Supplementary material for TASK2VEC : Task Embeddings for Meta-Learning

1. Additional experiments

1.1. Taskonomy tasks: embedding different tasks sharing the same input

We compute task embeddings of tasks from the Taskonomy dataset [3] using a ResNet-34 probe network pretrained on ImageNet[2] used for the other experiments, and a small subset of the available data.[1] Using the TASK2VEC embeddings we can compute the full distance matrix (Figure 1), which shows intuitive clustering into 2D, 3D and "semantic" tasks similarly to what observed by [3], but using far less compute (we use about 5 GPU hours for the whole matrix). Note in particular, we recover a meaningful embedding even for tasks for which the probe network was not optimized for (*e.g.*, depth regression whereas the probe network is trained for semantic classification).

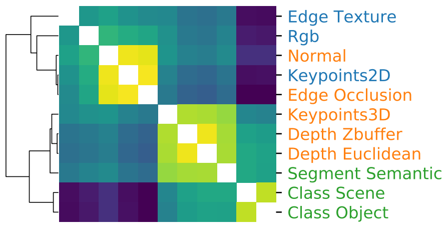

Figure 1: TASK2VEC distance between Taskonomy tasks. As in [3], similar tasks cluster together in the TASK2VEC embedding: (blue) 2D tasks, (orange) 3D tasks, (green) semantic tasks.

We use the same experimental setup as the other experiments (see also Section 4) with minor modifications. The initial part of a ResNet-34 architecture was pretrained on ImageNet and the final part of the network was replaced by a simple decoder. If the task is label prediction we use a linear classifier in the form `conv3x3 256` → `global avg. pool` → `linear` → `softmax`, where `conv3x3` is a convolution with 3 x 3 kernel, followed by a batch normalization layer and ReLU nonlinearity. If instead the task requires pixel-wise prediction (*e.g.*, segmentation) or regression (*e.g.*, depth prediction), we replace the final part of the network with a deconvolutional architecture: `conv3x3 256` → `conv3x3 256` → `upsample 32` → `conv3x3 128` → `upsample 64` → `conv3x3 128` → `upsample 128` → `conv3x3 64` → `upsample 256x256` → `conv3x3 32` → `conv1x1 out_channels`, where `upsample` denotes nearest neighbor upsampling of the filter responses to the specified resolution, and `conv1x1` is a convolution with kernel size 1 x 1 (without any non-linearity). The final output is fed to a pixel-wise softmax if the task requires pixelwise segmentation, or treated directly as the regression output if the task requires pixelwise regression. We train the decoder on each tasks, and then embed the task using the Fisher Information Matrix computed with respect to the pre-trained fixed weights.

1.2. Tasks with less samples have embeddings of smaller norm

Figure 2 shows that, as the number of training samples increases, the norm of the embedding also increases across a range of tasks. This is well aligned with the intuition that the norm of the embedding captures the complexity of the learning task, and that learning tasks with more samples are more complex to learn. Note that this property is implicitly exploited when selecting models using our asymmetric distance, since models trained on less data will be penalized (they are closer to the trivial zero embedding).

[1] https://github.com/alexsax/taskonomy-sample-model-1

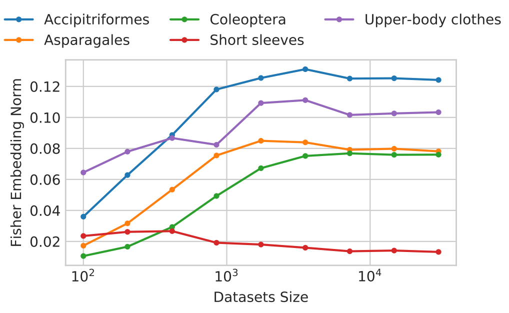

Figure 2: Norm of the task embedding as the number of samples in the dataset varies. Larger datasets generally have larger norms, hence TASK2VEC enables distinguishing tasks that only differ by the number of shots (training samples/class).

2. Complete error matrices for the meta-tasks

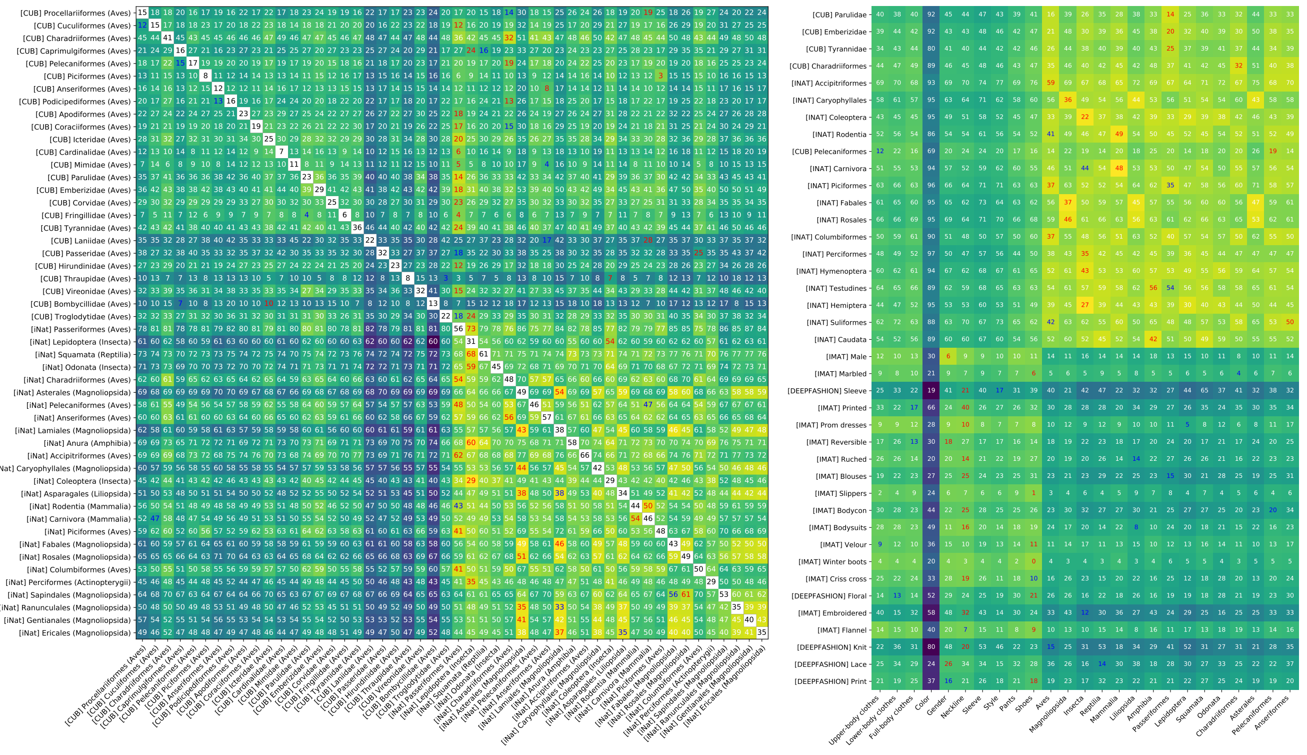

Figure 3: **Meta-tasks ground-truth error matrices and TASK2VEC selections.** (Best viewed magnified). **(Left)** Error matrix for the CUB+iNat meta-task. The numbers in each cell is the test error obtained by training a classifier on a given combination of task (rows) and expert (columns). The background color represent the Asymmetric TASK2VEC distance between the target task and the task used to train the expert. Numbers in red indicate the selection made by the model selection algorithm based on the Asymmetric TASK2VEC embedding. The (out-of-diagonal) optimal expert (when different from the one selected by our algorithm), is highlighted in blue. **(Right)** Same as before, but for the Mixed meta-task.

3. Description of datasets, tasks and meta-tasks

Our two model selection meta-tasks, **iNat+CUB** and **Mixed**, are curated as follows. For **iNat+CUB**, we generated 50 tasks and (the same) experts from iNaturalist and CUB. The 50 tasks consist of 25 iNaturalist tasks and 25 CUB tasks to provide a balanced mix from two datasets of the same domain. We generated the 25 iNaturalist tasks by grouping species into orders and then choosing the top 25 orders with the most samples. The number of samples for tasks shows the heavy-tail distribution typical of real data, with the top task having 64,100 samples (the *Passeriformes* order classification task), while most tasks have around $6,000$ samples.

The 25 CUB tasks were similarly generated with 10 order tasks but additionally has 15 Passeriformes family tasks: After grouping CUB into orders, we determined 11 usable order tasks (the only unusable order task, Gaviiformes, has only one species so it makes no sense to train on it). However, one of the orders—Passeriformes—dominated all other orders with 134 species when compared to 3-24 species of other orders. Therefore, we decided to further subdivide the Passeriformes order task into family tasks (*i.e.*, grouping species into families) to provide a more balanced partition. This resulted in 15 usable family tasks (*i.e.*, has more than one species) out of 22 family tasks. Unlike iNaturalist, tasks from CUB have only a few hundreds of samples and hence benefit more from carefully selecting an expert.

In the iNAT+CUB meta-task the classification tasks are the same tasks used to train the experts. To avoid trivial solutions (always selecting the expert trained on the task we are trying to solve) we test in a leave-one-out fashion: given a classficication task, we aim to select the best expert that was not trained on the same data.

For the **Mixed** meta-task, we chose 40 random tasks and 25 curated experts from all datasets. The 25 experts were generated from iNaturalist, iMaterialist and DeepFashion (CUB, having fewer samples than iNaturalist, is more appropriate as tasks). For iNaturalist, we trained 15 experts: 8 order tasks and 7 class tasks (species ordered by class), both with number of samples greater than 10,000. For DeepFashion, we trained 3 category experts (upper-body, lower-body, full-body). For iMaterialist, we trained 2 category experts (pants, shoes) and 5 multi-label experts by grouping attributes (color, gender, neckline, sleeve, style). For the purposes of clustering attributes into larger groups for training experts (and color coding the dots in Figure 1), we obtained a de-anonymized list of the iMaterialist Fashion attribute names from the FGVC contest organizers.

The 40 random tasks were generated as follows. In order to balance tasks among all datasets, we selected 5 CUB, 15 iNaturalist, 15 iMaterialist and 5 DeepFashion tasks. Within those datasets, we randomly pick tasks with a sufficient number of validation samples and maximum variety. For the iNaturalist tasks, we group the order tasks into class tasks, filter out the number of validation samples less than 100 and randomly pick order tasks within each class. For the iMaterialist tasks, we similarly group the tasks (e.g. category, style, pattern), filter out tasks with less than 1,000 validation samples and randomly pick tasks within each group. For CUB, we randomly select 2 order tasks and 3 Passeriformes family tasks, and for DeepFashion, we randomly select the tasks uniformly. All this ensures that we have a balanced variety of tasks.

For the **data efficiency** experiment, we trained on a subset of the tasks and experts in the Mixed meta-task: We picked the Accipitriformes, Asparagales, Upper-body, Short Sleeves for the tasks, and the Color, Lepidoptera, Upper-body, Passeriformes, Asterales for the experts. Tasks where selected among those that have more than 30,000 training samples in order to represent all datasets. The experts were also selected to be representative of all datasets, and contain both strong and very weak experts (such as the Color expert).

4. Details of the experiments

4.1. Training of experts and classifiers

Given a task, we train an *expert* on it by fine-tuning an off-the-shelf ResNet-34 pretrained on ImageNet[2]. Fine-tuning is performed by first fixing the weights of the network and retraining from scratch only the final classifier for 10 epochs using Adam, and then fine-tuning all the network together with SGD for 60 epochs with weight decay 5e-4, starting from learning rate 0.001 and decreasing it by a factor 0.1 at epochs 40.

Given an expert, we train a classifier on top of it by replacing the final classification layer and training it with Adam for 16 epochs. We use weight decay 5e-4 and learning rate 1e-4.

The tasks we train on generally have different number of samples and unbalanced classes. To limit the impact of this imbalance on the training procedure, regardless of the total size of the dataset, in each epoch we always sample 10,000 images with replacement, uniformly between classes. In this way, all epochs have the same length and see approximately the same number of examples for each class. We use this balanced sampling in all experiments, unless noted otherwise.

4.2. Computation of the TASK2VEC embedding

As the described in the main text, the TASK2VEC embedding is obtained by choosing a probe network, retraining the final classifier on the given task, and then computing the Fisher Information Matrix for the weights of the probe network.

[2] https://pytorch.org/docs/stable/torchvision/models.html

Unless specified otherwise, we use an off-the-shelf ResNet-34 pretrained on ImageNet as the probe network. The Fisher Information Matrix is computed in a robust way minimizing the loss function $L(\hat{w};\Lambda)$ with respect to the precision matrix Λ, as described before. To make computation of the embedding faster, instead of waiting for the convergence of the classifier, we train the final classifier for 2 epochs using Adam and then we continue to train it jointly with the precision matrix Λ using the loss $L(\hat{w};\Lambda)$. We constrain Λ to be positive by parametrizing it as $\Lambda = \exp(L)$, for some unconstrained variable L. While for the classifier we use a low learning rate (1e-4), we found it useful to use an higher learning rate (1e-2) to train L.

4.3. Training the MODEL2VEC embedding

As described in the main text, in the MODEL2VEC embedding we aim to learn a vector representation $m_j = F_j + b_j$ of the j-th model in the collection, which represents both the task the model was trained on (through the TASK2VEC embedding F_j), and the particularities of the model (through the learned parameter b_j).

We learn b_j by minimizing a k-way classification loss which, given a task t, aims to select the model that performs best on the task among a collection of k models. Multiple models may perform similarly and close to optimal: to preserve this information, instead of using a one-hot encoding for the best model, we train using soft-labels obtained as follows:

$$\hat{p}(y_i) = \text{Softmax}\Big(-\alpha \frac{\text{error}_i - \text{mean}(\text{error}_i)}{\text{std}(\text{error}_i)} \Big),$$

where $\text{error}_{i,j}$ is the ground-truth test error obtained by training a classifier for task i on top of the j-th model. Notice that for $\alpha \gg 1$, the soft-label y_i^j reduces to the one-hot encoding of the index of the best performing model. However, for lower α's, the vector y_i contains richer information about the relative performance of the models.

We obtain our prediction in a similar way: Let $d_{i,j} = d_{\text{asym}}(t_i, m_j)$, then we set our model prediction to be

$$p(y|d_{i,0}, \dots, d_{i,k}) = \text{Softmax}(-\gamma\, d_i),$$

where the scalar $\gamma > 0$ is a learned parameter. Finally, we learn both the m_j's and γ using a cross-entropy loss:

$$\mathcal{L} = \frac{1}{N} \sum_{i=0}^{N} \mathbb{E}_{y_i \sim \hat{p}}[p(y_i|d_{i,0}, \dots, d_{i,k})],$$

which is minimized precisely when $p(y|d_{i,0}, \dots, d_{i,k}) = \hat{p}(y_i)$.

In our experiments we set $\alpha = 20$, and minimize the loss using Adam with learning rate 0.05, weight decay 0.0005, and early stopping after 81 epochs, and report the leave-one-out error (that is, for each task we train using the ground truth of all other tasks and test on that task alone, and report the average of the test errors obtained in this way).

5. Analytic FIM for two-layer model

Assume we have data points $(x_i, y_i), i = 1 \dots n$ and $y_i \in \{0, 1\}$. Assume that a fixed feature extractor applied to data points x yields features $z = \phi(x) \in \mathbb{R}^d$ and a linear model with parameters w is trained to model the conditional distribution $p_i = P(y_i = 1|x_i) = \sigma\left(w^T \phi(x_i)\right)$, where σ is the sigmoid function. The gradient of the cross-entropy loss with respect to the linear model parameters is:

$$\frac{\partial \ell}{\partial w} = \frac{1}{N} \sum_i (y_i - p_i)\phi(x_i),$$

and the empirical estimate of the Fisher information matrix is:

$$\begin{aligned} F = \mathbb{E}\Big[\frac{\partial \ell}{\partial w} \left(\frac{\partial \ell}{\partial w}\right)^T\Big] &= \mathbb{E}_{y \sim p_w(y|x)} \frac{1}{N} \sum_i \phi(x_i)(y_i - p_i)^2 \phi(x_i)^T \\ &= \frac{1}{n} \sum_i \phi(x_i)(1 - p_i)p_i \phi(x_i)^T \end{aligned}$$

In general, we are also interested in the Fisher information of the parameters of the feature extractor $\phi(x)$ since this is independent of the specifics of the output space y (e.g., for k-way classification). Consider a 2-layer network where the feature extractor uses a sigmoid non-linearity:

$$p = \sigma(w^T z) \quad z_k = \sigma(U_k^T x)$$

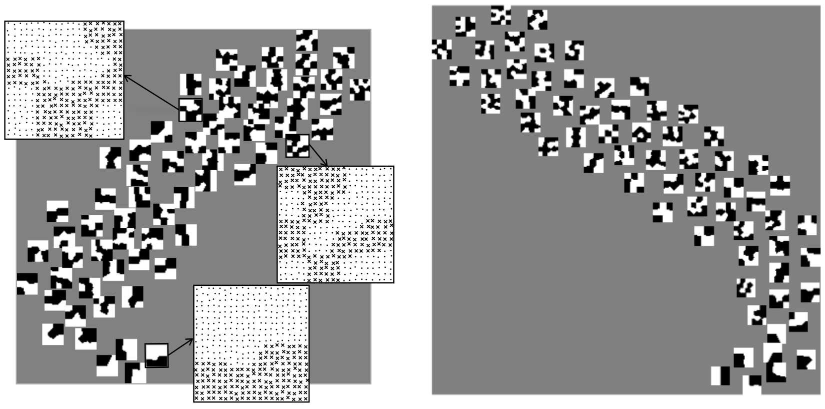
(a) Random linear + ReLU (b) Polynomial of degree three

Figure 4: Task embeddings computed with a probe network consisting of (a) 10 random linear + ReLU features and (b) degree three polynomial features projected to 2D using t-SNE. Each icon shows a binary classification task on a unit (three tasks are magnified on the left). In this case tasks cannot be distinguished based on the input domain. Both probe networks group tasks that are qualitatively similar together, with more complex that have have complicated decision boundaries well separated from the simpler ones.

and the matrix U specifies the feature extractor parameters and w are parameters of the task-specific classifier. Taking the gradient w.r.t. parameters we have:

$$\frac{\partial \ell}{\partial w_j} = (y-p)z_j$$
$$\frac{\partial \ell}{\partial U_{kj}} = (y-p)w_k z_k(1-z_k)x_j$$

The Fisher Information Matrix (FIM) consists of blocks:

$$\frac{\partial \ell}{\partial w_i}\left(\frac{\partial \ell}{\partial w_j}\right)^T = (y-p)^2 z_i z_j$$
$$\frac{\partial \ell}{\partial U_{ki}}\left(\frac{\partial \ell}{\partial w_j}\right)^T = (y-p)^2 z_j z_k(1-z_k)x_i$$
$$\frac{\partial \ell}{\partial U_{li}}\left(\frac{\partial \ell}{\partial U_{kj}}\right)^T = (y-p)^2 w_k z_k(1-z_k)w_l z_l(1-z_l)x_i x_j$$

We focus on the FIM of the probe network parameters which is independent of the dimensionality of the output layer and write it in matrix form as:

$$\frac{\partial \ell}{\partial U_l}\left(\frac{\partial \ell}{\partial U_k}\right)^T = (y-p)^2(1-z_k)z_k(1-z_l)z_l w_k w_l x x^T$$

Note that each block $\{l,k\}$ consists of the same matrix $(y-p)^2 \cdot xx^T$ multiplied by a scalar S_{kl} given as:

$$S_{kl} = (1-z_k)z_k(1-z_l)z_l w_k w_l$$

We can thus write the whole FIM as the expectation of a Kronecker product:

$$F = \mathbb{E}[(y-p)^2 \cdot S \otimes xx^T]$$

where the matrix S can be written as

$$S = ww^T \odot zz^T \odot (1-z)(1-z)^T$$

Given a task described by N training samples $\{(x_e, y_e)\}$, the FIM can be estimated empirically as

$$F = \frac{1}{N}\sum_e p_e(1-p_e)\cdot S_e \otimes x_e x_e^T$$
$$S_e = ww^T \odot z_e z_e^T \odot (1-z_e)(1-z_e)^T$$

where we take expectation over y w.r.t. the predictive distribution $y \sim p_w(y|x)$.

Example toy task embedding As noted in the main text, the FIM depends on the domain embedding, the particular task and its complexity. We illustrate these properties of the task embedding using an "toy" task space illustrated in Figure 4. We generate 64 binary classification tasks by clustering a uniform grid of points in the XY plane into $k \in [3, 16]$ clusters using k-means and assigning a half of them to one category. We consider two different feature extractors, which play the role of "probe network". One is a collection of polynomial functions of degree $d = 3$, the second is 10 random linear features of the form $\max(0, ax + by + c)$ where a and b are sampled uniformly between $[-1/2, 1/2]$ and c between $[-1, 1]$.

6. Robust Fisher Computation

Consider again the loss function (parametrized with the covariance matrix Σ instead of the precision matrix Λ for convenience of notation):

$$L(\hat{w}; \Sigma) = \mathbb{E}_{w\sim\mathcal{N}(\hat{w},\Sigma)}[H_{p_w,\hat{p}}(y|x)] + \beta\, KL(\mathcal{N}(\hat{w}, \Sigma)\,\|\,\mathcal{N}(0, \sigma^2 I)).$$

We will make use of the fact that the Fisher Information matrix is a positive semidefinite approximation of the Hessian H of the cross-entropy loss, and coincide with it in local minima [2]. Expanding to the second order around $\hat{w}$, we have:

$$\begin{aligned}
L(\hat{w}; \Sigma) =& \mathbb{E}_{w\sim\mathcal{N}(\hat{w},\Sigma)}[H_{p_{\hat{w}},\hat{p}}(y|x) + \nabla_w H_{p_{\hat{w}},\hat{p}}(y|x)(w-\hat{w}) + \frac{1}{2}(w-\hat{w})^T H(w-\hat{w})] + \beta\, KL(\mathcal{N}(\hat{w}, \Sigma)\,\|\,\mathcal{N}(0, \sigma^2 I))\\
=& H_{p_{\hat{w}},\hat{p}}(y|x) + \frac{1}{2}\operatorname{tr}(\Sigma H) + \beta\, KL(\mathcal{N}(\hat{w}, \Sigma)\,\|\,\mathcal{N}(0, \sigma^2 I))\\
=& H_{p_{\hat{w}},\hat{p}}(y|x) + \frac{1}{2}\operatorname{tr}(\Sigma H) + \frac{\beta}{2}\Big[\frac{\hat{w}^2}{\sigma^2} + \frac{1}{\sigma^2}\operatorname{tr}\Sigma + k\log\sigma^2 - \log(|\Sigma|) - k\Big]
\end{aligned}$$

where in the last line used the known expression for the KL divergence of two Gaussian. Taking the derivative with respect to Σ and setting it to zero, we obtain that the expression loss is minimized when $\Sigma^{-1} = \frac{2}{\beta}\Big(H + \frac{\beta}{2\sigma^2}I\Big)$, or, rewritten in term of the precision matrices, when

$$\Lambda = \frac{2}{\beta}\Big(H + \frac{\beta\lambda^2}{2}I\Big),$$

where we have introduced the precision matrices $\Lambda = \Sigma^{-1}$ and $\lambda^2 I = 1/\sigma^2 I$.

We can then obtain an estimate of the Hessian H of the cross-entropy loss at the point $\hat{w}$, and hence of the FIM, by minimizing the loss $L(\hat{w}, \Lambda)$ with respect to Λ. This is a more robust approximation than the standard definition, as it depends on the loss in a whole neighborhood of $\hat{w}$ of size $\propto \Lambda$, rather than from the derivatives of the loss at a point. To further make the estimation more robust, and to reduce the number of parameters, we constrain Λ to be diagonal, and constrain weights w_{ij} belonging to the same filter to have the same precision Λ_{ij}. Optimization of this loss can be performed easily using Stochastic Gradient Variational Bayes, and in particular using the local reparametrization trick of [1].

The prior precision λ^2 should be picked according to the scale of the weights of each layer. In practice, since the weights of each layer have a different scale, we found it useful to select a different λ^2 for each layer, and train it together with Λ.

References

[1] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015. 6

[2] James Martens. New perspectives on the natural gradient method. *CoRR*, abs/1412.1193, 2014. 6

[3] Amir R Zamir, Alexander Sax, William Shen, Leonidas Guibas, Jitendra Malik, and Silvio Savarese. Taskonomy: Disentangling task transfer learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3712–3722, 2018. 1