Feature Extractor Stacking for Cross-domain Few-shot Learning

A thesis

submitted in fulfilment

of the requirements for the Degree

of

Doctor of Philosophy in Computer Science

at

The University of Waikato

by

Hongyu Wang

THE UNIVERSITY OF
WAIKATO
Te Whare Wānanga o Waikato

2024

Abstract

Cross-domain few-shot learning (CDFSL) addresses learning problems where knowledge needs to be transferred from one or more source domains into an instance-scarce target domain with an explicitly different distribution. Recently published CDFSL methods generally construct a universal model that combines knowledge of multiple source domains into one feature extractor. This enables efficient inference but necessitates re-computation of the extractor whenever a new source domain is added. Some of these methods are also incompatible with heterogeneous source domain extractor architectures.

The first part of this thesis proposes feature extractor stacking (FES), a new CDFSL method for combining information from a collection of extractors, that can utilise heterogeneous pretrained extractors out of the box and does not maintain a universal model that needs to be re-computed when its extractor collection is updated. We present the basic FES algorithm, which is inspired by the classic stacked generalisation approach, and also introduce two variants: convolutional FES (ConFES) and regularised FES (ReFES). Given a target-domain task, these algorithms fine-tune each extractor independently, use cross-validation to extract training data for stacked generalisation from the support set, and learn a simple linear stacking classifier from this data. We evaluate our FES methods on the well-known Meta-Dataset benchmark, targeting image classification with convolutional neural networks, and show that they can achieve state-of-the-art performance.

The second part of this thesis proposes an efficient semi-supervised learning method that applies self-training to the classification head only and show that it can yield very consistent improvements in average performance in the Meta-Dataset benchmark for cross-domain few-shot learning when applied with FES and other contemporary methods utilising centroid-based classification.

The third part of this thesis proposes a bidirectional snapshot selection strategy for FES, leveraging its cross-validation process and the ordered nature of its snapshots, and demonstrates that a 95% snapshot reduction can be achieved while retaining the same level of accuracy.

Acknowledgements

First and foremost, I would like to express my sincerest gratitude to my chief supervisor Eibe Frank, my supervisors Bernhard Pfahringer and Geoffrey Holmes, and my ex-supervisor Michael Mayo. Throughout my study, they have always encouraged me to pursue avenues of most interest to me and taught me how to be self-reliant in research. Their guidance has been invaluable to me. I could not have asked for a more inspiring and compassionate panel of supervisors.

I am eternally grateful to my loved ones for always being there for me. I could not have reached thus far without their unconditional love and support through thick and thin.

I would like to thank the Ministry of Business, Innovation & Employment for funding the User-friendly Deep Learning project that my PhD is a part of.

I would like to thank Eibe for offering me work to develop a visual classifier model for New Zealand species as a side project. I am grateful to Eibe for his supervision, Paul Schlumbom for work performed together, the TAIAO programme for funding this project, and iNaturalist for providing the data.

I would like to thank the Artificial Intelligence Institute and its director Albert Bifet for providing me with opportunities to participate in AI conferences and events.

I would like to thank the School of Computing and Mathematical Sciences and the University of Waikato for being a wonderful environment to study in. In particular, I would like to thank lecturers Tony Smith and Tim Stokes, alumnus Henry Gouk, and fellow students Chen Zheng and Paul Schlumbom for their insightful discussions with me.

Lastly, I would like to thank my friends for being supportive of my endeavour. In particular, I would like to thank my watercolour teacher Haydn Rive and my vehicle instructor Ben Davis for enriching my study life with interesting and useful skills.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Machine learning research and applications have attained rapid advances in recent years. Both predictive accuracy of machine learning models and the complexity of the domains they are used for have seen impressive development. Evidently, contemporary state-of-the-art computer vision models can outperform human observations by a significant margin (He et al., 2015, 2016; Tan and Le, 2021), and large natural language processing models can encode and memorise considerably more knowledge than an average person (Vaswani et al., 2017; Brown et al., 2020). In general, more advanced models are more heavily parameterised with a higher learning capacity.

Along with the advent of increasingly more powerful machine learning models, demand for training data and computational resources is on the rise, as they generally need to increase in tandem with a model's capacity for it to achieve an optimal learning outcome. It has become commonplace for state-of-the-art models to require billions of human-labelled training instances and hundreds of thousands of GPU hours in training time (Brown et al., 2020; Dosovitskiy et al., 2021), rendering their training from scratch infeasible to most entities interested in utilising them, as only a few have such resources to train such models from scratch.

A common approach to circumventing this problem is transfer learning. A model is trained from scratch once through a process known as "pretrain-

ing", and the trained model distributed as a set of all its parameter values, normally called a checkpoint. One such checkpoint can later be loaded into the model's architecture to be refined on new data, which is referred to as "fine-tuning". The key to fine-tuning is the integration of information in the new data into the model's "experience" gained from the pretraining data, such that the fine-tuned model grasps concepts in the new data without forgetting crucial knowledge gained in pretraining. For naive transfer learning, which comprises fine-tuning all model parameters on new data, a sizeable labelled training set, generally containing at least a few thousand instances, is still required (Tan and Le, 2021; Dosovitskiy et al., 2021), as it becomes hard to learn from a smaller training set effectively without overfitting (Vinyals et al., 2016; Snell et al., 2017). In real-world applications such as classification of medical imagery and machine learning for environmental conservation, it may be infeasible to obtain a sizeable training set due to limited population sizes, privacy concerns, and expensive expert input.

Cross-domain few-shot learning (CDFSL) is a more sophisticated form of transfer learning that addresses the problem that deep learning methods, such as convolutional neural networks (CNN) for image classification, generally require a large amount of labelled training data to achieve high predictive accuracy when trained from scratch. CDFSL algorithms are designed for scenarios where only a few labelled training instances in the target domain are available in the form of a "support set". The aim is to nevertheless achieve high accuracy when predicting labels for instances of the target domain that have never been seen before as the "query set". This can generally only be achieved by applying transfer learning: taking knowledge gleaned from one or several source domains with large-scale training data and using this knowledge to inform learning in a few-shot target domain.

In CDFSL, the source domain(s) and the target domain are assumed to have potentially very distinct properties. This cross-domain setting is arguably more realistic than the "in-domain" scenario, used in some few-shot learning

literature (Vinyals et al., 2016), where the source and the target domains comprise mutually exclusive sets of classes obtained from the same dataset.[1] It also yields harder learning problems due to greater domain shift.

In a CDFSL setting with multiple source domains, an algorithm needs to both select relevant source domains and effectively transfer their knowledge into a target domain using a few-shot support set, as shown in Figure 1.1. Performance is measured by "meta-testing"—transferring model(s) using target domain support sets and evaluating their predictive accuracy on corresponding query sets. Recent work considering performance in image classification, which is the setting we also focus on in this thesis, shows that single-domain learning (SDL) and vanilla multi-domain learning (MDL), which applies one feature extractor and multiple classification heads, fail to achieve competitive performance compared to methods specifically designed for CDFSL (Triantafillou et al., 2020b; Li et al., 2021).

A majority of recently published CDFSL methods involve building a universal model from a collection of extractors, with each extractor being a deep neural network pretrained in a distinct source domain. This comprises the "meta-training" phase, which is performed before meta-testing begins. The universal-model paradigm is generally efficient when performing meta-testing because a single universal feature extractor is used and fine-tuned on the support set, usually in conjunction with a simple robust classifier that turns extracted feature vectors into predictions. However, training the universal model is computationally expensive, and some methods constrain all extractors to the same architecture as the intended universal model (Triantafillou et al., 2021), rendering them inapplicable to heterogeneous extractor collections that are likely to occur in real-world practice. Moreover, they may require adjustment based on pre-existing domain knowledge to function well. For example, given a source domain/extractor collection for image classification consisting

[1]Note that, strictly speaking, this also creates distinct domains because the joint probability distributions will differ. However, they will be strongly related.

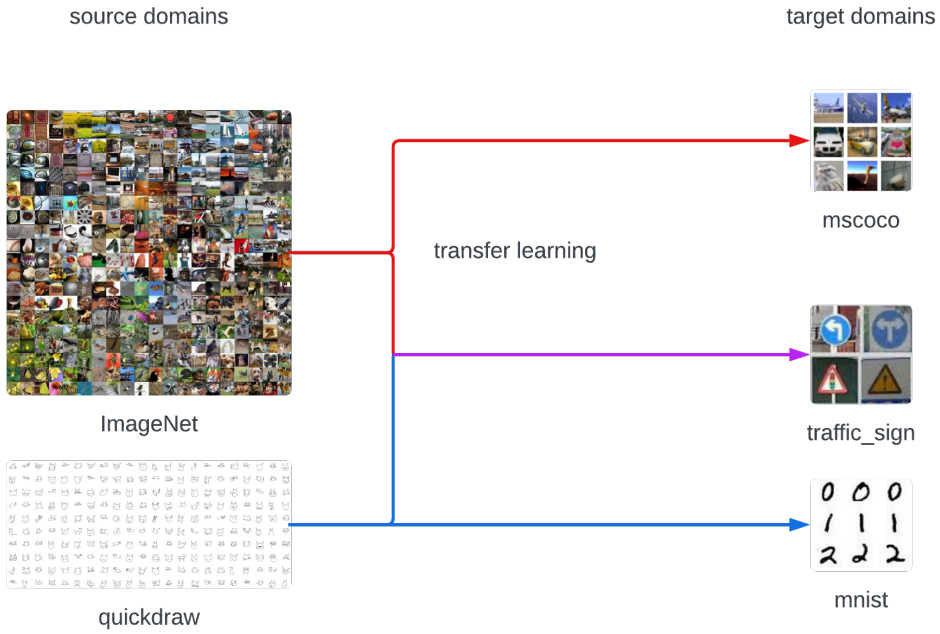


Figure 1.1: Example of CDFSL with multiple source domains. Given two source domains, ImageNet consisting of natural images and quickdraw consisting of hand-drawn symbols, transfer learning needs to select relevant source domains and fine-tune model(s) to each target domain. For mscoco, consisting of natural images, ImageNet needs to be selected; for mnist, consisting of hand-written digits, quickdraw needs to be selected; and for traffic_sign, consisting of natural images of robust symbols, both source domains need to be selected.

of ImageNet (Deng et al., 2009; Russakovsky et al., 2015), along with other, less comprehensive source domains, authors often assign greater importance to the ImageNet extractor during training (Triantafillou et al., 2021; Li et al., 2021). This achieves good performance on benchmarks, which normally include target domains such as CIFAR-10 that are quite similar to ImageNet in nature, but may not be as useful in real-world applications involving less similar data. Lastly, the process of deriving a universal model is non-incremental, which means it needs to be re-run whenever an extractor is updated or added, and normally requires access to the entire meta-training dataset (Triantafillou et al., 2021; Li et al., 2021).

This thesis investigates the following hypothesis:

Hypothesis 1.1. *CDFSL can be performed effectively without a universal model by suitably combining individual source domain feature extractors.*

Novel methods are proposed and empirical experiments are conducted to support this hypothesis.

1.1 Feature Extractor Stacking

As an alternative approach that avoids the shortcomings of universal model-based methods, we propose a novel "lazy" CDFSL method, termed feature extractor stacking (FES), in order to answer the following research question using experiments:

Research Question 1.1. *Can a CDFSL method attain state-of-the-art performance without relying on a universal model?*

FES fine-tunes each extractor independently and trains a classifier using a form of stacked generalisation (Wolpert, 1992) during meta-testing. The "meta-training" phase in FES consists solely of training individual feature extractors, one for each source domain, using standard single-domain supervised learning. In practical applications, it may be possible to skip meta-training

entirely if a set of suitable feature extractors has been obtained from other sources. FES is fully compatible with heterogeneous extractor collections, imposing no constraints on their architecture or fine-tuning configuration. It assumes equal importance of the extractors *a priori*, determining their task-specific relevance based purely on the support set, and does not require derivation of a universal model.

Along with the basic FES algorithm, which applies a simple linear stacking classifier, we present two variants: convolutional FES (ConFES) and regularised FES (ReFES). ConFES replaces the flat global kernel of FES with a hierarchy of depthwise convolutional kernels, reducing the number of parameters in the stacking classifier. ReFES applies fused lasso regularisation (Tibshirani et al., 2005) to the stacking classifier of FES to reduce the weights of irrelevant snapshots and induce smooth weight transition between adjacent snapshots.

We evaluate FES and its variants on the Meta-Dataset benchmark (Triantafillou et al., 2020b), which contains eight source domains and five target domains, and include five additional target domains: CropDisease, EuroSAT, ISIC, ChestX, and Food101 (Guo et al., 2020; Bossard et al., 2014). During our experiments, we discovered a sampling caveat in Meta-Dataset's official implementation, which could lead to potentially over-optimistic results, and implemented a fix for this issue. We show that FES outperforms three recent universal-model methods: URL (Li et al., 2021), FLUTE (Triantafillou et al., 2021), and a URL extractor with TSA fine-tuning (Li et al., 2022a), and advances the state of the art on Meta-Dataset. We also discuss practical advantages of FES in real-world scenarios, as FES can work with heterogeneous extractors out of the box and does not need to train a universal model.

1.2 Self-trained Centroids

Contemporary CDFSL methods (Li et al., 2022a; Triantafillou et al., 2021; Li et al., 2021) generally apply knowledge transfer by fine-tuning pretrained deep feature extractors, used in conjunction with a simple nearest-centroid classifier (Mensink et al., 2013; Snell et al., 2017), on the target dataset. However, they do not attempt to exploit unlabelled data during learning. In scenarios where additional target domain instances are available but lack labels, semi-supervised learning offers the prospect of improved performance. However, common semi-supervised methods are designed for relatively large sets of labelled data (Laine and Aila, 2017; Tarvainen and Valpola, 2017; Chen et al., 2020) and can easily go astray using small labelled sets. There exist a number of semi-supervised few-shot learning methods that apply semi-supervised learning to meta-training (Ren et al., 2018; Bateni et al., 2022; Xu et al., 2022; Islam et al., 2021), but literature is lacking on semi-supervised learning applied to CDFSL during meta-testing based on pretrained feature extractors, whether obtained with meta-training or not. We aim to answer the following research question using experiments:

Research Question 1.2. *Can a semi-supervised method be applicable to pretrained feature extractors and consistently improve CDFSL accuracy over supervised learning?*

We propose an efficient semi-supervised learning method applicable to pretrained feature extractors that keeps the feature extractors fixed after fine-tuning them on the labelled data and applies a classic semi-supervised learning method known as self-training to the classification head—the nearest-centroid classifier—only. Full self-training is a semi-supervised learning method that leverages unlabelled instances through an iterative process (Rosenberg et al., 2005): 1) train a model using the labelled dataset, 2) pseudo-label unlabelled instances with the trained model, and 3) update the labelled dataset with the unlabelled instances and their pseudo-labels. The labelled dataset consists of

only labelled instances during the first iteration of training and additionally includes unlabelled instances with their pseudo-labels in all following iterations. The train-label loop iterates until a stopping criterion is met.

The training step in this full self-training loop involves optimising all trainable parameters in the model and can become time-consuming if the feature extractor is heavily parameterised. More importantly, in few-shot learning, the small labelled dataset may provide insufficient guidance to reliably update such a large number of parameters in self-training. We address both issues by applying self-training to the centroid classifier only, yielding self-trained centroids (STC) for cross-domain few-shot learning. In this approach, feature vectors of labelled and unlabelled instances are extracted using the fine-tuned feature extractor, and the labelled feature vectors are used to compute initial centroids. Subsequently, these centroids are used in the nearest-centroid classifier to assign soft labels to the unlabelled feature vectors. Once these soft labels have been obtained, another set of centroids can be computed from the pseudo-labelled feature vectors. The two sets of centroids are averaged on a per-class basis to produce combined centroids. To form an iterative process, the new centroids can be used to soft-label the unlabelled instances again, for "labelled" and "unlabelled" centroids to subsequently be averaged again. To predict test instances, the combined centroids are used in the nearest-centroid classifier.

We apply STC to FES and other contemporary CDFSL methods, including URL, FLUTE, and a URL extractor with TSA fine-tuning. We evaluate them using the Meta-Dataset benchmark and show that STC used with 1000 unlabelled instances consistently improves average performance. We also demonstrate that STC is more efficient than full self-training based on updating all model parameters and performs better in cross-domain scenarios.

1.3 Bidirectional Snapshot Selection

Most recent top-performing CDFSL algorithms, such as URL, its TSA-based variant, and FLUTE, derive a universal model from a collection of source domains and then fit it to a support set using a small set of trainable parameters. In contrast, FES maintains a collection of source domain-specific models instead of a universal one and produces a weighted ensemble of model snapshots saved during support set fine-tuning and evaluated through cross-validation. FES outperforms the aforementioned universal model-based methods on the Meta-Dataset benchmark, achieving the current state of the art. FES also exhibits good interpretability as each model and snapshot's weight reflects its relevance to the support set learning task. On the other hand, it has a high storage cost as it saves a snapshot for every fine-tuning iteration for every model. This presents the following research question:

Research Question 1.3. *Can the storage cost of FES be reduced while retaining the same level of CDFSL accuracy?*

We endeavour to answer this question by taking inspiration from FES classifier heatmaps, which show that only a minority of snapshots are assigned significant weights when a FES model has been trained on a given support set: most snapshots are assigned small weights and are unlikely to individually have a meaningful impact on predictions.

Snapshot pruning is an obvious approach to reducing the storage cost of FES. This task can be framed as a feature subset selection problem where the predictions provided by every snapshot—in the form of logits—are considered features for the FES stacking classifier. Wrapper-based approaches, for example, using greedy search methods such as forward selection or backward elimination, can be applied to such problems by using cross-validation performance to determine the relevance of feature subsets (Kohavi and John, 1997). However, naive greedy search yields an infeasibly large search space when applying FES with a non-trivial number of snapshots, as it evaluates

combinations of a selected subset with each of the remaining features, yielding runtime that scales quadratically in the total number of snapshots. Fortunately, we can exploit the ordering of snapshots, provided by the sequence of fine-tuning steps for each extractor, to limit the set of available choices, only evaluating a small pool of potentially relevant candidates in each step of the search. This strategy can be interpreted as a form of linear forward selection (Gütlein et al., 2009).

We propose a bidirectional snapshot selection (BSS) strategy for pruning FES ensembles that performs linear forward selection with an initial candidate pool comprising snapshots at the beginning and the end of every extractor's fine-tuning. When a candidate is selected, the pool is replenished with the candidate's adjacent snapshot in fine-tuning. We show BSS achieves over 95% snapshot reduction on average while retaining the same level of accuracy as the unpruned ensemble and outperforms several baseline strategies.

1.4 Contributions and Thesis Organisation

The main contribution of this thesis is feature extractor stacking as a novel high-performance CDFSL algorithm based on stacked generalisation. Convolutional FES and regularised FES are proposed as variants of FES to further combat the potential for overfitting. Applying self-trained centroids is identified as a semi-supervised method applicable to FES to utilise additional unlabelled training instances. Bidirectional snapshot selection enables reduction of the storage cost of FES while maintaining the same level of accuracy.

The thesis is structured as follows:

- **Chapter 2** reviews foundational few-shot learning literature, the Meta-Dataset benchmark used in our evaluation, and recent CDFSL methods using multiple source domains.
- **Chapter 3** defines FES and its ConFES and ReFES variants. They are evaluated and compared to recent CDFSL methods to answer the

research question whether a CDFSL method can attain state-of-the-art performance without relying on a universal model.

- **Chapter 4** defines STC. Large-scale semi-supervised learning methods are evaluated in a CDFSL setting. STC is evaluated with FES and other recent CDFSL methods to answer the research question whether a semi-supervised method can be applicable to pretrained feature extractors and consistently improve CDFSL accuracy over supervised learning.

- **Chapter 5** defines BSS. Accuracy and snapshot reduction of FES pruned with BSS are evaluated to answer the research question whether the storage cost of FES can be drastically reduced while retaining the same level of CDFSL accuracy.

- **Chapter 6** summarises the contributions of this thesis, analyses their implications, discusses their limitations, and presents potential future research avenues.

The content of Chapter 3 is based on an article accepted by the Machine Learning Journal and reproduced with permission from Springer Nature. The accepted version of the article is available as Wang et al. (2024). The content of Chapter 4 is based on a paper presented at the Second Conference on Lifelong Learning Agents in Montreal, Canada in 2023 (Wang et al., 2023).

Chapter 2

Related Work

We first review foundational work on few-shot learning, most of which involves a single domain that is split into multiple source and target domains. We then describe the Meta-Dataset framework for evaluating CDFSL methods with multiple source and target domains. Following this, we discuss methods in more recent literature that are relevant to the work presented in this thesis.

2.1 Foundational Few-shot Learning Work

Early work on few-shot learning is mostly in-domain, as the source and target domains are splits of the same dataset with mutually exclusive classes. For example, miniImageNet is a commonly used dataset for in-domain few-shot learning evaluation, first proposed by Vinyals et al. (2016), with 80 source domain classes, 20 target domain classes, and 600 size 84×84 colour images per class. Ravi and Larochelle (2017) later proposed a different split with 64, 16, and 20 classes for training (source), validation, and testing (target) respectively. The validation split is commonly used for hyperparameter tuning. Although this in-domain setting is arguably less realistic than the cross-domain setting, methods and techniques from this setting, the most prominent ones of which are discussed in this section, have been adopted and expanded on by later CDFSL research.

Later work exploring in-domain few-shot learning focuses primarily on more

comprehensive meta-training datasets, more sophisticated extractor architectures, and transductive learning, for example, Hu et al. (2022a); Singh and Rad (2023); Hu et al. (2022b). This work is orthogonal to and out of the scope of this thesis.

2.1.1 Matching Networks

Vinyals et al. (2016) proposed matching networks, an attention-based model trained using episodic meta-learning. As a target domain learning task is to learn to classify unlabelled instances with a few labelled target domain instances, the episodic meta-learning process aims to imitate such tasks in the source domain. Episodes are sampled using source domain instances and their labels. Each episode contains a few-shot training set and a test set. A model is meta-trained by predicting labels for the test set using the labelled training set and having its parameters optimised to match the predicted labels to the true labels of the test set, as shown in Algorithm 1. This process is iterated over many episodes sampled from the source domain as the model both learns source domain features and becomes accustomed to the few-shot learning format.

Matching networks use a neural network, for example, a CNN, to extract feature vectors from the training and test sets. The training feature vectors serve as context to transform each individual training or test feature vector using long short-term memory (LSTM) (Hochreiter and Schmidhuber, 1997). After transformation, a test feature vector's label is computed by an attention mechanism using the training feature vectors and their labels. In Vinyals et al. (2016), the attention mechanism is a simple cosine similarity measure s, and the predicted label $\hat{y}$ is computed using its corresponding feature vector $\hat{x}$ and the N training feature vectors x_n and labels y_n, $1 \leq n \leq N$:

Algorithm 1 Commonly used framework for episodic few-shot meta-learning

Input: Source domain $S = \{(x_1, y_1), ..., (x_N, y_N)\}$ with N instance x and label y pairs. S has K classes, i.e., $\forall i \in [1, N] : y_i \in [1, K]$. Model f_θ is parameterised with θ, which takes x as input and produces a probabilistic prediction for y.

Parameters: Number of classes per episode N_C, number of support instances per class N_S, and number of query instances per class N_Q. Loss measure L.

Output: Episodic loss ℓ.

1: Randomly sample classes $C : C \subseteq [1, K], \|C\| = N_C$.
2: **for** each sampled class $c \in C$ **do**
3: Randomly sample support instance and label pairs belonging to c, i.e., $S_c = \{(x_i, y_i), ..., (x_j, y_j)\} : \|S_c\| = N_S, \forall (x_i, y_i) \in S_c : y_i = c$.
4: Randomly sample query pairs belonging to c, i.e., $Q_c = \{(x_i, y_i), ..., (x_j, y_j)\} : \|Q_c\| = N_Q, S_c \cap Q_c = \varnothing, \forall (x_i, y_i) \in Q_c : y_i = c$.
5: **end for**
6: Denote all support instance and label *pairs* as S, all query *instances* as Q^x, and all query *labels* as Q^y.
7: Use f_θ to classify Q^x based on S: $f_\theta(Q^x|S)$.
8: **return** episodic loss $\ell = L(f_\theta(Q^x|S), Q^y)$.

$$\hat{y} = \frac{\sum_{n=1}^{N} e^{s(\hat{x}, x_i)} y_i}{\sum_{n=1}^{N} e^{s(\hat{x}, x_i)}}. \tag{2.1}$$

During meta-training, the feature extractor and the LSTM module are optimised to match $\hat{y}$ to the test instance's true label in the source domain. During meta-testing, $\hat{y}$ is the matching networks' prediction of a transformed query feature vector $\hat{x}$ based on the N transformed support feature vectors x_n and labels y_n, $1 \leq n \leq N$.

2.1.2 Prototypical Networks

Snell et al. (2017) proposed prototypical networks, which use the same episodic meta-training process as matching networks in Algorithm 1 but aggregate instances of each class into a single prototype instead of using them individually for classification like matching networks. Each prototype is computed as the arithmetic average of all feature vectors belonging to the class. For class c with N_c instances $\{x_1, ..., x_{N_c}\}$, using a feature extractor f_θ, the prototype is

$$p_c = \frac{\sum_{n=1}^{N_c} f_\theta(x_n)}{N_c}. \tag{2.2}$$

A nearest-centroid classifier is used to predict an unlabelled instance's class probabilities with the prototypes. Given C classes and similarity measure s, the predicted probability of class c for instance $\hat{x}$ is

$$P(\hat{y} = c|\hat{x}) = \frac{e^{s(f_\theta(\hat{x}), p_c)}}{\sum_{i=1}^{C} e^{s(f_\theta(\hat{x}), p_i)}}. \tag{2.3}$$

Note that in one-shot learning scenarios, where each class contains only one support instance, prototypes and support instances become equivalent, $p_c = x_c$, and consequently so do prototypical networks and matching networks.

Snell et al. (2017) argue that while extensions to matching networks in Vinyals et al. (2016), such as support and query embedding function decoupling and fully-conditional embedding with bidirectional LSTMs, are applicable to prototypical networks too, simple design choices can be used to achieve the same level of performance, for example, using negative squared Euclidean distance instead of cosine similarity as the similarity measure and using a higher number of classes in each episode during meta-training than during meta-test.

2.1.3 Model-agnostic Meta-learning

Finn et al. (2017) proposed model-agnostic meta-learning (MAML), which explicitly optimises a model's parameters such that good generalisation per-

Algorithm 2 MAML meta-batch

Input: Source domain S. Model f_θ parameterised with θ.

Parameters: Meta-batch size I. Loss measure L. Base learning rate α and meta learning rate β. Base-level optimisation iterations J.

Output: Updated parameters θ'.

for each meta-batch member $i \in [1, I]$ **do**
  Use Algorithm 1 to sample an episode from S with support instances X_S, support labels Y_S, query instances X_Q, and query labels Y_Q.
  Use a copy of the parameters θ for base-level gradient optimisation in this episode: $\theta'_i = \theta$.
  for each base-level optimisation step $j \in [1, J]$ **do**
    Compute support set loss of the parameters θ'_i: $\ell_{(S,i)} = L(f_{\theta'_i}(X_S), Y_S)$.
    Apply base-level gradient descent to the parameters θ'_i using the support set loss: $\theta'_i = \theta'_i - \alpha \nabla_\theta \ell_{(S,i)}$.
  end for
  Compute query set loss of the parameters θ'_i: $\ell_{(Q,i)} = L(f_{\theta'_i}(X_Q), Y_Q)$.
end for
Apply meta-level gradient descent to the parameters θ using the aggregated query set loss of all episodes in this meta-batch: $\theta' = \theta - \beta \nabla_\theta \sum_{i=1}^{I} \ell_{(Q,i)}$.
return θ'.

formance can be obtained from a small number of labelled training instances in a new task using only a few gradient steps. Essentially, the goal of MAML is to train a model to be easy to fine-tune. MAML is applicable to few-shot regression, few-shot classification, and reinforcement learning. Our review focuses on few-shot classification.

MAML performs both base-level and meta-level parameter optimisation. At base-level, MAML optimises a model's parameters with an episode's support set instances and labels, and computes the optimised parameters' loss with the same episode's query set instances and labels. This loss is accumulated over a meta-batch of episodes, and at meta-level, MAML updates the

model's parameters once with this meta-batch's loss, which constitutes one MAML optimisation step. This process is shown in Algorithm 2.

By using multiple episodes for each optimisation step, MAML performs parameter updates with more comprehensive domain knowledge than using a single episode. By setting an episode's query set loss after optimisation on its support set as the explicit meta-level optimisation goal, MAML meta-trains its model to be easy to fine-tune given such few-shot episodes. During meta-testing, the model is simply fine-tuned naively using a support set and evaluated using the corresponding query set. Finn et al. (2017) found that, during meta-testing, using the same learning rate α as in base-level meta-training and performing more optimisation iterations than J in base-level meta-training leads to good classification performance.

2.1.4 Meta-meta classification

Chowdhury et al. (2022) proposed meta-meta classification. The idea is to meta-learn a selection model that uses a support set to aggregate the output of base models with a simple, fully connected neural network, which can be meta-learned models themselves, for example, MAML models.

Given a source domain, meta-meta classification samples a large number of subset distributions from it, uses an embedding function to map them into high-dimensional space, and clusters them using a k-means algorithm. Each cluster is used to train a base model, for example, a MAML model. Given a few-shot support set, a meta-meta classifier, which is a simple, fully connected neural network, is used to aggregate the output of these base models based on the support set, and the meta-meta classifier's predictions are aggregated predictions of the base models after they are fitted to the support set. Intuitively, the meta-meta classifier's goal is to identify and utilise base models whose training clusters are closely related to the support set. The meta-meta classifier is meta-trained using episodes sampled from the source domain.

2.1.5 In-domain Methods in CDFSL

Guo et al. (2020) proposed the Broader Study of Cross-Domain Few-Shot Learning (BSCD-FSL) benchmark, using the 64 training split classes of mini-ImageNet as the source domain and including four datasets as target domains: CropDisease, EuroSAT, ISIC, and ChestX, which, in this order, incur increasingly larger domain shift from the source domain.

The learning task of BSCD-FSL is to first meta-train a model on the source domain, and then meta-test it using few-shot episodes sampled from the target domains. This task can be carried out using meta-learning methods, such as the ones introduced by Vinyals et al. (2016); Snell et al. (2017); Finn et al. (2017), or using basic transfer learning by applying batch-pretraining to a feature extractor using source domain data and fine-tuning a subset of its parameters along with an episode-specific classifier on a target domain support set. Guo et al. (2020) found that in this cross-domain setting, where source and target distributions differ in more significant ways, exhibiting different forms, semantics, and concepts rather than just different classes as in traditional in-domain learning, simple transfer learning outperformed state-of-the-art in-domain meta-learning methods at the time. This led to work dedicated to cross-domain few-shot learning (CDFSL), including the methods presented in this thesis.

2.2 Meta-Dataset

Triantafillou et al. (2020b) proposed the Meta-Dataset benchmark specifically designed for evaluating CDFSL methods. It has multiple configurations; we describe the CDFSL configuration that we use, which is the one used by most recent publications in the field. It contains eight source domains: ILSVRC-2012 (ImageNet), Omniglot, Aircraft, CUB-200-2011 (Birds), Describable Textures, Quick Draw, Fungi, and VGG Flower. Recent work utilising Meta-Dataset (Requeima et al., 2019) has extended its original set of two target do-

mains, Traffic Signs and MSCOCO, by adding three more: MNIST, CIFAR10, and CIFAR100. For an even more comprehensive evaluation, we add four target domains from the CDFSL benchmark in Guo et al. (2020)—CropDisease, EuroSAT, ISIC, and ChestX—but additionally also employ Food101 (Bossard et al., 2014). Only the 250 sanitised test images in each Food101 class are used in our experiments.

The Meta-Dataset framework splits each source domain into three partitions: training, validation, and test. The partitions are mutually exclusive in terms of their classes, with the training partition containing approximately 70% of source domain classes and the validation and test partitions containing approximately 15% each. The training and validation partitions are made available to the CDFSL method for meta-training, where the training partition is generally used to train feature extractors and the validation split to aid hyperparameter tuning. The test partition is reserved for evaluating the CDFSL method by sampling few-shot episodes during meta-test.

In contrast, the entire target domain data can be used for sampling episodes to evaluate few-shot learning in these domains. It is important to note that, by definition, only tasks sampled from target domains truly measure CDFSL performance. Using terminology that is common in this context, good performance in these domains indicates "strong generalisation"; good performance on tasks sampled from source domain test partitions indicates "weak generalisation". Strong generalisation is required to bridge the large generalisation gap between between training and test distributions in few-shot dataset generalisation problems, which makes it the focus of CDFSL work, and weak generalisation performance is usually included for completeness and compatibility with previous work (Triantafillou et al., 2021). Source domains in CDFSL can be compared to training data in standard machine learning, and target domains can be compared to the corresponding test data. In this sense, weak generalisation performance is analogous to accuracy on training data, whereas strong generalisation performance is analogous to accuracy on test data.

The most commonly used method to evaluate CDFSL algorithms on Meta-Dataset is to generate 600 any-way any-shot episodes from each dataset and measure each algorithm's mean accuracy on these 600 episodes, as well as the 95% confidence interval. Episode generation for meta-testing involves the target domains for strong generalisation and test partitions of the source domains for weak generalisation. Any-way any-shot sampling means the number of classes for each episode and the number of support instances per class are arbitrary, leading to imbalanced support sets more representative of real-world scenarios than fixed-way fixed-shot episodes. The query set is balanced in the Meta-Dataset setting. We adhere to this evaluation method in our experiments.

The leaderboard in the official Meta-Dataset repository (Triantafillou et al., 2020a) lists URL (Li and Zhang, 2021), FLUTE (Triantafillou et al., 2021), and a URL model with TSA fine-tuning (Li et al., 2022a) as the top-3 methods in its ranking at the time of writing this thesis. We mainly compare to these three methods in our evaluation.

Meta-Album (Ullah et al., 2022) is a noteworthy new few-shot learning benchmark comprising 40 datasets from 10 unique domains, with each domain containing four datasets. The 10 domains of Meta-Album are: large animals, small animals, plants, plant diseases, microscopy, remote sensing, vehicles, manufacturing, human actions, and optical character recognition. This benchmark only became available towards the end of the experiments for this thesis and was therefore not used.

2.3 CDFSL Work

We cover recent CDFSL methods in three broad categories: ensemble methods (Figure 2.1), adaptation networks (Figure 2.2), and universal models (Figure 2.3). Papers introducing these methods mostly evaluate them using the Meta-Dataset benchmark. Performance and publication time of these methods are

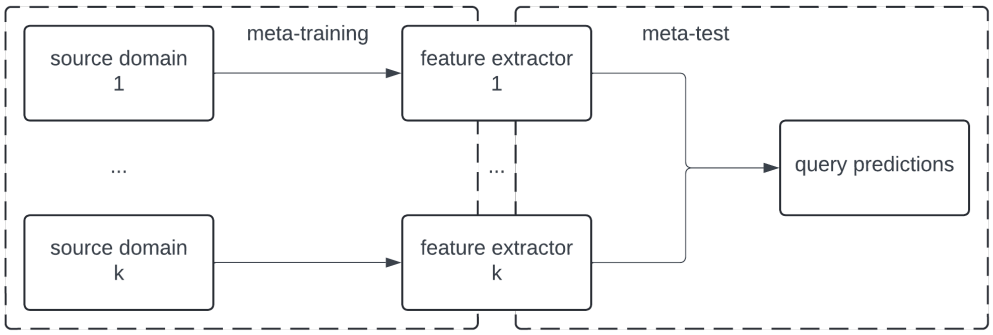


Figure 2.1: Ensemble methods maintain a collection of individually pretrained feature extractors and combine their output to form predictions in a few-shot episode.

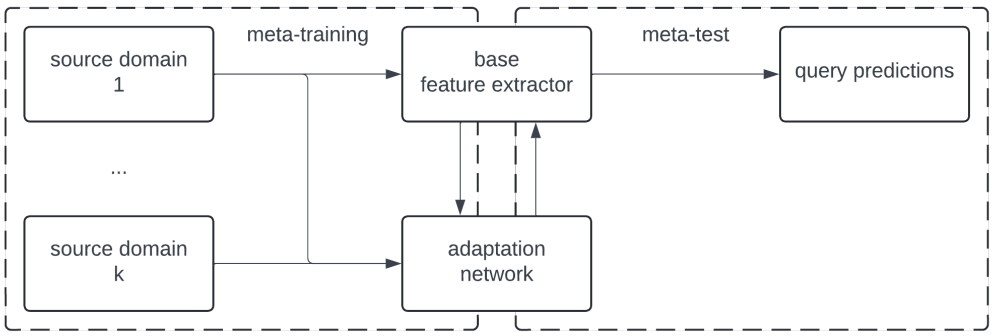


Figure 2.2: Adaptation network methods maintain a base feature extractor pretrained in one large source domain and an adaptation network pretrained in all source domains using the base extractor that is able to efficiently adapt the extractor to an episode.

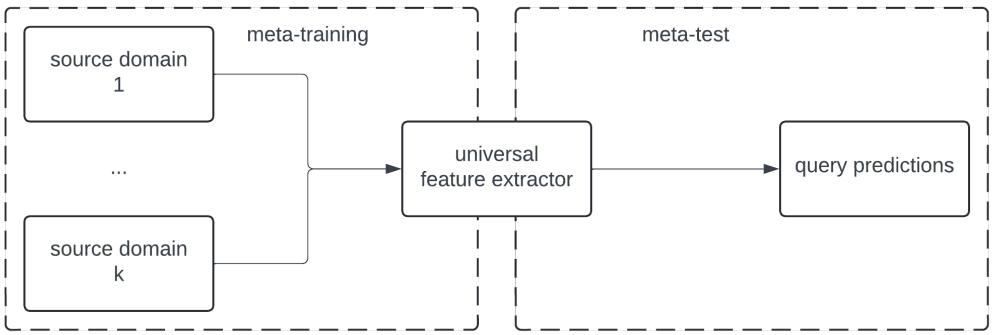


Figure 2.3: Universal model methods maintain a universal feature extractor pretrained in all source domains that is able to efficiently fit an episode.

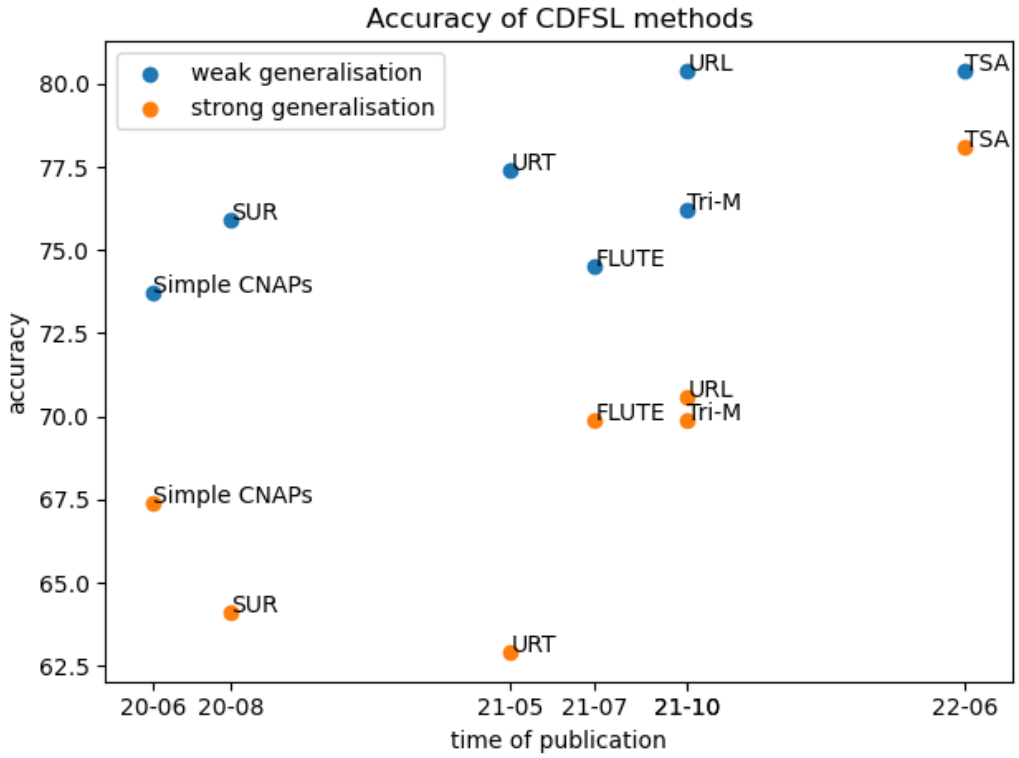


Figure 2.4: Meta-Dataset accuracy of CDFSL methods as reported in Li et al. (2022a) and their time of publication

shown in Figure 2.4, using Meta-Dataset accuracy reported in Li et al. (2022a).

2.3.1 Ensemble Methods

Ensemble methods such as SUR and URT maintain a collection of feature extractors pretrained in individual source domains. An instance's representation is formed by combining its feature vectors from all extractors. SUR combines feature vectors by multiplying each by a weight value trained on the support set using gradient descent, and it concatenates the weighted vectors. URT uses an attention mechanism trained on source domain episodes to produce a weighted average of feature vectors, and it can employ multi-head attention to produce multiple distinct vectors that are then concatenated.

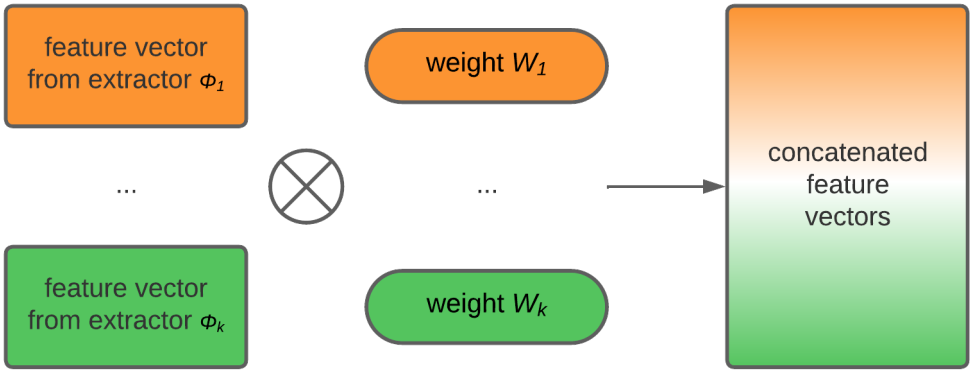


Figure 2.5: SUR maintains k feature extractors Φ and a weight array W with a weight associated with each extractor. Feature vectors from the extractors are multiplied by their corresponding weights and concatenated to form SUR's output vectors.

2.3.1.1 Selecting from Universal Representations

SUR (Dvornik et al., 2020) is a CDFSL method that utilises independently pretrained feature extractors directly for meta-testing. Most commonly, each extractor is pretrained using one source domain. Given K source domains and their extractors $f_{\Phi_1}, f_{\Phi_2}, ..., f_{\Phi_K}$ (or just $\Phi_1, \Phi_2, ..., \Phi_K$ for brevity), SUR maintains a weight array W consisting of K weights. Each weight corresponds to an extractor and represents its relevance to a task. Given an instance x, SUR uses each extractor Φ_k to extract a feature vector $f_{\Phi_k}(x)$ from x. It then weights the feature vectors with W before concatenating them, as shown in Figure 2.5, leading to

$$f_W(x) = \begin{bmatrix} W_1 \cdot f_{\Phi_1}(x) \\ ... \\ W_K \cdot f_{\Phi_K}(x) \end{bmatrix}. \tag{2.4}$$

Given a few-shot task with support set S containing N instances X and labels Y, SUR initialises W with a constant unit value, i.e., $W_k = 1$, which assumes no prior bias to any extractor. It extracts, weights, and concatenates support feature vectors leading to $f_W(X)$. As per Equation 2.2, for class c with N_c instances $\{x_1, ..., x_{N_c}\}$, its prototype of concatenated features is

$$p_c = \frac{\sum_{n=1}^{N_c} f_W(x_n)}{N_c}. \tag{2.5}$$

As per Equation 2.3, given C classes and similarity measure s, the estimated probability of class c for instance $\hat{x}$ using a nearest-centroid classifier is

$$P(\hat{y} = c|\hat{x}) = \frac{e^{s(f_W(\hat{x}),p_c)}}{\sum_{i=1}^{C} e^{s(f_W(\hat{x}),p_i)}}. \tag{2.6}$$

SUR uses gradient descent to optimise W to minimise loss on the support set S. Given loss function L, this is the loss

$$\ell_S = L(P(Y|X), Y). \tag{2.7}$$

Note that both concatenated features $f_W(X)$ and their prototypes p change as W changes. After optimisation of W is completed on S, given a query instance q, SUR predicts its probability for class c using the centroid classifier.

The optimised W can be interpreted as indicating the relevance of each extractor Φ to S, as it contains a single weight value corresponding to each extractor. An extractor with a high weight value contributes more to SUR's predictions, and therefore weight values correlate with relevance.

2.3.1.2 Universal Representation Transformer

URT (Liu et al., 2021a) also assigns a weight to each source domain extractor during meta-testing. However, it utilises a weight assignment model learned using meta-training and does not apply direct optimisation on the support set to obtain the weights. To this end, URT trains an attention mechanism (Vaswani et al., 2017) that learns to assign appropriate weights to source domain feature extractors based on a support set.

Given K extractors $\Phi_1, \Phi_2, ..., \Phi_K$, URT defines the universal representation of an instance x as *unweighted* feature vectors from the extractors concatenated

together, i.e., $ur(x) = concatenate(\Phi_1(x), \Phi_2(x), ..., \Phi_K(x))$. A class c's universal representation is defined as the centroid of its instances S_c's universal representations, i.e., $cur(c) = \frac{\sum_{x \in S_c} ur(x)}{\|S_c\|}$. Likewise, c's domain representation given extractor Φ_k is the centroid of S_c's representations from Φ_k, i.e., $cr_k(c) = \frac{\sum_{x \in S_c} \Phi_k(x)}{\|S_c\|}$.

For extractor Φ_k and class c, an attention head of URT is described by defining its queries $query_c$, keys $key_{k,c}$, and output attention score $attn_{k,c}$. The keys and queries have the same dimensionality qk_dim. The queries $query_c = W^{query} cur(c) + b^{query}$ with trainable weights W^{query} and biases b^{query}. The keys $key_{k,c} = W^{key} cr_k(c) + b^{key}$ with trainable weights W^{key} and biases b^{key}. The attention score of Φ_k in c is

$$attn_{k,c} = \frac{e^{\frac{query_c^\top key_{k,c}}{\sqrt{qk\_dim}}}}{\sum_{i=1}^{K} e^{\frac{query_c^\top key_{i,c}}{\sqrt{qk\_dim}}}}. \tag{2.8}$$

The per-class attention scores are averaged over all C classes to produce the attention score of extractor Φ_k based on the entire support set S, i.e. $attn_k = \frac{\sum_{c=1}^{C} attn_{k,c}}{C}$. Representation of instance x adapted by this attention head is an average of its feature vectors extracted by the K extractors weighted by the attention scores, i.e., $attn(x) = \sum_{k=1}^{K} attn_k \Phi_k(x)$. An attention head can focus on a single extractor by setting its score to one and the rest to zero or blend multiple extractors as a convex combination.

URT can employ multiple attention heads by concatenating their representations, i.e., $urt(x) = concatenate(attn_1(x), attn_2(x), ..., attn_H(x))$ with H attention heads. A regularisation loss is applied to encourage diversity in attention heads using the support set:

$$\ell_{attn} = \sum_{h=1}^{H} (1 - \sum_{k=1}^{K} attn_{k,h}^2)^2 + 2 \sum_{h_1=1}^{H} \sum_{h_2=h_1+1}^{H} (\sum_{k=1}^{K} attn_{k,h_1} \cdot attn_{k,h_2})^2, \tag{2.9}$$

where $attn_{k,h}$ denotes the attention score of the k-th extractor by the h-th attention head. Note that each attention head's output is averaged over its class-wise softmax output leading to a convex combination of extractors, i.e., $\sum_{k=1}^{K} attn_{k,h} = 1$. The first term in Equation 2.9 encourages each attention head to focus on one extractor, and the second term encourages the heads to produce different scores.

URT adopts a nearest-centroid classifier. As per Equation 2.2, for class c with N_c instances $\{x_1, ..., x_{N_c}\}$, its prototype of concatenated features is

$$p_c = \frac{\sum_{n=1}^{N_c} urt(x_n)}{N_c}. \tag{2.10}$$

As per Equation 2.3, given C classes and similarity measure s, the estimated probability of class c for query instance q using a nearest-centroid classifier is

$$P(y_q = c|q) = \frac{e^{s(urt(q),p_c)}}{\sum_{i=1}^{C} e^{s(urt(q),p_i)}}. \tag{2.11}$$

URT is evaluated using Meta-Dataset by meta-training on episodes sampled from the training partition, tuning hyperparameters with episodes sampled from the validation partition, and evaluating its accuracy using episodes sampled from the test partition.

2.3.2 Adaptation Networks

Adaptation network methods such as CNAPs and Tri-M maintain an extractor pretrained in a large source domain and an adaptation network that can efficiently adapt the extractor to the context of a few-shot episode. CNAPs and Tri-M both focus on using their adaptation networks to produce batch normalisation coefficients for the extractor based on the support set. The two methods differ in how they produce these coefficients: CNAPs produces one set per task, and Tri-M produces and combines one domain-specific set and one domain-cooperative set.

2.3.2.1 Conditional Neural Adaptive Processes

The CNAPs method, as proposed in Requeima et al. (2019), uses an extractor pretrained in a large source domain, for example, ImageNet (Deng et al., 2009; Russakovsky et al., 2015), and meta-trains adaptation networks, using episodes sampled from the source domains, to produce task-specific FiLM (Perez et al., 2018) transformations and a linear classifier for each few-shot episode. FiLM transformations are essentially scaling (multiplicative) and shift (additive) coefficients for batch normalisation (Ioffe and Szegedy, 2015) layers. This enables the composition of CNN feature maps to be fitted to a domain or task. In CDFSL, scarcity of labelled data means the convolutional parameters of a CNN—usually the majority of its parameters—cannot be fitted without significant overfitting, and full fine-tuning has been found to perform poorly (Vinyals et al., 2016; Snell et al., 2017). FiLM transformations offer the prospect of composing feature maps to fit a domain or task using a small set of trainable parameters without altering the convolutional parameters used to compute the feature maps. CNAPs uses adaptation networks to compute task-specific FiLM transformations; FLUTE (Triantafillou et al., 2021), discussed in Section 2.3.3.1, meta-trains source domain-specific FiLM transformations with one shared set of universal convolutional parameters and uses a blender network to produce a convex combination of FiLM transformations for each task.

A variant, simple CNAPs (Bateni et al., 2020), was later proposed utilising a non-parametric Mahalanobis distance (Galeano et al., 2015) measure in place of the classifier adaptation network of CNAPs (the FiLM transformation adaptation network is still used in simple CNAPs), reducing the parameter count and improving CDFSL performance. A transductive version of simple CNAPs was subsequently also proposed (Bateni et al., 2022), making use of clustering of query instances in feature space to achieve better performance than simple CNAPs, assuming that the query set is available as a batch instead of a sequential stream of incoming instances. As most other CDFSL methods do not rely on such an assumption, they cannot be compared to transductive

CNAPs on an even footing. Transductive CNAPs is used for comparison to semi-supervised methods in Chapter 4.

2.3.2.2 Multi-Mode Modulator

Tri-M (Liu et al., 2021b), akin to CNAPs, uses an extractor pretrained in a large-scale source domain and meta-trains a modulation network using source domain episodes to generate appropriate FiLM transformations for each few-shot episode. Tri-M maintains two sets of transformations—a domain-specific one and a domain-cooperative one—and its resulting FiLM transformation is a combination of the two. Tri-M determines a source domain for its domain-specific transformation in a way similar to how FLUTE (Triantafillou et al., 2021), discussed in the next section, utilises its blender network, except that Tri-M applies hard gating to select a single source domain. It uses an attention mechanism (Vaswani et al., 2017) to compose its domain-cooperative transformation from relevant source domains as a weighted combination. More specifically, a convex combination of domain-specific and domain-cooperative transformations is produced using the support set. This combination is used as the FiLM transformation for this task.

2.3.3 Universal Models

Universal model methods such as FLUTE and URL maintain a universal feature extractor that has been pretrained in all source domains and can be fine-tuned efficiently to fit a few-shot task. A FLUTE universal model comprises one set of convolutional weights pretrained in all source domains and a set of batch normalisation weights for each source domain. FLUTE produces and fine-tunes a convex combination of the batch normalisation weights using the support set, while the convolutional weights are frozen. A URL universal model is distilled from individual source domain extractors. URL fine-tunes a linear projection for the universal model's feature vectors based on a support set, while the model itself is frozen. TSA is a fine-tuning method originally

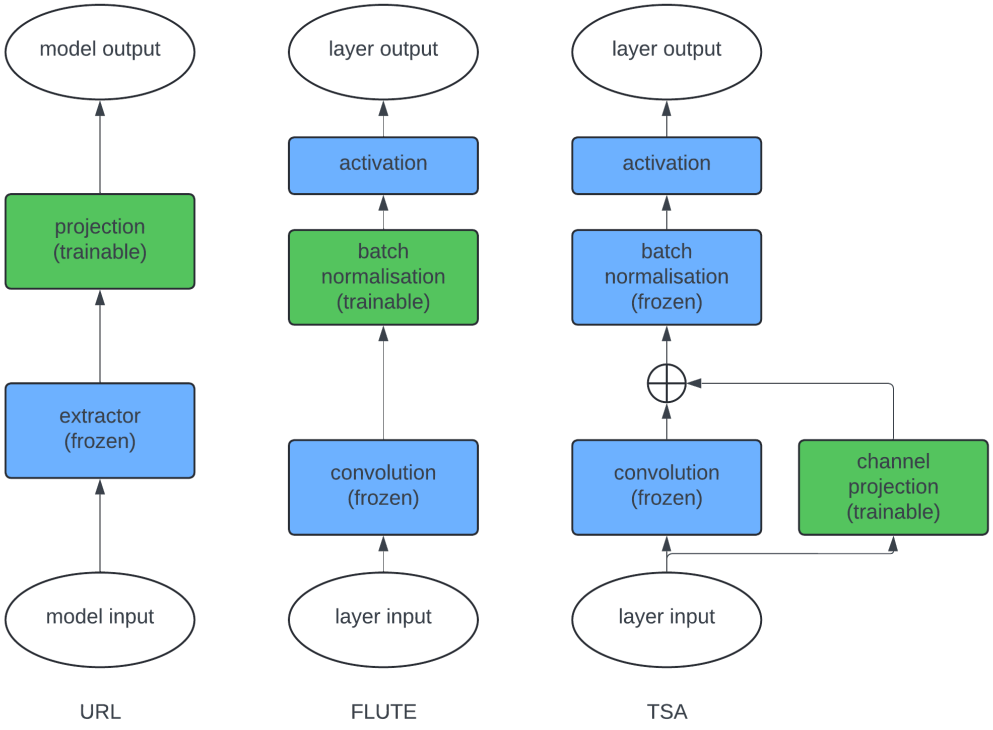


Figure 2.6: Fine-tuning methods used in URL, FLUTE, and TSA. Trainable modules are marked in green, while untrainable modules are marked in blue. URL fine-tuning is shown at model level, while FLUTE and TSA are shown at layer level.

developed using a URL model but applicable to CNNs in general. It attaches adaptors to convolutional layers to efficiently fit them to a support set with the layers' original weights frozen. The fine-tuning methods are shown in Figure 2.6.

2.3.3.1 Few-shot Learning with a Universal Template

Based on the FiLM approach (Perez et al., 2018), FLUTE (Triantafillou et al., 2021) trains a universal model in the source domains, employing the ResNet18 architecture (He et al., 2016) widely used in CDFSL, but maintaining a separate set of batch normalisation (Ioffe and Szegedy, 2015) parameters for each domain. The ResNet "template" contains one set of convolutional weights shared across all source domains, and only the batch normalisation parameters are specific to each source domain. FLUTE jointly trains the template

in all source domains. At each training iteration, a random source domain is selected—with ImageNet having a 50% probability of being selected and the other seven source domains evenly sharing the other 50% probability—and a batch of input data is sampled from the selected source domain. In forward propagation, the input batch flows through the shared convolutional layers and the selected domain's set of batch normalisation layers, and loss is computed by applying a cosine similarity classifier (Chen et al., 2019, 2021). A nuance of FLUTE training is that backpropagation is performed using a "meta-batch" of eight individual batches: the intention is to stabilise training by aggregating loss values across multiple domains. Hyperparameter tuning is performed using episodes sampled from source domain validation partitions.

When the template is trained, snapshots are frequently saved. The final template is chosen as the snapshot that performs best on the source domains' validation partitions. To establish performance, few-shot episodes are sampled from these partitions. For each episode, feature vectors are extracted using the shared convolutional layers and the domain's set of batch normalisation layers. Accuracy is computed using a nearest-centroid classifier (Mensink et al., 2013; Snell et al., 2017).

One more component of FLUTE, produced in a separate meta-training phase, is a blender network, which is a dataset classifier based on a permutation-invariant set encoder (Zaheer et al., 2017) followed by a linear layer. Given a batch of instances, the blender predicts, as a probability distribution, the source domain from which the batch is sampled. It is trained on batches sampled from the source domains' training partitions, and the final blender model is chosen using batches from the validation partitions.

Given a few-shot episode at meta-test time, the blender uses the support set to produce a probability distribution. These probabilities in turn are used to form a convex combination of the source-domain-specific batch normalisation weights. Along with the shared convolutional weights from the template, this forms the initial set of parameters for the ResNet18 feature extractor,

which is applied in conjunction with a nearest-centroid classifier. The model's batch normalisation parameters are then fine-tuned on the support set while its convolutional weights remain fixed.

2.3.3.2 Universal Representation Learning

The URL algorithm (Li et al., 2021) also generates a universal model. It first pretrains domain-specific ResNet18 extractors independently. Then, a separate ResNet18 feature extractor is trained to form a universal model by distillation. This model is trained to match each extractor's output feature vectors and logits using instances sampled from the extractor's corresponding domain. To this end, the universal model contains pairs of auxiliary domain-specific components that each comprise 1) a projection layer that transforms the universal extractor's feature vectors to match those of each domain-specific extractor, and 2) a classifier layer trained to match the logits produced by each extractor.

In the experiments by Li et al. (2021), ImageNet is made more prominent in distillation: ImageNet instances make up 50% of each mini-batch and the other seven source domains evenly make up the rest. Snapshots of the universal feature extractor are saved at predefined intervals during knowledge distillation. Episodes sampled from source domain validation partitions are used to select the best snapshot as a form of early stopping.

After meta-training, the auxiliary components of the universal model are discarded, leaving only the feature extractor. During meta-testing, this extractor is frozen, and a projection layer is initialised with an identity weight matrix and trained using the support set. The projected feature vectors are used to build a nearest-centroid classifier. Cosine similarity values between a feature vector to be classified and the centroids are used as logits. Fine-tuning minimises cross-entropy loss on the support set. Note that during fine-tuning, as the projection layer is optimised, projected support feature vectors change, and their centroids change as well. The fine-tuning effect can be interpreted

as forming better clusters with projected support feature vectors.

2.3.3.3 Task-Specific Adaptors

TSA (Li et al., 2022a) is a fine-tuning method especially designed for CDFSL. Given a pretrained extractor, trainable task-specific adaptors are attached to it, and the support set is used to optimise the adaptors with the extractor's original weights frozen. Like URL, TSA also attaches a trainable linear projection layer and a robust classifier to the end of the feature extractor during fine-tuning, but it adds further adaptor components. Among multiple configurations examined, the most effective approach for few-shot image classification found by Li et al. (2022a) is to attach channel projection matrices as residual connections to a model's convolutional layers. Li et al. (2022a) used TSA in conjunction with a URL-distilled universal extractor, but TSA can be applied to other CNN architectures as well.

Chapter 3

Feature Extractor Stacking

The SUR method for CDFSL discussed in Section 2.3.1.1 is appealing because it combines individual feature extractors. Unfortunately, it has been shown to perform worse than universal-model-based methods such as FLUTE. In this chapter, we investigate whether stacked generalisation can be used to create a competitive CDFSL method. We first cover literature in stacked generalisation and its applications. We then define the feature extractor stacking (FES) method, prove convexity of its optimisation, and introduce its two variants: convolutional FES (ConFES) and regularised FES (ReFES). Following this, we evaluate the FES methods, comparing to recent CDFSL methods using the Meta-Dataset benchmark, and analyse its performance.

3.1 Stacked Generalisation

Wolpert (1992) proposed stacked generalisation, which uses cross-validation to refine a set of base models' predictions by using their output as input to a meta-level model, henceforth referred to as "stacking classifier" to avoid confusion with the term "meta-learning" in the CDFSL setting. Stacked generalisation can be applied to a single base model, where the stacking classifier learns to correct errors in the base model's output. When applied to multiple base models, the stacking classifier learns to aggregate their output and can be seen as "a more sophisticated version of cross-validation" (Wolpert, 1992), where

basic cross-validation is winner-takes-all. We focus on the case with multiple base models.

The reason for cross-validation in stacked generalisation is to avoid data reuse and combat overfitting. Given an initialised base model Φ and a training set S, cross-validation splits S into a training partition S_{train} and a test partition S_{test}. Φ is trained on S_{train} leading to a trained model Ψ. Training data for the stacking classifier is extracted by Ψ using S_{test}. This process can be repeated using cross-validation folds to obtain training data for the stacking classifier from all instances in S. The stacking classifier can hence be trained using all instances in S while each instance is extracted by a model Ψ not trained on it. This process enables the stacking classifier to estimate each base model Φ's behaviour on new test instances after training.

Wolpert (1992) argued that some form of stacked generalisation is helpful in almost any real-world generalisation problem to minimise generalisation error. In computer vision problems related to the image classification problems considered in this thesis, Yeo et al. (2021) used stacked generalisation to combine signals from different cues to improve robustness of a computer vision model against distribution shift, for example, distortions in input images. Feuz and Cook (2015) applied stacking to remapping feature vector dimensions from a target domain back into multiple source domains separately using a labelled support set and building a base classifier from each source domain's pretrained hypothesis, and stacking outperformed methods such as voting ensembles in the activity recognition experiments of Feuz and Cook (2015).

3.2 CDFSL with Stacked Generalisation

Considering the CDFSL methods discussed in Section 2.3, the SUR method stands out because its meta-training process is straightforward: all it involves is pretraining individual source domain feature extractors. Once these have been obtained, SUR performs "lazy" learning in the sense that significant work

is only performed once the support set for a few-shot episode becomes available. This makes it very flexible because new extractors can be added at any time. However, SUR does not yield state-of-the-art performance. The new methods presented in this thesis are inspired by SUR and the old and established method of applying stacked generalisation to learning a classifier that combines predictions of multiple base classifiers. There are four primary differences between SUR and our stacking-based methods: 1) the source domain extractors are fined-tuned on the support set to extract more information from this data by attaching appropriate classifier layers to them, 2) two-fold cross-validation is used to generate training data for the stacking classifier to tackle overfitting, 3) the feature vectors of this training data consist of logits obtained from classifier layers attached to the extractors, and 4) multiple snapshots of each extractor are stored during fine-tuning and used to obtain sets of logits, adding further richness to the data available for training the stacking classifier.

In the following, we first explain the basic method of feature extractor stacking (FES) in detail and prove convexity of its optimisation, before describing two variants: convolutional FES (ConFES) and regularised FES (ReFES).

3.2.1 Feature Extractor Stacking

Given pretrained feature extractors, FES has three key components: fine-tuning extractors to obtain snapshots, two-fold cross-validation to produce training data for the stacking classifier, and training of the stacking classifier. Figure 3.1 depicts the FES framework.

3.2.1.1 Fine-tuning the Extractors

We use $f_{\Phi_1}, f_{\Phi_2}, ..., f_{\Phi_K}$ (or just $\Phi_1, \Phi_2, ..., \Phi_K$ for brevity) to denote the collection of pretrained feature extractors, where Φ represents the corresponding extractor's parameters and K is the number of source domains. The support set of a few-shot episode is denoted S and the query set Q. S contains N instances belonging to C classes. We fine-tune each extractor independently

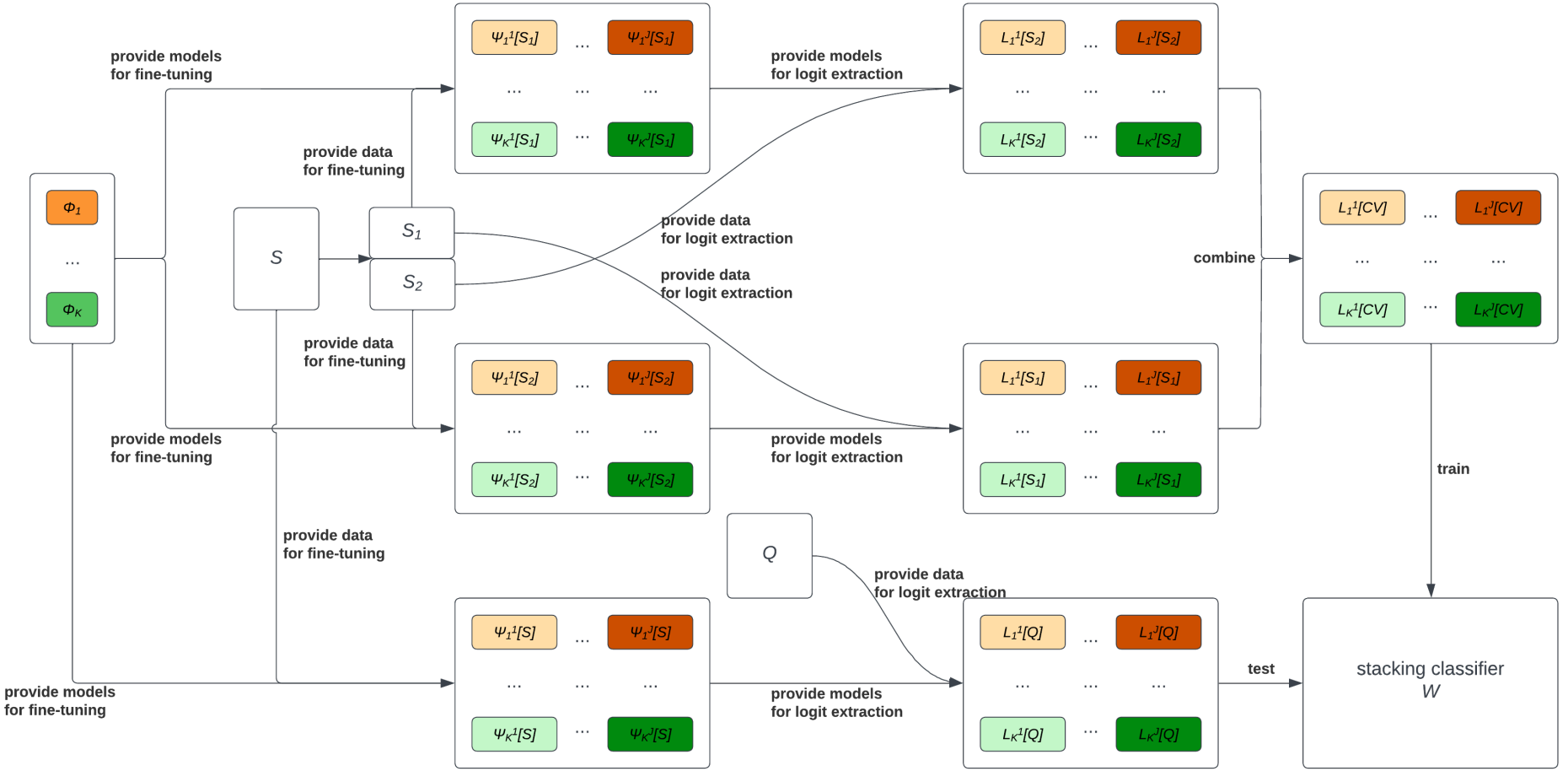


Figure 3.1: Framework of FES. Given an extractor collection with K extractors, each extractor Φ is set up as a network Ψ for fine-tuning. The support set S is split into S_1 and S_2 using stratified cross-validation. Each network Ψ is fine-tuned on one split, producing J snapshots in the process, and these snapshots are used to extract logits from the other split. Logits extracted from both splits are combined into cross-validated logits of the full support set, which are used to train a stacking classifier W to fit S's labels. The full S is then used to fine-tune Ψ, producing snapshots to extract logits for the query set Q. W takes Q's logits as input and predicts Q's labels.

on S. As f_Φ is a feature extractor, a classifier g with parameters Θ_1 is attached to f_Φ to produce logits. Auxiliary components with parameters Θ_2 may also be introduced to the model to aid fine-tuning, such as with TSA (Li et al., 2022a). The resulting model is defined as $h_\Psi = g_{\Theta_1} \circ f_{(\Phi,\Theta_2)}$, where we use Ψ to denote the combination of all parameters. It is possible for Θ_2 to be $\varnothing$, as auxiliary fine-tuning components are optional. J snapshots are saved sequentially at different fine-tuning iterations of h_Ψ. Each snapshot contains parameters $\Psi_k^j[S]$, where $k \in [1, K]$ and $j \in [1, J]$, with S denoting the fine-tuning set used.

3.2.1.2 Obtaining Training Data for Stacking

In stacked generalisation (Wolpert, 1992), cross-validation is employed to obtain training data for the stacking classifier to combat overfitting, and it is applied in FES as well. More specifically, we apply stratified two-fold cross-validation to the support set S, producing two splits S_1 and S_2, which will take turns serving as the training split S^{train} and the test split S^{test}. It is possible to employ more folds in FES, but using additional folds did not yield performance gains in our experiments.

Training on one of the training splits amounts to fine-tuning a network h_Ψ on this data. In principle, this could be done for a fixed number of iterations, and once complete, logits on the corresponding test split could be obtained as training data for the stacking classifier. However, this naive approach may not work well because it is not known how many iterations should be performed for fine-tuning to maximise accuracy of the full learning system. The approach we propose and evaluate in this thesis is instead based on the idea that we can take multiple snapshots of the models during fine-tuning and use all the snapshots' logits on the test folds for training the stacking classifier. In other words, the learning algorithm for the stacking classifier will be responsible for deciding which extractor snapshots are the most useful ones for making accurate predictions on the test folds.

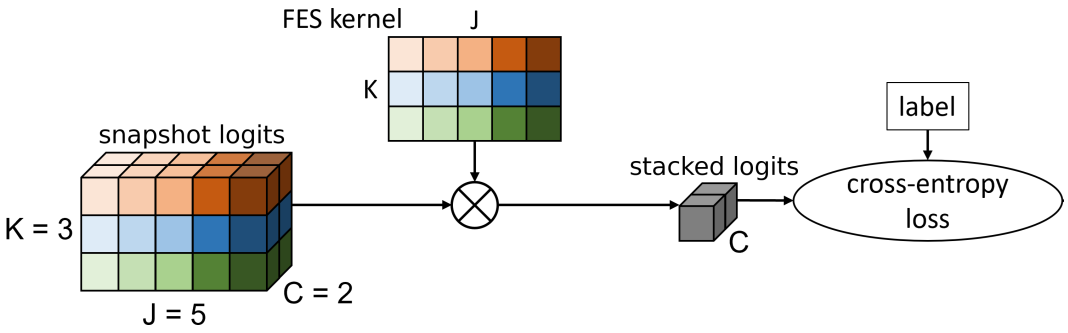


Figure 3.2: FES uses a global kernel to compute stacking classifier logits from the snapshots' base logits. The global kernel is essentially flat since it makes no use of the snapshots' temporal relations. For demonstration purposes, this figure and the following ones assume three extractors ($K = 3$), five fine-tuning snapshots per extractor ($J = 5$), and a two-class problem ($C = 2$).

More specifically, given a pair (S^{train}, S^{test}) and an extractor h_Ψ, we fine-tune h_Ψ on S^{train} with the same configuration used to obtain $h_{\Psi^j[S]}$, for example, optimiser, learning rate, etc., and save snapshots $h_{\Psi^j[S^{train}]}$ at the same iterations as $h_{\Psi^j[S]}$. Logits $L^j[S^{test}]$ are extracted from S^{test} with each $h_{\Psi^j[S^{train}]}$, i.e., $L^j[S^{test}] = h_{\Psi^j[S^{train}]}(S^{test})$. Using this approach, the two splits S_1 and S_2 can be used to alternately fine-tune extractors and produce logits $L^j[S_1]$ and $L^j[S_2]$, which are combined into $L^j[CV]$, i.e., logits for every support set instance extracted using cross-validation. Considering the logits from all K extractors jointly, $L^J_K[CV]$ is a tensor of shape $N \times K \times J \times C$, i.e., N support instances converted into logits for C classes extracted by $K \times J$ snapshot models, ready to serve as training data for the stacking classifier.

3.2.1.3 Stacking Classifier Training

The FES stacking classifier is a weight matrix W of shape $K \times J$, with W^j_k representing Ψ^j_k's weight. Given an instance l of shape $K \times J \times C$, the stacking classifier's output logits l^W are obtained using a simple weighted average:

$$l^W[c] = \sum_{k=1}^{K} \sum_{j=1}^{J} W^j_k \cdot l^j_k[c], \tag{3.1}$$

where c is one of the C classes. We compute the cross-entropy loss using the N support set logits L^W output by the stacking classifier and the one-hot-encoded

labels Y, i.e., $-\sum_{n=1}^{N} Y_n \log(\text{softmax}(L_n^W))$, which we minimise by training W. For interpretability, we constrain all values in W to be non-negative by clipping negative weights with ReLU. The FES stacking classifier is shown in Figure 3.2.

After training, W is used with Equation 3.1 to compute meta logits for the query set Q using the logits $L_K^J[Q]$ computed by the saved snapshots $\Psi_K^J[S]$. Then, a softmax function is used to obtain class probability estimates.

3.2.2 Proof of Convexity

Given a stacking instance l consisting of base logits obtained from the extractor snapshots, which the stacking classifier transforms into meta-level logits l^W, and the label c_y, the negative log-likelihood loss ℓ associated with the stacking classifier's parameters W is

$$\ell(W) = \log(\sum_{i=1}^{C} e^{l^W[c_i]}) - l^W[c_y]. \tag{3.2}$$

To prove that optimising FES is a convex problem, we show that for any two values of W, named A and B, a convex combination of the loss on A and the loss on B is never smaller than the loss obtained for the corresponding convex combination of the parameter values A and B, i.e.,

$$\ell(\lambda A + (1-\lambda)B) \leq \lambda\ell(A) + (1-\lambda)\ell(B), \lambda \in [0,1]. \tag{3.3}$$

Applying Equation 3.2 to Equation 3.3, we get

$$\log(\sum_{i=1}^{C} e^{l^{(\lambda A+(1-\lambda)B)}[c_i]}) - l^{(\lambda A+(1-\lambda)B)}[c_y] \leq$$
$$\lambda(\log(\sum_{i=1}^{C} e^{l^A[c_i]}) - l^A[c_y]) + (1-\lambda)(\log(\sum_{i=1}^{C} e^{l^B[c_i]}) - l^B[c_y]),$$

which can be simplified into

$$\log(\sum_{i=1}^{C} e^{l^{(\lambda A+(1-\lambda)B)}[c_i]}) \leq \lambda \log(\sum_{i=1}^{C} e^{l^A[c_i]}) + (1-\lambda)\log(\sum_{i=1}^{C} e^{l^B[c_i]}), \tag{3.4}$$

because using Equation 3.1, we have

$$
\begin{aligned}
& l^{(\lambda A+(1-\lambda)B)}[c_y] \\
=& \sum_{k=1}^{K}\sum_{j=1}^{J}(\lambda A_k^j + (1-\lambda)B_k^j)\cdot l_k^j[c_y] \\
=& \sum_{k=1}^{K}\sum_{j=1}^{J}\lambda A_k^j \cdot l_k^j[c_y] + \sum_{k=1}^{K}\sum_{j=1}^{J}(1-\lambda)B_k^j \cdot l_k^j[c_y] \\
=& \lambda\sum_{k=1}^{K}\sum_{j=1}^{J}A_k^j \cdot l_k^j[c_y] + (1-\lambda)\sum_{k=1}^{K}\sum_{j=1}^{J}B_k^j \cdot l_k^j[c_y] \\
=& \lambda l^A[c_y] + (1-\lambda)l^B[c_y].
\end{aligned}
$$

Similarly, Equation 3.4 can be transformed using Equation 3.1 into

$$\log(\sum_{i=1}^{C} e^{\lambda l^A[c_i]+(1-\lambda)l^B[c_i]}) \leq \lambda\log(\sum_{i=1}^{C} e^{l^A[c_i]}) + (1-\lambda)\log(\sum_{i=1}^{C} e^{l^B[c_i]}). \quad (3.5)$$

It is known that the LogSumExp function $LSE(x) = \log(\sum_{i=1}^{n} e^{x_i})$ is convex. Therefore, we have

$$\forall n \in \mathbb{Z}^+, \alpha, \beta \in \mathbb{R}^n : LSE(\lambda\alpha+(1-\lambda)\beta) \leq \lambda LSE(\alpha)+(1-\lambda)LSE(\beta). \quad (3.6)$$

Hence, Equation 3.5 is true because we can make the following assignments:

$$
\begin{aligned}
n &= C, \\
\alpha_i &= l^A[c_i], \\
\beta_i &= l^B[c_i].
\end{aligned}
$$

This completes the proof of Equation 3.3, and thus optimising FES on a single instance l is a convex problem. As the sum of convex functions is a convex function, optimising FES on a full batch L is also a convex problem. Therefore, FES is a convex optimisation problem.

3.2.3 Convolutional Feature Extractor Stacking

The basic FES approach does not exploit the temporal relation between logits obtained from adjacent snapshots produced during fine-tuning. Convolutional

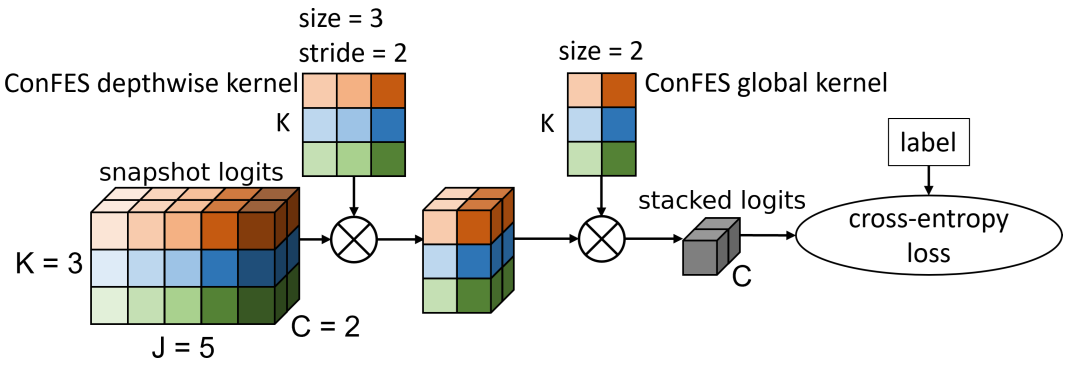


Figure 3.3: ConFES replaces the flat kernel of FES with a two-level kernel hierarchy. The base-level kernel is a one-dimensional depthwise, i.e., feature-extractor-wise, convolutional kernel, with predefined kernel and stride sizes. The high-level kernel is global like the one in FES but applied to the output of the base-level kernel, which requires substantially fewer parameters.

FES (ConFES) replaces the global kernel of FES with a kernel hierarchy, as shown in Figure 3.3, to treat the collection of logits as a time series. The hierarchy comprises one or more lower-level one-dimensional depthwise convolutional kernels and a top-level global kernel. The depthwise kernels condense the logit output sequence from each extractor's snapshots into a 1D feature map, while keeping the extractors separate, and the global kernel summarises the feature maps produced by the lower-level kernels.

ConFES is motivated by the assumption that when each extractor is fine-tuned on the support set, it undergoes gradual changes between iterations, and the logits output by sequentially saved snapshots can be considered a time series. Therefore, 1D convolutions can be used to discern informative patterns in the time series data and compute feature maps, which are smaller in size than the raw logit time series, and therefore require fewer parameters in the global kernel than standard FES.

Given K extractors and J snapshots for each extractor, FES requires $K \times J$ parameters. Assuming a two-level ConFES hierarchy, with a base-level convolutional kernel of size J_b and stride T, the feature map for each extractor will be of length $J_m = \frac{J-J_b}{T} + 1$, leading to a global kernel size of $K \times (\frac{J-J_b}{T} + 1)$. Including the $K \times J_b$ parameters in the convolutional kernel, ConFES contains

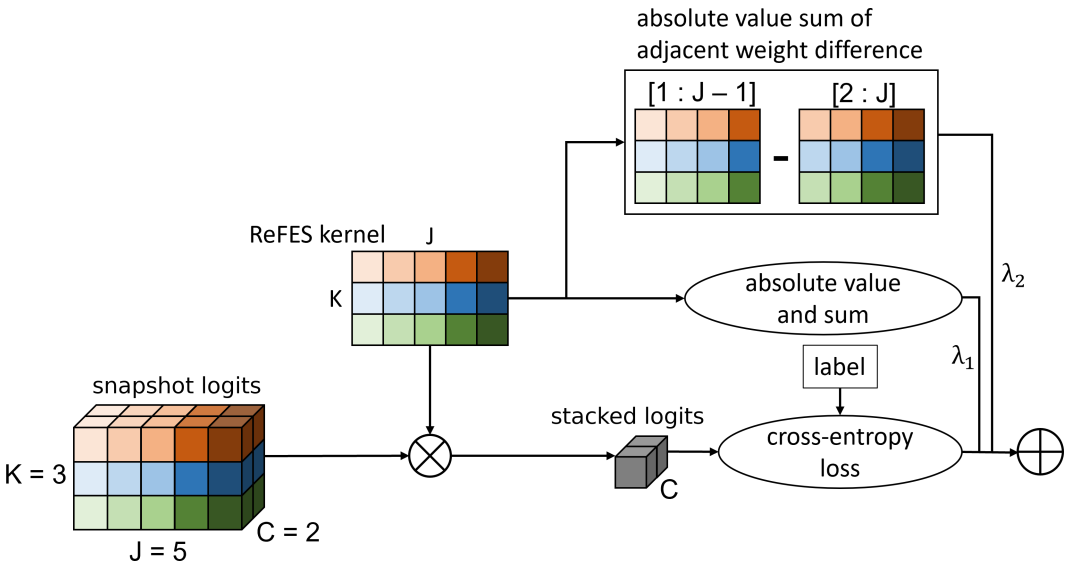


Figure 3.4: ReFES uses the same global kernel as FES and applies fused lasso regularisation to the kernel's training process. Fused lasso drives each individual weight towards zero with a regularisation strength of λ_1 and applies depthwise smoothing to the weight matrix by penalising the weight difference between adjacent snapshots with a regularisation strength of λ_2.

$K \times (\frac{J-J_b}{T} + 1 + J_b)$ parameters. In practice, it can generally be assumed that $J \gg 1$: a two-level ConFES architecture should be configured so that $J \gg J_b \geq T \gg 1$ in order to cover all snapshots with significantly fewer parameters than FES.

ConFES utilises the sequential relation of each extractor's snapshots through its lower-level 1D depthwise convolutional layers and exhibits substantially fewer parameters than FES, making it less prone to overfitting. Note that Figure 3.3 is simplified for demonstration purposes and does not reflect well that ConFES maintains fewer parameters; for a practical example of ConFES kernels, please refer to Figure 3.6.

3.2.4 Regularised Feature Extractor Stacking

To combat overfitting, an alternative to reducing the number of parameters is to perform regularisation. Regularised FES (ReFES) introduces fused lasso regularisation (Tibshirani et al., 2005) to the stacking classifier used in FES, as shown in Figure 3.4. Non-zero weights are penalised with a strength of λ_1,

and each feature-extractor-wise weight sequence is smoothed with a strength of λ_2. The loss is a combination of cross-entropy loss and depthwise fused lasso loss, as formulated in Equation 3.7, given K extractors, J snapshots per extractor, and a 2D global kernel W of shape $K \times J$.

$$\ell = \ell_{\text{cross-entropy}} + \lambda_1 \sum_{k=1}^{K} \sum_{j=1}^{J} \|W_k^j\| + \lambda_2 \sum_{k=1}^{K} \sum_{j=1}^{J-1} \|W_k^j - W_k^{j+1}\|. \tag{3.7}$$

In addition to encouraging sparse weights like standard lasso to improve robustness by reducing the number of contributing snapshots, fused lasso also encourages smaller differences between adjacent weights (Tibshirani et al., 2005). Each extractor's snapshots are ordered by their fine-tuning iterations, and adjacent snapshots are likely to be similar. By applying fused lasso regularisation, differences between adjacent weights are penalised, and weight sequences are smoothed.

The stratified two-fold splits S_1 and S_2 can be used to select appropriate λ_1 and λ_2 values for a few-shot episode. In the spirit of grid search with cross-validation, a ReFES stacking classifier is trained on the logits of one split, for example, $L_K^J[S_1]$, and tested on the logits of the other split, for example, $L_K^J[S_2]$. Different values for λ_1 and λ_2 can be explored and the best configuration selected based on the combined accuracy on the two folds. This configuration is then used to train a newly initialised ReFES stacking classifier on the full set of cross-validation logits $L_K^J[CV]$, and this stacking classifier is used to label the query set instances Q based on their logits $L_K^J[Q]$.

3.2.5 Handling Single-instance Classes

Meta-Dataset's sampling scheme (Triantafillou et al., 2020b) sometimes produces support sets containing single-instance classes. During cross-validation, single-instance classes need to be removed: if a class' only instance is in the test split S^{test}, then the training split S^{train} will have no instance of that class. FES and its variants can train their stacking classifiers on a subset of the support classes $C_{sub} \leq C$, because their kernels only encode the weights of

the snapshots, and are inherently independent of the number of classes C. In Figures 3.2, 3.3, and 3.4, C can simply be replaced by C_{sub} during training.

Given a strict one-shot problem, where all classes exhibit exactly one instance, FES cross-validation is infeasible, as all classes need to be removed during cross-validation, leading to $L_K^J[CV] = \varnothing$. Therefore, support logits obtained from ordinary fine-tuning need to be used in place of cross-validation logits, i.e., $L_K^J[S]$ is used to train the FES classifier W instead of using $L_K^J[CV]$.

3.3 Experimental Setup

To evaluate FES and its variants on the Meta-Dataset benchmark described in Section 2.2, we use an extractor collection containing eight extractors, each independently pretrained on a Meta-Dataset source domain. In our primary set of experiments, all extractors are ResNet18 models (He et al., 2016) and identical to the source domain extractors used in the publication introducing URL (Li et al., 2021). Note that the extractors are trained on the training split of the source domain data only. The source domain validation split is used to select a trained checkpoint.

FES is compatible with any fine-tuning algorithm that is applicable to the individual extractors. In our experiments, we save a snapshot of each extractor before fine-tuning and save a snapshot after each iteration. We evaluate FES with three fine-tuning methods used by state-of-the-art CDFSL methods in the literature and discussed in Sections 2.3.3.3, 2.3.3.2, and 2.3.3.1 of this thesis respectively:

- TSA (Li et al., 2022a)—matrix residual adaptors attached to convolutional layers, and a fully-connected layer to project feature vectors.
- URL (Li et al., 2021)—only a fully-connected layer to project feature vectors.
- FLUTE (Triantafillou et al., 2021)—scaling and shift factors of batch

normalisation layers.

When performing each fine-tuning method for FES, we use the hyperparameters as stated in the source publications, including optimiser type, learning rate, number of iterations, etc., and we compare FES to each source method. The URL (Li et al., 2021) and TSA (Li et al., 2022a) papers fine-tune their feature extractors for 40 iterations, leading to 41 FES snapshots per extractor, and they use cosine similarity scores scaled by a factor of 10 as logits. The FLUTE (Triantafillou et al., 2021) paper fine-tunes its feature extractor for six iterations, leading to seven FES snapshots per extractor, and it uses cosine similarity scores without scaling as logits.

We adhere to the TSA, URL, and FLUTE papers when replicating and evaluating their methods as benchmarks. Pretrained universal extractors are obtained from the official repositories, and hyperparameter settings are consistent with the papers' specifications. Note that both the URL and TSA papers used the same URL-distilled universal extractor, and their difference is in fine-tuning. URL only fine-tunes a feature projection, while TSA additionally fine-tunes convolutional channel projections.

We use an LBFGS optimiser to train the stacking classifier, applying the optimiser's default hyperparameters in the PyTorch library (Paszke et al., 2019), except that we utilise its line search function. A ridge regularisation of strength $1e^{-2}$ is applied to FES and ConFES to make the LBFGS optimiser more numerically stable. Adjusting the regularisation strength up or down by an order of magnitude does not substantially affect classification accuracy.

Meta-Dataset's sampling randomness may cause one or two percent accuracy fluctuation of evaluated methods between different runs, as also stated in URL and TSA's code repositories (Li et al., 2022b). This fluctuation may exceed the 95% confidence interval of most results, so to eliminate it, we sample 600 episodes from each domain once in Meta-Dataset. The sampled episodes are cached and then used to evaluate all CDFSL methods. In a dataset, the numbers of classes and instances are randomly sampled for each episode,

Table 3.1: Meta-Dataset episode statistics

	support size			class count			mean shot		
Datasets	min	mean	max	min	mean	max	min	mean	max
ilsvrc_2012	8	380.28	498	6	15.13	50	1	33.53	83.00
omniglot	5	94.38	378	5	19.11	47	1	4.86	9.20
aircraft	5	333.92	497	5	10.04	15	1	35.05	76.60
cu_birds	8	318.22	494	5	17.46	30	1	19.69	46.20
dtd	5	290.94	498	5	6.00	7	1	48.64	97.20
quickdraw	9	410.91	497	5	27.47	50	1	19.83	84.60
fungi	6	350.79	494	5	26.38	50	1	16.31	65.43
vgg_flower	7	290.72	497	5	10.55	16	1	28.36	73.80
traffic_sign	11	416.66	497	5	24.54	43	1	22.03	98.40
mscoco	9	418.97	498	5	23.07	40	1	23.10	96.60
mnist	5	325.46	498	5	7.52	10	1	44.33	99.60
cifar10	7	318.78	498	5	7.47	10	1	44.16	99.40
cifar100	9	409.28	497	5	27.32	50	1	19.74	84.60
CropDisease	8	425.02	498	5	21.77	38	1	24.30	95.40
EuroSAT	5	332.65	498	5	7.54	10	1	45.39	99.60
ISIC	5	282.67	498	5	6.01	7	1	47.45	99.60
ChestX	5	280.74	498	5	5.91	7	1	47.54	99.40
Food101	7	420.13	498	5	26.99	50	1	20.53	98.40

which means that different episodes can contain different numbers of classes and instances. In an episode, the number of instances is randomly sampled for each class, which means that different classes can contain different numbers of instances, and episodes can be class-imbalanced. However, the query set is stratified and always contains 10 instances per class.

Triantafillou et al. (2021) pointed out that Meta-Dataset instances need to be shuffled during sampling in case of datasets with particular ordering, for example, traffic_sign contains consecutive frames from the same video, but their shuffling solution was implemented as a moving window of size 1,000 for streams of instances of each class, which we found to be potentially insufficient, leading to approximately 1% better accuracy in mscoco and 3% better accuracy in ChestX than true random sampling. We found that a window size of 10,000 yielded virtually the same level of accuracy as true random sampling, but nevertheless use true random sampling in our experiments. Instances in each class are fully randomised and have equal chance of being selected, and episodes

are completely independent of each other. Statistics of our sampling run are shown in Table 3.1. Using exactly the same sampled episodes for each learning scheme compared also enables us to perform a paired t-test on a per-dataset basis as a more sensitive statistical difference test than simply comparing two algorithms' mean accuracy and confidence intervals.

Considering the complexity of the optimisation problem when learning the stacking classifier, it is worth noting that the FES and ReFES stacking classifiers each maintain $8 \times 41 = 328$ parameters if the extractors are fine-tuned for 40 iterations, and $8 \times 7 = 56$ parameters if the extractors are fine-tuned for 6 iterations.

ConFES is applied with a two-level hierarchy comprising a low-level depth-wise 1D convolutional kernel and a high-level global kernel. For 40-iteration fine-tuning, the convolutional kernel has size $L = 9$ with stride $T = 4$, leading to a feature sequence/global kernel of length 9. Consequently, ConFES has $8 \times 9 + 8 \times 9 = 144$ parameters in total. For 6-iteration fine-tuning, the convolutional kernel has size 3 with stride 2, leading to a global kernel of length 3, and therefore ConFES contains $8 \times 3 + 8 \times 3 = 48$ parameters in total. All parameters are initialised with a constant $(1\mathrm{e}^{-3})^{\frac{1}{h}}$, where h is the number of hierarchical levels in the stacking classifier. Therefore, FES and ReFES are initialised with $1\mathrm{e}^{-3}$, and a two-level ConFES hierarchy is initialised with $(1\mathrm{e}^{-3})^{\frac{1}{2}}$. This initialisation is deterministic and ensures that the product of weights from all levels is close to $1\mathrm{e}^{-3}$, which is small enough for optimisation to go in either direction, but also big enough to avoid exceedingly small derivatives in gradient-based optimisers.

To facilitate grid search for the λ_1 and λ_2 values of ReFES, a pool of eight potential values is provided for each hyperparameter: 1, $1\mathrm{e}^{-1}$, $1\mathrm{e}^{-2}$, $1\mathrm{e}^{-3}$, $1\mathrm{e}^{-4}$, $1\mathrm{e}^{-5}$, $1\mathrm{e}^{-6}$, and 0.

3.4 Results

We present CDFSL results of FES, ConFES, ReFES, and the competing methods URL, FLUTE, and a URL extractor with TSA fine-tuning, on the Meta-Dataset benchmark, and show that FES and its variants advance the state of the art on this benchmark. We then visually analyse an example of trained FES, ConFES, and ReFES kernels. Lastly, we examine the ability of FES, ConFES, and ReFES to omit snapshots with their non-negative kernels.

3.4.1 Meta-Dataset Results

Results are organised by fine-tuning algorithms used, to provide a comparison between different CDFSL algorithms with the same fine-tuning scheme. The universal model of URL (Li et al., 2021), applied with TSA fine-tuning (Li et al., 2022a), is the most recent and strongest CDFSL approach in the literature. Hence, we compare to this universal-model approach first, applying TSA fine-tuning in our FES methods as well in this comparison. Following that, we present experiments with the simpler fine-tuning approach used in the original URL (Li et al., 2021) paper. Finally, we evaluate FLUTE (Triantafillou et al., 2021) fine-tuning, which fine-tunes batch norm parameters only, and compare to the FLUTE universal template model.

Results with TSA fine-tuning are shown in Table 3.2, and paired t-test results based on the 600 individual accuracy values per dataset are shown in Table 3.3. Results with URL fine-tuning are shown in Tables 3.4 and 3.5, and those with FLUTE fine-tuning are shown in Tables 3.6 and 3.7.

In these tables, mean accuracy over 600 episodes and 95% confidence intervals are shown for each algorithm and dataset, and weak and strong generalisation accuracy and ranks averaged over all individual episodes are listed below the datasets, respectively as "WG acc", "WG rank", "SG acc", and "SG rank". The best result of each row is shown in **bold**. If a paired t-test between a FES algorithm and the corresponding universal model/template (in

Table 3.2: Meta-Dataset results with TSA fine-tuning

Dataset	TSA	FES	ConFES	ReFES
ilsvrc_2012	**56.8±1.1**	56.2±1.1 •	56.3±1.1 •	56.2±1.2 •
omniglot	**95.0±0.4**	93.3±0.6 •	93.3±0.7 •	93.6±0.6 •
aircraft	**88.4±0.5**	87.6±0.8 •	87.9±0.7 •	87.9±0.8 •
cu_birds	**81.5±0.7**	79.9±0.8 •	80.0±0.8 •	79.8±0.9 •
dtd	**77.1±0.7**	76.2±0.8 •	76.3±0.8 •	76.4±0.8 •
quickdraw	82.0±0.6	83.4±0.6 ○	**83.5±0.6** ○	83.4±0.6 ○
fungi	68.3±1.1	69.4±1.1 ○	**69.7±1.1** ○	69.6±1.1 ○
vgg_flower	**92.1±0.5**	91.9±0.7	91.9±0.7	**92.1±0.6**
WG acc	**80.15**	79.74	79.86	79.88
WG rank	2.58	2.50	**2.45**	2.47
traffic_sign	82.8±0.9	84.9±1.0 ○	85.1±1.0 ○	**85.2±1.0** ○
mscoco	53.8±1.1	54.1±1.0 ○	**54.4±1.0** ○	**54.4±1.0** ○
mnist	96.6±0.4	97.1±0.5 ○	97.1±0.5 ○	**97.2±0.5** ○
cifar10	**79.9±0.8**	78.1±0.9 •	78.2±0.9 •	78.8±0.9 •
cifar100	70.3±1.0	70.4±1.1	70.6±1.0 ○	**70.8±1.0** ○
CropDisease	84.4±0.8	88.1±0.7 ○	**88.3±0.7** ○	**88.3±0.7** ○
EuroSAT	**89.6±0.5**	88.8±0.6 •	89.1±0.6 •	89.3±0.6
ISIC	48.4±0.9	**49.5±0.9** ○	49.3±0.9 ○	48.9±0.9 ○
ChestX	27.2±0.6	27.7±0.6 ○	27.7±0.6 ○	**27.8±0.6** ○
Food101	53.4±1.2	55.2±1.1 ○	**55.5±1.1** ○	55.2±1.1 ○
SG acc	68.64	69.39	69.53	**69.59**
SG rank	2.85	2.49	2.34	**2.32**

Table 3.3: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results with TSA fine-tuning

	WG				SG			
	TSA	FES	ConFES	ReFES	TSA	FES	ConFES	ReFES
TSA	-	2	2	2	-	7	8	8
FES	5	-	4	3	2	-	6	6
ConFES	5	0	-	2	2	0	-	3
ReFES	5	1	1	-	1	1	2	-

Table 3.4: Meta-Dataset results with URL fine-tuning

Dataset	URL	FES	ConFES	ReFES
ilsvrc_2012	**56.6±1.1**	56.1±1.1 •	56.1±1.1 •	55.9±1.2 •
omniglot	**94.5±0.4**	93.1±0.6 •	93.4±0.6 •	93.5±0.6 •
aircraft	**87.7±0.5**	87.1±0.8 •	87.5±0.7	87.5±0.7
cu_birds	**80.7±0.7**	79.1±0.8 •	79.2±0.8 •	79.0±0.8 •
dtd	**76.1±0.6**	75.0±0.8 •	75.1±0.8 •	74.9±0.8 •
quickdraw	82.0±0.6	**83.2±0.6** ◦	**83.2±0.6** ◦	83.1±0.6 ◦
fungi	69.5±1.1	69.6±1.1	**69.8±1.1**	69.6±1.1
vgg_flower	**91.4±0.5**	90.7±0.7 •	90.5±0.7 •	90.8±0.6 •
WG acc	**79.81**	79.24	79.35	79.29
WG rank	2.52	2.50	**2.47**	2.51
traffic_sign	62.6±1.2	65.2±1.2 ◦	**65.3±1.2** ◦	65.0±1.2 ◦
mscoco	52.7±1.0	52.6±1.0	52.7±1.0	**52.8±1.0**
mnist	94.6±0.4	96.3±0.5 ◦	96.4±0.5 ◦	**96.5±0.5** ◦
cifar10	71.4±0.8	71.7±0.8	**71.9±0.8**	**71.9±0.8**
cifar100	62.6±1.1	62.7±1.1	62.7±1.1	**62.8±1.1**
CropDisease	80.5±0.8	87.2±0.7 ◦	87.2±0.7 ◦	**87.4±0.7** ◦
EuroSAT	**86.6±0.5**	86.1±0.6 •	86.1±0.6 •	86.3±0.6
ISIC	45.5±0.8	48.4±0.9 ◦	48.3±0.9 ◦	**48.9±0.9** ◦
ChestX	26.5±0.6	**27.2±0.6** ◦	27.1±0.6 ◦	27.1±0.6 ◦
Food101	51.9±1.1	**53.9±1.1** ◦	**53.9±1.1** ◦	**53.9±1.1** ◦
SG acc	63.49	65.13	65.16	**65.26**
SG rank	2.96	2.40	2.34	**2.31**

Table 3.5: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results with URL fine-tuning

	WG				SG			
	URL	FES	ConFES	ReFES	URL	FES	ConFES	ReFES
URL	-	1	1	1	-	6	6	6
FES	6	-	3	2	1	-	3	5
ConFES	5	1	-	1	1	0	-	3
ReFES	5	3	3	-	0	1	1	-

Table 3.6: Meta-Dataset results with FLUTE fine-tuning

Dataset	FLUTE	FES	ConFES	ReFES
ilsvrc_2012	50.2±1.1	**54.1±1.1** ∘	**54.1±1.1** ∘	53.9±1.2 ∘
omniglot	93.9±0.5	94.3±0.6 ∘	94.8±0.5 ∘	**94.9±0.5** ∘
aircraft	86.8±0.6	**87.4±0.7** ∘	87.1±0.9	**87.4±0.7** ∘
cu_birds	**79.3±0.8**	78.4±0.9 •	78.5±0.9 •	78.3±0.9 •
dtd	68.8±0.8	**74.3±0.9** ∘	**74.3±0.8** ∘	74.2±0.9 ∘
quickdraw	79.1±0.7	82.9±0.6 ∘	**83.0±0.6** ∘	82.7±0.6 ∘
fungi	59.4±1.2	68.7±1.1 ∘	**69.1±1.1** ∘	68.8±1.1 ∘
vgg_flower	91.0±0.6	92.5±0.6 ∘	92.5±0.6 ∘	**92.6±0.6** ∘
WG acc	76.06	79.08	**79.18**	79.10
WG rank	3.24	2.27	**2.22**	2.27
traffic_sign	57.9±1.1	72.6±1.1 ∘	**72.8±1.1** ∘	72.3±1.1 ∘
mscoco	48.2±1.0	51.7±1.1 ∘	**51.8±1.1** ∘	**51.8±1.1** ∘
mnist	95.7±0.4	97.4±0.4 ∘	**97.6±0.3** ∘	97.5±0.4 ∘
cifar10	**78.6±0.7**	75.2±0.9 •	75.2±0.9 •	75.1±0.9 •
cifar100	**67.5±1.0**	67.2±1.1 •	67.2±1.0	67.0±1.1 •
CropDisease	78.0±0.8	**86.5±0.7** ∘	86.4±0.7 ∘	**86.5±0.7** ∘
EuroSAT	81.6±0.6	**88.3±0.6** ∘	**88.3±0.6** ∘	88.0±0.6 ∘
ISIC	46.1±1.0	**48.7±1.0** ∘	**48.7±1.0** ∘	48.1±0.9 ∘
ChestX	26.3±0.5	**27.8±0.6** ∘	27.7±0.6 ∘	**27.8±0.6** ∘
Food101	45.7±1.1	**52.1±1.1** ∘	52.0±1.1 ∘	52.0±1.1 ∘
SG acc	62.56	66.75	**66.77**	66.61
SG rank	3.24	2.23	**2.21**	2.32

Table 3.7: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results with FLUTE fine-tuning

	WG				SG			
	FLUTE	FES	ConFES	ReFES	FLUTE	FES	ConFES	ReFES
FLUTE	-	7	6	6	-	7	8	8
FES	1	-	7	5	3	-	9	8
ConFES	1	0	-	0	2	0	-	1
ReFES	1	0	7	-	2	1	6	-

the leftmost column) returns a p value less than 0.05, the null hypothesis (that there is no statistically significant difference) is rejected, and the FES result is marked with either $\circ$ if it has higher accuracy, or $\bullet$ if its competitor has higher accuracy.

The tables showing paired t-test results are split by weak generalisation (the eight source domains) and strong generalisation (the ten target domains). Each value indicates the number of datasets where the algorithm in the value's column significantly outperforms the algorithm in its row according to the paired t-test.

When using the same fine-tuning scheme, FES and its variants outperform their competitor CDFSL algorithms—building a universal model using knowledge distillation for URL and its TSA fine-tuning variant, and training a universal template with FiLM layers for FLUTE—in strong generalisation, where learning problems qualify as being cross-domain. The FES algorithms achieve better average accuracy and obtain more wins than losses in paired t-tests. They also rank higher than their competitors based on average rank.

Considering results with all three fine-tuning methods, the FES algorithms consistently outperform their competitors by a substantial margin on traffic_sign, CropDisease, and Food101, while being outperformed on cifar10 and cifar100. This phenomenon may indicate that FES and its variants perform better in domains that are more specialised, while their competitors gain an edge on datasets more similar to ImageNet, such as the CIFAR datasets. This speculation is supported by the fact that the competitor methods artificially attach greater importance to ImageNet when their universal models are obtained (Triantafillou et al., 2021; Li et al., 2021, 2022a).

All three FES variants exhibit good CDFSL performance. Which variant is to be preferred depends on each specific use case: FES is the simplest and most versatile; ConFES maintains a smaller number of parameters and therefore a more manipulable search space; and ReFES uses regularisation to achieve smoother and more interpretable snapshot selections.

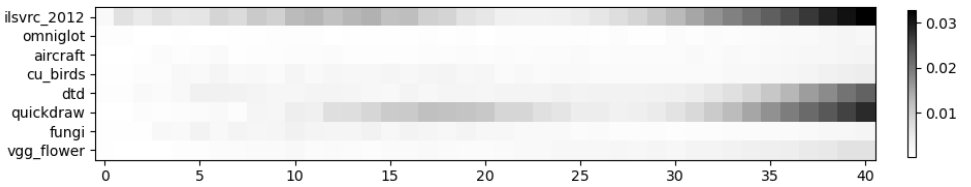
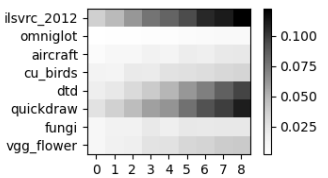
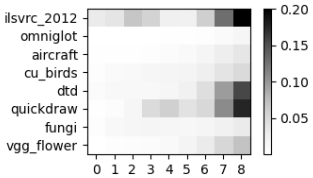


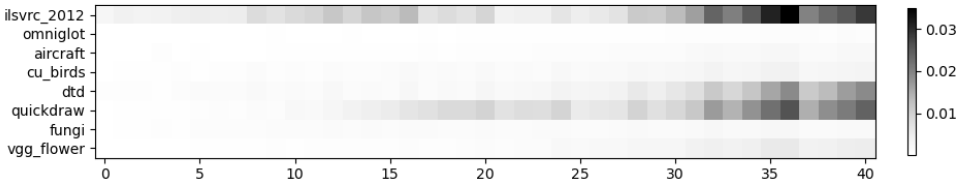
Figure 3.5: FES kernel for traffic_sign



(a) ConFES depthwise kernel



(b) ConFES global kernel



(c) ConFES expanded kernel

Figure 3.6: ConFES kernels for traffic_sign

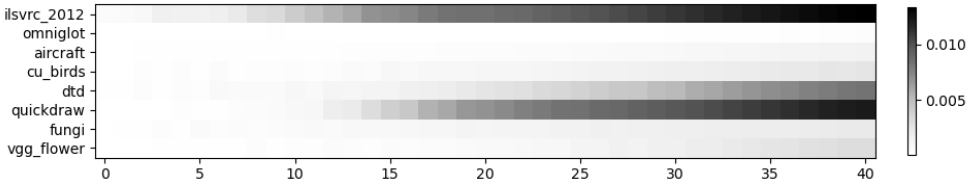


Figure 3.7: ReFES kernel for traffic_sign

3.4.2 Weight Visualisation

Weights of the FES, ConFES, and ReFES kernels after fine-tuning with TSA on traffic_sign are visualised in Figures 3.5, 3.6, and 3.7. The weights are averaged over 600 episodes.

ConFES maintains two kernels: a low-level depthwise 1D convolutional kernel (3.6a) and a high-level global kernel (3.6b). The two kernels can be expanded back into a global kernel (3.6c) for interpretation because the output of the convolutional kernel 3.6a serves as direct input to the global kernel 3.6b, without any intermediate non-linear activation. Figure 3.6 demonstrates how ConFES emulates a 328-parameter FES kernel with only 144 parameters. The stepped pattern in the expanded ConFES kernel, where every fourth snapshot is assigned relatively greater weight than its neighbours, is an artefact of 1D convolution—with a kernel size of 9 and a stride size of 4, this pattern results from kernel overlaps.

FES determines that the fine-tuned ilsvrc_2012 (ImageNet) and quickdraw extractors are the most prominent contributors to its predictions, indicated by the dark regions on the right end of these two extractors' rows in Figure 3.5. ConFES and ReFES arrive at similar conclusions regarding contributors, but exhibit characteristics that reflect their classifiers' behaviours: ConFES shows stepped patterns due to 1D convolution as in Figure 3.6; ReFES shows smoother weight changes due to fused lasso regularisation as in Figure 3.7.

Additional heatmaps visualising kernel weights on the other target domains are presented in Figures A.1-A.9 in Appendix A.

3.4.3 Snapshot Omission

As FES kernels are constrained to be non-negative by clipping their weights with ReLU, some snapshots may have their corresponding weights set to 0 after clipping, which means logits from these snapshots do not contribute to the aggregated meta logits, and these snapshots can be omitted. They do not need to be saved and are not used for inference.

Table 3.8: Percentage of snapshots omitted by the stacking classifier

	FES			ConFES			ReFES		
Dataset	TSA	URL	FLUTE	TSA	URL	FLUTE	TSA	URL	FLUTE
ilsvrc_2012	72.2	72.4	65.9	58.1	56.0	33.5	38.2	38.3	40.0
omniglot	51.2	51.0	45.5	42.7	43.5	36.2	22.1	27.0	24.0
aircraft	71.0	72.5	58.6	50.6	48.3	32.5	37.0	37.1	30.7
cu_birds	73.2	71.3	68.3	62.5	58.4	48.8	46.6	45.2	41.8
dtd	68.6	72.5	64.8	53.4	56.6	34.5	30.7	39.5	28.7
quickdraw	69.0	70.0	69.1	58.6	57.7	34.5	31.7	36.5	42.3
fungi	73.2	74.1	71.9	65.3	64.2	46.8	46.3	46.5	42.5
vgg_flower	64.2	66.3	46.7	40.2	42.5	26.3	24.4	28.6	24.3
traffic_sign	75.6	79.1	81.6	61.9	73.3	74.5	29.7	55.6	70.5
mscoco	79.2	76.8	70.2	60.6	55.3	38.1	35.0	34.8	39.4
mnist	62.5	68.6	51.4	39.3	44.9	34.8	19.4	21.0	27.2
cifar10	78.6	80.3	68.0	67.4	65.1	37.3	37.4	35.3	39.9
cifar100	78.1	74.3	65.0	61.0	56.8	28.8	35.4	32.9	40.3
CropDisease	73.4	73.7	52.2	51.2	48.8	20.6	23.9	23.1	26.2
EuroSAT	75.6	74.8	61.6	62.4	63.3	32.1	28.2	27.3	29.6
ISIC	70.1	72.1	67.0	58.6	62.1	40.0	25.7	33.7	25.6
ChestX	85.2	82.9	77.3	75.4	67.8	44.8	44.3	44.2	42.2
Food101	80.6	76.5	61.4	62.9	53.7	26.9	53.6	37.4	36.9

Table 3.8 shows the average percentage of snapshots omitted by a FES, ConFES, or ReFES stacking classifier, using TSA, URL, or FLUTE fine-tuning. Note that ConFES omission rates are computed using the expanded kernel, because zero values need to exist in the expanded kernel, instead of merely in one of ConFES' hierarchical kernels, for the corresponding snapshots to be omitted. A higher omission percentage is considered better because omitting snapshots saves storage space and inference computation. Among the three methods, FES achieves the highest percentage of omission, generally between 60% and 80%, followed by ConFES, which achieves 30% to 70% omission in general, while ReFES achieves the least amount of omission, mostly below 40%. FES achieves higher omission rates than ConFES and ReFES, but trades off mean strong generalisation accuracy as shown in Tables 3.2, 3.4, and 3.6.

Table 3.9: Ablation results with TSA fine-tuning

	FES without cross-validation			FES		
Dataset	first	last	all	first	last	all
ilsvrc_2012	52.3±1.2	52.7±1.2	53.5±1.2	54.5±1.1	**56.4±1.2**	56.2±1.1
omniglot	**94.5±0.5**	90.2±0.7	90.6±0.7	**94.5±0.5**	93.3±0.7	93.3±0.6
aircraft	85.7±0.7	78.5±1.1	78.1±1.1	87.1±0.6	**87.7±0.8**	87.6±0.8
cu_birds	75.9±0.9	73.1±1.1	74.2±1.1	78.9±0.8	79.8±0.9	**79.9±0.8**
dtd	72.2±0.8	75.5±0.9	75.9±0.8	72.9±0.8	**76.8±0.8**	76.2±0.8
quickdraw	82.8±0.6	81.9±0.8	82.5±0.7	83.0±0.6	**83.4±0.6**	**83.4±0.6**
fungi	64.6±1.2	57.8±1.3	59.1±1.3	69.2±1.1	69.2±1.1	**69.4±1.1**
vgg_flower	90.0±0.6	89.9±0.8	90.1±0.7	90.3±0.6	**92.1±0.7**	91.9±0.7
WG acc	77.25	74.95	75.50	78.80	**79.84**	79.74
traffic_sign	48.9±1.1	84.8±1.0	85.0±1.0	49.2±1.1	**85.8±0.9**	84.9±1.0
mscoco	45.9±1.1	49.8±1.0	51.9±1.0	48.3±1.0	53.6±1.0	**54.1±1.0**
mnist	95.8±0.4	96.5±0.5	96.7±0.5	95.6±0.5	**97.2±0.5**	97.1±0.5
cifar10	66.9±0.9	72.2±1.0	74.8±1.0	68.6±0.8	**78.6±0.9**	78.1±0.9
cifar100	57.0±1.1	65.2±1.1	68.0±1.1	59.1±1.1	**70.7±1.0**	70.4±1.1
CropDisease	81.9±0.8	87.0±0.7	87.8±0.7	82.6±0.7	88.0±0.7	**88.1±0.7**
EuroSAT	81.3±0.6	87.7±0.7	88.5±0.6	81.5±0.6	**89.5±0.6**	88.8±0.6
ISIC	46.2±0.8	47.0±0.9	47.8±0.9	46.4±0.8	47.9±0.9	**49.5±0.9**
ChestX	25.0±0.5	27.2±0.6	27.6±0.6	24.7±0.5	27.2±0.6	**27.7±0.6**
Food101	49.4±1.2	50.9±1.2	51.9±1.2	52.3±1.1	54.5±1.1	**55.2±1.1**
SG acc	59.83	66.83	68.00	60.83	69.30	**69.39**

3.5 Ablation Study

We perform an ablation study by removing cross-validation from the framework and/or using only the first or last snapshots in fine-tuning. When cross-validation is not used, training logits for the "stacking" classifier are extracted from the support set using snapshots fine-tuned on the entire support set, akin to how one-shot episodes are handled in Section 3.2.5. When using only the first or last snapshots, the stacking classifier is a degenerate weight kernel with a singleton dimension for fine-tuning iterations, simply containing one weight value for each extractor. Results are shown in Tables 3.9, 3.10, and 3.11, organised by the fine-tuning algorithm used.

The results show that methods using cross-validation outperform their counterparts without cross-validation. Moreover, using all snapshots achieves better performance than using only the first or last snapshots in terms of mean strong generalisation performance for URL and TSA fine-tuning. For strong

Table 3.10: Ablation results with URL fine-tuning

	FES without cross-validation			FES		
Dataset	first	last	all	first	last	all
ilsvrc_2012	52.3±1.2	52.6±1.2	52.8±1.2	54.5±1.1	**56.3±1.2**	56.1±1.1
omniglot	**94.5±0.5**	90.1±0.7	90.8±0.7	**94.5±0.5**	93.0±0.7	93.1±0.6
aircraft	85.7±0.7	82.0±1.1	82.3±1.0	87.1±0.6	**87.3±0.8**	87.1±0.8
cu_birds	75.9±0.9	73.8±1.0	74.4±1.0	78.9±0.8	79.0±0.9	**79.1±0.8**
dtd	72.2±0.8	73.9±0.8	73.7±0.8	72.9±0.8	**75.4±0.8**	75.0±0.8
quickdraw	82.8±0.6	82.1±0.7	82.3±0.7	83.0±0.6	83.1±0.6	**83.2±0.6**
fungi	64.6±1.2	62.2±1.3	63.0±1.3	69.2±1.1	69.5±1.1	**69.6±1.1**
vgg_flower	90.0±0.6	89.7±0.8	89.5±0.7	90.3±0.6	**90.9±0.7**	90.7±0.7
WG acc	77.25	75.80	76.10	78.80	**79.31**	79.24
traffic_sign	48.9±1.1	65.0±1.2	65.2±1.2	49.2±1.1	**65.4±1.2**	65.2±1.2
mscoco	45.9±1.1	49.3±1.1	49.9±1.1	48.3±1.0	51.2±1.0	**52.6±1.0**
mnist	95.8±0.4	96.3±0.5	96.5±0.5	95.6±0.5	**96.4±0.5**	96.3±0.5
cifar10	66.9±0.9	69.8±0.9	70.6±0.9	68.6±0.8	71.1±0.8	**71.7±0.8**
cifar100	57.0±1.1	60.8±1.2	61.6±1.2	59.1±1.1	**62.8±1.1**	62.7±1.1
CropDisease	81.9±0.8	86.6±0.7	86.7±0.7	82.6±0.7	87.0±0.7	**87.2±0.7**
EuroSAT	81.3±0.6	86.1±0.6	**86.3±0.6**	81.5±0.6	86.0±0.6	86.1±0.6
ISIC	46.2±0.8	46.7±0.9	45.7±0.9	46.4±0.8	47.6±0.9	**48.4±0.9**
ChestX	25.0±0.5	27.3±0.6	**27.4±0.6**	24.7±0.5	26.4±0.6	27.2±0.6
Food101	49.4±1.2	49.5±1.2	50.2±1.2	52.3±1.1	53.1±1.1	**53.9±1.1**
SG acc	59.83	63.74	64.01	60.83	64.70	**65.13**

generalisation with FLUTE fine-tuning, using only the last snapshots leads to better performance. This could be due to the smaller number of fine-tuning iterations performed by FLUTE, as the last snapshots constitute a more substantial part of all snapshots. It is worth noting that cross-validation is helpful even when only using the first snapshots before any fine-tuning because the training logits are computed using a nearest centroid classifier, and cross-validation keeps the support instances for logit extraction separate from those used to compute the centroids, hence avoiding instance re-use and reducing overfitting.

3.6 Heterogeneous Extractors

FES and its variants operate in logit space, which means they are independent of the architecture and feature size of each extractor. Therefore, they can

Table 3.11: Ablation results with FLUTE fine-tuning

	FES without cross-validation			FES		
Dataset	first	last	all	first	last	all
ilsvrc_2012	49.9±1.2	50.5±1.2	50.6±1.2	53.2±1.1	53.5±1.2	**54.1±1.1**
omniglot	93.0±0.6	93.2±0.6	93.3±0.6	94.1±0.5	94.2±0.6	**94.3±0.6**
aircraft	83.7±0.9	83.1±1.0	83.2±1.0	86.9±0.7	87.3±0.8	**87.4±0.7**
cu_birds	72.7±1.0	73.4±1.1	73.7±1.1	76.8±0.9	78.3±0.9	**78.4±0.9**
dtd	72.0±0.8	73.9±0.9	73.9±0.9	72.5±0.8	**74.5±0.9**	74.3±0.9
quickdraw	81.2±0.7	81.2±0.7	81.4±0.7	82.7±0.6	81.8±0.6	**82.9±0.6**
fungi	60.4±1.3	59.0±1.3	59.3±1.3	**68.7±1.1**	67.5±1.1	**68.7±1.1**
vgg_flower	90.9±0.7	91.7±0.7	91.7±0.7	91.8±0.6	**92.6±0.6**	92.5±0.6
WG acc	75.48	75.75	75.89	78.34	78.71	**79.08**
traffic_sign	53.0±1.1	71.7±1.1	71.0±1.1	53.2±1.1	**72.9±1.1**	72.6±1.1
mscoco	44.5±1.1	49.0±1.1	49.0±1.1	47.1±1.0	51.6±1.1	**51.7±1.1**
mnist	96.0±0.4	97.1±0.5	97.1±0.5	96.2±0.4	**97.5±0.4**	97.4±0.4
cifar10	68.3±0.9	73.6±0.9	73.7±0.9	70.1±0.8	**75.4±0.9**	75.2±0.9
cifar100	59.0±1.2	64.2±1.2	64.2±1.2	61.4±1.1	**67.3±1.0**	67.2±1.1
CropDisease	83.1±0.8	86.1±0.7	85.6±0.7	84.2±0.7	**86.5±0.7**	**86.5±0.7**
EuroSAT	86.2±0.6	88.2±0.6	88.1±0.6	86.1±0.6	**88.5±0.6**	88.3±0.6
ISIC	48.2±0.9	45.0±0.9	45.0±0.9	**48.8±0.9**	**48.8±0.9**	48.7±1.0
ChestX	26.2±0.5	27.5±0.6	27.4±0.6	25.9±0.6	**27.8±0.6**	**27.8±0.6**
Food101	45.7±1.2	48.1±1.2	48.2±1.2	49.0±1.1	51.9±1.1	**52.1±1.1**
SG acc	61.02	65.05	64.93	62.20	**66.82**	66.75

naturally work with heterogeneous extractor collections. We demonstrate this by replacing the ResNet18 ImageNet extractor in the source domain collection with a more advanced Small EfficientNetV2 model (Tan and Le, 2021) pre-trained on the 21K-class version of ImageNet, while keeping the seven other source domain ResNet18 extractors unchanged. The Small EfficientNetV2 model produces feature vectors of length 1280, as opposed to feature vectors of length 512 generated by ResNet18.

URL-style fine-tuning is used: a square matrix is used for feature projection and the matrix is initialised as an identity matrix. The results are shown in Table 3.12, and are compared to results of all eight extractors being ResNet18 models. Usage of the EfficientNetV2 model consistently improves FES performance in both weak and strong generalisation. Note that the evaluation's main purpose is to show FES compatibility with heterogeneous model zoos, and its results are not directly comparable to the main results because

Table 3.12: Results of replacing the ResNet18 ImageNet extractor with a Small EfficientNetV2 pretrained on the 21K version of ImageNet, while the other seven extractors remain the same

	ResNet18			EfficientNetV2Small21K		
Dataset	FES	ConFES	ReFES	FES	ConFES	ReFES
ilsvrc_2012	56.1±1.1	56.1±1.1	55.9±1.2	63.5±1.0	**63.7±1.0**	63.1±1.1
omniglot	93.1±0.6	93.4±0.6	**93.5±0.6**	93.0±0.7	**93.5±0.6**	93.3±0.6
aircraft	87.1±0.8	87.5±0.7	87.5±0.7	87.4±0.8	**87.7±0.7**	87.5±0.8
cu_birds	79.1±0.8	79.2±0.8	79.0±0.8	80.6±0.8	**80.9±0.8**	80.4±0.9
dtd	75.0±0.8	75.1±0.8	74.9±0.8	81.5±0.8	**81.7±0.7**	81.0±0.8
quickdraw	83.2±0.6	83.2±0.6	83.1±0.6	**83.3±0.6**	**83.3±0.6**	83.2±0.6
fungi	69.6±1.1	69.8±1.1	69.6±1.1	70.0±1.1	**70.2±1.1**	70.0±1.1
vgg_flower	90.7±0.7	90.5±0.7	90.8±0.6	96.2±0.5	**96.9±0.3**	96.2±0.5
WG acc	79.24	79.35	79.29	81.94	**82.24**	81.84
traffic_sign	65.2±1.2	65.3±1.2	65.0±1.2	**65.6±1.1**	65.5±1.1	65.4±1.2
mscoco	52.6±1.0	52.7±1.0	52.8±1.0	60.7±1.0	**60.8±1.0**	60.0±1.0
mnist	96.3±0.5	96.4±0.5	96.5±0.5	96.3±0.5	96.5±0.5	**96.6±0.5**
cifar10	71.7±0.8	71.9±0.8	71.9±0.8	82.9±0.7	**83.0±0.8**	82.7±0.8
cifar100	62.7±1.1	62.7±1.1	62.8±1.1	73.9±1.0	**74.0±1.0**	73.8±1.0
CropDisease	87.2±0.7	87.2±0.7	87.4±0.7	**89.6±0.6**	89.5±0.6	89.5±0.6
EuroSAT	86.1±0.6	86.1±0.6	86.3±0.6	89.1±0.6	**89.2±0.6**	89.0±0.6
ISIC	48.4±0.9	48.3±0.9	**48.9±0.9**	48.6±0.9	48.5±0.9	**48.9±0.9**
ChestX	**27.2±0.6**	27.1±0.6	27.1±0.6	26.7±0.6	26.6±0.6	26.8±0.6
Food101	53.9±1.1	53.9±1.1	53.9±1.1	61.7±1.0	**61.8±1.0**	61.6±1.0
SG acc	65.13	65.16	65.26	69.51	**69.54**	69.43

the 21K-class ImageNet dataset used to pretrain the EfficientNetV2 model contains the Meta-Dataset ImageNet test split, which makes the ImageNet evaluation over-optimistic; moreover, test classes in the other domains may also be present in the 21K pretraining classes.

Since the EfficientNetV2 model is much more advanced than ResNet18, we investigate whether it dominates the extractor collection and effectively makes the other ResNet18 extractors irrelevant by performing FES using the single EfficientNetV2 extractor, with results in Table 3.13. Interestingly, all three FES variants obtain very similar accuracy when applied to only one EfficientNetV2 extractor, while their differences are shown more clearly when applied to a collection of eight extractors. Although using only EfficientNetV2 leads to better performance in a number of ImageNet-adjacent domains, for example, ilsvrc_2012, dtd, vgg_flower, mscoco, and cifar10, it under-performs in

Table 3.13: Comparison between applying FES to an ImageNet-pretrained EfficientNetV2 extractor alone and applying FES to an extractor collection containing it and the seven other ResNet18 source domain extractors

	EfficientNetV2 only			EfficientNetV2 and ResNet18s		
Dataset	FES	ConFES	ReFES	FES	ConFES	ReFES
ilsvrc_2012	**64.0±1.0**	**64.0±1.0**	63.9±1.0	63.5±1.0	63.7±1.0	63.1±1.1
omniglot	58.0±1.3	58.1±1.3	57.6±1.3	93.0±0.7	**93.5±0.6**	93.3±0.6
aircraft	63.9±1.0	63.8±1.0	63.6±1.0	87.4±0.8	**87.7±0.7**	87.5±0.8
cu_birds	76.0±0.8	76.0±0.8	76.0±0.8	80.6±0.8	**80.9±0.8**	80.4±0.9
dtd	**82.2±0.6**	**82.2±0.6**	82.1±0.6	81.5±0.8	81.7±0.7	81.0±0.8
quickdraw	60.0±1.0	60.0±1.0	59.8±1.0	**83.3±0.6**	**83.3±0.6**	83.2±0.6
fungi	51.0±1.2	51.0±1.2	50.9±1.2	70.0±1.1	**70.2±1.1**	70.0±1.1
vgg_flower	**97.2±0.2**	**97.2±0.2**	**97.2±0.2**	96.2±0.5	96.9±0.3	96.2±0.5
WG acc	69.04	69.04	68.89	81.94	**82.24**	81.84
traffic_sign	60.3±1.2	60.2±1.2	60.5±1.2	**65.6±1.1**	65.5±1.1	65.4±1.2
mscoco	61.1±1.0	**61.2±1.0**	60.6±1.0	60.7±1.0	60.8±1.0	60.0±1.0
mnist	87.1±0.7	87.1±0.7	87.2±0.7	96.3±0.5	96.5±0.5	**96.6±0.5**
cifar10	**83.3±0.6**	**83.3±0.6**	83.2±0.6	82.9±0.7	83.0±0.8	82.7±0.8
cifar100	73.7±0.9	73.7±0.9	73.7±0.9	73.9±1.0	**74.0±1.0**	73.8±1.0
CropDisease	85.5±0.7	85.4±0.7	85.4±0.7	**89.6±0.6**	89.5±0.6	89.5±0.6
EuroSAT	87.1±0.6	87.0±0.6	87.0±0.6	89.1±0.6	**89.2±0.6**	89.0±0.6
ISIC	46.1±0.9	46.0±0.9	46.0±0.9	48.6±0.9	48.5±0.9	**48.9±0.9**
ChestX	25.2±0.5	25.2±0.5	25.1±0.5	26.7±0.6	26.6±0.6	**26.8±0.6**
Food101	61.2±1.0	61.2±1.0	61.0±1.0	61.7±1.0	**61.8±1.0**	61.6±1.0
SG acc	67.06	67.03	66.97	69.51	**69.54**	69.43

most other domains, especially those significantly different from ImageNet, for example, omniglot, aircraft, quickdraw, fungi, traffic_sign, mnist, CropDisease, ISIC, and ChestX.

Our EfficientNetV2 evaluation indicates: 1) FES and its variants are compatible with heterogeneous extractor collections, and 2) they are robust to discrepancies in extractor architectures and able to select relevant models from a diverse model zoo.

3.7 Runtime and Memory Consumption

FES requires no universal extractor, which means the meta-training phase only requires pretraining a collection of extractors, similar to SUR. The cost for this is reduced to zero if pretrained extractors are readily available. However, FES

Table 3.14: ResNet152 feature extractors with URL fine-tuning

Dataset	URL18	MDL152	URL152	FES18	ConFES18	ReFES18
ilsvrc_2012	56.6±1.1	**59.3±1.1**	**59.3±1.1**	56.1±1.1	56.1±1.1	55.9±1.2
omniglot	94.5±0.4	94.5±0.4	**94.8±0.4**	93.1±0.6	93.4±0.6	93.5±0.6
aircraft	87.7±0.5	90.7±0.4	**91.3±0.4**	87.1±0.8	87.5±0.7	87.5±0.7
cu_birds	80.7±0.7	**85.0±0.6**	84.9±0.6	79.1±0.8	79.2±0.8	79.0±0.8
dtd	76.1±0.6	77.7±0.6	**78.3±0.6**	75.0±0.8	75.1±0.8	74.9±0.8
quickdraw	82.0±0.6	**83.3±0.6**	**83.3±0.6**	83.2±0.6	83.2±0.6	83.1±0.6
fungi	69.5±1.1	73.3±1.0	**74.9±1.0**	69.6±1.1	69.8±1.1	69.6±1.1
vgg_flower	91.4±0.5	**93.3±0.4**	91.9±0.5	90.7±0.7	90.5±0.7	90.8±0.6
WG acc	79.81	82.14	**82.34**	79.24	79.35	79.29
traffic_sign	62.6±1.2	57.2±1.2	61.9±1.2	65.2±1.2	**65.3±1.2**	65.0±1.2
mscoco	52.7±1.0	52.3±1.0	**55.1±1.0**	52.6±1.0	52.7±1.0	52.8±1.0
mnist	94.6±0.4	93.6±0.5	92.9±0.5	96.3±0.5	96.4±0.5	**96.5±0.5**
cifar10	71.4±0.8	73.2±0.7	**75.7±0.7**	71.7±0.8	71.9±0.8	71.9±0.8
cifar100	62.6±1.1	65.7±1.0	**67.3±1.0**	62.7±1.1	62.7±1.1	62.8±1.1
CropDisease	80.5±0.8	81.0±0.8	82.0±0.7	87.2±0.7	87.2±0.7	**87.4±0.7**
EuroSAT	86.6±0.5	86.1±0.6	**87.0±0.5**	86.1±0.6	86.1±0.6	86.3±0.6
ISIC	45.5±0.8	45.5±0.8	45.4±0.8	48.4±0.9	48.3±0.9	**48.9±0.9**
ChestX	26.6±0.6	25.9±0.5	26.3±0.5	**27.2±0.6**	27.1±0.6	27.1±0.6
Food101	51.9±1.1	55.3±1.1	**55.4±1.0**	53.9±1.1	53.9±1.1	53.9±1.1
SG acc	63.50	63.58	64.90	65.13	65.16	**65.26**

is more expensive in the meta-testing phase in terms of both computation and storage, as it needs to fine-tune each extractor and save their snapshots instead of utilising a single universal extractor. The good performance of FES could be attributed to its increased capacity, as it maintains individual extractors instead of a single universal extractor. In the context of Meta-Dataset, FES maintains eight extractors, which means 8× parameters compared to a universal model of the same architecture. Hence, in an additional experiment, we investigate larger universal models with capacities comparable to FES.

Originally, Li et al. (2021) distill eight ResNet18 extractors into a universal ResNet18 extractor. We distilled a universal ResNet152 (He et al., 2016) extractor using the same process. ResNet18 has 11M parameters while ResNet152 has 60M. We elected to use the same eight ResNet18 extractors for distillation, because pretraining eight ResNet152 extractors from scratch is prohibitively expensive for us, and this avoids introducing a confounding factor to meta-model evaluation because different base-model architectures may

Table 3.15: ResNet152 feature extractors with TSA fine-tuning. Note that the "URL" in the column titles refers to the universal representation learning process used to meta-train the models, not the feature projection fine-tuning used by the URL algorithm during meta-test. All methods in this table use TSA fine-tuning.

Dataset	URL18	MDL152	URL152	FES18	ConFES18	ReFES18
ilsvrc_2012	56.8±1.1	59.3±1.1	**59.9±1.1**	56.2±1.1	56.3±1.1	56.2±1.2
omniglot	**95.0±0.4**	94.7±0.4	**95.0±0.4**	93.3±0.6	93.3±0.7	93.6±0.6
aircraft	88.4±0.5	91.2±0.4	**92.2±0.4**	87.6±0.8	87.9±0.7	87.9±0.8
cu_birds	81.5±0.7	84.8±0.6	**85.0±0.6**	79.9±0.8	80.0±0.8	79.8±0.9
dtd	77.1±0.7	78.9±0.7	**79.2±0.7**	76.2±0.8	76.3±0.8	76.4±0.8
quickdraw	82.0±0.6	83.4±0.6	83.4±0.6	83.4±0.6	**83.5±0.6**	83.4±0.6
fungi	68.3±1.1	73.0±1.0	**74.5±1.0**	69.4±1.1	69.7±1.1	69.6±1.1
vgg_flower	92.1±0.5	**93.4±0.5**	92.2±0.6	91.9±0.7	91.9±0.7	92.1±0.6
WG acc	80.15	82.34	**82.68**	79.74	79.86	79.88
traffic_sign	82.8±0.9	76.4±1.0	80.0±0.9	84.9±1.0	85.1±1.0	**85.2±1.0**
mscoco	53.8±1.1	54.1±1.0	**56.9±1.0**	54.1±1.0	54.4±1.0	54.4±1.0
mnist	96.6±0.4	95.5±0.5	95.4±0.5	97.1±0.5	97.1±0.5	**97.2±0.5**
cifar10	79.9±0.8	79.3±0.8	**81.6±0.7**	78.1±0.9	78.2±0.9	78.8±0.9
cifar100	70.3±1.0	70.5±1.0	**73.1±1.0**	70.4±1.1	70.6±1.0	70.8±1.0
CropDisease	84.4±0.8	82.7±0.8	84.4±0.7	88.1±0.7	**88.3±0.7**	**88.3±0.7**
EuroSAT	**89.6±0.5**	88.6±0.6	**89.6±0.5**	88.8±0.6	89.1±0.6	89.3±0.6
ISIC	48.4±0.9	45.6±0.9	47.3±0.9	**49.5±0.9**	49.3±0.9	48.9±0.9
ChestX	27.2±0.6	25.5±0.6	27.3±0.6	27.7±0.6	27.7±0.6	**27.8±0.6**
Food101	53.4±1.2	56.2±1.1	**57.3±1.1**	55.2±1.1	55.5±1.1	55.2±1.1
SG acc	68.64	67.44	69.29	69.39	69.53	**69.59**

encompass source domain semantics differently. We also pretrained a universal ResNet152 model using "vanilla" multi-domain learning (MDL). One feature extractor is pretrained with all eight source domains' data using eight classification heads, one for each domain. Compared to official ResNet18 URL training, we halved the mini-batch size (and doubled the number of iterations) to fit ResNet152 URL or MDL training in the 48GB memory of an NVIDIA A6000 GPU—the most advanced at our disposal. Tables 3.14 and 3.15 show their results with URL or TSA fine-tuning respectively, and compare them to using the official ResNet18 URL model, as well as FES variants with ResNet18 extractor collections. As TSA fine-tuning has high memory consumption, we forwent adaptors in the first and second convolutional blocks (shown to have a small impact on accuracy by Li et al. (2022a)) to fit the ResNet152 TSA

Table 3.16: Computational resource consumption of FES variants using TSA fine-tuning, compared to the official TSA algorithm applied to a URL ResNet18 or ResNet152 extractor. From left to right: CDFSL method, fine-tuning time, stacking classifier training time, number of pretrained parameters that are frozen during fine-tuning, number of trainable parameters during fine-tuning, number of parameters that need to be stored, number of stacking classifier parameters, and amount of GPU memory required for fine-tuning.

method	FT T	MC T	frozen P	trainable P	stored P	MC P	FT mem
URL18	10.13s	-	11M	1.5M	11M + 1.5M	-	8.2GB
URL152	103.88s	-	60M	7.3M	60M + 7.3M	-	32.7GB
FES18		0.06s				328	8.3GB
ConFES18	151.29s	0.06s	11M×8	1.5M×8	11M×8 + 1.5M×8×41	144	8.3GB
ReFES18		9.45s				328	8.3GB

experiments on our NVIDIA A6000 GPU. In both tables, the ResNet152 URL model generally outperforms the ResNet18 URL and ResNet152 MDL models, and it achieves best average weak generalisation accuracy. Its mean strong generalisation accuracy is comparable to that of the FES variants, but individual results show that the methods excel at different tasks: the ResNet152 URL model performs better on mscoco, cifar10, cifar100, EuroSAT, and Food101, while the FES methods perform better on traffic_sign, mnist, CropDisease, ISIC, and ChestX—it appears that the ResNet152 URL model is better at ImageNet-adjacent tasks, while the FES methods are better at tasks that differ more substantially from ImageNet.

Table 3.16 compares the cost of FES inference using an NVIDIA A6000 GPU to that of URL ResNet18 and ResNet152 extractors. TSA fine-tuning is used by all methods in this table. It is worth pointing out that due to the few-shot nature of each episode, meta-testing is generally not time consuming. Table 3.16 represents the approximate upper bound of FES computation cost, because 1) the time presented in the table was measured using the largest traffic_sign episode in our cached sample, which contains 497 support instances, whereas smaller episodes consume less time, 2) URL and FLUTE fine-tuning

Table 3.17: Comparing the official URL model to a URL model distilled without favouring ImageNet

Dataset	URL_official	URL_equal
ilsvrc_2012	**56.6±1.1**	52.6±1.1
omniglot	94.5±0.4	**95.0±0.4**
aircraft	87.7±0.5	**88.6±0.5**
cu_birds	**80.7±0.7**	80.0±0.7
dtd	**76.1±0.6**	72.0±0.7
quickdraw	82.0±0.6	**82.1±0.6**
fungi	**69.5±1.1**	68.4±1.1
vgg_flower	**91.4±0.5**	89.5±0.6
mean WG acc	**79.81**	78.53
traffic_sign	62.6±1.2	**63.0±1.2**
mscoco	**52.7±1.0**	47.2±1.0
mnist	94.6±0.4	**95.1±0.4**
cifar10	**71.4±0.8**	66.5±0.8
cifar100	**62.6±1.1**	56.9±1.1
CropDisease	**80.5±0.8**	79.9±0.8
EuroSAT	**86.6±0.5**	83.5±0.6
ISIC	**45.5±0.8**	44.8±0.8
ChestX	**26.6±0.6**	**26.6±0.6**
Food101	**51.9±1.1**	49.0±1.1
mean SG acc	**63.50**	61.25

are much less time-consuming than TSA, and 3) Section 3.4.3 shows that a portion of the snapshots does not in fact need to be computed and stored.

FES requires approximately $2 \times K$ as much backpropagation as a universal extractor fine-tuned once, where 2 represents one fine-tuning run on the cross-validated support set (performed in two splits) and another on the full support set, and K represents the number of extractors. This is reflected in Table 3.16 as fine-tuning time for the FES methods is approximately 16 times that of fine-tuning the URL ResNet18 model. Time required to train a FES or ConFES stacking classifier is relatively trivial, while ReFES requires more time to determine its regularisation strength using grid search with cross-validation. FES stores multiple snapshots of each extractor during fine-tuning, but not all model parameters need to be saved. Only weights that are updated during fine-tuning need to be saved in snapshots, as the other unchanged weights can be loaded from the original extractor. Common CDFSL fine-

tuning algorithms only update a relatively small set of weights: FLUTE fine-tunes batch normalisation weights, URL fine-tunes a feature projection, and TSA fine-tunes channel projections and a feature projection. Therefore, FES snapshots are normally lightweight. Table 3.16 shows that FES with TSA fine-tuning needs to store approximately 580M parameters—2.32GB—which can fit in most modern GPUs during inference. As FES can fine-tune its extractors sequentially, its memory requirement is comparable to fine-tuning a single extractor with the same method. On the other hand, FES can easily be parallelised to fine-tune multiple extractors at once, should multiple GPUs be available. Chapter 5 investigates pruning FES snapshots to reduce its storage cost.

Considering computational effort required for meta training, it is worth noting that even though a universal extractor only needs to be trained once, this training process may take days (for ResNet18) to weeks (for ResNet152) on an NVIDIA A6000 GPU; if an individual extractor is added or updated, training of a universal extractor needs to be performed again. In this context, it is worth considering that the official URL model was distilled in a process favouring ImageNet by including as many ImageNet instances as the other seven source domains combined in each mini-batch (Li et al., 2021). We distilled an alternative URL model while treating all source domains equally. Their comparison is shown in Table 3.17. The official model performs better in a majority of domains. This indicates that URL distillation may require external knowledge to focus on the right domains to achieve optimal performance. FES and its variants treat all extractors equally *a priori* and determine their task-specific relevance based purely on the support set.

3.8 Conclusion

We present the stacking-based CDFSL method FES and the variants ConFES and ReFES. The FES algorithms create snapshots from fine-tuning indepen-

dent extractors on the support set, use cross-validation to avoid overfitting from support data reuse, and train a simple stacking classifier to appropriately weight the snapshots. FES, ConFES, and ReFES advance the state of the art on the Meta-Dataset benchmark.

Perhaps more importantly, the FES approaches have some practical advantages in real-world scenarios compared to recent methods based on universal models. FES can work with out-of-the-box heterogeneous extractors. If the extractors are readily available, FES does not require their pretraining data down-stream. Its stacking classifier requires little hyperparameter tuning. FES is also computationally cheaper, unless the number of few-shot learning tasks is very large, for example, in the thousands, where the total cost of performing FES on all tasks begins to exceed that of training a universal model once. Therefore, to field practitioners who wish to use extractors and fine-tuning algorithms specific to their work, FES is likely more flexible and user-friendly than universal-model methods.

Chapter 4

Self-trained Centroids

Exploiting unlabelled data is an obvious approach to improving performance in few-shot scenarios, and in this chapter, we introduce a self-training method for semi-supervised learning that is specifically designed for few-shot learning. We first define the self-trained centroids (STC) method. We then cover literature on large-scale semi-supervised learning and semi-supervised methods in CDFSL. Following this, we evaluate STC using CDFSL methods including FES on the Meta-Dataset benchmark and analyse their performance.

4.1 Semi-supervised CDFSL with Self-trained Centroids

A semi-supervised few-shot learning episode contains, as its training data, a labelled set L and an unlabelled set U. L is of size N, containing instances $X \in \mathbb{R}^{I \times N}$ with I input dimensions and their labels $Y \in [0, C)^N$ belonging to classes C. $U \in \mathbb{R}^{I \times M}$ contains M instances also belonging to C but their labels are unknown. The learning task is to fit a model, pretrained on source domains, using L and U, and evaluate it using a separate test/query set Q. The goal of a semi-supervised CDFSL method is to learn from L and U and outperform its supervised counterpart learning from only L. We first formulate the STC learning algorithm for cases where it is applied with a single feature extractor

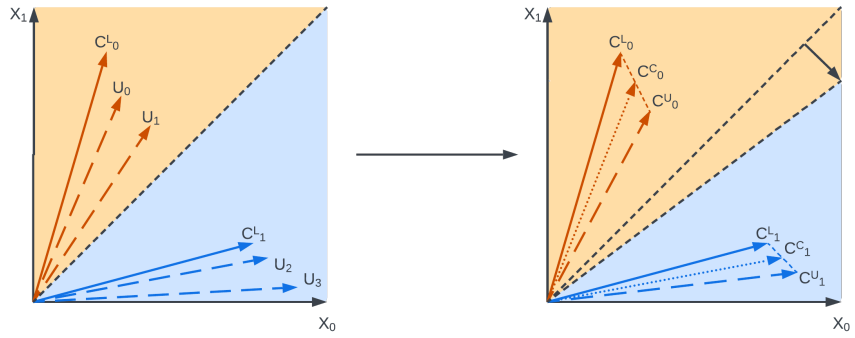


Figure 4.1: Visualisation of STC. In the left diagram, labelled centroids (solid vectors, C_0^L and C_1^L) are used to soft-label unlabelled feature vectors (dashed vectors, U_0 - U_3) with cosine similarity distance. The black dashed line represents the decision boundary. In the right diagram, "labelled" and "unlabelled" centroids are averaged to produce combined centroids (dotted lines, C_0^C and C_1^C). This results in a shift in the decision boundary.

and then explain how it can be used in FES ensembles. An illustration of the STC algorithm is given in Figure 4.1. Like most semi-supervised learning methods in the literature, STC relies on the cluster assumption (van Engelen and Hoos, 2020): data points in the same cluster belong to the same class.

Given a feature extractor Φ, we fine-tune it on L by using a nearest-centroid classifier as the classification head. In CDFSL, common pretraining methods for obtaining the feature extractor include vanilla pretraining on a single source domain, as it is applied in FES, knowledge distillation (Li et al., 2021), and universal template training (Triantafillou et al., 2021). Common fine-tuning methods include feature projection (Li et al., 2021), batch normalisation fine-tuning (Triantafillou et al., 2021), and task-specific adaptors (Li et al., 2022a). All these methods can be used in conjunction with STC. To this end, we use the fine-tuned Φ' to extract feature vectors $X^F \in \mathbb{R}^{J \times N}$ with J feature dimensions from X, and $U^F \in \mathbb{R}^{J \times M}$ from U. We compute class centroids $X^C \in \mathbb{R}^{J \times C}$ for the labelled set using X^F and Y, with S_j representing the set of indices of labelled instances belonging to a particular class j:

$$X_j^C = \frac{1}{|S_j|} \sum_{i \in S_j} X_i^F, S_j = \{k : Y_k = j\}, j = 1, ..., C. \tag{4.1}$$

Once we have the set of centroid vectors, X^C, we can use them to assign soft labels to U^F by applying a similarity measure s and the softmax function. For each unlabelled feature vector U_i^F, its soft labels P_i^U are computed as:

$$P_i^U(Y_i^U = j|U_i^F) = \frac{e^{s(U_i^F, X_j^C)}}{\sum_{i=1}^{C} e^{s(U_i^F, X_i^C)}}. \tag{4.2}$$

Here, we use the same s as the one applied in the nearest-centroid classifier during fine-tuning. The exact formulation depends on the fine-tuning method that is applied. Li et al. (2021) and Li et al. (2022a) use cosine similarity scaled by a factor of 10, while Triantafillou et al. (2021) use cosine similarity without scaling.

Once soft labels have been obtained, we can compute class centroids $U^C \in \mathbb{R}^{J \times C}$ for the unlabelled set based on U^F and the corresponding soft labels P^U by using these soft labels to calculate weighted averages:

$$U_j^C = \frac{\sum_{i=1}^{|U^F|} P_i^U(Y_i^U = j|U_i^F) \cdot U_i^F}{\sum_{i=1}^{|U^F|} P_i^U(Y_i^U = j|U_i^F)}. \tag{4.3}$$

The final set of centroids $C^C \in \mathbb{R}^{J \times C}$ is subsequently obtained as a simple arithmetic average of X^C and U^C, giving equal weight to the centroids from the data with ground-truth labels and the centroids from the pseudo-labelled data:

$$C_j^C = \frac{X_j^C + U_j^C}{2}. \tag{4.4}$$

An iterative self-training process can be formed to further refine the soft labels by then using C^C instead of X^C to soft-label U^F, i.e., replacing X^C with C^C in Equation 4.2, and repeating Equations 4.2 - 4.4. However, interestingly, in line with the findings of Ren et al. (2018), whose method we discuss in

the next section, we found that simply performing Equations 4.1 - 4.4 once is often sufficient: iterating the process leads to relatively little benefit overall and may even harm in some cases, despite it being common procedure in full self-training with larger datasets (Rosenberg et al., 2005).

After training, predictions for a query instance Q_i are made with its feature vector Q_i^F and the final centroids C^C:

$$P_i^Q(Y_i^Q = j|Q_i^F) = \frac{e^{s(Q_i^F, C_j^C)}}{\sum_{i=1}^{C} e^{s(Q_i^F, C_i^C)}}. \tag{4.5}$$

The above description of STC is based on a single feature extractor. However, the previous chapter has shown that the classic stacked generalisation approach to ensemble learning, in the form of feature extractor stacking, can be used to obtain state-of-the-art accuracy on CDFSL problems: a stacking classifier, for example, a very simple weight kernel, is trained using cross-validated predictions obtained from a set of source domain backbones ("base models") during fine-tuning and learns to appropriately combine predictions from all the snapshots available. Fortunately, it is straightforward to apply STC in conjunction with FES. Given a stacking classifier that takes fine-tuned base model logits as input, in order to re-purpose it with STC for semi-supervised learning, we simply apply STC to each fine-tuned base model and replace the logits normally obtained from the centroid classifier based on supervised training with logits obtained from STC.

4.2 Semi-supervised Learning Literature

There appears to be comparatively little work on semi-supervised cross-domain few-shot learning. Ren et al. (2018) do not consider cross-domain learning but do investigate semi-supervised learning with prototypical networks (Snell et al., 2017) by soft-labelling unlabelled instances and adjusting the prototypes/centroids with them. As a prototypical network is meta-trained using few-shot episodes sampled from source domains, these episodes are modified

to contain unlabelled instances but meta-testing is performed "in-domain", as the source and target domains are different class partitions of the same domain. Note that soft labels are essential to facilitate backpropagation in a prototypical network's training.

In contrast to the method proposed in Ren et al. (2018), STC is independent of, and separated from, a feature extractor's pretraining and fine-tuning, and thus compatible with any feature extractor that applies nearest-centroid classification. The semi-supervised learning step in STC means that it can be applied with soft or hard labels—but soft labels are generally preferable in CDFSL given high uncertainty in some unlabelled instances. Lastly, STC is designed for cross-domain tasks, which is arguably more relevant for practical applications: Guo et al. (2020) showed that several meta-learning approaches at the time, including prototypical networks, underperform in CDFSL settings.[1]

A number of semi-supervised few-shot learning methods pretrain (or "meta-train") their feature extractors using semi-supervised learning like in Ren et al. (2018): transductive CNAPs (Bateni et al., 2022) uses query instances for feature adaptation, GCT (Xu et al., 2022) converts instances into graph nodes aided by unlabelled instances, dynamic distillation (Islam et al., 2021) uses augmented unlabelled instances to fit a teacher-student network pair, and Li and Zhang (2021) create additional meta-training tasks for meta-learners with large language models using vocabulary tokenisation and self-supervision. In contrast, we evaluate all semi-supervised learning methods using feature extractors pretrained in a supervised manner, thus anticipating a scenario that is likely to occur in practical applications, and focus on the effect of semi-supervised learning during meta-testing. We compare STC to

[1]Note that Ren et al. (2018) considered an "inference only" baseline in their experiments, where the feature extractor received supervised pretraining, and unlabelled instances were only used to adjust centroids, which is comparable to STC. We consider STC to be a generalisation of this baseline and show that it can be applied during meta-testing to various centroid-based CDFSL methods.

several generic semi-supervised methods applied in this setting, including full self-training (Rosenberg et al., 2005), which fits all trainable parameters to soft-labelled instances, Pi-Model (Laine and Aila, 2017), which optimises consistency between logits of different copies of augmented unlabelled instances, temporal ensembling (Laine and Aila, 2017), which optimises consistency between logits of augmented unlabelled instances and their exponential moving average, Mean Teacher (Tarvainen and Valpola, 2017), which uses augmented unlabelled instances to optimise consistency between a student model and a teacher model given by an exponential moving average of previous students, and SimCLR (Chen et al., 2020), which optimises agreement between projections of feature vectors of different copies of augmented unlabelled instances. Among the methods that implement semi-supervised meta-training for the feature extractor, transductive CNAPs accommodates multiple source domains, which makes it applicable to our evaluation scenario, so is included in our comparison.

We apply STC to URL, FLUTE, a URL extractor with TSA fine-tuning, as well as the FES version of these methods, and evaluate them on the Meta-Dataset benchmark (Triantafillou et al., 2020b).

4.3 Experimental Setup

Meta-Dataset produces a supervised few-shot episode for training and evaluating a few-shot learner by first sampling several classes from the test split of a dataset, and then sampling labelled training and test instances from these classes (Triantafillou et al., 2020b). We continue to use the cached episodes used in Chapter 3, which follow the official specifications and sample episodes each containing 5 to 50 classes, with up to 500 labelled (potentially class-imbalanced) instances in total, as well as 10 query instances per class. To enable semi-supervised learning, we pool the remaining instances in the sampled classes that have not been selected as training or test instances, and we

Table 4.1: Count and average size of unlabelled sets with fewer than 1000 instances

dataset	sub-1000 unlabelled count	sub-1000 unlabelled average size
ilsvrc_2012	0	-
omniglot	600	96.7
aircraft	554	523.3
cu_birds	553	487.6
dtd	600	369.1
quickdraw	0	-
fungi	258	462.6
vgg_flower	564	508.9
traffic_sign	0	-
mscoco	0	-
mnist	0	-
cifar10	0	-
cifar100	145	531.3
CropDisease	0	-
EuroSAT	0	-
ISIC	0	-
ChestX	0	-
Food101	9	878.7

then randomly select 1000 instances from the pool as the unlabelled set of the episode based on the assumption that obtaining 1000 unlabelled instances per task is generally achievable in practical CDFSL scenarios. The 1000 unlabelled instances can potentially be class-imbalanced. In some cases where the classes are small, there may be fewer than 1000 instances in the pool, and the entire pool is used as the unlabelled set. We cache sampled semi-supervised CDFSL episodes and use the same cached episodes to evaluate all methods, which avoids variance between sampling runs and facilitates paired t-tests. Paired testing increases statistical power compared to unpaired methods by considering the difference in accuracy on a per-episode basis. Table 4.1 shows the number of episodes, out of 600 sampled per dataset, that failed to obtain 1000 unlabelled instances, and the average size of those particular episodes.

We first evaluate four well-known semi-supervised learning methods for large labelled datasets from the literature: Pi-Model, temporal ensembling,

Mean Teacher, and SimCLR, and compare them to full self-training, by applying all of them to URL (Li et al., 2021) with a linear projection fitted to feature vectors extracted by a fixed universal feature extractor, used in conjunction with a nearest-centroid classifier. The universal ResNet18 feature extractor is downloaded from the official URL repository. The linear projection in URL is treated as the optimisable parameter set. We used the first 20 episodes of each source domain to tune the hyperparameters of these methods, which led to a multiplier of 100 for consistency loss in Pi-Model, temporal ensembling, and Mean Teacher, an α of 0.5 for the exponential moving average in temporal ensembling and Mean Teacher, and a multiplier of 1 for agreement loss in SimCLR. For the consistency loss, we found that taking the mean instead of the sum of the squared differences of the logits leads to more stable performance, presumably due to the varying number of classes in different episodes. We also evaluate transductive CNAPs with our cached episodes. We use the hyperparameters from (Bateni et al., 2022), and the ResNet18 checkpoint downloaded from the official repository, reporting results using the query set as unlabelled data for transduction. We found that including the unlabelled set as additional data degraded performance.

Following this, we thoroughly evaluate the STC method by applying it to URL (Li et al., 2021), FLUTE (Triantafillou et al., 2021), and a URL extractor with TSA fine-tuning (Li et al., 2022a), as well as their counterparts in a FES ensemble. All fine-tuning processes and hyperparameters are kept consistent with the original papers, all feature extractors are ResNet18 models downloaded from the official sources, and STC is simply applied to the feature extractors post-fine-tuning. A single design choice was made for STC: a plain arithmetic average is used to aggregate labelled and unlabelled centroids instead of an average weighted by the number of instances. This was based on the intuition that a weighted average can be overwhelmed by a large number of noisy unlabelled instances and lead to instability, and a few source domain episodes were sufficient to confirm this. Non-iterative results,

obtained after the first iteration of self-training in STC, are presented in tables and compared to non-iterative full self-training for URL, FLUTE, and a URL extractor with TSA fine-tuning, but not their FES counterparts, because performing non-iterative full self-training with FES is prohibitively expensive computation-wise. We also show plots visualising the accuracy of STC across 20 self-training iterations.

4.4 Results

We first show that semi-supervised algorithms for large labelled datasets may not be well-suited for CDFSL, and justify this claim by showing better performance of simple non-iterative full self-training when using URL as the case study. We then present results obtained by applying non-iterative STC to a range of state-of-the-art CDFSL methods, and show that it achieves improved performance over supervised learning and full self-training. Lastly, we present accuracy-over-iteration plots for iterative STC.

4.4.1 Common Semi-supervised Algorithm in CDFSL

Table 4.2 shows common semi-supervised methods applied to URL, and compares them to the supervised approach. For each dataset, mean accuracy of 600 few-shot episodes is reported, along with the 95% confidence interval. Results are averaged for source and target domains separately, as only target domain tasks are truly cross-domain, and their accuracy represents "strong generalisation" (SG) performance; while source domains represent "weak generalisation" (WG). The best results for each dataset are marked **bold**.

Only non-iterative full self-training achieves higher average accuracy than supervised URL in terms of SG performance, while Pi-Model, temporal ensembling, Mean Teacher, SimCLR, and iterative full self-training all perform worse. Transductive CNAPs achieves top accuracy in several target domains but its average SG performance is not as strong as that of the URL-based

Table 4.2: Pi-Model, temporal ensembling, Mean Teacher, SimCLR, non-iterative (1 iteration) and iterative (20 iterations) full self-training applied to URL with 1000 unlabelled instances, compared to supervised URL. Transductive CNAPs results are provided in the rightmost column.

URL	Sup	Pi	TE	MT	SCLR	ST-1	STC-20	T-CNAPs
ilsvrc_2012	**56.6±1.1**	56.5±1.1	**56.6±1.1**	**56.6±1.1**	56.4±1.1	**56.6±1.1**	**56.6±1.1**	55.8±1.1
omniglot	94.5±0.4	94.5±0.4	94.5±0.4	94.5±0.4	94.4±0.4	**95.1±0.4**	**95.1±0.3**	93.4±0.5
aircraft	87.7±0.5	87.5±0.5	87.5±0.5	87.6±0.5	87.3±0.5	87.8±0.5	**88.1±0.4**	82.1±0.7
cu_birds	80.7±0.7	80.9±0.7	80.9±0.7	80.8±0.7	80.6±0.7	81.2±0.7	**81.3±0.6**	77.7±0.8
dtd	**76.1±0.6**	75.8±0.7	75.8±0.6	75.9±0.6	75.6±0.6	75.8±0.6	76.0±0.6	68.3±0.7
quickdraw	82.0±0.6	81.9±0.6	82.0±0.6	82.0±0.6	81.9±0.6	82.4±0.6	**82.7±0.6**	78.2±0.7
fungi	69.5±1.1	69.2±1.0	69.4±1.0	69.3±1.0	69.5±1.1	70.7±1.0	**71.2±1.0**	50.0±1.3
vgg_flower	91.4±0.5	91.4±0.5	91.4±0.5	91.4±0.5	91.3±0.5	91.8±0.5	**92.1±0.4**	91.3±0.5
WG acc	79.8	79.7	79.8	79.8	79.6	80.2	**80.4**	74.6
traffic_sign	**62.6±1.2**	**62.6±1.2**	62.0±1.1	61.3±1.1	62.0±1.2	62.3±1.2	61.5±1.2	57.3±1.1
mscoco	52.7±1.0	52.3±1.0	52.8±1.0	52.7±1.0	53.0±1.0	**53.9±1.0**	53.5±0.9	48.5±1.0
mnist	94.6±0.4	94.5±0.5	94.0±0.4	93.8±0.5	94.2±0.4	94.7±0.4	92.2±1.0	**95.3±0.3**
cifar10	71.4±0.8	71.0±0.8	71.3±0.8	71.1±0.8	71.6±0.8	71.7±0.8	71.3±0.8	**72.1±0.7**
cifar100	62.6±1.1	62.4±1.1	62.6±1.1	62.4±1.1	62.5±1.1	**63.0±1.1**	62.6±1.1	61.3±1.1
CropDisease	80.5±0.8	80.8±0.8	80.9±0.8	78.8±0.8	80.0±0.8	**81.0±0.8**	80.5±0.8	79.4±0.8
EuroSAT	86.5±0.5	**86.7±0.5**	85.3±0.5	85.8±0.5	86.4±0.5	86.6±0.5	85.8±0.6	78.9±0.6
ISIC	45.5±0.8	45.3±0.8	44.1±0.8	44.0±0.8	45.6±0.8	**46.9±0.9**	46.8±0.9	44.1±0.8
ChestX	26.6±0.6	26.5±0.6	26.4±0.5	26.6±0.6	26.5±0.6	26.8±0.6	26.8±0.6	**27.1±0.6**
Food101	51.9±1.1	51.4±1.0	52.0±1.1	51.6±1.1	**52.2±1.1**	52.1±1.1	51.8±1.1	51.2±1.1
SG acc	63.5	63.4	63.1	62.8	63.4	**63.9**	63.3	61.5

methods. Non-iterative full self-training always yields higher accuracy than its iterative counterpart in SG, indicating that full self-training, which is commonly iterative in the literature, may exhibit instability in CDFSL as more iterations are performed. Overall, the positive results for full self-training provide the motivation for investigating the more efficient and, as it turns out, more robust STC algorithm, which only applies self-training to the centroids, in the following sections.

4.4.2 STC CDFSL Evaluations

Tables 4.3, 4.5, and 4.7 show non-iterative STC performance—for TSA, URL, and FLUTE fine-tuning respectively—compared to that of supervised learning and non-iterative full self-training. In each of the three tables, the five columns show results for 1) the supervised base algorithm ("base"), 2) full self-training applied to the base algorithm ("base-FST"), 3) the supervised FES ensemble of the base algorithm ("FES"), 4) STC applied to the base algorithm ("base-STC"), and 5) STC applied to FES ("FES-STC"). Like before, mean accuracy is reported with the 95% confidence interval. Mean WG and SG ranks are computed using individual episode accuracy values. Paired t-tests, which are generally more sensitive than 95% confidence intervals, are performed to compute the p value between the methods using their accuracy values in all individual episodes. A p smaller than 0.05 is deemed to indicate a statistically significant difference. Among the columns, "base" and "base-FST" are compared to "base-STC", while "FES" is compared to "FES-STC". If $p < 0.05$, $\bullet$ indicates an algorithm's STC counterpart has better performance, and $\circ$ indicates the algorithm performs statistically significantly better than its STC counterpart. In addition, "base-STC" is compared to "FES-STC". If $p < 0.05$, $+$ indicates that the semi-supervised FES ensemble has better performance, while $-$ indicates better performance for the semi-supervised base algorithm.

The results show that for URL and TSA fine-tuning, STC consistently exhibits higher estimated accuracy than supervised learning and full self-training

Table 4.3: Comparison of STC, supervised learning, and non-iterative full self-training using TSA fine-tuning

TSA	base	base-ST	FES	base-STC	FES-STC
ilsvrc_2012	56.8±1.1 •	56.8±1.1 •	56.2±1.1 •	**57.2±1.1**	56.6±1.1 −
omniglot	95.0±0.4 •	95.3±0.4 •	93.3±0.6 •	**95.7±0.3**	94.0±0.6 −
aircraft	88.4±0.5 •	88.6±0.5 •	87.6±0.8 •	**88.8±0.5**	87.9±0.7 −
cu_birds	81.5±0.7 •	81.8±0.7 •	79.9±0.8 •	**82.2±0.7**	80.2±0.8 −
dtd	**77.1±0.7**	76.8±0.7	76.2±0.8	77.0±0.6	76.2±0.8 −
quickdraw	82.0±0.6 •	82.4±0.6 •	83.4±0.6 •	82.7±0.6	**83.8±0.6** +
fungi	68.3±1.1 •	69.0±1.0 •	69.4±1.1 •	70.0±1.0	**70.9±1.1** +
vgg_flower	92.1±0.5 •	92.3±0.5 •	91.9±0.7 •	**92.8±0.5**	92.4±0.6 −
WG acc	80.15	80.38	79.74	**80.80**	80.25
WG rank	3.30	3.04	3.10	**2.75**	2.81
traffic_sign	82.8±0.9 •	84.0±0.9 ○	84.9±1.0 •	83.8±0.9	**86.0±1.0** +
mscoco	53.8±1.1 •	53.9±1.1 •	54.1±1.0 •	54.7±1.0	**55.5±1.0** +
mnist	96.6±0.4 •	96.6±0.4 •	97.1±0.5 •	97.0±0.3	**97.4±0.4** +
cifar10	79.9±0.8 •	80.2±0.8 •	78.1±0.9 •	**80.4±0.7**	78.6±0.8 −
cifar100	70.3±1.0 •	70.4±1.0 •	70.4±1.1 •	**70.9±1.0**	**70.9±1.0**
CropDisease	84.4±0.8 •	85.0±0.8 •	88.1±0.7 •	85.6±0.7	**89.1±0.6** +
EuroSAT	89.6±0.5 •	**90.0±0.5**	88.8±0.6 •	89.9±0.5	89.0±0.6 +
ISIC	48.4±0.9 •	48.0±0.9 •	49.5±0.9 •	49.5±0.9	**51.4±0.9** +
ChestX	27.2±0.6 •	27.6±0.6	27.7±0.6 •	27.6±0.6	**28.4±0.6** +
Food101	53.4±1.2 •	53.3±1.2 •	55.2±1.1 •	53.8±1.2	**55.5±1.1** +
SG acc	68.64	68.90	69.39	69.32	**70.18**
SG rank	3.51	3.21	2.93	2.92	**2.43**

Table 4.4: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results with TSA fine-tuning

	WG					SG				
	B	B-ST	F	B-STC	F-STC	B	B-ST	F	B-STC	F-STC
base	-	6	2	7	3	-	5	7	10	8
base-ST	1	-	1	7	2	0	-	5	7	8
FES	5	6	-	7	7	2	2	-	4	10
base-STC	0	0	1	-	2	0	1	3	-	7
FES-STC	4	4	0	6	-	2	2	0	2	-

Table 4.5: Comparison of STC, supervised learning, and non-iterative full self-training using URL fine-tuning

URL	base	base-ST	FES	base-STC	FES-STC
ilsvrc_2012	56.6±1.1	56.6±1.1	56.1±1.1 •	**56.7±1.1**	56.3±1.1 −
omniglot	94.5±0.4 •	**95.1±0.4** •	93.1±0.6 •	**95.1±0.4**	93.8±0.6 −
aircraft	87.7±0.5 •	87.8±0.5	87.1±0.8	**87.9±0.5**	87.2±0.7 −
cu_birds	80.7±0.7 •	81.2±0.7 •	79.1±0.8	**81.3±0.7**	79.2±0.8 −
dtd	**76.1±0.6** ○	75.8±0.6	75.0±0.8 ○	75.8±0.6	74.8±0.8 −
quickdraw	82.0±0.6 •	82.4±0.6 •	83.2±0.6 •	82.6±0.6	**83.5±0.6** +
fungi	69.5±1.1 •	70.7±1.0 •	69.6±1.1 •	**71.0±1.0**	70.9±1.1
vgg_flower	91.4±0.5 •	91.8±0.5 •	90.7±0.7 •	**91.9±0.4**	90.9±0.6 −
WG acc	79.81	80.18	79.24	**80.29**	79.58
WG rank	3.19	2.91	3.10	**2.85**	2.95
traffic_sign	62.6±1.2	62.3±1.2 •	**65.2±1.2** ○	62.6±1.2	64.9±1.2 +
mscoco	52.7±1.0 •	**53.9±1.0**	52.6±1.0 •	53.8±1.0	53.5±1.0
mnist	94.6±0.4 •	94.7±0.4 •	96.3±0.5 •	95.1±0.4	**96.6±0.5** +
cifar10	71.4±0.8 •	71.7±0.8 •	71.7±0.8 •	72.0±0.7	**72.1±0.8**
cifar100	62.6±1.1 •	**63.0±1.1**	62.7±1.1 •	**63.0±1.1**	62.9±1.1
CropDisease	80.5±0.8 •	81.0±0.8 •	87.2±0.7 •	81.4±0.8	**87.9±0.6** +
EuroSAT	86.6±0.5 •	86.6±0.5 •	86.1±0.6	**86.9±0.5**	86.2±0.6 −
ISIC	45.5±0.8 •	46.9±0.9	48.4±0.9 •	46.7±0.9	**50.1±0.9** +
ChestX	26.5±0.6	26.8±0.6	**27.2±0.6**	26.8±0.6	**27.2±0.6**
Food101	51.9±1.1 •	52.1±1.1 •	53.9±1.1 •	52.3±1.1	**54.0±1.1** +
SG acc	63.49	63.90	65.13	64.06	**65.54**
SG rank	3.48	3.25	2.71	3.09	**2.47**

Table 4.6: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results with URL fine-tuning

	WG					SG				
	B	B-ST	F	B-STC	F-STC	B	B-ST	F	B-STC	F-STC
base	-	6	1	6	2	-	6	6	8	9
base-ST	1	-	1	5	1	1	-	5	6	5
FES	6	7	-	7	5	1	3	-	3	7
base-STC	1	0	1	-	1	0	0	5	-	5
FES-STC	4	5	1	6	-	0	1	1	1	-

Table 4.7: Comparison of STC, supervised learning, and non-iterative full self-training using FLUTE fine-tuning

FLUTE	base	base-ST	FES	base-STC	FES-STC
ilsvrc_2012	50.2±1.1 ○	50.8±1.1 ○	**54.1±1.1**	49.6±1.1	54.0±1.1 +
omniglot	93.9±0.5 ●	93.7±0.5 ●	94.3±0.6 ●	95.0±0.4	**95.4±0.5** +
aircraft	86.8±0.6 ●	86.0±0.6 ●	**87.4±0.7**	87.1±0.5	**87.4±0.7**
cu_birds	79.3±0.8 ●	78.5±0.8 ●	78.4±0.9	**79.8±0.7**	78.5±0.9 −
dtd	68.8±0.8 ○	68.0±0.7 ●	**74.3±0.9**	68.5±0.7	**74.3±0.8** +
quickdraw	79.1±0.7	78.5±0.7 ●	**82.9±0.6**	79.0±0.7	**82.9±0.6** +
fungi	59.4±1.2 ●	60.5±1.2 ●	68.7±1.1 ●	61.8±1.2	**70.0±1.1** +
vgg_flower	91.0±0.6 ●	90.9±0.5 ●	92.5±0.6 ●	91.2±0.5	**92.8±0.5** +
WG acc	76.06	75.86	79.08	76.50	**79.41**
WG rank	3.44	3.73	2.29	3.33	**2.21**
traffic_sign	57.9±1.1 ○	54.1±1.1 ●	**72.6±1.1**	55.9±1.1	72.5±1.1 +
mscoco	48.2±1.0	48.6±1.0 ○	51.7±1.1 ●	48.3±1.0	**52.6±1.0** +
mnist	95.7±0.4 ●	95.0±0.4 ●	97.4±0.4 ●	96.2±0.3	**97.7±0.4** +
cifar10	78.6±0.7 ●	**79.0±0.7**	75.2±0.9 ●	**79.0±0.7**	75.6±0.8 −
cifar100	67.5±1.0 ●	67.4±1.0 ●	67.2±1.1 ○	**67.7±1.0**	67.0±1.0 −
CropDisease	78.0±0.8 ●	78.6±0.8 ○	86.5±0.7 ●	78.2±0.8	**87.0±0.6** +
EuroSAT	81.6±0.6 ○	79.9±0.6 ●	88.3±0.6	80.8±0.6	**88.4±0.6** +
ISIC	46.1±1.0 ●	48.7±0.9 ●	48.7±1.0 ●	49.0±0.9	**51.5±0.9** +
ChestX	26.3±0.5	26.4±0.5 ○	27.8±0.6 ●	26.1±0.5	**28.4±0.6** +
Food101	45.7±1.1 ○	46.7±1.1 ○	**52.1±1.1**	45.5±1.1	52.0±1.1 +
SG acc	62.56	62.44	66.75	62.67	**67.27**
SG rank	3.48	3.57	2.35	3.45	**2.16**

Table 4.8: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results with FLUTE fine-tuning

	WG					SG				
	B	B-ST	F	B-STC	F-STC	B	B-ST	F	B-STC	F-STC
base	-	2	7	5	7	-	5	8	5	8
base-ST	5	-	7	7	7	3	-	7	5	8
FES	1	0	-	2	3	2	1	-	2	6
base-STC	2	1	5	-	6	3	4	7	-	8
FES-STC	1	0	0	1	-	2	2	1	2	-

in SG. For FLUTE fine-tuning, STC has better average SG performance but its relative performance varies among datasets. FES with STC also consistently exhibits higher estimated SG accuracy than supervised FES with URL or TSA fine-tuning, as well as higher average SG accuracy for FLUTE fine-tuning. Finally, considering SG and analogously to the results for purely supervised learning in Chapter 3, STC in a FES ensemble exhibits higher estimated accuracy than STC in the base algorithm counterpart. It is worth re-iterating that STC achieves higher estimated accuracy at minimal added computational cost: it only requires unlabelled feature vectors and their soft labels from forward propagation, while full self-training requires backpropagation for re-fitting after soft-labelling.

4.4.3 Iterative STC

Figure 4.2 shows how iterative STC SG performance changes over 20 iterations. Values depicted are differences between STC and supervised learning. Values higher than 0 indicate that STC achieves higher estimated accuracy than supervised learning, and values lower than 0 indicate the opposite. Each row of figures represents a CDFSL method (a base algorithm or its FES counterpart), and each column represents a different unlabelled set size (10, 100, or 1000). The increased accuracy of non-iterative STC with 1000 unlabelled instances vs. supervised learning, discussed above and reported in Tables 4.3, 4.5, and 4.7, is reflected in the sharp change from iteration 0 (supervised accuracy) to iteration 1 (non-iterative STC accuracy) in the rightmost column of the figure.

With 1000 unlabelled instances, STC accuracy generally does not change significantly from iteration 1 to 20, which indicates that one iteration (non-iterative STC) is sufficient to achieve optimal performance, although there are small improvements for some target domains when applying STC iteratively using TSA fine-tuning. Clearly, STC is more stable than full self-training in CDFSL, as Table 4.2 shows that more iterations lead to worse SG performance

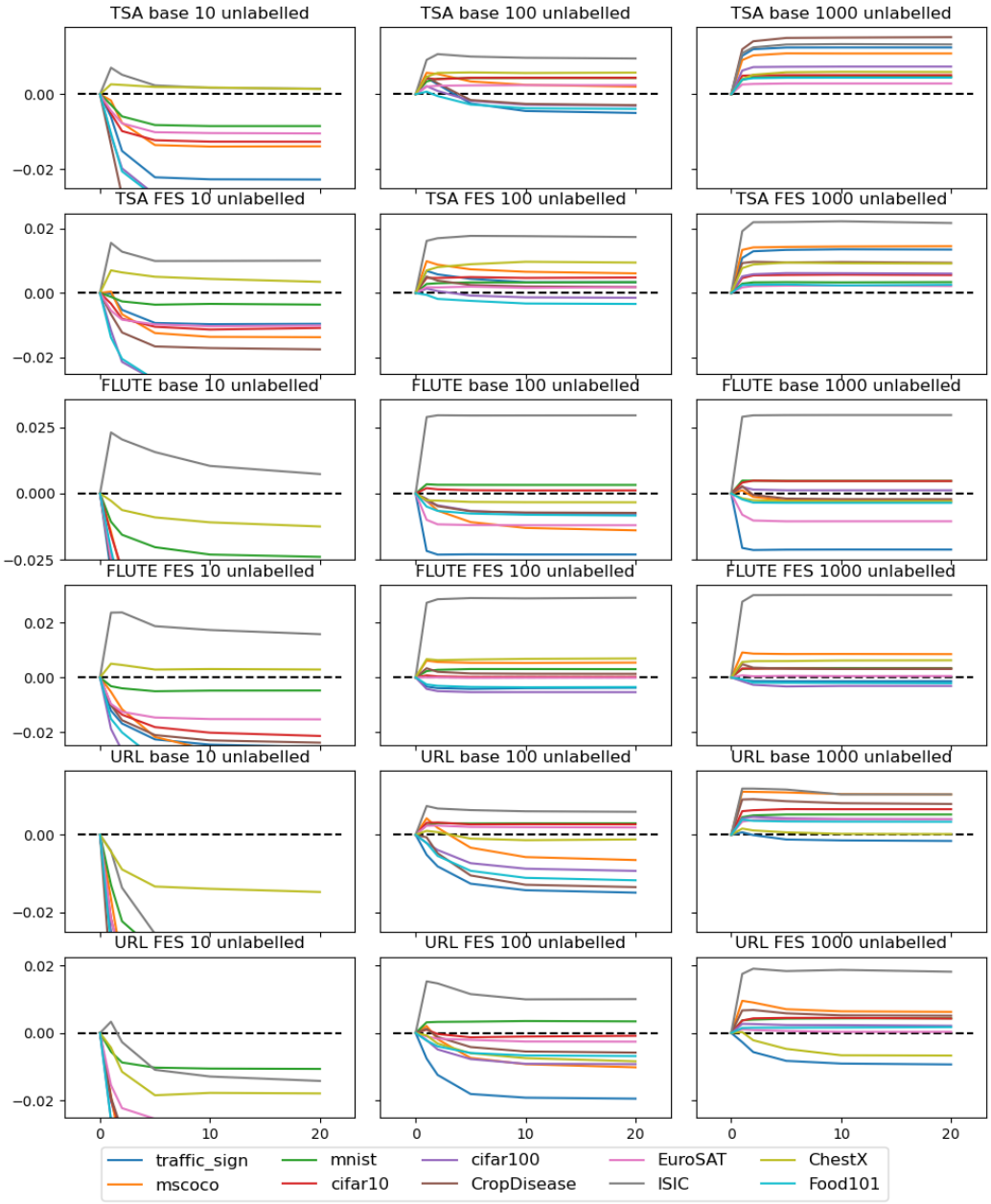


Figure 4.2: Iterative STC accuracy (relative to supervised) in 20 iterations given 10, 100, or 1000 unlabelled instances

for full self-training with the same 1000 unlabelled instances. Comparing the three columns in Figure 4.2, STC performs better with more unlabelled instances, as 1000 unlabelled instances lead to better performance in general, and especially consistently with TSA fine-tuning and its FES variant, while 10 unlabelled instances mostly lead to lower accuracy, with the worst drop being -0.06. (We cut off the display for better visualisation of a more densely-populated range.) The ISIC dataset appears to benefit most from STC in most settings, whereas traffic_sign reacts negatively to STC in multiple cases. STC generally shows a stronger tendency to decay over iterations when applied on fewer unlabelled instances. For datasets like mnist and traffic_sign, decay can be observed using FLUTE or URL fine-tuning even with 1000 unlabelled instances.

In general, STC performance tends to either remain stable or decay after the first iteration, so non-iterative STC is the safer option over iterative STC. However, iterative STC is stable with TSA fine-tuning and 1000 unlabelled instances, and small performance gains can be observed over the iterations on some target domains. As the results in Tables 4.3, 4.5, and 4.7 show that a URL extractor with TSA fine-tuning achieves higher estimated accuracy than FLUTE and URL as a CDFSL base algorithm, whether used with FES or not, iterative STC and TSA fine-tuning (with FES) should be used for the best possible CDFSL results in this setting. Even when using 100 unlabelled images per episode, most datasets still exhibit improved performance over supervised learning when applying FES-TSA.

4.5 Conclusion

We show semi-supervised learning algorithms for large labelled datasets may be unsuitable for CDFSL as they frequently exhibit lower estimated accuracy than purely supervised learning. We propose STC, an efficient semi-supervised learning method that is more robust against data scarcity and domain shift

and is compatible with a range of state-of-the-art CDFSL methods utilising nearest-centroid classification, including FES, URL, FLUTE, and a URL extractor with TSA fine-tuning. We evaluate STC extensively and show that it generally improves these CDFSL methods' average performance on the Meta-Dataset benchmark when applied with a moderate number of 1000 unlabelled instances. STC requires no additional backpropagation beyond applying supervised learning, which means it is computationally efficient.

Chapter 5

Bidirectional Snapshot Selection

We have seen that FES is a highly competitive CDFSL method that can be further improved using semi-supervised learning. However, a potential drawback in practice is that it consumes a comparably large amount of memory because it stores individual feature extractors and multiple snapshots, as discussed in Section 3.7. In this chapter, we consider how to prune the FES model using methods inspired by wrapper-based feature subset selection in classic machine learning, such as decision tree induction, treating the *set* of features extracted by one extractor or snapshot as the item to be available for selection. We first cover literature on wrapper-based feature selection. We then define the bidirectional snapshot selection (BSS) method for pruning FES snapshots. Following this, we evaluate BSS against several pruning baselines and analyse its accuracy impact and pruning effectiveness.

5.1 Wrapper-based Feature Selection

Greedy feature subset selection using "wrapper" methods that apply cross-validation was popularised by seminal work presented in Kohavi and John (1997). Commonly, it is implemented using forward selection as the search strategy, so it starts with an empty subset of features. Then, given a set of candidate features, it separately includes each candidate in the existing subset and evaluates the subset's features in concert using cross-validation. The can-

didate that leads to the highest cross-validation performance is selected into the feature subset. Intuitively, the feature with the highest cross-validation performance on its own is first selected into the empty subset, then the one with the highest cross-validation performance in concert with the first selection is selected into the subset, and so on. Commonly, the search proceeds until all candidates have been selected or no remaining candidate improves cross-validation performance when added to the existing feature subset, and this subset is returned as the resulting feature selection. This approach of evaluating a candidate jointly with the existing subset is more robust against correlated features than evaluating the candidate independently, as a feature needs to yield the highest improvement of the existing subset's predictive performance to merit its selection. Note that consecutive FES snapshots in fine-tuning are inherently correlated.

Linear forward selection is a variant of forward selection that only evaluates a limited pool of candidates at each step (Gütlein et al., 2009). Given a feature set without explicit order, this method ranks each feature by its individual cross-validation performance and initialises a pool of a user-specified size with the top-ranked features. During the search, this pool can be optionally replenished by the remaining features in ranked order. Gütlein et al. (2009) found linear forward selection is faster, finds smaller subsets, and sometimes improves accuracy compared to naive forward selection. As FES snapshots of the same extractor are implicitly ordered because they are sequentially generated during fine-tuning, this search technique is a natural fit for computationally efficient pruning of FES ensembles.

5.2 Pruning FES with Bidirectional Snapshot Selection

We now present bidirectional snapshot selection (BSS) for pruning FES snapshots, shown as pseudo code in Algorithm 3 and visualised in Figure 5.1.

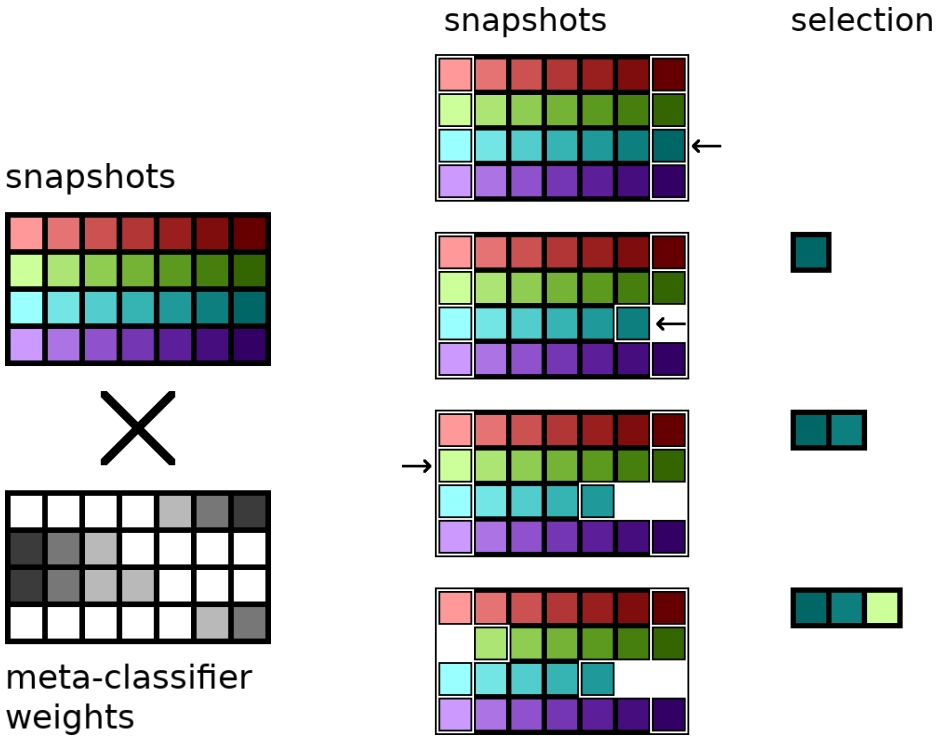
Algorithm 3 Bidirectional snapshot selection

Input: Support set instances X and labels Y, K extractors $\Phi_1, ..., \Phi_K$

Parameter: fine-tuning iterations J and patience P

Output: Snapshot subset $\mathbb{S}$

Split the support set into X_1, X_2 and Y_1, Y_2 using stratified cross-validation.
for $k \in [1, K]$ **do**
    Fine-tune Φ_k using X_1 and Y_1 and save snapshots $\Phi_{k,0}^{CV_1}, ..., \Phi_{k,J}^{CV_1}$.
    Fine-tune Φ_k using X_2 and Y_2 and save snapshots $\Phi_{k,0}^{CV_2}, ..., \Phi_{k,J}^{CV_2}$.
end for
Let selected subset $\mathbb{S} = \varnothing$, temporary subset $\mathbb{T} = \varnothing$, best loss $\ell_{best} = \infty$, patience value $PV = P$, candidate pool $\mathbb{P} = \{(k, j) | k \in [1, K], j \in \{0, J\}\}$.
while $\mathbb{P} \neq \varnothing \wedge PV \geq 0$ **do**
    Let temporary best cross-validation loss $\ell_{temp} = \infty$.
    for $p \in \mathbb{P}$ **do**
        Apply $\{\Phi_e^{CV_1} | e \in \mathbb{T} \cup \{p\}\}$ to X_2 to extract logits L^{CV_2}.
        Apply $\{\Phi_e^{CV_2} | e \in \mathbb{T} \cup \{p\}\}$ to X_1 to extract logits L^{CV_1}.
        Train FES with L^{CV_1} and Y_1 and obtain loss ℓ_{CV_2} with L^{CV_2} and Y_2.
        Train FES with L^{CV_2} and Y_2 and obtain loss ℓ_{CV_1} with L^{CV_1} and Y_1.
        if $\ell_{CV_1} + \ell_{CV_2} < \ell_{temp}$ **then**
            Let best candidate $b = p$, $\ell_{temp} = \ell_{CV_1} + \ell_{CV_2}$.
        end if
    end for
    Let $\mathbb{T} = \mathbb{T} \cup \{b\}$; $\mathbb{P} = (\mathbb{P} \cup Pre_b \cup Suc_b) \setminus \mathbb{T}$.
    if $\ell_{temp} < \ell_{best}$ **then**
        Let $\mathbb{S} = \mathbb{T}, \ell_{best} = \ell_{temp}$, $PV = P$.
    else
        Let $PV = PV - 1$.
    end if
end while
return $\mathbb{S}$



(a) (b)

Figure 5.1: (a) The FES stacking classifier weights snapshots to combine their predictions. In the matrix of snapshots, each colour represents a feature extractor, and its tone represents a snapshot's fine-tuning iteration. In the weight matrix, darker colours represent higher weights assigned to corresponding snapshots. (b) Bidirectional snapshot selection. Snapshots in the candidate pool are marked with light borders. The arrow marks the snapshot whose inclusion in the selected subset leads to the largest decrease in cross-validated loss. When a snapshot is selected, its predecessor or successor (depending on direction) from fine-tuning is added to the pool.

Given a support set containing instances X and labels Y, K source domain feature extractors $\Phi_1, ..., \Phi_K$, each to be fine-tuned for J iterations, and a patience value $P \in \mathbb{N}$ dictating the number of consecutive iterations without improvement permitted by the algorithm, BSS returns a subset of the $K \times (J + 1)$ snapshots.

BSS leverages the 2-fold stratified cross-validation performed by FES to guide the search, where the support set is split into two partitions of instances, which alternate to serve for extractor fine-tuning and logit extraction, leading to two partitions of logits. Fine-tuning extractors is generally the most computationally heavy process of FES, and BSS can utilise its outcome without extra cost. BSS alternates the two logit partitions to use one to train the FES stacking classifier weights and the other as input to the trained stacking classifier to obtain stacked logits. The stacked logits are used with their corresponding labels to compute cross-validation loss, and losses of the two partitions are summed to represent cross-validation loss of the evaluated subset. Selection of subsets is based on minimising this cross-validation loss.

The search starts with an empty subset and initialises its candidate pool with the first and last snapshots of each extractor's fine-tuning process comprising $2 \times K$ snapshots. Then, it evaluates each candidate by measuring cross-validation loss on the existing subset (initially empty) in addition to the candidate. The candidate that leads to the highest decrease in cross-validation loss is selected into the subset and removed from the pool; an adjacent snapshot is used to replenish the pool. This snapshot is the selected candidate's successor if it is at the beginning of fine-tuning or predecessor if it is at the end, and it is only added to the pool if not already in the subset to prevent redundant selections. For brevity, in Algorithm 3, given a snapshot of the k-th extractor at the j-th iteration $b = (k, j)$, its predecessor set is denoted as $Pre_b = \{(k, j - 1)\}$ if $j - 1 \geq 0$ else $\varnothing$, and its successor set is denoted as $Suc_b = \{(k, j + 1)\}$ if $j + 1 \leq J$ else $\varnothing$.

When the patience parameter P has value zero, if no candidate in the pool

decreases cross-validation loss when added to the subset, the search terminates, and the current subset is returned. For $P > 0$, the process only terminates if no loss decrease is achieved in P consecutive steps. In this case, the candidate leading to the lowest loss increase is tentatively selected, and its selection is only confirmed when a later step within the patience limit achieves lower loss than the previous best subset. Termination returns the subset with the lowest cross-validation loss explored by the search. Depletion of the candidate pool also leads to termination of the search.

Once the subset is returned, FES can be performed on the subset in the same way as it is performed on the full set of snapshots: Weights for the stacking classifier are trained on cross-validation logits of the snapshot subset; when producing predictions, these weights are used to aggregate the output of the same subset of snapshots fine-tuned using all support set data.

5.3 Experimental Setup

We perform evaluation with the Meta-Dataset benchmark, with eight source domains and ten target domains (Triantafillou et al., 2020b). We continue to use the cached episodes sampled in Chapter 3: each episode contains 5 to 50 classes, up to 100 support set instances per class, up to 500 (potentially class-imbalanced) support set instances in total, and 10 query set instances per class. Caching ensures that all pruning strategies are evaluated using the same episodes.

We use FES with TSA fine-tuning to evaluate pruning strategies. FES hyperparameters are consistent with Chapter 3, and TSA hyperparameters are consistent with Li et al. (2022a). Each of the eight source domain extractors is fine-tuned with TSA for 40 iterations, leading to $8 \times 41 = 328$ snapshots in total.

We evaluate three baselines: 1) FES without pruning, 2) exhaustively searching through all *extractor* combinations where selecting an extractor leads

to inclusion of all of its 41 snapshots, and 3) exhaustively searching through all combinations of the extractors' *final* snapshots. We evaluate BSS with patience $= 0$ as our main method. In an ablation study, we evaluate two unidirectional snapshot selection strategies that are akin to BSS but only search in one direction: unidirectional forward snapshot selection (UFSS) initialises its candidate pool with only snapshots from the beginning of fine-tuning, and conversely, unidirectional backward snapshot selection (UBSS) initialises its candidate pool with only snapshots from the end of fine-tuning. We also evaluate BSS, UFSS, and UBSS with patience values from 0, with no patience, to 328, which ensures depletion of the candidate pool. Lastly, we combine the methods in Chapters 3, 4, and 5, and evaluate FES with BSS and STC with 1,000 additional unlabelled instances, and compare it to 1) full supervised FES, 2) full FES with STC, and 3) supervised FES pruned with BSS. We perform paired t-tests with $p = 0.05$ using accuracy of individual episodes to determine statistical significance of differences in accuracy on individual datasets.

5.4 Results

We first compare BSS to the baseline strategies. Then, we visualise pruning results as heatmaps. Following this, we conduct an ablation study with UFSS and UBSS and consider varying patience values. Lastly, we evaluate FES with BSS and STC, and compare to FES without one or both of BSS and STC.

5.4.1 Comparison to Baselines

Table 5.1 compares performance of BSS to that of the baselines. Accuracy and percentage of snapshots remaining after pruning, i.e., $|\mathbb{S}|/(K \times (J+1))$, are reported with their 95% confidence intervals. Mean WG and SG performance and ranks are also reported. Each baseline is evaluated against BSS in paired t-tests. If a statistical significance is detected, i.e., $p < 0.05$, $\bullet$ indicates better accuracy of BSS, whereas $\circ$ indicates better baseline accuracy.

Table 5.1: BSS compared to the baselines: no pruning (full), exhaustively searching through extractors (EE), and exhaustively searching through the last snapshot of each extractor (EL)

	BSS		full	EE		EL	
Dataset	accuracy	ratio	accuracy	accuracy	ratio	accuracy	ratio
ilsvrc.	**56.3±1.2**	3.8±.3	56.2±1.1•	**56.3±1.1**	50.2±1.3	**56.3±1.2**	1.4±.03
omnig.	93.1±0.7	12.3±.8	**93.3±0.6**◦	93.2±0.7	47.5±1.9	**93.3±0.7**◦	1.3±.04
aircraft	**87.8±0.8**	4.9±.4	87.6±0.8	87.6±0.8	50.0±1.3	87.7±0.8	1.3±.03
cu_birds	**80.0±0.9**	4.0±.3	79.9±0.8	**80.0±0.8**	48.9±1.3	79.8±0.9•	1.4±.03
dtd	76.5±0.8	3.2±.3	76.2±0.8•	76.2±0.8•	49.7±1.3	**76.7±0.8**◦	1.3±.03
quickd.	**83.6±0.6**	2.9±.2	83.4±0.6•	83.3±0.6•	42.0±1.3	83.3±0.6•	1.3±.03
fungi	69.5±1.1	3.2±.3	69.4±1.1	**69.7±1.1**◦	43.1±1.4	69.2±1.1•	1.3±.03
vgg_flo.	91.8±0.7	5.6±.5	91.9±0.7	91.9±0.7	53.4±1.3	**92.1±0.7**◦	1.4±.03
WG avg	**79.83**	4.99	79.74	79.78	48.10	79.80	1.34
WG rank	2.49		2.51	**2.45**		2.54	
traffic_s.	**85.7±0.9**	2.9±.3	84.9±1.0•	84.8±1.0•	59.6±1.5	**85.7±0.9**◦	1.4±.03
mscoco	**54.5±1.0**	3.4±.2	54.1±1.0•	54.3±1.0•	59.5±1.6	53.6±1.0•	1.5±.03
mnist	97.0±0.5	6.4±.6	**97.1±0.5**	96.9±0.5•	40.8±1.2	97.0±0.5	1.2±.03
cifar10	78.3±0.9	2.7±.2	78.1±0.9	78.3±0.9	46.4±1.5	**78.5±0.9**◦	1.2±.03
cifar100	**70.6±1.1**	3.5±.3	70.4±1.1•	70.3±1.1•	49.4±1.5	**70.6±1.1**	1.5±.03
CropD.	88.0±0.7	4.6±.4	**88.2±0.7**◦	87.9±0.7	56.9±1.2	87.8±0.7•	1.6±.03
EuroS.	89.3±0.6	2.5±.2	88.8±0.6•	89.0±0.6•	41.3±1.3	**89.4±0.6**	1.2±.03
ISIC	48.3±0.9	3.4±.2	**49.5±0.9**◦	49.2±0.9◦	56.3±1.4	47.8±1.0•	1.5±.03
ChestX	27.0±0.6	3.2±.3	**27.7±0.6**◦	27.6±0.6◦	55.4±1.6	26.7±0.6•	1.4±.03
Food.	**55.7±1.1**	2.9±.2	55.2±1.1•	55.3±1.1•	48.7±1.2	54.6±1.1•	1.6±.03
SG avg	**69.44**	3.55	69.40	69.36	51.43	69.17	1.41
SG rank	**2.37**		2.51	2.53		2.59	

Table 5.2: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results of BSS and the baselines

	WG				SG			
	BSS	F	EE	EL	BSS	F	EE	EL
bidirectional snapshot selection	-	1	1	3	-	3	2	2
full	3	-	2	2	5	-	2	4
exhaustive extractor	2	0	-	2	6	3	-	5
exhaustive last snapshot	3	1	2	-	5	5	4	-

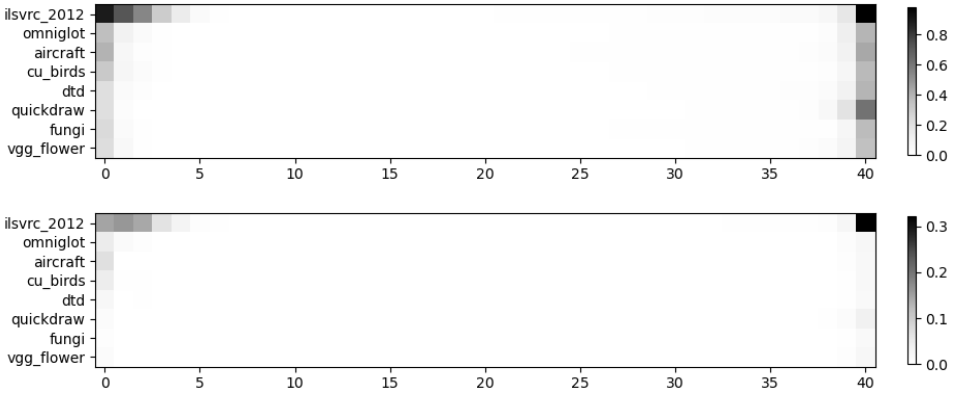
Table 5.2 summarises paired t-test results comparing every strategy in Table 5.1 against every other strategy, organised by WG and SG. Each value represents the number of domains where the strategy in its column achieves significantly higher accuracy than the strategy in its row with $p < 0.05$.

Compared to FES without pruning, BSS achieves higher average SG accuracy while the two exhaustive baselines perform worse. Exhaustive extractor search only removes approximately half of all snapshots. Exhaustively searching through combinations of the final snapshots achieves impressive reduction, as only 8 of the total 328 are considered, but its SG performance is the weakest among the four methods. BSS achieves the highest accuracy while achieving over 95% snapshot reduction. It also has the most wins against the other strategies in the paired t-tests.

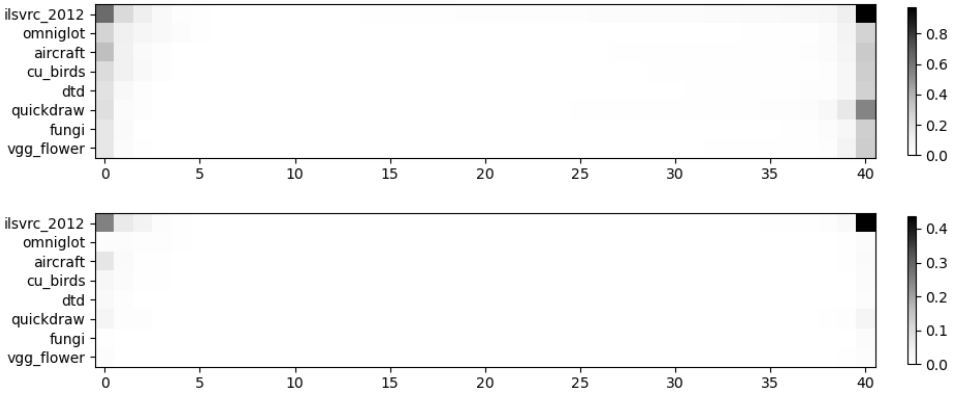
We have also evaluated several other baselines but their accuracy-reduction balance is inferior to the strategies presented in Table 5.1, so are presented in Appendix B. These baseline strategies include: 1) exhaustive search including only each extractor's snapshot with the lowest individual cross-validation loss in the search space, 2) greedy forward selection with at most one snapshot per extractor, performed by including all snapshots in the initial pool and when a snapshot is selected, removing all candidates of the same extractor, 3) greedy forward selection with at most one snapshot per extractor but allowing replacement of a snapshot in the selected subset with another snapshot of the same extractor, and 4) ranking all snapshots by individual cross-validation loss, iterating through the ranking, and selecting snapshots whose inclusion in the subset decreases cross-validation loss.

5.4.2 Visualisation

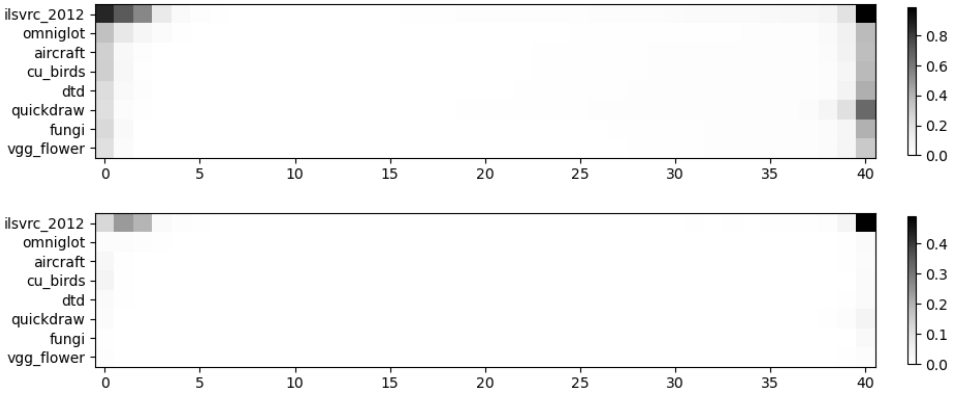
Figures 5.2-5.5 visualise BSS results for the target domains. In each pair of graphs, the top one represents the frequency of each snapshot being selected, and the bottom one represents average weight assigned to each snapshot by FES (noting that unselected snapshots receive zero weight). Results are based



(a) mscoco snapshot frequencies and weights

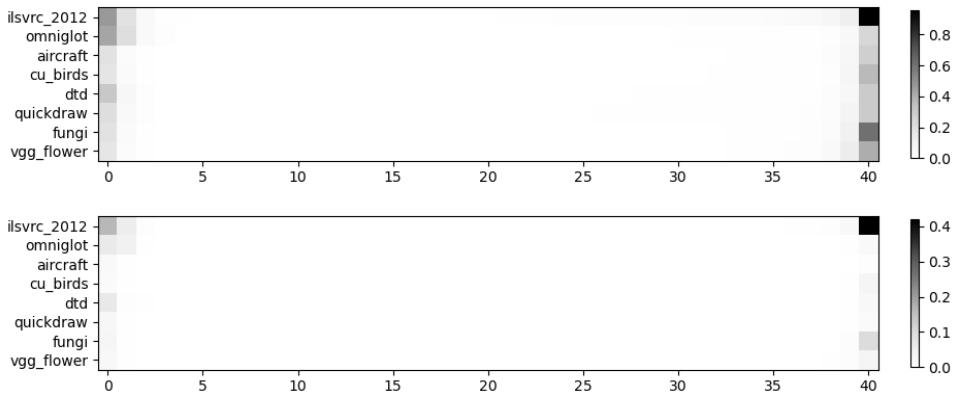


(b) cifar10 snapshot frequencies and weights

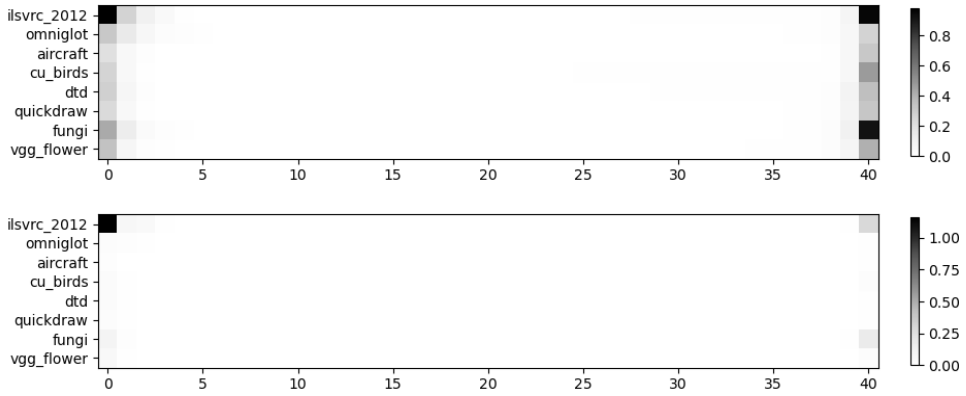


(c) cifar100 snapshot frequencies and weights

Figure 5.2: ImageNet-heavy target domains



(a) EuroSAT snapshot frequencies and weights



(b) Food101 snapshot frequencies and weights

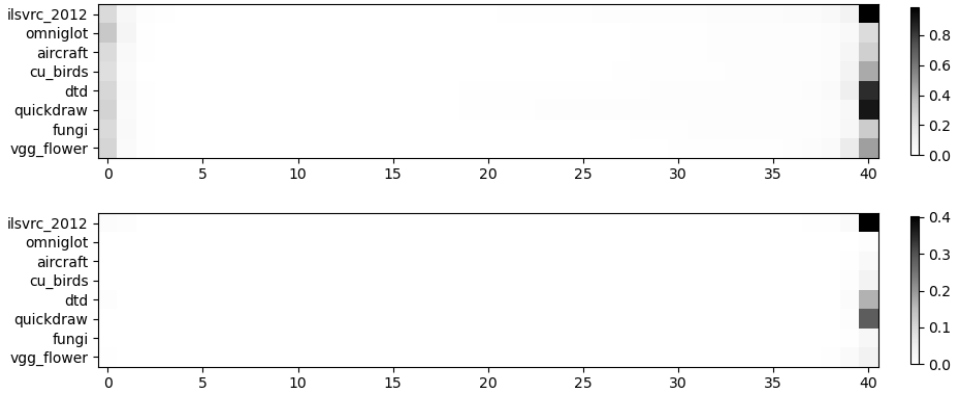
Figure 5.3: ImageNet-heavy target domains

on the 600 episodes of the corresponding target domain.

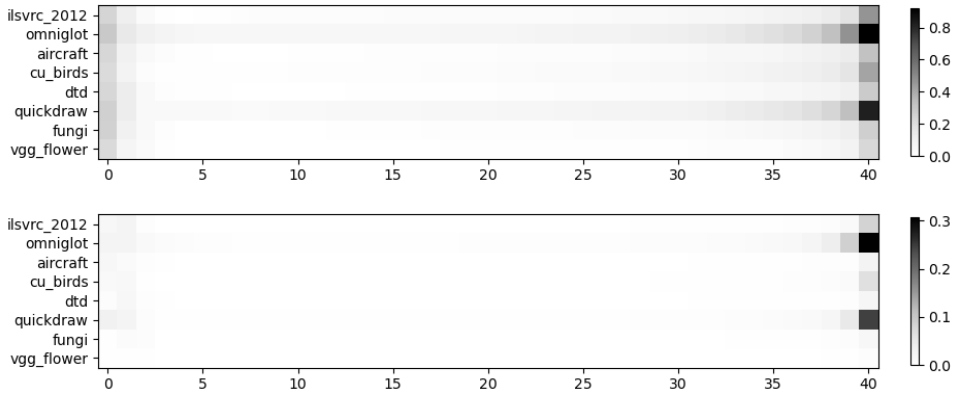
These figures indicate that Meta-Dataset target domains are ImageNet-heavy, as ImageNet snapshots are dominantly selected and weighted in mscoco, cifar10, cifar100, EuroSAT, and Food101. The ImageNet snapshot before fine-tuning is the most heavily weighted in Food 101, as opposed to the one at the end of fine-tuning in the other four domains. BSS also improves FES accuracy in Food101 by 0.5% as shown in Table 5.1.

The other five target domains, comprising traffic_sign, mnist, CropDisease, ISIC, and ChestX, exhibit more diverse compositions of source domains. This is consistent with intuition: they are more different from ImageNet semantically than the other domains.

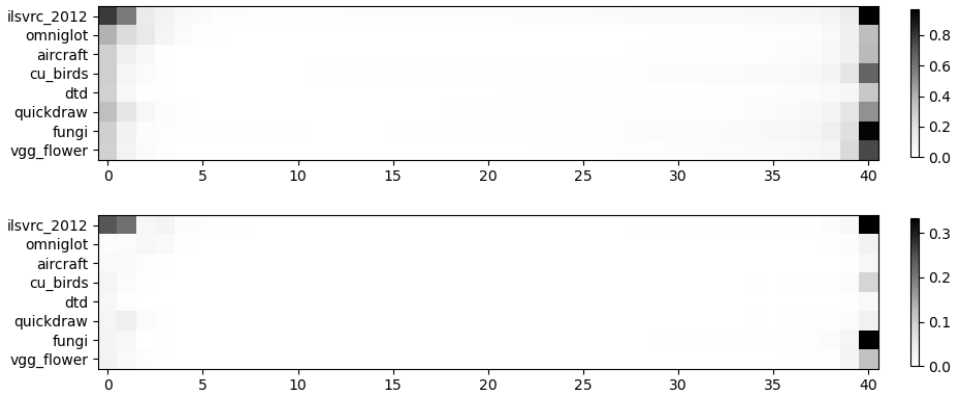
These results indicate that it may be beneficial for a CDFSL benchmark



(a) traffic_sign snapshot frequencies and weights

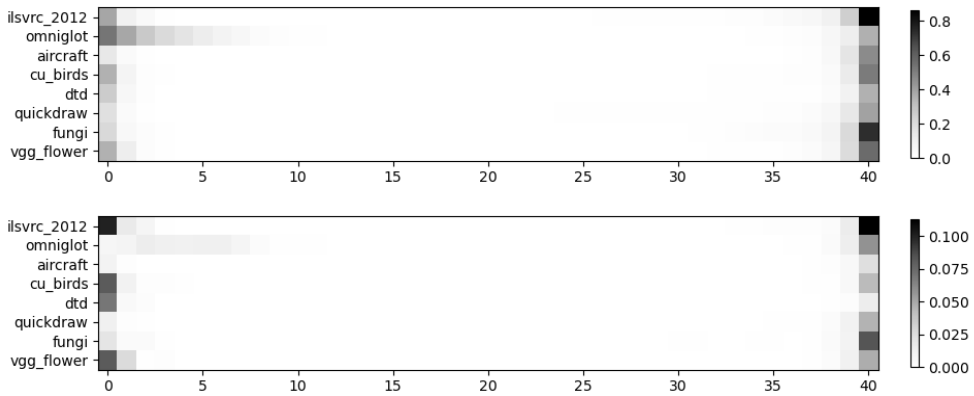


(b) mnist snapshot frequencies and weights

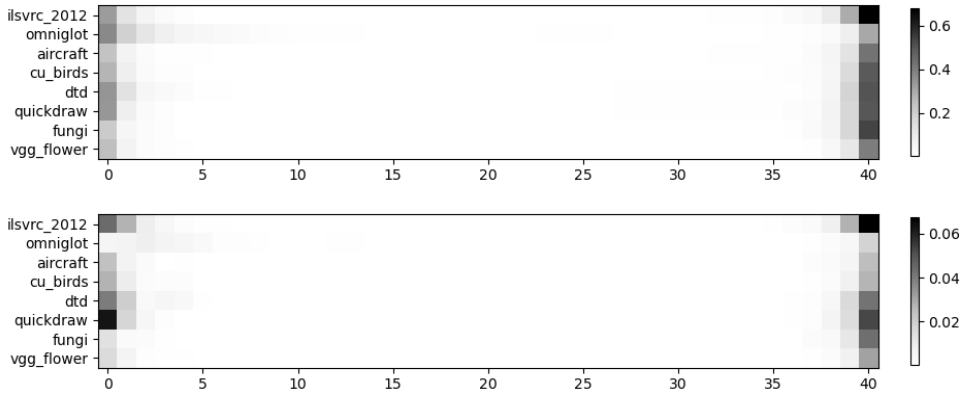


(c) CropDisease snapshot frequencies and weights

Figure 5.4: More diverse target domains



(a) ISIC snapshot frequencies and weights



(b) ChestX snapshot frequencies and weights

Figure 5.5: More diverse target domains

to contain a diverse set of target domains to better measure an algorithm's transfer learning capabilities and represent practical use scenarios. CDFSL methods that produce interpretable compositions of extractors and fine-tuning iterations, such as FES and BSS, can help determine a benchmark's target domain diversity.

Overall, these figures show that BSS is able to remove many snapshots while retaining and sometimes even enhancing interpretability. Moreover, snapshots at the beginning and the end of fine-tuning can both be relevant to a task.

5.4.3 Ablation Study

Table 5.3 shows BSS, UFSS, and UBSS results. Each strategy is performed either with no patience or with maximum patience, which guarantees candidate

Table 5.3: Ablation study results with BSS, UFSS, and UBSS, performed with either zero or maximum patience

	BSS $P = 0$ (Bi)		BSS $P = 328$ (BiP)		UFSS $P = 0$ (F)		UFSS $P = 328$ (FP)		UBSS $P = 0$ (Ba)		UBSS $P = 328$ (BaP)	
Dataset	acc	ratio	acc	ratio	acc	ratio	acc	ratio	acc	ratio	acc	ratio
ilsvrc_2012	56.3±1.2	3.8±0.3	**56.4±1.2**	22.8±1.5	56.0±1.1 •	6.7±0.3	56.2±1.1 •	44.0±2.3	56.3±1.2	7.2±0.4	56.3±1.1	32.7±1.8
omniglot	93.1±0.7	12.3±0.8	93.3±0.7 ○	31.0±2.1	**93.8±0.6** ○	14.3±0.8	93.3±0.6 ○	39.3±2.3	93.2±0.7	11.5±0.8	93.3±0.7 ○	32.7±2.2
aircraft	**87.8±0.8**	4.9±0.4	87.7±0.8	30.1±1.6	**87.8±0.6**	11.1±0.4	87.7±0.8	55.5±2.2	87.7±0.8	5.5±0.4	87.7±0.8	29.5±1.8
cu_birds	80.0±0.9	4.0±0.3	**80.1±0.8**	24.0±1.4	79.6±0.8 •	8.3±0.4	79.8±0.9 •	44.4±2.2	79.8±0.9 •	5.9±0.4	79.9±0.8	55.6±2.4
dtd	**76.5±0.8**	3.2±0.3	76.4±0.8	18.6±1.4	75.7±0.8 •	7.8±0.3	76.0±0.8 •	39.8±2.1	**76.5±0.8**	3.4±0.3	76.4±0.8	24.3±1.6
quickdraw	**83.6±0.6**	2.9±0.2	83.5±0.6 •	16.2±1.1	83.2±0.6 •	6.7±0.3	83.3±0.6 •	40.9±2.1	83.4±0.6 •	5.6±0.4	83.4±0.6 •	26.0±1.6
fungi	69.5±1.1	3.2±0.3	69.5±1.1	26.5±1.5	**69.7±1.1** ○	7.4±0.3	69.5±1.1	42.7±2.0	**69.7±1.1**	6.6±0.4	69.6±1.1 ○	32.4±1.9
vgg_flower	91.8±0.7	5.6±0.5	91.9±0.7	19.9±1.4	91.5±0.7 •	15.1±0.6	91.7±0.7 •	44.7±1.9	**92.0±0.7** ○	4.9±0.5	**92.0±0.7** ○	20.8±1.7
WG avg	79.83	4.99	**79.85**	23.64	79.66	9.68	79.69	43.91	79.83	6.33	79.83	31.75
WG rank	**3.33**		3.35		3.87		3.62		3.43		3.39	
traffic_sign	**85.7±0.9**	2.9±0.3	85.4±1.0 •	30.6±1.9	71.9±1.1 •	4.6±0.2	84.5±1.0 •	85.9±1.7	**85.7±0.9** ○	2.3±0.3	85.4±0.9 •	19.7±1.6
mscoco	**54.5±1.0**	3.4±0.2	54.3±1.0 •	43.5±2.1	53.0±1.0 •	3.3±0.2	53.8±1.0 •	76.0±2.1	53.7±1.0 •	4.2±0.3	53.9±1.0 •	69.3±2.5
mnist	97.0±0.5	6.4±0.6	**97.1±0.5**	20.9±1.4	97.0±0.5	8.7±0.5	96.9±0.5	28.4±1.7	97.0±0.5	5.5±0.6	97.0±0.5	21.1±1.6
cifar10	78.3±0.9	2.7±0.2	78.2±0.9	37.8±2.2	74.6±0.9 •	3.1±0.2	77.3±0.9 •	63.2±2.6	**78.5±0.9** ○	2.6±0.2	78.2±0.9	40.5±2.1
cifar100	**70.6±1.1**	3.5±0.3	70.5±1.1 •	36.4±2.0	66.5±1.1 •	3.7±0.2	70.0±1.1 •	71.6±2.4	**70.6±1.1**	5.3±0.4	70.4±1.1 •	53.2±2.5
CropDisease	88.0±0.7	4.6±0.4	**88.1±0.7** ○	34.4±1.7	85.4±0.7 •	5.7±0.2	87.7±0.7 •	60.0±2.2	87.8±0.7 •	4.2±0.4	87.9±0.7	40.8±1.8
EuroSAT	89.3±0.6	2.5±0.2	89.3±0.6	20.7±1.6	85.7±0.6 •	3.1±0.2	88.2±0.6 •	53.3±2.6	**89.4±0.6**	2.1±0.2	89.3±0.6	23.1±1.6
ISIC	48.3±0.9	3.4±0.2	49.0±0.9 ○	38.3±2.0	47.2±0.9 •	4.1±0.2	**49.4±0.9** ○	66.1±2.2	47.7±1.0 •	2.5±0.1	48.5±0.9	37.1±1.8
ChestX	27.0±0.6	3.2±0.3	27.5±0.6 ○	58.5±2.2	25.4±0.5 •	3.6±0.2	**27.6±0.6** ○	69.8±2.2	27.0±0.6	2.2±0.3	27.2±0.6	40.0±2.1
Food101	**55.7±1.1**	2.9±0.2	**55.7±1.1**	20.1±1.7	53.3±1.1 •	3.7±0.1	54.4±1.1 •	51.7±2.8	54.6±1.1 •	4.6±0.3	55.2±1.1 •	81.2±2.1
SG avg	69.44	3.55	**69.51**	34.12	66.00	4.36	68.98	62.60	69.20	3.55	69.30	42.60
SG rank	**3.03**		3.05		4.75		3.61		3.32		3.23	

Table 5.4: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results of BSS and its unidirectional variants

	WG						SG					
	Bi	BiP	F	FP	Ba	BaP	Bi	BiP	F	FP	Ba	BaP
BSS $P = 0$	-	1	2	1	1	3	-	3	0	2	2	0
BSS $P = 328$	1	-	1	0	0	1	3	-	0	1	2	1
UFSS $P = 0$	5	5	-	4	5	5	9	9	-	9	8	9
UFSS $P = 328$	5	5	2	-	2	6	7	8	0	-	5	6
UBSS $P = 0$	2	2	1	0	-	2	4	5	0	2	-	4
UBSS $P = 328$	1	2	1	0	0	-	4	4	0	2	3	-

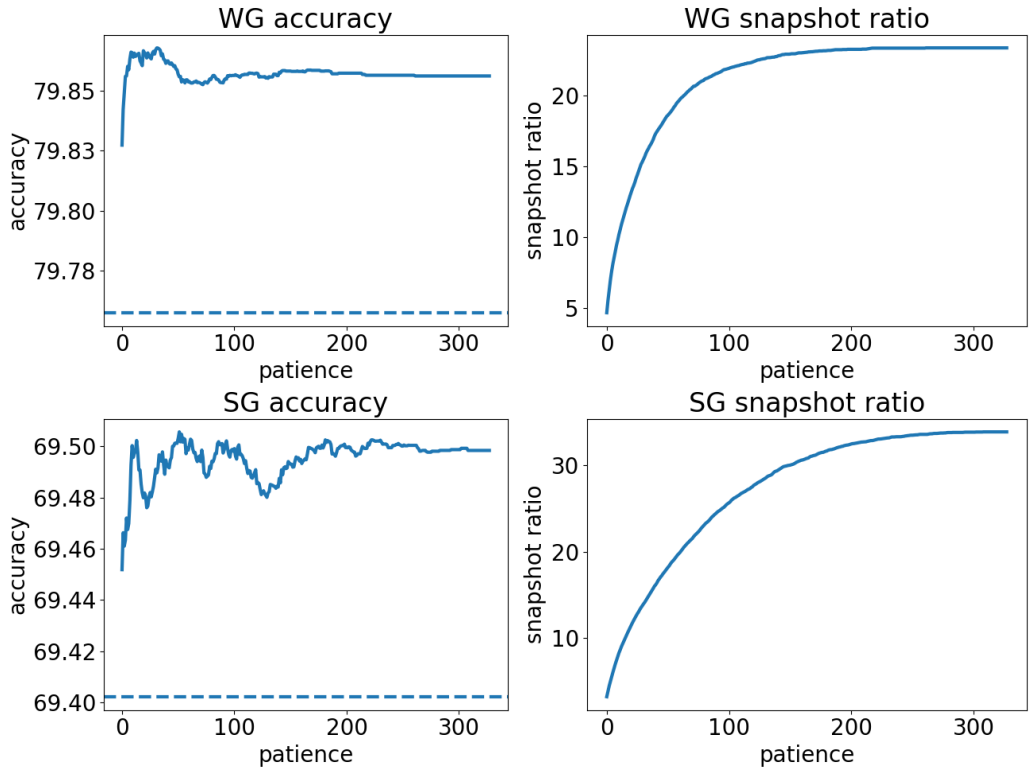


Figure 5.6: BSS performance with respect to patience

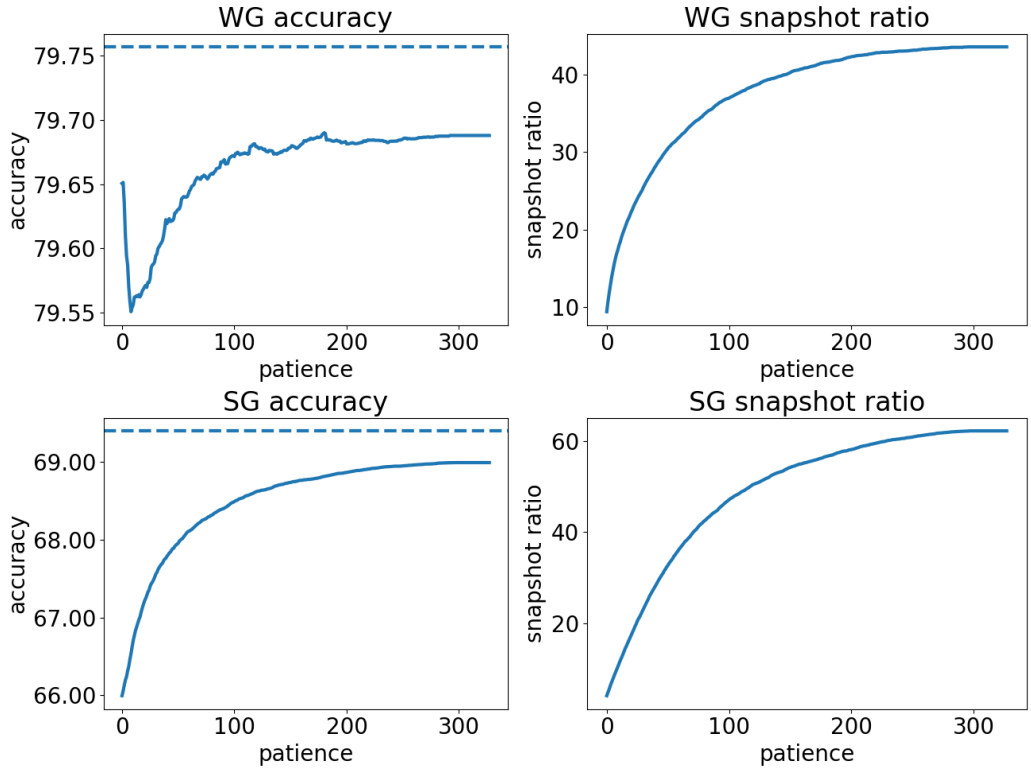


Figure 5.7: UFSS performance with respect to patience

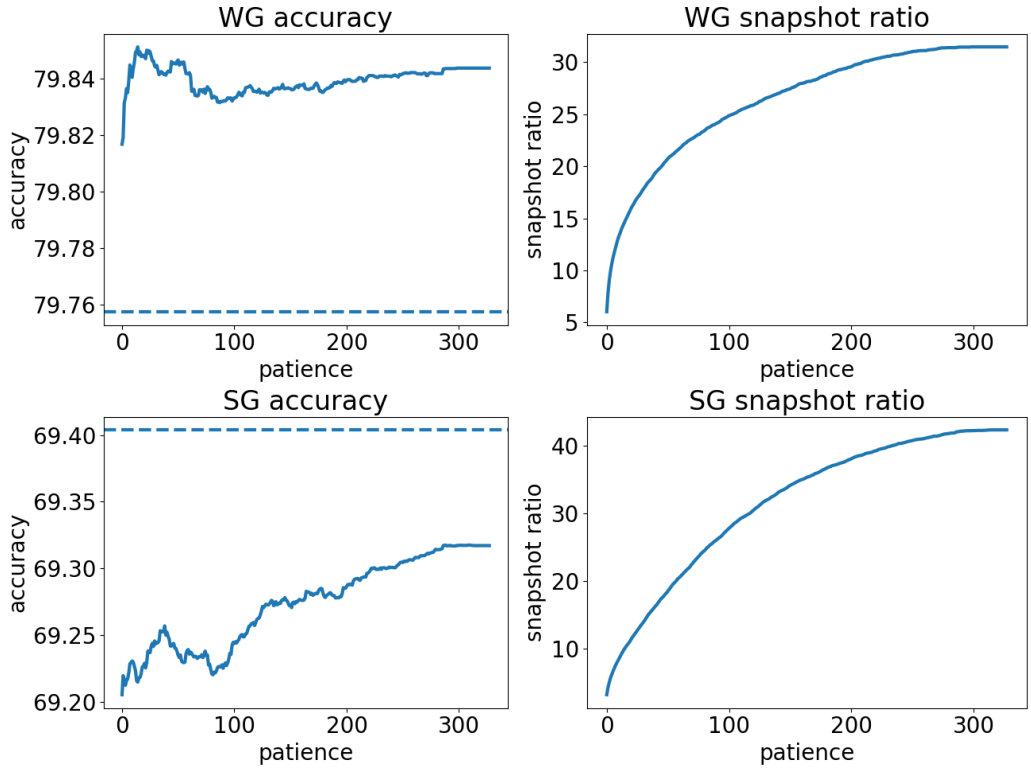


Figure 5.8: UBSS performance with respect to patience

pool depletion. Mean WG and SG performance of the three strategies with respect to patience values between 0 and 328 are shown in Figures 5.6, 5.7, and 5.8, with the dashed line representing accuracy without pruning. Table 5.4 shows paired t-test wins of every strategy in our ablation study against every other strategy.

For both, zero and maximum patience, BSS achieves higher accuracy and snapshot reduction than the two unidirectional variants (Table 5.3). This again indicates that snapshots relevant to a task can be at both the beginning and the end of fine-tuning, and the common practice of adopting the model at the end of fine-tuning may not be best in every CDFSL scenario. Given the same strategy, maximum patience leads to better accuracy but lower reduction. The mean accuracy difference between zero and maximum patience is small for BSS and UBSS, while it is more substantial for UFSS.

Figures 5.6, 5.7, and 5.8 show that only the bidirectional approach consistently outperforms unpruned FES on average in SG, while UFSS and UBSS are consistently inferior. Figure 5.6 also shows that BSS accuracy only varies slightly ($\pm 0.07\%$) given different patience values, while the percentage of remaining snapshots increases substantially with higher patience. Given these observations, we conclude that BSS without patience achieves the best balance between accuracy and reduction.

5.4.4 Semi-supervised Pruning with STC and BSS

Table 5.5 shows FES results with BSS pruning ($P = 0$) and STC semi-supervised learning with 1000 unlabelled instances. The results are compared to those obtained from supervised FES without pruning, as well as FES with only STC semi-supervised learning and FES with only BSS pruning. FES with BSS and STC is set as the reference method in Table 5.5, and the other methods are evaluated against it using the paired t-test. A $\bullet$ indicates significantly better reference method performance according to the paired t-test, and a $\circ$ indicates significantly better performance from the other method. Table 5.6

Table 5.5: Results of FES with/without STC and/or BSS

	BSS STC		full sup	full STC	BSS sup	
Dataset	acc	ratio	acc	acc	acc	ratio
ilsvrc_2012	**56.7±1.1**	4.1±0.2	56.2±1.1 •	56.6±1.1	56.3±1.2 •	3.8±0.3
omniglot	93.7±0.7	13.1±0.8	93.3±0.6 •	**94.0±0.6** ○	93.1±0.7 •	12.3±0.8
aircraft	**88.0±0.7**	4.9±0.3	87.6±0.8 •	87.9±0.7	87.8±0.8 •	4.9±0.4
cu_birds	**80.3±0.8**	4.8±0.3	79.9±0.8 •	80.2±0.8 •	80.0±0.9 •	4.0±0.3
dtd	**76.5±0.8**	3.3±0.3	76.2±0.8 •	76.2±0.8 •	**76.5±0.8**	3.2±0.3
quickdraw	**83.9±0.6**	3.4±0.2	83.4±0.6 •	83.8±0.6 •	83.6±0.6 •	2.9±0.2
fungi	**70.9±1.1**	3.6±0.3	69.4±1.1 •	**70.9±1.1**	69.5±1.1 •	3.2±0.3
vgg_flower	92.3±0.7	5.9±0.5	91.9±0.7 •	**92.4±0.6**	91.8±0.7 •	5.6±0.5
WG avg	**80.29**	5.39	79.74	80.25	79.83	4.99
WG rank	**2.34**		2.67	2.35	2.64	
traffic_sign	**86.5±0.9**	2.8±0.3	84.9±1.0 •	86.0±1.0 •	85.7±0.9 •	2.9±0.3
mscoco	**55.5±1.0**	3.5±0.3	54.1±1.0 •	**55.5±1.0**	54.5±1.0 •	3.4±0.2
mnist	97.3±0.5	6.8±0.6	97.1±0.5 •	**97.4±0.4**	97.0±0.5 •	6.4±0.6
cifar10	**79.0±0.8**	2.7±0.2	78.1±0.9 •	78.6±0.8 •	78.3±0.9 •	2.7±0.2
cifar100	**71.2±1.0**	3.6±0.3	70.4±1.1 •	70.9±1.0 •	70.6±1.1 •	3.5±0.3
CropDisease	88.8±0.7	4.4±0.3	88.1±0.7 •	**89.1±0.6** ○	88.0±0.7 •	4.6±0.4
EuroSAT	**89.6±0.6**	2.5±0.2	88.8±0.6 •	89.0±0.6 •	89.3±0.6 •	2.5±0.2
ISIC	49.7±1.0	3.2±0.2	49.5±0.9	**51.4±0.9** ○	48.3±0.9 •	3.4±0.2
ChestX	27.2±0.6	3.0±0.2	27.7±0.6 ○	**28.4±0.6** ○	27.0±0.6	3.2±0.3
Food101	**56.0±1.1**	3.1±0.2	55.2±1.1 •	55.5±1.1 •	55.7±1.1 •	2.9±0.2
SG avg	70.08	3.56	69.39	**70.18**	69.44	3.55
SG rank	**2.22**		2.83	2.25	2.71	

Table 5.6: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results of FES with/without BSS and/or STC

	WG				SG			
	BSTC	Fsup	FSTC	Bsup	BSTC	Fsup	FSTC	Bsup
BSS STC	-	0	1	0	-	1	3	0
full sup	8	-	7	2	8	-	10	5
full STC	3	0	-	1	5	0	-	2
BSS sup	7	1	5	-	9	3	8	-

shows paired t-test results of all four methods compared to each other.

Table 5.5 shows that semi-supervised learning with STC consistently improves FES performance in both pruned and unpruned settings. FES with STC pruned with BSS achieves slightly lower average strong generalisation accuracy than its unpruned counterpart, but with only 3.56% of the snapshots on average. According to the paired t-test, FES with STC pruned with BSS obtains five wins and three losses against its unpruned counterpart in strong generalisation, as shown in Tables 5.5 and 5.6. This may explain the pruned version's higher ranking despite its lower mean accuracy compared to the unpruned version.

FES with STC and patient BSS pruning ($P = 328$) is also evaluated, but its performance is not substantially different from that of the greedy ($P = 0$) version. Its results can be found in Appendix C.

5.5 Conclusion

We propose BSS, a pruning strategy for FES based on linear forward selection that leverages its cross-validation process and the ordered nature of FES snapshots. BSS achieves over 95% average snapshot reduction in FES with TSA fine-tuning while retaining state-of-the-art accuracy in strong generalisation. It outperforms several baselines and unidirectional snapshot selection strategies, indicating that relevant snapshots can be found at both the beginning and the end of fine-tuning. We also show that visualisation of BSS models aids semantic analysis of target domains. Lastly, we evaluate BSS with STC and show their combination maintains the advantages of both algorithms: increased accuracy with additional unlabelled instances is achieved with over 95% average snapshot reduction.

Chapter 6

Conclusion

This thesis began by introducing cross-domain few-shot learning and presenting an argument that recent universal model-based CDFSL methods have drawbacks in meta-training overhead and extractor collection flexibility, limiting their use in some practical scenarios. Hypothesis 1.1 stated that CDFSL can be performed effectively without a universal model by suitably combining individual source domain-pretrained feature extractors.

We presented feature extractor stacking based on classic stacked generalisation for CDFSL. FES creates snapshots from fine-tuning independent extractors on the support set, uses cross-validation to avoid overfitting from support data reuse, and trains a simple stacking classifier to appropriately weight the snapshots as a form of snapshot selection. We also presented two variants of FES: convolutional FES and regularised FES. ConFES utilises a 1D depthwise kernel hierarchy to reduce the parameter count of its stacking classifier. ReFES applies fused lasso regularisation to training its stacking classifier to smooth weight distributions over adjacent snapshots. FES, ConFES, and ReFES were shown to advance the state of the art on the Meta-Dataset benchmark, outperforming recent universal model-based methods such as universal representation learning, few-shot learning with a universal template, and a URL model fine-tuned with task-specific adaptors. This answers Research Question 1.1: a CDFSL method such as FES can attain state-of-the-art performance without

relying on a universal model.

We discussed practical advantages of FES in real-world scenarios compared to recent methods based on universal models:

1. FES can work with out-of-the-box heterogeneous extractors.

2. If the extractors are readily available, FES does not require their pre-training data down-stream.

3. The FES stacking classifier requires little hyperparameter tuning.

4. FES is computationally cheaper, unless the number of few-shot learning tasks is very large, for example, in the thousands, and the total cost of performing FES on all tasks begins to exceed that of training a universal model once.

We argued that, to field practitioners who wish to use extractors and fine-tuning algorithms specific to their work, FES is likely more flexible and user-friendly than universal-model methods.

Semi-supervised learning is an intuitive approach to improving few-shot accuracy with additional unlabelled data. We investigated semi-supervised CDFSL scenarios where additional unlabelled instances are available for few-shot training. It was shown that semi-supervised learning algorithms for large labelled datasets may be unsuitable for CDFSL as they easily go astray due to label scarcity and frequently exhibit lower estimated accuracy than purely supervised learning. We proposed self-trained centroids, an efficient semi-supervised learning method for CDFSL based on nearest-centroid classification that is more robust against data scarcity and domain shift. STC applies self-training to the nearest-centroid classification head only instead of all trainable parameters. STC is compatible with FES, as well as a range of recent CDFSL methods utilising nearest-centroid classification, including URL, FLUTE, and a URL extractor with TSA fine-tuning. We evaluated STC extensively and showed that it generally improves these CDFSL methods' average performance

on the Meta-Dataset benchmark when applied with a moderate number of 1000 unlabelled instances. This answers Research Question 1.2: a semi-supervised method such as STC can be applied to pretrained feature extractors and consistently improve CDFSL accuracy over supervised learning. STC is also computationally efficient because it requires no additional backpropagation beyond applying supervised learning.

FES with all its snapshots incurs a high storage cost. We investigated pruning methods to reduce this storage cost by pruning snapshots deemed irrelevant to a few-shot task. We proposed bidirectional snapshot selection, a pruning strategy for FES based on linear forward selection that leverages its cross-validation process and the ordered nature of FES snapshots. BSS performs linear forward selection starting with FES snapshot at the beginning and end of fine-tuning and replenishes its selection pool with neighbours of selected snapshots. BSS achieves over 95% average snapshot reduction in FES with TSA fine-tuning while retaining state-of-the-art accuracy in strong generalisation. It outperforms several baselines and unidirectional snapshot selection strategies, indicating that relevant snapshots can be found at both the beginning and the end of fine-tuning. We showed that visualisation of BSS models aids semantic analysis of target domains. Lastly, we combined our proposed novel methods in this thesis—FES, STC, and BSS—and showed that they function well together, as they achieve state-of-the-art CDFSL accuracy on the meta-dataset benchmark with only 5% of FES snapshots on average using TSA fine-tuning. This answers Research Question 1.3: BSS can be applied to FES to drastically reduce its storage cost while retaining the same level of CDFSL accuracy.

The results of the previous chapters are summarised in Figure 6.1. All methods are evaluated using the same cached episodes.

We conclude that our findings support Hypothesis 1.1 that CDFSL can be performed effectively without a universal model by suitably combining individual source domain feature extractors.

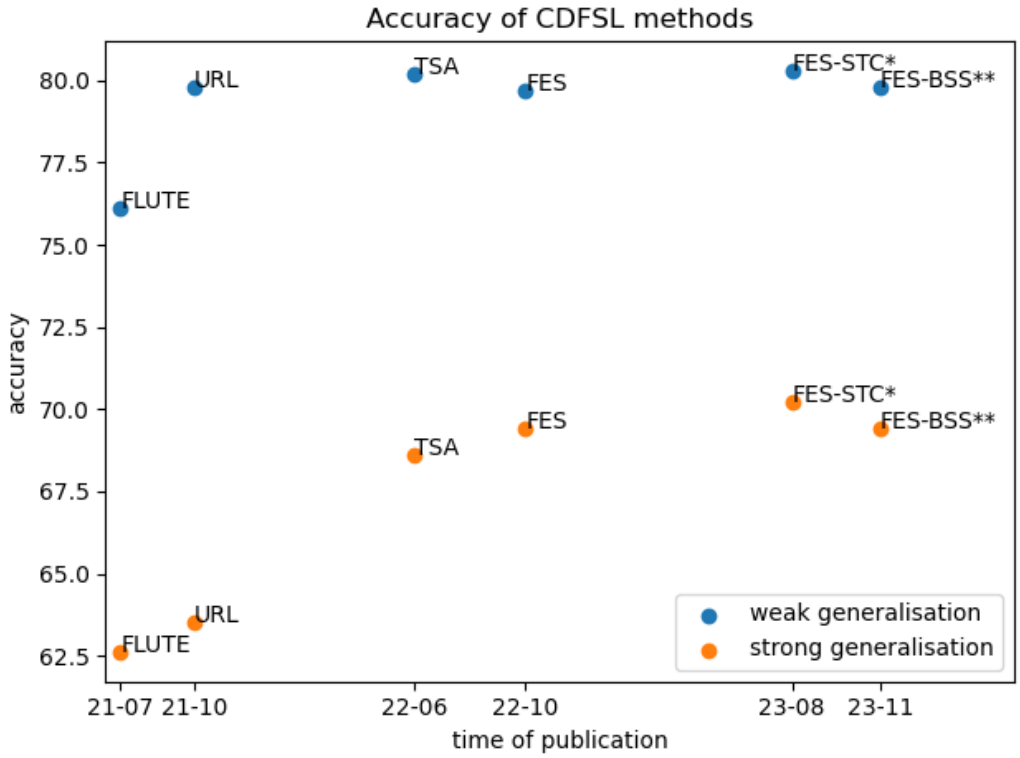


Figure 6.1: Meta-Dataset accuracy of FES, STC, and BSS compared to methods in the literature. The results are from the previous chapters and based on the same cached episodes. FES, FES-STC, and FES-BSS presented here use TSA fine-tuning. *FES-STC has access to an additional 1000 unlabelled instances per few-shot episode. **BSS is yet to be published.

6.1 Limitations and Future Work

FES exhibits good CDFSL performance with multiple source domains. It may be feasible to generalise it to other multi-domain learning problems, for example, multi-domain transfer learning with a more substantial amount of labelled target domain training data. More training data may lead to longer fine-tuning time for each extractor, so it may be beneficial to adopt methods and techniques that improve fine-tuning efficiency and/or prune irrelevant extractors in early fine-tuning iterations.

FES maintains no prior bias to any source domain extractor, and its posterior bias depends on the support set only. In scenarios where prior knowledge is available regarding source and target domain relations, it may be beneficial to enable the user to apply explicit prior biases to certain source domains.

This could be achieved in the form of regularisation, for example, different regularisation pressures are applied to weights associated with different source domains.

FES produces a linear combination of snapshots, which is unable to facilitate nonlinear interactions among outputs of different snapshots. This may limit FES' power to combine knowledge from different extractors and recognise patterns in an extractor's fine-tuning behaviour. A nonlinear stacking classifier may be a good starting point for addressing this issue, although some desirable properties of the linear classifier might not persist in this case, such as robustness against overfitting and convex optimisation.

Stability of each snapshot against outliers in a support set may be measured using its accuracy discrepancy between the two cross-validation folds, and a more accurate measure may be obtained by increasing the number of cross-validation folds and/or repeating cross-validation over multiple runs. Mechanisms may be implemented in FES to detect outliers in support sets and favour snapshots that exhibit better stability.

Chapter 4 focuses on showing the benefits obtained with simple centroid-based self-training. It may be possible to modify STC in certain ways to make better use of an iterative process, in order to achieve consistent performance gains over multiple iterations and ultimately better semi-supervised CDFSL performance. Another potential modification would be to weigh the unlabelled centroids less when averaging with the labelled centroids if the unlabelled set is small, which may reduce STC performance loss with very small unlabelled sets.

STC applies to in-class unlabelled instances, as the unlabelled instances belong to the same set of classes as the labelled instances. Initial results not included in this thesis show that out-of-class unlabelled instances are unsuitable for STC and usually lead to performance deterioration, even when the unlabelled instances are in the same domain as the labelled instances. For example, given a transport classification problem to differentiate cars and mo-

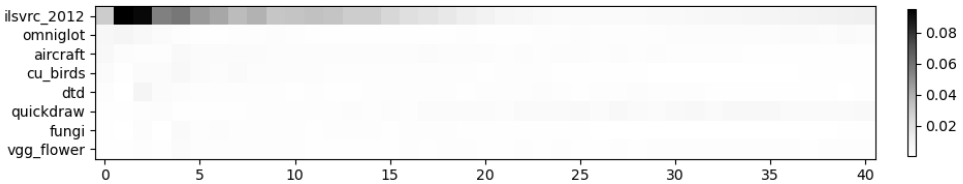
torcycles with labelled training images in both classes, unlabelled images of carts and bicycles should still be helpful as they also belong to the transport domain and have features relevant to the classification task, for example, the number of wheels. STC is currently unable to reliably utilise out-of-class in-domain unlabelled instances, but it may be beneficial to investigate whether it can be modified to exploit such information: in practice, out-of-class instances may be present in the unlabelled set as either noise or additional data.

BSS adds neighbours of selected snapshots into its pool, which means its selections will be a set of consecutive snapshot sequences. This leads to a more efficient search space but also excludes a significant set of possible combinations—those containing non-consecutive snapshots. New BSS-based methods could be developed to skip snapshots when adding new ones to the pool or remove previously selected snapshots. The main challenges in doing so are maintaining a manageable search space and limiting overfitting when selecting snapshots.

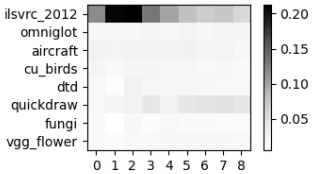
Appendix A

Additional Heatmaps

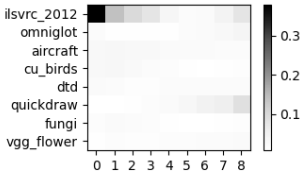
Additional heatmaps visualising kernel weights on target domains with TSA fine-tuning are shown by Figures A.1-A.9.



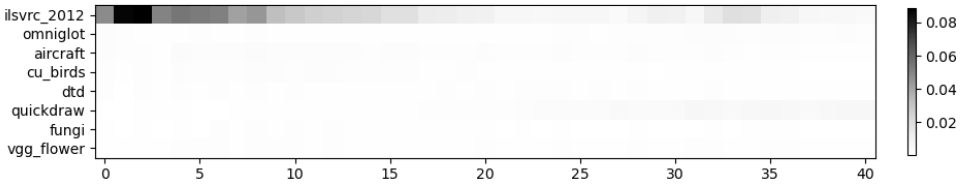
(a) FES kernel



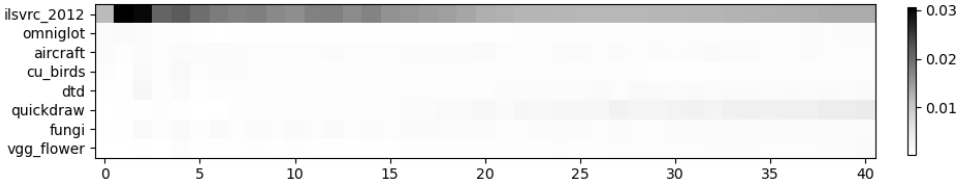
(b) ConFES depthwise kernel



(c) ConFES global kernel

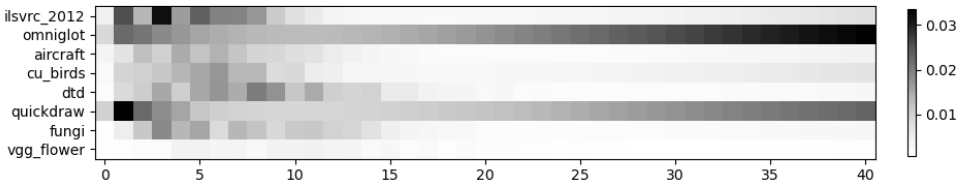


(d) ConFES expanded kernel

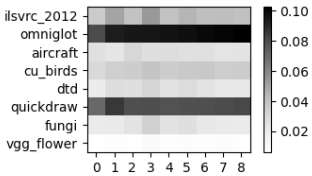


(e) ReFES kernel

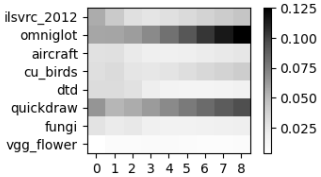
Figure A.1: Kernels for mscoco



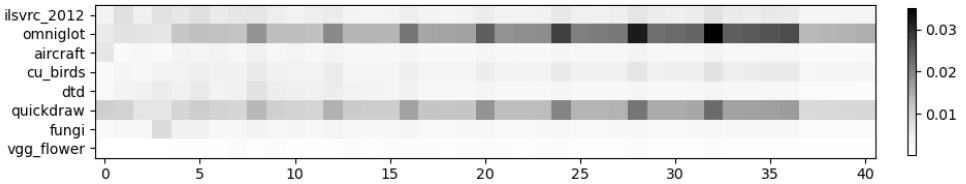
(a) FES kernel



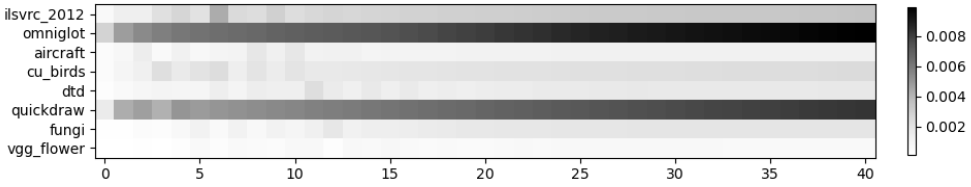
(b) ConFES depthwise kernel



(c) ConFES global kernel

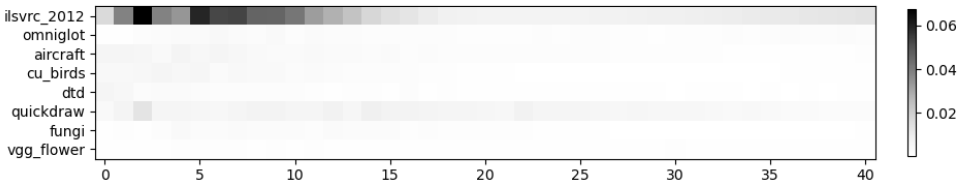


(d) ConFES expanded kernel

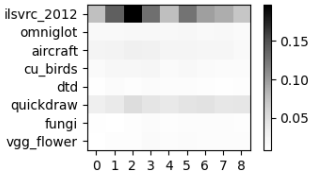


(e) ReFES kernel

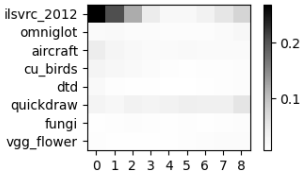
Figure A.2: Kernels for mnist



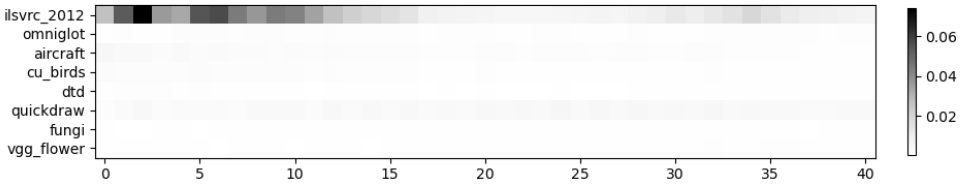
(a) FES kernel



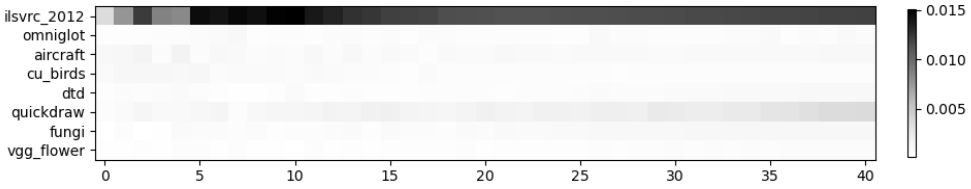
(b) ConFES depthwise kernel



(c) ConFES global kernel

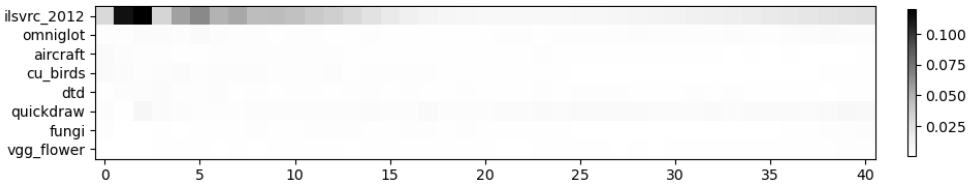


(d) ConFES expanded kernel

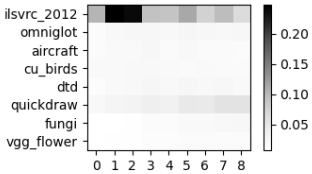


(e) ReFES kernel

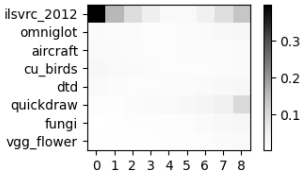
Figure A.3: Kernels for cifar10



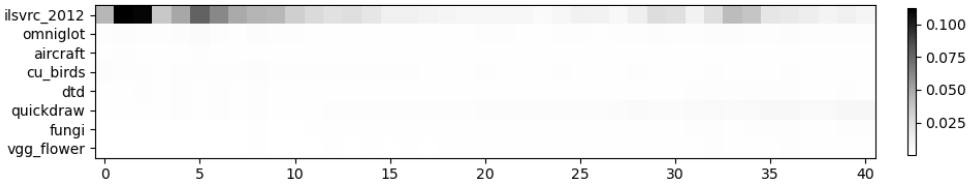
(a) FES kernel



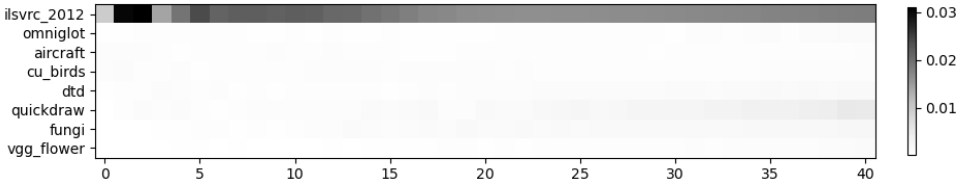
(b) ConFES depthwise kernel



(c) ConFES global kernel

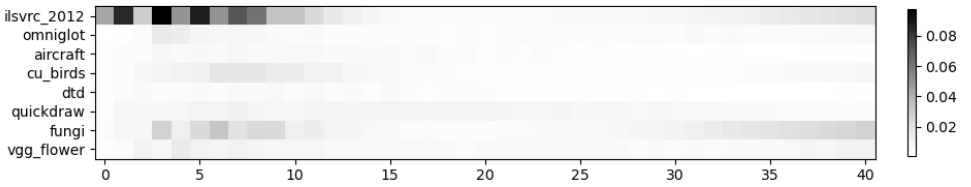


(d) ConFES expanded kernel

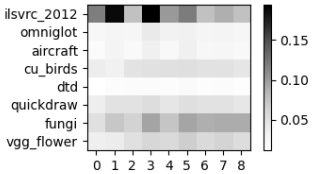


(e) ReFES kernel

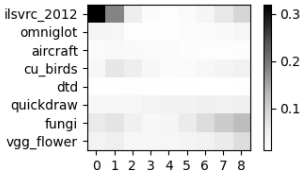
Figure A.4: Kernels for cifar100



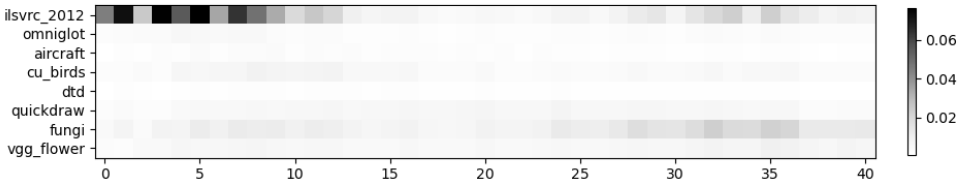
(a) FES kernel



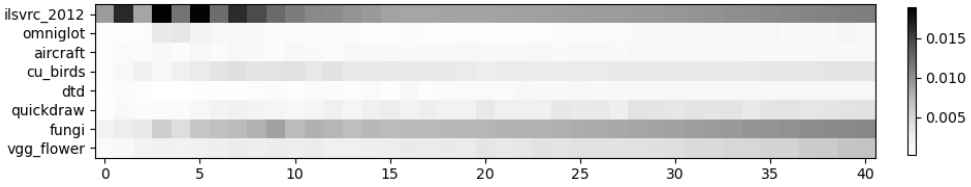
(b) ConFES depthwise kernel



(c) ConFES global kernel

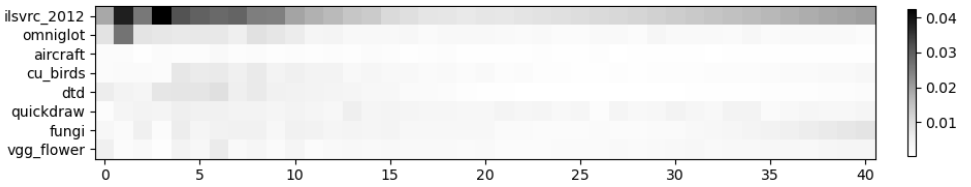


(d) ConFES expanded kernel

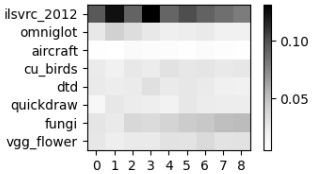


(e) ReFES kernel

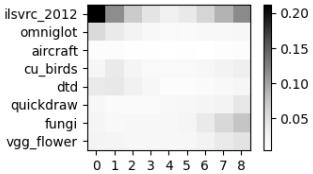
Figure A.5: Kernels for CropDisease



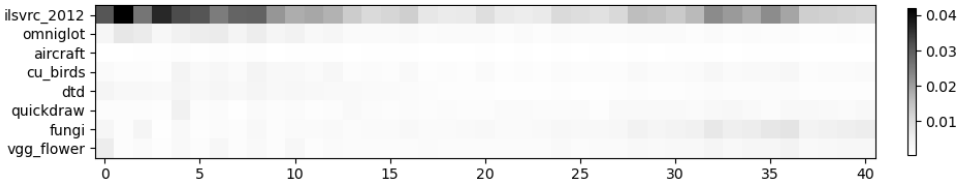
(a) FES kernel



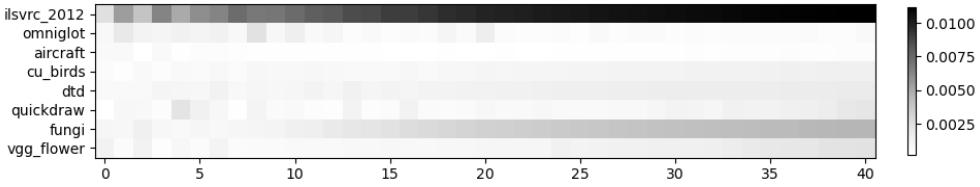
(b) ConFES depthwise kernel



(c) ConFES global kernel

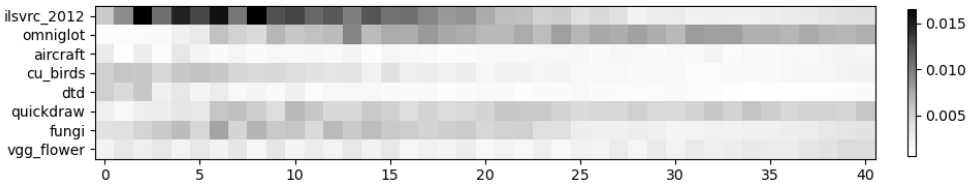


(d) ConFES expanded kernel

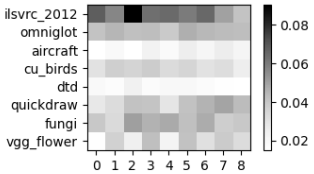


(e) ReFES kernel

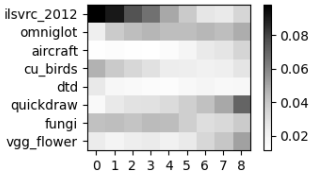
Figure A.6: Kernels for EuroSAT



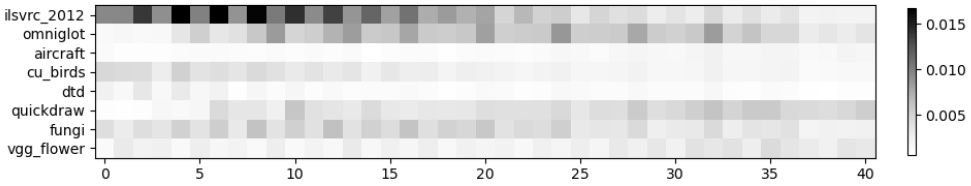
(a) FES kernel



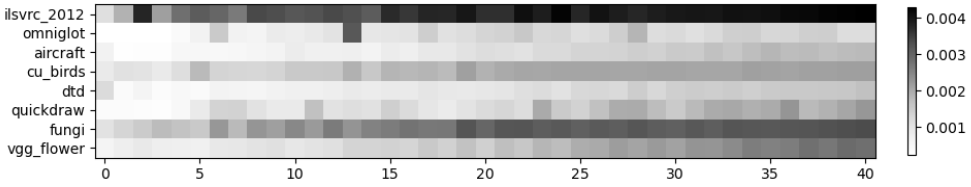
(b) ConFES depthwise kernel



(c) ConFES global kernel

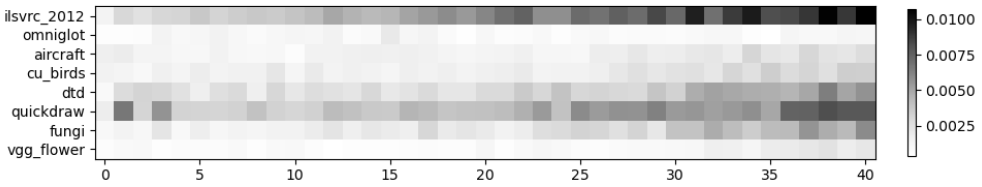


(d) ConFES expanded kernel

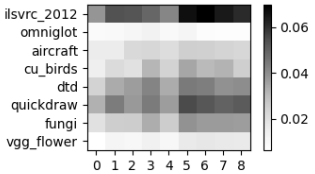


(e) ReFES kernel

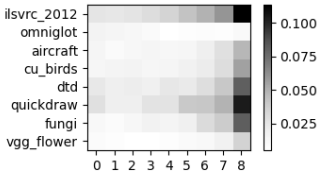
Figure A.7: Kernels for ISIC



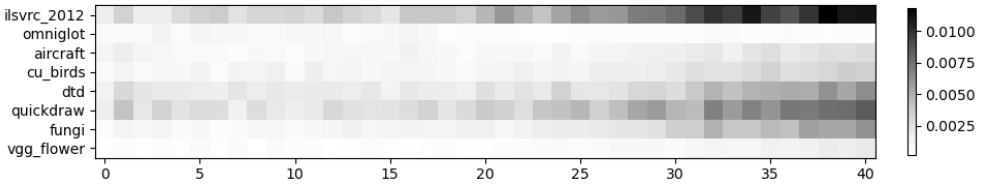
(a) FES kernel



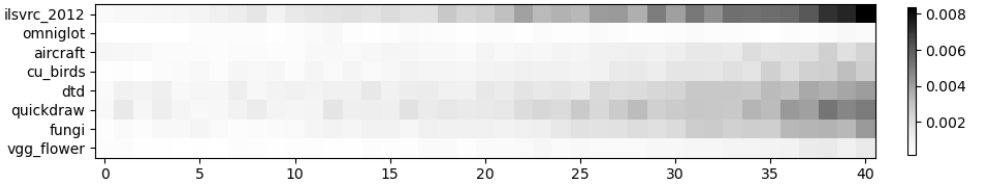
(b) ConFES depthwise kernel



(c) ConFES global kernel

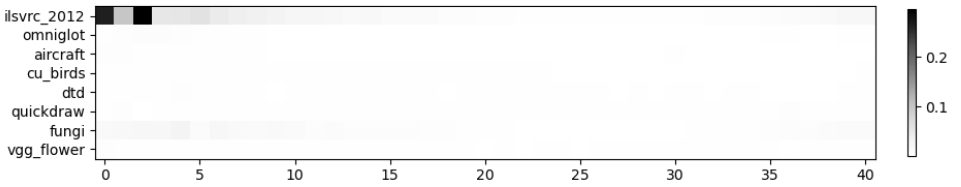


(d) ConFES expanded kernel

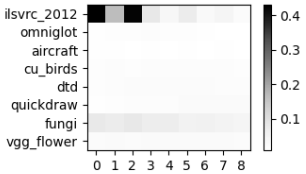


(e) ReFES kernel

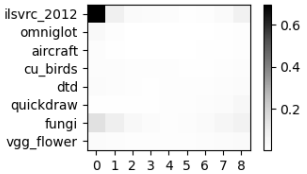
Figure A.8: Kernels for ChestX



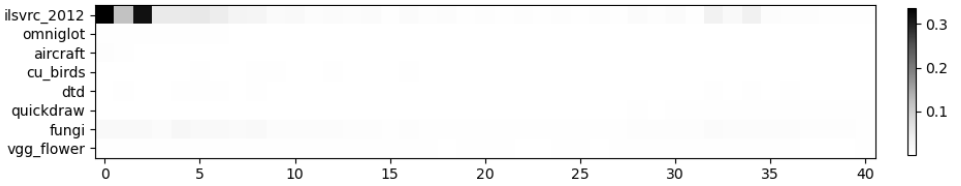
(a) FES kernel



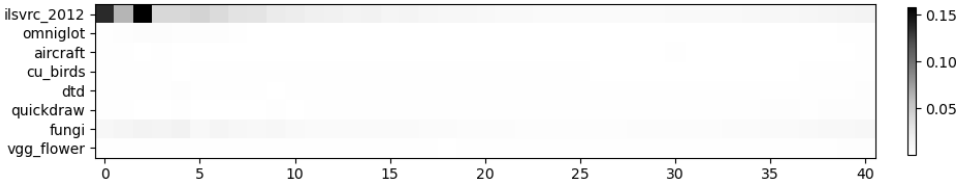
(b) ConFES depthwise kernel



(c) ConFES global kernel



(d) ConFES expanded kernel



(e) ReFES kernel

Figure A.9: Kernels for Food101

Appendix B

Additional Pruning Baselines

Additional baselines are compared to BSS and full FES without pruning in Table B.1. They include: 1) exhaustive search including only each extractor's snapshot with the lowest individual cross-validation loss in the search space, 2) greedy forward selection with at most one snapshot per extractor, performed by including all snapshots in the initial pool and when a snapshot is selected, removing all candidates of the same extractor, 3) greedy forward selection with at most one snapshot per extractor but allowing replacement of a snapshot in the selected subset with another snapshot of the same extractor, and 4) ranking all snapshots by individual cross-validation loss, iterating through the ranking, and selecting snapshots whose inclusion in the subset decreases cross-validation loss. Note that baselines 1), 2), and 3) select at most one snapshot per extractor, leading to small confidence intervals in snapshot ratios that are below the precision shown in Table B.1.

Table B.1: BSS and full FES compared to additional baselines

	BSS		full	exhaustive best		one per extractor		one per extr. w. repl.		ranked iteration	
Dataset	accuracy	ratio	accuracy	accuracy	ratio	accuracy	ratio	accuracy	ratio	accuracy	ratio
ilsvrc_2012	**56.3±1.2**	3.8±0.3	56.2±1.1	55.9±1.2	1.3±0.0	55.9±1.1	1.8±0.0	55.9±1.1	1.7±0.0	56.2±1.1	12.8±0.5
omniglot	93.1±0.7	12.3±0.8	**93.3±0.6**	**93.3±0.7**	1.3±0.0	**93.3±0.7**	1.6±0.0	**93.3±0.7**	1.6±0.0	**93.3±0.7**	17.7±1.0
aircraft	**87.8±0.8**	4.9±0.4	87.6±0.8	**87.8±0.8**	1.3±0.0	87.6±0.8	1.8±0.0	87.6±0.8	1.8±0.0	87.7±0.8	11.4±0.6
cu_birds	**80.0±0.9**	4.0±0.3	79.9±0.8	79.8±0.9	1.4±0.0	79.6±0.9	1.7±0.0	79.6±0.9	1.7±0.0	**80.0±0.8**	14.0±0.5
dtd	**76.5±0.8**	3.2±0.3	76.2±0.8	75.7±0.8	1.3±0.0	75.9±0.8	1.7±0.0	76.0±0.8	1.6±0.0	76.1±0.8	11.3±0.5
quickdraw	**83.6±0.6**	2.9±0.2	83.4±0.6	83.3±0.6	1.3±0.0	83.2±0.6	1.7±0.0	83.3±0.6	1.7±0.0	83.4±0.6	12.0±0.4
fungi	69.5±1.1	3.2±0.3	69.4±1.1	69.1±1.1	1.3±0.0	69.4±1.1	1.7±0.0	69.4±1.1	1.7±0.0	**69.7±1.1**	12.3±0.5
vgg_flower	91.8±0.7	5.6±0.5	91.9±0.7	**92.1±0.7**	1.4±0.0	91.9±0.7	1.8±0.0	91.9±0.7	1.8±0.0	92.0±0.7	12.1±0.6
WG avg	**79.83**	4.99	79.74	79.63	1.33	79.60	1.73	79.63	1.70	79.80	12.95
traffic_sign	**85.7±0.9**	2.9±0.3	84.9±1.0	84.6±1.0	1.5±0.0	84.9±1.0	2.1±0.0	84.8±1.0	2.1±0.0	84.9±1.0	8.3±0.4
mscoco	**54.5±1.0**	3.4±0.2	54.1±1.0	51.3±1.0	1.6±0.0	51.6±1.1	2.3±0.0	51.4±1.1	2.3±0.0	53.8±1.0	12.1±0.4
mnist	97.0±0.5	6.4±0.6	**97.1±0.5**	97.0±0.5	1.2±0.0	96.9±0.5	1.9±0.0	96.9±0.5	1.9±0.0	**97.1±0.5**	9.8±0.6
cifar10	**78.3±0.9**	2.7±0.2	78.1±0.9	75.5±0.9	1.3±0.0	76.9±0.9	2.0±0.0	76.8±0.9	2.0±0.0	77.5±0.9	9.4±0.4
cifar100	**70.6±1.1**	3.5±0.3	70.4±1.1	68.4±1.1	1.5±0.0	68.6±1.1	2.2±0.0	68.6±1.1	2.2±0.0	69.9±1.1	12.0±0.5
CropDisease	88.0±0.7	4.6±0.4	**88.2±0.7**	87.8±0.7	1.6±0.0	87.6±0.7	2.3±0.0	87.5±0.7	2.3±0.0	88.1±0.7	13.9±0.5
EuroSAT	**89.3±0.6**	2.5±0.2	88.8±0.6	88.0±0.6	1.2±0.0	88.6±0.6	2.0±0.0	88.5±0.6	1.9±0.0	88.8±0.6	9.4±0.4
ISIC	48.3±0.9	3.4±0.2	**49.5±0.9**	48.0±1.0	1.5±0.0	48.4±0.9	2.2±0.0	48.2±0.9	2.2±0.0	49.2±0.9	11.0±0.4
ChestX	27.0±0.6	3.2±0.3	**27.7±0.6**	27.0±0.6	1.5±0.0	27.2±0.6	2.3±0.0	27.1±0.6	2.3±0.0	27.5±0.6	12.4±0.4
Food101	**55.7±1.1**	2.9±0.2	55.2±1.1	52.0±1.1	1.6±0.0	53.3±1.1	2.1±0.0	53.5±1.1	2.0±0.0	55.2±1.1	11.6±0.3
SG avg	**69.44**	3.55	69.40	67.96	1.45	68.40	2.14	68.33	2.12	69.20	10.99

Appendix C

Patient BSS with STC

Table C.1 shows FES results with BSS pruning in supervised learning or with STC semi-supervised learning with 1000 unlabelled instances. BSS is evaluated with zero or maximum patience. The supervised results are compared to the semi-supervised STC results with the same patience value. The supervised methods are set as the reference method in Table C.1, and the STC counterparts are evaluated against them using the paired t-test. A $\bullet$ indicates significantly better supervised performance according to the paired t-test, and a $\circ$ indicates significantly better performance from STC. Table C.2 shows paired t-test results of all four methods compared to each other. STC improves performance of FES pruned by BSS with both zero and maximum patience, but patience does not change accuracy or snapshot reduction rate either in supervised learning or with STC.

Table C.1: BSS results either in supervised learning or with STC in semi-supervised learning, performed with either zero or maximum patience

	supervised greedy		STC greedy		supervised patient		STC patient	
Dataset	acc	ratio	acc	ratio	acc	ratio	acc	ratio
ilsvrc_2012	56.3±1.2	3.8±0.3	**56.7±1.1**∘	4.1±0.2	56.4±1.2	22.8±1.5	**56.7±1.1**∘	22.4±1.5
omniglot	93.1±0.7	12.3±0.8	93.7±0.7∘	13.1±0.8	93.3±0.7	31.0±2.1	**93.9±0.6**∘	32.1±2.0
aircraft	87.8±0.8	4.9±0.4	**88.0±0.7**∘	4.9±0.3	87.7±0.8	30.1±1.6	87.9±0.7	33.1±1.6
cu_birds	80.0±0.9	4.0±0.3	**80.3±0.8**∘	4.8±0.3	80.1±0.8	24.0±1.4	**80.3±0.8**∘	24.9±1.4
dtd	**76.5±0.8**	3.2±0.3	**76.5±0.8**	3.3±0.3	76.4±0.8	18.6±1.4	76.4±0.8	16.9±1.3
quickdraw	83.6±0.6	2.9±0.2	**83.9±0.6**∘	3.4±0.2	83.5±0.6	16.2±1.1	**83.9±0.6**∘	19.1±1.2
fungi	69.5±1.1	3.2±0.3	**70.9±1.1**∘	3.6±0.3	69.5±1.1	26.5±1.5	70.8±1.1∘	26.2±1.5
vgg_flower	91.8±0.7	5.6±0.5	92.3±0.7∘	5.9±0.5	91.9±0.7	19.9±1.4	**92.4±0.6**∘	19.9±1.4
WG avg	79.83	4.99	**80.29**	5.39	79.85	23.64	**80.29**	24.33
WG rank	2.64		**2.34**		2.65		2.36	
traffic_sign	85.7±0.9	2.9±0.3	**86.5±0.9**∘	2.8±0.3	85.4±1.0	30.6±1.9	86.3±0.9∘	25.9±1.7
mscoco	54.5±1.0	3.4±0.2	**55.5±1.0**∘	3.5±0.3	54.3±1.0	43.5±2.1	**55.5±1.0**∘	39.2±2.0
mnist	97.0±0.5	6.4±0.6	**97.3±0.5**∘	6.8±0.6	97.1±0.5	20.9±1.4	**97.3±0.5**∘	21.5±1.5
cifar10	78.3±0.9	2.7±0.2	**79.0±0.8**∘	2.7±0.2	78.2±0.9	37.8±2.2	78.7±0.8∘	37.2±2.2
cifar100	70.6±1.1	3.5±0.3	**71.2±1.0**∘	3.6±0.3	70.5±1.1	36.4±2.0	71.0±1.0∘	33.3±2.0
CropDisease	88.0±0.7	4.6±0.4	88.8±0.7∘	4.4±0.3	88.1±0.7	34.4±1.7	**89.0±0.7**∘	32.3±1.7
EuroSAT	89.3±0.6	2.5±0.2	**89.6±0.6**∘	2.5±0.2	89.3±0.6	20.7±1.6	89.5±0.6∘	21.6±1.6
ISIC	48.3±0.9	3.4±0.2	49.7±1.0∘	3.2±0.2	49.0±0.9	38.3±2.0	**50.5±0.9**∘	36.8±2.1
ChestX	27.0±0.6	3.2±0.3	27.2±0.6	3.0±0.2	27.5±0.6	58.5±2.2	**28.2±0.6**∘	60.4±2.2
Food101	55.7±1.1	2.9±0.2	**56.0±1.1**∘	3.1±0.2	55.7±1.1	20.1±1.7	55.9±1.1∘	22.2±1.7
SG avg	69.44	3.55	70.08	3.56	69.51	34.12	**70.19**	33.04
SG rank	2.76		2.24		2.77		**2.23**	

Table C.2: Statistically significant number of wins of column algorithm over row algorithm using paired t-test results of BSS in supervised settings or with STC in semi-supervised settings

	WG				SG			
	sup	STC	supP	STCP	sup	STC	supP	STCP
supervised $P = 0$	-	7	1	6	-	9	3	10
STC $P = 0$	0	-	0	1	0	-	0	3
supervised $P = 328$	1	7	-	6	3	9	-	10
STC $P = 328$	0	2	0	-	0	3	0	-

References

Bateni, P., Barber, J., van de Meent, J., and Wood, F. (2022). Enhancing few-shot image classification with unlabelled examples. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 1597–1606. IEEE.

Bateni, P., Goyal, R., Masrani, V., Wood, F., and Sigal, L. (2020). Improved few-shot visual classification. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA*, pages 14481–14490. Computer Vision Foundation / IEEE.

Bossard, L., Guillaumin, M., and Gool, L. V. (2014). Food-101 - mining discriminative components with random forests. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland*, volume 8694 of *Lecture Notes in Computer Science*, pages 446–461. Springer.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. E. (2020). A simple framework for contrastive learning of visual representations. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pages 1597–1607. PMLR.

Chen, W., Liu, Y., Kira, Z., Wang, Y. F., and Huang, J. (2019). A closer look at few-shot classification. In *7th International Conference on Learning Representations, New Orleans, LA, USA*. OpenReview.net.

Chen, Y., Liu, Z., Xu, H., Darrell, T., and Wang, X. (2021). Meta-baseline: Exploring simple meta-learning for few-shot learning. In *2021 IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada*, pages 9042–9051. IEEE.

Chowdhury, A., Chaudhari, D., Chaudhuri, S., and Jermaine, C. (2022). Meta-meta classification for one-shot learning. In *IEEE/CVF Winter Conference on Applications of Computer Vision, WACV 2022, Waikoloa, HI, USA, January 3-8, 2022*, pages 1628–1637. IEEE.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Miami, Florida, USA*, pages 248–255. IEEE Computer Society.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. (2021). An image is worth 16x16 words: Transformers for image recognition at scale. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

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 - 16th European Conference, Glasgow, UK, volume 12355 of *Lecture Notes in Computer Science*, pages 769–786. Springer.

Feuz, K. D. and Cook, D. J. (2015). Transfer learning across feature-rich heterogeneous feature spaces via feature-space remapping (FSR). *ACM Trans. Intell. Syst. Technol.*, 6(1):3:1–3:27.

Finn, C., Abbeel, P., and Levine, S. (2017). Model-agnostic meta-learning for fast adaptation of deep networks. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR.

Galeano, P., Joseph, E., and Lillo, R. E. (2015). The Mahalanobis distance for functional data with applications to classification. *Technometrics*, 57(2):281–291.

Guo, Y., Codella, N., Karlinsky, L., Codella, J. V., Smith, J. R., Saenko, K., Rosing, T., and Feris, R. (2020). A broader study of cross-domain few-shot learning. In *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK*, volume 12372 of *Lecture Notes in Computer Science*, pages 124–141. Springer.

Gütlein, M., Frank, E., Hall, M. A., and Karwath, A. (2009). Large-scale attribute selection using wrappers. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2009, part of the IEEE Symposium Series on Computational Intelligence 2009, Nashville, TN, USA, March 30, 2009 - April 2, 2009*, pages 332–339. IEEE.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 1026–1034. IEEE Computer Society.

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

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hu, S. X., Li, D., Stühmer, J., Kim, M., and Hospedales, T. M. (2022a). Pushing the limits of simple pipelines for few-shot learning: External data and fine-tuning make a difference. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR 2022, New Orleans, LA, USA, June 18-24, 2022*, pages 9058–9067. IEEE.

Hu, Y., Pateux, S., and Gripon, V. (2022b). Squeezing backbone feature distributions to the max for efficient few-shot learning. *Algorithms*, 15(5):147.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, Lille, France*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 448–456. JMLR.org.

Islam, A., Chen, C. R., Panda, R., Karlinsky, L., Feris, R., and Radke, R. J. (2021). Dynamic distillation network for cross-domain few-shot recognition with unlabeled data. In Ranzato, M., Beygelzimer, A., Dauphin, Y. N., Liang, P., and Vaughan, J. W., editors, *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, pages 3584–3595.

Kohavi, R. and John, G. H. (1997). Wrappers for feature subset selection. *Artif. Intell.*, 97(1-2):273–324.

Laine, S. and Aila, T. (2017). Temporal ensembling for semi-supervised learning. In *5th International Conference on Learning Representations, ICLR*

2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings. OpenReview.net.

Li, W., Liu, X., and Bilen, H. (2021). Universal representation learning from multiple domains for few-shot classification. In *2021 IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada*, pages 9506–9515. IEEE.

Li, W., Liu, X., and Bilen, H. (2022a). Cross-domain few-shot learning with task-specific adapters. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition, New Orleans, LA, USA*, pages 7151–7160. IEEE.

Li, W.-h., Liu, X., and Bilen, H. (2022b). Universal representation learning and task-specific adaptation for few-shot learning. `https://github.com/VICO-UoE/URL`. Accessed: 2022-09-29.

Li, Y. and Zhang, J. (2021). Semi-supervised meta-learning for cross-domain few-shot intent classification. In *Proceedings of the 1st Workshop on Meta Learning and Its Applications to Natural Language Processing*, pages 67–75.

Liu, L., Hamilton, W. L., Long, G., Jiang, J., and Larochelle, H. (2021a). A universal representation transformer layer for few-shot image classification. In *9th International Conference on Learning Representations, Virtual Event, Austria*. OpenReview.net.

Liu, Y., Lee, J., Zhu, L., Chen, L., Shi, H., and Yang, Y. (2021b). A multi-mode modulator for multi-domain few-shot classification. In *2021 IEEE/CVF International Conference on Computer Vision, Montreal, QC, Canada*, pages 8433–8442. IEEE.

Mensink, T., Verbeek, J., Perronnin, F., and Csurka, G. (2013). Distance-based image classification: Generalizing to new classes at near-zero cost. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(11):2624–2637.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Köpf, A., Yang, E. Z., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32, Vancouver, BC, Canada*, pages 8024–8035.

Perez, E., Strub, F., de Vries, H., Dumoulin, V., and Courville, A. C. (2018). FiLM: Visual reasoning with a general conditioning layer. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, the 30th innovative Applications of Artificial Intelligence, and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence, New Orleans, Louisiana, USA*, pages 3942–3951. AAAI Press.

Ravi, S. and Larochelle, H. (2017). Optimization as a model for few-shot learning. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net.

Ren, M., Triantafillou, E., Ravi, S., Snell, J., Swersky, K., Tenenbaum, J. B., Larochelle, H., and Zemel, R. S. (2018). Meta-learning for semi-supervised few-shot classification. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net.

Requeima, J., Gordon, J., Bronskill, J., Nowozin, S., and Turner, R. E. (2019). Fast and flexible multi-task classification using conditional neural adaptive processes. In *Advances in Neural Information Processing Systems 32, Vancouver, BC, Canada*, pages 7957–7968.

Rosenberg, C., Hebert, M., and Schneiderman, H. (2005). Semi-supervised self-training of object detection models. In *7th IEEE Workshop on Applications of Computer Vision / IEEE Workshop on Motion and Video Computing*

(WACV/MOTION 2005), 5-7 January 2005, Breckenridge, CO, USA, pages 29–36. IEEE Computer Society.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. S., Berg, A. C., and Fei-Fei, L. (2015). Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252.

Singh, A. and Rad, H. J. (2023). Transductive decoupled variational inference for few-shot classification. *Trans. Mach. Learn. Res.*, 2023.

Snell, J., Swersky, K., and Zemel, R. S. (2017). Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30, Long Beach, CA, USA*, pages 4077–4087.

Tan, M. and Le, Q. V. (2021). Efficientnetv2: Smaller models and faster training. In *Proceedings of the 38th International Conference on Machine Learning, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR.

Tarvainen, A. and Valpola, H. (2017). Mean teachers are better role models: Weight-averaged consistency targets improve semi-supervised deep learning results. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 1195–1204.

Tibshirani, R., Saunders, M., Rosset, S., Zhu, J., and Knight, K. (2005). Sparsity and smoothness via the fused lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(1):91–108.

Triantafillou, E., Larochelle, H., Zemel, R. S., and Dumoulin, V. (2021). Learning a universal template for few-shot dataset generalization. In *Proceedings*

of the 38th International Conference on Machine Learning, Virtual Event, volume 139 of *Proceedings of Machine Learning Research*, pages 10424–10433. PMLR.

Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P., and Larochelle, H. (2020a). Meta-dataset. `https://github.com/google-research/meta-dataset`. Accessed: 2023-11-16.

Triantafillou, E., Zhu, T., Dumoulin, V., Lamblin, P., Evci, U., Xu, K., Goroshin, R., Gelada, C., Swersky, K., Manzagol, P., and Larochelle, H. (2020b). Meta-dataset: A dataset of datasets for learning to learn from few examples. In *8th International Conference on Learning Representations, Addis Ababa, Ethiopia*. OpenReview.net.

Ullah, I., Carrion, D., Escalera, S., Guyon, I. M., 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 *Thirty-sixth Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

van Engelen, J. E. and Hoos, H. H. (2020). A survey on semi-supervised learning. *Mach. Learn.*, 109(2):373–440.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems 30, Long Beach, CA, USA*, pages 5998–6008.

Vinyals, O., Blundell, C., Lillicrap, T., Kavukcuoglu, K., and Wierstra, D. (2016). Matching networks for one shot learning. In *Advances in Neural Information Processing Systems 29, Barcelona, Spain*, pages 3630–3638.

Wang, H., Frank, E., Pfahringer, B., and Holmes, G. (2023). Self-trained centroid classifiers for semi-supervised cross-domain few-shot learning. In

Chandar, S., Pascanu, R., Sedghi, H., and Precup, D., editors, *Proceedings of The 2nd Conference on Lifelong Learning Agents*, volume 232 of *Proceedings of Machine Learning Research*, pages 481–492. PMLR.

Wang, H., Frank, E., Pfahringer, B., Mayo, M., and Holmes, G. (2024). Feature extractor stacking for cross-domain few-shot learning. *Machine Learning*, 113(1):121–158.

Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5(2):241–259.

Xu, R., Xing, L., Shao, S., Zhao, L., Liu, B., Liu, W., and Zhou, Y. (2022). GCT: graph co-training for semi-supervised few-shot learning. *IEEE Trans. Circuits Syst. Video Technol.*, 32(12):8674–8687.

Yeo, T., Kar, O. F., and Zamir, A. (2021). Robustness via cross-domain ensembles. In *2021 IEEE/CVF International Conference on Computer Vision, ICCV 2021, Montreal, QC, Canada, October 10-17, 2021*, pages 12169–12179. IEEE.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Póczos, B., Salakhutdinov, R., and Smola, A. J. (2017). Deep sets. In *Advances in Neural Information Processing Systems 30, Long Beach, CA, USA*, pages 3391–3401.