CS 502 - The Self-Optimal-Transport Layer for Few-Shot Classification on Biomedical Datasets

Nicolas Wagner Rémi Delacourt

Abstract

The scope is the study and evaluation of few-shot learning algorithms for applications in biomedicine. Many few-shot learning methods focus on using non-parametric learners to capture key differences between different classes. This report focuses on the evaluation of such a few-shot learning method, proposed by the paper *The Self-Optimal-Transport Feature Transform* (Shalam & Korman, 2022).

1. Introduction

We study the paper *The Self-Optimal-Transport Feature Transform* (Shalam & Korman, 2022), which proposes a new method applicable in the few-shot learning setting, which they claim to be efficient in general. The goal here is to implement the method to perform few-shot classification on two biomedical datasets, and to evaluate its performance in this context, comparing it with reference few-shot learning methods.

The two datasets we focus on are the Tabula-Muris dataset, which is a cell type annotation task across tissues, and the SwissProt dataset, which is a gene function prediction task from the sequence information.

The codebase for our project can be found at github.com/TheImpressiveDonut/SelfOptimalTransport.

2. Description of the algorithm

2.1. High-level view

The method proposed is as follows. We assume that the model for few-shot classification is composed of a feature-extractor part and a task processing (classifier) part. The idea of the method is to insert a feature transform layer T between these two parts. One can see T as an extension of the feature-extractor part, as it embeds the features into a new embedding space. T is thus an independent layer which further embeds the features, and can be inserted in any general network. It is a parameterless layer, and its interpretation is described further below.

Optimal-transport (OT) problems enable to calculate distances between distributions or sets of features. T is a layer that solves an OT problem to enable capturing the relative similarities between the given samples in the batch. The method used to solve this OT problem is differentiable, enabling T to be compatible with any gradient-based training methods. The interpretation of what T computes is probabilistic: for a given input data i in the batch, its output is a vector, in which component j gives the probability of i being from the same class as the other input data j, relative to all the other inputs from the batch.
So the intuition of T is that it will enforce the feature extractor to extract relevant information that differentiates points from different classes, and solving an OT problem enforces to geometrically separate points of different classes and group points of same class.

2.2. Optimal-transport-based feature transform T

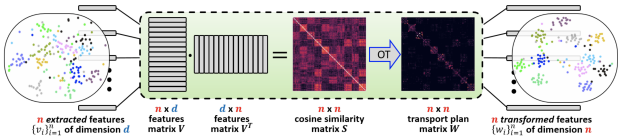

Figure 1. Computation pipeline of the optimal-transport-based feature transform T, from (Shalam & Korman, 2022)

T receives as input a batch of data of size N, which passed through the feature extractor of embedding dimension D (it thus receives a matrix $X \in \mathbb{R}^{N \times D}$). The output of T is a matrix $W \in \mathbb{R}^{N \times N}$. Computing W is done in two steps.

The first step is computing the squared-euclidean pairwise distance matrix D such that $D_{ij} = \|x_i - x_j\|^2$.

The second step is to solve an optimal transport (OT) problem by computing the OT plan matrix between itself and the all-ones vector (so that the resulting matrix represents probability distributions), under the cost matrix D (the costs are the distances). This part is related to the OT problem formulation, and it can be reformulated as a max-flow min-cost matching problem, which better demonstrates the intuition behind using this technique in our setting. We invite the reader to read part 3 of the paper, as describing this part here

goes out of the scope of this report.
Since the costs are the distances, the solution of the OT problem (which is a minimization problem) will assign high values to points close to each-other and low values to points far-away from each-other. T will thus force the feature-extractor to geometrically separate points of different classes and group points of the same class. This is how we get the probabilistic interpretation given previously.

T depends on the batch size: the output dimension of the layer is equal to the batch size. Since the classifier has a fixed input dimension, this constrains us to use the same batch-size throughout the experiment. And this also constrains to use the same batch size for the training, validation and testing. Another inconvenience is that during data loading, we have to drop the remaining data if it is not enough to form a batch.

2.3. Technical details to implement T

In the first step of T, we compute the pairwise distances between the points. For this we unit-normalize each x_i to obtain $V \in \mathbb{R}^{N \times D}$ such that $\|v_i\| = 1$ for each v_i a row in V. Now, $D_{ij} = 2(1 - \cos(v_i, v_j)) = 2(1 - v_i \cdot v_j^\top)$ and we obtain $D = 2(1 - V \cdot V^\top)$.

For the second step, we must mention a technical detail important for our implementation. We name the resulting solution matrix of the OT problem $\tilde{W}$. Since the entries of $\tilde{W}$ are relative probabilities, $\tilde{W}_{ij} = \tilde{W}_{ji}$ is the relative probability of i and j being from the same class. Since we already know that i and i are from the same class, some technical changes are made in the optimal-transport (OT) formulation so that $\tilde{W}_{ii} = 0$ and doesn't affect the other values. The change is that we force D_{ii} to be very large, so that the OT solution has no chance of matching i with itself. The distance value is set to $\alpha = 10^5$, which is sufficiently big, considering we unit-normalized the feature vectors in the first step.

To solve the OT problem, we implement the Sinkhorn method, as it is highly efficient (Cuturi, 2013). This requires to modify the OT formulation to insert a regularizer, which introduces a hyperparameter λ. On top of this, we actually use the Sinkhorn method in log-space for better numerical stability and apply an exponential to the resulting solution matrix. This final OT formulation and the log-Sinkhorn method to solve it is described in the paper from (Cuturi, 2013), and we implement our algorithm based on it. The algorithm is iterative and moves closer to the optimum at each iteration. We thus have an additional hyperparameter, which is the maximum number of iterations.
As a final step, we set 1 in the diagonal of $\tilde{W}$ to be coherent with our probabilistic interpretation. The resulting matrix is W.

As mentioned, T outputs vectors of dimensions equal to the batch size. We thus have to set the input dimension of the classifier to be equal to the batch size, and this value is different in each method we try.
In ProtoNet and MatchingNet, the train and val batch sizes are set to $n_{way} * (n_{support} + n_{query})$. In Baseline and Baseline++, the train batch size should be manually set to $n_{way} * (n_{support} + n_{query})$. Finally, the MAML method requires the support and query number to be equal, and since these values are different in the reference code, we modify the query number to be equal to the support number.

3. Experiments

3.1. Method

To evaluate the performance and impact of T, we insert T in the given reference methods of the project. The reference methods are Baseline and Baseline++ (Chen et al., 2020), MAML (Finn et al., 2017), ProtoNet (Snell et al., 2017), and MatchingNet (Vinyals et al., 2017).

We compare the performance of each method equipped with T, relative to its counterpart unequipped with T. We also compare the results of ProtoNet equipped with T with the results of the paper, which evaluates the performance of T on its insertion in ProtoNet. Since we work with different datasets as in the paper, this is a way to validate if T is a general improvement to many few-shot classification algorithms and use-cases, as they claim in the paper, or not.

In some algorithms such as Baseline++, we also pre-train the model without inserting T, and then fine-tune the model with T inserted, without freezing the backbone.

3.2. Hyperparameter tuning

We compared our implementation of log-Sinkhorn with an implementation from an external library (Flamary et al., 2021) which is more general. The results show similar performance in all the methods, which strengthens our trust in the performance of our own algorithm implementation. As our implementation is more specific to our use-case, we decided to go with ours.

We tried tuning multiple hyperparameters on the different methods using grid search, looking at the validation accuracy to decide. The hyperparameters related to our layer T are λ and the *number of iterations*. The hyperparameters related to the different methods we decided to tune are *number of epochs*, *number of episodes* and *learning rate*.

We made λ vary from 0.1 to 100 and *number of iterations* from 10 to 100. The results show that performance depends heavily on the regularizer λ, but the *number of iterations* has no notable impact. We concluded that 10 iterations are enough to find a correct OT solution for T to work

effectively. It is an advantage because we need to spend less time computing the layer, and has lower impact on long-running methods (for instance, MAML runs two times longer with T: $\sim$20min instead of $\sim$10min).

Using different *number of epochs* and *number of episodes* gave all the different methods using T no significant advantage over the methods without T. Contrary to that, the *learning rate* had clearly a bigger impact. Therefore, our analysis in the report focuses on tuning λ and the *learning rate*, when adding T to a method.

3.2.1. Tuning on ProtoNet

Hyperparameter tuning outcomes using the ProtoNet method varied depending on the dataset, with distinct settings: a *learning rate* of 0.001 and λ value of 10 for the Tabula-Muris dataset, and a *learning rate* of 0.01 combined with λ set to 5 for the SwissProt dataset. An illustration of how the λ hyperparameter impacts performance can be seen in Figure 2.

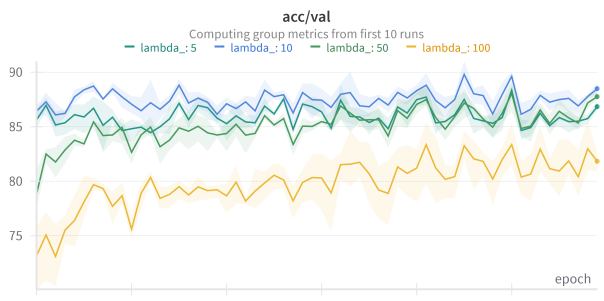

Figure 2. Validation accuracy when running ProtoNet with layer T, on Tabula-Muris, varying the hyperparameter λ

3.2.2. Tuning on other methods

We went on and proceeded in the same way for tuning the hyperparameters in the other methods. As will be described in more detail later, only ProtoNet, Baseline and Baseline++ are relevant methods to insert T into. For Baseline and Baseline++ we obtain a *learning rate* of 0.01 and $\lambda = 5$.

3.3. ProtoNet

We observed an small advantage of employing T specifically in ProtoNet, slightly enhancing performance exclusively in the Tabula-Muris dataset (Table 1). This dataset contains more samples, which may explain the effectiveness of T compared to SwissProt, as it enables better discrimination between diverse samples. Additionally, the differing batch sizes, 50 for Tabula-Muris and 100 for SwissProt, could have contributed to T facing challenges in effectively separating a larger volume of samples. These results do not showcase and confirm what was found in the original paper (Shalam & Korman, 2022), as we do not obtain a notable improvement on ProtoNet equipped with T.

Table 1. Final test accuracy when using or not the SOT layer, with the Tabula-Muris and SwissProt dataset

ProtoNet	SwissProt	Tabula-Muris
w/o. SOT	**60.3**$\pm$ **7.2**	89.9$\pm$ 8.8
w. SOT	59.3$\pm$ 8.0	**90.0**$\pm$ **8.4**

3.4. MatchingNet & MAML

Running MatchingNet and MAML methods on 5-way 5-shot classification with T inserted in their model gave outstanding performance. On both methods, the test accuracy goes higher than 99% during the first 10 epochs. We suspect that T exposes the regular labeling structure of the batch data when it is fed in the method, and the classifier of the model is able to very accurately learn this structure with the help of T. T is thus able to cheat and helps the classifier learn the patterns of the dataloader and of the two methods. In MatchingNet, MAML and ProtoNet, the partition of the data is regular: the labels are in increasing order, given by `np.repeat(range(self.n_way), self.n_query)`. This explains what we've just outlined, and the classifiers of MatchingNet and MAML are able to capture such structure. It is important to note that this doesn't affect ProtoNet, because in ProtoNet the support set is aggregated into prototypes, and the classification is done by selecting the nearest prototype, so the label structure in the batch doesn't affect the prediction.

This suggests that using T in this context might be incompatible with these two meta-learning algorithms.

3.5. Baseline & Baseline++

Results of Baseline and Baseline++ equipped with T are not as good as without T (see Table 2).
Something to note is that T gives probabilities of relative similarities between the given samples, and this is all we give as input to the classifier. One question arises: the classifier is able to distinguish between classes, but how does it determine the actual class labels ? If we do a permutation of the class labels, we could still get a similar output for T (e.g. if distance between classes are all the same). The loss enables to fix the permutation of the labels, and so the classifier is able to learn the correct classification labels for training samples, but how can it assign the correct label at test time ? It could correctly assign different labels for two samples of different classes, but it may incorrectly assign the label values. Label information might be encoded in the

output of T by indirect similarity encoding, but it may not be sufficiently informative to fix a permutation of the labels. We can think of giving more information to the classifier that would serve as reference point for the label assignments, thus fixing the label permutation.
This is what we do, by concatenating the embeddings from the feature extractor (before applying T) to the output vectors of T. That way, the model is able to extract information specific to the class it wants to predict, and at the same time it is able to better differentiate between the classes using information from T.

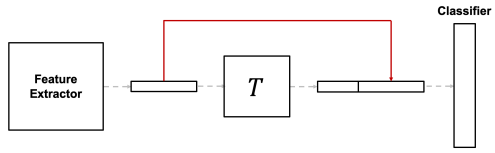

Figure 3. Illustration of the concatenation done for Baseline and Baseline++

With this concatenation we obtain better results, the model is able to use the information from T to prevent overfitting and improve on the validation and test accuracy.

Table 2. Final test accuracy when using Baseline++ on the two datasets, first without using T, then inserting T, then inserting T and concatenating

BASELINE++	SWISSPROT	TABULA-MURIS
W/O. SOT	57.5±7.7	83.1±10.4
W. SOT	50.5±7.9	77.4±9.3
W. SOT & CONCAT	**62.1±7.5**	**85.0±9.7**

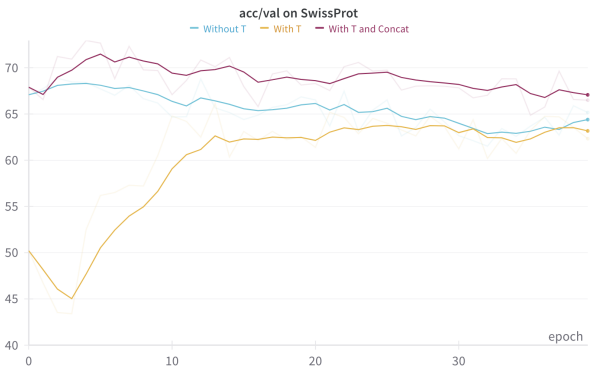

Figure 4. Validation accuracy when running Baseline++ on SwissProt, without SOT, with SOT, and with SOT and concatenation.

In Baseline++, we also pre-train the model without inserting T, and then fine-tune the model with T inserted, without freezing the backbone, and without using concatenation. This enables the model to first learn for the classification objective, and then during fine-tuning to learn to geometrically cluster the classes using T. The results are in Table 3

Table 3. Final test accuracy when pre-training Baseline++ without T, and then fine-tuning using T,on the SwissProt dataset

SWISSPROT	PRE-TRAIN	FINE-TUNE
BASELINE++	57.5±7.7	60.2±7.6

4. Conclusion & Future Work

Throughout our project, we thoroughly examined the Self-Optimal-Transport Feature Transform (T) in the few-shot classification context, particularly with biomedicine datasets. Our goal was to assess T's efficiency, as suggested in *The Self-Optimal-Transport Feature Transform* (Shalam & Korman, 2022), by incorporating it into various few-shot learning methods.

While exploring the effect of T on the learning process of different methods, promising outcomes emerged specifically in transfer-learning approaches (such as Baseline) and metric-learning approaches (such as ProtoNet). However, issues surfaced in other meta-learning methods, suggesting that one must be careful when incorporating T inside few-shot classification methods for it to be effective.

As future work, it would be interesting to investigate the effect of changing the batch size on the performance of few-shot classification methods equipped with T.

References

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., and Huang, J.-B. A closer look at few-shot classification, 2020.

Cuturi, M. Sinkhorn distances: Lightspeed computation of optimal transportation distances, 2013.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks, 2017.

Flamary, R., Courty, N., and Gramfort, A. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22 (78):1–8, 2021. URL http://jmlr.org/papers/v22/20-451.html.

Shalam, D. and Korman, S. The self-optimal-transport feature transform, 2022.

Snell, J., Swersky, K., and Zemel, R. S. Prototypical networks for few-shot learning, 2017.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. Matching networks for one shot learning, 2017.