Task representation for transfer learning

Mahdi Nikdan, Christoph Lampert

April 14, 2023

Abstract

In this paper, we investigate a transfer learning problem with the setting that, given a set of training tasks and a test task, we want to transfer knowledge from one of the training tasks to the test task. Here, we focus on how to choose which one of the training tasks to transfer from. We take a meta learning approach of presenting methods to encode tasks into representation vectors, and then leveraging the similarity of these vectors to decide whether transfer learning between two tasks will be beneficial. We introduce two Naive and Transfer methods. The Naive method averages the features extracted by a pretrained ImageNet feature extractor to represent a task, and we show that this representation performs better than the baseline in clustering the similar tasks together. On the other hand, our Transfer method leverages a transfer loss, which will ensure that the representation similarity correlate with transfer performance between two tasks. We show that the Transfer method has a performance comparable with the baseline.

Keywords: Transfer Learning, Meta Learning

1 Introduction

Transfer learning aims to boost the performance of a learning algorithm on a new task, using the knowledge extracted from a similar task. However, it is not easy to measure how similar two tasks are. In this report, we focus on the problem of formulating and measuring the similarity between tasks, and then leveraging this similarity metric to predict whether transfer learning between two tasks will be beneficial.

To tackle this problem, we employ the approach of representing a task with a vector, and define the distance of tasks as the distance between their corresponding representations. More specifically, we look for a function $\tau(.)$ that, given a task T, $\tau(T)$ is the representation vector of T. Our exact transfer learning problem can be summarized as follows: given a number of training tasks $\{T_1, T_2, ..., T_n\}$ and a test task T^*, we aim to find one of the training tasks T_i that is the most similar to T^*. We do so by encoding the tasks into vectors using the τ function, and choosing T_i which has the closest vector to that of T^* (closeness is measured by a vector similarity metric, e.g., cosine similarity). We then perform transfer learning from T_i to T^*. We assume there are limited training samples available for the test task, since this is the case where transfer learning is particularly useful.

As the most related work, Task2Vec [1] defines a similar problem and takes a similar approach to tackle it. They exploit the diagonal elements of the Fisher Information Matrix (FIM) to extract representations for a task. However, they require training a classifier in test time in order to extract the representation. In addition, there is no clear intuition why this representation should be suitable to use for predicting how well the knowledge can be transferred between two tasks.

In this paper, we present two Naive and Transfer methods. The Naive method exploits a pretrained feature extractor on ImageNet and only averages the features extracted from the input data to encode the task. The Transfer method, on the other hand, optimizes a transfer loss which measures how well the knowledge can be transferred between two tasks with high representation similarity. In contrast

to Task2Vec, the loss in our Transfer method ensures that the representations are extracted in a way that their similarities correlate with the transfer learning performance. Also, in test time, none of our methods require training for extracting the representations. We empirically show that our methods perform better than Task2Vec in clustering similar tasks together. Additionally, for different methods, we measure the correlation between representation similarity and transfer learning performance between two tasks. While our Transfer method outperforms our Naive method (as expected), both of them surprisingly perform worse that Task2Vec.

2 Method

2.1 Problem Formulation

We start by defining a classification task T as a dataset $D = \{X, y\}$, where X and y are the inputs and labels, respectively. Given a set of training tasks $S = \{T_1, T_2, ..., T_n\}$, our goal is to find a function $\tau(.)$ that encodes a task T to a d dimensional vector, i.e., $\tau(T) \in \mathbb{R}^d$.

In test time, given a new task T^*, we choose a task T_k from S that

$$k = \underset{i \in \{1,2,...,n\}}{\arg\min}\ \mathcal{D}(T^*, T_i)$$

where $\mathcal{D}(T^*, T_i) = d(\tau(T^*), \tau(T_i))$ and d is a distance metric over vectors. Then we use transfer learning to exploit the knowledge learned from T_k for improving the performance on T^*.

2.2 Naive Method

Here we present a simple way of encoding a task, which requires no training even on the training tasks. Suppose M is a model pretrained on ImageNet. M consists of two parts: a feature extractor ϕ and a linear classifier c, so that $\forall x : M(x) = c(\phi(x))$.

For a task $T = \{X, y\}$, we define:

$$\tau(T) = \frac{1}{|X|} \sum_{x \in X} \phi(x)$$

as the representation of T. Since no training is required for extracting these representations, it can also easily be done at test time on a new task T^*, even in the case where we only have access to few samples of T^*. It is worth mentioning that for simplicity, we are not taking the labels of T (i.e., y) into account, and we assume that only considering X will suffice for extracting representations. Figure 1 demonstrates an overview of this method.

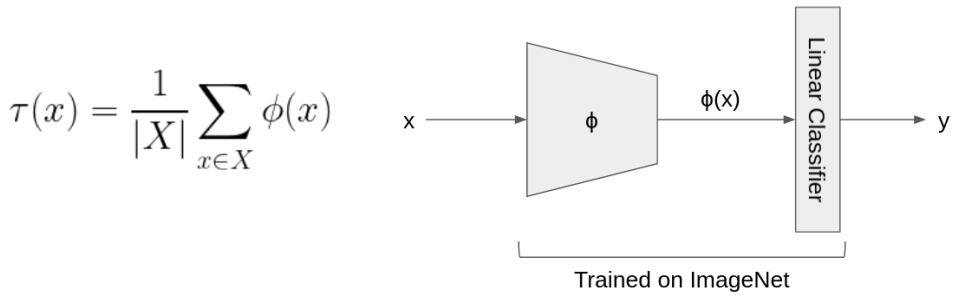

Figure 1: The Naive method. The model is trained on ImageNet, and the task representation is calculated as the average of features extracted from inputs.

2.3 Transfer Method

This method utilizes a similar pretrained network on ImageNet, but here the feature extractor ϕ is divided into two parts ϕ' and θ so that $\phi(x) = \theta(\phi'(x))$. In summary, the idea is to train one ϕ' module and use it as the task representation extractor, and train different θ modules for different tasks in a way that they are suitable for transfer learning using the representations extracted by ϕ'. See Figure 2 for an illustration. In what follows, we explain the role of each module in more details:

- ϕ': It is shared among all tasks and trained jointly on all training tasks. We use it similar to ϕ in the Naive method to extract the task representation:

$$\tau(T) = \frac{1}{|X|} \sum_{x \in X} \phi'(x)$$

  where $T = \{X, y\}$. Note that, once ϕ' is trained, there is no training required in test time for extracting the representation.

- θ: We train one θ for each training task, i.e., we train $\theta_1, \theta_2, ..., \theta_n$. This module is used for transfer learning, meaning that given a new task, we do not train a new θ and only use one of $\theta_1, \theta_2, ..., \theta_n$. In particular, given a test task T^*, the goal is that if we let

$$k = \underset{i \in \{1,2,...,n\}}{\arg\min}\ d(\tau(T^*), \tau(T_i))$$

  with respect to a distance metric d, then we use the parameters of θ_k for T^* and only train a linear classifier c^*.

- c: This module is a linear classifier, which we train for each (train or test) task separately. Note that c is a linear classifier and can be trained efficiently on a test task, even when we only have access to limited number of samples. The reason that we do not perform transfer learning on the classifiers is that often tasks have different number of classes, which makes it challenging to transfer classifiers.

$$\tau(x) = \frac{1}{|X|} \sum_{x \in X} \phi'(x)$$

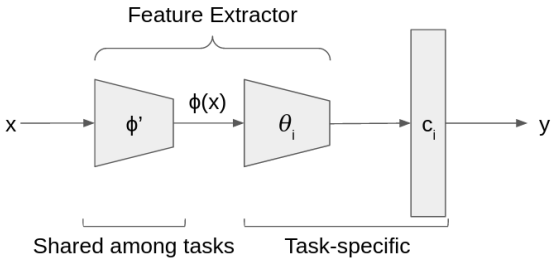

Figure 2: The Transfer method. The ϕ' module is the feature extractor shared among all tasks, the θ_i module is task-specific and is used for transfer learning, and the c_i module is the task-specific linear classifier.

Now we explain the training process. Recall that we have a set of training tasks $S = \{T_1, T_2, ..., T_n\}$, and we intend to train one shared ϕ' and n different θ modules, along with n different linear classifiers. We do the training based on two objectives:

1. The main classification loss, which on average measures how well the model works on a task T_i, when the corresponding θ_i and c_i (trained for the same task) are employed. In particular:

$$L^{\text{main}} = \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}(T_i; \phi', \theta_i, c_i)$$

   where $\mathcal{L}$ is the cross entropy loss.

2. The transfer loss, which on average measures how well our model works on a task T_i if we transfer learning from other training tasks. Particularly, the transfer loss is defined as follows:

$$L^{\text{transfer}} = \frac{1}{n}\sum_{i=1}^{n}\mathcal{L}(T_i;\phi',\theta_k,c_i)$$

where $k = \underset{j\in\{1,2,...,n\},j\neq i}{\arg\min} d(\tau(T_i),\tau(T_j))$ with respect to a distance metric d. In summary, for a task T_i, we find the closest task T_k to it (based on the representations extracted by ϕ'), and then use θ_k instead of θ_i.

Finally, we train $\phi', \theta_1, \theta_2, ..., \theta_n, c_1, c_2, ..., c_n$ end to end on the loss $L = L^{\text{main}} + \lambda L^{\text{transfer}}$ where λ is a hyper-parameter.

3 Experiments

3.1 Dataset

Following the setting in [1], we use the iNaturalist dataset [3], which contains the 437,513 training, 24,426 validation, and 149,394 test images of 8,142 species, along with their classification labels. In addition, they record to which taxonomical ranks each species belongs. Similar to [1], we create a task for classifying the species under each *order* (which is one of the taxonomical ranks). We noticed that around 20% of tasks only have one class, which makes them trivial. Unlike [1], we removed these tasks and ended up with 208 task. We selected 20% of tasks and kept them for test time.

3.2 Implementation Details

A torchvision ResNet18 model is employed in all experiments. In the Transfer method, we set θ to be the two layers '*layer4*' and '*avgpool*', c to be the '*fc*' layer, and ϕ' to be the rest of layers. We also put $\lambda = 1$, and use $1 - \text{cosine similarity}$ as the distance metric d. At training time, we employ an Adam optimizer with learning rate 0.001 for 1000 epochs, with each epoch containing 100 batches, and each batch consisting of 16 samples of each training task. At test time, given a test task T^*, we train a classifier c^* for 30 epochs using the same optimizer.

3.3 Clustering Results

Here we will try to evaluate how well the extracted representations are, without performing any transfer learning. To do this, we follow the approach in [1] to compare the similarities given by our methods and the taxonomical similarity between diffrenet tasks, e.g., we generally expect the representation of a plant to be closer to another plant than an animal. Figure 3 shows a comparison between Task2Vec [1] and our Naive and Transfer methods. Each row shows the similarity of a task with other tasks (yellow means very similar). As we can see, our methods seem to have more clearly clustered the similar tasks together. For instance, *Accipitriformes* and *Odonata* are far away from their similar species in Task2Vec, which is not the case for our methods.

To provide a quantitative comparison, we notice that all the species are coming from 5 *kingdoms* (which is another taxonomical rank). Hence, we use the extracted representations to cluster species into 5 classes, and then measure how well the clusters match the ground truth kingdoms. For this, we exploit two well-known clustering metrics, Normalized Mutual Information (NMI) and Average Purity. NMI is defined as follows:

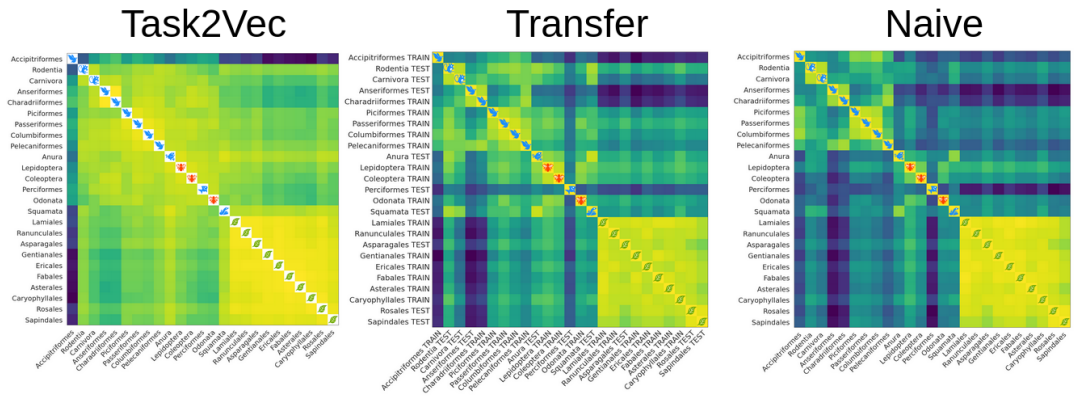

Figure 3: The clustering results. The heatmaps compare our methods with Task2Vec [1] to see how well they can cluster the similar tasks together. In the Transfer heatmap, it is also indicated whether each tasks is a training or a test task.

$$\text{NMI}(Y, C) = \frac{2 \times I(Y, C)}{H(Y) + H(C)}$$

where Y and C represent the ground truth classes and the clusters, respectively. I is the mutual information and H is the entropy. Also, for each class c in clustering C, the purity is defined as:

$$\text{Purity(Y,c)} = \frac{1}{|c|} \max_j |Y_j \cap c|$$

where Y_j indicates the elements in j'th ground truth class. Taking the mean of this Purity metric over different classes in C results in the Average Purity. Table 1 compares Task2Vec [1] with our Naive method. As we can see, our Naive method outperforms Task2Vec with a large margin.

Table 1: Clustering metrics

	NMI	Purity
Task2Vec	0.21	0.25
Naive	0.55	0.69

3.4 Transfer Results

As the main goal of this paper, here we exploit the extracted task representations to perform transfer learning. In the setting of the Transfer method, given a test task T^*, we choose 10 random training tasks $T_{r_1}, T_{r_2}, ..., T_{r_{10}}$. Then for each $i \in \{1, 2, ..., 10\}$, we use the module θ_{r_i} and train a linear classifier c^* for the task T^*. We then compute the correlation between the final accuracy of the model on T^* and the similarity $\tau(T^*, T_{r_i})$. This correlation shows whether we can rely on the calculated similarities to decide which training task to transfer from. The results are presented in Table 2. This tables shows that our Transfer method outperforms the Naive method with a large margin, which indicates the effectiveness of our training approach in the Transfer method. However, Task2Vec [1] surprisingly performs even better that our Transfer method.

Table 2: Correlation between representations similarity and transfer accuracy

	Task2Vec	Naive	Transfer
Correlation	0.47	0.25	0.4

4 Conclusion and Future Work

In this paper, we investigated a transfer learning problem, where we need to choose a task to transfer from. We presented two methods to represent a task with a vector, and then used the representations to measure similarity between tasks. Our Naive method only exploits a pretrained feature extractor to extract task representations, while our Transfer method uses a more complex loss function to optimize for our transfer learning setting. We compared our methods with the recent Task2Vec [1] paper. We used the extracted representations to cluster the species of the dataset into 5 classes and compared them with the 5 ground truth kingdoms of species. Our results show that the Naive method significantly outperforms Task2Vec. In addition, for each method we measured the correlation between the representation similarity of two tasks and the transfer accuracy between them and showed that our Transfer method significantly outperforms our Natve method. However, Task2Vec surprisingly works better than both of them.

Both our methods only use the input data (X) to represent a task, while the output labels (y) can play an essential rule in extracting effective representations. One way of incorporating y into the representations is to borrow ideas from Neural Processes [2] and Deep Sets [4]. More specifically, one can leverage the permutation invariance in the dataset samples to first encode the dataset (X and y) into a vector, and then use this vector for performing classifications by a meta model. This vector can then be used as a representation of the dataset/task. We will leave this idea for future work.

References

[1] Alessandro Achille, Michael Lam, Rahul Tewari, Avinash Ravichandran, Subhransu Maji, Charless C Fowlkes, Stefano Soatto, and Pietro Perona. Task2vec: Task embedding for meta-learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6430–6439, 2019.

[2] Marta Garnelo, Jonathan Schwarz, Dan Rosenbaum, Fabio Viola, Danilo J Rezende, SM Eslami, and Yee Whye Teh. Neural processes. *arXiv preprint arXiv:1807.01622*, 2018.

[3] 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*, pages 8769–8778, 2018.

[4] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. Deep sets. *Advances in neural information processing systems*, 30, 2017.