help adapt learned layers to particular tasks. This observation has been previously exploited in the parallel ordering setting, for learning task-specific batch normalization scaling parameters (Bilen and Vedaldi, 2017) and controlling communication between columns (Misra et al., 2016). Similarly, in the permuted ordering setting, the constraint induced by Eq. 5 can be reduced by adding task-specific scalars $\{s_i\}_{i=2}^T$ such that $F_i = s_i G_i G_{(i+1 \bmod T)} \ldots G_{(i-1 \bmod T)}$, and $s_1 = 1$. The constraint given by Eq. 5 then reduces to

$$\mathrm{tr}(F_i/s_i) = \mathrm{tr}(F_{i+1}/s_{i+1}) \; \forall \, 1 \leq i < T \implies s_{i+1} = s_i \big(\mathrm{tr}(F_{i+1})/\mathrm{tr}(F_i)\big), \tag{6}$$

which are defined when $\mathrm{tr}(F_i) \neq 0 \; \forall \, i < T$. Importantly, the number of task-specific parameters does not depend on m, which is useful for scalability as well as encouraging maximal sharing between tasks. The idea of using a small number of task-specific scaling parameters is incorporated in the soft ordering approach introduced in the next section.

3.3 Soft ordering of shared layers

Permuted ordering tests the parallel ordering assumption, but still fixes an *a priori* layer ordering for each task before training. Here, a more flexible *soft ordering* approach is introduced, which allows jointly trained models to learn *how* layers are applied while simultaneously learning the layers themselves. Consider again a core network of depth D with layers $W_1, \ldots, W_D$ learned and shared across tasks. The soft ordering model for task t_i is defined as follows:

$$y_i^k = \sum_{j=1}^{D} s_{(i,j,k)} (\phi_k[W_j(y_i^{k-1})]), \text{ with } \sum_{j=1}^{D} s_{(i,j,k)} = 1 \; \forall \, (i,k), \tag{7}$$

where $y_i^0 = \mathcal{E}_i(x_i)$, $y_i = \mathcal{D}_i(y_i^D)$, and each $s_{(i,j,k)}$ is drawn from S: a tensor of learned scales for each task t_i for each layer W_j at each depth k. Figure 3 shows an example of a resulting depth three model. Motivated by Section 3.2 and previous work (Misra et al., 2016), S adds only D^2 scaling parameters per task, which is notably not a function of the size of any W_j. The constraint that all $s_{(i,j,k)}$ sum to 1 for any (i,k) is implemented via softmax, and emphasizes the idea that a soft *ordering* is what is being learned; in particular, this formulation subsumes any fixed layer ordering σ_i by $s_{(i,\sigma_i(k),k)} = 1 \; \forall \, (i,k)$. S can be learned jointly with the other learnable parameters

in the W_k, $\mathcal{E}_i$, and $\mathcal{D}_i$ via backpropagation. In training, all $s_{(i,j,k)}$ are initialized with equal values, to reduce initial bias of layer function across tasks. It is also helpful to apply dropout after each shared layer. Aside from its usual benefits (Srivastava et al., 2014), dropout has been shown to be useful in increasing the generalization capacity of shared representations (Devin et al., 2016). Since the trained layers in Eq. 7 are used for different tasks and in different locations, dropout makes them more robust to supporting different functionalities. These ideas are tested empirically on the MNIST, UCI, Omniglot, and CelebA data sets in the next section.

4 Empirical Evaluation of Soft Layer Ordering

These experiments evaluate soft ordering against fixed ordering MTL and single-task learning. The first experiment applies them to closely related MNIST tasks, the second to "superficially unrelated" UCI tasks, the third to the real-world problem of Omniglot character recognition, and the fourth to large-scale facial attribute recognition. In each experiment, single task, parallel ordering (Eq. 2), permuted ordering (Eq. 3), and soft ordering (Eq. 7) train an equivalent set of core layers. In permuted ordering, the order of layers were randomly generated for each task each trial. See Appendix A for additional details, including additional details specific to each experiment.

4.1 Intuitively related tasks: MNIST digit$_1$-vs.-digit$_2$ binary classification

Binary classification problems derived from the MNIST hand-written digit dataset are a common test bed for evaluating deep learning methods that require multiple tasks, e.g., (Fernando et al., 2017; Kirkpatrick et al., 2017; Yang and Hospedales, 2017). Here, the goal of each task is to distinguish between two distinct randomly selected digits. To evaluate the ability of multitask models to exploit related tasks that have disparate representations, each $\mathcal{E}_i$ is a random frozen fully-connected ReLU layer with output size 64. There are four core layers, each a fully-connected ReLU layer with 64 units. Each $\mathcal{D}_i$ is an unshared dense layer with a single sigmoid binary classification output.

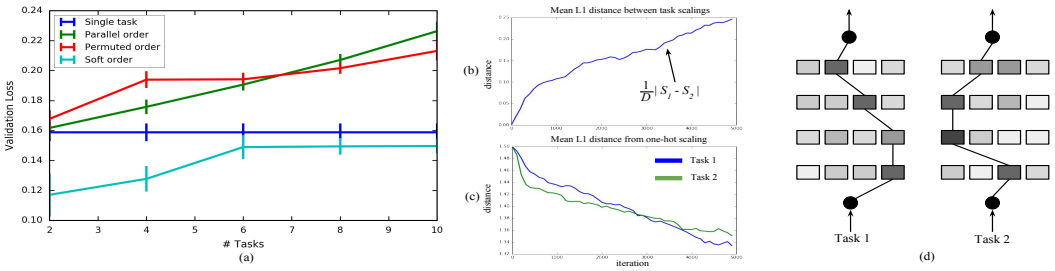


Figure 4: **MNIST results.** (a) Relative performance of permuted compared to parallel ordering improves as the number of tasks increases, while soft order outperforms the other methods for all numbers of tasks. For a representative two-task soft order experiment (b) the layer-wise distance between scalings of the tasks increases by iteration, and (c) the scalings move towards a hard ordering. (d) The final learned relative scale of each shared layer at each depth for each task is indicated by shading, with the strongest path drawn, showing that a distinct soft order is learned for each task (• marks the shared model boundary).

Results are shown in Figure 4. Relative performance of permuted ordering compared to parallel ordering increases with the number of tasks trained jointly (Figure 4a). This result is consistent with the hypothesis that parallel ordering has increased negative effects as the number of tasks increases. In contrast, soft ordering outperforms single-task learning, even at ten tasks, for which MTL has $\approx 1/10$ the parameters of single-task learning. Figure 4b-d show what soft ordering actually learns: The scalings for tasks diverge as layers specialize to different functions for different tasks.

4.2 Superifically unrelated tasks: Joint training of ten popular UCI datasets

The next experiment evaluates the ability of soft ordering to integrate information across a diverse set of "superficially unrelated" tasks. Ten tasks are taken from some of most popular UCI classification data sets (Lichman, 2013). Descriptions of these tasks are given in Figure 5a. Inputs and outputs

(a)

Dataset	Input Features	Output classes	Samples
Australian credit	14	2	690
Breast cancer	30	2	569
Ecoli	7	8	336
German credit	24	2	1000
Heart disease	13	5	303
Hepatitis	19	2	155
Iris	4	3	150
Pima diabetes	8	2	768
Wine	13	3	178
Yeast	8	10	1484

(b)

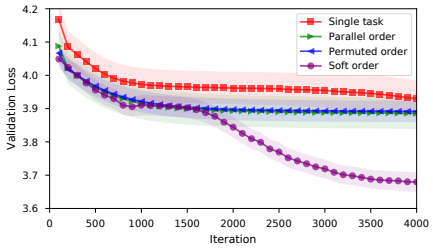


Figure 5: **UCI data sets and results.** (a) The ten UCI tasks used in joint training; the varying types of problems and dataset characteristics show the diversity of this set of tasks. (b) Combined loss over all ten tasks by iteration. Permuted and parallel order yield marginal improvements over single-task learning, while soft order decisively outperforms the other methods.

have no a priori shared meaning across tasks. Each $\mathcal{E}_i$ is a learned fully-connected ReLU layer with output size 32. There are four core layers, each a fully-connected ReLU layer with 32 units. Each $\mathcal{D}_i$ is an unshared dense softmax layer for the given number of classes. The results in Figure 5(b) show that, while parallel and permuted marginally outperform single-task learning, soft ordering significantly outperforms the other methods. With a flexible layer ordering, the model is eventually able to exploit significant regularities underlying these seemingly disparate domains.

4.3 Extension to convolutions: Multi-alphabet character recognition

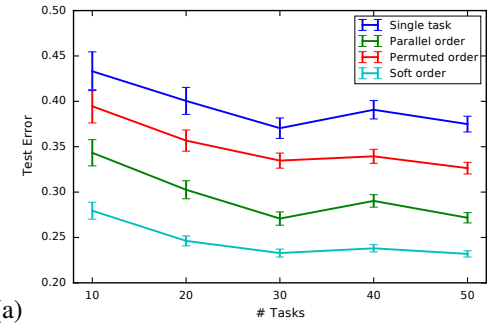
The Omniglot dataset (Lake et al., 2015) consists of fifty alphabets, each of which induces a different character recognition task. Deep MTL approaches have recently shown promise on this dataset (Yang and Hospedales, 2017). It is a useful benchmark for MTL because the large number of tasks allows analysis of performance as a function of the number of tasks trained jointly, and there is clear intuition for how knowledge of some alphabets will increase the ability to learn others. Omniglot is also a good setting for evaluating the ability of soft ordering to learn how to compose layers in different ways for different tasks: it was developed as a problem with inherent composibility, e.g., similar kinds of strokes are applied in different ways to draw characters from different alphabets (Lake et al., 2015). Consequently, it has been used as a test bed for deep generative models (Rezende et al., 2016). To evaluate performance for a given number of tasks T, a single random ordering of tasks was created, from which the first T tasks are considered. Train/test splits are created in the same way as previous work (Yang and Hospedales, 2017), using 10% or 20% of data for testing.

This experiment is also a scale-up of the previous experiments in that it evaluates soft ordering of convolutional layers. The models are made as close as possible in architecture to previous work (Yang and Hospedales, 2017), while allowing soft ordering to be applied. There are four core layers, each convolutional followed by max pooling. $\mathcal{E}_i(x_i) = x_i \; \forall \; i$, and each $\mathcal{D}_i$ is a fully-connected softmax layer with output size equal to the number of classes. The results show that soft ordering is able to consistently outperform other deep MTL approaches (Figure 6). The improvements are robust to the number of tasks (Figure 6a) and the amount of training data (Figure 6b), suggesting that soft ordering, not task complexity or model complexity, is responsible for the improvement.

4.4 Large-scale Application: Facial Attribute Recognition

Although facial attributes are all high-level concepts, they do not intuitively exist at the same level of a shared hierarchy (even one that is learned; Lu et al., 2017). Rather, these concepts are related in multiple subtle and overlapping ways in semantic space. This experiment investigates how a soft ordering approach, as a component in a larger system, can exploit these relationships.

The CelebA dataset consists of ≈200K 178×218 color images, each with binary labels for 40 facial attributes (Liu et al., 2015b). In this experiment, each label defines a task, and parallel and soft order models are based on a ResNet-50 vision model (He et al., 2016), which has also been used in recent state-of-the-art approaches to CelebA (Günther et al., 2017; He et al., 2017). Let $\mathcal{E}_i$ be a ResNet-50 model truncated to the final average pooling layer, followed by a linear layer projecting the embedding to size 256. $\mathcal{E}_i$ is shared across all tasks. There are four core layers, each a dense



(a)

Deep MTL method	10% Test Split	20% Test Split
STL	34.36 (± 0.53)	35.92 (± 0.74)
UD-MTL	29.98 (± 1.33)	29.53 (± 0.99)
DMTRL-LAF	31.08 (± 0.65)	33.37 (± 0.97)
DMTRL-Tucker	29.67 (± 1.25)	31.11 (± 1.16)
DMTRL-TT	28.78 (± 0.61)	30.61 (± 0.65)
Single task (ours)	38.49 (± 0.87)	38.10 (± 0.88)
Parallel order	27.17 (± 0.57)	28.24 (± 0.67)
Permuted order	32.64 (± 0.64)	33.18 (± 0.74)
Soft order	**23.19** (± 0.34)	**24.11** (± 0.48)

(b)

Figure 6: **Omniglot results.** (a) Error by number of tasks trained jointly. Soft ordering significantly outperforms single task and both fixed ordering approaches for each number of tasks; (b) Errors with all 50 tasks for different training set sizes. The first five methods are previous deep MTL results (Yang and Hospedales, 2017), which use multitask tensor factorization methods in a shared parallel ordering. Soft ordering significantly outperforms the other approaches, showing the approach scales to real-world tasks requiring specialized components such as convolutional layers.

ReLU layer with 256 units. Each $\mathcal{D}_i$ is an unshared dense sigmoid layer. Two models were trained: one with parallel ordering and one with soft ordering. Existing work that used a ResNet-50 vision model showed that using a parallel order multitask model improved test accuracy over single-task learning from 89.63 to 90.42 (He et al., 2017). With our faster training strategy and the added core layers, our parallel ordering model achieves a test accuracy of 89.79. The soft ordering model yields a substantial improvement beyond this to 91.21, demonstrating that soft ordering can add value to a larger deep learning system. Note that previous work has shown that adaptive weighting of task loss (He et al., 2017; Rudd et al., 2016), data augmentation and ensembling (Günther et al., 2017), and a larger underlying vision model (Lu et al., 2017) each can also yield significant improvements. Aside from soft ordering, none of these improvements alter the *multitask topology*, so their benefits are expected to be complementary to that of soft ordering demonstrated in this experiment. By coupling them with soft ordering, greater improvements should be possible.

5 Visualizing the Behavior of Soft Ordering Layers

The success of soft layer ordering suggests that layers learn functional primitives with similar effects in different contexts. To explore this idea qualitatively, the following experiment uses generative visual tasks. The goal of each task is to learn a function $(x, y) \to v$, where (x, y) is a pixel coordinate and v is a brightness value, all normalized to $[0, 1]$. Each task is defined by a single image of a "4" drawn from the MNIST dataset; all of its pixels are used as training data. Ten tasks are trained using soft ordering with four shared dense ReLU layers of 100 units each. $\mathcal{E}_i$ is a linear encoder that is shared across tasks, and $\mathcal{D}_i$ is a global average pooling decoder. Thus, task models are distinguished completely by their learned soft ordering scaling parameters s_t. To visualize the behavior of layer l at depth d for task t, the predicted image for task t is generated across varying magnitudes of $s_{(t,l,d)}$. The results for the first two tasks and the first layer are shown in Table 1. Similar function is observed in each of the six contexts, suggesting that the layers indeed learn functional primitives.

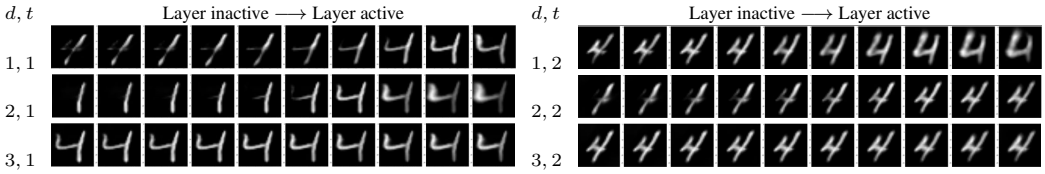


Table 1: **Example behavior of a soft order layer**. For each task t, and at each depth d, the effect of increasing the activation of of this particular layer is to expand the left side of the "4" in a manner appropriate to the functional context (e.g., the magnitude of the effect decreases with depth). Results for other layers are similar, suggesting that the layers implement functional primitives.

6 DISCUSSION AND FUTURE WORK

In the interest of clarity, the soft ordering approach in this paper was developed as a relatively small step away from the parallel ordering assumption. To develop more practical and specialized methods, inspiration can be taken from recurrent architectures, the approach can be extended to layers of more general structure, and applied to training and understanding general functional building blocks.

Connections to recurrent architectures. Eq. 7 is defined recursively with respect to the learned layers shared across tasks. Thus, the soft-ordering architecture can be viewed as a new type of recurrent architecture designed specifically for MTL. From this perspective, Figure 3 shows an unrolling of a *soft layer module*: different scaling parameters are applied at different depths when unrolled for different tasks. Since the type of recurrence induced by soft ordering does not require task input or output to be sequential, methods that use recurrence in such a setting are of particular interest (Liang and Hu, 2015; Liao and Poggio, 2016; Pinheiro and Collobert, 2014; Socher et al., 2011; Zamir et al., 2016). Recurrent methods can also be used to reduce the size of S below $O(TD^2)$, e.g., via recurrent hypernetworks (Ha et al., 2016). Finally, Section 4 demonstrated soft ordering where shared learned layers were fully-connected or convolutional; it is also straightforward to extend soft ordering to shared layers with internal recurrence, such as LSTMs (Hochreiter and Schmidhuber, 1997). In this setting, soft ordering can be viewed as inducing a higher-level recurrence.

Generalizing the structure of shared layers. For clarity, in this paper all core layers in a given setup had the same shape. Of course, it would be useful to have a generalization of soft ordering that could subsume any modern deep architecture with many layers of varying structure. As given by Eq. 7, soft ordering requires the same shape inputs to the element-wise sum at each depth. Reshapes and/or resampling can be added as adapters between tensors of different shape; alternatively, a function other than a sum could be used. For example, instead of learning a weighting across layers at each depth, a probability of applying each module could be learned in a manner similar to adaptive dropout (Ba and Frey, 2013; Li et al., 2016) or a sparsely-gated mixture of experts (Shazeer et al., 2017). Furthermore, the idea of a soft ordering of layers can be extended to soft ordering over modules with more general structure, which may more succinctly capture recurring modularity.

Training generalizable building blocks. Because they are used in different ways at different locations for different tasks, the shared trained layers in permuted and soft ordering have learned more general functionality than layers trained in a fixed location or for a single task. A natural hypothesis is that they are then more likely to generalize to future unseen tasks, perhaps even without further training. This ability would be especially useful in the small data regime, where the number of trainable parameters should be limited. For example, given a collection of these layers trained on a previous set of tasks, a model for a new task could learn how to apply these building blocks, e.g., by learning a soft order, while keeping their internal parameters fixed. Learning an efficient set of such generalizable layers would then be akin to learning a set of *functional primitives*. Such functional modularity and repetition is evident in the natural, technological and sociological worlds, so such a set of functional primitives may align well with complex real-world models. This perspective is related to recent work in reusing modules in the parallel ordering setting (Fernando et al., 2017). The different ways in which different tasks learn to use the same set of modules can also help shed light on how tasks are related, especially those that seem superficially disparate (e.g., by extending the analysis performed for Figure 4d), thus assisting in the discovery of real-world regularities.

7 CONCLUSION

This paper has identified *parallel ordering* of shared layers as a common assumption underlying existing deep MTL approaches. This assumption restricts the kinds of shared structure that can be learned between tasks. Experiments demonstrate how direct approaches to removing this assumption can ease the integration of information across plentiful and diverse tasks. *Soft ordering* is introduced as a method for learning how to apply layers in different ways at different depths for different tasks, while simultaneously learning the layers themselves. Soft ordering is shown to outperform parallel ordering methods as well as single-task learning across a suite of domains. These results show that deep MTL can be improved while generating a compact set of multipurpose functional primitives, thus aligning more closely with our understanding of complex real-world processes.

REFERENCES

M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

J. Ba and B. Frey. Adaptive dropout for training deep neural networks. In *NIPS*, pages 3084–3092. 2013.

H. Bilen and A. Vedaldi. Integrated perception with recurrent multi-task neural networks. In *NIPS*, pages 235–243. 2016.

H. Bilen and A. Vedaldi. Universal representations: The missing link between faces, text, planktons, and cat breeds. *CoRR*, abs/1701.07275, 2017.

R. Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer US, 1998.

F. Chollet et al. Keras, 2015.

R. Collobert and J. Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *ICML*, pages 160–167, 2008.

C. Devin, A. Gupta, T. Darrell, P. Abbeel, and S. Levine. Learning modular neural network policies for multi-task and multi-robot transfer. *CoRR*, abs/1609.07088, 2016.

D. Dong, H. Wu, W. He, D. Yu, and H. Wang. Multi-task learning for multiple language translation. In *ACL*, pages 1723–1732, 2015.

C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel, and D. Wierstra. Pathnet: Evolution channels gradient descent in super neural networks. *CoRR*, abs/1701.08734, 2017.

M. Günther, A. Rozsa, and T. E. Boult. AFFACT - alignment free facial attribute classification technique. *CoRR*, abs/1611.06158v2, 2017.

D. Ha, A. M. Dai, and Q. V. Le. Hypernetworks. *CoRR*, abs/1609.09106, 2016.

K. Hashimoto, C. Xiong, Y. Tsuruoka, and R. Socher. A joint many-task model: Growing a neural network for multiple NLP tasks. *CoRR*, abs/1611.01587, 2016.

K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778, 2016.

K. He, Z. Wang, Y. Fu, R. Feng, Y.-G. Jiang, and X. Xue. Adaptively weighted multi-task deep network for person attribute classification. 2017.

S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. ISSN 0899-7667.

J. T. Huang, J. Li, D. Yu, L. Deng, and Y. Gong. Cross-language knowledge transfer using multilingual deep neural network with shared hidden layers. In *ICASSP*, pages 7304–7308, 2013.

Z. Huang, J. Li, S. M. Siniscalchi, I.-F. Chen, J. Wu, and C.-H. Lee. Rapid adaptation for deep neural networks through multi-task learning. In *INTERSPEECH*, 2015.

M. Jaderberg, V. Mnih, W. M. Czarnecki, T. Schaul, J. Z. Leibo, D. Silver, and K. Kavukcuoglu. Reinforcement learning with unsupervised auxiliary tasks. In *ICLR*, 2017.

B. Jou and S.-F. Chang. Deep cross residual learning for multitask visual recognition. In *MM*, pages 998–1007, 2016.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, et al. Overcoming catastrophic forgetting in neural networks. *PNAS*, 114(13):3521–3526, 2017.

B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

Z. Li, B. Gong, and T. Yang. Improved dropout for shallow and deep learning. In *NIPS*, pages 2523–2531. 2016.

M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *CVPR*, 2015.

Q. Liao and T. A. Poggio. Bridging the gaps between residual learning, recurrent neural networks and visual cortex. *CoRR*, abs/1604.03640, 2016.

M. Lichman. UCI machine learning repository, 2013.

X. Liu, J. Gao, X. He, L. Deng, K. Duh, and Y. Y. Wang. Representation learning using multi-task deep neural networks for semantic classification and information retrieval. In *NAACL*, pages 912–921, 2015a.

Z. Liu, P. Luo, X. Wang, and X. Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, 2015b.

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

M. Luong, Q. V. Le, I. Sutskever, O. Vinyals, and L. Kaiser. Multi-task sequence to sequence learning. In *ICLR*, 2016.

M. H. Mahmud. On universal transfer learning. *Theoretical Computer Science*, 410(19):1826 – 1846, 2009. ISSN 0304-3975.

M. M. Mahmud and S. Ray. Transfer learning using Kolmogorov complexity: Basic theory and empirical evaluations. In *NIPS*, pages 985–992. 2008.

I. Misra, A. Shrivastava, A. Gupta, and M. Hebert. Cross-stitch networks for multi-task learning. In *CVPR*, 2016.

P. Pinheiro and R. Collobert. Recurrent convolutional neural networks for scene labeling. In *ICML*, pages 82–90, 2014.

R. Ranjan, V. M. Patel, and R. Chellappa. Hyperface: A deep multi-task learning framework for face detection, landmark localization, pose estimation, and gender recognition. *CoRR*, abs/1603.01249, 2016.

D. Rezende, Shakir, I. Danihelka, K. Gregor, and D. Wierstra. One-shot generalization in deep generative models. In *ICML*, pages 1521–1529, 2016.

E. M. Rudd, M. Günther, and T. E. Boult. MOON: A mixed objective optimization network for the recognition of facial attributes. In *ECCV*, pages 19–35, 2016.

A. A. Rusu, N. C. Rabinowitz, G. Desjardins, H. Soyer, et al. Progressive neural networks. *CoRR*, abs/1606.04671, 2016.

A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *CoRR*, abs/1312.6120, 2013.

M. L. Seltzer and J. Droppo. Multi-task learning in deep neural networks for improved phoneme recognition. In *ICASSP*, pages 6965–6969, 2013.

N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *ICLR*, 2017.

R. Socher, C. C.-Y. Lin, A. Y. Ng, and C. D. Manning. Parsing natural scenes and natural language with recursive neural networks. In *ICML*, pages 129–136, 2011.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *JMLR*, 15(1):1929–1958, 2014.

S. Toshniwal, H. Tang, L. Lu, and K. Livescu. Multitask Learning with Low-Level Auxiliary Tasks for Encoder-Decoder Based Speech Recognition. *CoRR*, abs/1704.01631, 2017.

A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *NIPS*, pages 550–558. 2016.

Z. Wu, C. Valentini-Botinhao, O. Watts, and S. King. Deep neural networks employing multi-task learning and stacked bottleneck features for speech synthesis. In *ICASSP*, pages 4460–4464, 2015.

Y. Yang and T. Hospedales. Deep multi-task representation learning: A tensor factorisation approach. In *ICLR*, 2017.

A. R. Zamir, T. Wu, L. Sun, W. Shen, J. Malik, and S. Saverese. Feedback networks. *CoRR*, abs/1612.09508, 2016.

Y. Zhang and D. Weiss. Stack-propagation: Improved representation learning for syntax. *CoRR*, abs/1603.06598, 2016.

Z. Zhang, L. Ping, L. C. Chen, and T. Xiaoou. Facial landmark detection by deep multi-task learning. In *ECCV*, pages 94–108, 2014.

A Experimental Details

All experiments were run with the Keras deep learning framework Chollet et al. (2015), using the Tensorflow backend (Abadi et al., 2015). All experiments used the Adam optimizer with default parameters (Kingma and Ba, 2014).

In each iteration of multitask training, a random batch for each task is processed, and the results are combined across tasks into a single update. Compared to alternating batches between tasks (Luong et al., 2016), processing all tasks simultaneously simplified the training procedure, and led to faster and lower final convergence. When encoders are shared, the inputs of the samples in each batch are the same across tasks. Cross-entropy loss was used for all classification tasks. The overall validation loss is the sum over all per task validation losses.

In each experiment, single task, parallel ordering (Eq. 2), permuted ordering (Eq. 3), and soft ordering (Eq. 7) trained an equivalent set of core layers. In permuted ordering, the order of layers was randomly generated for each task each trial. Several trials were run for each setup to produce confidence bounds.

A.1 MNIST experiments

Input pixel values were normalized to be between 0 and 1. The training and test sets for each task were the MNIST train and test sets restricted to the two selected digits. A dropout rate of 0.5 was applied at the output of each core layer.

When randomly selecting the pairs of digits that define a set of tasks, digits were selected without replacement within a task, and with replacement across tasks, so there were 45 possible tasks, and 45^k possible sets of tasks of size k.

A.2 UCI experiments

For all tasks, each input feature was scaled to be between 0 and 1. For each task, training and validation data were created by a random 80-20 split. This split was fixed across trials. A dropout rate of 0.8 was applied at the output of each core layer.

A.3 Omniglot experiments

To enable soft ordering, the output of all shared layers must have the same shape. For comparability, the models were made as close as possible in architecture to previous work (Yang and Hospedales, 2017), in which models had four sharable layers, three of which were 2D convolutions followed by 2×2 max-pooling, of which two had 3×3 kernels. So, in this experiment, to evaluate soft ordering of convolutional layers, there were four core layers, each a 2D convolutional layer with ReLU activation and kernel size 3×3. Each convolutional layer was followed by a 2×2 max-pooling layer. The number of filters for each convolutional layer was set at 53, which makes the number of total model parameters as close as possible to the reference model. A dropout rate of 0.5 was applied at the output of after each core layer.

The Omniglot dataset consists of 101×101 black-and-white images. There are fifty alphabets of characters and twenty images per character. To be compatible with the shapes of shared layers, the input was zero-padded along the third dimension so that its shape was $101 \times 101 \times 53$, i.e., with the

first 101×101 slice containing the image data and the remainder zeros. To evaluate approaches on k tasks, a random ordering of the fifty tasks was created and fixed across all trials. In each trial, the first k tasks in this ordering were trained jointly.

A.4 CELEBA EXPERIMENTS

The training, validation, and test splits provided by Liu et al. (2015b) were used. There are ≈160K images for training, ≈20K for validation, and ≈20K for testing. The dataset contains 20 images of each of approximately ≈10K celebrities. The images for a given celebrity occur in only one of the three dataset splits, so models must also generalize to new human identities.

The weights for ResNet-50 were initialized with the pre-trained imagenet weights provided in the Keras framework Chollet et al. (2015). Image preprocessing was done with the default Keras image preprocessing function, including resizing all images to 224×224. A dropout rate of 0.5 was applied at the output of after each core layer. The experiments used a batch size of 32. After validation loss converges via Adam, models are trained with RMSProp with learning rate $1e^{-5}$, which is a similar approach to that used by Günther et al. (2017).

A.5 EXPERIMENTS ON VISUALIZING LAYER BEHAVIOR

To produce the resulting image for a fixed model, the predictions at each pixel locations were generated, denormalized, and mapped back to the pixel coordinate space. The loss used for this experiment was mean squared error (MSE). Since all pixels for a task image are used for training, there is no sense of generalization to unseen data within a task. As a result, no dropout was used in this experiment.

Task models are distinguished completely by their learned soft ordering scaling parameters s_t, so the joint model can be viewed as a generative model which generates different 4's for varying values of s_t. To visualize the behavior of layer l at depth d for task t, the output of the model for task t was visualized while sweeping $s_{(t,l,d)}$ across $[0, 1]$. To enable this sweeping while keeping the rest of the model behavior fixed, the softmax for each task at each depth was replaced with a sigmoid activation. Note that due to the global avgerage pooling decoder, altering the weight of a single layer has no observable effect at depth four.