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

Matt van den Nieuwenhuijzen, Carola Doerr, Jan N van Rijn, Henry Gouk

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

Matt van den Nieuwenhuijzen[1] **Carola Doerr**[2] **Henry Gouk**[3] **Jan N. van Rijn**[1]

[1]Leiden Institute of Advanced Computer Science, Leiden University, the Netherlands
[2]Sorbonne Université, CNRS, LIP6, France
[3]School of Informatics, University of Edinburgh, United Kingdom

Abstract When applying a neural network to address a new learning problem, rather than training a network from scratch, it is common practice to utilise a network pre-trained on a related dataset and then fine-tune it to the data of the target task. This poses the question: which pre-trained network should be selected? This work investigates this problem in the context of three different dataset relationships: same-source, same-domain, and cross-domain. We utilise Meta-Album, which offers an extensive collection of datasets from various unrelated domains. We first split each of the 30 datasets of Meta-Album into a meta-train dataset and meta-test dataset, then create pre-trained models for each meta-train dataset, and finally compare the performances of the pre-trained models in a fine-tuning context when applied to meta-test tasks. We categorise the performances into the three dataset relationship groups and find that the same-source category performs best in terms of accuracy. Then, using meta-features calculated on the meta-train dataset and meta-test tasks, we train statistical meta-models that are employed to select the best pre-trained model for a given meta-test task. Our best meta-model identifies the best-performing model in ~ 25% of the cases. It improves upon a baseline that selects the best average model by more than 30%.

1 Introduction

Solving learning problems with neural networks commonly requires large amounts of data and resources. For this reason, a popular approach to solving new learning problems is to select a neural network that was pre-trained on related data and fine-tune it to the new task instead of training from scratch. This way, fewer data and computational resources are required to achieve good results. 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 [35, 37]. This can, for example, be achieved by adapting pre-trained models using regularisation methods specifically designed for the fine-tuning setting [51, 25, 13, 19]. 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 most appropriate auxiliary data for a deep metalearning or deep transfer learning problem.

The key question guiding our research is: *Given an image classification task and a collection of pre-trained classification models, can we select the best one using a set of meta-features describing the data on which this model was pre-trained?* To answer this question, we employ the recently published Meta-Album [43], which currently bundles 30 image datasets from 10 different domains. The diversity of Meta-Album allows us to test how meta-train and meta-test datasets relate to each other in a fine-tuning context for three different dataset relationships: **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 the performance of different pre-trained models when 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 basic pixel-based statistics of the datasets and the validation accuracy after one epoch of fine-tuning as meta-features.

Related work. The following works have approached a slightly related problem using a deep metalearning approach [18], in the sense that they train one large model encompassing all data. (1) SUR (*Selecting from Universal Representations*) by Dvornik et al. [6], where the feature representations of different pre-trained models are selected and linearly combined to solve unseen tasks and its expansion to a multi-headed URT (Universal Representation Transform) by Liu et al. [26]. (2) Quick-Tune by Pineda-Arango et al. [32], where Bayesian optimization is used to find the best candidate for a fine-tuning job, given the fine-tuning validation accuracy. (3) ATS (*Adaptive Task Scheduler*) by Yao et al. [50], where meta-learning training tasks are selected based on their expected contribution to the learner instead of random sampling. Contrary to those works, our work takes a more modular approach. As such, we compare our work to standard baselines from the algorithm selection literature [2, 40]. This work is a compressed version based on the work done in a master's thesis [44].

2 Experimental Methodology

Our methods consist of five consecutive stages, further illustrated in the appendix.

1. Data preparation. For 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 uniformly split each Meta-Album dataset into a meta-train and meta-test set. This split is based on classes (some belong to the meta-train set, others to the meta-test set). This 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 uniform class-split is split further on a class level into different N-way k-shot classification tasks (with $N = 5$, $k = 32$). This task split is also performed uniformly using seed t. See Figure 4 for more details.

2. Pre-training models. We use the meta-train datasets from the previous stage to create pre-trained classification models. 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. We perform this uniform split based on a fixed seed. For each training epoch, we record the loss and accuracy of the model on the support and query set. After the model is trained, we store it to be used later in the fine-tuning stage of our method. See Figure 5 for more details.

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 (generated using the same seed s to ensure class separation). Fine-tuning all pre-trained models on all datasets allows us to investigate which pre-trained models transfer well to which tasks. The data of each task (meta-test set) 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. We do this for 50 epochs. Note that the support set contains only 160 images, so that an epoch is relatively fast. We record the loss and accuracy on the support and query set for each epoch of fine-tuning. We measure the mean accuracy of the last 10 epochs to express fine-tuning performance, which is later used to train the meta-models. See Figure 6 for more details.

4. Meta-model training. This step provides a methodology that, given a target dataset, selects a proper pre-trained model to fine-tune. We create a meta-dataset, where each record contains meta-features describing the dataset the network was trained on (pre-training data) and meta-features

Figure 1: Matrix showing how the different datasets of Meta-Album relate to each other in a fine-tuning context. 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 coloured relatively to other cells in the same column, where green corresponds to a higher relative value and red to a lower relative value.

describing the task it was fine-tuned to. The target value is the network's validation accuracy on the task to which it was fine-tuned. An overview of all the different meta-features [2, 40] that we use in this work is shown in Table 3 (more details about the used features can be found in Section 3). 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, which are used to train and validate the meta-models. See Figure 7 for more details.

5. **Meta-model evaluation**. The different meta-models and baseline models are evaluated on their ability to select the most suitable pre-trained model for a given meta-test task. We apply the meta-models to the test split of our meta-dataset and collect their predictions. Since the meta-models predict validation accuracy on the task it was fine-tuned to, we transform the predictions into a ranked representation, ranking the pre-trained models in order of predicted performance. 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 the meta-model, we report the actual rank of the selected model and calculate the *selection loss*, which is the difference between the performance of the selected pre-trained model and the performance of the best-performing pre-trained model on that dataset. See Figure 8 for more details.

3 Selecting Pre-trained Models

We first analyze the general results of the various pre-trained models fine-tuned to various tasks. Figure 1 shows a matrix where the performance of the different pre-trained models on all different meta-test datasets is visualized. The results on the diagonal represent the tasks that were fine-tuned to a task that comes from the same source. As the diagonal is generally green, this indicates that

these models perform better than models that were pre-trained on other datasets. The fine-tuning jobs belong to the top-5 models in 298 out of the 450 cases (66%). We also show the distribution of our meta-data on the left of Figure 2.

Next, we train different meta-models to predict fine-tuning performance based on different feature sets. The meta-models take the meta-features corresponding to the meta-test task and the available pre-trained models as input and predict the performance of this pre-trained model on this task. This prediction can be used to select a pre-trained model for a given dataset, and this is how we evaluate these meta-models. To ensure a realistic setting, we exclude the same-source pre-trained model to be selected by the meta-model, corresponding to the real-world setting where a model pre-trained on the same data generally does not exist. Additionally, we show various baseline models.

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. The meta-model is trained to predict 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 predicted top-ranking combination is selected. When multiple combinations are given the same top-ranking prediction (i.e. the model has a tie between various options), the pre-trained model with the actual lowest fine-tuning performance is selected to mitigate potential randomness in the evaluation. This is to avoid an overly optimistic reported performance. We use the so-called *selection loss* to evaluate the predictions of the meta-models and the baselines. We define the selection loss as the difference between the selected pre-trained model's actual fine-tuning performance and the best pre-trained model; therefore, lower values indicate better performance. This procedure is repeated for all meta-test datasets, resulting in a leave-one-out (the chosen meta-test dataset) cross-validation setting.

Our initial meta-models were trained on feature sets that include only statistical measures of the raw data (see the upper section of Table 3). Although these meta-models performed better than the random selection baseline, their performances were underwhelming, with only a $\sim$ 7% accuracy in selecting the best model out of 30. This motivated us to add a low-cost proxy (LCP) to our feature sets. We define the LCP as the validation fine-tuning accuracy after one epoch of fine-tuning.

We compare the results for our meta-models with three baseline selection methods. 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 1, and using the average value as a prediction for the excluded cell. The *random expected* baseline model visualizes the expected outcome of a model that randomly selects a pre-trained model for a given task. 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.

Results. In Figure 3, we show how often the different meta-models selected a pre-trained model in the best, top-2, etc. category for the given tasks. On the right of Figure 2, we compare our best initial (pixel-based) meta-model *meta-model*, our best meta-model using the LCP feature *meta-model + LCP*, and the baseline meta-models in their ability to minimize the loss with respect to the best performing (available) pre-trained model. In Figure 2, we compare the distributions of *selection loss* of the different pre-trained models and baselines.

In Figure 3, we observe that the *meta-model + LCP* meta-model correctly identifies the best-performing pre-trained model in $\sim$ 25% of cases. In terms of *selection loss*, all our meta-models could select models that never miss out on more than 35% fine-tuning performance, and half of their selections have a selection loss of $\sim$ 7% or less.

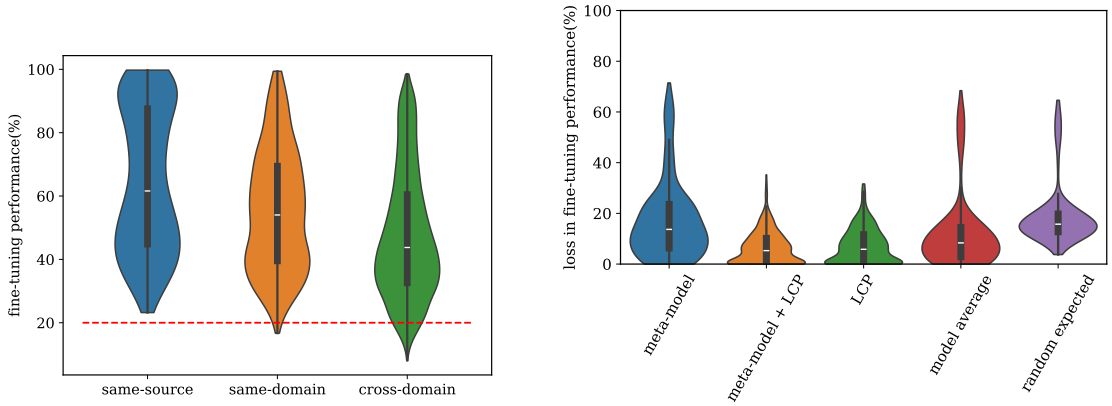

Figure 2: (left) Violin plots with embedded box plots of the fine-tuning performance for records that belong to the **same-source** (diagonal), **same-domain** and **cross-domain** dataset relationships. The dashed red line indicates the baseline fine-tuning performance at 20%. (right) Violin plot with embedded box plots showing the *selection loss* of different meta-models and baseline methods.

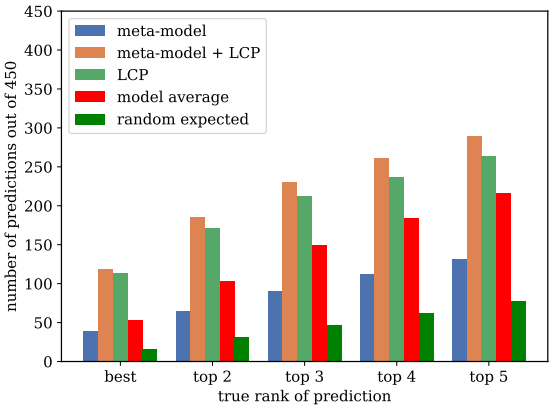

Figure 3: Bar chart showing how often the different meta-models selected a pre-trained model that was in the top-1 (best), top-2, etc., for the given meta-test task.

4 Discussion

We investigated dataset relatedness in a fine-tuning context. We confirmed that pre-trained models perform best when they are fine-tuned to unseen tasks from the same source dataset as their original training data, compared to their performance on tasks from the same domain or data from completely different domains. This result implies the importance of validating deep transfer learning and deep metalearning methods using multiple data sources, as validation with partitions of a large, singular dataset does not resemble real-world applications and can return biased results.

Subsequently, we trained meta-models that predict fine-tuning performance for a given pre-train dataset and task, using pixel-based features of the aforementioned datasets and the validation accuracy after one epoch of fine-tuning. Our meta-models could identify the best performing pre-trained model in $\sim$ 25% of cases, improving upon the baseline that selects the average best pre-trained model. Interestingly, the LCP feature (being a model trained on 1 epoch) in isolation was also a good indicator, suggesting further research towards aggressive early discarding methods [7].

Broader Impact Statement.

After careful reflection, the authors have determined that this work presents no notable negative impacts to society or the environment.

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

References

[1] Andriluka, M., Pishchulin, L., Gehler, P., and Schiele, B. (2014). 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.

[2] Brazdil, P., van Rijn, J. N., Soares, C., and Vanschoren, J. (2022). *Metalearning: Applications to Automated Machine Learning and Data Mining*. Springer, 2nd edition.

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

[4] Cheng, G., Han, J., and Lu, X. (2017). Remote sensing image scene classification: Benchmark and state of the art. *CoRR*, abs/1703.00121.

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

[6] Dvornik, N., Schmid, C., and Mairal, J. (2020). 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.

[7] Egele, R., Mohr, F., Viering, T. J., and Balaprakash, P. (2024). The unreasonable effectiveness of early discarding after one epoch in neural network hyperparameter optimization. *Neurocomputing*, 597:127964.

[8] Fritz, M., Hayman, E., Caputo, B., and Eklundh, J. (2006). The KTH-TIPS database. `https://www.csc.kth.se/cvap/databases/kth-tips/index.html`.

[9] G., G. and J., A. P. (2019). Identification of plant leaf diseases using a nine-layer deep convolutional neural network. *Computers & Electrical Engineering*, 76:323–338.

[10] Gamper, J., Koohbanani, N. A., Benet, K., Khuram, A., and Rajpoot, N. (2019). Pannuke: an open pan-cancer histology dataset for nuclei instance segmentation and classification. In *European Congress on Digital Pathology*, pages 11–19. Springer.

[11] Gamper, J., Koohbanani, N. A., Graham, S., Jahanifar, M., Khurram, S. A., Azam, A., Hewitt, K., and Rajpoot, N. (2020). Pannuke dataset extension, insights and baselines. *arXiv preprint arXiv:2003.10778*.

[12] Garcin, C., Joly, A., Bonnet, P., Affouard, A., Lombardo, J., Chouet, M., Servajean, M., Lorieul, T., and Salmon, J. (2021). 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*.

[13] Gouk, H., Hospedales, T., and Pontil, M. (2021). Distance-based regularisation of deep networks for fine-tuning. In *9th International Conference on Learning Representations, ICLR 2021*.

[14] Gundogdu, E., Solmaz, B., Yucesoy, V., and Koc, A. (2017). Marvel: A large-scale image dataset for maritime vessels. In *Computer Vision – ACCV 2016*, pages 165–180. Springer International Publishing.

[15] Hasler, D. and Süsstrunk, S. (2003). Measuring colorfulness in natural images. In *Human Vision and Electronic Imaging VIII*, volume 5007 of *SPIE Proceedings*, pages 87–95. SPIE.

[16] He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016*, pages 770–778. IEEE Computer Society.

[17] Heidi M. Sosik, Emily E. Peacock, E. F. B. (2015). Annotated plankton images - data set for developing and evaluating classification methods. `https://hdl.handle.net/10.1575/1912/7341`.

[18] Hospedales, T. M., Antoniou, A., Micaelli, P., and Storkey, A. J. (2022). Meta-learning in neural networks: A survey. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 44(9):5149–5169.

[19] Huisman, M., Plaat, A., and van Rijn, J. N. (2024). Understanding transfer learning and gradient-based meta-learning techniques. *Machine Learning*, 113(7):4113–4132.

[20] Khosla, A., Jayadevaprakash, N., Yao, B., and Fei-Fei, L. (2011). Novel dataset for fine-grained image categorization. In *First Workshop on Fine-Grained Visual Categorization, IEEE Conference on Computer Vision and Pattern Recognition*.

[21] Krause, J., Stark, M., Deng, J., and Fei-Fei, L. (2013). 3d object representations for fine-grained categorization. In *4th International IEEE Workshop on 3D Representation and Recognition (3dRR-13)*.

[22] Kylberg, G. (2011). The kylberg texture dataset v. 1.0. External report (Blue series) 35, Centre for Image Analysis, Swedish University of Agricultural Sciences and Uppsala University, Uppsala, Sweden.

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

[24] Li, H., Dou, X., Tao, C., Wu, Z., Chen, J., Peng, J., Deng, M., and Zhao, L. (2020). Rsi-cb: A large-scale remote sensing image classification benchmark using crowdsourced data. *Sensors*, 20(6):1594.

[25] Liu, B., Cai, Y., Guo, Y., and Chen, X. (2021a). 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.

[26] Liu, L., Hamilton, W. L., Long, G., Jiang, J., and Larochelle, H. (2021b). A universal representation transformer layer for few-shot image classification. In *9th International Conference on Learning Representations, ICLR 2021*.

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

[28] Mallikarjuna, P., Targhi, A. T., Fritz, M., Hayman, E., Caputo, B., and Eklundh, J. (2006). The KTH-TIPS 2 database. https://www.csc.kth.se/cvap/databases/kth-tips/index.html.

[29] Nilsback, M. and Zisserman, A. (2008). 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.

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

[31] Picek, L., Sulc, M., Matas, J., Jeppesen, T. S., Heilmann-Clausen, J., Læssøe, T., and Frøslev, T. (2022). 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.

[32] Pineda-Arango, S., Ferreira, F., Kadra, A., Hutter, F., and Grabocka, J. (2023). Quick-tune: Quickly learning which pretrained model to finetune and how. *CoRR*, abs/2306.03828.

[33] Piosenka, G. (2020). Birds 400 - species image classification. https://www.kaggle.com/datasets/gpiosenka/100-bird-species.

[34] Piosenka, G. (2023). 100 sports image classification. https://www.kaggle.com/datasets/gpiosenka/sports-classification.

[35] Ramesh, R. and Chaudhari, P. (2022). Model zoo: A growing brain that learns continually. In *The Tenth International Conference on Learning Representations, ICLR 2022*. OpenReview.net.

[36] Roopashree, S. and Anitha, J. (2020). Medicinal leaf dataset. https://data.mendeley.com/datasets/nnytj2v3n5/1.

[37] Schürholt, K., Taskiran, D., Knyazev, B., Giró-i-Nieto, X., and Borth, D. (2022). 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*.

[38] Serret, H., Deguines, N., Yikweon, J., Lois, G., and Julliard, R. (2019). 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.

[39] Singh, D., Jain, N., Jain, P., Kayal, P., Kumawat, S., and Batra, N. (2020). 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.

[40] Smith-Miles, K. A. (2008). Cross-disciplinary perspectives on meta-learning for algorithm selection. *ACM Computing Surveys (CSUR)*, 41(1):6:1–6:25.

[41] Sun, H., Tu, W.-W., and Guyon, I. M. (2021). Omniprint: A configurable printed character synthesizer. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021*.

[42] Thul, P. J., Akesson, L., Wiking, M., Mahdessian, D., Geladaki, A., Ait Blal, H., Alm, T., Asplund, A., Bjork, L., and Breckels, L. M. (2017). A subcellular map of the human proteome. *Science*, 356(6340).

[43] Ullah, I., Carrión-Ojeda, D., Escalera, S., Guyon, I., Huisman, M., Mohr, F., van Rijn, J. N., Sun, H., Vanschoren, J., and Vu, P. A. (2022). 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*.

[44] van den Nieuwenhuijzen, M. (2024). Selecting pre-trained models for transfer learning with data-centric meta-features. Master's thesis, https://theses.liacs.nl/2872.

[45] Wu, X., Zhan, C., Lai, Y., Cheng, M., and Yang, J. (2019). 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.

[46] Wu, Z. (2019). Muti-type Aircraft of Remote Sensing Images: MTARSI. https://zenodo.org/record/3464319#.Y0FAAlJByrc.

[47] Xian, Y., Lampert, C. H., Schiele, B., and Akata, Z. (2019). 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.

[48] Xiao, Z., Long, Y., Li, D., Wei, C., Tang, G., and Liu, J. (2017). High-resolution remote sensing image retrieval based on cnns from a dimensional perspective. *Remote Sensing*, 9(7):725.

[49] Yao, B., Jiang, X., Khosla, A., Lin, A. L., Guibas, L., and Fei-Fei, L. (2011). Human action recognition by learning bases of action attributes and parts. In *2011 International Conference on Computer Vision, ICCV 2011*, pages 1331–1338.

[50] Yao, H., Wang, Y., Wei, Y., Zhao, P., Mahdavi, M., Lian, D., and Finn, C. (2021). 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.

[51] You, K., Kou, Z., Long, M., and Wang, J. (2020). Co-tuning for transfer learning. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020*.

[52] Zielinski, B., Plichta, A., Misztal, K., Spurek, P., Brzychczy-Wloch, M., and Ochonska, D. (2017). Deep learning approach to bacterial colony classification. *PLOS ONE*, 12(9):1–14.

Submission Checklist

1. For all authors...

   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] See Section 1 and the research question.

   (b) Did you describe the limitations of your work? [Yes] Our methods are explained in detail in Section 2.

   (c) Did you discuss any potential negative societal impacts of your work? [Yes] The authors have determined that this work presents no notable negative impacts to society.

   (d) Did you read the ethics review guidelines and ensure that your paper conforms to them? `https://2022.automl.cc/ethics-accessibility/` [Yes] The paper conforms to the guidelines.

2. If you ran experiments...

   (a) Did you use the same evaluation protocol for all methods being compared (e.g., same benchmarks, data (sub)sets, available resources)? [Yes] All evaluations have been performed in the same way for all methods.

   (b) Did you specify all the necessary details of your evaluation (e.g., data splits, pre-processing, search spaces, hyperparameter tuning)? [Yes] See Section B.

   (c) Did you repeat your experiments (e.g., across multiple random seeds or splits) to account for the impact of randomness in your methods or data? [Yes] See Section B.

   (d) Did you report the uncertainty of your results (e.g., the variance across random seeds or splits)? [Yes] For example, see Figure 1.

   (e) Did you report the statistical significance of your results? [No]

   (f) Did you use tabular or surrogate benchmarks for in-depth evaluations? [No] all experiments are the result of direct experimentation on the Meta-Album dataset. Note that the results of the second experiment use the results of the first experiment (Figure 1) as basis.

   (g) Did you compare performance over time and describe how you selected the maximum duration? [No] No time comparison was included.

   (h) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See Section B.

   (i) Did you run ablation studies to assess the impact of different components of your approach? [No]

3. With respect to the code used to obtain your results...

   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results, including all requirements (e.g., `requirements.txt` with explicit versions), random seeds, an instructive `README` with installation, and execution commands (either in the supplemental material or as a URL)? [Yes] See the GitHub repository for these documents.

   (b) Did you include a minimal example to replicate results on a small subset of the experiments or on toy data? [Yes] There are minimal example scripts provided in the GitHub repository.

   (c) Did you ensure sufficient code quality and documentation so that someone else can execute and understand your code? [No] Currently, the quality and documentation of the code is suboptimal.

(d) Did you include the raw results of running your experiments with the given code, data, and instructions? [Yes] The GitHub repository includes a csv file with the raw data from the experiments.

(e) Did you include the code, additional data, and instructions needed to generate the figures and tables in your paper based on the raw results? [No] Procedures to generate the figures from the data are currently not included in the GitHub repository.

4. If you used existing assets (e.g., code, data, models)…

(a) Did you cite the creators of used assets? [Yes] See Section A.

(b) Did you discuss whether and how consent was obtained from people whose data you're using/curating if the license requires it? [N/A] No license required explicit consent.

(c) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No] we use the Meta-Album datasets, which have been carefully checked and curated by the original authors.

5. If you created/released new assets (e.g., code, data, models)…

(a) Did you mention the license of the new assets (e.g., as part of your code submission)? [Yes] The license can be found in the corresponding GitHub repository.

(b) Did you include the new assets either in the supplemental material or as a URL (to, e.g., GitHub or Hugging Face)? [Yes] See Section B.

6. If you used crowdsourcing or conducted research with human subjects…

(a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A] no crowdsourcing or human subjects were involved.

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A] no crowdsourcing or human subjects were involved.

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A] no crowdsourcing or human subjects were involved.

7. If you included theoretical results…

(a) Did you state the full set of assumptions of all theoretical results? [N/A] no theoretical results were included.

(b) Did you include complete proofs of all theoretical results? [N/A] no theoretical results were included.

A Data

We use Meta-Album as the primary data source for our experiments. Meta-Album is a collection of image classification datasets designed for metalearning research [43]. The current version of Meta-Album features 30 datasets from 10 domains, each containing three 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. For this reason, 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, except 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 in the micro version, every dataset features exactly 20 classes.

In this work, we 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 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 artefacts and to facilitate the up- or down-scaling. These processing steps were also adjusted to preserve recognizability by the human eye.

B Implementation

B.1 Setup

For our experiments, we apply the following methods described in Section 2. In the data preparation stage, 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 five 5-way tasks, again using an 80 versus 20 percent ratio for the support set and query set. The task split is also performed randomly, based on a chosen seed t, where t corresponds directly to the task number. Since every class has exactly 40 examples, 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 Meta-Album dataset, 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 [16] 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

Domain Name	Dataset Name	Dataset ID	#Classes	#Images	Source
Large Animals	Birds	BRD	315	12600	Birds 400 [33]
	Dogs	DOG	120	4800	Stanford Dogs [20]
	Animals with Attributes	AWA	50	2000	AWA [47]
Small Animals	Plankton	PLK	86	3440	WHOI [17]
	Insects 2	INS_2	102	4080	Pest Insects [45]
	Insects	INS	104	4120	SPIPOLL [38]
Plants	Flowers	FLW	102	4080	Flowers [29]
	PlantNet	PLT_NET	25	1000	PlantNet [12]
	Fungi	FNG	25	1000	Danish Fungi [31]
Plant Diseases	PlantVillage	PLT_VIL	38	1520	PlantVillage [9]
	Medicinal Leaf	MED_LF	25	1000	Medicinal Leaf [36]
	PlantDoc	PLT_DOC	27	1080	Plant Doc [39]
Microscopy	Bacteria	BCT	33	1320	DiBas [52]
	PanNuke	PNU	19	760	PanNuke [10, 11]
	Subcel. Human Protein	PRT	21	840	Protein Atlas [42]
Remote Sensing	RESICS	RESISC	45	1800	RESICS45 [4]
	RSICB	RSICB	45	1800	RSICB 128 [24]
	RSD	RSD	38	1520	RSD46 [27, 48]
Vehicles	Cars	CRS	196	7840	Cars [21]
	Airplanes	APL	21	840	Multi-type Aircraft [46]
	Boats	BTS	26	1040	MARVEL [14]
Manufacturing	Textures	TEX	64	2560	KTH-TIPS Kylberg UIUC [8, 28, 22, 23]
	Textures DTD	TEX_DTD	47	1880	Texture DTD [5]
	Textures ALOT	TEX_ALOT	250	10000	Texture ALOT [3]
Human Actions	100 Sports	SPT	73	2920	100 Sports [34]
	Stanford 40 Actions	ACT_40	39	1560	Stanford 40 Actions [49]
	MPII Human Pose	ACT_410	29	1160	MPII Human Pose [1]
Optical Char. Recog.	OmniPrint-MD-mix	MD_MIX	706	28240	OmniPrint [41]
	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. Table taken from [43].

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, 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. After 50 epochs, the model is stored and forms the pre-trained model.

After all pre-trained models are created, 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, optimiser, and corresponding hyper-parameters as 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. After the 50 epochs have been completed, we take the mean validation accuracy of the last ten 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 ten epochs as the *fine-tuning performance* (we take the mean value of the last ten epochs to limit possible distortions due to the

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 [30].

stochastic nature of the learning process). We repeat this for every model and task combination within a seed s, resulting in $30 \times 30 \times 5 = 4\,500$ records per seed.

We accompany every record in our meta-dataset with features that describe the involved datasets. Table 3 shows an overview of the different features we use. Averages and standard deviations of colour channels are calculated over all pixels to prevent possible distortions from taking double averages. In contrast, averages and standard deviations of colourfulness and entropy are calculated per image and then averaged. Section B.2 provides more details on the different features.

We repeat the complete procedure as described above for three seeds. The complete meta-dataset contains $3 \times 4\,500 = 13\,500$ entries. It combines the validation accuracy of the fine-tuning jobs with at most 40 features of their corresponding datasets, depending on the chosen feature set. We repeated the procedure using a ResNet34 network backbone, but it did not result in (noteworthy) different performances across the models.

e utilised the ALICE (Academic Leiden Interdisciplinary Cluster Environment) facility for our experiments. The various GPU compute nodes that ALICE offers were used in parallel to run our experiments, for which an overview can be found via the following link: `https://pubappslu.atlassian.net/wiki/spaces/HPCWIKI/pages/37519378/About+ALICE`.

The scripts and procedures used to run the experiments can be found via the following GitHub link: `https://anonymous.4open.science/r/data-centric-meta-learning-469F`.

B.2 Features and baselines

An overview of the different features that we provided to our meta-models can be found in Table 3. The measure of colorfulness is calculated as follows (according to Hasler and Süsstrunk [15]).

$$rg = R - G \tag{1}$$

$$yb = \frac{1}{2}(R + G) - B \tag{2}$$

$$\sigma_{rgyb} = \sqrt{\sigma_{rg}^2 + \sigma_{yb}^2} \tag{3}$$

$$\mu_{rgyb} = \sqrt{\mu_{rg}^2 + \mu_{yb}^2} \tag{4}$$

$$M = \sigma_{rgyb} + 0.3 \cdot \mu_{rgyb} \tag{5}$$

Here, M is the measurement of colourfulness, whereas R, G, and B represent the red, green, and blue values (respectively) of a single image from the dataset. First, M is calculated in parallel for all images in a given dataset. Subsequently, the average and standard deviation of M are calculated as measures of colourfulness for the dataset.

In our experiments, we use the baselines *model average*, *random expected* and *LCP*. They are implemented as follows:

1. *Model average.* Given a training dataset providing the performance of the available pre-trained models when fine-tuned to various tasks, select the pre-trained model for an unseen task with the highest average performance in the provided training data.
2. *Random expected.* Given k equals the number of available pre-trained models, expect to select the n-th ranking model in $1/k$ selections.
3. *LCP.* Given a collection of pre-trained models and an unseen task, select the model with the highest accuracy after one epoch of fine-tuning on the unseen task.

For the most factual description of our implementation, we refer to the GitHub repository.

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.
colourfulness mean	the mean colourfulness of the dataset (as defined by Hasler and Susstrunk [15]).
colourfulness std	the standard deviation in colourfulness, 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 3: A descriptive overview of the different meta-features 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.

C More details on methods

More details and visualizations of the methods can be found in Figures 4, 5, 6, 7 and 8.

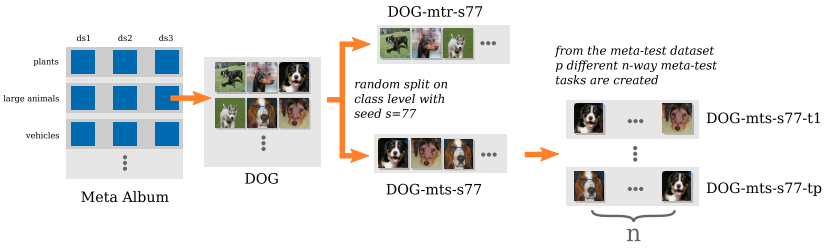

Figure 4: A schematic example of how we prepare the data for our experiments. Each dataset of Meta-Album consists of multiple classes. We select a dataset (in this case, DOG) and randomly split it on a class level using a random seed s. This results in a meta-train dataset (top) and a meta-test dataset (bottom), which we label DOG-mtr-s77 and DOG-mts-s77, respectively. The meta-test dataset is split further into p multi-class classification tasks (all having n classes). A pre-trained network is trained on this meta-train set (see Figure 5). The meta-test tasks are used to later fine-tune the pre-trained models (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.

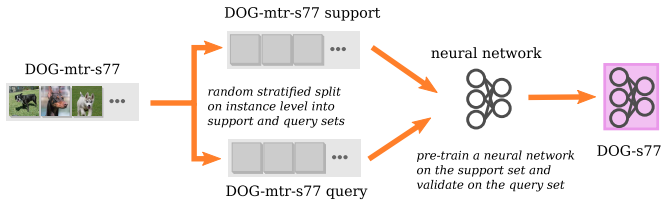

Figure 5: A schematic example of how we create the pre-trained models. We take a meta-train dataset (for example, DOG-mtr-s77) as created in Figure 4 and randomly split it further into a meta-train support dataset and a meta-train query dataset. The split is performed on an instance level and in a stratified 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. 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 Meta-Album dataset.

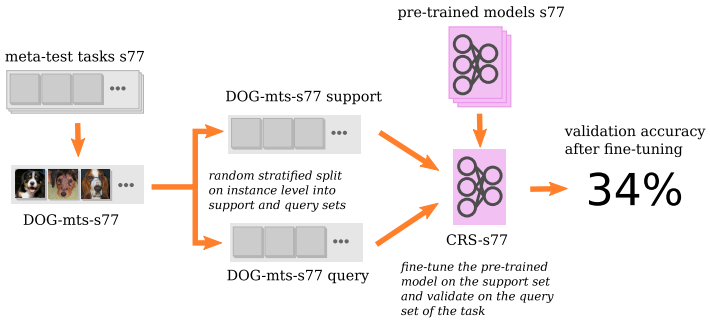

Figure 6: A schematic example of fine-tuning a pre-trained model to a new task. We select a meta-test task (DOG-mts-s77) as created in Figure 4 and split it further into a meta-test task support dataset and a meta-test task query dataset, similarly to Figure 5. Next, we select a pre-trained network (Cars with $s = 77$, or CRS-s77) as created in Figure 5. The layers of the pre-trained network are frozen except for the last layer. Then, the partially frozen pre-trained network is fine-tuned to the meta-test task support set. The fine-tuning process is validated with the meta-test task query set. After fine-tuning has been completed, the validation accuracy is recorded and later used as the target value for the meta-models (see Figure 7).

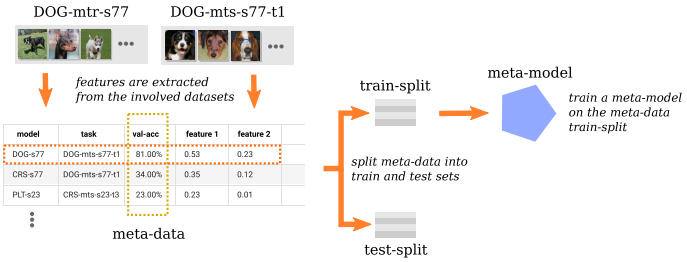

Figure 7: A schematic example of the assembly of meta-data and the training of meta-models. The meta-data consists of the recorded fine-tuning accuracy and meta-features of the involved datasets per fine-tuning job. The meta-data is split into a train and test set. The train set of the meta-data is used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

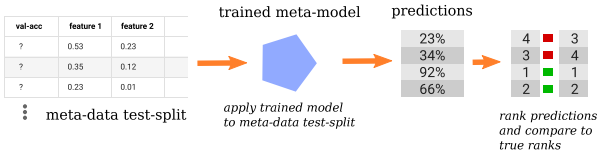

Figure 8: A schematic example of the meta-model validation. The test split of the meta-data is given to a trained meta-model. The meta-model then outputs the predicted values for the validation accuracy. The ranking of the predictions is compared to the true ranking of the validation accuracy of the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

D Results using pixel-based meta-features

Figure 9 shows the results of the methods that utilize primarily the pixel-based meta-features.

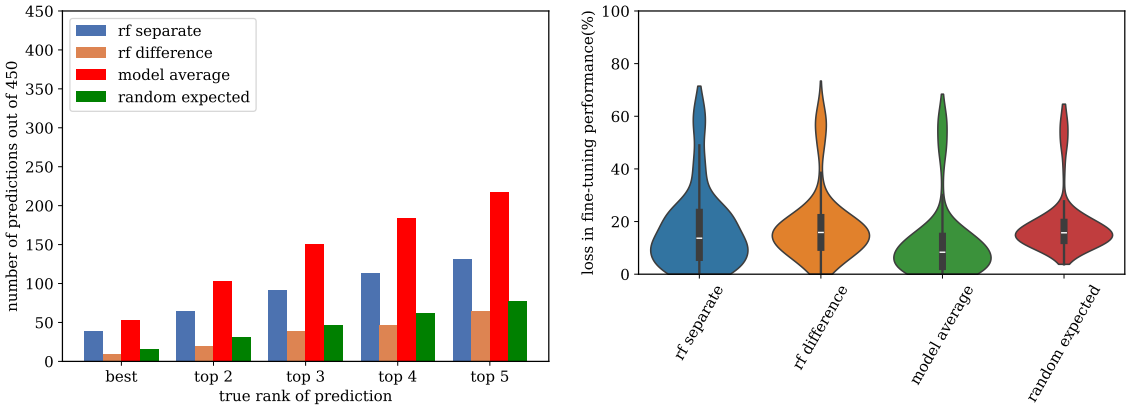

Figure 9: (left) Bar chart comparing the rank of the selections of different meta-models and baseline meta-models. Both *rf separate* and *rf difference* correspond to random forest meta-models trained on different feature sets. (right) Violin plot with embedded box plots showing the *selection loss* of different meta-models and baseline meta-models. Both *rf separate* and *rf difference* correspond to random forest meta-models trained on different feature sets.