Universiteit Leiden

Master Computer Science

Selecting Pre-trained Models for Transfer Learning with Data-centric Meta-features

Name:	Matt van den Nieuwenhuijzen
Student ID:	s2042096
Date:	[06/04/2024]
Specialisation:	Advanced computing and systems
1st supervisor:	Dr. J.N. van Rijn
2nd supervisors:	Dr. H. Gouk (University of Edinburgh) Dr. C. Doerr (Sorbonne Université, CNRS)

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

1 Introduction **1**

2 Background **3**
2.1 Supervised deep machine learning . . . 3
2.2 Deep metalearning . . . 4
2.3 Evaluation of deep metalearning systems . . . 5
2.4 Deep transfer learning and fine-tuning . . . 6
2.5 Residual neural networks (ResNets) . . . 7
2.6 Automated algorithm selection and meta-models . . . 7
2.7 Data . . . 8
2.8 Box and violin plots . . . 9

3 Related work **11**

4 Methods and general experimental setup **13**
4.1 Data preparation . . . 14
4.2 Pre-training models . . . 14
4.3 Fine-tuning . . . 14
4.4 Meta-model training . . . 16
4.5 Meta-model validation . . . 16
4.6 General experimental setup . . . 16

5 Transferability of datasets **19**
5.1 Setup and results . . . 19
5.2 Discussion . . . 20

6 Selecting pre-trained models **27**
6.1 Setup . . . 27
6.2 Results pixel-based features . . . 31
6.3 Results low-cost proxy . . . 31
6.4 Discussion . . . 31

7 Conclusion and future work **39**

1 Introduction

The success of deep learning is undeniable. Recent developments in commercially available AI show that if suitable training data is abundant, deep learning systems with impressive capabilities are possible. Especially in computer vision, better performance is expected as the amount of data increases [1]. Unfortunately, many domains and contexts exist where data and resources are scarce, and similar impressive machine-learning results cannot be achieved from scratch.

This problem has inspired computer scientists to search for ways to employ other, *auxiliary* data when data on the target task itself is sparse. Two active research fields that apply this idea from different perspectives are deep metalearning and deep transfer learning. The field of deep metalearning uses auxiliary data to create deep learning systems that can adapt quickly to different tasks [2]. These systems then require less data to adapt to the target tasks. In contrast, the field of deep transfer learning focuses on reusing previously acquired knowledge for a new task rather than creating systems that can quickly adapt to new tasks. In practice, this commonly involves using the auxiliary data to pre-train large deep learning models (or *foundation models*) and subsequently adapting these systems to new input data, e.g. by fine-tuning the parameters of a pre-trained model [3].

Although deep metalearning has brought forth many successful methods to train efficiently adaptable deep learning systems, its success has yet to go beyond experimental contexts. We believe a reason for this restraint is the unknown a priori meaning of *related data* in the context of machine learning. In other words: it is unclear what properties make a potential auxiliary dataset related to the target dataset so that the metalearning method is successful. This is especially a problem for real-world scenarios, where self-collected data is unlikely to resemble research datasets. Furthermore, we believe that research datasets might be more related than expected, which can make methods seem more successful than they are. For example, an N-way k-shot classification experiment within MiniImageNet might result in biased outcomes, as the examples and classes within this dataset could be more related than expected. This moves us to investigate how different datasets relate to each other in a deep learning context, and to challenge intuitions that are based on the conceptual relatedness found within (scientific) datasets and dataset domains.

A similar problem is found in the field of deep transfer learning. As more fully trained neural models are publicly released, model hubs or *model zoos* offer an ever-growing collection of resources that can be used to transfer to a different task [4, 5]. This can for example be achieved by fine-tuning pre-trained models, where only a small part of the model parameters is adjusted to new data, which is found to be an effective method for alleviating the training data requirement [6, 7]. However, a successful selection strategy to equip the optimal pre-trained model for a target task does not exist yet. Since deep learning methods often require computationally expensive operations before they can be evaluated (e.g. training via backpropagation), it is unrealistic to search exhaustively for the best match. This motivates us to develop a data-centric method to select the right auxiliary data for a deep metalearning or deep transfer learning problem.

Since the field of deep metalearning currently primarily focuses on computer vision problems, the scope of this work is also limited to this problem domain. For our experiments, we employ the recently published Meta-Album [8], which currently bundles 30 image datasets from 10 different domains (such as plants and remote-sensing). The diversity of Meta-Album allows us to test how meta-train and meta-test datasets relate to each other in a deep learning context for three different dataset relationships. **same-source** (where the meta-train and meta-test dataset originate from the same source), **same-domain** (where the meta-train and meta-test dataset belong to the same domain but do not have the same source), and **cross-domain** (where the meta-train and meta-test dataset belong to different domains). We use the different datasets of Meta-Album to record how well different pre-trained models can be fine-tuned to tasks from various datasets and domains. Then, we train meta-models to predict this performance for a given pre-trained model and task combination, using primarily basic pixel-based statistics of the datasets. Using this method, we try to answer the following research questions.

1. Can we expect pre-trained models to transfer well to tasks from the same source dataset via fine-tuning?
2. Do pre-trained models transfer better within their domain than cross-domain?
3. Given an image classification task and a collection of pre-trained classification models, can we select the best pre-trained model using pixel-based meta-features of the source datasets of the model and task?

We use the following structure to present our work. We start by providing all related background in Section 2. This includes the principles of supervised deep machine learning, deep metalearning, deep transfer learning, and automated algorithm selection. We continue with an overview of related, recent work in Section 3. Then, we explain our proposed methods and how we implemented them in Section 4. The setup and results of our experiments are divided over Section 5, corresponding to research questions 1 and 2, and Section 6, corresponding to research question 3. We conclude this work by reviewing the research questions in the context of the obtained results in Section 7 and provide suggestions for future work based on our observations.

2 Background

This section gives an overview and clarification of the concepts that form the foundation of this research. First, the principles and challenges of deep machine learning are discussed. Then, we introduce two research fields, deep metalearning and deep transfer learning, that aspire to solve some of these challenges. We continue with an outline of the field of automated algorithm selection. Finally, we provide some more practical background information on the data that we used for our experiments, and on specific methods and concepts that play a role in our work, like residual neural networks and box plots.

2.1 Supervised deep machine learning

Machine learning is a subfield of artificial intelligence that brings forth systems that can *learn* from data to solve problems without the need for explicit instructions on how to solve the task. In *supervised* machine learning, this is achieved by providing the system with examples of the problem and their corresponding solutions, and subsequently allowing the model to learn from these examples. In more formal terms, a supervised machine learning system is given the task of finding a function $f : X \to Y$ that maps values in some input space X to the right values in some output space Y. For example, finding a function that maps any vector input $\boldsymbol{x}$ to its Euclidean norm y.

Machine learning systems that apply this principle come in various forms and sizes. A rapidly growing and successful subfield of machine learning is *deep* learning [9]. Deep learning systems aim to model f with multi-layered or *deep* neural networks that represent a parameterized function $f_{\boldsymbol{\theta}}$. How a neural network transforms a given input is determined by its parameters $\boldsymbol{\theta}$. Given a dataset D containing m pairs of complementary inputs and outputs $(\boldsymbol{x}_i, y_i)_{i=1}^{m}$, the goal is to find optimal parameters $\boldsymbol{\theta}^*$ that satisfy

$$\boldsymbol{\theta}^* := \arg\min_{\boldsymbol{\theta}} \mathcal{L}_D(\boldsymbol{\theta}) \tag{1}$$

Where $\mathcal{L}_D$ is an empirical loss function that measures how well the function outputs correspond to the intended outputs from dataset D.

The search space increases rapidly with the number of parameters in $\boldsymbol{\theta}$. Finding the globally optimal parameters for deep neural networks that embody a large number of parameters is therefore infeasible. However, it is possible to iteratively approximate locally optimal parameters. This is achieved by applying gradient descent with respect to the loss function, also known as *training*, allowing adjustment of the parameters along the steepest descent of the loss function. Gradient descent is applied iteratively, where each iteration takes the following form

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{L}_D(\boldsymbol{\theta}^{(t)}) \tag{2}$$

Where $\nabla_{\boldsymbol{\theta}^{(t)}}$ is the gradient with respect to $\boldsymbol{\theta}^{(t)}$, t the iteration number and α a constant that determines how pronounced the updates of the parameters are, also known as the *learning rate*. Determining how many gradient steps are needed to arrive at a good parameter configuration is a research field on its own [10]. In this research, we use a fixed number of iterations to train our deep neural networks.

There are many variations on gradient descent as described above. A commonly used form of gradient descent is *stochastic* gradient descent (SGD). In stochastic gradient descent, the parameters of the neural network are updated for every batch of the training dataset (a fixed-sized sub-collection of the training dataset), instead of per epoch (a full run over all the training data). The size of the batches in SGD forms an additional hyper-parameter. An added benefit of using SGD with a relatively small batch size is a faster convergence rate [11]. In this work, we use exclusively SGD with a batch size of 16, replicating the settings that were used in the work of Ullah et al. [8].

Another common addition to gradient descent is the usage of *momentum* [12]. Momentum can

be implemented in different ways. In our implementation, the formula for gradient descent with momentum becomes as follows

$$\boldsymbol{v}^{(t+1)} = \mu \boldsymbol{v}^{(t)} + \nabla_{\boldsymbol{\theta}^{(t)}} \mathcal{L}_D(\boldsymbol{\theta}^{(t)}) \tag{3}$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \boldsymbol{v}^{(t+1)} \tag{4}$$

Where μ is the momentum coefficient, which is another hyper-parameter. The main intuition behind momentum in gradient descent is to allow the gradient descent to *pick up speed* when the gradient repeatedly points into the same direction, which helps to reach local minima with fewer iterations. Using momentum is therefore found to often improve the convergence of neural networks. It is for this reason that we use SGD with momentum for our experiments, with a momentum coefficient of 0.9 (again, replicating the settings that were used in the work of Ullah et al. [8]).

After a deep learning system has been trained, the desired outcome is that the system *generalizes*. This means that the system also performs well on problem examples beyond the examples that the system was allowed to learn from. A common problem in supervised machine learning that makes this goal difficult to achieve is *overfitting* [13]. This happens when a system learns to model the provided problem examples and their solutions instead of a general method to solve them. An example would be memorizing multiplication tables instead of learning how to perform multiplication. If a multiplication problem is presented that was not included in the tables, you will not be able to solve it. Similarly, overfitted systems cannot solve problem instances beyond the problem instances that they were allowed to learn from.

Overfitting can happen for different reasons. However, in general, it is caused by models that provide more complexity than what is needed to solve the problem. In the case of deep neural networks, we often have limited control over the complexity of the resulting model besides the choice of architecture. Therefore, it is key to provide enough diverse training data to fully employ the complexity a deep neural network can provide to prevent overfitting. From this follows that more complex neural network architectures generally require more training data to be properly trained, and it is for this reason that deep machine learning systems often require vast amounts of data to achieve good results. Besides providing enough training data, applying other methods to limit the risk of overfitting is known as regularization [14]. Examples of regularization in deep learning are *dropout* [15] (temporarily disabling a subset of the network connections to prevent co-adaptations of neurons, or *neurons fixing the problems that other neurons create*), early-stopping (stopping gradient descent at the right moment to prevent the model from learning specific features of the training data) and weight-decay (preventing weights in a neural network to become too large) [16].

2.2 Deep metalearning

As described in the previous section, deep learning is facilitated by gradient descent, which allows us to tune the parameters of deep learning systems so that they generate the desired output. However, iteratively approximating good parameters, or *learning*, is expensive. It requires extensive training data and many iterations to train a deep learning system properly. Even when these conditions are met, deep learning systems are not guaranteed to yield the desired results, for example, due to suboptimal parameter initialization [17] or overfitting (see the previous section). Therefore, methods that optimize the deep learning process and require less data to train a good deep learning model are in high demand.

One field of research that aims to optimize the deep learning process is metalearning. The literal meaning of metalearning is beyond learning. In practice, this means analyzing and evaluating how machine learning systems learn and subsequently learning to optimize the learning process from this prior learning experience. This effectively allows the employment of other (auxiliary) data to optimize the learning process on tasks for which there is little training data.

This general description holds for most metalearning methods. However, the term metalearning

is used in various scopes, which can lead to confusion. In this research, we distinguish two forms of metalearning: *traditional* metalearning, which is used in the context of algorithm selection and hyperparameter optimization, and *deep* metalearning, which focuses on the optimization of the learning process of deep learning methods. In this section, we discuss the principles of deep metalearning.

Deep metalearning aims to minimize the amount of target data needed by deep learning systems, by producing *adaptable* deep learning systems. A deep learning system is considered to be *adaptable* when it can learn an unseen task quickly with limited associated training data [2]. In other words, the metalearning objective is for the system to *learn how to learn*. This can also be stated more formally. We define g_ω as a deep metalearning method that we apply to create an adaptable system. How we apply g_ω depends on ω, which we define as *meta-knowledge*. Meta-knowledge can come in many forms, like trainable parameters, distance measures, and initialization algorithms. If we apply our method g_ω to a distribution of tasks $\boldsymbol{\mathcal{T}}$, we can formulate the metalearning objective as follows [2]

$$\omega^* := \arg\min_{\omega} \mathbb{E}_{\mathcal{T}_j \sim \boldsymbol{\mathcal{T}}}[\mathcal{L}_{\mathcal{T}_j}(g_\omega(\mathcal{T}_j, \mathcal{L}_{\mathcal{T}_j}))] \tag{5}$$

We can dissect the metalearning objective in two parts. The first part is stated between the square brackets and is known as the *inner learning level*. In this part of the equation, we apply our metalearning method g on task $\mathcal{T}_j$ using previously acquired meta-knowledge ω. After our metalearning method has been applied, we evaluate how well the system has *learned* task $\mathcal{T}_j$ by computing the loss with respect to the task symbolized by $\mathcal{L}_{\mathcal{T}_j}$. The rest of the equation is known as the *outer learning level* and states that we want to find meta-knowledge ω that minimizes the probable loss after applying metalearning method g on any task from our task distribution $\boldsymbol{\mathcal{T}}$.

The field of deep metalearning has brought forth various successful methods that find and apply meta-knowledge to create deep learning systems that can adapt quickly to new tasks. They can be roughly divided into three classes

- **Metric-based**. Metalearning methods from this class aim to find feature spaces that are useful for a range of tasks. Examples of metric-based metalearning methods are prototypical networks [18], Siamese networks [19] and matching networks [20].
- **Model-based**. Metalearning methods from this class learn to (partially) adapt the deep learning model to a presented task. Examples are memory augmented neural networks (MANNs) [21] and Meta Networks [22].
- **Optimization-based**. Metalearning methods from this class regard metalearning as a bi-level optimization problem. The first optimization level is a model learning a given task. The second optimization level is identified as the tuning of ω to provide maximum adaptation abilities in the first optimization level. Examples of optimization-based metalearning techniques are model-agnostic metalearning (MAML) [23] and long short-term memory metalearning (LSTM metalearner) [24].

2.3 Evaluation of deep metalearning systems

Since the goal of deep metalearning systems is to adapt to unseen tasks quickly, they are often evaluated in *few-shot learning* settings. This means that the system is only given very few training examples (shots) to learn a new task. Currently, deep metalearning methods are mostly applied to image classification tasks. In these contexts, few-shot learning translates to N-way (number of classes) k-shot classification with low values for k. An illustration of N-way k-shot classification can be found in Figure 1.
The three main forms of few-shot learning for classification problems can be identified as follows.

- **Zero-shot learning**. This means that the learning system is provided with no training examples to learn a new task [26, 27]. In computer vision, methods that solve zero-shot learning problems often rely on pre-trained representations and use distance measures to distinguish the representations of different classes.

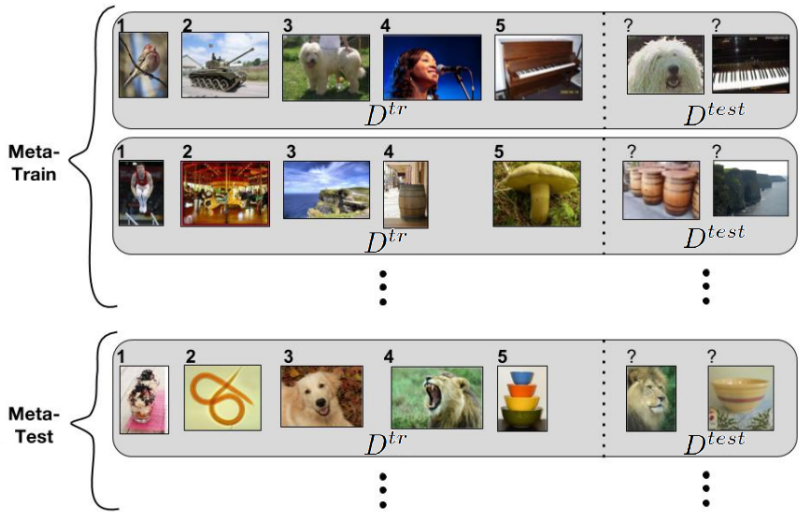

Figure 1: N-way k-shot learning with $N = 5$ and $k = 1$. The support set of each meta-train or meta-test dataset consists of 5 classes with only 1 example. The query set of each meta-train or meta-test set includes different examples of the classes that were present in the support set. This image was adapted from Ravi and Larochelle [25].

- **One-shot learning**. Here, the learning system is provided with a single example per class. Examples of applied one-shot learning can be found in the works of Vinyals et al. [20] and Fei-Fei et al. [28].
- **Multi-shot learning**. This describes all contexts where the learning system is provided with more than one example per class. There is no exact boundary where multi-shot learning can no longer be considered few-shot learning. However, for proper evaluation of deep metalearning methods, the number of training examples should be much smaller than the number of examples that are required to train a model from scratch.

In addition to few-shot learning, deep metalearning systems can also be evaluated in their ability to adapt to tasks that are less related to the training data. In this research, we distinguish three different data relationships (as also stated in the introduction in Section 1).

- **same-source**. In this category, both the training and evaluation tasks originate from the same source dataset. Experiments within this category are the easiest to set up compared to the other categories, as they only require a single dataset with a large number of classes (e.g. miniImageNet [20]). However, these experiments fail to resemble real-world scenarios, where the target task is unlikely to originate from the same source as the auxiliary data. An example of this setup can be found in [23].
- **same-domain**. In this category, the training and evaluation tasks originate from different datasets that are considered to belong to the same domain. There is no exact definition of what a domain is. Examples of domains in image classification are Optical Character Recognition [29], Microscopy [30] and Remote Sensing [31].
- **cross-domain**. In this category, the training and evaluation tasks originate from different datasets that do not belong to the same domain. This is considered the most challenging category, as training and evaluation data are unlikely to be related. Examples of deep metalearning in few-shot cross-domain settings can be found in [32, 33].

2.4 Deep transfer learning and fine-tuning

Another field that investigates the adaptability of machine learning models is the field of transfer learning. The main idea of transfer learning is to transfer knowledge from one machine learning task

to help solve another machine learning task more efficiently [3]. In deep transfer learning, we apply this idea to deep machine learning systems. In this regard, deep transfer learning shares many similarities to deep metalearning, where previously obtained knowledge is also used to help solve a new task via deep learning. However, a key distinction is that deep transfer learning assumes previously acquired knowledge and focuses on the transfer of this knowledge, whereas deep metalearning focuses on the acquisition and application of knowledge that leads to adaptability.

Examples of deep transfer learning are reusing feature representations [34, 35, 36], sharing, combining, or exchanging parameters between models [37, 38], and (partially) fine-tuning previously trained neural networks to solve new tasks [39, 40].

A basic form of deep transfer learning is fine-tuning, which involves making small and partial adjustments to the parameters of a *pre-trained* deep machine-learning model to adapt it to a new target task. The method is based on the expectation that if the pre-training data and target-task data can be described by similar features, the optimal parameters of the pre-trained model are close to the optimal parameters for the target-task data. In other words: the features that are encoded by the different layers of the pre-trained model are expected to be informative for the new task. In practice, fine-tuning means freezing most layers (or features) of the pre-trained model from input to output, except for the last layer(s). The partially frozen pre-trained network is then trained on the new data, only applying gradient descent for the non-frozen layers. Because most model parameters are fixed, much fewer gradient descent update steps are needed for the model to converge and the gradient is easier to compute. This makes fine-tuning less computationally expensive than regular training and lowers the amount of data needed to train the model for the given task. However, good results can only be expected if enough parameters can be shared between the pre-training and target-task data.

2.5 Residual neural networks (ResNets)

Deeper neural networks yield better performance on image classification tasks [9, 41]. However, deeper networks are much harder to optimize compared to their shallower counterparts. The sub-optimal convergence of deeper neural networks is known as the *degradation* problem. To solve this problem, He et al. [42] introduce a residual learning framework: Instead of a series or *block* of neural network layers representing a function $h(\mathbf{x})$, with $\mathbf{x}$ the input vector, they are trained to represent the residual function $h(\mathbf{x})-\mathbf{x}$. This is achieved by adding the original input to the layer block output, or $f(\mathbf{x})+\mathbf{x}$. A residual neural network consists of a series of these residual blocks. Each block contains 2 or more neural layers and represents the function $\mathbf{y} = f_\theta(\mathbf{x}) + \mathbf{x}$, with θ the parameters of the considered layers. Chaining many of these residual blocks allows for the creation of very deep neural networks while mitigating the effect of the degradation problem. It is for this reason residual neural networks or *ResNets* have become a conventional framework for the creation of deep neural networks. In this research, a ResNet consisting featuring 18 convolutional layers (or short ResNet18) is used for the models, where every pair of two layers features a residual connection.

2.6 Automated algorithm selection and meta-models

If we regard a collection of pre-trained models as a set of algorithms that can solve tasks, we can rephrase the problem that we described in the introduction as follows: given a set of algorithms that solve a type of problem, which algorithm should be chosen for a specific problem instance or *task*? The field of automated algorithm selection investigates methods to resolve this question. More formally, given a set of tasks $\mathbf{T} = \{T_1,\ ...\ T_k\}$ of problem type P, a set of applicable algorithms $\mathbf{A} = \{A_1,\ ...\ A_n\}$, and a metric m that measures how well an algorithm $A \in \mathbf{A}$ performs on a task $T \in \mathbf{T}$, the goal is to find a selection strategy S that maximizes the overall performance on $\mathbf{T}$ by applying the right algorithm to the individual tasks [43].

The selection strategy S is always based on previously acquired data on the performance of the available algorithms on different tasks of type P. Depending on certain characteristics of task T, a given algorithm A might perform well or poorly on T. In automated algorithm selection, we search

for computable characteristics or *features* that best predict the performance of an algorithm from set **A** on a task from **T**. If these features are found, a selection strategy can determine which algorithm should be chosen for a previously unseen task based on its features. Ideally, the features should be easy to compute and provide a computational advantage over iteratively finding the best algorithm for a task. Additionally, the features should be computable and informative for a large variety of tasks. Which features are informative enough to guide a selection strategy depends on the problem type and the considered algorithms.

An important part of creating a successful selection system is then to find the right features for the right problems. For example, these works investigate feature sets for the propositional satisfiability problem (SAT) [44, 45, 46]. Similarly, feature sets for other problem types are investigated, like the traveling salesman problem (TSP) [47] and constant satisfaction problems (CSP) [48]. In this work, we make a first attempt to find features that can predict dataset compatibility in a metalearning context.

An algorithm selection strategy can come in different forms. In this research, we train a random forest regressor to act as an algorithm selector. Random forest regression is an ensemble learning strategy for solving regression problems. It involves creating a collection of regression trees and taking the mean of their outputs to form a final prediction. In our application, the final prediction resembles the expected performance of an algorithm-task pair. During the formation of the regression trees, bootstrap aggregating or *bagging* is applied, meaning that the trees are trained on different random subsets of the training data. Similarly, the features that a regression tree can consider for its next data split is also a random subset of the total features, also known as *feature bagging*. The synergy of ensembling, bagging, and feature bagging reduces the risk of overfitting the model to the training data.

2.7 Data

We use Meta-Album as the main data source for our experiments. Meta-Album is a collection of image classification datasets designed for metalearning research [8]. The current version of Meta-Album features 30 datasets from 10 different domains, with each domain contributing 3 datasets to the total. An overview of all domains and corresponding datasets can be seen in Table 1. The diverse and extensive set of domains that Meta-Album offers makes it a suitable choice for experiments in cross-domain settings. It is for this reason that Meta-Album was chosen to form the heart of this research.

Each dataset of Meta-Album is available in four different versions.

- **Original data**. This version features the dataset as it was originally published. The resolution and processing of images differ between datasets. See Table 1 for references to the sources of the original data.
- **Extended**. In the extended version, all images have been up- or down-scaled to a resolution of 128x128 using various processing steps, depending on the original data. All classes of the dataset have at least 40 examples. The number of classes per dataset is at least 20 and differs between different datasets.
- **Mini**. The mini version is the same as the extended version, with the exception that it is class-balanced. Each class now has exactly 40 examples. The number of classes is still at least 20 and differs between datasets.
- **Micro**. The micro version is the same as the mini version, but now every dataset features exactly 20 classes.

In this work, we exclusively use the mini version of the Meta-Album datasets, to ensure class-balance in our experiments.

The creators of Meta-Album have carefully processed all images to remove potential biases between

Domain Name	Dataset Name	Dataset ID	#Classes	#Images	Source
Large Animals	Birds	BRD	315	12600	Birds 400 [49]
	Dogs	DOG	120	4800	Stanford Dogs [50]
	Animals with Attributes	AWA	50	2000	AWA [51]
Small Animals	Plankton	PLK	86	3440	WHOI [52]
	Insects 2	INS_2	102	4080	Pest Insects [53]
	Insects	INS	104	4120	SPIPOLL [54]
Plants	Flowers	FLW	102	4080	Flowers [55]
	PlantNet	PLT_NET	25	1000	PlantNet [56]
	Fungi	FNG	25	1000	Danish Fungi [57]
Plant Diseases	PlantVillage	PLT_VIL	38	1520	PlantVillage [58]
	Medicinal Leaf	MED_LF	25	1000	Medicinal Leaf [59]
	PlantDoc	PLT_DOC	27	1080	Plant Doc [60]
Microscopy	Bacteria	BCT	33	1320	DiBas [30]
	PanNuke	PNU	19	760	PanNuke [61, 62]
	Subcel. Human Protein	PRT	21	840	Protein Atlas [63]
Remote Sensing	RESICS	RESISC	45	1800	RESICS45 [31]
	RSICB	RSICB	45	1800	RSICB 128 [64]
	RSD	RSD	38	1520	RSD46 [65, 66]
Vehicles	Cars	CRS	196	7840	Cars [67]
	Airplanes	APL	21	840	Multi-type Aircraft [68]
	Boats	BTS	26	1040	MARVEL [69]
Manufacturing	Textures	TEX	64	2560	KTH-TIPS Kylberg UIUC [70, 71, 72, 73]
	Textures DTD	TEX_DTD	47	1880	Texture DTD [74]
	Textures ALOT	TEX_ALOT	250	10000	Texture ALOT [75]
Human Actions	100 Sports	SPT	73	2920	100 Sports [76]
	Stanford 40 Actions	ACT_40	39	1560	Stanford 40 Actions [77]
	MPII Human Pose	ACT_410	29	1160	MPII Human Pose [78]
Optical Char. Recog.	OmniPrint-MD-mix	MD_MIX	706	28240	OmniPrint [29]
	OmniPrint-MD-5-bis	MD_5_BIS	706	28240	
	OmniPrint-MD-6	MD_6	703	28120	

Table 1: An overview of all featured datasets of Meta-Album with their corresponding domains.

the different datasets. This includes up- or down-scaling all images to a resolution of 128x128. The processing steps were adapted to the different datasets to reduce artifacts and to facilitate the up- or down-scaling. These processing steps were also adjusted to preserve recognizability by the human eye.

2.8 Box and violin plots

Throughout this work, we will present data using box plots, violin plots, and ensembles of both visualizations. This subsection explains how both types of plots are constructed and how to interpret them.

Box plots The box plot was first formally introduced in 1977 by Tukey et al. [79]. Its main purpose is to provide a robust summary of the distribution of numerical data. This is achieved by visualizing the following statistical features of the data at hand:

- **The median**, which is the data point that separates the higher half of the data from the lower half. The median is shown as a line within the box of the boxplot.
- **The quartiles**, which are medians of the higher and lower half of the data. The quartiles are indicated by the edges of the box.

- **The extremes**, which are often chosen to be the last data points within the range of 1.5 times the interquartile range (IQR, the distance between the quartiles) above and below the quartiles. The extremes are indicated by the whiskers that stick out of the box.
- **The outliers**, which are all the data points that are found beyond the extremes. They are presented as dots or circles beyond the whiskers. Outliers are sometimes omitted from the figure to improve readability.

Box plots allow for quick inspection and comparison of distributions of data. Another advantage is that all indicators always correspond to actual data points. However, in the case of multimodal data, box plots are unable to properly visualize the different modes.

The box plots that we use in this work show all characteristics as explained in the listing. We often omit the outliers to make the figures more readable. In some cases we let the whiskers resemble the complete range of the data instead of 1.5 times the IQR. This exception is explicitly stated for the corresponding figures.

Violin plots Violin plots were introduced by Hintze and Nelson [80] in 1998 as an improvement on box plots. In contrast to box plots, violin plots show the full (estimated) distribution of the data. This allows violin plots to visualize multimodal data.

Violin plots are formed by kernel density estimation (KDE). The KDE is formed by applying a kernel function to each data point and summing up the results. More formally, given data points $(x_1, x_2, ..., x_n)$, we can define an estimate $\hat{f}$ of the actual density function f as follows:

$$\hat{f}_h(x) = \frac{1}{nh} \sum_{i-1}^{n} \mathcal{K}(\frac{x - x_i}{h}) \tag{6}$$

Where $\mathcal{K}(x)$ is the kernel function. The parameter h is known as the bandwidth and controls the range of influence of a single data point, in turn controlling how smooth the KDE will be. Since the KDE is an estimation of the true density function, assumptions are made that fill in gaps in the provided data. This means that some parts of a violin plot do not correspond to actual data points, in contrast to box plots. Another possible distortion of violin plots is that the smooth kernel functions can cross the natural boundaries of the data (e.g. probability always falling in the range between 0 and 1). This can be resolved by cutting the kernel functions at the lowest and highest data points.

In this work, violin plots are created by the use of a standard normal kernel function. The value for h is chosen according to Scott's rule [81]. We cut the KDE at the lowest and highest data points to prevent the KDE from passing natural boundaries.

3 Related work

In this section, we present published research from recent years that has a scope that is similar to the work that we present. Specifically, we look at research that incorporates a selection strategy in the context of deep metalearning to improve performance.

Selecting feature representations

A possible method to adapt previously trained models to a new task is to gather a collection of pre-trained models and subsequently select and combine their individual feature representations depending on the given task.

Dvornik et al. [82] propose a system called SUR (Selecting from Universal Representations). SUR is a selection algorithm that learns how to linearly combine different feature representations for the best results on a given target task. In this sense, SUR can be regarded as a **metric-based** method (see Section 2). The different feature representations are generated by feature extractors that are pre-trained on different datasets. They apply their method to image classification datasets and tasks, where a nearest-centroid classifier is trained on the selected feature representations of the support set to make predictions on the query set. They find that the selection algorithm creates mostly sparse linear combinations of the different feature representations. In most cases, the feature representation that is assigned the highest weight originates from the same domain as the target task or *same-source*.

Liu et al. [83] expand on the work of Dvornik et al. by employing a transformer network to adapt the combined outputs of different pre-trained models (the universal representation) to the provided target task. Similarly to the work of Dvornik et al., a single-head universal representation transform (URT) can provide a linear combination of feature representations of different pre-trained models. In addition, they propose a multi-headed URT, which allows for the linear combination of different linear combinations of pre-trained models. Depending on how the URT heads are initialized, they form feature representations from different pre-trained models. The authors show an example of a URT head that selects primarily feature representations from pre-trained models that were created using the same source data as the task.

Li et al. [33] propose to adapt a distance-based classifier to new tasks by calibrating the distance matrix to an unseen task with the help of its support data. To achieve this, the original distance matrix is complemented by a non-linear transformed version of itself. The non-linear transformation is formed by principle component analysis and aims to reduce the distance representation to the most informative features while suppressing uninformative training features, which in essence can be regarded as selecting feature representations. Before the two distance matrices are combined, they are both calibrated by linearly combining a Euclidean-distance-based ranking with a more class contextual ranking representation of the feature-mapped instances.

Task selection

Another way to incorporate selection to improve metalearning performance, is to select meta-train tasks instead of random sampling. The meta-train tasks are selected to provide the most benefit to the metalearning system. The work of Yao et al. [84] is an example of this. They strive to optimize the meta-model training process by selecting tasks from a task distribution more informedly, instead of uniform sampling. For this, they propose an adaptive task scheduler (ATS). The ATS is formed by training a neural scheduler that considers the effect different candidate tasks have on the meta-model parameters. Using this method, they can better select tasks that are more beneficial to the meta-model training progress, at the cost of higher computational demand for training the scheduler. Yao et al. tested this by applying the ATS in a situation where the distribution of meta-train tasks was supplemented with noisy tasks, and they found that the ATS was able to assign a lower sampling weight to the noisy tasks. Because the ATS capitalizes on the dynamics of bi-level optimization, it can only be applied to the optimization-based metalearning methods as discussed in Section 2.1.

Selecting parameters

It is also possible to partially select the initialisation parameters of a neural network based on the given task. Triantafillou et al. [85] consider a metalearning system where a pre-trained *universal template* of parameters can be adapted to new tasks by providing additional, task-specific parameters. To achieve this, they form a feature extractor by combining pre-trained (universal) parameters and a selection of dataset-specific parameters. The task-specific parameters are initialized by a pre-trained set that corresponds to one of the pre-training datasets. To determine which set of task-specific initialization parameters should be used, the authors propose a dataset classification network that can predict from which dataset a set of images originates. The output of this network is used to determine the compatibility between a set of examples and a set of initialization parameters. This is key according to the authors, as they report that the system is unlikely to recover from a bad initialization.

Selecting pre-trained models

Finally and most closely related to our work is the work of Pineda-Arango et al. [86]. They propose a Combined Algorithm Selection and Hyperparameter Optimization (CASH) technique which they call Quick-Tune. Quick-Tune uses Bayesian optimization to determine which pre-trained models are most likely to be successful candidates to be fine-tuned to a given task. For this, they use the validation error during fine-tuning. More time is invested in pre-trained models that show the most promising progress in their validation accuracy over the fine-tuning epochs. They conclude that their method outperforms large transformer backbones for computer-vision tasks. In contrast to their work, we base our selection strategy on additional meta-features that are based on the raw data of the datasets. However, we have also found that the fine-tuning validation accuracy has high predictive value for selecting the best performing pre-trained model.

4 Methods and general experimental setup

This section explains the different stages and components that form the methods of this research. For clarity, we divide the methods into five consecutive stages.

1. **Data preparation**. In this stage, we split datasets from Meta-Album into meta-train datasets and meta-test tasks, that are used later to create pre-trained models and fine-tune them. See Section 4.1.

2. **Training**. In this stage, we create a pre-trained model for each meta-train dataset. See Section 4.2.

3. **Fine-tuning**. In the fine-tuning stage, we take the pre-trained models we created in Stage 2 and fine-tune them to the meta-test tasks we created in Stage 1. After fine-tuning, we record the accuracy of the fine-tuned network, which we will use later as the target value of our meta-dataset in Stages 4 and 5. See Section 4.3.

4. **Meta-data and meta-model**. This stage involves extracting features from the meta-train datasets and meta-test tasks and pairing them with the recorded accuracies to form our meta-data. We then use a part of the meta-data to train a meta-model to predict the recorded accuracy based on the provided dataset-specific features. See Section 4.4.

5. **Meta-model validation**. Finally, we use the remaining part of the meta-data to validate the meta-model. We rank the absolute accuracy predictions of the meta-model and compare the predicted ranking to the true ranking of our meta-data. See Section 4.5.

As can be distilled from this overview, many parts of the methods involve the division of datasets into train datasets and test datasets or tasks. We use different random seeds to control how data is randomly split in these operations. Seed s is used to split datasets from Meta-Album into meta-train and meta-test splits. Therefore, by changing the seed s, the composition of these data subsets changes, ultimately leading to different pre-trained models and fine-tuning tasks. Seed t is used to split meta-test datasets further into tasks, and corresponds to the task number. Finally, a fixed, singular seed is used throughout the stages to split datasets into support and query datasets.

Throughout this section, we refer to various entities that play a role in our methods. Here, we present an overview of the most important entities and their corresponding description.

- **Source dataset**. A complete dataset that forms the source of different, class-separate meta-train and meta-test datasets.
- **Meta-train dataset**. A subset of a source dataset that is used to pre-train a model.
- **Meta-test dataset**. A subset of a source dataset that is used to create different tasks to which pre-trained models are fine-tuned.
- **Meta-test task**. A subset of a meta-test dataset that forms a task to which a pre-trained model is fine-tuned. In a classification context, a meta-test task forms an N-way k-shot classification task (see Section 2.3).
- **(Pre-trained) models**. Deep learning models that were pre-trained on a meta-train dataset and that are later fine-tuned to a meta-test task.
- **Meta-models**. Models that predict the validation accuracy of a pre-trained model on the query set of a meta-test task after it was fine-tuned to its corresponding support set.
- **Support set**. The portion of a dataset that is used to train or fine-tune a deep learning model.
- **Query set**. The portion of a dataset that is used to validate the performance of a deep learning model (the model is not allowed to change its parameters with this data).

Further details on the different stages of our methods are outlined in the following subsections. The final subsection explains our actual implementation of the methods that form the general setup for our experiments in the subsequent sections. Further details on the experiment-specific implementations can be found in Section 5 and Section 6.

4.1 Data preparation

To achieve a fair evaluation of the fine-tuning performance of the different pre-trained networks, we need to ensure that the classes of the meta-train dataset do not overlap with the classes of the meta-test dataset. For this, we carefully split each Meta-Album dataset into two random, class-separate partitions. The random class-based split is controlled by a single seed s. Since seed s controls which classes are used to train the pre-trained models and which classes are used to fine-tune these models later on, we can create different models by changing s. The resulting meta-test dataset after the random class-split is split further on a class level into p different N-way tasks. This task split is also performed randomly using seed t, which corresponds directly to the number of the task. This process is illustrated in Figure 2.

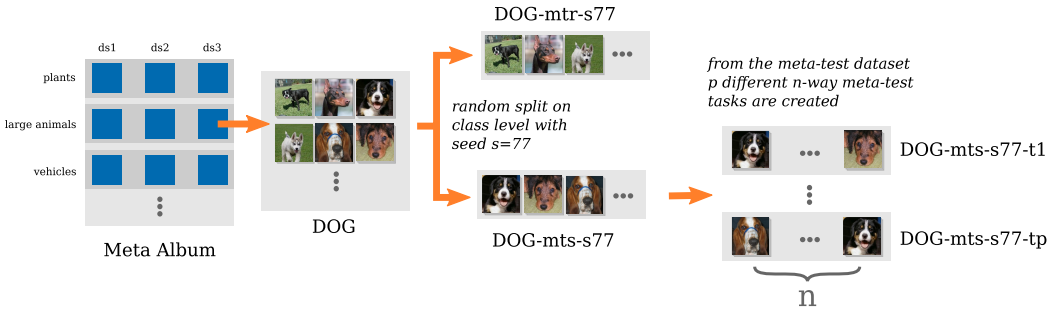

Figure 2: A schematic example of how we prepare the data for our experiments. We start with Meta-Album, which is a collection of datasets from various domains (see Section 2.7). Each dataset of Meta-Album consists of multiple classes (typically > 25, see Table 1). We select a dataset (in this case DOG) and randomly split it on a class level using a random seed s (in this case $s = 77$). This results in a meta-train dataset (top) and a meta-test dataset (bottom), which we label DOG-mtr-s77 and DOG-mts-s77 respectively. The meta-test dataset is split further into p N-way meta-test tasks. The meta-train set is used to create a pre-trained network (see Figure 3). The meta-test tasks are used to later fine-tune the pre-trained models to (see Figure 6). This data preparation is applied to all datasets of Meta-Album and can be repeated for multiple seeds s to form different pre-trained models and tasks.

4.2 Pre-training models

To create a pre-trained classification model, we use a meta-train dataset that we created in the previous stage. We split the meta-train dataset on an instance level into two stratified sets: a support set, which contains the training examples, and a query set, which we will use to validate the performance of the model in between training epochs. Since the split is stratified, both sets contain the same amount of examples per class. We perform this split randomly based on a fixed seed. For each training epoch, we record the loss and accuracy of the model on the support (training) and query (validation) set. Figure 5 shows an example of the development of these measures for each epoch. After the model is trained, we store it to be used later in the fine-tuning stage of our method. See Figure 3 for a schematic example.

4.3 Fine-tuning

After all meta-test tasks have been formed and all models have been pre-trained, we fine-tune every pre-trained model to every task (that was generated using the same seed s to ensure class separation).

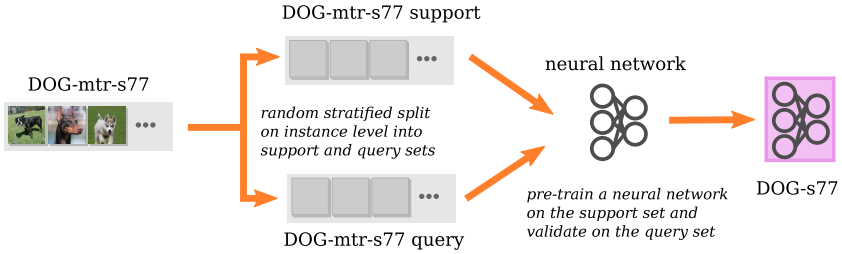

Figure 3: A schematic example of how we create the pre-trained models. We take a meta-train dataset (for example, DOG-mtr-s77) as created in Figure 2, and randomly split it further into a meta-train support dataset and a meta-train query dataset. The split is performed on an instance level and in a stratified fashion, meaning that both the support and query datasets contain different examples from the same classes and that the classes are equally represented. The support set is then used to train a neural network, while the query set is used to validate the performance during training. An example of train and validation accuracy and loss during training can be observed in Figure 5. After training has been completed, the neural network resembles a pre-trained model of the source dataset (DOG-s77), which will later be fine-tuned for new tasks (see Figure 6). In this way, a pre-trained model is created for every dataset of Meta-Album. The procedure can be repeated for multiple seeds s to create different pre-trained models for the same source dataset.

We refer to the act of fine-tuning a single pre-trained model to a single task as a fine-tuning job. For each fine-tuning job, the meta-test task is split into a support and query set, similar to how the meta-train dataset was split in the previous step. The support set is then provided to the pre-trained model as training data. In contrast to the training stage, only the last non-output layer is adjusted this time. We record the loss and accuracy on the support and query set for each epoch of fine-tuning. An example of these values over the epochs is shown in Figure 4. The final validation accuracy on the query set is later used to train the meta-models. A schematic overview of the fine-tuning stage is shown in Figure 6.

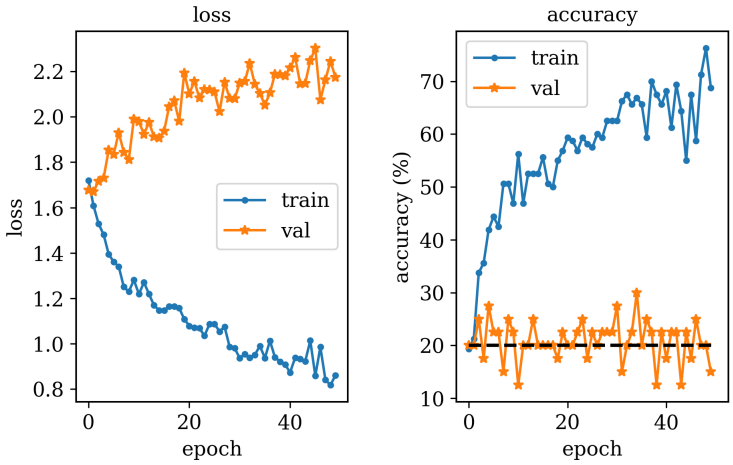

Figure 4: The training and validation loss and accuracy per fine-tuning epoch of a ResNet18 trained on the PLK (plankton) meta-train dataset transferred to task 0 of the MD_MIX (OmniPrint-MD-mix) meta-test dataset. The dashed black line shows the baseline performance of $1/c \cdot 100\%$ with c the number of classes in the meta-train dataset, in this case $c = 5$

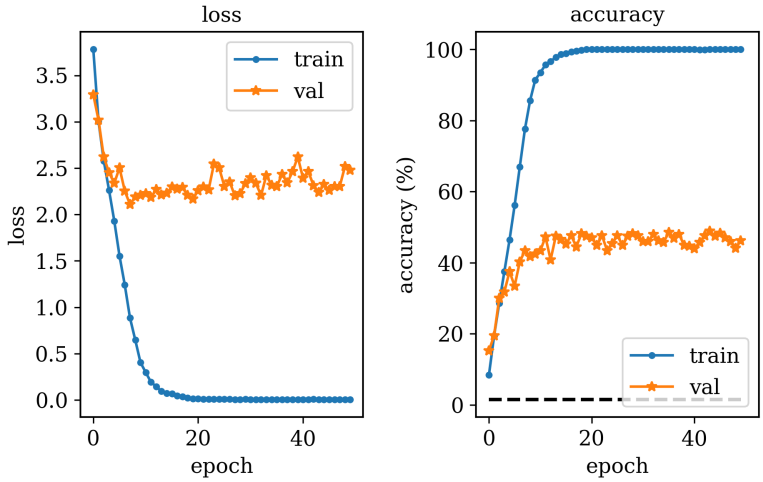

Figure 5: The training and validation loss and accuracy per training epoch of a ResNet18 architecture trained on the PLK (plankton) meta-train dataset. The dashed black line shows the baseline performance of $1/c \cdot 100\%$ with c the number of classes in the meta-train dataset, in this case $c = 68$.

4.4 Meta-model training

Using the validation accuracy values from all fine-tuning jobs, we construct a dataset, where each record of this dataset is complemented by features that correspond to the concerning fine-tuning job. An overview of all the different meta-features that we use in this work is shown in Table 4 (more details about the used features can be found in Section 6). This dataset forms our *meta-dataset* and serves as training and test data for different meta-models that predict the fine-tuning validation accuracy. After the meta-dataset is constructed, it is split into a training and test partition. In the implementation we use for our experiments, we split the data for leave-one-out cross-validation on a meta-test task level (see Section 6 for more details). The training partition is used to train different meta-models and create baseline models. The test split is used to validate the performance of these models. A schematic overview of the fine-tuning stage is shown in Figure 7.

4.5 Meta-model validation

The different meta-models and baseline models are evaluated on their ability to select the right pre-trained model for a given meta-test task. In essence, this is similar to the principles of automated algorithm selection (see Section 2.6). The meta-test tasks can be regarded as the different tasks **T** of problem type P, in this case image classification. The pre-trained models can be regarded as the different algorithms **A**, that we apply, in this case fine-tune, to solve the tasks. We apply the meta-models to the test split of our meta-dataset and collect their predictions. Since the meta-models predict absolute fine-tuning validation accuracy, we transform the predictions into a ranked representation. In this form, the predictions of the meta-models can be used as a selection strategy, where we pick the pre-trained model with the highest predicted rank for a given fine-tuning task. To evaluate a predicted ranking, we compare it to the actual ranking of the recorded performances in the test split of the meta-dataset. See Figure 8 for a schematic representation of this procedure.

4.6 General experimental setup

For our experiments, we apply the methods described in the previous sections as follows. In the data preparation stage as described in Section 4.1, we use an 80 versus 20 percent train-test ratio for the meta-train and meta-test datasets respectively. This means that the meta-train dataset contains 80 percent of the original dataset classes versus 20 percent in the meta-test dataset. The class split is performed randomly, based on a chosen seed s. The meta-test datasets are split further into 5 5-way tasks, again using an 80 versus 20 percent ratio for the support-set and query-set. The task split is

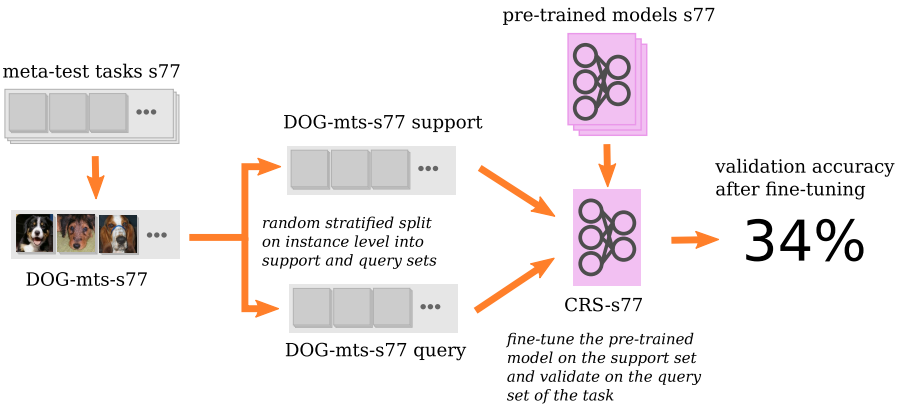

Figure 6: A schematic example of the fine-tuning of a pre-trained model to a new task. We select a meta-test task (DOG-mts-s77) as created in Figure 2 and split it further into a meta-test task support dataset and a meta-test task query dataset, similarly to Figure 3. Next, we select a pre-trained network (Cars with $s = 77$, or CRS-s77) as created in Figure 3. The layers of the pre-trained network are frozen, except for the last layer. Then, the partially frozen pre-trained network is trained (fine-tuned) to the meta-test task support set. The fine-tuning process is validated with the meta-test task query set. An example of train and validation accuracy and loss during fine-tuning can be observed in Figure 4. After fine-tuning has been completed, the validation accuracy (accuracy of the fine-tuned network on the query set) is recorded. In this example, the CRS-s77 model was able to correctly classify 34% of the examples of the query set of the DOG-mts-s77 task after fine-tuning. The recorded validation accuracy is later used as the target value for the meta-models (see Figure 7). This procedure is repeated for all possible pairs of tasks and pre-trained models that originate from the same seed. Tasks and pre-trained models from different seeds are never combined, as the class separation between meta-train and meta-test is not guaranteed between different seeds.

also performed randomly, this time based on a chosen seed t, where t corresponds directly to the task number. Since every class has exactly 40 examples (we use the *mini* version of each Meta Album dataset, see Section 2.7), this results in a 32-image per class support-set and an 8-image per class query-set for every task. We repeat the procedure for every dataset of Meta-Album, resulting in a collection of 30 meta-train datasets and 150 meta-test tasks per seed s.

We use the meta-train datasets to create pre-trained models. A ResNet18 neural network architecture, as described in Section 2.5, is used as the backbone of the models. We split the meta-train dataset into a support and query set, again using an 80 versus 20 percent ratio. The support set is provided to the model as training data in batches with a size of 16 examples. The network output is evaluated using cross-entropy-loss with respect to the true label of the provided examples. Stochastic gradient descent (SGD) is used to tune the model parameters with respect to the loss, using a learning rate of 1e−3 and a momentum of 0.9. After all batches of training data are processed in this way, the model is evaluated using the query set, on which it is not allowed to tune its parameters. The loss and accuracy of the model on the query set are recorded and later used to inspect how the model progressed through the epochs. See Figure 5 for an example of the recorded validation accuracy and loss over the epochs. In total, we repeat this procedure for 50 epochs. After 50 epochs, the model is stored and forms the pre-trained model.

After all pre-trained models are formed, we fine-tune them to the previously created meta-test tasks. This is done similarly to how the models were trained. We split the meta-test task on an instance level into a support and query set using an 80 versus 20 percent ratio. Again, 50 epochs are performed, using the same criterion, optimizer, and corresponding hyper-parameters as we used during pre-training. This time, however, we freeze all layers of the ResNet18 architecture, except for the last layer before the output layer. After each epoch, we record the validation accuracy on the query set of the meta-test task. See Figure 4 for an example of the validation and training accuracy over the

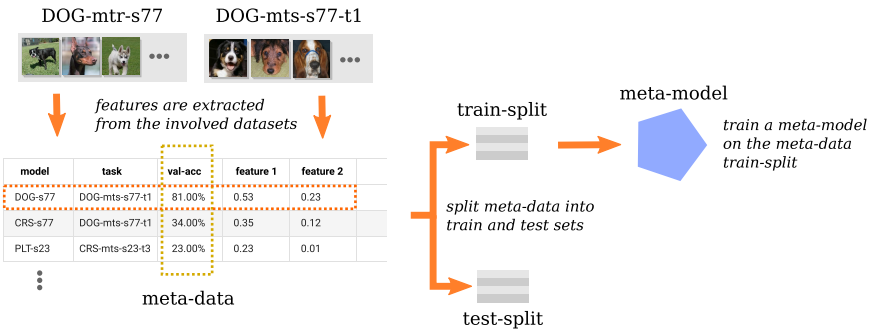

Figure 7: A schematic example of the assembly of meta-data and the training of meta-models. The meta-data consists of the recorded fine-tuning accuracy and meta-features of the involved datasets per fine-tuning job. The meta-data is split into a train and test set. The train set of the meta-data is used to train a meta-model that predicts the validation accuracy using the provided features.

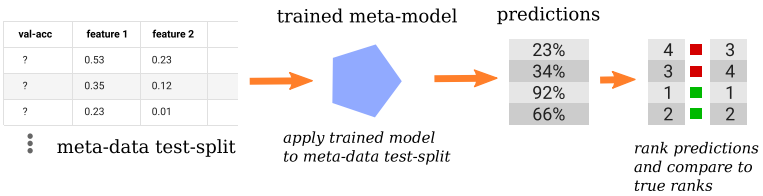

Figure 8: A schematic example of the meta-model validation. The test split of the meta-data is given to a trained meta-model. The meta-model then outputs the predicted values for the validation accuracy. The predictions are ranked and compared to the ranking of the true ranking of the validation accuracy.

epochs during fine-tuning. After the 50 epochs have been completed, we take the mean validation accuracy of the last 10 epochs and include it in our meta-dataset. For the sake of readability, we will refer to this average of the validation accuracy of the last 10 epochs as the *fine-tuning performance*. The reason we use the average of the last 10 epochs instead of the last epoch is to mitigate stochastic effects that are present in the training procedure, as can be seen in Figure 4. We repeat this for every model and task combination within a seed s, resulting in $30 \times 30 \times 5 = 4500$ records per seed.

We accompany every record in our meta-dataset with features that correspond to the involved datasets. An overview of the different features that we use is shown in Table 4. Averages and standard deviations of color channels are calculated over all pixels, to prevent possible distortions from taking double averages. In contrast, averages and standard deviations of colorfulness and entropy are calculated per image and then averaged. More details on how we formed and applied our feature sets can be found in Section 6.

We repeat the complete procedure as described above for 3 seeds. The complete meta-dataset then contains $3 \times 4500 = 13500$ entries, combining the validation accuracy of the fine-tuning jobs with at most 40 features of their corresponding datasets, depending on the chosen feature set. The complete procedure was also repeated using a ResNet34 network backbone but resulted in no noteworthy difference in performances across the models.

The scripts and procedures that were used to run the experiments can be found via the following GitHub link: `https://github.com/MTTVDN/data-centric-meta-learning`.

5 Transferability of datasets

In this section, we aim to answer research questions 1 and 2 (see Section 1). For this, we compare pre-trained models from a given Meta-Album dataset in their ability to transfer to tasks from another given Meta-Album dataset. We refer to this ability of different pre-trained models from dataset A to transfer to different tasks from dataset B as the *transferability* of A to B. The transferability is expressed in percentages and is defined as the average fine-tuning performance per meta-train dataset (pre-trained models) and meta-test dataset (tasks) combination. For example: the transferability of the BRD dataset to the PLK dataset is the average fine-tuning performance of all models that were pre-trained on the BRD meta-train dataset in fine-tuning jobs to tasks from the PLK meta-test dataset. We compare this measure of transferability for the three different categories that describe the *relationship* between the meta-train dataset and the meta-test dataset (a more general description of the categories can be found in Section 2.3):

- **same-source**. The meta-train dataset and the meta-test dataset originate from the same source dataset. For example, a model trained to classify 50 classes of birds from the BRD dataset is fine-tuned to solve a 5-way task with different classes of birds from the BRD dataset.
- **same-domain**. The meta-train dataset and meta-test dataset originate from different source datasets that are considered to belong to the same domain (note that this relationship is disjoint from the **same-source** relationship). For example: a model trained to classify 50 classes of birds from the BRD dataset is fine-tuned to a task with 5 different classes of dogs from the DOG dataset. Both the BRD and DOG dataset belong to the domain *large animals*.
- **cross-domain**. The meta-train dataset and the meta-test dataset originate from different source datasets that are also not in the same domain. For example: a model trained to classify 40 classes of insects from the INS dataset is fine-tuned to a classification task involving 5 classes of boats from the BTS dataset. The INS dataset belongs to the *small animals* domain, whereas the BTS dataset belongs to the *vehicles* domain.

In the following subsections, we first describe how the different visualizations are created to analyze our meta-dataset and transferability. We continue with a discussion where we share our observations of the statistical properties of the meta-dataset and relate our observations to the initial research questions.

5.1 Setup and results

To visualize the transferability between the meta-train datasets and meta-test datasets for the different dataset relationships, we created a matrix where rows represent the meta-train dataset and columns represent the meta-task dataset. We refer to this type of matrix as a *transferability matrix*. In addition to the singular cells that are separated by white lines, the transferability matrix also shows thicker, black lines between groups of cells to indicate the domains as defined by the authors of Meta-Album. All cells within a box with a black outline belong to the same domain. The transferability matrix that visualizes the true recorded transfer performances is referred to as the *true* transferability matrix and is shown in Figure 9. We make this distinction, as the following sections will also introduce transferability matrices for the predictions of meta-models. The cells of a transferability matrix represent all fine-tuning jobs that involve pre-trained models that were trained on a meta-train dataset derived from the corresponding row dataset, and tasks from a meta-test dataset of the corresponding column dataset. Inside the cells, the mean and corresponding standard deviation of the fine-tuning performance of these fine-tuning jobs are reported. The fine-tuning performance of an individual fine-tuning job is measured as the mean validation accuracy of the last 10 fine-tuning epochs out of 50 (for example, Figure 4 shows a fine-tuning job with a transfer performance of $\sim 20\%$). The color of the cells illustrates the transferability relative to the transferability of other cells in the same column. A green cell has a relatively high transferability and a red cell has a relatively low transferability. We also show the same data in a transferability matrix with a color scale that corresponds to the range of the complete matrix in Figure 11.

Since our experiments with meta-models in Section 6 are focused on the rank of the pre-trained models for a given task, we also present the true transferability matrix in a rank-based format in Figure 10 to allow for better comparison. In this representation, cells report how often the concerned pre-trained models occur in the top-5 of models, given the target tasks from a given column. Since each cell represents 15 different tasks, the cell values range from 0 (red) to 15 (green) and the total sum per column is always $5 \times 15 = 75$. The right-most column shows the sum per row and the top-5 row sums have been accented. The cell in the right-bottom corner reports the trace of the matrix.

Additionally, we show violin plots with embedded box plots of the fine-tuning performance, where we compare the distributions of different subsets of the meta-dataset. Examples are Figure 12 and Figure 13. The box plots inside the violin plots show the global median (central white line) and the lower and higher quartile (as shown by the boundaries of the thicker rectangle). The vertically extending lines of the box plots resemble the whiskers, which indicate 1.5 times the interquartile range (IQR, the distance between the lower and higher quartile) above the higher quartile and below the lower quartile.

5.2 Discussion

Our first observations are concerned with Figure 9, showing the true transferability matrix. In the coloring of this matrix, we observe an accented diagonal structure, corresponding to the same-source category. This indicates that models that were pre-trained on data from the same source dataset as the meta-test task perform better than models that were pre-trained on other datasets. The same diagonal is also present in the coloring of the rank-based representation in Figure 10. In total, the fine-tuning jobs that are represented by the diagonal belong to the top-5 models in 298 out of the 450 cases (66%) (for comparison: if all pre-trained models are equally likely to be part of the top-5, the expected sum of the diagonal is $\frac{5}{30} \times 15 \times 30 = 75$ or 17%). These observations indicate that in terms of rank (relative to other pre-trained models), we can expect pre-trained models to transfer well to tasks from the same source dataset via fine-tuning.

In terms of absolute performance, this picture is more nuanced. In Figure 12, we see that fine-tuning jobs that belong to the same-source category do not necessarily yield good absolute performance. It is not uncommon for a pre-trained model that is involved in a same-source fine-tuning context to only mildly improve on the baseline fine-tuning performance of 20%, as can be observed in the lower ranges of the same-source distribution plot in Figure 12. Examples of low performance by the same-source model can also be seen in Figure 15. We are therefore unable to conclude that pre-trained models are always well-adaptable to tasks from the same source dataset in a fine-tuning context in terms of absolute performance. However, since we know that fine-tuning jobs from the same-source category rank generally higher than other model-task combinations, regardless of absolute performance, we can expect fine-tuning jobs from the other categories to perform worse, even if the performance on the diagonal is already low. This suggests the existence of *hard tasks*, which are tasks to which no pre-trained model can be fine-tuned well, regardless of their dataset relationship. In Figure 15, we show examples of meta-task datasets that show these characteristics. In total, we have recorded 15 tasks on which models from the same-source category reached a performance of less than 30%. We see that for most of these tasks, few or no other models could perform notably better than the same-source models.

To gain more insight into the range of performances across the different datasets, we created Figure 14. This figure shows box plots of the recorded fine-tuning performances for all meta-test datasets, where the box plots are ordered left to right from smallest to largest range. The whiskers of the box plots show the full range of the considered data subset (instead of the conventional 1.5× IQR, see Section 2.8). Inspection of the left-most box plots reveals that there exist meta-test datasets that bring forth tasks that are exclusively easy or exclusively hard for the available pre-trained models to solve.

One other notable observation from Figure 12 is the bimodal distribution of fine-tuning performance values in the same-source category. The same bimodal distribution is not found in the other cate-

gories. This suggests that there is a specific advantage that pre-training on the same source dataset gives and that it is not always applicable. Further comparison of the distributions reveals that in the lower end of performances, the same-source category improves on the same-domain category, which in turn improves on the cross-domain category. Especially for fine-tuning jobs that were least successful in a given category, we see improvement as we go from right to left. A portion of fine-tuning performances in the cross-domain category even falls below the 20% baseline performance, which corresponds to random guessing the class in the classification task. In the same-source category, only improvements with respect to random guessing can be observed.

In addition to the diagonal, we observe pronounced rows in the coloring of both Figure 9 and Figure 10. This implies the existence of foundation models that generalize over all different tasks better than others. From the top-5 ranking counts shown in the right-most column of Figure 10, we find that pre-trained models from the datasets *BRD, FLW, RSICB, TEX* and *TEX_ALOT* form the overall top-5. We compare the absolute fine-tuning performance of this overall top-5 to the rest in Figure 13. For better comparison of the distributions, we exclude the same-source records from the top-5 records. We observe that in terms of absolute performance, the difference between the top-5 models and the other models is again less clear-cut. The distribution of the top-5 models category has slight bimodal features, again suggesting that in some specific cases, these models offer an advantage.

The apparent effect of the meta-train and meta-test dataset belonging to the same domain is inconclusive. The advantage of the two datasets belonging to the same domain is most strongly visible for the OCR domain (Optical Character Recognition, see Table 1) in both Figure 9 and Figure 10. The Large Animals domain (BRD, DOG, AWA) and the Remote Sensing domain (RESICS, RSICB, RSD) also hint at this effect in Figure 10. However, since both domains happen to include a meta-train dataset that produces models that belong to the overall top-5, it is unclear how much of the advantage can be attributed to the same-domain category. For the other domains, a possible correlation is even more inconclusive. This is also reflected in Figure 12, where the difference between the same-domain and cross-domain distribution is small. The most notable difference is the distribution of the mid-range performances between 50% and 80%, where we see a slight advantage for the same-domain category.

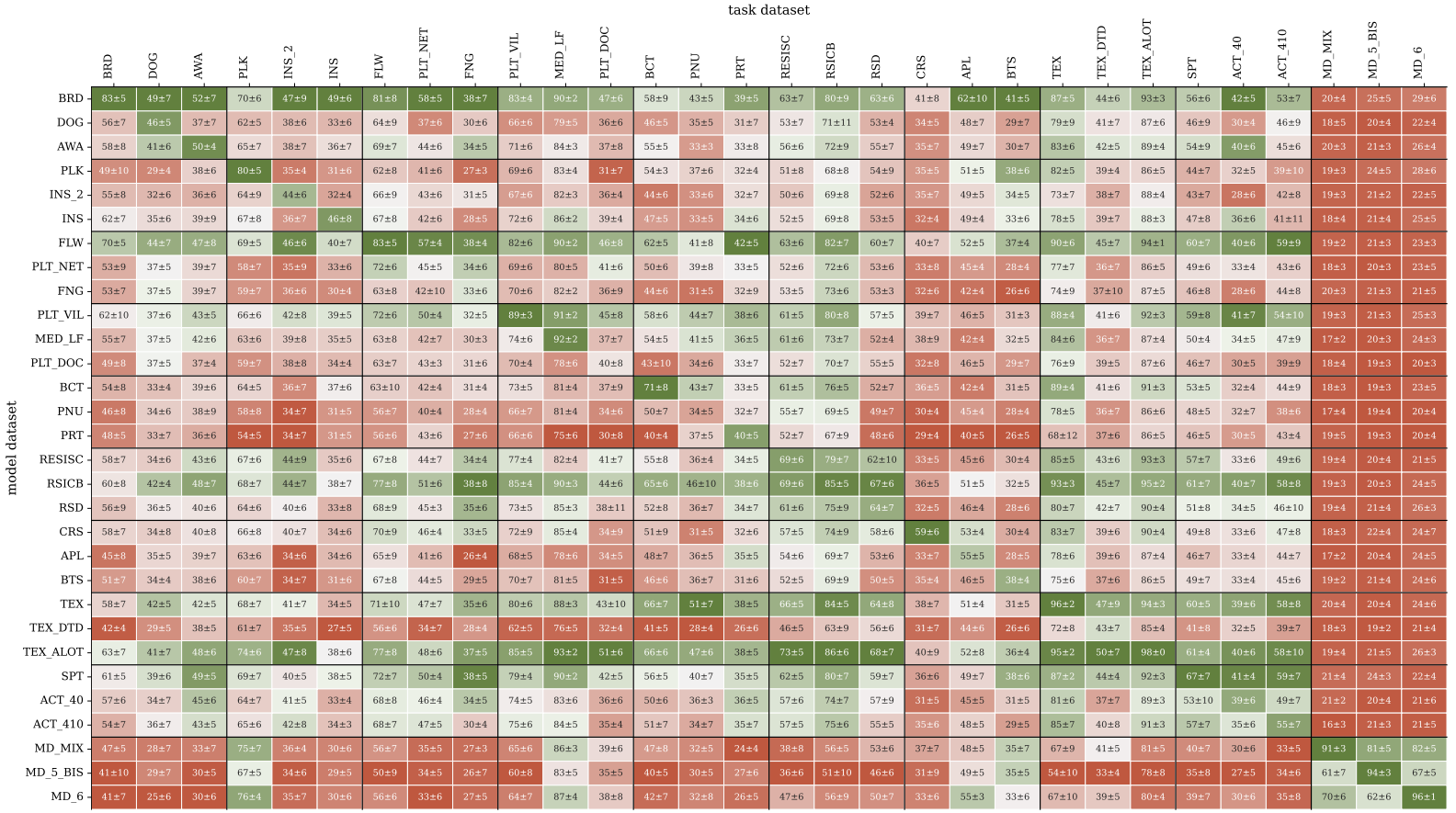

Figure 9: The true transfer matrix. Each cell reports the mean and standard deviation of the transfer validation accuracy of the pre-trained models originating from the dataset of its row label fine-tuned to tasks from the dataset of its column label. The black accented lines that bundle groups of 9 cells separate the different dataset domains as defined by the authors of Meta-Album. The cells are colored relatively to other cells in the same column, where green corresponds to a higher relative value and red to a lower relative value.

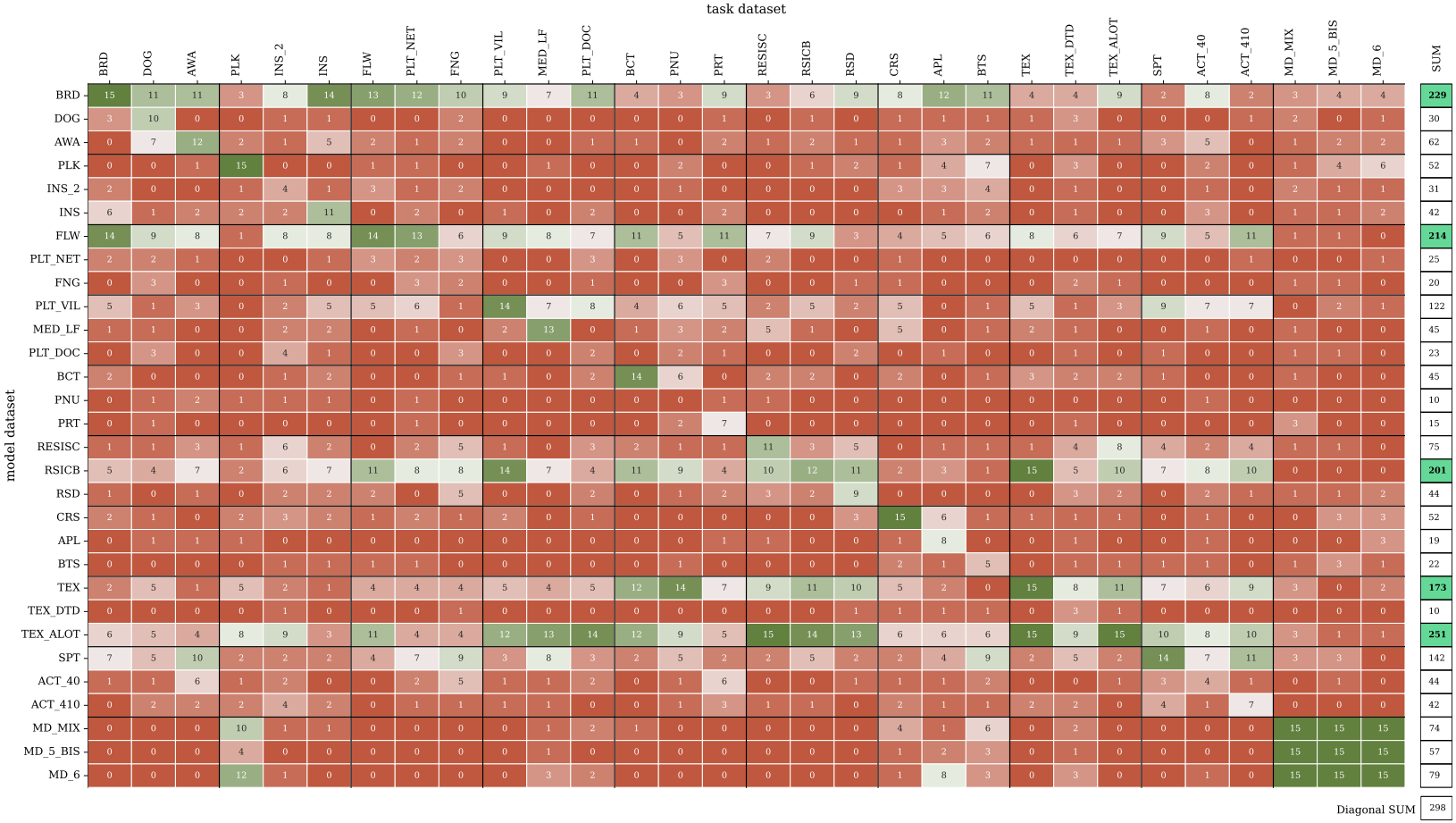

Figure 10: The true transfer matrix with rank counts. Each cell shows how often a pre-trained model from a given row is part of the top-5 models for the tasks of a given column. of the transfer validation accuracy of the pre-trained models originating from the dataset of its row label fine-tuned to tasks from the dataset of its column label. The accented lines separate the different dataset domains. The right-most column shows the total sum per row. The top-5 rows have been accented.

Figure 11: The true transfer matrix with absolute colors. Each cell reports the mean and standard deviation of the transfer validation accuracy of the pre-trained models originating from the dataset of its row label fine-tuned to tasks from the dataset of its column label. The black accented lines that bundle groups of 9 cells separate the different dataset domains as defined by the authors of Meta-Album. The cells are colored with a color scale that corresponds to the range of the complete matrix. Green indicates a higher relative value and red indicates a lower relative value.

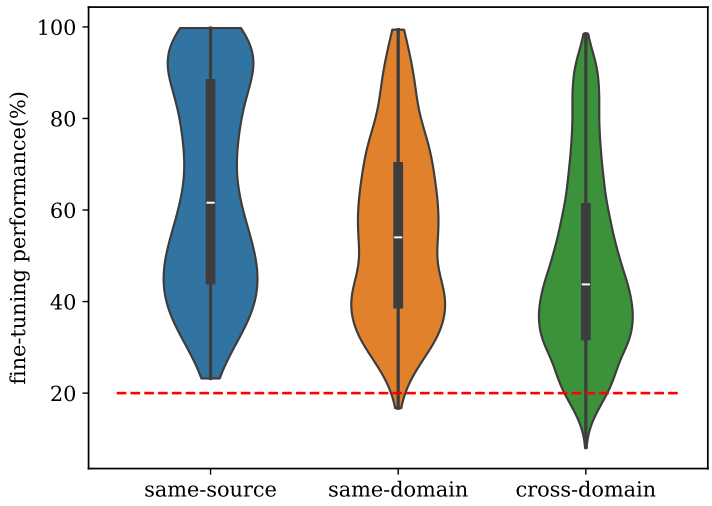

Figure 12: Violin plots with embedded box plots of the fine-tuning performance for records that belong to the **same-source** (diagonal), **same-domain** and **cross-domain** dataset relationships. The dashed red line indicates the baseline fine-tuning performance at 20%.

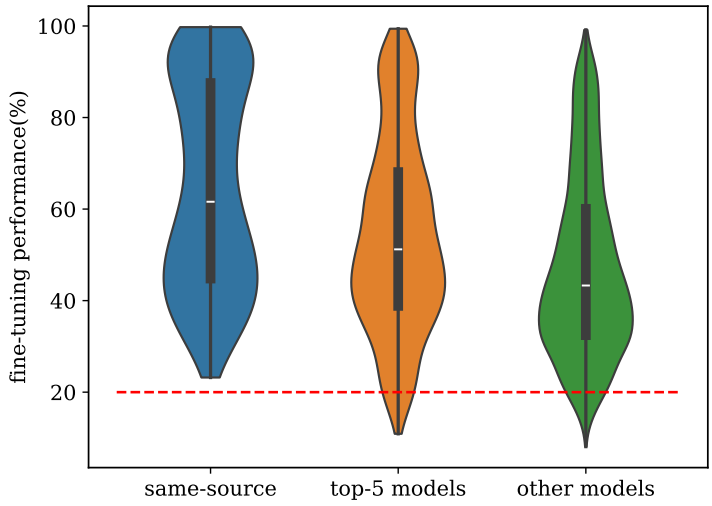

Figure 13: Violin plots with embedded box plots of the fine-tuning validation accuracy for different subsets of the meta-data. The violin plot labeled *same-source* presents the fine-tuning validation accuracy of all records that lie on the diagonal of the true rank-count matrix shown in Figure 10. The violin plot labeled *top-5* corresponds to records that involve a pre-trained model that belongs to the top-5 models as identified in the true rank-count matrix. Finally, the violin plot labeled *other models* corresponds to all records that do not involve pre-trained models from the top-5. The dashed red line indicates the baseline fine-tuning performance at 20%.

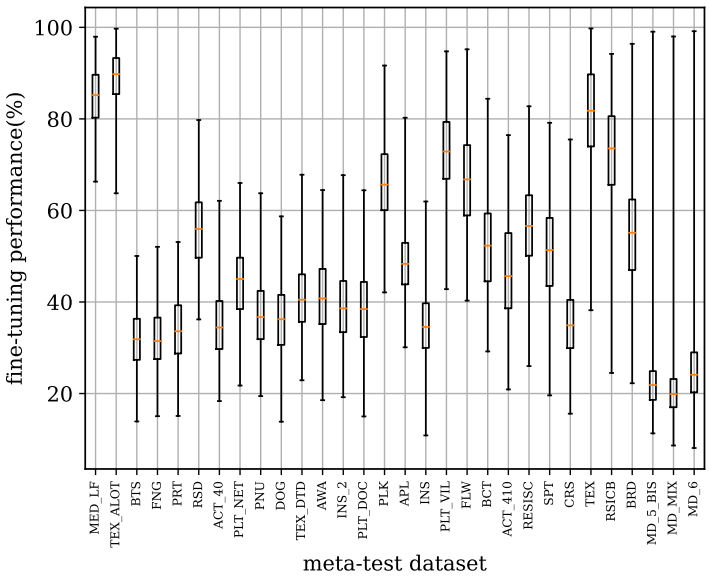

Figure 14: A series of box plots, showing the distribution of fine-tuning performances for all meta-test datasets, ordered from smallest to largest range (left to right). For each boxplot, the orange line indicates the median, the box shows the range of the 1st and 3rd quartile, and the whiskers indicate the **full range** of the data (in contrast to other box plots).

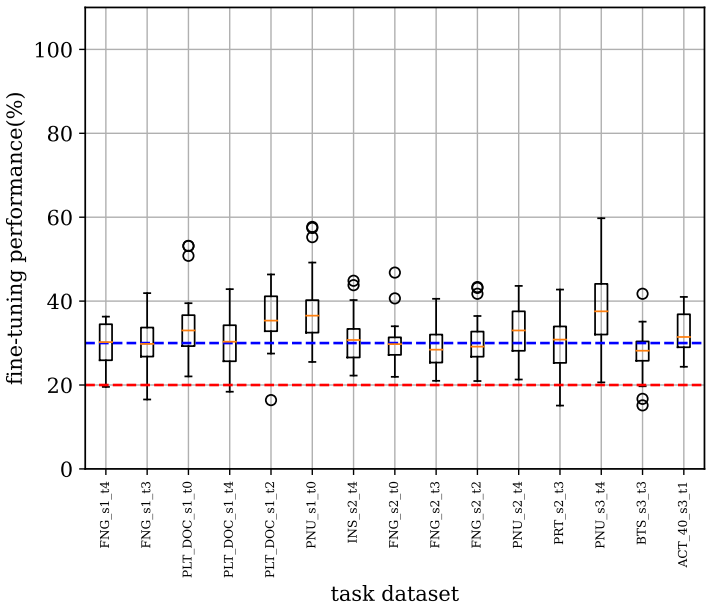

Figure 15: 15 examples of *hard tasks*, where the fine-tuning performance of the same-source models was below 30% (indicated by the blue dashed line). The dashed red line indicates the baseline fine-tuning performance at 20%. The circles display the outlying data points.

Parameter	Value
n_estimators	500
criterion	squared_error
max_depth	7
min_samples_split	2
min_samples_leaf	1
max_features	0.5

Table 2: Overview of the parameters that we used to create the random forest models with the scikit-learn RandomForestRegressor class implementation of random forest regression [87].

6 Selecting pre-trained models

In this section, we aim to answer research question 3 as follows. With the data from our meta-dataset, we train different meta-models to predict fine-tuning performance. The meta-models are created using random forest regression, by providing various feature sets that are based on dataset characteristics. The purpose of these meta-models is to select the best pre-trained model for a meta-test task, given the features corresponding to the meta-test task and the available pre-trained models. In the following experiments, we evaluate the meta-models on their ability to select the best available pre-trained model for a given meta-test task. We supplement the evaluation of the meta-models with various baseline measures, to put the performances of the meta-models into perspective.

The next subsections are structured as follows. We start by explaining how we trained our meta-models, how we created our baselines, and what experiments we conducted to evaluate and compare their performance. Next, we present the results of our experiments in two sections. In the first section, we show the results of the experiments that adhere to the **scientific** context, where we include the pre-trained models that were trained on data from the same source dataset as the task. In other words: the same-source pre-trained model for a given task is available for selection. In the second section, we show the results of the experiments that resemble a more **realistic** context, where we exclude these pre-trained models. We finish with a discussion of all results, where we compare the results from both contexts and share our observations.

6.1 Setup

The previously introduced meta-dataset provides the training data and validation data for the different meta-models and baseline meta-models. From this meta-dataset, we split off all records where the meta-test task is derived from a specific meta-test dataset, and use the remainder as training data for the meta-model. The meta-models are created using random forest regression. We use the implementation provided by the scikit-learn Python package under the class RandomForestRegressor [87] using the parameters shown in Table 2. Once the meta-model is trained, it predicts the transfer validation accuracy of the split-off records. The model predictions are then ranked according to the predicted transfer validation accuracy. For each meta-test task, the top-ranking combination is selected. When multiple combinations are given the same top-ranking prediction, the candidate with the actual lowest fine-tuning performance is selected to mitigate potential randomness in the evaluation. In other words: if a model cannot decide between multiple models, the worst possible model is selected (the following results therefore all show the worst-case scenario for the meta-models and baselines). Then, the selected combinations are ranked according to their true rank. Finally, we count how many times the meta-model selected a pre-trained model in the top-1 (best), top-2, top-3, top-4, and top-5 for the given tasks of the split-off meta-test dataset. This procedure is repeated for all meta-test datasets, resulting in a leave-one-out (the chosen meta-test dataset) cross-validation setting. When all folds of the leave-one-out cross-validation have been completed, we compare the meta-models and baselines on their achieved ranking score for the former five categories (best, top-2, etc).

Feature set	Separate pixel-based features for meta-train and meta-test dataset	Differences in pixel-based features between meta-train and meta-test dataset	Validation accuracy after one epoch of fine-tuning
rf separate	✓	✗	✗
rf difference	✗	✓	✗
rf separate LCP	✓	✗	✓
rf difference LCP	✗	✓	✓
rf full LCP	✓	✓	✓

Table 3: Overview of the different feature sets that were used to train the different meta-models via random forest regression. The different pixel-based features are shown in the top part of Table 4.

In addition to counting the ranks of the model selection, we use a *selection loss* metric on which we also evaluate the meta-models and baselines. The selection loss is defined as the difference between the actual fine-tuning performance of the selected pre-trained model and the actual fine-tuning performance of the best pre-trained model. A low selection loss therefore means that the meta-model selected a pre-trained model that performed similarly to the actual best pre-trained model in terms of absolute fine-tuning performance, regardless of its rank.

We test the different meta-models according to this procedure in 2 different contexts. First, a **scientific context** where a pre-trained model originating from the same dataset as the target task is available for selection (same-source). Second, a **realistic context**, where a pre-trained model originating from the same dataset is not available for selection.

For both contexts, we evaluate meta-models that were trained with different feature sets. We refer to these meta-models by the name of their feature set. We start with two feature sets that include only statistical measures of the raw data, which are shown in the upper section of Table 4. Since these features are calculated using only the pixel values of all images that are included in a dataset, we refer to these features as *pixel-based*. The pixel-based features are initially calculated separately for the meta-train dataset and the meta-test task. The first feature set is labeled as *rf separate* and includes the pixel-based features of the meta-train dataset and the meta-test task separately. The second feature set is labeled as *rf difference* and only includes the absolute difference between the pixel-based features of the meta-train dataset and the meta-test task.

We continue by adding a low-cost proxy (LCP) feature to the former pixel-based features. We define the LCP as the validation fine-tuning accuracy after one epoch of fine-tuning. Using the LCP, we create three new feature sets. A complete overview of the different feature sets and how they are composed is shown in Table 3.

The meta-models that result from using the former feature sets are compared with three baseline meta-models. The *model average* baseline model uses the average fine-tuning performance value of a pre-trained model as the prediction for a fine-tuning job involving this pre-trained model. This can be interpreted as taking all values of a row (corresponding to a model dataset), except for a single cell (corresponding to a column of the meta-test dataset), from Figure 11, and using the average value as a prediction for the excluded cell. An example of the predictions of the *model average* baseline model is shown in Figure 16. For comparison, we show the predictions of the *rf separate* meta-model in Figure 17. The *random expected* baseline model visualizes the expected outcome of a model that randomly selects a pre-trained model for a given task. The *same-source* baseline model always selects the pre-trained model that was pre-trained on the same source dataset from which the given task originates. This baseline model can only be applied in the *scientific* context, where this pre-trained model can be selected. Finally, we use the *LCP* baseline model, which selects pre-trained models purely on the fine-tuning validation accuracy after one epoch of fine-tuning. This baseline is only included when meta-models using the LCP feature are involved.

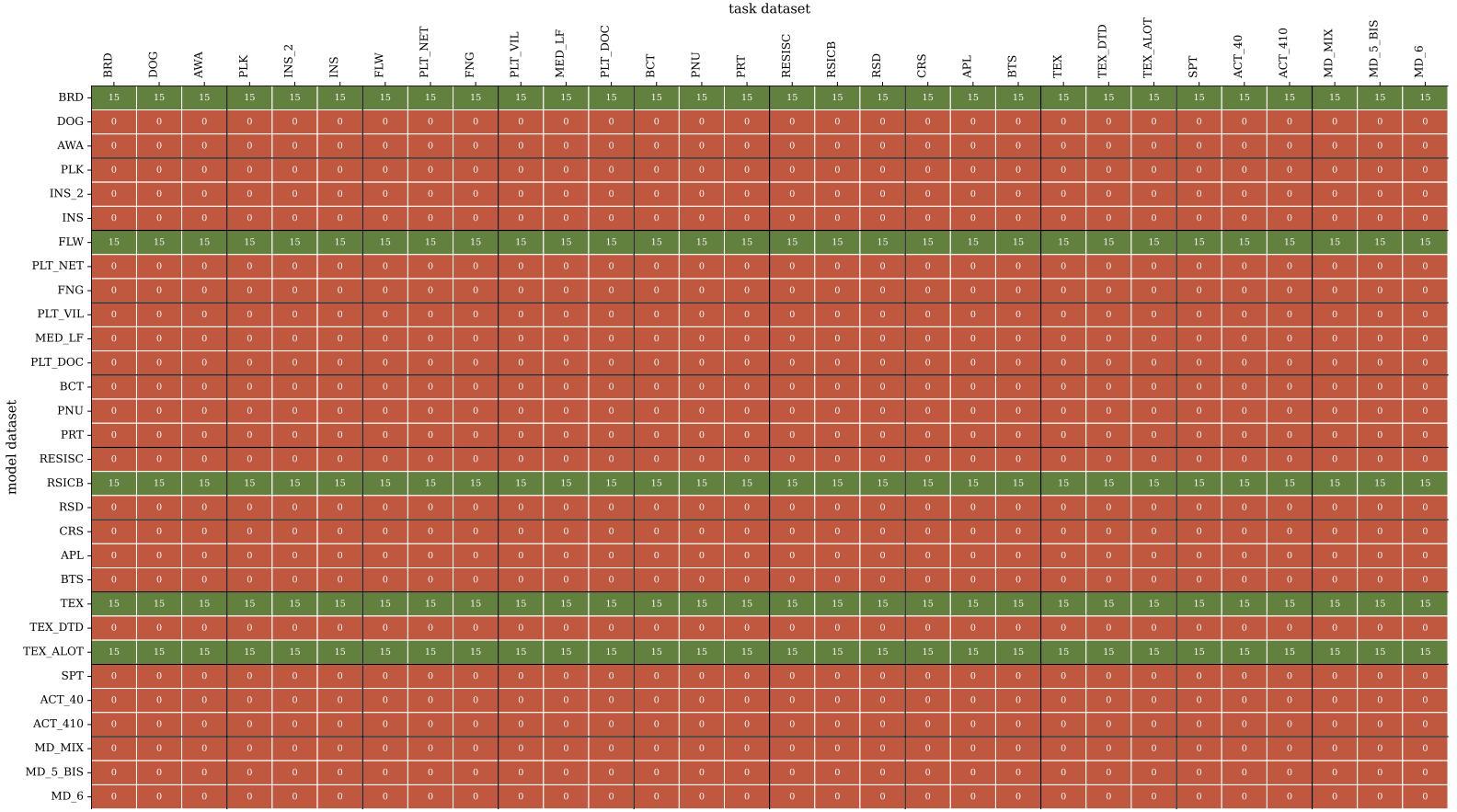

Figure 16: The transfer matrix with rank counts as predicted by the model average baseline. Each cell shows how often a pre-trained model from a given row is predicted to be part of the top-5 models for the tasks of a given column. of the transfer validation accuracy of the pre-trained models originating from the dataset of its row label fine-tuned to tasks from the dataset of its column label. The accented lines separate the different dataset domains.

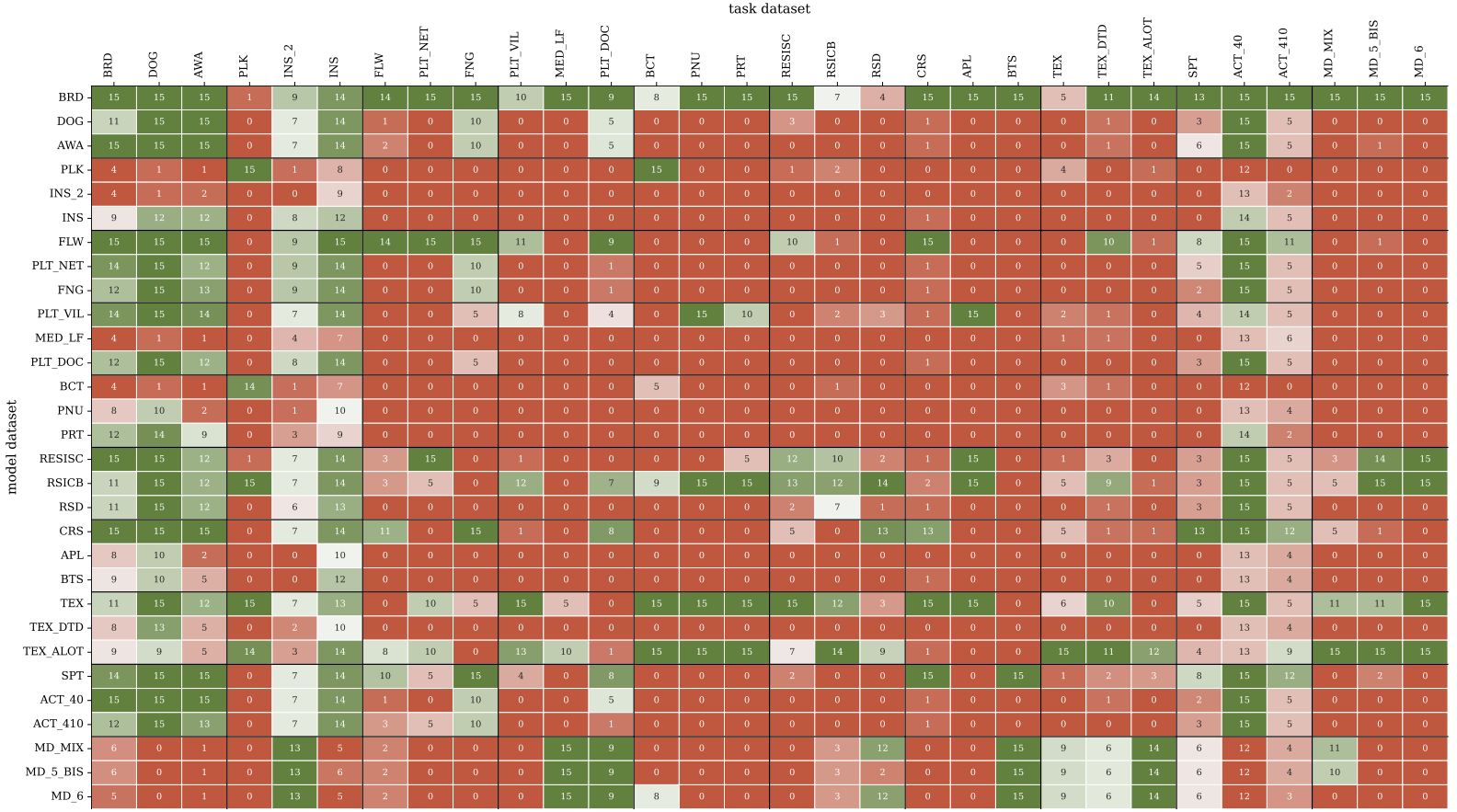

Figure 17: The transfer matrix with rank counts as predicted by the meta-model that was trained on the rf separate feature set. Each cell shows how often a pre-trained model from a given row is predicted to be part of the top-5 models for the tasks of a given column. of the transfer validation accuracy of the pre-trained models originating from the dataset of its row label fine-tuned to tasks from the dataset of its column label. The accented lines separate the different dataset domains.

Feature name	Description
global mean	the mean pixel brightness value of the dataset.
global std	the standard deviation of the pixel brightness values of the dataset.
mean_c*	the mean pixel value in channel c (r, g or b) of the dataset.
std_c*	the standard deviation of pixel values in channel c (r, g or b) of the dataset.
colorfulness mean	the mean colorfulness of the dataset (as defined by Hasler and Susstrunk [88]).
colorfulness std	the standard deviation in colorfulness, as defined by Hasler and Susstrunk, of the dataset.
entropy mean	the mean Shannon entropy of the dataset
entropy std	the standard deviation in Shannon entropy of the dataset
low-cost proxy	the validation accuracy of the first fine-tuning epoch
target accuracy	the average validation accuracy of the last 10 fine-tuning steps of a foundation model applied to the target task

Table 4: A descriptive overview of the different meta-features that are calculated for all meta-training-set and meta-test-task pairs. Features with the c* placeholder are calculated separately for the red, green, and blue channels of the images. Except for *target accuracy* and *low-cost proxy*, all features are calculated equally for the meta-training-set and the meta-test-task.

6.2 Results pixel-based features

In Figure 18 we compare the meta-models *rf separate* and *rf difference* and the baseline meta-models in their ability to select pre-trained models from top-5 for a given meta-test task in the *scientific* context. We use a bar chart to show how often the selection of a meta-model belonged to the top-1, top-2, top-3, top-4 and top-5 for all meta-models and baseline meta-models. In Figure 19 we present violin plots with embedded box plots to show the *selection loss* for the same meta-models and baseline meta-models in the *scientific* context.

We continue by showing the same plots for the *realistic* context in Figure 20 and Figure 21. In these plots, the same-source baseline model is omitted, as these pre-trained models cannot be selected in the *realistic* context.

6.3 Results low-cost proxy

The results involving the LCP feature are structured similarly to the previous results. In Figure 22 we compare the meta-models *rf separate LCP*, *rf difference LCP* and *rf full LCP*, and the baseline meta-models in their ability to select pre-trained models from top-5 for a given meta-test task in the *scientific* context. In addition to the previous baselines, we add an *LCP* baseline, which predicts the rank of a pre-trained model based on its first fine-tuning epoch validation accuracy. In Figure 19 we present violin plots with embedded box plots to show the *selection loss* for the same meta-models and baseline meta-models in the *scientific* context.

The same plots are shown in Figure 23 and Figure 25 for the *realistic* context. In these plots, the same-source baseline model is omitted, as these pre-trained models cannot be selected in the *realistic* context.

In addition, we show the distribution of the difference between the validation accuracy after one epoch of fine-tuning and the final fine-tuning performance in Figure 27.

6.4 Discussion

In the first set of experiments, we use feature sets that only consist of pixel-based features. In Figure 18 we observe that the random forest meta-model that uses the difference in pixel-based features performs worse than selecting pre-trained models at random (random expected). In this regard, this meta-model proves to be unusable for model selection. Next, we find that the *rf separate* meta-model,

which bases its predictions on the separate pixel-based features of the meta-train dataset and meta-test task, improves on the other random forest meta-model. For selecting the best pre-trained model, the *rf separate* model performs similarly to the model average baseline. However, both meta-models are only able to select the best-performing model in $\sim 7\%$ of cases. For identifying models from the other ranks, the model average baseline proves to be better than the *rf separate* model. Similarly to our observations from Section 5, we find that same-source pre-trained models often belong to the top-5 performing pre-trained models for a given task.

In Figure 19 we get a better view of the selection distribution of the meta-models. The biggest difference in the distributions of *rf separate* and *rf difference* is found in the density around a loss of 0%, which is reflected in the bars for the best category shown in Figure 18. In other ranges of the graph, we see that the distributions of both meta-models are similar. Both meta-models have selected pre-trained models that have a loss of more than 70%. The model average baseline is slightly better compared to the two random forest meta-models, but the difference is small. The *same-source* baseline outperforms all other shown meta-models and baselines. It made no selection with a loss of more than 35%. This further shows that if the same-source model is not the best-performing model, other models rarely perform much better.

We show the same plots for the *realistic setting* in Figure 20 and Figure 21. Apart from a slight improvement in the *model average* baseline, the performance of the meta-models and baselines remains similar to the realistic setting in terms of rank counts. In the case of the distributions, we also see no mentionable differences apart from the shortening of the tail. We suspect this is due to the smaller differences between the fine-tuning performances that exclude the same-source performances.

The unimpressive results from using solely pixel-based features inspired us to explore the use of a low-cost proxy (LCP) in the form of the fine-tuning validation accuracy after one epoch of fine-tuning. In Figure 22 we show the rank-counts after introducing the LCP feature to our feature sets for the scientific context. We observe a great improvement of the meta-models in comparison to the results shown in Figure 18. Using the *rf full LCP* feature set, we were able to correctly identify the best-performing pre-trained model in more than a third of the cases. The figure also shows that with the *rf full LCP* feature set, we slightly improve upon the *same-source* baseline. The *rf separate LCP* and *rf difference LCP* feature set perform slightly worse than the *rf full LCP* feature set. We also observe that these two meta-models add little to no benefit to the *LCP* baseline.

In terms of *selection loss*, we also see large differences after introducing the LCP feature in Figure 24. All meta-models were able to select models that never miss out on more than 40% fine-tuning performance. We also observe that in terms of distribution, all meta-models and the *LCP* and *same-source* baselines are very similar.

In the realistic context, shown in Figure 23, we see a small decrease in performance across all random forest meta-models in contrast to the scientific context. In addition, the difference in performance between the feature sets has disappeared. The performance difference between the *LCP* baseline and the meta-models is also further decreased. In terms of *selection loss*, shown in Figure 25, we observe that the distributions of the meta-models have become thicker in the 0% to 20% range. This shows us that when the meta-models are not able to select the best-performing pre-trained model, the pre-trained model they do select often performs slightly worse than the best option.

Finally, we wanted to investigate how similar the LCP feature is to the final fine-tuning performance, to rule out the possibility that the pre-trained models had already converged after one epoch of fine-tuning. For this, we created Figure 27, where we show a histogram for the differences between the value for the LCP feature and the final fine-tuning performance for all records in our meta-dataset. We find that the LCP value is not exclusively equal to the final fine-tuning performance and that different developments of the fine-tuning accuracy are possible after the first fine-tuning epoch.

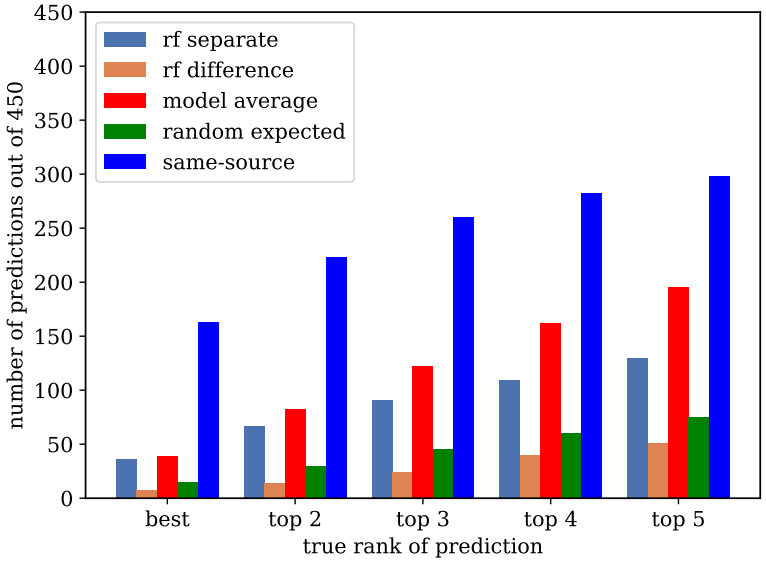

Figure 18: Bar chart showing how often different meta-models and baseline meta-models selected a pre-trained model that belongs to the top-1 (best), top-2, top-3, top-4, and top-5 for a given meta-test task in the *scientific* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on the corresponding feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model always selects the pre-trained model that was trained on the same source data as the meta-test task.

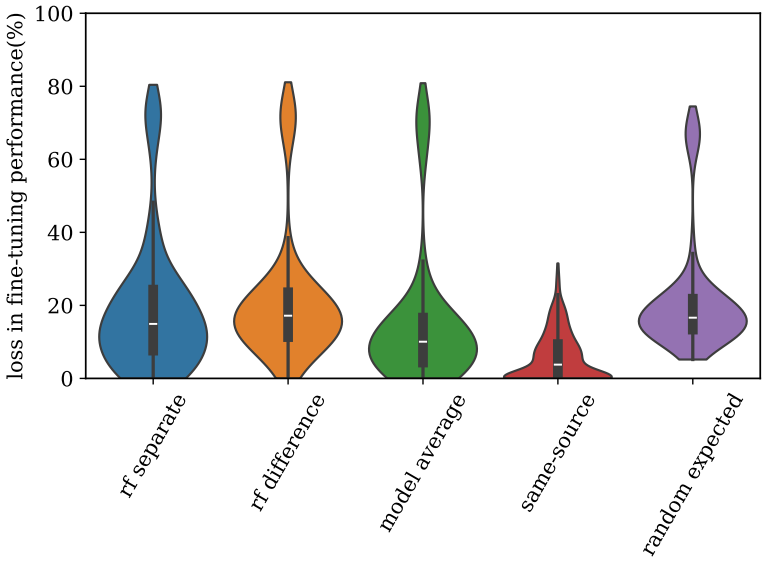

Figure 19: Violin plot with embedded box plots showing the *selection loss* of different meta-models and baseline meta-models in the *scientific* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random* baseline model shows the expected loss when selecting pre-trained models at random, which is calculated as the average loss of all pre-trained models per meta-test task. The *same-source* baseline model always selects the pre-trained model that was trained on the same source data as the meta-test task.

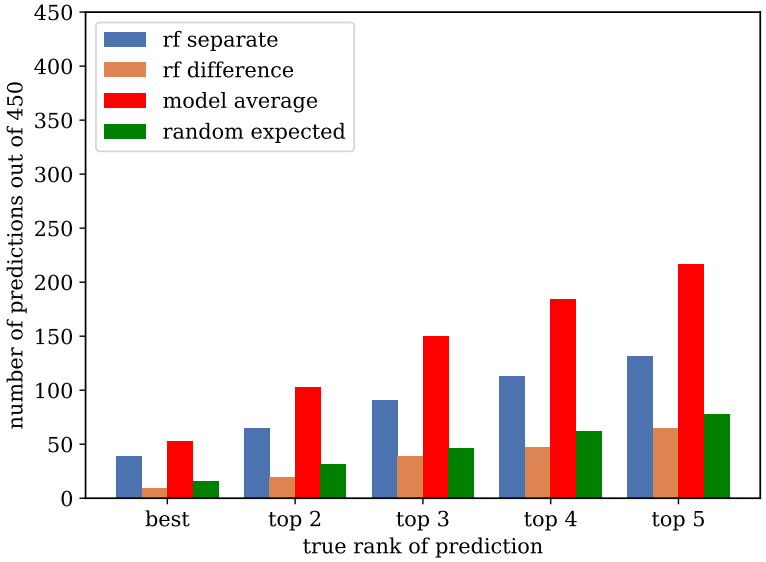

Figure 20: Bar chart showing how often different meta-models and baseline meta-models selected a pre-trained model that belongs to the top-1 (best), top-2, top-3, top-4, and top-5 for a given meta-test task in the *realistic* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model as included in Figure 18 is excluded from this figure, as the *realistc* context does not allow selection of a pre-trained model that was trained on the same source data as the given meta-test task.

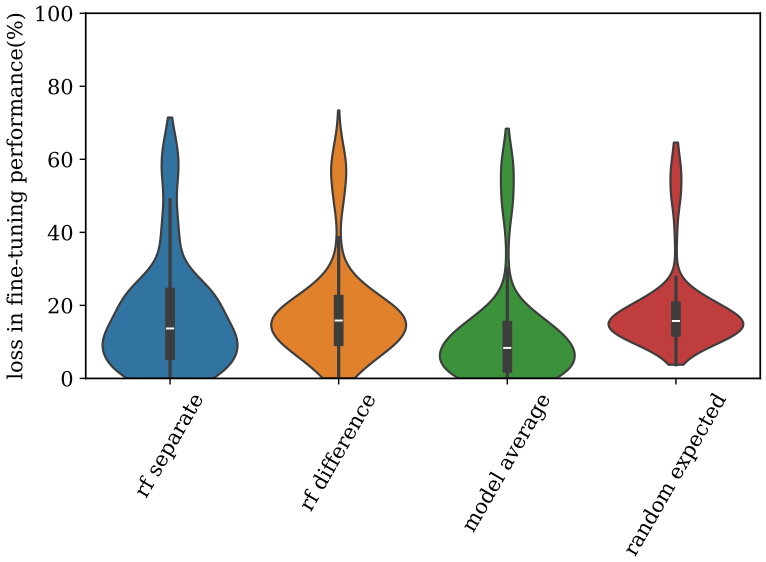

Figure 21: Violin plot with embedded box plots showing the *selection loss* of different meta-models and baseline meta-models in the *realistic* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model always selects the pre-trained model that was trained on the same source data as the meta-test task.

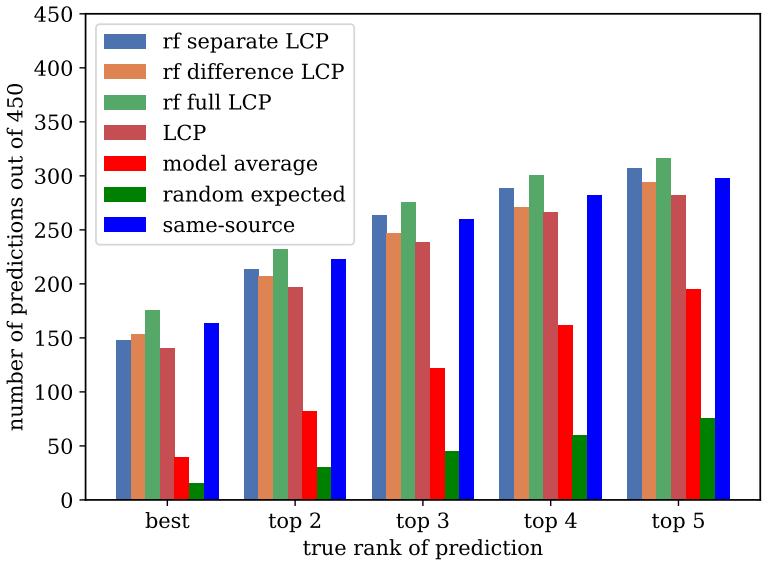

Figure 22: Bar chart showing how often different meta-models and baseline meta-models selected a pre-trained model that belongs to the top-1 (best), top-2, top-3, top-4, and top-5 for a given meta-test task in the *scientific* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model always selects the pre-trained model that was trained on the same source data as the meta-test task.

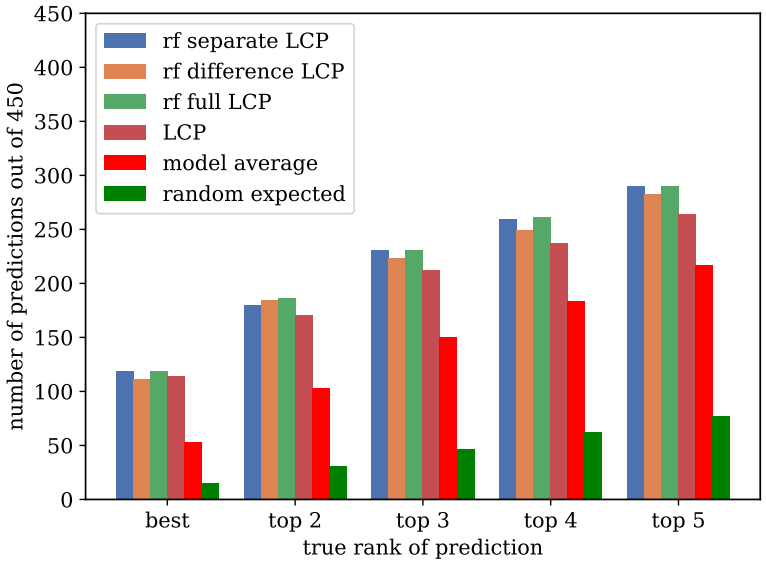

Figure 23: Bar chart showing how often different meta-models and baseline meta-models selected a pre-trained model that belongs to the top-1 (best), top-2, top-3, top-4, and top-5 for a given meta-test task in the *realistic* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model as included in Figure 18 is excluded from this figure, as the *realistc* context does not allow selection of a pre-trained model that was trained on the same source data as the given meta-test task.

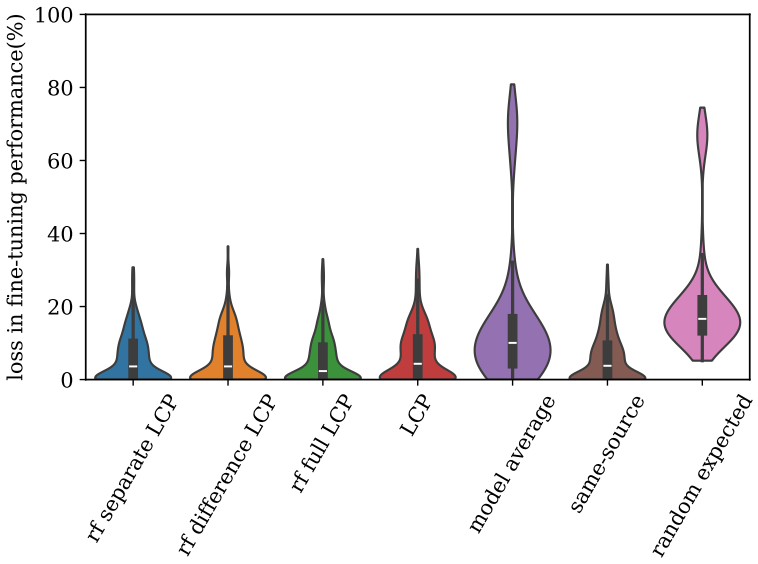

Figure 24: Violin plot with embedded box plots showing the *selection loss* of different meta-models and baseline meta-models in the *scientific* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model always selects the pre-trained model that was trained on the same source data as the meta-test task.

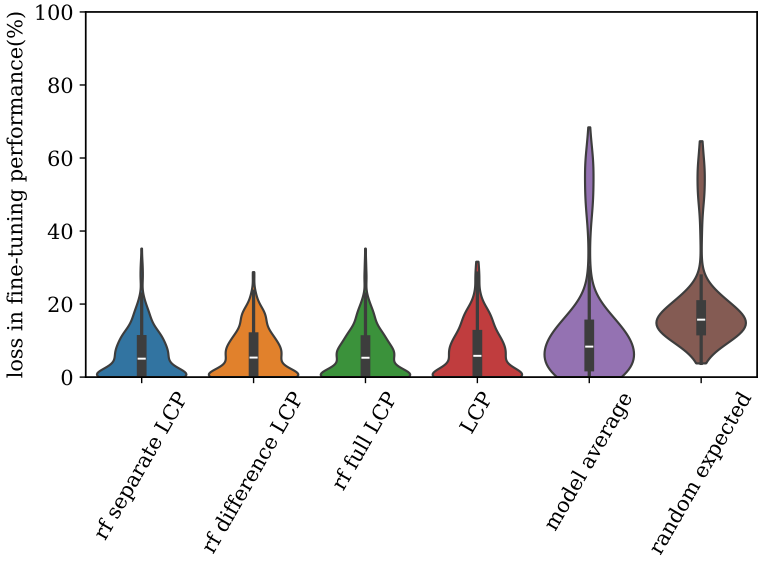

Figure 25: Violin plot with embedded box plots showing the *selection loss* of different meta-models and baseline meta-models in the *realistic* context. Both *rf separate* and *rf difference* correspond to random forest meta-models that were trained on different feature sets. The *model average* baseline uses the average performance of a pre-trained model on the training data as the prediction for the fine-tuning performance. The *random expected* baseline model shows the expected number of predictions in a given rank group when selecting at random. The *same-source* baseline model always selects the pre-trained model that was trained on the same source data as the meta-test task.

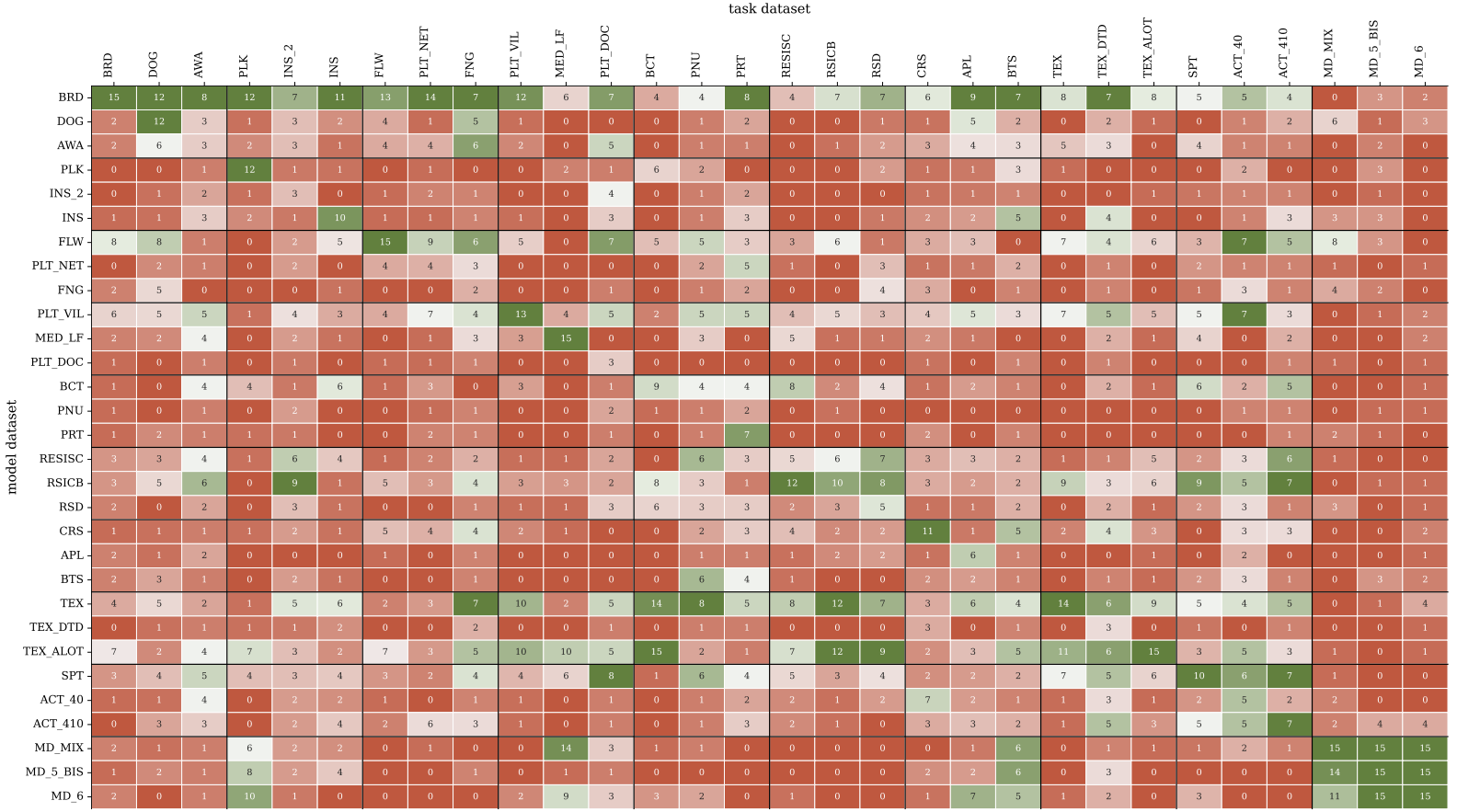

Figure 26: The transfer matrix with rank counts as predicted by the meta-model that was trained on the *rf full LCP* feature set. Each cell shows how often a pre-trained model from a given row is predicted to be part of the top-5 models for the tasks of a given column. of the transfer validation accuracy of the pre-trained models originating from the dataset of its row label fine-tuned to tasks from the dataset of its column label. The accented lines separate the different dataset domains.

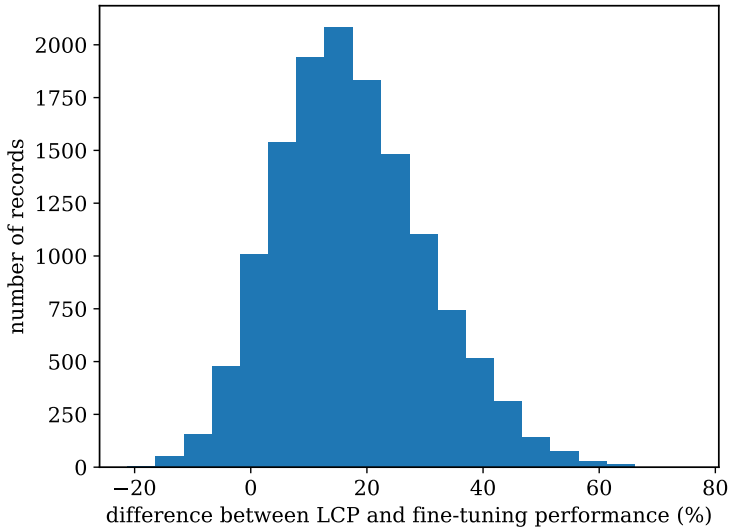

Figure 27: Histogram of the difference between the final fine-tuning performance and the low-cost proxy (fine-tuning performance after 1 epoch of fine-tuning).

7 Conclusion and future work

The large amount of high-quality training data that is needed to properly train a deep learning system forms an obstacle to its application in domains where data is sparse. This problem has led to the idea of employing auxiliary data to support the learning process, which is applied in the research fields of deep metalearning and deep transfer learning. Both research fields bring forth successful methods to train well-performing deep learning systems with auxiliary data and a small amount of training data for the target task. However, they also rely on the condition that the auxiliary data is related to the target task. Currently, no proven method exists for selecting the right auxiliary data for a given deep learning task, making it difficult to use scientific solutions in real-life applications. This has inspired us to investigate dataset relatedness in the context of deep metalearning and deep transfer learning.

Specifically, we were interested to know how well pre-trained image classification models from a given dataset can be fine-tuned to tasks from another dataset depending on the relationship between these datasets. Using the recently published Meta-Album dataset, we were able to test this for 30 different datasets, that originate from 10 different domains. The structure of Meta-Album then allowed us to formulate and compare three different dataset relationships: **same-source**, **same-domain**, and **cross-domain**.

Our initial research question regarded the same-source dataset relationship. Since image datasets generally contain classes that are conceptually related, we intuitively expect the classes to also be related in a computer-vision context. We wanted to test this intuition and asked

> *Can we expect pre-trained models to transfer well to tasks from the same source dataset via fine-tuning?*

In terms of absolute achieved accuracy on the target task, we found that we cannot expect this. Half of the fine-tuning performances that we recorded for the same-source category are less than 60% and a quarter of the performances are less than 45%. We have also observed that pre-trained models regularly achieved a fine-tuning performance that is comparable to random guessing. At the same time, we found that a quarter of the recorded performances are more than 85%. Both extremes form a bimodality in the distribution of fine-tuning performances, which is not observed in the distributions of the same-domain and cross-domain categories. Therefore, we conclude that in some cases, fine-tuning a pre-trained model that was pre-trained on the same source dataset as the task is advantageous. It remains unknown why this does not apply to all records that belong to the same-source category.

In terms of relative performance, the same-source category outperforms the other categories. In 66% of cases, the pre-trained model from the same-source category belonged to the top-5 performing models out of 30 for a given task and in more than 33% cases the same-source model performed best. This means that when same-source instances have a low absolute performance, instances from the other categories in general have a similar or lower performance. We have found datasets that bring forth exclusively easy tasks, where the fine-tuning performance is high, regardless of which pre-trained model is employed. Similarly, we have found datasets that bring forth exclusively hard tasks, where all pre-trained models show weak performance. What makes datasets exclusively easy or hard in this fine-tuning context is to be discovered in future work.

In a similar style, we wanted to know if the dataset domain relationship was reflected in the fine-tuning performance. As datasets that belong to the same domain are usually conceptually more related than datasets from different domains, we expected them to be more related in a fine-tuning context as well. Based on this intuition, we asked

> *Do pre-trained models transfer better within their domain than cross-domain?*

We have found no decisive evidence to conclude this. Of all the domains that were featured in our experiments, we have only found strong same-domain versus cross-domain performance differences for the Optical Character Recognition (OCR) domain. For other domains, there were either no or unclear

indications to suspect an advantage with respect to the cross-domain category. This is reflected in the distributions of fine-tuning performances, where we observed no notable difference between the same-domain and cross-domain categories.

Finally, our third research question involves the idea of selecting the best pre-trained model for a given task, using only pixel-based information about the datasets of the pre-trained model and the task. Specifically, we asked

> *Given an image classification task and a collection of pre-trained classification models, can we select the best pre-trained model using pixel-based meta-features of the source datasets of the model and task?*

To answer this question, we created different feature sets that are based on the statistical properties of the raw data of the image datasets. Using the feature set, we trained various statistical meta-models. Unfortunately, we were unable to successfully select the top-performing models for a given task using meta-models that base their predictions exclusively on pixel-based meta-features of the involved datasets. Although our best pixel-based meta-model performed better than random selection, we could only identify the best-performing pre-trained model in 7% of cases. We were also unable to outperform the model average baseline, which selects a pre-trained model for a task based on its average performance on all other tasks. To put our rank-based performance measure in perspective, we defined a *selection loss* metric that captures how much worse the selected pre-trained is with respect to the best-performing pre-trained model. In terms of this selection loss, the performance of our meta-model is comparable to the performance of the model average baseline, and half of the selections of our meta-model lose out on approximately 25% to 5% fine-tuning performance.

To improve the performance of our meta-models, we introduced an additional *low-cost proxy* feature, which we defined as the validation accuracy after one epoch of fine-tuning. Using this additional feature, we were able to greatly improve the performance of our meta-models, where the best one was able to identify the best-performing pre-trained model in more than a third of the cases. In addition, our best meta-model performed better than the same-source baseline, where the selected model always originates from the same source dataset as the given task. The pixel-based meta-features proved to be of added benefit, as our best meta-model also improves on the low-cost proxy baseline, where the pre-trained model with the highest low-cost proxy value is selected. We conclude that the validation accuracy after one epoch of fine-tuning has great predictive power and that it can enable statistical meta-models to effectively select the right pre-trained model for a given fine-tuning task.

Many questions have also remained unsolved, which we will state here for inspiration for future work. First, we were unable to find an explanation for why there are two local peaks in the distribution of fine-tuning performances for the same-source category. We wonder what causes some pre-trained models to transfer exceptionally well to tasks from the same source dataset and why some pre-trained models do not have this apparent advantage in this context. Second, we have found datasets that produce tasks to which no pre-trained model could be fine-tuned with good performance. Similarly, we have found datasets for which the opposite is true. We currently have no explanation of what causes some datasets to produce exclusively hard or easy tasks in the context of our experiments. A third remarkable observation is the apparent cohesion within the optical character recognition domain, which is not observed in other domains. It remains unclear why only this domain has this feature. A fourth question that remains unanswered is what causes pre-trained models to sometimes perform worse than random guessing when they are fine-tuned to tasks from other domains. Finally, we see potential in the idea of selecting auxiliary data on a class level or even instance level using the pixel-based features that were used to train the meta-models. In this way, pre-training datasets can be formed from a larger data collection to specifically suit the intended target task.

Acknowledgements

This work was performed using the ALICE compute resources provided by Leiden University.

References

[1] Chen Sun, Abhinav Shrivastava, Saurabh Singh, and Abhinav Gupta. Revisiting unreasonable effectiveness of data in deep learning era. In *IEEE International Conference on Computer Vision, ICCV 2017*, pages 843–852. IEEE Computer Society, 2017.

[2] Mike Huisman, Jan N. van Rijn, and Aske Plaat. A survey of deep meta-learning. *Artificial Intelligence Review*, 54(6):4483–4541, 2021.

[3] Ricardo Ribani and Maurício Marengoni. A survey of transfer learning for convolutional neural networks. In *32nd SIBGRAPI Conference on Graphics, Patterns and Images Tutorials*, pages 47–57. IEEE, 2019.

[4] Rahul Ramesh and Pratik Chaudhari. Model zoo: A growing brain that learns continually. In *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net, 2022.

[5] Konstantin Schürholt, Diyar Taskiran, Boris Knyazev, Xavier Giró-i-Nieto, and Damian Borth. Model zoos: A dataset of diverse populations of neural network models. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

[6] Kaichao You, Zhi Kou, Mingsheng Long, and Jianmin Wang. Co-tuning for transfer learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*, 2020.

[7] Bingyan Liu, Yifeng Cai, Yao Guo, and Xiangqun Chen. Transtailor: Pruning the pre-trained model for improved transfer learning. In *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021, Thirty-Third Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The Eleventh Symposium on Educational Advances in Artificial Intelligence, EAAI 2021*, pages 8627–8634. AAAI Press, 2021.

[8] Ihsan Ullah, Dustin Carrión-Ojeda, Sergio Escalera, Isabelle Guyon, Mike Huisman, Felix Mohr, Jan N. van Rijn, Haozhe Sun, Joaquin Vanschoren, and Phan Anh Vu. Meta-album: Multi-domain meta-dataset for few-shot image classification. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022*, 2022.

[9] Yann LeCun, Yoshua Bengio, and Geoffrey E. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000*, pages 402–408. MIT Press, 2000.

[11] Léon Bottou and Olivier Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems, NIPS 2007*, pages 161–168. Curran Associates, Inc., 2007.

[12] Ilya Sutskever, James Martens, George E. Dahl, and Geoffrey E. Hinton. On the importance of initialization and momentum in deep learning. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013*, volume 28 of *JMLR Workshop and Conference Proceedings*, pages 1139–1147. JMLR.org, 2013.

[13] Douglas M. Hawkins. The problem of overfitting. *Journal of Chemical Information and Modeling*, 44(1):1–12, 2004.

[14] Yingjie Tian and Yuqi Zhang. A comprehensive survey on regularization strategies in machine learning. *Information Fusion*, 80:146–166, 2022.

[15] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[16] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

[17] Dmytro Mishkin and Jiri Matas. All you need is a good init. In *4th International Conference on Learning Representations, ICLR 2016*, 2016.

[18] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, pages 4077–4087, 2017.

[19] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015*, volume 37, 2015.

[20] Oriol Vinyals, Charles Blundell, Tim Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, pages 3630–3638, 2016.

[21] Adam Santoro, Sergey Bartunov, Matthew M. Botvinick, Daan Wierstra, and Timothy P. Lillicrap. Meta-learning with memory-augmented neural networks. In *Proceedings of the 33nd International Conference on Machine Learning, ICML 2016*, volume 48 of *JMLR Workshop and Conference Proceedings*, pages 1842–1850. JMLR.org, 2016.

[22] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 2554–2563. PMLR, 2017.

[23] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 2017.

[24] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

[25] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017*. OpenReview.net, 2017.

[26] Ming-Wei Chang, Lev-Arie Ratinov, Dan Roth, and Vivek Srikumar. Importance of semantic representation: Dataless classification. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 830–835. AAAI Press, 2008.

[27] Hugo Larochelle, Dumitru Erhan, and Yoshua Bengio. Zero-data learning of new tasks. In *Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, AAAI 2008*, pages 646–651. AAAI Press, 2008.

[28] Li Fei-Fei, Robert Fergus, and Pietro Perona. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(4):594–611, 2006.

[29] Haozhe Sun, Wei-Wei Tu, and Isabelle M Guyon. Omniprint: A configurable printed character synthesizer. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

[30] Bartosz Zielinski, Anna Plichta, Krzysztof Misztal, Przemyslaw Spurek, Monika Brzychczy-Wloch, and Dorota Ochonska. Deep learning approach to bacterial colony classification. *PLOS ONE*, 12(9):1–14, 2017.

[31] Gong Cheng, Junwei Han, and Xiaoqiang Lu. Remote sensing image scene classification: Benchmark and state of the art. *CoRR*, abs/1703.00121, 2017. URL `http://arxiv.org/abs/1703.00121`.

[32] Wei-Hong Li, Xialei Liu, and Hakan Bilen. Cross-domain few-shot learning with task-specific adapters. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022*, pages 7151–7160. IEEE, 2022.

[33] Pan Li, Shaogang Gong, Chengjie Wang, and Yanwei Fu. Ranking distance calibration for cross-domain few-shot learning. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022*, pages 9089–9098. IEEE, 2022.

[34] Simon Kornblith, Jonathon Shlens, and Quoc V. Le. Do better imagenet models transfer better? In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, pages 2661–2671. Computer Vision Foundation / IEEE, 2019.

[35] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014*, volume 32 of *JMLR Workshop and Conference Proceedings*, pages 647–655. JMLR.org, 2014.

[36] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014*, pages 512–519. IEEE Computer Society, 2014.

[37] Edwin V. Bonilla, Kian Ming Adam Chai, and Christopher K. I. Williams. Multi-task gaussian process prediction. In *Advances in Neural Information Processing Systems 20, Proceedings of the Twenty-First Annual Conference on Neural Information Processing Systems*, pages 153–160. Curran Associates, Inc., 2007.

[38] Jing Gao, Wei Fan, Jing Jiang, and Jiawei Han. Knowledge transfer via multiple model local structure mapping. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 283–291. ACM, 2008.

[39] Edna Chebet Too, Li Yujian, Sam Njuki, and Liu Yingchun. A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, 161:272–279, 2019.

[40] Heechul Jung, Sihaeng Lee, Junho Yim, Sunjeong Park, and Junmo Kim. Joint fine-tuning in deep neural networks for facial expression recognition. In *2015 IEEE International Conference on Computer Vision, ICCV 2015*, pages 2983–2991. IEEE Computer Society, 2015.

[41] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott E. Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2015*, pages 1–9. IEEE Computer Society, 2015.

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

[43] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives. *Evolutionary Computation*, 27(1):3–45, 2019.

[44] Eugene Nudelman, Kevin Leyton-Brown, Holger H. Hoos, Alex Devkar, and Yoav Shoham. Understanding random SAT: beyond the clauses-to-variables ratio. In Mark Wallace, editor, *Principles and Practice of Constraint Programming, CP 2004*, volume 3258 of *Lecture Notes in Computer Science*, pages 438–452. Springer, 2004.

[45] Frank Hutter, Lin Xu, Holger H. Hoos, and Kevin Leyton-Brown. Algorithm runtime prediction: Methods & evaluation. *Artificial Intelligence*, 206:79–111, 2014.

[46] Marco Maratea, Luca Pulina, and Francesco Ricca. Applying machine learning techniques to ASP solving. In *Technical Communications of the 28th International Conference on Logic Programming, ICLP 2012*, volume 17 of *LIPIcs*, pages 37–48. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2012.

[47] Kate Smith-Miles and Jano I. van Hemert. Discovering the suitability of optimisation algorithms by learning from evolved instances. *Annals of Mathematics and Artificial Intelligence*, 61(2):87–104, 2011.

[48] George Boukeas, Constantinos Halatsis, Vassilis Zissimopoulos, and Panagiotis Stamatopoulos. Measures of intrinsic hardness for constraint satisfaction problem instances. In *SOFSEM 2004: Theory and Practice of Computer Science, 30th Conference on Current Trends in Theory and Practice of Computer Science*, volume 2932 of *Lecture Notes in Computer Science*, pages 184–195. Springer, 2004.

[49] Gerald Piosenka. Birds 400 - species image classification. Kaggle, 2020. URL `https://www.kaggle.com/datasets/gpiosenka/100-bird-species`.

[50] Aditya Khosla, Nityananda Jayadevaprakash, Bangpeng Yao, and Li Fei-Fei. Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[51] Yongqin Xian, Christoph H. Lampert, Bernt Schiele, and Zeynep Akata. Zero-shot learning - A comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 41(9):2251–2265, 2019.

[52] Emily F. Brownlee Heidi M. Sosik, Emily E. Peacock. Annotated plankton images - data set for developing and evaluating classification methods., 2015.

[53] Xiaoping Wu, Chi Zhan, Yu-Kun Lai, Ming-Ming Cheng, and Jufeng Yang. IP102: A large-scale benchmark dataset for insect pest recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019*, pages 8787–8796. Computer Vision Foundation / IEEE, 2019.

[54] Hortense Serret, Nicolas Deguines, Jang Yikweon, Gregoire Lois, and Romain Julliard. Data quality and participant engagement in citizen science: Comparing two approaches for monitoring pollinators in france and south korea. *Citizen Science: Theory and Practice*, 4:22, 2019.

[55] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Sixth Indian Conference on Computer Vision, Graphics & Image Processing, ICVGIP 2008*, pages 722–729. IEEE Computer Society, 2008.

[56] Camille Garcin, Alexis Joly, Pierre Bonnet, Antoine Affouard, Jean-Christophe Lombardo, Mathias Chouet, Maximilien Servajean, Titouan Lorieul, and Joseph Salmon. Pl@ntnet-300k: a plant image dataset with high label ambiguity and a long-tailed distribution. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021*, 2021.

[57] Lukás Picek, Milan Sulc, Jirí Matas, Thomas S. Jeppesen, Jacob Heilmann-Clausen, Thomas Læssøe, and Tobias Frøslev. Danish fungi 2020 - not just another image recognition dataset. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022*, pages 3281–3291. IEEE, 2022.

[58] Geetharamani G. and Arun Pandian J. Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers & Electrical Engineering*, 76:323–338, 2019.

[59] Roopashree S and Anitha J. Medicinal leaf dataset. 2020.

[60] Davinder Singh, Naman Jain, Pranjali Jain, Pratik Kayal, Sudhakar Kumawat, and Nipun Batra. Plantdoc: A dataset for visual plant disease detection. In *Proceedings of the 7th ACM IKDD CoDS and 25th COMAD*, CoDS COMAD 2020, pages 249–253. Association for Computing Machinery, 2020.

[61] Jevgenij Gamper, Navid Alemi Koohbanani, Ksenija Benet, Ali Khuram, and Nasir Rajpoot. Pannuke: an open pan-cancer histology dataset for nuclei instance segmentation and classification. In *European Congress on Digital Pathology*, pages 11–19. Springer, 2019.

[62] Jevgenij Gamper, Navid Alemi Koohbanani, Simon Graham, Mostafa Jahanifar, Syed Ali Khurram, Ayesha Azam, Katherine Hewitt, and Nasir Rajpoot. Pannuke dataset extension, insights and baselines. *arXiv preprint arXiv:2003.10778*, 2020.

[63] Peter J Thul, Lovisa Akesson, Mikaela Wiking, Diana Mahdessian, Aikaterini Geladaki, Hammou Ait Blal, Tove Alm, Anna Asplund, Lars Bjork, and Lisa M Breckels. A subcellular map of the human proteome. *Science*, 356(6340), 2017.

[64] Haifeng Li, Xin Dou, Chao Tao, Zhixiang Wu, Jie Chen, Jian Peng, Min Deng, and Ling Zhao. Rsi-cb: A large-scale remote sensing image classification benchmark using crowdsourced data. *Sensors*, 20(6):1594, 2020.

[65] Yang Long, Yiping Gong, Zhifeng Xiao, and Qing Liu. Accurate object localization in remote sensing images based on convolutional neural networks. *IEEE Transactions on Geoscience and Remote Sensing*, 55(5):2486–2498, 2017.

[66] Zhifeng Xiao, Yang Long, Deren Li, Chunshan Wei, Gefu Tang, and Junyi Liu. High-resolution remote sensing image retrieval based on cnns from a dimensional perspective. *Remote Sensing*, 9(7):725, 2017.

[67] Jonathan Krause, Michael Stark, Jia Deng, and Li Fei-Fei. 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*, 2013.

[68] Zhize Wu. Muti-type Aircraft of Remote Sensing Images: MTARSI, 2019.

[69] Erhan Gundogdu, Berkan Solmaz, Veysel Yucesoy, and Aykut Koc. Marvel: A large-scale image dataset for maritime vessels. In Shang-Hong Lai, Vincent Lepetit, Ko Nishino, and Yoichi Sato, editors, *Computer Vision – ACCV 2016*, pages 165–180. Springer International Publishing, 2017.

[70] Mario Fritz, E. Hayman, B. Caputo, and J. Eklundh. The kth-tips database. URL `https://www.csc.kth.se/cvap/databases/kth-tips/index.html`.

[71] P. Mallikarjuna, Alireza Tavakoli Targhi, Mario Fritz, E. Hayman, B. Caputo, and J. Eklundh. The kth-tips 2 database. 2006. URL `https://www.csc.kth.se/cvap/databases/kth-tips/index.html`.

[72] Gustaf Kylberg. The kylberg texture dataset v. 1.0. External report (Blue series) 35, 2011.

[73] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. A sparse texture representation using local affine regions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8): 1265–1278, 2005.

[74] M. Cimpoi, S. Maji, I. Kokkinos, S. Mohamed, and A. Vedaldi. Describing textures in the wild. In *Proceedings of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2014.

[75] Gertjan J. Burghouts and Jan-Mark Geusebroek. Material-specific adaptation of color invariant features. *Pattern Recognition Letters*, 30(3):306–313, 2009.

[76] Gerald Piosenka. 100 sports image classification. URL `https://www.kaggle.com/datasets/gpiosenka/sports-classification`.

[77] Bangpeng Yao, Xiaoye Jiang, Aditya Khosla, Andy Lai Lin, Leonidas Guibas, and Li Fei-Fei. Human action recognition by learning bases of action attributes and parts. In *2011 International Conference on Computer Vision, ICCV 2011*, pages 1331–1338, 2011.

[78] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3686–3693, 2014.

[79] John W Tukey et al. *Exploratory data analysis*, volume 2. Reading, MA, 1977.

[80] Jerry L Hintze and Ray D Nelson. Violin plots: a box plot-density trace synergism. *The American Statistician*, 52(2):181–184, 1998.

[81] David W Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[82] Nikita Dvornik, Cordelia Schmid, and Julien Mairal. Selecting relevant features from a multi-domain representation for few-shot classification. In *Computer Vision - ECCV 2020 - Part X*, volume 12355 of *Lecture Notes in Computer Science*, pages 769–786. Springer, 2020.

[83] Lu Liu, William L. Hamilton, Guodong Long, Jing Jiang, and Hugo Larochelle. A universal representation transformer layer for few-shot image classification. In *9th International Conference on Learning Representations, ICLR 2021*, 2021.

[84] Huaxiu Yao, Yu Wang, Ying Wei, Peilin Zhao, Mehrdad Mahdavi, Defu Lian, and Chelsea Finn. Meta-learning with an adaptive task scheduler. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021*, pages 7497–7509, 2021.

[85] Eleni Triantafillou, Hugo Larochelle, Richard S. Zemel, and Vincent Dumoulin. Learning a universal template for few-shot dataset generalization. In *Proceedings of the 38th International Conference on Machine Learning, ICML 2021*, volume 139 of *Proceedings of Machine Learning Research*, pages 10424–10433. PMLR, 2021.

[86] Sebastian Pineda-Arango, Fabio Ferreira, Arlind Kadra, Frank Hutter, and Josif Grabocka. Quick-tune: Quickly learning which pretrained model to finetune and how. *CoRR*, abs/2306.03828, 2023.

[87] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[88] David Hasler and Sabine Süsstrunk. Measuring colorfulness in natural images. In *Human Vision and Electronic Imaging VIII*, volume 5007 of *SPIE Proceedings*, pages 87–95. SPIE, 2003.