TASK2VEC: Task Embedding for Meta-Learning

Alessandro Achille[1,2] Michael Lam[1] Rahul Tewari[1] Avinash Ravichandran[1]
Subhransu Maji[1,3] Charless Fowlkes[1,4] Stefano Soatto[1,2] Pietro Perona[1,5]

achille@cs.ucla.edu {michlam,tewarir,ravinash,smmaji,fowlkec,soattos,peronapp}@amazon.com

[1]AWS [2]University of California, Los Angeles [3]University of Massachusetts, Amherst [4]University of California, Irvine [5]Caltech

Abstract

We introduce a method to generate vectorial representations of visual classification tasks which can be used to reason about the nature of those tasks and their relations. Given a dataset with ground-truth labels and a loss function, we process images through a "probe network" and compute an embedding based on estimates of the Fisher information matrix associated with the probe network parameters. This provides a fixed-dimensional embedding of the task that is independent of details such as the number of classes and requires no understanding of the class label semantics. We demonstrate that this embedding is capable of predicting task similarities that match our intuition about semantic and taxonomic relations between different visual tasks. We demonstrate the practical value of this framework for the meta-task of selecting a pre-trained feature extractor for a novel task. We present a simple meta-learning framework for learning a metric on embeddings that is capable of predicting which feature extractors will perform well on which task without actually fine-tuning the model. Selecting a feature extractor with task embedding yields performance close to the best available feature extractor, with substantially less computational effort than exhaustively training and evaluating all available models.

1. Introduction

The success of deep learning in computer vision is due in part to the fact that models trained for one task can often be used on related tasks. Yet, no general framework exists to describe and reason about relations between tasks. We introduce the TASK2VEC embedding, a technique to represent tasks as elements of a vector space based on the Fisher Information Matrix. The norm of the embedding correlates with the complexity of the task, while the distance between embeddings captures semantic similarities between tasks (Fig. 1). When other natural distances are available, such as the taxonomic distance in biological classification, we find they correlate well with the embedding distance (Fig. 2). We also introduce an asymmetric distance on the embedding space that correlates with transferability between tasks.

Computation of the embedding leverages a duality between parameters (weights) and outputs (activations) in a deep neural network (DNN). Just as the activations of a DNN trained on a complex visual recognition task are a rich representation of the input images, we show that the gradients of the weights relative to a task-specific loss are a rich representation of the task itself. Given a task defined by a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$ of labeled samples, we feed the data through a pre-trained reference convolutional neural network which we call a "*probe network*", and compute the diagonal Fisher Information Matrix (FIM) of the network filter parameters to capture the structure of the task (Sect. 3). Since the architecture and weights of the probe network are fixed, the FIM provides a fixed-dimensional representation of the task which is independent of, e.g., how many categories there are. We show that this embedding simultaneously encodes the "difficulty" of the task, statistics of the input domain, and which features extracted by the probe network are discriminative for the task (Sect. 3.2).

Our task embedding can be used to reason about the space of tasks and solve meta-tasks. As a motivating example, we study the problem of selecting the best pre-trained feature extractor to solve a new task (Sect. 4). This is particularly valuable when there is insufficient data to train or fine-tune a generic model, and transfer of knowledge is essential. To select an appropriate pre-trained model, we design a joint embedding of models and tasks in the same vector space, which we call MODEL2VEC. We formulate this as a meta-learning problem where the objective is to find an embedding such that that models whose embeddings are close to a task exhibit good performance on that task.

We present large-scale experiments on a library of 1,460 fine-grained classification tasks constructed from existing computer vision datasets. These tasks vary in the level of difficulty and have orders of magnitude variation in training set size, mimicking the heavy-tailed distribution of real-world tasks. Our experiments show that using TASK2VEC to select an expert from a collection of 156 feature extractors outperforms the standard practice of fine-tuning a generic model trained on ImageNet. We find that the selected expert is close to optimal while being orders of magnitude faster than brute-force selection.

2. Background and Related Work

What metric should be used on the space of tasks? This depends critically on the meta-task we are considering. For the purpose of model selection there are several natural metrics that may be considered.

Domain distance. Tasks distinguished by their domain can be understood simply in terms of image statistics. Due to the bias of different datasets, sometimes a benchmark task can be identified just by looking at a few images [34]. The question of determining what summary statistics are useful (analogous to our choice of probe network) has also been considered. For example [9] train an autoencoder that learns to extract fixed dimensional summary statistics that can reproduce many different datasets accurately. However, for general vision tasks, the domain is insufficient (detecting pedestrians and reading license plates are different tasks that share the same domain of street scene images).

Taxonomic distance. Some collections of tasks come with a natural notion of semantic similarity based on a taxonomic hierarchy. We may say that classifying dog breeds is closer to classification of cats than it is to classification of plant species. When each task is specified by a set of categories in a subtree of the hierarchy, the tasks inherit a distance from the taxonomy. In this setting, we can define

$$D_{\text{tax}}(t_a, t_b) = \min_{i \in S_a, j \in S_b} d(i, j),$$

where S_a, S_b are the sets of categories in task t_a, t_b and $d(i, j)$ is an ultrametric or graph distance on the taxonomy. However such taxonomies are not available for all tasks and visual similarity need not be correlated with semantic similarity.

Transfer distance. Another notion that abstracts away the details of domains and labels is the transfer distance between task. For a specified DNN architecture, this is defined as the gain in performance from pre-training a model on task t_a and then fine-tuning for task t_b relative to the performance of simply training on task t_b using a fixed initialization (random or generic pre-trained)[1]. We write

$$D_{\text{ft}}(t_a \to t_b) = \frac{\mathbb{E}[\ell_{a\to b}] - \mathbb{E}[\ell_b]}{\mathbb{E}[\ell_b]},$$

where the expectations are taken over all training runs with the selected architecture, training procedure and network initialization, ℓ_b is the final test error obtained by training on task b from the chosen initialization, and $\ell_{a\to b}$ is the error obtained instead when starting from a solution to task a and then fine-tuning (with the selected procedure) on task t_b.

This notion of task transfer is the focus of Taskonomy [39], which explores knowledge transfer in a curated collection of 26 visual tasks, ranging from classification to 3D reconstruction, defined on a common domain. They compute transfer distances between pairs of tasks and use the results to compute a directed hierarchy. Adding novel tasks to the hierarchy requires computing the transfer distance to all other tasks in the collection.

In contrast, we show it is possible to directly produce a task embedding in constant time without computing pairwise distances. This makes it feasible to experiment on a much larger library of 1,460 classification tasks in multiple domains. The large task collection and cheap cost of embedding allows us to tackle new meta-learning problems.

Fisher Kernels and Fisher Information. Our work takes inspiration from Jaakkola and Hausler [16]. They propose the "Fisher Kernel", which uses the gradients of a generative model score function as a representation of similarity between data items

$$K(x^{(1)}, x^{(2)}) = \nabla_\theta \log P(x^{(1)}|\theta)^T F^{-1} \nabla_\theta \log P(x^{(2)}|\theta).$$

Here, $P(x|\theta)$ is a parameterized generative model and F is the Fisher information matrix. This provides a way to utilize generative models in the context of discriminative learning. Variants of the Fisher kernel have found wide use as a representation of images [28, 29] and other structured data such as protein molecules [17] and text [30]. Since the generative model can be learned on unlabeled data, several works have investigated the use of Fisher kernel for unsupervised learning [14, 31]. [35] learns a metric on the Fisher kernel representation similar to our metric learning approach. Our approach differs in that we use the FIM as a representation of a whole dataset (task) rather than using model gradients as representations of individual data items.

Fisher Information for CNNs. Our approach to task embedding makes use of the Fisher Information matrix of a neural network as a characterization of the task. Use of Fisher information for neural networks was popularized by Amari [6] who advocated optimization using natural gradient descent which leverages the fact that the FIM is a parameterization-independent metric on statistical models. Recent work has focused on approximators of FIM appropriate in this setting (see e.g., [12, 10, 25]). FIM has also been proposed for various regularization schemes [5, 8, 22, 27], to analyze learning dynamics of deep networks [4], and to overcome catastrophic forgetting [19].

[1] We improperly call this a distance but it is not necessarily symmetric or even positive.

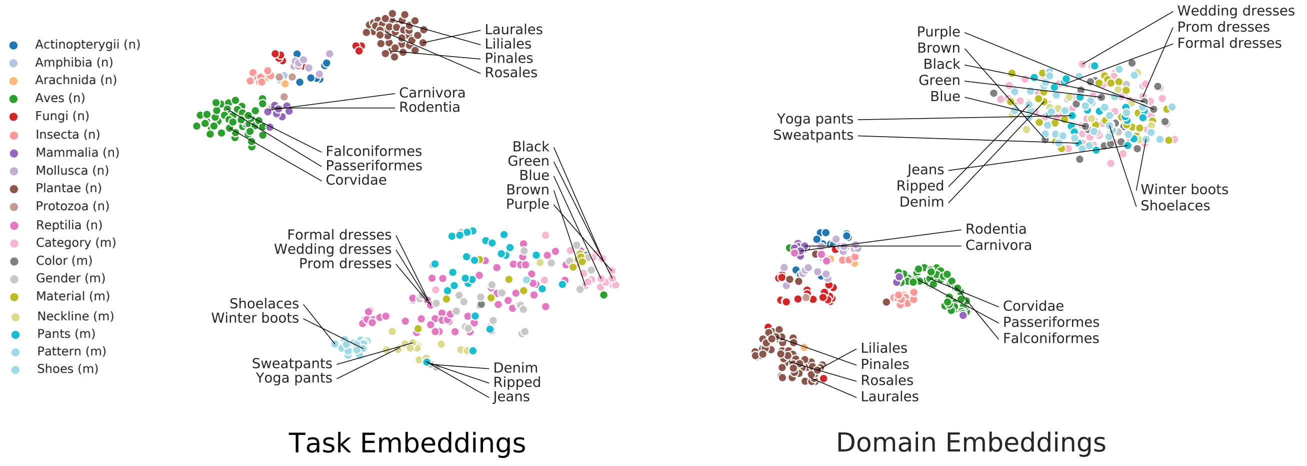


Figure 1: **Task embedding across a large library of tasks** (best seen magnified). **(Left)** T-SNE visualization of the embedding of tasks extracted from the iNaturalist, CUB-200, iMaterialist datasets. Colors indicate ground-truth grouping of tasks based on taxonomic or semantic types. Notice that the bird classification tasks extracted from CUB-200 embed near the bird classification task from iNaturalist, even though the original datasets are different. iMaterialist is well separated from iNaturalist, as it entails very different tasks (clothing attributes). Notice that some tasks of similar type (such as color attributes) cluster together but attributes of different task types may also mix when the underlying visual semantics are correlated. For example, the tasks of jeans (clothing type), denim (material) and ripped (style) recognition are close in the task embedding. **(Right)** T-SNE visualization of the domain embeddings (using mean feature activations) for the same tasks. Domain embedding can distinguish iNaturalist tasks from iMaterialist tasks due to differences in the two problem domains. However, the fashion attribute tasks on iMaterialist all share the same domain and only differ in their labels. In this case, the domain embeddings collapse to a region without recovering any sensible structure.

Meta Learning and Model Selection. Meta-learning has a long history with much recent work dedicated to meta-tasks such as neural architecture search, hyper-parameter estimation and robust few-shot learning. The meta-learning problem of model selection seeks to choose from a library of classifiers to solve a new task [33, 2, 20]. Unlike our approach, these previous techniques usually address the question via land-marking or active testing, in which a few different models are evaluated and performance of the remainder estimated by extension. This can be viewed as a problem of completing unknown entries in a matrix defined by performance of each model on each task. In computer vision, this idea has been explored for selecting a detector for a new category out of a large library of detectors [26, 40, 38].

3. Task Embeddings via Fisher Information

Given an observed input image x and an unknown task variable y (*e.g.*, a label), a deep network is a family of functions $p_w(y|x)$ parametrized by weights w, trained to approximate the posterior $p(y|x)$ by minimizing the (possibly regularized) cross entropy loss $H_{p_w,\hat{p}}(y|x) = \mathbb{E}_{x,y\sim\hat{p}}[-\log p_w(y|x)]$, where $\hat{p}$ is the empirical distribution defined by the training set $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^N$. It is useful, especially in transfer learning, to think of the network as composed of two parts: a feature extractor which computes some representation $z = \phi_w(x)$ of the input data, and a "head," or classifier, which predicts the distribution $p(y|z)$ given the representation z.

Not all network weights are equally important in predicting the task variable. The importance, or "informative content", of a weight for the task can be quantified by considering a perturbation $w' = w + \delta w$ of the weights, and measuring the average Kullbach-Leibler (KL) divergence between the original output distribution $p_w(y|x)$ and the perturbed one $p_{w'}(y|x)$. To second-order approximation, this is

$$\mathbb{E}_{x\sim\hat{p}}\, KL(p_{w'}(y|x) \,\|\, p_w(y|x)) = \delta w^T \cdot F \cdot \delta w + o(\delta w^2),$$

where F is the Fisher information matrix (FIM):

$$F = \mathbb{E}_{x,y\sim\hat{p}(x)p_w(y|x)}\left[\nabla_w \log p_w(y|x) \nabla_w \log p_w(y|x)^T\right].$$

that is, the covariance of the scores (gradients of the log-likelihood) with respect to the model parameters.

The FIM is a Riemannian metric on the space of probability distributions [7], and provides a measure of the information a particular parameter (weight or feature) contains about the joint distribution $p_w(x, y) = p_w(y|x)\hat{p}(x)$. If the classification performance for a given task does not depend strongly on a parameter, the corresponding entries in the

FIM will be small. The FIM is also related to the (Kolmogorov) complexity of a task, a property that can be used to define a computable metric of the learning distance between tasks [3]. Finally, the FIM can be interpreted as an easy-to-compute positive semidefinite upper-bound to the Hessian of the cross-entropy loss and coincides with it at local minima [24]. In particular, "flat minima" correspond to weights that have, on average, low Fisher information [5, 13].

3.1. TASK2VEC embedding using a probe network

While the network activations capture the information in the input image which are needed to infer the image label, the FIM indicates the set of feature maps which are more informative for solving the current task. Following this intuition, we use the FIM to represent the task itself. However, the FIMs computed on different networks are not directly comparable. To address this, we use a single "probe" network pre-trained on ImageNet as a feature extractor and re-train only the classifier layer on any given task, which usually can be done efficiently. After training is complete, we compute the FIM for the feature extractor parameters.

Since the full FIM is unmanageably large for rich probe networks based on CNNs, we make two additional approximations. First, we only consider the diagonal entries, which implicitly assumes that correlations between different filters in the probe network are not important. Second, since the weights in each filter are usually not independent, we average the Fisher Information for all weights in the same filter. The resulting representation thus has fixed size, equal to the number of filters in the probe network. We call this embedding method TASK2VEC.

Robust Fisher computation. Since the FIM is a local quantity, it is affected by the local geometry of the training loss landscape, which is highly irregular in many deep network architectures [21], and may be too noisy when trained with few samples. To avoid this problem, instead of direct computation, we use a more robust estimator that leverages connections to variational inference. Assume we perturb the weights $\hat{w}$ of the network with Gaussian noise $\mathcal{N}(0, \Lambda)$ with precision matrix Λ, and we want to find the optimal Λ which yields a good expected error, while remaining close to an isotropic prior $\mathcal{N}(\hat{w}, \lambda^2 I)$. That is, we want to find Λ that minimizes:

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

where H is the cross-entropy loss and β controls the weight of the prior. Notice that for $\beta = 1$ this reduces to the Evidence Lower-Bound (ELBO) commonly used in variational inference. Approximating to the second order, the optimal value of Λ satisfies (see Supplementary Material):

$$\frac{\beta}{2N}\Lambda = F + \frac{\beta \lambda^2}{2N} I.$$

Therefore, $\frac{\beta}{2N}\Lambda \sim F + o(1)$ can be considered as an estimator of the FIM F, biased towards the prior $\lambda^2 I$ in the low-data regime instead of being degenerate. In case the task is trivial (the loss is constant or there are too few samples) the embedding will coincide with the prior $\lambda^2 I$, which we will refer to as the *trivial embedding*. This estimator has the advantage of being easy to compute by directly minimizing the loss $L(\hat{w}; \Sigma)$ through Stochastic Gradient Variational Bayes [18], while being less sensitive to irregularities of the loss landscape than direct computation, since the value of the loss depends on the cross-entropy in a neighborhood of $\hat{w}$ of size Λ^{-1}. As mentioned previously, we estimate one parameter per filter, rather than per weight, which in practice means that we constrain $\Lambda_{ii} = \Lambda_{jj}$ whenever w_i and w_j belongs to the same filter. In this case, optimization of $L(\hat{w}; \Lambda)$ can be done efficiently using the local reparametrization trick of [18].

3.2. Properties of the TASK2VEC embedding

The task embedding we just defined has a number of useful properties. For illustrative purposes, consider a two-layer sigmoidal network for which an analytic expression can be derived (see Supplementary Materials). The FIM of the feature extractor parameters can be written using the Kronecker product as

$$F = \mathbb{E}_{x, y \sim \hat{p}(x) p_w(y|x)}[(y - p)^2 \cdot S \otimes x x^T]$$

where $p = p_w(y = 1|x)$ and the matrix $S = w w^T \odot z z^T \odot (1 - z)(1 - z)^T$ is an element-wise product of classifier weights w and first layer feature activations z. It is informative to compare this expression to an embedding based only on the dataset domain statistics, such as the (non-centered) covariance $C_0 = \mathbb{E}\left[x x^T\right]$ of the input data or the covariance $C_1 = \mathbb{E}\left[z z^T\right]$ of the feature activations. One could take such statistics as a representative *domain embedding* since they only depend on the marginal distribution $p(x)$ in contrast to the FIM *task embedding*, which depends on the joint distribution $p(x, y)$. These simple expressions highlight some important (and more general) properties of the Fisher embedding we now describe.

Invariance to the label space. The task embedding does not directly depend on the task labels but only on the predicted distribution $p_w(y|x)$ of the trained model. Information about the ground-truth labels y is encoded in the weights w which are a sufficient statistic of the task [5]. In particular, the task embedding is invariant to permutations of the labels y and has fixed dimension (number of filters of the feature extractor) regardless of the output space (e.g., k-way classification with varying k).

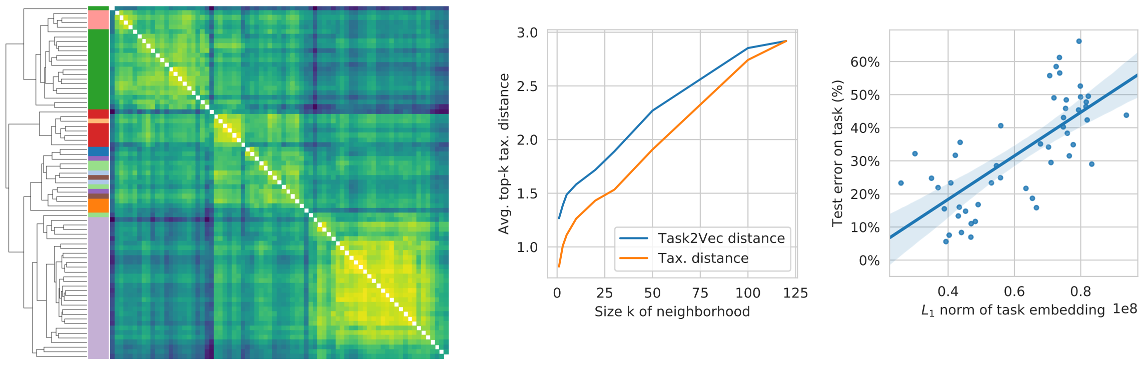


Figure 2: **Distance between species classification tasks.** **(Left)** Task similarity matrix ordered by hierarchical clustering. Note that the dendrogram produced by the task similarity matches the taxonomic clusters (indicated by color bar). **(Center)** For tasks extracted from iNaturalist and CUB, we compare the cosine distance between tasks to their taxonomical distance. As the size of the task embedding neighborhood increases (measured by number of tasks in the neighborhood), we plot the average taxonomical distance of tasks from the neighborhood center. While the task distance does not perfectly match the taxonomical distance (whose curve is shown in orange), it shows a good correlation. Difference are both due to the fact that taxonomically close species may need very different features to be classified, creating a mismatch between the two notions of distance, and because for some tasks in iNaturalist too few samples are provided to compute a good embedding. **(Right)** Correlation between L_1 norm of the task embedding (distance from origin) and test error obtained on the task.

Encoding task difficulty. As we can see from the expressions above, if the model is very confident in its predictions, $\mathbb{E}[(y-p)^2]$ goes to zero. Hence, the norm of the task embedding $\|F\|_\star$ scales with the difficulty of the task for a given feature extractor ϕ. Fig. 2 (Right) shows that even for more complex models trained on real data, the FIM norm correlates with test performance.

Encoding task domain. Data points x that are classified with high confidence, i.e., p is close to 0 or 1, will have a lower contribution to the task embedding than points near the decision boundary since $p(1-p)$ is maximized at $p = 1/2$. Compare this to the covariance matrix of the data, C_0, to which all data points contribute equally. Instead, in TASK2VEC the information on the domain is based on data near the decision boundary (task-weighted domain embedding).

Encoding useful features for the task. The FIM depends on the curvature of the loss function with the diagonal entries capturing the sensitivity of the loss to model parameters. Specifically, in the two-layer model, one can see that if a given feature is uncorrelated with y, the corresponding blocks of F are zero. In contrast, a domain embedding based on feature activations of the probe network (e.g., C_1) only reflects which features vary over the dataset without indication of whether they are relevant to the task.

3.3. Symmetric and asymmetric TASK2VEC metrics

By construction, the Fisher embedding on which TASK2VEC is based captures fundamental information about the structure of the task. We may therefore expect that the distance between two embeddings correlates positively with natural metrics on the space of tasks. However, there are two problems in using the Euclidean distance between embeddings: the parameters of the network have different scales, and the norm of the embedding is affected by complexity of the task and the number of samples used to compute the embedding.

Symmetric TASK2VEC distance. To make the distance computation robust, we propose to use the cosine distance between normalized embeddings:

$$d_{\text{sym}}(F_a, F_b) = d_{\cos}\Big(\frac{F_a}{F_a + F_b}, \frac{F_b}{F_a + F_b}\Big),$$

where $d_{\cos}$ is the cosine distance, F_a and F_b are the two task embeddings (*i.e.*, the diagonal of the Fisher Information computed on the same probe network), and the division is element-wise. This is a symmetric distance which we expect to capture semantic similarity between two tasks. For example, we show in Fig. 2 that it correlates well with the taxonomical distance between species on iNaturalist.

On the other hand, precisely for this reason, this distance is ill-suited for tasks such as model selection, where the (intrinsically asymmetric) transfer distance is more relevant.

Asymmetric TASK2VEC distance. In a first approximation, that does not consider either the model or the training procedure used, positive transfer between two tasks depends both on the similarity between two tasks and on the complexity of the first. Indeed, pre-training on a general but complex task such as ImageNet often yields a better result than fine-tuning from a close dataset of lower complexity. In our case, complexity can be measured as the distance from the trivial embedding. This suggests the following asymmetric score, again improperly called a "distance" despite being asymmetric and possibly negative:

$$d_{\text{asym}}(t_a \to t_b) = d_{\text{sym}}(t_a, t_b) - \alpha d_{\text{sym}}(t_a, t_0),$$

where t_0 is the trivial embedding, and α is an hyperparameter. This has the effect of bringing more complex models closer. The hyper-parameter α can be selected based on the meta-task. In our experiments, we found that the best value of α ($\alpha = 0.15$ when using a ResNet-34 pretrained on ImageNet as the probe network) is robust to the choice of meta-tasks.

3.4. MODEL2VEC: Co-embedding models and tasks

By construction, the TASK2VEC distance ignores details of the model and only relies on the task. If we know what task a model was trained on, we can represent the model by the embedding of that task. However, in general we may not have such information (*e.g.*, black-box models or hand-constructed feature extractors). We may also have multiple models trained on the same task with different performance characteristics. To model the joint interaction between task and model (*i.e.*, architecture and training algorithm), we aim to learn a joint embedding of the two.

We consider for concreteness the problem of learning a joint embedding for model selection. In order to embed models in the task space so that those near a task are likely to perform well on that task, we formulate the following meta-learning problem: Given k models, their MODEL2VEC embedding are the vectors $m_i = F_i + b_i$, where F_i is the task embedding of the task used to train model m_i (if available, else we set it to zero), and b_i is a learned "model bias" that perturbs the task embedding to account for particularities of the model. We learn b_i by optimizing a k-way cross entropy loss to predict the best model given the task distance (see Supplementary Material):

$$\mathcal{L} = \mathbb{E}[-\log p(m \mid d_{\text{asym}}(t, m_0), \dots, d_{\text{asym}}(t, m_k))].$$

After training, given a novel query task t, we can then predict the best model for it as the $\arg\min_i d_{\text{asym}}(t, m_i)$, that is, the model m_i embedded closest to the query task.

4. Experiments

We test TASK2VEC on a large collection of tasks and models with a range of task similarities[2]. Our experiments aim to test both qualitative properties of the embedding and its performance on meta-learning tasks. We use an off-the-shelf ResNet-34 pretrained on ImageNet as our probe network, which we found to give the best overall performance (see Sect. 4.2). The collection of tasks is generated starting from the following four main datasets. **iNaturalist** [36]: Each task corresponds to species classification in a given taxonomical order. For instance, the *"Rodentia task"* is to classify species of rodents. Each task is defined on a separate subset of the images in the original dataset; that is, the domains of the tasks are disjoint. **CUB-200** [37]: We use the same procedure as iNaturalist to create tasks. All tasks are classifications inside orders of birds (the *aves* taxonomical class) and have generally much fewer training samples than corresponding tasks in iNaturalist. **iMaterialist** [1] and **DeepFashion** [23]: Each image in these datasets is associated with several binary attributes (*e.g.*, style attributes) and categorical attributes (*e.g.*, color, type of dress, material). We binarize the categorical attributes and consider each attribute as a separate task. Notice that in this case, all tasks share the same domain and are naturally correlated.

In total, our collection of tasks has 1460 tasks (207 iNaturalist, 25 CUB, 228 iMaterialist, 1000 DeepFashion). While a few tasks have many training examples (*e.g.*, hundred thousands), most have just hundreds or thousands of samples. This simulates the heavy-tail distribution of data in real-world applications.

For model selection experiments, we assemble a library of "expert" feature extractors. These are ResNet-34 models pre-trained on ImageNet and then fine-tuned on a specific task or collection of related tasks (see Supplementary Materials for details). We also consider a "generic" expert pre-trained on ImageNet without any fine-tuning. Finally, for each combination of expert feature extractor and task, we trained a linear classifier on top of the expert in order to solve the selected task using the expert.

In total, we trained 4,100 classifiers, 156 feature extractors and 1,460 embeddings. The total effort to generate the final results was about 1,300 GPU hours.

Meta-tasks. In Sect. 4.2, for a given task we aim to predict, using TASK2VEC , which expert feature extractor will yield the best classification performance. In particular, we formulate two model selection meta-tasks: **iNat + CUB** and **Mixed**. The first consists of 50 tasks and experts from iNaturalist and CUB, and aims to test fine-grained expert selection in a restricted domain. The second contains a mix of 26 curated experts and 50 random tasks extracted from all datasets, and aims to test model selection between different domains and tasks (see Supplementary Material for details).

4.1. Task Embedding Results

Task Embedding qualitatively reflects taxonomic distance for iNaturalist. For tasks extracted from the iNaturalist dataset (classification of species), the taxonomical

[2]Here we discuss only classification tasks, the supplement describes additional experiments on pixel-labeling and regression tasks

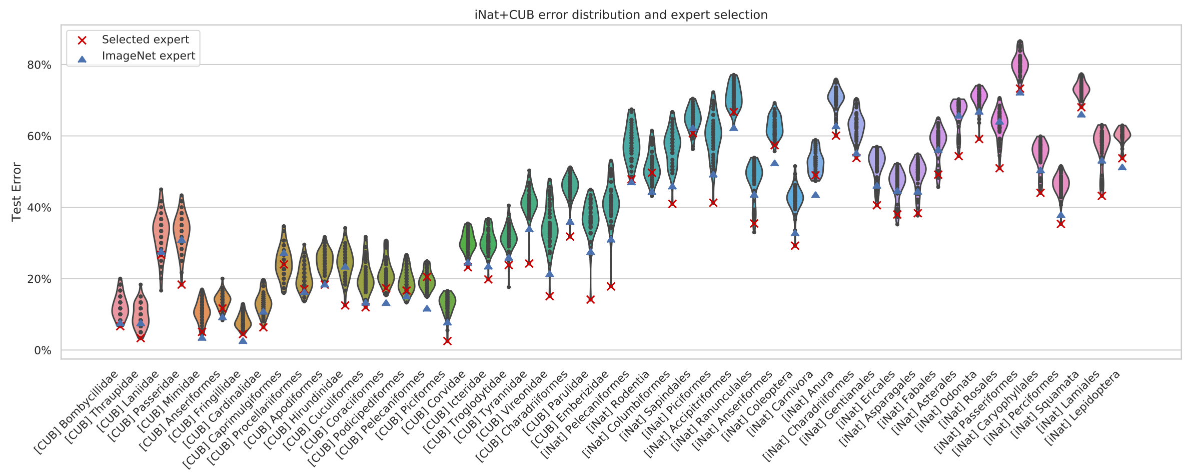


Figure 3: **TASK2VEC often selects the best available experts.** Violin plot of the test error distribution (shaded) on tasks from the CUB-200 dataset (columns) obtained by training a linear classifier over several expert feature extractors (points). Most specialized feature extractors perform similarly on a given task, and similar or worse than a generic feature extractor pre-trained on ImageNet (blue triangles). However, in some cases a carefully chosen expert, trained on a related task, can greatly outperform all others (long lower whiskers). The model selection algorithm based on TASK2VEC can predict an expert to use for the task (red cross, lower is better) and often recommends the optimal, or near optimal, feature extractor without performing an expensive brute-force training and evaluation over all available experts. Columns are ordered by norm of the task embedding vector. Tasks with lower embedding norm have lower error and more "complex" task (task with higher embedding norm) tend to benefit more from a specialized expert.

distance between orders provides a natural metric of the semantic similarity between tasks. In Fig. 2, we compare the symmetric TASK2VEC distance with the taxonomical distance, showing strong agreement.

Task embedding for iMaterialist. In Fig. 1, we show a t-SNE visualization of the embedding for iMaterialist and iNaturalist tasks. Task embedding yields interpretable results: Tasks that are correlated in the dataset, such as binary classes corresponding to the same categorical attribute, may end up far away from each other and close to other tasks that are semantically more similar (*e.g.*, the *jeans* category task is close to the *ripped* attribute and the *denim* material). In the visualization, this non-trivial grouping is reflected in the mixture of colors of semantically related nearby tasks.

We also compare the TASK2VEC embedding with a domain embedding baseline, which only exploits the input distribution $p(x)$ rather than the task distribution $p(x, y)$. While some tasks are highly correlated with their domain (*e.g.*, tasks from iNaturalist), other tasks differ only on the labels (*e.g.*, all the attribute tasks of iMaterialist, which share the same clothes domain). Accordingly, the domain embedding recovers similar clusters on iNaturalist. However, on iMaterialst, domain embedding collapses all tasks to a single uninformative cluster (not a single point due to slight noise in embedding computation).

Task Embedding encodes task difficulty. The scatter-plot in Fig. 1 compares the norm of embedding vectors vs. performance of the best expert (or task specific model for cases where pre-train and test tasks coincide). As suggested by the analysis for the two-layer model, the norm of the task embedding also correlates with the complexity of the task for real tasks and model architectures.

4.2. Model Selection

Given a task, our aim is to select an expert feature extractor that maximizes the classification performance on that task. We propose two strategies: (1) embed the task and select the feature extractor trained on the most similar task, and (2) jointly embed the models and tasks, and select a model using the learned metric (see Section 3.4). Notice that (1) does not use knowledge of the model performance on various tasks, which makes it more widely applicable but requires we know what task a model was trained for and may ignore the fact that models trained on slightly different tasks may still provide an overall better feature extractor (for example by over-fitting less to the task they were trained on).

In Table 1 we compare the overall results of the various proposed metrics on the model selection meta-tasks. On both the iNat+CUB and Mixed meta-tasks, the Asymmetric TASK2VEC model selection is close to the ground-truth optimal, and significantly improves over both chance, and over using an generic ImageNet expert. Notice that our method has $O(1)$ complexity, while searching over a collection of

Meta-task	Optimal	Chance	ImageNet	TASK2VEC	Asymmetric TASK2VEC	MODEL2VEC
iNat + CUB	31.24	+59.52%	+30.18%	+42.54%	+9.97%	**+6.81%**
Mixed	22.90	+112.49%	+75.73%	+40.30%	+29.23%	**+27.81%**

Table 1: **Model selection performance.** Average error obtained on two meta-learning datasets by exhaustive search to select the optimal expert, and relative error increase when using cheaper model selection methods. A general model pre-trained on ImageNet performs better than picking an expert at random (chance). However, picking an expert using the Asymmetric TASK2VEC distance achieves substantially better performance and can be further improved by meta-learning (MODEL2VEC).

N experts is $O(N)$.

Error distribution. In Fig. 3, we show in detail the error distribution of the experts on multiple tasks. It is interesting to observe that the classification error obtained using most experts clusters around some mean value, and little improvement is observed over using a generic expert. On the other hand, for each task there often exist a few experts that can obtain significantly better performance on the task than a generic expert. This confirms the importance of having access to a large collection of experts when solving a new task, especially if few training data are available. TASK2VEC provides an efficient way to index this collection and identify an appropriate expert for a new task without brute-force search.

Dependence on task dataset size. Selecting pre-trained experts is especially important when the novel query task has relatively few training samples. In Fig. 4, we show how the performance of TASK2VEC model selection varies as a function of dataset size. Even for tasks with few samples, TASK2VEC selection performs nearly as well as using the optimal (ground-truth) expert. If in addition to training a classifier, we fine-tune the selected expert, error decreases further. At all dataset sizes, we see that the expert selected by TASK2VEC significantly outperforms the standard baseline of using a generic ImageNet pre-trained model.

Choice of probe network. In Table 2, we show that DenseNet [15] and ResNet architectures [11] perform significantly better when used as probe networks to compute the TASK2VEC embedding than a VGG [32] architecture.

5. Discussion

TASK2VEC is an efficient way to represent a task as a fixed dimensional vector with several appealing properties.

	Chance	VGG-13	DenseNet-121	ResNet-13
Top-10	+13.95%	+4.82%	+0.30%	**+0.00%**
All	+59.52%	+38.03%	+10.63%	**+9.97%**

Table 2: **Choice of probe network.** Mean increase in error relative to optimal (ground-truth) expert selection on the iNat+CUB meta-task for different choices of the probe-network. We also report the performance on a subset of 10 tasks with the most training samples to show the effect of data size in choosing a probe architecture.

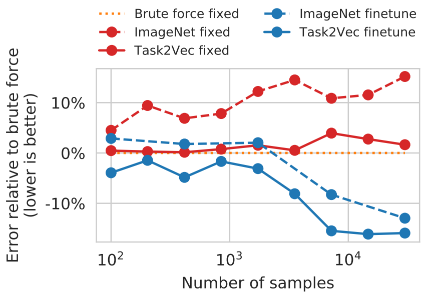


Figure 4: **TASK2VEC improves outperforms baselines at different dataset sizes:** Performance of model selection on a subset of 4 tasks as a function of the number of samples available to train relative to optimal model selection (dashed orange). Training a classifier on the fixed feature extractor selected by TASK2VEC (solid red) is always better than using a generic ImageNet feature extractor (dashed red). The same holds when fine-tuning the feature extractor (blue curves). In the low-data regime, the fixed pre-trained experts selected by TASK2VEC even outperform costly fine-tuning of a generic ImageNet feature extractor (dashed blue).

The embedding norm correlates with the task difficulty, and the cosine distance between embeddings is predictive of both natural semantic distances between tasks (e.g., the taxonomic distance for species classification) and the fine-tuning distance for transfer learning. Having a representation of tasks paves the way for a wide variety of meta-learning tasks. In this work, we focused on selection of an expert feature extractor in order to solve a new task, and showed that using TASK2VEC to select an expert from a collection can improve test performance over the de facto baseline of using an ImageNet pre-trained model while adding only a small overhead to the training process.

We demonstrate that TASK2VEC is scalable to thousands of tasks, allowing us to reconstruct a topology on the task space, and to test meta-learning solutions. The current experiments highlight the usefulness of our methods. Even so, our collection does not capture the full complexity and variety of tasks that one may encounter in real-world situations. Future work should further test effectiveness, robustness, and limitations of the embedding on larger and more diverse collections.

References

[1] iMaterialist Fashion Challenge at FGVC5 workshop, CVPR. `https://www.kaggle.com/c/imaterialist-challenge-fashion-2018`. 6

[2] Salisu Mamman Abdulrahman, Pavel Brazdil, Jan N van Rijn, and Joaquin Vanschoren. Speeding up algorithm selection using average ranking and active testing by introducing runtime. *Machine learning*, 107(1):79–108, 2018. 3

[3] Alessandro Achille, Glen Mbeng, Giovanni Paolini, and Stefano Soatto. The dynamic distance between learning tasks: From Kolmogorov complexity to transfer learning via quantum physics and the information bottleneck of the weights of deep networks. *Proc. of the NIPS Workshop on Integration of Deep Learning Theories (ArXiv: 1810.02440)*, October 2018. 4

[4] Alessandro Achille, Matteo Rovere, and Stefano Soatto. Critical learning periods in deep neural networks. *Proc. of the Intl. Conf. on Learning Representations (ICLR). ArXiv:1711.08856*, 2019. 2

[5] Alessandro Achille and Stefano Soatto. Emergence of invariance and disentanglement in deep representations. *Journal of Machine Learning Research (ArXiv 1706.01350)*, 19(50):1–34, 2018. 2, 4

[6] Shun-Ichi Amari. Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276, 1998. 2

[7] Shun-Ichi Amari and Hiroshi Nagaoka. Methods of information geometry, volume 191 of translations of mathematical monographs. *American Mathematical Society*, 13, 2000. 3

[8] Sanjeev Arora, Rong Ge, Behnam Neyshabur, and Yi Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018. 2

[9] Harrison Edwards and Amos Storkey. Towards a neural statistician. *arXiv preprint arXiv:1606.02185*, 2016. 2

[10] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *arXiv preprint arXiv:1703.03400*, 2017. 2

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016. 8

[12] Tom Heskes. On natural learning and pruning in multilayered perceptrons. *Neural Computation*, 12(4):881–901, 2000. 2

[13] Sepp Hochreiter and Jürgen Schmidhuber. Flat minima. *Neural Computation*, 9(1):1–42, 1997. 4

[14] Alex D Holub, Max Welling, and Pietro Perona. Combining generative models and fisher kernels for object recognition. In *IEEE International Conference on Computer Vision*, volume 1, pages 136–143. IEEE, 2005. 2

[15] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017. 8

[16] Tommi Jaakkola and David Haussler. Exploiting generative models in discriminative classifiers. In *Advances in Neural Information Processing Systems*, pages 487–493, 1999. 2

[17] Tommi S Jaakkola, Mark Diekhans, and David Haussler. Using the Fisher kernel method to detect remote protein homologies. In *ISMB*, volume 99, pages 149–158, 1999. 2

[18] 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. 4

[19] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, page 201611835, 2017. 2

[20] Rui Leite, Pavel Brazdil, and Joaquin Vanschoren. Selecting classification algorithms with active testing. In *International workshop on machine learning and data mining in pattern recognition*, pages 117–131. Springer, 2012. 3

[21] Hao Li, Zheng Xu, Gavin Taylor, and Tom Goldstein. Visualizing the loss landscape of neural nets. *arXiv preprint arXiv:1712.09913*, 2017. 4

[22] Tengyuan Liang, Tomaso Poggio, Alexander Rakhlin, and James Stokes. Fisher-Rao metric, geometry, and complexity of neural networks. *arXiv preprint arXiv:1711.01530*, 2017. 2

[23] Ziwei Liu, Ping Luo, Shi Qiu, Xiaogang Wang, and Xiaoou Tang. DeepFashion: Powering robust clothes recognition and retrieval with rich annotations. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1096–1104, 2016. 6

[24] James Martens. New perspectives on the natural gradient method. *CoRR*, abs/1412.1193, 2014. 4

[25] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature. In *International Conference on Machine Learning (ICML)*, pages 2408–2417, 2015. 2

[26] Pyry Matikainen, Rahul Sukthankar, and Martial Hebert. Model recommendation for action recognition. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 2256–2263. IEEE, 2012. 3

[27] Youssef Mroueh and Tom Sercu. Fisher GAN. In *Advances in Neural Information Processing Systems*, pages 2513–2523, 2017. 2

[28] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. Improving the fisher kernel for large-scale image classification. In *European Conference on Computer Vision*, pages 143–156. Springer, 2010. 2

[29] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. Image classification with the Fisher vector: Theory and practice. *International Journal of Computer Vision*, 105(3):222–245, 2013. 2

[30] Craig Saunders, Alexei Vinokourov, and John S Shawe-taylor. String kernels, Fisher kernels and finite state automata. In *Advances in Neural Information Processing Systems*, pages 649–656, 2003. 2

[31] Matthias Seeger. Learning with labeled and unlabeled data. Technical Report EPFL-REPORT-161327, Institute for Adaptive and Neural Computation, University of Edinburgh, 2000. 2

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 8

[33] Michael R Smith, Logan Mitchell, Christophe Giraud-Carrier, and Tony Martinez. Recommending learning algorithms and their associated hyperparameters. *arXiv preprint arXiv:1407.1890*, 2014. 3

[34] Antonio Torralba and Alexei A Efros. Unbiased look at dataset bias. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1521–1528. IEEE, 2011. 2

[35] Laurens Van Der Maaten. Learning Discriminative Fisher Kernels. In *International Conference on Machine Learning*, volume 11, pages 217–224, 2011. 2

[36] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The iNaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, 2018. 6

[37] C. Wah, S. Branson, P. Welinder, P. Perona, and S. Belongie. The Caltech-UCSD Birds-200-2011 Dataset. Technical Report CNS-TR-2011-001, California Institute of Technology, 2011. 6

[38] Yu-Xiong Wang and Martial Hebert. Model recommendation: Generating object detectors from few samples. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1619–1628, 2015. 3

[39] 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. 2

[40] Peng Zhang, Jiuling Wang, Ali Farhadi, Martial Hebert, and Devi Parikh. Predicting failures of vision systems. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3566–3573, 2014. 3

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).

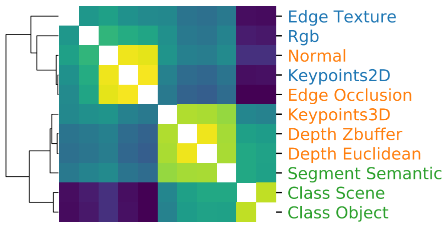


Figure 1: TASK2VEC distance between Taskonomy tasks. As in [3], similar tasks cluster together in the TASK2VEC embedding: (blue) 2D tasks, (orange) 3D tasks, (green) semantic tasks.

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` $\rightarrow$ `global avg. pool` $\rightarrow$ `linear` $\rightarrow$ `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` $\rightarrow$ `conv3x3 256` $\rightarrow$ `upsample 32` $\rightarrow$ `conv3x3 128` $\rightarrow$ `upsample 64` $\rightarrow$ `conv3x3 128` $\rightarrow$ `upsample 128` $\rightarrow$ `conv3x3 64` $\rightarrow$ `upsample 256x256` $\rightarrow$ `conv3x3 32` $\rightarrow$ `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

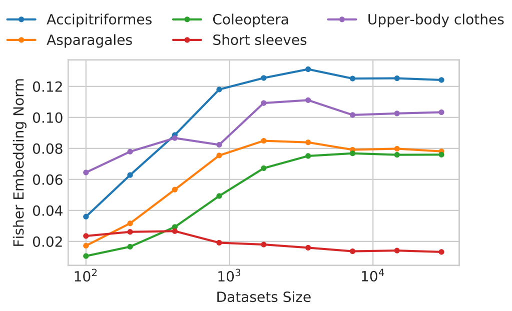
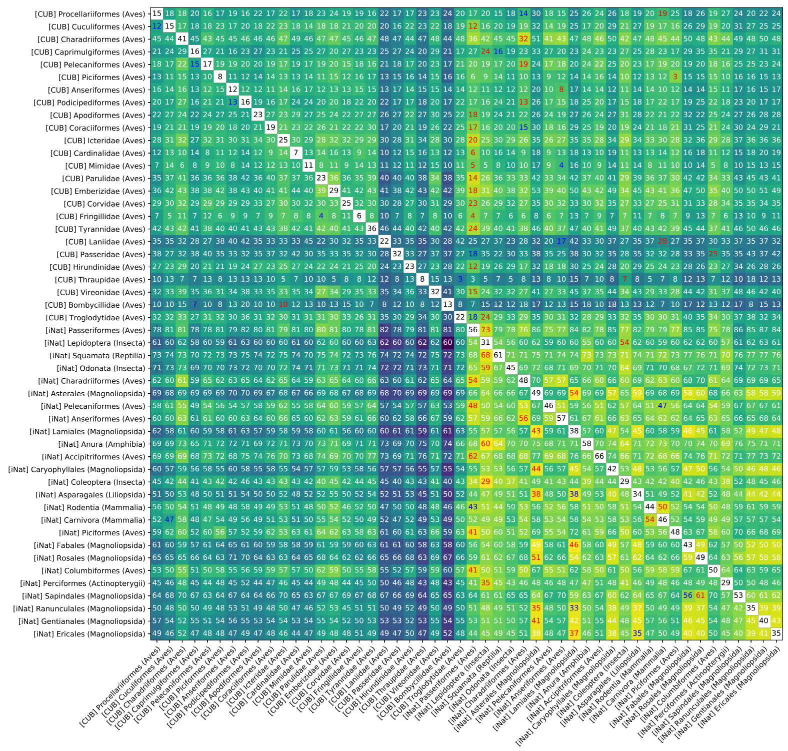


Figure 2: Norm of the task embedding as the number of samples in the dataset varies. Larger datasets generally have larger norms, hence TASK2VEC enables distinguishing tasks that only differ by the number of shots (training samples/class).

2. Complete error matrices for the meta-tasks



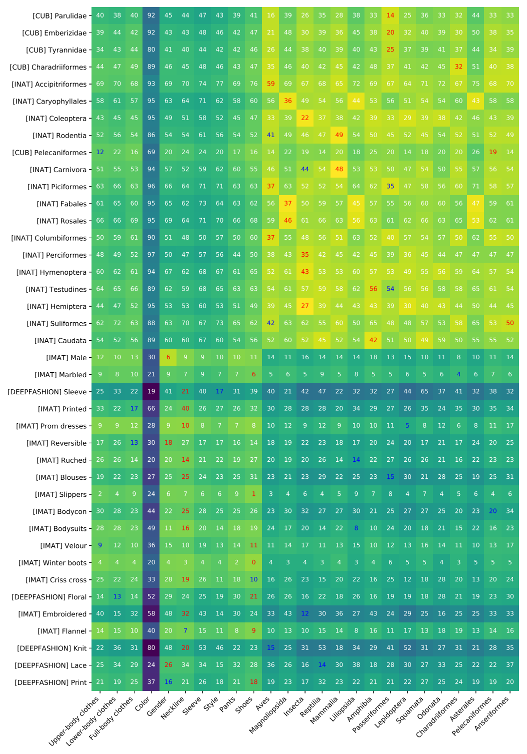


Figure 3: **Meta-tasks ground-truth error matrices and TASK2VEC selections.** (Best viewed magnified). **(Left)** Error matrix for the CUB+iNat meta-task. The numbers in each cell is the test error obtained by training a classifier on a given combination of task (rows) and expert (columns). The background 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:

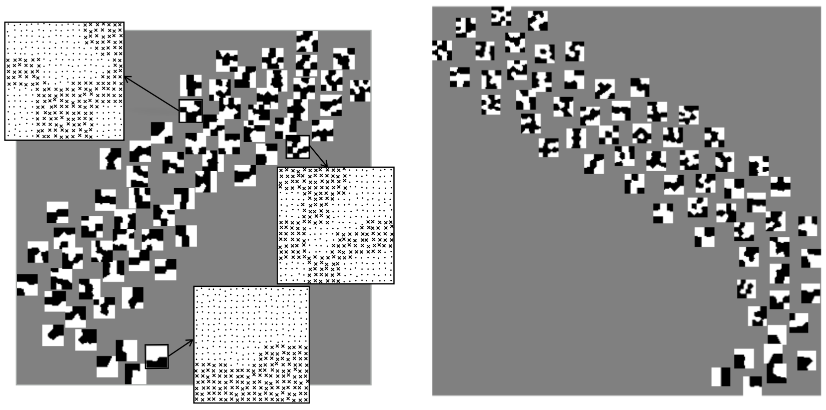
$$\frac{\partial\ell}{\partial w} = \frac{1}{N}\sum_i (y_i - p_i)\phi(x_i),$$

and the empirical estimate of the Fisher information matrix is:

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

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