Data-Efficient Machine Learning with Applications to Cardiology

by

Aniruddh Raghu

B.A., University of Cambridge (2018)
M.Eng., University of Cambridge (2018)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
at the
MASSACHUSETTS INSTITUTE OF TECHNOLOGY
February 2024

Authored by:	Aniruddh Raghu Department of Electrical Engineering and Computer Science September 29, 2023
Certified by:	John V. Guttag Dugald C. Jackson Professor of Electrical Engineering and Computer Science Thesis Supervisor
Certified by:	Collin M. Stultz Nina T. and Robert H. Rubin Professor in Medical Engineering and Science and Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

Data-Efficient Machine Learning with Applications to Cardiology

by

Aniruddh Raghu

Submitted to the Department of Electrical Engineering and Computer Science
on September 29, 2023, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

Deep learning models have demonstrated impressive capabilities in many settings including computer vision, natural language generation, and speech processing. However, an important shortcoming of these models is that they often need to be trained on large datasets in order to be most effective. In domains such as medicine, large datasets are not always available, and thus there is a need for data-efficient models that perform well even in limited data regimes. In this thesis, motivated by this need, we present four contributions to data-efficient machine learning: (1) analyzing and improving few-shot learning, where we study a popular few-shot learning algorithm (Model Agnostic Meta-Learning) and provide insights as to why it is effective, proposing a simplified version that offers substantial computational benefits; (2) improving supervised learning on small clinical datasets of electrocardiograms (ECGs), where we develop a new data augmentation strategy for ECGs that helps boost performance on a range of predictive problems; (3) improving pre-training through the use of nested optimization, introducing an efficient gradient based algorithm to jointly optimize model parameters and pre-training algorithm design choices; and (4) developing a new self-supervised learning pipeline for complex clinical time series, where the design of the pipeline is driven by the multimodal, multi-dimensional nature of real-world clinical time series data. Unifying several of these contributions is the application area of cardiovascular medicine, a setting in which machine learning has the potential to improve patient care and outcomes.

Thesis Supervisor: John V. Guttag
Title: Dugald C. Jackson Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Collin M. Stultz
Title: Nina T. and Robert H. Rubin Professor in Medical Engineering and Science and Electrical Engineering and Computer Science

Acknowledgments

This thesis would not have been possible without the support and encouragement of many individuals.

First, I would like to thank my co-advisors, Collin Stultz and John Guttag. I first got to know Collin and John near the end of the first year of my PhD, when I spoke to them about changing research directions and collaborating with them. They were both extremely understanding of my situation, and went out of their way to make me feel welcome in their labs. Since we started working together in my second year, I have learned a lot from them, particularly in terms of thinking carefully about clinical applications and picking problems to work on that could lead to real clinical impact. I am grateful to them for actively encouraging me to conduct research at the intersection of new machine learning methods and clinical applications. Above all, they are kind and supportive people who go above and beyond for their students, and actively foster a friendly lab environment. I am fortunate to have had them both as my PhD advisors.

I would also like to thank my third thesis committee member, Phillip Isola, for making the time to be on my committee and for the helpful conversations about the work. Meetings with Phillip were valuable in many ways, and one aspect that will stay with me is how quickly he was able to distill the key contributions from a given research project. These insights gave me an improved understanding of the important takeaways from my own research.

Next, I would like to thank the many research collaborators and mentors I have had over the years. First, thank you to Marzyeh Ghassemi and Pete Szolovits, whom I worked with at MIT as an undergraduate researcher. They were my first research mentors, and were instrumental in getting me interested in this area of ML for clinical applications. I certainly would not have set off on this journey without their guidance in those early years.

I have been fortunate to have had the opportunity to work with various research groups during my PhD. In my first year, I worked with Dina Katabi's group, where I

learned various skills that have helped me over my research journey – a special thank you to Mingmin Zhao, Yingcheng Liu, Kreshnik Hoti, Hao Wang, and Abbas Zeitoun.

From my second year onwards, I've been fortunate to have been a member of two amazing research labs – the Computational Cardiovascular Research group (CCRG), and the Clinical and Applied Machine Learning group (CAML). Both labs are comprised of members who are not only skilled researchers but also kind and helpful people, which has made my PhD journey that much more enjoyable. Thank you to Adrian, Amy, Maggie, Davis, Harini, Divya, Katie L, Jose, Emily, Marianne, Hallee, Katie M, Marianne, Victor, Andrew, Helen, and Sheila from CAML, and to Paul, Wanzghi, Erik, Katherine, Ridwan, Daphne, Teya, Hyewon, Payal, Charles, Daniel, Mihir, and Megumi from CCRG. I want to give a special mention to the direct collaborators I've had in both groups over the years: Adrian Dalca, Katherine Young, Nate Diamant, Erik Reinertsen, Divya Shanmugam, Daphne Schlesinger, Payal Chandak, and Ridwan Alam. Collaborating with you all has helped me grow significantly as a researcher.

There are also many individuals outside these groups that have been wonderful collaborators and mentors in my PhD journey. This includes my sister Maithra Raghu, Geoffrey Hinton and Simon Kornblith at Google Research in Toronto, Matthew McDermott from Peter Szolovits' group at MIT, Vivek Natarajan and the team at Google Health AI, David Duvenaud and Jonathan Lorraine at the University of Toronto, Eugene Pomerantsev and Paige Stockwell at Massachusetts General Hospital, Tom Pollard and Ben Moody at the MIT Laboratory for Computational Physiology – thank you all! In particular, I am very fortunate to have worked with Geoff, David, Maithra, and Simon while interning during my second year summer – the work from that project helped directly inform two of the central contributions of this thesis.

Next, an enormous thank you to the many friends in Boston who have made this time at MIT that much more enjoyable: Yamin, Luke, Arjun, Sohil, Vibhaa, Nalini, Sarath, Abhin, Kevin, Stella, Vivek, Rohan and Sushruth. I have been fortunate to have had some incredible adventures with you all during these PhD years, including many hiking trips in New Hampshire and Utah, beautiful long walks and runs around

greater Boston, comically bad movie nights, and enjoying Boston's best ice cream flavour at New City Microcreamery. A mention also for the many friends I've made on the MIT Cricket team, MIT Swara, and MIT HSC – it's been great to keep up with playing cricket and cultural activities in my time here. A special thank you also to David, my roommate of five years.

Finally, an enormous thank you to my family, without whom this journey would have not been possible. To my parents, my sister Maithra, my brother-in-law Arun, my wife Kavya, and her family – thank you for everything.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In recent years, high-capacity neural network models have demonstrated impressive capabilities in a diverse range of domains. This includes applications to image recognition and generation [103, 30, 158], to natural language processing and generation [141, 46, 25], and to speech recognition and generation [140, 209, 6].

Despite being effective for many problems, neural network models still suffer from various shortcomings that preclude their widespread use in many domains. One important challenge is that these models often need to be trained with large datasets to be sufficiently performant, and in many domains, sufficiently large datasets are not available.

The challenge of *data scarcity* — having only limited data with which to build a machine learning (ML) model —- is particularly noticeable in ML applications in medicine. For example:

1. **Patients with a rare combination of phenotypes.** Suppose we wish to predict an outcome such as 1-year hospital readmission risk, for a patient with a rare combination of phenotypes, e.g.: a 30 year old Asian patient with cardiovascular disease, chronic kidney disease and a history of smoking. We might have few other patients in our dataset that resemble this individual.

2. **Predicting labels that are hard to measure.** Suppose we wish to infer a physiological quantity such as mean Pulmonary Capillary Wedge Pressure

(mPCWP) from an electrocardiogram (ECG). To build a model for this problem, we require a dataset of (ECG, mPCWP) pairs. However, ground-truth labels for mPCWP can only be obtained accurately through an invasive procedure. As a result, we only have small labelled datasets available for model development.

Since large datasets are not available for many problems, there is an important need for data-efficient ML models that perform effectively given only limited (labeled) training data. This need directly motivates the work in this thesis, where we present four contributions to data-efficient machine learning:

Understanding and Improving Few-Shot Learning (FSL) Algorithms (Chapter 3). FSL is a popularly studied data-efficient learning paradigm. In FSL, at training time, we are given many related *tasks* (e.g., multiple single N-way classification problem), and each task typically has limited labelled data (e.g., 1 labelled example for each of the N classes). The goal in FSL is to use these training tasks to learn a predictive model that then generalizes effectively to new data-scarce tasks at testing time.

One highly effective FSL algorithm is *Model Agnostic Meta-Learning (MAML)* [55]. MAML incorporates a nested optimization approach to FSL, and has been a widely used strategy, generating significant follow-on work [130, 57, 159, 83, 56, 66, 184]. Despite its successes, there remain fundamental open questions on MAML: why is it performant, and what exactly does the nested optimization structure of the algorithm achieve?

We address these questions, using tools from representation analysis of neural networks. Through a systematic empirical investigation, we discover that the effectiveness of MAML is mostly due to a paradigm we call feature reuse. Based on this observation, we formulate a simplified version of the MAML algorithm, demonstrating that it performs almost identically to the original MAML algorithm on benchmark few-shot classification and reinforcement learning tasks. Importantly, our simplified algorithm offers significant computational benefits over MAML.

Data Augmentation for Supervised Learning on Electrocardiograms (Chapter 4). Data augmentation during training [74, 197, 168, 88, 43, 44] is a useful strategy to improve the predictive performance of supervised learning models in data-scarce regimes – effectively increasing the size of the training dataset. However, existing data augmentation strategies are not well-suited to electrocardiogram (ECG) data, an important data modality used by clinicians to diagnose and monitor various cardiovascular conditions [160, 54, 24]. The main challenge with applying data augmentations to ECGs is that no single augmentation strategy is suitable for all ECG predictive tasks – the effect of augmentations on model performance is highly task-dependent.

This chapter outlines a novel parametric data augmentation strategy for ECGs, *TaskAug*, which addresses limitations of current ECG data augmentation approaches by defining a flexible augmentation policy that is learned on a per-task basis. We present an algorithm to efficiently optimize the parameters of TaskAug, leveraging recent work in nested optimization and implicit differentiation [113]. In an empirical study (across three datasets and eight predictive tasks) of TaskAug and other augmentation strategies, we find that TaskAug is competitive with or improves on other methods for all problems we examined.

Nested Optimization for Improved Pre-Training. Pre-training (PT) followed by fine-tuning (FT) is a popular and effective paradigm for training neural network models in data-scarce regimes [50, 167, 60, 82, 141, 46, 102, 207, 100, 84]. It is typically used when we want to build a predictive model for a task with limited labeled data (the fine-tuning task), but have access to a large, related dataset (the pre-training dataset). A model is learned in two phases: the model parameters are first optimized using the large pre-training dataset, and then refined from that initialization using the fine-tuning dataset.

A challenge with this paradigm is that the PT algorithms include high-dimensional, complex PT *meta-parameters*, e.g. parameterized data augmentation policies or task weights [30, 74, 201]. These meta-parameters can significantly affect the quality of

pre-trained models and the pre-trained model's suitability for fine-FT [30], and thus finding techniques to set their values optimally is important. However, optimizing PT meta-parameters is challenging, and existing methods to do so (e.g., random search, grid search, Bayesian optimization) are computationally expensive.

We outline a new, scalable gradient-based PT algorithm that jointly optimizes PT meta-parameters and model parameters over a single PT run. The core component of our algorithm is a gradient estimator that efficiently approximates PT meta-parameter gradients through the two stages of optimization (PT & FT), composing a constant-memory implicit differentiation approximation for the longer PT stage and exact backpropagation through training for the shorter FT stage.

In experiments, we demonstrate that using our algorithm for PT results in improved performance after FT when compared to various baselines in two settings: multitask PT on biological graph-structured data, and self-supervised PT on ECG data.

Self-Supervised Learning for Complex Clinical Time-Series (Chapter 5). Self-supervised learning (SSL) is a widely used approach for pre-training neural networks on unlabelled datasets prior to fine-tuning on downstream tasks with limited labelled data [30, 76, 14]. Significant prior work has applied self-supervised pre-training methods to clinical time-series data from the intensive care unit (ICU) [120, 195, 182, 204]. This is an ideal application for SSL since patients are closely monitored and generate a profusion of rich, unlabelled time-series data.

Although these works develop effective SSL strategies for clinical time series, they are designed for unimodal time series alone, such as a sequence of structured features (e.g., lab values and vitals signs) or an individual high-dimensional physiological signal (e.g., an electrocardiogram). These methods cannot be readily extended to model time series that exhibit multimodality — a common occurrence in real-world clinical datasets — where structured features *and* high-dimensional data are recorded at each timestep in the sequence.

We tackle this gap, and outline a novel SSL pipeline for these complex, multimodal

clinical time-series. Key to our method is a multiscale loss function, involving an SSL loss applied both at the level of the entire sequence and at the level of the individual high-dimensional data points in the sequence. This allows us to capture information at both scales. Experimental evaluation on two clinical datasets indicates that our approach outperforms various baselines.

In addition to the unifying theme of more data-efficient machine learning, there are two other themes that tie together subsets of these contributions:

1. **Nested Optimization.** Chapters 2, 3, and 4 all involve the use of nested optimization algorithms. In Chapter 2, we outline a simplified nested optimization algorithm for FSL, based on MAML (itself a nested optimization algorithm). In Chapter 3, our algorithm to learn TaskAug parameters leverages recent work in hyperparameter learning via nested optimization [113]. In Chapter 4, we propose a new nested optimization algorithm for learning PT meta-parameters.

2. **Applications to Cardiovascular Medicine.** A major motivating application area for the work in this thesis is cardiovascular medicine, since cardiovascular disease is a major cause of mortality and morbidity worldwide [21]. ML methods have shown promise in assisting with the diagnosis and monitoring of patients with cardiovascular disease [152].

   In Chapter 3, the data augmentation approaches are developed for various ECG predictive problems. ECGs are of great utility to clinicians in diagnosing and monitoring various cardiovascular conditions [160, 54, 24], and significant recent work has studied using neural networks to predict cardiac abnormalities, diseases, and outcomes directly from ECGs [70, 152, 63, 47, 98, 142]. In Chapter 4, we evaluate our algorithm on the task of detecting cardiac abnormalities from the ECG. In Chapter 5, we develop our SSL approach for multimodal time series from patients with cardiovascular disease, focusing on a task of clinical interest. In Appendix E, we outline another contribution that is focused on this application domain, in which we train a neural network model to assist with

the non-invasive monitoring of patients with heart failure, by detecting when a patient's mean Pulmonary Capillary Wedge Pressure is elevated. Since this contribution does not directly relate to data-efficient ML, we defer it to the appendix.

The rest of this thesis is structured as follows. Chapters 2-5 discuss each of these contributions in more detail, and Chapter 6 discusses the overall findings and potential future directions from this work.

Chapter 2

Understanding and Improving Few-Shot Learning Algorithms

2.1 Introduction

A central problem in data-efficient machine learning is *few-shot learning*, where new tasks are learned with a small number of labelled datapoints. A significant body of work has looked at tackling this challenge using meta-learning approaches [99, 189, 172, 55, 161, 155, 130]. Broadly speaking, these approaches define a family of tasks, some of which are used for training and others solely for evaluation. A meta-learning algorithm looks at learning properties that generalize across the different training tasks, and result in fast and efficient learning of the evaluation tasks.

One highly successful meta-learning algorithm is *Model Agnostic Meta-Learning (MAML)* [55]. At a high level, the MAML algorithm is comprised of two optimization loops. The outer loop (in the spirit of meta-learning) aims to find an effective *meta-initialization*, from which the inner loop can perform efficient *adaptation* – optimize parameters to solve new tasks with few labelled examples. This algorithm, with deep neural networks as the underlying model, has been highly influential, with significant follow on work, such as first order variants [130], probabilistic extensions [57], augmentation with generative modelling [159], and others [83, 56, 66, 184].

Despite the popularity of MAML, and the numerous followups and extensions,

there remains a fundamental open question on the basic algorithm. Does the meta-initialization learned by the outer loop result in *rapid learning* on unseen test tasks (efficient but significant changes in the representations) or is the success primarily due to *feature reuse* (with the meta-initialization already providing high quality representations)? In this chapter, we explore this question and its consequences. Our main contributions are:

- We perform layer freezing experiments and latent representational analysis of MAML, finding that feature reuse is the predominant reason for efficient learning.

- Based on these results, we propose the *ANIL (Almost No Inner Loop)* algorithm, a significant simplification to MAML that removes the inner loop updates for all but the head (final layer) of a neural network during training *and* inference. ANIL performs identically to MAML on standard benchmark few-shot classification and RL tasks, and offers computational benefits over MAML.

- We study the effect of the head of the network, finding that once training is complete, the head can be removed, and the representations can be used without adaptation to perform unseen tasks, which we call the *No Inner Loop (NIL)* algorithm.

- We study different training regimes, e.g. multiclass classification, multitask learning, etc, and find that the task specificity of MAML/ANIL at training appears to facilitate the learning of better features. We also find that multi-task training, a popular baseline with no task specificity, performs worse than random features in the few-shot learning setup.

- We discuss rapid learning and feature reuse in the context of other meta-learning approaches.

2.2 Related Work

MAML [55] is a highly popular meta-learning algorithm for few-shot learning, achieving competitive performance on benchmark few-shot learning problems [99, 189, 172, 161, 155, 130]. It is part of the family of optimization-based meta-learning algorithms, with other members of this family presenting variations around how to learn the weights of the task-specific classifier. For example [105, 64, 23, 104, 210] first learn functions to embed the support set and target examples of a few-shot learning task, before using the test support set to learn task specific weights to use on the embedded target examples. [71] proceeds similarly, using a Bayesian approach. The method of [13] explores a related approach, focusing on applications in text classification.

Of these optimization-based meta-learning algorithms, MAML has been especially influential, inspiring numerous direct extensions in recent literature [3, 57, 66, 159]. Most of these extensions critically rely on the core structure of the MAML algorithm, incorporating an outer loop (for meta-training), and an inner loop (for task-specific adaptation). There is little prior work analyzing why this central part of the MAML algorithm is practically successful. In this chapter, we focus on this foundational question, examining how and why MAML leads to effective few-shot learning. To do this, we utilize analytical tools such as Canonical Correlation Analysis (CCA) [150, 124] and Centered Kernel Alignment (CKA) [101] to study the neural network representations learned with the MAML algorithm, which also demonstrates MAML's ability to learn effective features for few-shot learning.

Insights from this analysis lead to a simplified algorithm, ANIL, which almost completely removes the inner optimization loop with no reduction in performance. Prior works [211, 91] have proposed algorithms where some parameters are only updated in the outer loop and others only in the inner loop. However, these works are motivated by different questions, such as improving MAML's performance or learning better representations, rather than conducting an analysis of the MAML algorithm and studying this core question of rapid learning vs feature reuse.

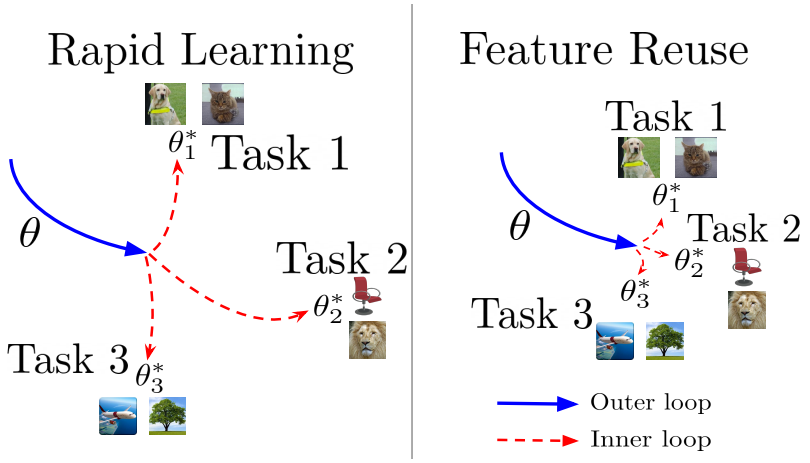


Figure 2-1: **Rapid learning and feature reuse paradigms.** In Rapid Learning, outer loop training leads to a parameter setting that is well-conditioned for fast learning, and inner loop updates result in task specialization. In Feature Reuse, the outer loop leads to parameter values corresponding to reusable features, from which the parameters do not move significantly in the inner loop.

2.3 MAML, Rapid Learning, and Feature Reuse

Our goal is to understand whether the MAML algorithm efficiently solves new tasks through predominantly *rapid learning* or *feature reuse*. In rapid learning, large representational and parameter changes occur during adaptation to each new task as a result of favorable weight conditioning from the meta-initialization. In feature reuse, the meta-initialization already contains highly useful features that can mostly be reused as is for new tasks, so little task-specific adaptation o the representation occurs. Figure 2-1 shows a schematic of these two paradigms.

We start off by overviewing the details of the MAML algorithm, and then we study the rapid learning vs feature reuse question via layer freezing experiments and analyzing latent representations of models trained with MAML. The results strongly support feature reuse as the predominant factor behind MAML's success. In Section 2.4, we explore the consequences of this, providing a significant simplification of MAML, the ANIL algorithm, and in Section 2.6, we outline the connections to meta-learning more broadly.

2.3.1 Overview of MAML

The MAML algorithm finds an *initialization* for a neural network so that new tasks can be learnt with very few examples (k examples from each class for k-shot learning) via two optimization loops:

- **Outer Loop:** Updates the initialization of the neural network parameters (often called the *meta-initialization*) to a setting that enables fast adaptation to new tasks.

- **Inner Loop:** Performs *adaptation*: takes the outer loop initialization, and, *separately for each task*, performs a few gradient updates over the k labelled examples (the *support set*) provided for adaptation.

More formally, we first define our base model to be a neural network with meta-initialization parameters θ; let this be represented by f_θ. We have have a distribution $\mathcal{D}$ over tasks, and draw a batch $\{T_1, ..., T_B\}$ of B distinct tasks from $\mathcal{D}$. For each task T_b, we have a *support set* of examples $\mathcal{S}_{T_b}$, which are used for inner loop updates, and a *target set* of examples $\mathcal{Z}_{T_b}$, which are used for outer loop updates. Let $\theta_i^{(b)}$ signify θ after i gradient updates for task T_b, and let $\theta_0^{(b)} = \theta$. In the inner loop, during each update, we compute

$$\theta_m^{(b)} = \theta_{m-1}^{(b)} - \alpha \nabla_{\theta_{m-1}^{(b)}} \mathcal{L}_{S_{T_b}}(f_{\theta_{m-1}^{(b)}(\theta)}) \tag{1}$$

for m fixed across all tasks, where $\mathcal{L}_{S_{T_b}}(f_{\theta_{m-1}^{(b)}(\theta)})$ is the loss on the support set of T_b after $m-1$ inner loop updates.

We then define the meta-loss as

$$\mathcal{L}_{meta}(\theta) = \sum_{b=1}^{B} \mathcal{L}_{\mathcal{Z}_{T_b}}(f_{\theta_m^{(b)}(\theta)})$$

where $\mathcal{L}_{Z_{T_b}}(f_{\theta_m^{(b)}(\theta)})$ is the loss on the target set of T_b after m inner loop updates, making clear the dependence of $f_{\theta_m^{(b)}}$ on θ. The outer optimization loop then updates

θ as

$$\theta = \theta - \eta \nabla_\theta \mathcal{L}_{meta}(\theta)$$

At test time, we draw unseen tasks $\{T_1^{(test)}, ..., T_n^{(test)}\}$ from the task distribution, and evaluate the loss and accuracy on $\mathcal{Z}_{T_i^{(test)}}$ after inner loop adaptation using $\mathcal{S}_{T_i^{(test)}}$ (e.g. loss is $\mathcal{L}_{\mathcal{Z}_{T_i^{(test)}}}\left(f_{\theta_m^{(i)}(\theta)}\right)$).

2.3.2 Rapid Learning or Feature Reuse?

We now turn our attention to the key question: *Is MAML's efficacy predominantly due to rapid learning or to feature reuse?* In investigating this question, there is an important distinction between the *head* (final layer) of the network and the earlier layers (the *body* of the network). In each few-shot learning task, there is a different alignment between the output neurons and classes. For instance, in task $\mathcal{T}_1$, the (wlog) five output neurons might correspond, in order, to the classes (dog, cat, frog, cupcake, phone), while for a different task, $\mathcal{T}_2$, they might correspond, in order, to (airplane, frog, boat, car, pumpkin). This means that the head must necessarily change for each task to learn the new alignment, and for the rapid learning vs feature reuse question, we are primarily interested in the behavior of the body of the network. We return to this in more detail in Section 2.5, where we present an algorithm (NIL) that does not use a head at test time.

To study rapid learning vs feature reuse in the network body, we perform two sets of experiments: (1) We evaluate few-shot learning performance when freezing parameters after MAML training, without test time inner loop adaptation; (2) We use representational similarity tools to directly analyze how much the network features and representations change through the inner loop. We use the MiniImageNet dataset, a popular standard benchmark for few-shot learning [1], and with the standard convolutional architecture in [55]. Results are averaged over three random seeds. Full implementation details are in Appendix A.2.

[1]MiniImageNet is a more diverse few-shot classification benchmark than the other dataset we examine in this chapter, Omniglot, and therefore we focus on MiniImageNet in these experiments.

Table 2.1: **Freezing successive layers (preventing inner loop adaptation) minimally affects accuracy, supporting feature reuse as the dominant mode of operation.** To test the amount of feature reuse happening in the inner loop adaptation, we test the accuracy of the model when we freeze (prevent inner loop adaptation) a contiguous block of layers at test time. We find that freezing even all four convolutional layers of the network (all layers except the network head) minimally affects accuracy. This supports the feature reuse hypothesis: layers do not have to change rapidly at adaptation time, since they already contain effective features from the meta-initialization.

Freeze layers	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
None	46.9 ± 0.2	63.1 ± 0.4
1	46.5 ± 0.3	63.0 ± 0.6
1,2	46.4 ± 0.4	62.6 ± 0.6
1,2,3	46.3 ± 0.4	61.2 ± 0.5
1,2,3,4	46.3 ± 0.4	61.0 ± 0.6

Freezing Layer Representations

To study the impact of the inner loop adaptation, we freeze a contiguous subset of layers of the network, during the inner loop at test time (after using the standard MAML algorithm, incorporating both optimization loops, for training). In particular, the frozen layers are not updated at all to the test time task, and must reuse the features learned by the meta-initialization that the outer loop converges to. We compare the few-shot learning accuracy when freezing to the accuracy when allowing inner loop adaptation.

Results are shown in Table 2.1. We observe that even when freezing all layers in the network body, performance changes quite minimally. This suggests that the meta-initialization has already learned good enough features that can be reused as is, without needing to perform any rapid learning for each test time task. One observation is that we a slight performance difference in the 5shot experiments. This is likely because the support set in the 5shot experiments is larger, allowing for more extensive adaptation of layers in the inner loop. However, the overall performance difference is small.

Representational Similarity Experiments

We next study how much the latent representations (the latent functions) learned by the neural network change during the inner loop adaptation phase. Following several recent works [150, 162, 124, 117, 151, 65, 16] we measure this by applying Canonical Correlation Analysis (CCA) to the latent representations of the network. CCA provides a way to the compare representations of two (latent) layers L_1, L_2 of a neural network, outputting a similarity score between 0 (not similar at all) and 1 (identical). For full details, see [150, 124]. In our analysis, we take L_1 to be a layer before the inner loop adaptation steps, and L_2 after the inner loop adaptation steps. We compute CCA similarity between L_1, L_2, averaging the similarity score across different random seeds of the model and different test time tasks. Full details are in Appendix A.2.2

The result is shown in Figure 2-2, left pane. Representations in the body of the network (the convolutional layers) are highly similar, with CCA similarity scores of > 0.9, indicating that the inner loop induces little to no functional change. By contrast, the head of the network, which does change significantly in the inner loop, has a CCA similarity of less than 0.5. To further validate this, we also compute CKA (Centered Kernel Alignment) [101] (Figure 2-2 right), another similarity metric for neural network representations, which illustrates the same pattern. These representational analysis results strongly support the feature reuse hypothesis, with further results in the Appendix, Sections A.2.3 and A.2.4 providing yet more evidence.

Feature Reuse Happens Early in Learning

Having observed that the inner loop does not significantly affect the learned representations with a fully trained model, we extend our analysis to see whether the inner loop affects representations and features earlier on in training. We take MAML models at 10000, 20000, and 30000 iterations into training, perform freezing experiments (as in Section 2.3.2) and representational similarity experiments (as in Section 2.3.2).

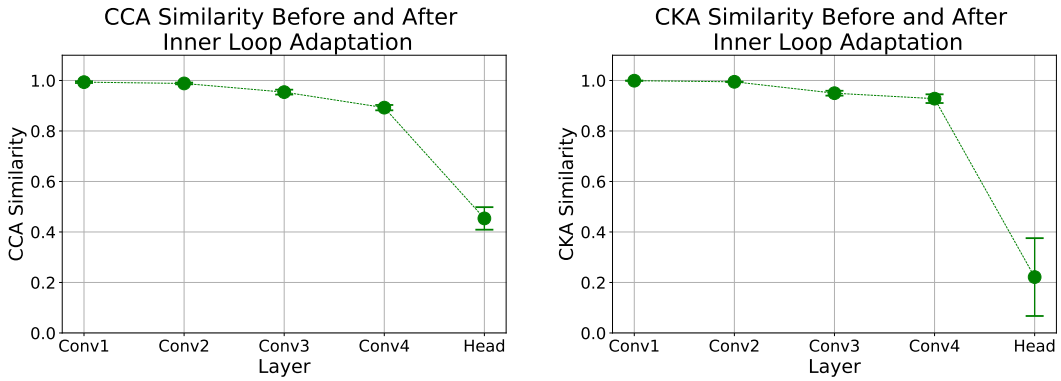


Figure 2-2: **High CCA/CKA similarity between representations before and after adaptation for all layers except the head.** We compute CCA/CKA similarity between the representation of a layer before the inner loop adaptation and after adaptation. We observe that for all layers except the head, the CCA/CKA similarity is almost 1, indicating perfect similarity. This suggests that these layers do not change much during adaptation, but mostly perform feature reuse. Note that there is a slight dip in similarity in the higher conv layers (e.g. conv3, conv4); this is likely because the slight representational differences in conv1, conv2 have a compounding effect on the representations of conv3, conv4. The head of the network must change significantly during adaptation, and this is reflected in the much lower CCA/CKA similarity.

Results in Figure 2-3 demonstrate that feature reuse dominates from early in training. The CCA similarity between activations pre and post inner loop update on MiniImageNet-5way-5shot are high for the body (just like Figure 2-2), and, similar to Table 2.1, test accuracy remains approximately the same when freezing contiguous subsets of layers. This indicates that even early on in training, significant feature reuse is taking place, with the inner loop having little effect on learned representations. Results for 1shot MiniImageNet are in Appendix A.2.5, and show similar trends.

2.4 The ANIL (Almost No Inner Loop) Algorithm

In the previous section we saw that for all layers except the head of the neural network, the meta-initialization learned by the outer loop of MAML results in effective learned parameters that can be reused as is on new tasks. Inner loop adaptation does not significantly change the representations of these layers, even from early on in training. This suggests a natural simplification of the MAML algorithm: the *ANIL (Almost*

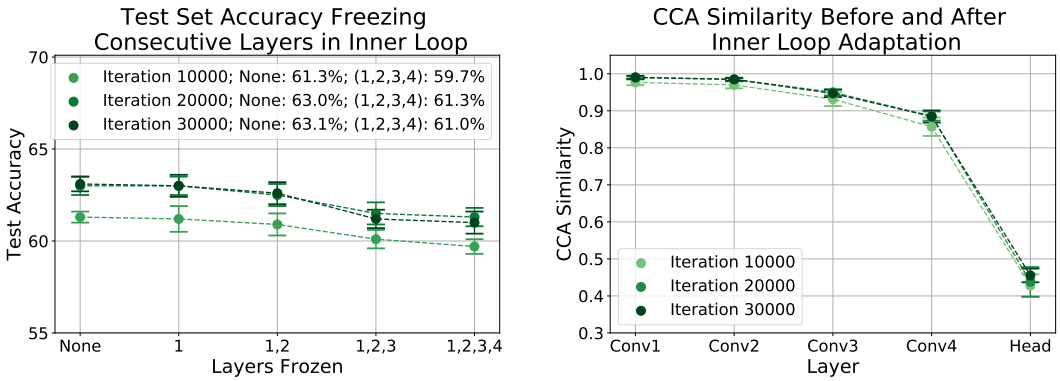


Figure 2-3: **Inner loop updates have little effect on learned representations from early on in learning.** Left pane: we freeze contiguous blocks of layers (no adaptation at test time), on MiniImageNet-5way-5shot and see almost identical performance. Right pane: representations of all layers except the head are highly similar pre/post adaptation – i.e. features are being reused. This is true from early (iteration 10000) in training.

No Inner Loop) algorithm.

In ANIL, during training **and** testing, we *remove* the inner loop updates for the network body, and apply inner loop adaptation *only to the head*. The head requires the inner loop to allow it to align to the different classes in each task. In Section 2.5.1 we consider another variant, the *NIL (No Inner Loop) algorithm*, that removes the head entirely at test time, and uses learned features and cosine similarity to perform effective classification, thus avoiding inner loop updates altogether.

For the ANIL algorithm, let $\theta = (\theta_1, ..., \theta_l)$ be the (meta-initialization) parameters for the l layers of the network. Following the notation of Section 2.3.1, let $\theta_m^{(b)}$ be the parameters after m inner gradient updates for task $\mathcal{T}_b$. In ANIL, we have that:

$$\theta_m^{(b)} = \left(\theta_1, \ldots, (\theta_l)_{m-1}^{(b)} - \alpha \nabla_{(\theta_l)_{m-1}^{(b)}} \mathcal{L}_{S_b}(f_{\theta_{m-1}^{(b)}})\right)$$

, i.e., only the final layer gets the inner loop updates. As before, we then define the meta-loss, and compute the outer loop gradient update. Note that this is distinct to the freezing experiments, where we only removed the inner loop at inference time. Figure 2-4 presents the difference between MAML and ANIL, and Appendix A.3.1 considers a simple example of the gradient update in ANIL, showing how the ANIL update differs from MAML.

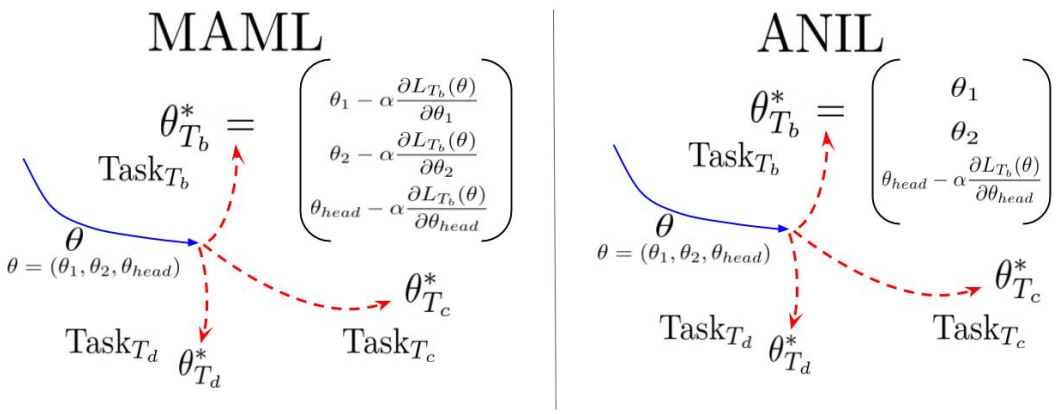


Figure 2-4: **Schematic of MAML and ANIL algorithms.** The difference between the MAML and ANIL algorithms: in MAML (left), the inner loop (task-specific) gradient updates are applied to all parameters θ, which are initialized with the meta-initialization from the outer loop. In ANIL (right), only the parameters corresponding to the network head θ_{head} are updated by the inner loop, during training **and** testing.

Results of ANIL on Standard Benchmarks: We evaluate ANIL on few-shot image classification and RL benchmarks, using the same model architectures as the original MAML authors, for both supervised learning and RL. Further implementation details are in Appendix A.3.4. The results in Table 2.2 (mean and standard deviation of performance over three random initializations) show that ANIL matches the performance of MAML on both few-shot classification (accuracy) and RL (average return, the higher the better), demonstrating that the inner loop adaptation of the body is not required for learning good features.

Computational benefit of ANIL: Since ANIL almost has no inner loop, it significantly speeds up both training and inference. Table 2.3 shows results from a comparison of the computation time for MAML, First Order MAML, and ANIL, during training and inference, with the TensorFlow implementation described previously, on both MiniImageNet domains. These results are average time for executing forward and backward passes during training (above) and a forward pass during inference (bottom), for a task batch size of 1, and a target set size of 1. Results are averaged over 2000 such batches. Speedup is calculated relative to MAML's execution time. Each batches' images were loaded into memory before running the TensorFlow

Table 2.2: **ANIL matches the performance of MAML on few-shot image classification (top) and reinforcement learning (bottom).** We evaluate MAML and ANIL on both few-shot image classification benchmarks (top table) and few-shot reinforcement learning (RL) benchmarks (bottom table). For image classification, we report mean and standard deviation of accuracy over three random initializations, and for RL, we report the mean and standard deviation of the return (a commonly used metric) over three random initializations. We find that MAML and ANIL have comparable performance on both classes of benchmarks.

Method	Omniglot-20way-1shot	Omniglot-20way-5shot	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML	93.7 ± 0.7	96.4 ± 0.1	46.9 ± 0.2	63.1 ± 0.4
ANIL	96.2 ± 0.5	98.0 ± 0.3	46.7 ± 0.4	61.5 ± 0.5

Method	HalfCheetah-Direction	HalfCheetah-Velocity	2D-Navigation
MAML	170.4 ± 21.0	-139.0 ± 18.9	-20.3 ± 3.2
ANIL	363.2 ± 14.8	-120.9 ± 6.3	-20.1 ± 2.3

computation graph, to ensure that data loading time was not captured in the timing. Experiments were run on a single NVIDIA Titan-Xp GPU.

During training, we see that ANIL is as fast as First Order MAML (which does not compute second order terms during training), and about 1.7x as fast as MAML. This leads to a significant overall training speedup, especially when coupled with the fact that the rate of learning for ANIL and MAML is very similar; see learning curves in Figure 2-5. Note that unlike First Order MAML, ANIL also performs comparably to MAML on benchmark tasks (on some tasks, First Order MAML performs worse [55]). During inference, ANIL achieves over a 4x speedup over MAML (and thus also 4x over First Order MAML, which is identical to MAML at inference time). Both training and inference speedups illustrate the significant computational benefit of ANIL over MAML.

MAML and ANIL Models Show Similar Behavior: MAML and ANIL perform almost equally well on few-shot learning benchmarks, illustrating that removing the inner loop during training does not hinder performance. To study the behavior of MAML and ANIL models further, we plot learning curves for both algorithms on MiniImageNet-5way-5shot, Figure 2-5. We see that loss and accuracy for both algorithms look similar throughout training. We also look at CCA and CKA scores of the

Table 2.3: **ANIL offers significant computational speedup over MAML, during both training and inference.** Table comparing execution times and speedups of MAML, First Order MAML, and ANIL during training (above) and inference (below) on MiniImageNet domains. Speedup is calculated relative to MAML's execution time.

	Training: 5way-1shot			Training: 5way-5shot		
	Mean (s)	Median (s)	Speedup	Mean (s)	Median (s)	Speedup
MAML	0.15	0.13	1	0.68	0.67	1
First Order MAML	0.089	0.083	1.69	0.40	0.39	1.7
ANIL	0.084	0.072	1.79	0.37	0.36	1.84

	Inference: 5way-1shot			Inference: 5way-5shot		
	Mean (s)	Median (s)	Speedup	Mean (s)	Median (s)	Speedup
MAML	0.083	0.078	1	0.37	0.36	1
ANIL	0.020	0.017	4.15	0.076	0.071	4.87

Table 2.4: **MAML and ANIL models learn comparable representations.** Comparing CCA/CKA similarity scores of the of MAML-ANIL representations (averaged over network body), and MAML-MAML and ANIL-ANIL similarity scores (across different random seeds) shows algorithmic differences between MAML/ANIL do not result in vastly different types of features learned.

Model Pair	CCA Similarity	CKA Similarity
MAML-MAML	0.51	0.83
ANIL-ANIL	0.51	0.86
ANIL-MAML	0.50	0.83

representations learned by both algorithms, Table 2.4. We observe that MAML-ANIL representations have the same average similarity scores as MAML-MAML and ANIL-ANIL representations, suggesting the algorithms learn comparable features. Further learning curves and representational similarity results are presented in Appendices A.3.2 and A.3.3.

2.5 Contributions of the Network Head and Body

So far, we have seen that MAML predominantly relies on feature reuse, with the network body (all layers except the last layer) already containing good features at meta-initialization. We also observe that such features can be learned without inner loop adaptation during training (ANIL algorithm). The head, however, requires inner

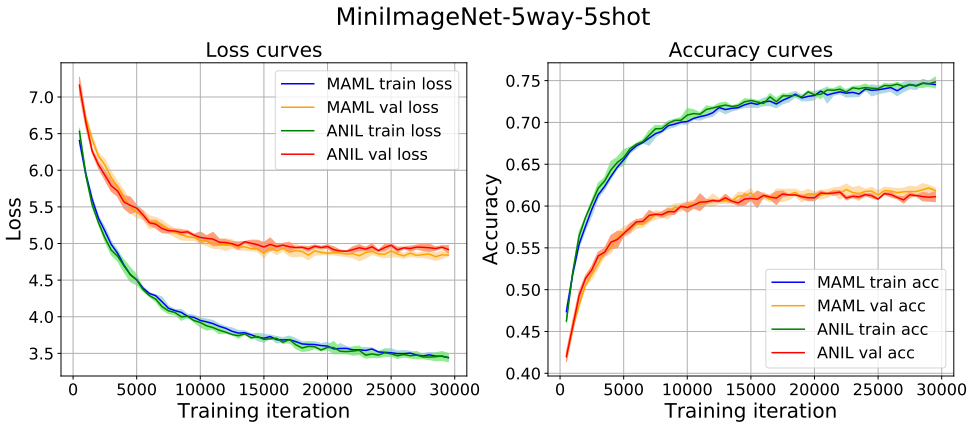


Figure 2-5: **MAML and ANIL learn at similar rates**. Loss and accuracy curves for MAML and ANIL on MiniImageNet-5way-5shot, illustrating how MAML and ANIL behave similarly through the training process.

loop adaptation to enable task specificity.

In this section, we explore the contributions of the network head and body. We first ask: *How important is the head at test time, when good features have already been learned?* Motivating this question is that the features in the body of the network needed no adaptation at inference time, so perhaps they are themselves sufficient to perform classification, with no head. In Section 2.5.1, we find that test time performance is largely determined by the quality of these representations, and we can use similarity of the frozen meta-initialization representations to perform unseen tasks, *removing the head entirely*. We call this the NIL (No Inner Loop) algorithm.

Given this result, we next study how useful the head is at training (in ensuring the network body learns good features). We look at multiple different training regimes (some without the head) for the network body, and evaluate the quality of the representations. We find that MAML/ANIL result in the best representations, demonstrating the importance of the head during training for feature learning in a few-shot learning setup.

Table 2.5: **NIL algorithm performs as well as MAML and ANIL on few-shot image classification.** Accuracy of MAML, ANIL, and NIL on few-shot image classification benchmarks. We see that with no test-time inner loop, and just learned features, NIL performs comparably to MAML and ANIL, indicating the strength of the learned features, and the relative lack of importance of the head at test time.

Method	Omniglot-20way-1shot	Omniglot-20way-5shot	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML	93.7 ± 0.7	96.4 ± 0.1	46.9 ± 0.2	63.1 ± 0.4
ANIL	96.2 ± 0.5	98.0 ± 0.3	46.7 ± 0.4	61.5 ± 0.5
NIL	96.7 ± 0.3	98.0 ± 0.04	48.0 ± 0.7	62.2 ± 0.5

2.5.1 The Head at Test Time and the NIL (No Inner Loop) Algorithm

We study how important the head and task specific alignment are when good features have *already* been learned (through training) by the meta-initialization. At test time, we find that the representations can be used directly, with *no* adaptation, which leads to the *No Inner Loop (NIL) algorithm*:

1. Train a few-shot learning model with ANIL/MAML algorithm as standard. We use ANIL training.

2. At test time, remove the head of the trained model. For each task, first pass the k labelled examples (support set) through the body of the network, to get their penultimate layer representations. Then, for a test example, compute cosine similarities between its penultimate layer representation and those of the support set, using these similarities to weight the support set labels, as in [189].

The results for the NIL algorithm, following ANIL training, on few-shot classification benchmarks are given in Table 2.5. Despite having no network head and no task specific adaptation, NIL performs comparably to MAML and ANIL.

2.5.2 Training Regimes for the Network Body

The NIL algorithm and results of Section 2.5.1 lead to the question of how important task alignment and the head are during training to ensure good features. Here, we study this question by examining the quality of features arising from different training

Table 2.6: **MAML/ANIL training leads to superior features learned, supporting importance of head at training.** We compare the mean/standard deviation of accuracy for the different methods on the benchmark tasks. Training with MAML/ANIL leads to higher accuracy when compared to other methods which do not have task specific heads, supporting the importance of the head at training.

Method	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML training-NIL head	48.4 ± 0.3	61.5 ± 0.8
ANIL training-NIL head	48.0 ± 0.7	62.2 ± 0.5
Multiclass training-NIL head	39.7 ± 0.3	54.4 ± 0.5
Multitask training-NIL head	26.5 ± 1.1	34.2 ± 3.5
Random features-NIL head	32.9 ± 0.6	43.2 ± 0.5
NIL training-NIL head	38.3 ± 0.6	43.0 ± 0.2

regimes for the body. We look at (i) MAML and ANIL training; (ii) *multiclass classification*, where all of the training data and classes (from which training tasks are drawn) are used to perform standard classification; (iii) *multitask training*, a standard baseline, where no inner loop or task specific head is used, but the network is trained on all the tasks at the same time; (iv) *random features*, where the network is not trained at all, and features are frozen after random initialization; (v) NIL at training time, where there is no head and cosine distance on the representations is used to get the label.

After training, we apply the NIL algorithm to evaluate test performance, and quality of features learned at training. The results are shown in Table 2.6. MAML and ANIL training performs best. Multitask training, which has no task specific head, performs the worst, even worse than random features (adding evidence for the need for task specificity at training to facilitate feature learning.) Using NIL during training performs worse than MAML/ANIL. These results suggest that the head is important at training to learn good features in the network body.

In Appendix A.4.1, we study test time performance variations from using a MAML/ANIL head instead of NIL, finding (as suggested by Section 2.5.1) very little performance difference. Additional results on similarity between the representations of different training regimes is given in Appendix A.4.2.

2.6 Feature Reuse in Other Meta-Learning Algorithms

Up till now, we have examined the MAML algorithm, and demonstrated empirically that the algorithm's success is primarily due to feature reuse, rather than rapid learning. We now discuss rapid learning vs feature reuse more broadly in meta-learning. By combining our results with an analysis of evidence reported in prior work, we find support for many meta-learning algorithms succeeding via feature reuse.

2.6.1 Optimization and Model Based Meta-Learning

MAML falls within the broader class of *optimization based* meta-learning algorithms, which at inference time, directly optimize model parameters for a new task using the support set. MAML has inspired many other optimization-based algorithms that utilize the same two-loop structure [105, 159, 57]. Our analysis so far has thus yielded insights into the importance of feature reuse vs rapid learning question for this class of algorithms. Another broad class of meta-learning consists of *model based* algorithms, which also have notions of rapid learning and feature reuse.

In the model-based setting, the meta-learning model's parameters are *not* directly optimized for the specific task on the support set. Instead, the model conditions its output on some representation of the task definition. One way to achieve this conditioning is to *jointly encode* the entire support set in the model's latent representation [189, 176], enabling it to adapt to the characteristics of each task. This constitutes rapid learning for model based meta-learning algorithms.

An alternative to joint encoding would be to encode each member of the support set independently, and apply a cosine similarity rule (as in [189]) to classify an unlabelled example. This mode of operation is purely feature reuse – it does not use information defining the task to directly influence the decision function.

If joint encoding gave significant test-time improvement over non-joint encoding, then this would suggest that rapid learning of the test-time task is taking place, as task specific information is being utilized to influence the model's decision function. However, on analyzing results in prior literature, this improvement appears to be

relatively small. In e.g. Matching Networks [189], using joint encoding one reaches 44.2% accuracy on MiniImageNet-5way-1shot, whereas with independent encoding one obtains 41.2%. More refined models suggest the gap is even smaller. For instance, in [32], many methods for one shot learning were re-implemented and studied, and baselines without joint encoding achieved 48.24% accuracy in MiniImageNet-5way-1shot, whilst other models using joint encoding such as Relation Net [176] achieve very similar accuracy of 49.31% (they also report MAML, at 46.47%). As a result, we believe that the dominant mode of "feature reuse" rather than "rapid learning" is what has currently dominated both MAML-styled optimization based meta-learning *and* model based meta-learning.

2.7 Scope and Limitations

Scope of study. Our study in this chapter focused on the Model Agnostic Meta-Learning (MAML) algorithm and mostly examined its applications to few-shot image classification, considering two standard datasets: Omniglot and MiniImageNet. We did not extend our empirical study to the many variations of MAML that have since been proposed – this could be a valuable direction of future work. Studying whether our findings hold for more diverse few-shot learning datasets from e.g., MetaDataset [184], and in other domains, such as few-shot medical image segmentation [27], are also open questions. In few-shot learning problems that have substantial distribution shift between the training and testing tasks, e.g., some of the formulations in MetaDataset, our approach of freezing the entire network body may not work well and adaptation may be necessary to obtain good performance.

Few-shot learning setup. In this work, we focused on the standard setup of 'small-K' few-shot learning, where the number of examples per class in the support set is small (e.g., 1-5). Given more data in the support set, it is likely that our main finding — inner-loop adaptation of MAML is not necessary for the network body — would not hold. A large support set contains more predictive signal to usefully

optimize the body parameters in the inner loop, and we would expect performance to improve with adaptation (as is seen in fine-tuning in transfer learning). However, if this is the intended use-case, we might not want to use meta-learning based few-shot learning at all, and instead (partially) fine-tune a large pre-trained model [178].

2.8 Conclusion

In this chapter, we focused on a particular paradigm of data-efficient machine learning: few-shot learning. We studied the question of why a highly popular few-shot learning algorithm — Model Agnostic Meta-Learning (MAML) — is effective. Specifically, we investigated whether MAML predominantly relies on rapid learning (large parameter changes at adaptation time) or feature reuse (re-use the learned representations at adaptation time). Through a series of experiments, we found that *feature reuse* is the dominant component in MAML's efficacy on benchmark datasets. This insight led to the ANIL (Almost No Inner Loop) algorithm, a simplification of MAML that has almost identical performance on standard image classification and reinforcement learning benchmarks, and provides computational benefits. We further study the importance of the head (final layer) of a neural network trained with MAML, discovering that the body (lower layers) of a network is sufficient for few-shot classification at test time, allowing us to remove the network head for testing (NIL algorithm) and still match performance. We connected these results to the broader literature in meta-learning, identifying feature reuse to be a common mode of operation for other meta-learning algorithms also.

Follow-up work. Since the work in this chapter was published [144], there have been several follow-up works investigating this question of rapid learning vs feature reuse in different few-shot learning problems, ranging from theoretical studies supporting our findings [39], to empirical studies that design new algorithms to encourage rapid learning in settings with greater distribution shift [132]. Few-shot learning more generally has been a topic of interest in recent work in computer vision [97, 2, 27]

and natural language understanding [25] – these recent works typically rely on some notion of feature reuse, with general-purpose representations being learned in a broad pre-training phase.

Chapter 3

Data Augmentation for Supervised Learning on Electrocardiograms

3.1 Introduction

In Chapter 2, we studied one paradigm of data-efficient machine learning: few-shot learning, in which we are presented many related tasks, each with limited data, and wish to learn an effective, generalizable predictive model. In this chapter, we turn our attention to a different data-efficient machine learning setting: supervised learning problems in which we have access to a single, small labelled dataset, and we wish to learn an effective predictive model.

The focus of this chapter is data-scarce supervised learning problems on electrocardiogram data. Electrocardiography is used widely in medicine as a non-invasive and relatively inexpensive method of measuring the electrical activity in an individual's heart. The electrocardiogram (ECG) is the output of electrocardiography, and is of great utility to clinicians in diagnosing and monitoring various cardiovascular conditions [160, 54, 24].

In recent years, there has been significant interest in automatically predicting cardiac abnormalities, diseases, and outcomes directly from ECGs using neural network models [70, 152, 63, 47, 98, 142]. Although these models demonstrate impressive results, training them often requires large labelled datasets with paired ECGs and

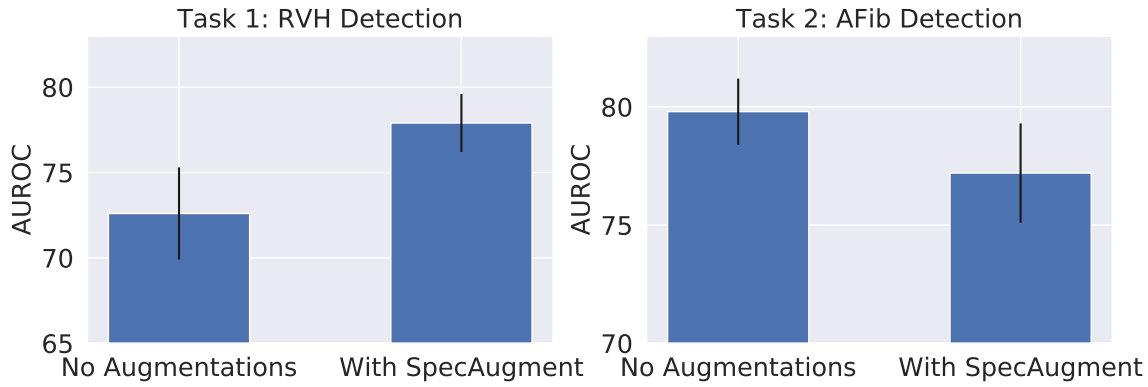


Figure 3-1: **The effect of data augmentation on ECG prediction tasks is task-dependent.** We examine the mean/standard error of AUROC over 5 runs when applying SpecAugment [135], a data augmentation method, to two different ECG prediction tasks. We observe performance improvement in one setting (left, Right Ventricular Hypertrophy), and performance reduction in another (right, Atrial Fibrillation).

labels. In some situations, it is challenging to construct such datasets. For example, consider inferring abnormal central hemodynamics (e.g., elevated mean Pulmonary Capillary Wedge Pressure) from the ECG; this is important when monitoring patients with heart failure or pulmonary hypertension [164, 147]. Accurate hemodynamics labels are only obtainable through specialized invasive studies [8, 79], and hence it is difficult to obtain large datasets with paired ECGs and hemodynamics variables.

Data augmentation [74, 197, 168, 88, 43, 44] during training has been demonstrated to be a useful strategy to improve the predictive performance of models in data-scarce regimes. However, there exists limited work studying data augmentation for ECGs. A key problem with applying standard data augmentations is that fine-grained information within ECGs, such as relative amplitudes of portions of beats, carry predictive signal: augmentations may worsen performance if such predictive signal is destroyed. Furthermore, the effectiveness of data augmentations with ECGs varies on a task-specific basis – applying the same augmentation for two different tasks could help performance in one case, and hurt performance in another (Figure 3-1).

In this chapter, we take steps towards addressing these issues. Our contributions are as follows:

- We propose *TaskAug*, a new task-dependent augmentation strategy. TaskAug

defines a flexible augmentation policy that is optimized on a per-task basis. We outline an efficient learning algorithm for this that leverages recent work in nested optimization and implicit differentiation. [113].

- We conduct an empirical study of TaskAug and other augmentation strategies on ECG predictive problems. We consider three datasets and eight different predictive tasks, which cover different classes of cardiac abnormalities.
- We analyze the results from our evaluation, finding that many augmentation strategies do not work well across all tasks. Given its task-specific nature, TaskAug is competitive with or improves on other methods for the problems we examined.
- We study the learned TaskAug policies, finding that they offer insights as to what augmentations are most appropriate for different tasks.
- We provide a summary of findings and best practices to assist future studies exploring data augmentation for ECG tasks.

3.2 Related Work

Data augmentation for time-series. Prior research on time-series data augmentation includes: (1) large-scale surveys exploring the impact of augmentation on various downstream modalities [88, 89, 197]; and (2) specific methods for particular modalities, including speech signals [135, 136], wearable device signals [186], and time series forecasting [11, 171]. There is relatively little work exploring how augmentation can impact performance for ECG-based prediction tasks, with prior studies mostly restricted to considering single tasks [73, 12]. In contrast, in this chapter, we begin by evaluating a set of data augmentation methods on many different ECG predictive tasks, studying when and why augmentations may help. In addition, the data augmentation strategy proposed in this chapter, *TaskAug*, can be readily adapted to new predictive tasks, unlike in existing works where the methods may be designed for a specific downstream task.

There is related work on using data augmentation for contrastive pre-training with ECGs [63, 98, 143, 121]. These works are complementary to the contributions in this chapter; we focus specifically on supervised learning (rather than contrastive pre-training), and we hypothesize that the proposed augmentation pipeline could be used in these prior methods for improved contrastive learning.

Designing and learning data augmentation policies. The structure of *TaskAug*, our proposed augmentation strategy, was inspired by related work on flexible data augmentation policies in computer vision [43, 44, 74]. We extend these ideas to ECG predictive tasks by (1) selecting appropriate transformations for ECG data, and (2) allowing for class-specific transformation strengths. Since such policies introduce many hyperparameters, we use a bi-level optimization algorithm to enable scalable policy learning [113, 146].

3.3 Problem Setup and Notation

We focus on supervised binary classification problems from ECG data. Let $x \in \mathbb{R}^{12\times T}$ refer to a 12-lead ECG of T samples and $y \in \{0, 1\}$ refer to a binary target. We let $\mathcal{D} = \{(x_n, y_n)\}_{n=1}^{N}$ refer to a dataset of N ECG-label pairs.

Let $f(x; \theta) \rightarrow \hat{y}$ be a neural network model with parameters θ that outputs a predicted label $\hat{y}$ given x as input. Network parameters are optimized to minimize the average binary cross entropy loss $\mathcal{L}_{BCE}$ on the training dataset $\mathcal{D}^{(\text{train})}$.

We restrict our study to single label binary classification problems in order to study the effect of data augmentation on a per-task basis. One can extend this to multilabel binary classification by letting y be a vector of several different binary labels and training the network to produce a vector of predictions.

Training with Data Augmentation. Let $A(x, y; \phi) \rightarrow \tilde{x}$ refer to a data augmentation function with hyperparameters ϕ that takes the input ECG x and its label y and outputs an augmented version $\tilde{x}$. Note that this formulation implicitly assumes that the augmentation is label preserving, since it does not change the label y. Where

relevant, the augmentation hyperparameters ϕ may control the strength/probability of applying an augmentation.

The process of training with data augmentation [1] amounts to:

1. Sample a data point and label pair from the training set: $(x, y) \sim \mathcal{D}^{(\text{train})}$.
2. Apply the augmentation $A : x \mapsto \tilde{x}$, to transform the original input x to an augmented version $\tilde{x}$.
3. Use the pair $(\tilde{x}, y)$ in training.

A comment on test-time augmentation. We focus in this chapter on augmentations applied during the training process. A related family of techniques consider *test-time augmentation* applied at inference time [166], but this is not within the scope of the current investigation.

3.4 Data Augmentation Methods

We now describe the data augmentation methods considered in our experiments. We also present our new, learnable data augmentation method that can be used to find task-specific augmentation policies, and an algorithm to optimize its parameters.

3.4.1 Existing Data Augmentation Methods

We evaluate the following set of existing data augmentation strategies, which includes operations in the signal (time-domain) space, frequency space, and interpolated signal space, providing good coverage of the possible space of augmentations.

Time Masking. This is a commonly used method in time-series and ECG data augmentation work [88, 63]. We mask out (set to zero) a contiguous fraction $w \in [0, 1]$ of the original signal of length T, We choose a random starting sample t_s and set all samples $[t_s, t_s + wT] = 0$.

[1] For the SMOTE baseline this process is slightly different; details are in Section 3.4.

SpecAugment. A highly popular method for augmenting speech signals [135, 136]. We follow the approach from [98], and apply masking (setting components to zero) in the time and frequency domains as follows. We take the Short-Time Fourier Transform (STFT) of the input signal, and independently mask a fraction w of the temporal bins and frequency bins (this involves setting the complex valued entries in these bins to $0+0j$). The inverse STFT is then used to map the signal back to the time domain.

Discriminative Guided Warping (DGW). Introduced in [89], this method uses Dynamic Time Warping (DTW) [125, 22] to warp a source ECG to match a representative reference signal that is dissimilar to examples from other classes.

SMOTE [29]. A commonly used oversampling strategy, the SMOTE algorithm generates new synthetic examples of the minority class by interpolating minority class samples. Given that many ECG prediction problems are characterized by significant class imbalance, oversampling algorithms are important methods to consider. In contrast to the other methods, the SMOTE algorithm generates an augmented dataset prior to any training, based on a predefined training set size, rather than augmenting examples at each training iteration (as presented in Section 3.4). We set this value to achieve a balanced number of the two classes.

3.4.2 *TaskAug*: A New Augmentation Policy

Motivation. The approaches mentioned so far are simple to implement and can be effective for various problems. However, they are fairly inflexible, given each individually uses only one or two fixed transformations. With ECGs, recall that it is unclear on a per-task basis which augmentations may help or worsen performance (Figure 3-1). Designing a more flexible augmentation strategy that is optimized on a per-task basis could help with this problem, and we now describe such an approach – *TaskAug*.

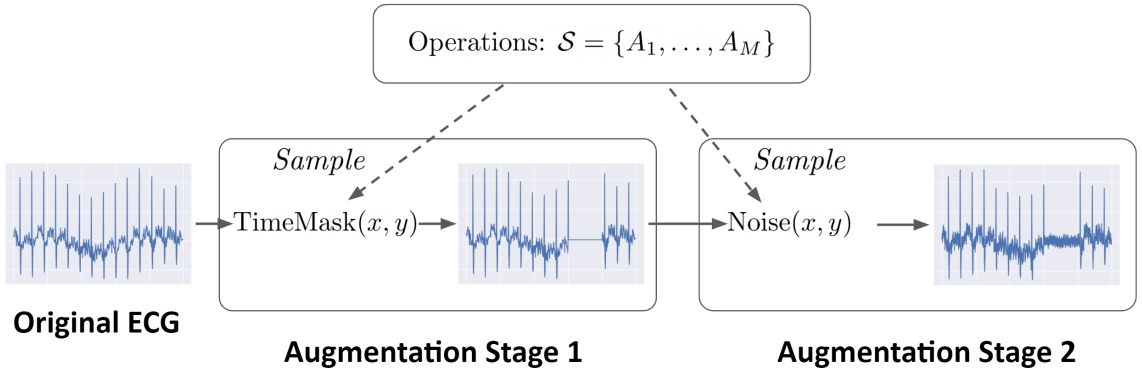


Figure 3-2: **Structure of TaskAug.** Augmentations to apply are sampled from a set of available operations, and applied in sequence. Here we show an example with $K = 2$ stages of augmentation. For clarity, we omit details relating to the per-class magnitudes and probabilities of sampling.

High-level structure. We define a set of operations $\mathcal{S} = \{A_1, \ldots, A_M\}$, each of which is an augmentation function of the form $A_i(x, y; \mu_0, \mu_1)$, where x is the input data point to the augmentation function, y is the label, and $\{\mu_0, \mu_1\}$ represents the augmentation strengths for datapoints of class label 0 and class label 1 respectively. We separately parameterize the augmentation strengths for each class because transformations may corrupt predictive information in the signal for one class but not the other.

The overall augmentation policy consists of a set of K *stages*, where at each stage we: (1) sample an augmentation function A_i to apply; and (2) apply it to the input signal to that stage. This allows composing combinations of operations in a stochastic manner. A high-level schematic is shown in Figure 3-2.

Mathematical definition. The policy is defined following [74]. At each augmentation stage $k \in \{1, \ldots, K\}$ we have a set of operation selection parameters $\pi^{(k)} \in [0, 1]^M$, where $\sum_i \pi_i^{(k)} = 1 \;\; \forall k$. Each vector $\pi^{(k)}$ parameterizes a categorical distribution such that each entry $\pi_i^{(k)}$ represents the probability of selecting operation i at augmentation stage k. We obtain a reparameterizable sample from this categorical distribution (using the Gumbel-Softmax trick, [90, 116]) at each stage to select

the operation to use, as follows:

$$u \sim \text{Categorical}(\pi^{(k)}) \quad \texttt{\# Note that } u \in \mathbb{R}^M \tag{3.1}$$

$$i = \arg\max u \tag{3.2}$$

$$\tilde{x} = \frac{u_i}{\texttt{stop\_grad}(u_i)} A_i(x, y; \mu_0, \mu_1). \tag{3.3}$$

Why the multiplicative factor? We use the multiplicative factor $\frac{u_i}{\texttt{stop\_grad}(u_i)}$ to allow gradient flow to the operation selection parameters π. If we just selected $i = \arg\max u$ and had no scaling in Eqn 3.3, there would be no gradient flow to π, since the arg max operation is not differentiable.

The denominator of this scaling factor is necessary because u_i, obtained from the reparameterized sample from the categorical distribution, is not one-hot. The resulting fraction used as the scaling factor always has magnitude 1, since $|\texttt{stop\_grad}(u_i)| = |u_i|$. When we take the gradient, we get:

$$\frac{\partial}{\partial \pi} \frac{u_i}{\texttt{stop\_grad}(u_i)} = \frac{1}{\texttt{stop\_grad}(u_i)} \frac{\partial u_i}{\partial \pi},$$

so the $\texttt{stop\_grad}(u_i)$ acts as a scaling term.

Suppose a particular augmentation function A_i with strength parameters μ_0 and μ_1 is obtained following Eqns 3.1 and 3.2. Then, denoting the input to this augmentation stage as x with label y, the function A_i that computes the augmented output is defined as:

$$A_i(x, y; \mu_0, \mu_1) = t_i(x; s), \tag{3.4}$$

where t_i is the transformation applied to the signal (e.g., time masking), and s is the transformation strength, computed as follows: $s = y\mu_1 + (1 - y)\mu_0$. See Appendix B.1 for a detailed example of the different steps in applying TaskAug.

Extension to multiclass and multilabel settings. Our instantiation of TaskAug is for the binary classification setting, since this is the scenario we consider in our

experiments. One way to extend this formulation to multiclass/multilabel problems is by defining an operation selection probability matrix and strength matrix at each augmentation stage. The operation selection probabilities and operation strengths for a given example are then obtained by taking the matrix product of the relevant parameter matrix and the label vector y. This is not necessarily an optimal approach – a more general setup is computing the probabilitites/strengths for transformations through a learnable non-linear transformation of the label vector.

Optimizing Policy Parameters

Although the defined policy is flexible, it introduces many new parameters – for a binary problem, there are M operation selection parameters for the categorical distributions at each stage, and 2 strength parameters at each stage, resulting in $K \times (M + 2)$ total parameters [2]. Finding effective values for these parameters with random/grid search or Bayesian optimization is computationally expensive since they require training models many times with different parameter settings. Amortized strategies such as hypernetworks do not scale well to hyperparameter spaces of this size. We therefore use a gradient-based learning scheme to learn these parameters online.

We optimize policy parameters to minimize a model's validation loss, which is computed using non-augmented data. Following prior work [113, 74, 146], we alternate gradient updates on the network parameters θ and the augmentation parameters ϕ by iterating the following steps:

- Optimize the model parameters θ for P steps: at each step, sample a batch (x, y) of data from $\mathcal{D}^{(\text{train})}$, augment the batch with the augmentation policy to obtain $(\tilde{x}, y)$, compute the predicted label $\hat{y}$, and update the model parameters using gradient descent: $\theta \leftarrow \theta - \eta \nabla \mathcal{L}(y, \hat{y})$. Let the base model parameters after P update steps be denoted as $\hat{\theta}(\phi)$.

- Compute the validation loss $\mathcal{L}_V$ using $\hat{\theta}(\phi)$ and an un-augmented batch from

[2] With the multiclass extension, this could be substantially larger.

the validation dataset.

- Perform a gradient update on the augmentation parameters ϕ using implicit differentiation, as follows. First, we use the chain rule to re-express the gradient wrt the augmentation parameters:

$$\frac{\partial \mathcal{L}_V}{\partial \phi} = \frac{\partial \mathcal{L}_V}{\partial \hat{\theta}} \times \frac{\partial \hat{\theta}}{\partial \phi}.$$

We compute this gradient of interest by evaluating both terms on the RHS of this equation.

The first term on the RHS can be found exactly using standard backpropagation, since it is represents the gradient of a simple function of the network's output (the loss) wrt the model parameters.

The second term is more challenging to compute – it represents the gradient of the (partially) optimized base model parameters wrt TaskAug policy parameters. Computing this naively by explicitly differentiating through several steps of optimization is too computationally expensive. To get around this, we re-express this second term using the implicit function theorem (IFT) as in [113]. Using $\mathcal{L}_T$ to denote the training loss, the IFT allows us to write this second term as:

$$\frac{\partial \hat{\theta}}{\partial \phi} = -\left[\frac{\partial^2 \mathcal{L}_T}{\partial \theta \, \partial \theta^T}\right]^{-1} \times \frac{\partial^2 \mathcal{L}_T}{\partial \theta \, \partial \phi^T} \Big|_{\hat{\theta}(\phi)}, \tag{3.5}$$

which is a product of an inverse Hessian and a matrix of mixed partial derivatives. Adopting the algorithm from [113], we approximate this with a truncated Neumann series with 1 term, and implicit vector-Jacobian products. The augmentation parameters are then updated: $\phi \leftarrow \phi - \eta \frac{\partial \mathcal{L}_V}{\partial \phi}$.

Training Algorithm. Presented more formally, the algorithm to jointly optimize base model parameters and TaskAug policy parameters is given in Algorithm 1, mirroring the approach used in [146]. Note that this a nested optimization algo-

rithm, learning the TaskAug parameters by optimizing through the training of the base model. A similar idea is presented in Chapter 2 when optimizing a meta-initialization for few-shot learning, and in Chapter 4 when optimizing pre-training meta-parameters.

Algorithm 1 Optimizing TaskAug parameters.

Initialize base model parameters θ and TaskAug parameters ϕ
for $t = 1, \ldots, T$ **do**
  Compute training loss, $\mathcal{L}_T(\theta)$
  Compute $\frac{\partial \mathcal{L}_T}{\partial \theta}$
  Update $\theta \leftarrow \theta - \eta_\theta \frac{\partial \mathcal{L}_T}{\partial \theta}$
  if $t \,\%\, P == 0$ **then**
    Set $\hat{\theta} = \theta$
    Compute the validation loss, $\mathcal{L}_V(\hat{\theta})$
    Compute $\frac{\partial \mathcal{L}_V}{\partial \hat{\theta}}$
    Approximate $\frac{\partial \hat{\theta}}{\partial \phi}$ using Equation 3.5.
    Compute the derivative $\frac{\partial \mathcal{L}_V}{\partial \phi} = \frac{\partial \mathcal{L}_V}{\partial \hat{\theta}} \times \frac{\partial \hat{\theta}}{\partial \phi}$ using the previous two steps.
    Update $\phi \leftarrow \phi - \eta_\phi \frac{\partial \mathcal{L}_V}{\partial \phi}$
  end if
end for

By using this algorithm, augmentation parameters are learned on a per-task basis. Analyzing the learned parameters helps to understand which augmentations are useful for different problems – we return to this in Section 3.5.2.

Computational cost. Optimizing policy parameters in this manner is significantly more computationally efficient than running a grid search over parameter values. With $P = 1$, running this algorithm has about $2 - 3\times$ the computational cost of training without any augmentations. Further discussion on how to set P is given in Appendix B.1.2.

3.5 Experiments

We evaluate the data augmentation strategies on ECG prediction tasks. We have two main experimental questions: (1) in what settings can data augmentation be

beneficial, and (2) when data augmentation does help, which augmentation strategies are most effective? To investigate these questions, we consider a range of settings that cover three different 12-lead ECG datasets and eight prediction tasks of varying difficulty, class imbalance, and training set sizes.

3.5.1 Experimental Setup

Datasets and Tasks

We highlight key information about our datasets and tasks here, with a summary in Table 3.1.

Dataset A is from Massachusetts General Hospital (MGH) and contains paired 12-lead ECGs and labels for different cardiac abnormalities. Of the available labels in the dataset, we select Right Ventricular Hypertrophy (RVH) and Atrial Fibrillation (AFib) as two of the predictive tasks in our evaluation. These were chosen because (1) they have been previously studied as prediction targets from the ECG [42, 109], and (2) they have low positive prevalence: 1% for RVH, and 5% for AFib, and therefore help to understand the impact of data augmentation for imbalanced prediction problems.

Dataset B is PTB-XL [191, 61], an open-source dataset of 12-lead ECGs. Each ECG has labels for four different categories of cardiac abnormality. This dataset has been used in prior work to evaluate ECG predictive models [63, 98]. We also use this dataset in Chapter 4, where we consider this task of predicting cardiac abnormalities from the ECG in a semi-supervised learning setting.

Dataset C is from the same hospital (MGH) as Dataset A and contains paired ECGs and labels for two hemodynamics parameters, Cardiac Output (CO) and Pulmonary Capillary Wedge Pressure (PCWP). These measures of cardiac health are important in deciding treatment strategies for patients with cardiac disease [202,

Dataset	Task name	Prevalence	Abnormality type	#ECGs/#patients
Dataset A	Right Ventricular Hypertrophy (RVH)	1%	Structural	705057/705057
	Atrial Fibrillation (AFib)	5%	Electrical	705057/705057
Dataset B (PTB-XL)	Hypertrophy (HYP)	12%	Structural	21837/18885
	ST/T Change (STTC)	22%	Ischemia	21837/18885
	Conduction Disturbance (CD)	24%	Electrical	21837/18885
	Myocardial Infarction (MI)	25%	Ischemia	21837/18885
Dataset C	Low Cardiac Ouput (CO)	4%	Hemodynamics	6290/4051
	High Pulmonary Capillary Wedge Pressure (PCWP)	26%	Hemodynamics	6290/4051

Table 3.1: Summary information about the datasets and tasks considered in our empirical evaluation.

86, 174]. As outlined in Chapter 1, these parameters can typically only be measured accurately through an invasive cardiac catheterization procedure. [8, 79]. As a result, datasets with paired ECGs and hemodynamics measurements are relatively small. Considering the use of data augmentations to improve model performance in this limited data regime is therefore clinically relevant. We specifically consider inferring abnormally low Cardiac Output, and abnormally high mean Pulmonary Capillary Wedge Pressure. In Chapter 4 and Appendix E we return to this question of non-invasive estimation of hemodynamics parameters – in Chapter 4, inferring hemodynamics from multimodal ICU time series, and in Appendix E, focusing on a specific cohort of patients with heart failure (an important application area for these predictive models).

Note that the tasks considered cover different classes of cardiac abnormalities: ischemia (MI, STTC), structural (HYP, RVH), electrical (CD, AFib), and abnormal hemodynamics (low CO, high PCWP).

Dataset splitting. Since the value of data augmentation can depend on the amount of training data, we train on different dataset sizes. For the non-hemodynamic tasks (Datasets A and B), we generate development datasets with 1000, 2500, and 5000 ECGs. On the more challenging hemodynamics inference tasks (Dataset C), for elevated PCWP, we consider two settings: using a development set of size 1000, and using the full dataset. For low CO, we only use the full dataset, since reducing the

dataset size led to poor quality models.

In each setting, we split datasets into development and testing sets on a patient-level (no patient is in both sets). We split the development set into an 80-20 training-validation split.

TaskAug Transformations

Based on prior work in time series and ECG data augmentation [88, 121] we use the following transformations in the TaskAug policy. Mathematical descriptions are in Appendix B.1.

- **Random temporal warp:** The signal is warped with a random, diffeomorphic temporal transformation. This is formed by sampling from a zero mean, fixed variance Gaussian at each temporal location in the signal to obtain a velocity field, and then integrating and smoothing (following [9, 10]) to generate a temporal displacement field, which is applied to the signal. The variance is the strength parameter, with higher variance indicating more warping.

- **Baseline wander:** A low-frequency sinusoidal component is added to the signal, with the amplitude of the sinusoid representing the strength.

- **Gaussian noise:** IID Gaussian noise is added to the signal, with the strength parameter representing the variance of the Gaussian.

- **Magnitude scale:** The signal amplitude is scaled by a number drawn from a scaled uniform distribution, with the scale being the strength parameter.

- **Time mask:** A random contiguous section of the signal is masked out (set to zero) [3].

- **Random temporal displacement:** The entire signal is translated forwards or backwards in time by a random temporal offset, drawn from a uniform distribution scaled by a strength parameter.

[3]Optimizing the size of the masked proportion is difficult since it is not differentiable. As a result, we fix the strength of this transformation (the masking proportion)

Note that our instantiation of the augmentation policy could utilize many more operations, but we keep it to this number for simplicity and to assist in interpreting the learned policies.

Implementation Details

Network architecture. We standardize the network architecture to be a 1D convolutional network, based on the ResNet-18 architecture, since prior work has shown architectures of this form to be effective with ECG data [47]. Full architectural details are in the appendix.

Training Details. On Datasets A and B, all models are trained for 100 epochs, using early stopping based on validation loss. For the hemodynamics inference problems on Dataset C, we train models for 50 epochs with early stopping (since we observed significant overfitting after this point). We consider 15 random development/testing set splits for Datasets A and C (lower prevalences for some tasks meant that performance was more variable with fewer runs), and 5 splits for Dataset B. We train models using the Adam optimizer and a learning rate of 1e-3. This value resulted in stable and effective training across all models. As evaluation, we take the model that obtains the best validation set performance and evaluate its AUROC on the held-out testing set, and report mean/standard error across runs. We also report results for a baseline (NoAugs) that does not use any data augmentation.

Augmentation Hyperparameters. In TaskAug, we set the number of augmentation stages to $K = 2$ (defined in Section 3.4.2), following prior work [74]. For the number of model optimization steps P (defined in Section 3.4.2), we evaluate both $P = 1$ and $P = 5$, and select the best performing setting based on validation set loss. Further discussion on the choice of P is in Appendix B.1.

For Time Masking and SpecAugment, we search over the masking window, considering $w \in \{0.1, 0.2\}$ for SpecAugment (range based on [98]) and $w \in \{0.1, 0.2, 0.5\}$ for Time Masking (range based on [63]).

	Dataset A		Dataset B			
	RVH	AFib	MI	HYP	STTC	CD
NoAugs	72.6 ± 2.7	79.8 ± 1.4	80.0 ± 0.8	**84.3 ± 1.4**	87.6 ± 0.8	82.2 ± 0.6
TaskAug	**78.4 ± 1.9**	**82.8 ± 1.0**	**82.3 ± 0.5***	83.7 ± 0.5	**87.8 ± 0.4**	83.1 ± 0.4
SMOTE	75.9 ± 1.8	79.0 ± 1.4	81.2 ± 0.6	80.4 ± 0.6	87.0 ± 0.5	82.6 ± 0.8
DGW	73.6 ± 1.7	77.4 ± 1.5	81.1 ± 0.6	83.9 ± 0.7	87.5 ± 0.5	81.8 ± 1.0
SpecAug	77.9 ± 1.7	77.2 ± 2.1	81.1 ± 0.7	83.5 ± 0.8	87.7 ± 0.4	82.2 ± 0.7
TimeMask	72.8 ± 2.1	77.9 ± 1.9	81.1 ± 1.3	82.9 ± 0.7	87.7 ± 0.7	**83.8 ± 1.1**

Table 3.2: **Augmentation strategies improve AUROC on detecting most cardiac abnormalities in the low-sample regime ($N = 1000$), and TaskAug is among the best-performing methods.** Table shows mean and standard error of AUROC (best-performing method bolded, second best underlined, statistically significant ($p < 0.05$) improvement over NoAugs marked *). The impact of augmentations is task-dependent, with some tasks (such as RVH, MI) showing improved performance on average with almost all strategies, and others (HYP) showing no improvement with any strategy. TaskAug is among the best methods across tasks, and improves performance on tasks such as AFib where no other augmentations help.

3.5.2 Results

Quantitative Results

Non-hemodynamics tasks. We first analyze performance of augmentation strategies on the non-hemodynamics tasks. Given that performance improvements are most evident in the lowest sample regimes for both datasets ($N = 1000$), we focus on this setting with results shown in Table 3.2. Results for the higher sample regimes are in the appendix. We summarize key findings here.

The value of augmentation varies by task. For some tasks such as RVH and MI, almost all augmentation strategies lead to performance improvements. On other tasks such as STTC and HYP, performance is the same or worse when applying augmentations. The improvement seen with RVH could be due to the fact that it is particularly low prevalence (1%), so all augmentation strategies have an oversampling effect and thus boost performance.

TaskAug performs well on average. TaskAug almost always improves on the NoAugs baseline, and even boosts performance on some tasks where other augmentations worsen performance (AFib). Although TaskAug does not always result in a

	Dataset C		
	Low CO	High PCWP: $N = 1000$	High PCWP: All Data
NoAugs	65.9 ± 1.2	66.7 ± 0.7	74.4 ± 0.5
TaskAug	<u>68.2 ± 1.0</u>	**67.9 ± 0.7**	**75.1 ± 0.4**
SMOTE	66.0 ± 1.4	67.2 ± 0.5	73.6 ± 0.5
DGW	**68.3 ± 0.9**	66.4 ± 0.6	74.9 ± 0.4
SpecAug	66.1 ± 0.9	66.4 ± 1.3	<u>75.0 ± 0.4</u>
TimeMask	66.8 ± 1.1	<u>67.3 ± 0.4</u>	74.6 ± 0.4

Table 3.3: **Training with data augmentation improves AUROC on two hemodynamics inference tasks, and TaskAug again is among the best-performing methods.** Table shows mean and standard error of AUROC (best-performing method bolded, second best underlined). All methods are comparable with or improve on the no augmentation baseline for Low CO prediction, possibly because of the low prevalence of the label (4%). The performance of methods on the High PCWP task is more variable across the two sample sizes. TaskAug obtains improvements in all three settings considered.

statistically significant ($p < 0.05$) improvement in AUROC, it is the only method to significantly improve AUPRC over NoAugs on the low-prevalance tasks, RVH and AFib (see Appendix B.3, Table B.1).

When TaskAug results in lower performance than other augmentation strategies (e.g., TimeMasking for CD), it is still competitive with these methods and never causes a statistically significant reduction in performance compared to other methods. This suggests that for a new task, it may always be worth using TaskAug to see if performance is boosted. We hypothesise that TaskAug's efficacy is due to its flexible and learned nature, examined in ablation studies (Section 3.5.2).

Performance improvements are smaller on Dataset B. The maximum improvement over the NoAugs baseline in Dataset A (5.8%) is greater than the maximum improvement in Dataset B (2.3%). We hypothesise two reasons for this. Firstly, the prevalence in Dataset B is higher, meaning that augmentations may not have as much of an effect at $N = 1000$. We study this in Appendix B.3, Table B.8, where we examine performance at the $N = 500$ data regime for Dataset B, and find that the maximum improvement (obtained with TaskAug for MI) goes up to 4%.

Secondly, Dataset A has narrower label definitions than Dataset B, and this affects performance, especially with TaskAug. The HYP, STTC, and CD classes of

abnormalities in Dataset B aggregrate many sub-categories together (see Appendix B.2), and these sub-categories might each benefit from different augmentations. In contrast, the labels in Dataset A are fine-grained, and so TaskAug, which optimizes augmentations on a per-task basis, learns more appropriate augmentation strategies. This hypothesis is supported by the fact that with MI (a more fine-grained label than HYP, CD, and STTC) we observe improvements over the NoAugs baseline (clearly seen in the $N = 500$ regime, Appendix B.3, Table B.8).

Performance improvements at higher samples are lower, as seen in the results in Appendix B.3. Augmentations do not worsen performance, however, and some tasks (STTC, CD) benefit a small amount, $\sim +1\%$ AUROC.

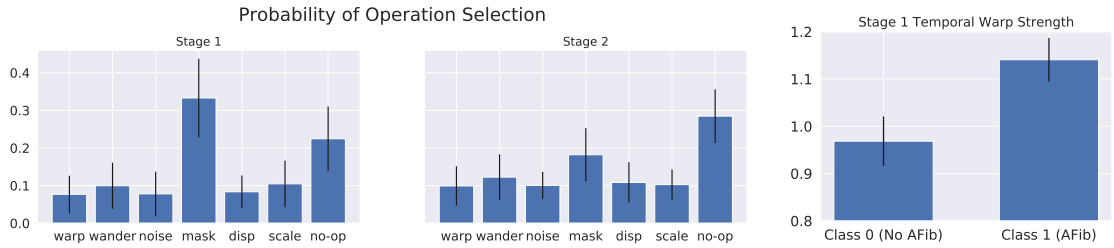
Hemodynamics tasks. Table 3.3 presents results for performance on the challenging hemodynamics prediction tasks. All methods are comparable with or improve on the no augmentation baseline for low CO prediction, likely because of the low prevalence of the positive label (4%). For inferring high PCWP, at both low sample and higher samples, TaskAug obtains improvements in performance (though not significant at the $p < 0.05$ level); however, other methods do not consistently improve on the no augmentation baseline. Although improvements in AUROC are not statistically significant, we observe significant improvements with TaskAug in AUPRC for low CO detection (see Appendix B.3, Table B.3). Again, we see that the benefit of augmentation varies with the task, prevalance, and dataset size, and that TaskAug is better than or competitive with other strategies.

Analyzing Learned Policies

We analyze the learned policies for three of the predictive tasks: AFib, PCWP, and RVH (Appendix B.3).

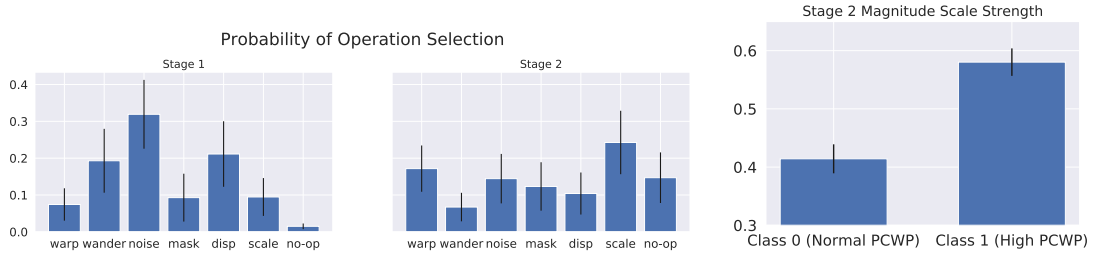
AFib, Figure 3-3. We see that time mask has a high probability of selection (Figure 3-3a). Since AFib is characterized in the ECG by an irregular R peak-R peak interval [42], which is often present regardless of which section of ECG is selected,

Figure 3-3: **The TaskAug policy for Atrial Fibrillation detection.** We focus on the probability of selecting each transformation in both augmentation stages (left) and the optimized temporal warp strengths in the first stage (right). We show the mean/standard error of these optimized policy parameters over 15 runs. Given the characteristic features of AFib (e.g., irregular R-R interval), Time Masking is likely to be label preserving and therefore it is sensible that it has a high probability of selection. The temporal warp strength for positive samples is higher than that for negative samples, which makes sense since time warping a negative sample too strongly could change its label.



(a) Operation selection probabilities (b) Warp strengths

Figure 3-4: **The TaskAug policy for detecting elevated Pulmonary Capillary Wedge Pressure.** We focus on the probability of selecting each transformation in both augmentation stages (left) and the optimized magnitude scaling strengths in the second stage (right). We show the mean/standard error of these optimized policy parameters over 15 runs. There exists little domain knowledge about what features in the ECG may encode elevated PCWP, so examining the learned augmentations here could provide hypotheses of invariances in the data. Of interest is that the positive class is augmented with stronger magnitude scaling than the negative class, suggesting that scaling negative examples could affect their labels.



(a) Operation selection probabilities (b) Magnitude scale strength

time masking is likely label preserving, and is a sensible choice. Considering the learned time warp strength in Figure 3-3b, we observe that signals labelled negative for AFib are warped less strongly than those with AFib, again sensible since time warping may affect the label of a signal and introduce AFib in a signal where it was not originally present.

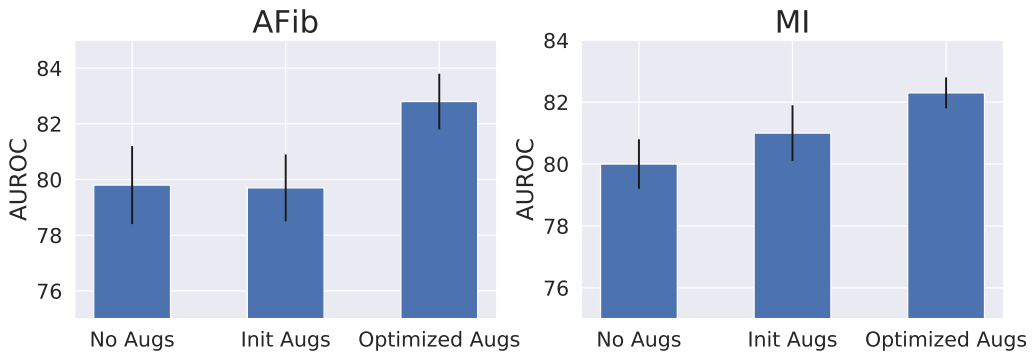


Figure 3-5: **Optimizing the TaskAug policy parameters results in performance improvements.** We show the mean/standard error of AUROC over 15 runs for AFib and over 5 runs for MI. Without optimizing policy parameters (InitAug), performance is comparable to not using augmentations at all, indicating the importance of learning the policy parameters.

PCWP, Figure 3-4. We have limited domain understanding of what augmentations might be label preserving and help model performance, since detecting high PCWP from ECGs is not something clinicians are typically able to do [164]. Analyzing the augmentations could provide hypotheses about what features in the data encode the class label. Noise, displacement, and baseline wander all obtain higher weight in the first stage, and scaling obtains higher weight in the second stage. The high weight assigned to noise could be to help the model build invariance to it, and not use it as a predictive aspect of the signal. Studying the magnitude scaling in Figure 3-4b, we see positive examples are scaled significantly more than negative examples. It is possible that negative examples are more sensitive to scale, and scaling them pushes them into positive example space. The positive examples may have more variance in scaling, and thus scaling them further has less of an effect.

Ablation Studies

How much does optimizing augmentations help? Our results show that TaskAug offers improvements in performance. In Figure 3-5, we examine how the optimization of the augmentation policy parameters (operation selection probabilities and magnitudes, Section 3.4.2) affects performance, considering the AFib and MI detection tasks and $N = 1000$. We compare the performance of optimizing the policy param-

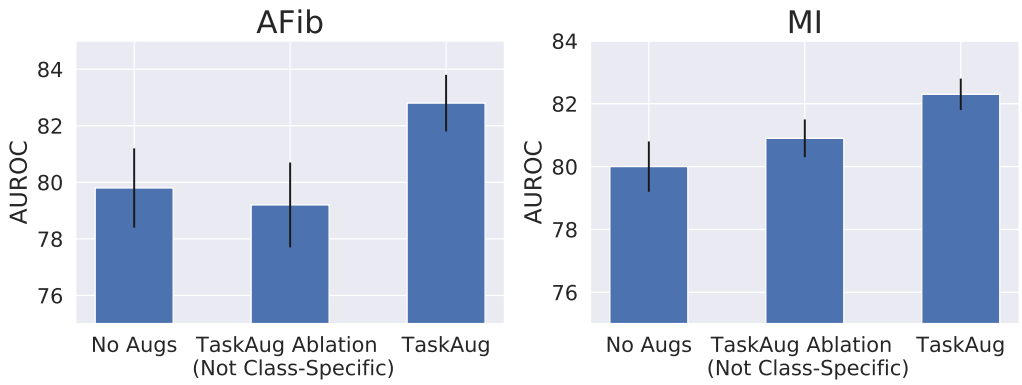


Figure 3-6: **Class-specific magnitude parameters in TaskAug lead to improvements in performance.** We show the mean/standard error of AUROC over 15 runs for AFib and over 5 runs for MI. The improvements from using class-specific magnitude parameters is particularly clear for tasks such as AFib where some operations may not be label preserving.

eters vs. keeping them fixed at their initialized values and training. We observe improvements in performance through the optimization process, suggesting that it is not only the range of augmentations that leads to improved performance, but also the optimization of the policy parameters. In Appendix B.3, we study this at different dataset sizes and find that performance is improved by optimization at each size.

How much do class-specific magnitudes help? TaskAug instantiates magnitude parameters for the augmentation operations on a per-class basis, as described in Section 3.4.2, allowing positive and negative examples to be augmented differently. We examine this further, considering the AFib and MI detection tasks and $N = 1000$. We compare performance using class-specific magnitude parameters (the positive and negative examples have independent augmentation magnitudes μ_1 and μ_0) vs. using global magnitude parameters (the positive and negative examples are forced to have the same augmentation magnitude: $\mu = \mu_0 = \mu_1$). Results are shown in Figure 3-6. We observe noticeable improvements in performance with class-specific magnitude parameters, demonstrating the importance of independently specifying magnitudes for the two classes. In Appendix B.3, we study this at different dataset sizes and find that performance is improved at each size.

Summary and Best Practices

- Training with data augmentations does not always improve model performance, and might even hurt it. The impact of augmentation depends on nature of the task, positive class prevalence, and dataset size.

- Augmentations are most often useful in the low-sample regime. Where the prevalence is particularly low (see results for RVH detection) various augmentation strategies improve performance, perhaps by functioning as a form of oversampling.

- Data augmentations do not always improve performance at high sample sizes, but do not hurt it.

- TaskAug, our proposed augmentation strategy, is the most effective method on average, and could therefore be the first augmentation strategy one tries on a new ECG prediction problem. TaskAug defines a flexible augmentation policy that is optimized on a task-dependent basis, which directly contributes to its effectiveness.

- TaskAug also offers insights as to what augmentations are most effective for a given problem, which could be useful in novel prediction tasks (e.g., hemodynamics inference) to suggest what aspects of the ECG determine the class label.

3.6 Scope and Limitations

Nature of predictive tasks. We focused in this chapter on developing a task-adaptive data augmentation strategy for data-scarce binary prediction tasks from ECGs. We did not consider extensions to multiclass or multilabel settings, both of which are valuable directions of future work (and are discussed in the methods section of this chapter).

Choice of base transformations. Our data augmentation approach incorporates a base set of ECG transformations. In our implementation, we kept the size of this set to be relatively small for simplicity; however, we could include a larger number of transformations in this set.

More broadly, when considering appropriate base transformations, there are some points that we did not consider in detail in this work, namely: (1) Ensuring that the transformations *always* preserve physiological consistency of the signal (such as the voltage relation between different ECG leads); and (2) Transforming signals in a fine-grained fashion (such as only operating on specific parts of the cardiac cycle). Note that if the set of base transformations is chosen poorly, or none of the base transformations are label preserving for the predictive task(s) of interest, then our method will not be effective.

Hyperparameter initialization. An important detail in using our method is to ensure that the initialization point for the augmentation strengths and probabilities are reasonable. If these are poorly initialized (e.g., apply a destructive masking transformation that masks almost the entire signal), then the gradient based approach we outline would not be able to recover a useful augmentation strategy.

3.7 Conclusion

In this chapter, we studied a particular data-scarce machine learning task: improving supervised learning performance on small, labelled datasets of electrocardiogram (ECG) data. This direction of inquiry was motivated by the fact that certain clinically relevant ECG predictive tasks (such as inferring abnormal hemodynamics, as discussed in Chapter 1) face data scarcity challenges, with it being difficult to construct large paired datasets of ECGs and labels. Our focus in this chapter was to explore whether data augmentation strategies, which have been effective in many data-scarce prediction problems, could be used to improve model performance on data-scarce ECG prediction tasks.

We conducted an experimental evaluation of existing data augmentation strategies on three ECG datasets and eight distinct predictive tasks, and found that these existing strategies are not always helpful, and on some tasks, may even worsen performance. To improve on the shortcomings of existing augmentation strategies, we proposed *TaskAug*, a new, learnable data-augmentation strategy for ECGs. TaskAug is among the strongest performing methods on all predictive tasks, and the learned TaskAug augmentation policies are additionally interpretable, providing insight as to what transformations are most important for different problems. We compiled a set of best practices from our empirical evaluation, specifying what characteristics of predictive problems (for example, low prevalence of the positive class) might result in augmentations being most useful.

Chapter 4

Nested Optimization for Improved Pre-Training

4.1 Introduction

So far in this thesis, we have presented contributions to two paradigms of data-efficient machine learning (ML): few-shot learning (Chapter 2) and supervised learning on small, labelled datasets (Chapter 3). In this chapter and the next, we investigate a different data-efficient ML paradigm: pre-training (PT) followed by fine-tuning (FT).

In PT followed by FT, the situation is typically as follows. We wish to train a predictive model for a task with limited labeled data (the FT task), and additionally have access to a large, related dataset (the PT dataset), which may be labelled or unlabelled. Learning a model proceeds in two phases. We first use the PT dataset to learn an effective initialization for the model parameters (we *pre-train* the model). Then, from this initialization, the model's parameters are optimized further for the FT task, using the (often) small, labelled FT dataset.

This paradigm has seen widespread use in transfer learning [50, 167, 60, 82, 141, 46, 102, 207, 100, 84] and semi-supervised learning [31, 30, 76] (semi-supervised learning is the focus of Chapter 5), and has led to impressive performance in many domains, including computer vision [50, 167, 60, 102, 207, 100], natural language processing [82, 141, 46, 111, 95], graph structured prediction [84], and clinical machine learning

[119, 120, 4, 126].

Although powerful, the PT & FT paradigm introduces additional complexity in the form of high-dimensional, complex PT hyperparameters, such as parameterized data augmentation policies used in contrastive representation learning [30, 74] or the use of task, class, or instance weighting variables in multi-task PT to avoid negative transfer [201]. These hyperparameters can significantly affect the quality of pre-trained models [30], and thus finding techniques to set their values optimally is an important area of research.

Choosing optimal PT hyperparameter values is challenging, and existing methods do not work well. Simple approaches such as random or grid search are inefficient since evaluating a hyperparameter setting requires performing the full, two-stage PT & FT optimization, which may be prohibitively computationally expensive. Gradient-free approaches, such as Bayesian optimization or evolutionary algorithms [94, 173, 122], are also limited in how well they scale to this setting. Gradient-based approaches [115, 112, 114, 113] can be used online to jointly learn hyperparameters and model parameters and can scale to millions of hyperparameters [113], but typically deal with a standard *single-stage* learning problem (e.g., normal supervised learning) and are therefore not directly applicable to the *two-stage* PT & FT learning problem.

In this chapter, we address this gap and propose a method for high-dimensional PT hyperparameter optimization. We first formalize a variant of the PT & FT paradigm, which we call *meta-parameterized pre-training* (Figure 4-1), where *meta-parameters* refer to arbitrary PT hyperparameters or parameterizable architectural choices that can be optimized to improve the learned representations.[1] We outline a nested optimization problem characterizing the optimal meta-parameters and propose a gradient-based method to learn meta-parameters. Our contributions are:

- We formalize *meta-parameterized pre-training*, a variant of the pre-training and fine-tuning (PT & FT) paradigm where PT is augmented to incorporate *meta-parameters*: arbitrary structures that can be optimized to improve learned rep-

[1]We use the term *meta-parameter* since these structures do not directly affect inference of the final model after FT, but instead inform the process of learning this model (by modulating the PT process).

resentations.

- We propose a scalable gradient-based algorithm to learn meta-parameters using a novel method to obtain meta-parameter gradients through the two-stage PT & FT process. Our gradient estimator composes a constant-memory implicit differentiation approximation for the longer PT stage and exact backpropagation through training for the shorter FT stage.

- We show that our algorithm recovers near-optimal meta-parameters in toy experiments on synthetic data.

- We demonstrate that our algorithm improves performance over baselines in two real-world experimental domains involving PT and FT: multitask learning to predict protein functions from graph-structured data [84], and semi-supervised learning to predict cardiac abnormalities from electrocardiography data.

4.2 Related Work

Gradient-based hyperparameter optimization (HO): There are roughly two groups of gradient-based HO algorithms. The simpler and less scalable approach differentiates through training [49, 115]. The other approach assumes that optimization reaches a fixed point, and approximates the best-response Jacobian [20, 112, 114, 113]. Neither of these approaches can be straightforwardly applied to scalably differentiate through two stages of optimization (PT & FT). Direct differentiation through both stages would be too memory-intensive. Approximating the best-response Jacobian using the IFT as in [113] twice is feasible, but requires changing the FT objective to include a proximal term [153], and tuning two sets of interacting approximations. Instead, we compose a constant-memory IFT approximation for the lengthy PT stage with an exact backprop-through-training for the shorter FT stage.

Applications of Nested Optimization: Many prior works frame learning as nested optimization, including few-shot learning [55, 3, 57, 153, 66, 159, 144, 211,

91, 104], neural network teaching [52, 53, 175, 145], learning data augmentation and reweighting strategies [92, 74, 157, 169, 85], and auxiliary task learning [129, 137, 110]. The majority of this work studies nested optimization in the standard one-stage supervised learning paradigm, unlike our setting: the two-stage PT & FT problem. The most closely related works to ours are [201], where PT task weights are learned for a multitask PT problem using electronic health record data, and [203], where a masking policy is learned for masked language modelling PT. In contrast to our work, which introduces the more general framing of meta-parameter optimization, [201] and [203] are focused only on specific instantiations of meta-parameters as task weights and masking policies. The learning algorithms in these works either differentiate directly through truncated PT & FT [203] (which may not be scalable to longer PT/large encoder models), or leverage extensive first-order approximations [201], unlike our more generally applicable approach.

4.3 Problem Setup and Preliminaries

In this section, we define the meta-parameterized pre-training problem, and compare it to traditional fine-tuning and pre-training. A full glossary of notation is in Appendix C.1, Table C.1.

Notation Let the subscript $\bullet$ be a placeholder for either PT (pre-training) or FT (fine-tuning), $\mathcal{X} \subseteq \mathbb{R}^d$ be our input domain, $\mathcal{Y}_\bullet$ and $\hat{\mathcal{Y}}_\bullet$ be the true and predicted output spaces for some model respectively, and $\Theta, \Psi_\bullet, \Phi$ be spaces of parameters for models. We will use $f_\bullet : \mathcal{X}; (\Theta, \Psi_\bullet) \to \hat{\mathcal{Y}}_\bullet$ to refer to a parametric model, with the semicolon separating the input space from the parameter spaces. We then define $f_\bullet = f_\bullet^{(\text{head})} \circ f^{(\text{feat})}$, such that $f^{(\text{feat})}(\cdot; \boldsymbol{\theta} \in \Theta)$ is a *feature extractor* that is transferable across learning stages (e.g., pre-training to fine-tuning), and $f_\bullet^{(\text{head})}(\cdot; \boldsymbol{\psi} \in \Psi_\bullet)$ is a stage-specific *head* that is not transferable. Given a data distribution $\mathsf{x}_\bullet, \mathsf{y}_\bullet \sim \mathcal{D}_\bullet$, parametric model $f_\bullet$, and loss function $\mathcal{L}_\bullet : \hat{\mathcal{Y}}_\bullet \times \mathcal{Y}_\bullet \to \mathbb{R}$, we will also define for convenience a corresponding expected loss $L_\bullet : \Theta, \Psi_\bullet \to \mathbb{R}$

via $L_\bullet(\boldsymbol{\theta}, \boldsymbol{\psi}_\bullet; \mathcal{D}_\bullet) = \mathbb{E}_{\mathcal{D}_\bullet}\left[\mathcal{L}_\bullet(f_\bullet(\boldsymbol{x}_\bullet; \boldsymbol{\theta}, \boldsymbol{\psi}_\bullet), y_\bullet)\right]$. We also adopt the convention that the output of the argmin operator is *any* arbitrary minimum, rather than the set of possible minima, to avoid complications in notation.

4.3.1 Problem Formulation

Supervised Learning (Fig. 4-1A). In a fully-supervised setting (our *fine-tuning* domain), we are given a data distribution $\mathcal{D}_{\text{FT}}$, model f, and loss $\mathcal{L}_{\text{FT}}$. Using a *learning algorithm* Alg_{FT} (e.g., SGD) that takes as input initial parameters $\boldsymbol{\theta}_{\text{FT}}^{(0)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}$, our goal is to approximate the $\mathcal{L}_{\text{FT}}$-optimal parameters:

$$\boldsymbol{\theta}_{\text{FT}}^*, \boldsymbol{\psi}_{\text{FT}}^* = \text{Alg}_{\text{FT}}(\boldsymbol{\theta}_{\text{FT}}^{(0)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}) \approx \underset{\boldsymbol{\theta}\in\Theta, \boldsymbol{\psi}\in\Psi_{\text{FT}}}{\text{argmin}} \; L_{\text{FT}}(\boldsymbol{\theta}, \boldsymbol{\psi}; \mathcal{D}_{\text{FT}}).$$

Pre-training (Fig. 4-1B). For tasks where data is scarce, we can additionally incorporate a *pre-training* step and approximate the optimal initial parameters for FT (i.e., the final pre-trained weights are used as initialization weights of the FT stage), again via an optimization algorithm Alg_{PT}: [2]

$$\boldsymbol{\theta}_{\text{PT}}^* = \text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}) \approx \underset{\boldsymbol{\theta}\in\Theta}{\text{argmin}} \, L_{\text{FT}}(\text{Alg}_{\text{FT}}(\boldsymbol{\theta}, \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}); \mathcal{D}_{\text{FT}}).$$

Meta-Parameterized PT (Fig. 4-1C). In *Meta-Parameterized PT*, we recognize that, in addition to taking as input the PT parameters $\boldsymbol{\theta}$, Alg_{PT} is itself parameterized by a set of *meta-parameters* $\boldsymbol{\phi} \in \Phi$: arbitrary, potentially high dimensional quantities that inform the structure of the algorithm directly. These could represent weighting strategies, data augmentation policies, or sampling processes. The optimal meta-parameters $\boldsymbol{\phi}^{(\text{opt})}$ are the solution to the following *meta-PT* optimization problem:

$$\boldsymbol{\phi}^{(\text{opt})} = \underset{\boldsymbol{\phi}\in\Phi}{\text{argmin}} \, L_{\text{FT}}\left(\text{Alg}_{\text{FT}}\left(\text{Alg}_{\text{PT}}\left(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}\right), \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}\right); \mathcal{D}_{\text{FT}}\right).$$

[2] Note that we discard the PT head $\boldsymbol{\psi}_{\text{PT}}^*$ here as only the PT feature extractor $\boldsymbol{\theta}_{\text{PT}}^*$ is transferred.

4.3.2 Example: Multitask Meta-Parameterized Pre-Training

To make our notation concrete, we instantiate our setup for a multitask pre-training problem.

Problem: Suppose we have a multitask classification dataset, $(\mathcal{X} \times \mathcal{Y})^N$ such that $\mathcal{Y} = \mathcal{Y}_1 \times \cdots \times \mathcal{Y}_K$ consists of labels for K distinct tasks. Of this full set of tasks, we are interested only in a subset of M tasks, $S = \{t_1, \ldots, t_M\} \subseteq \{1, \ldots, K\}$.

Supervised FT: Under supervised FT alone, we can directly average a cross-entropy loss $\mathcal{L}_{\text{CE}}$ over *only the tasks in* S:

$$\mathcal{L}_{\text{FT}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{M} \sum_{j=1}^{M} \mathcal{L}_{\text{CE}}(\hat{y}^{(t_j)}, y^{(t_j)}),$$

and then solve this problem via SGD.

PT: If we assume that S is a *random* subset of the full set of tasks, we can introduce a PT stage over all tasks:

$$\mathcal{L}_{\text{PT}}(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{K} \sum_{i=1}^{K} \mathcal{L}_{\text{CE}}(\hat{y}^{(i)}, y^{(i)}),$$

followed by FT on S alone. As S is a random subset, leveraging all tasks for PT is well motivated and may improve performance.

Meta-Parameterized PT: In the case where T is not a random subset, the PT strategy described above is no longer well-motivated. However, using meta-parameterized PT, we can still effectively pre-train by introducing the meta-parameters that weight the tasks $\boldsymbol{\phi} = \begin{bmatrix} \phi_1 & \ldots & \phi_K \end{bmatrix}$ and modulate the loss function $\mathcal{L}_{\text{PT}}$:

$$\mathcal{L}_{\text{PT}}(\hat{\boldsymbol{y}}, \boldsymbol{y}; \boldsymbol{\phi}) = \sum_{i=1}^{K} \phi_i \mathcal{L}_{CE}(\hat{y}^{(i)}, y^{i}).$$

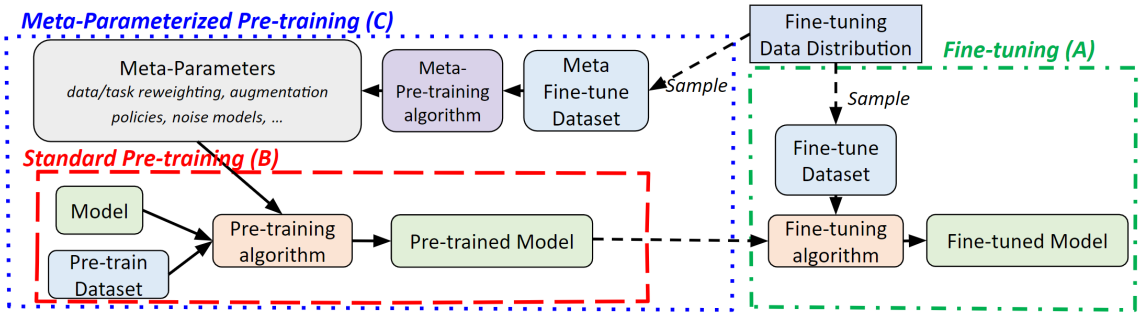


Figure 4-1: **Meta-Parameterized Pre-Training.** A paradigm where meta-parameters — rich, potentially high dimensional structures that generalize PT hyperparameters — are incorporated in PT to improve the learned representations. Meta-parameters are optimized in a *meta-PT* phase, using data from FT task(s) in a meta-FT dataset. The FT and meta-FT datasets are (potentially overlapping) samples from the FT data distribution.

With optimal meta-parameters $\boldsymbol{\phi}^{(\text{opt})}$, the PT stage will leverage only that subset of tasks that best informs the final FT performance. This setting mirrors our real-world experiment in Section 4.6.

4.4 Methods: Optimizing Meta-Parameters

We now introduce our gradient-based algorithm to optimize meta-parameters. We first describe how to efficiently approximate meta-parameter gradients through the two-stage PT and FT optimization. We then present our algorithm, and outline practical considerations when using it.

4.4.1 Efficient Computation of Meta-Parameter Gradients

We begin by defining:

$$g(\boldsymbol{\phi}; \boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}) = L_{\text{FT}}\Big(\underbrace{\text{Alg}_{\text{FT}}\big(\overbrace{\text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi})}^{\text{Parameter } \boldsymbol{\theta}_{\text{PT}}}, \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}\big)}_{\text{Parameters } \boldsymbol{\theta}_{\text{FT}}, \boldsymbol{\psi}_{\text{FT}}}; \mathcal{D}_{\text{FT}} \Big), \tag{4.1}$$

so that $\boldsymbol{\phi}^{(\text{opt})} = \text{argmin}_{\boldsymbol{\phi} \in \Phi}\, g(\boldsymbol{\phi})$.

We also define two best-response values:

$$\boldsymbol{\theta}^*_{\mathrm{PT}}(\boldsymbol{\phi}) = \mathrm{Alg}_{\mathrm{PT}}(\boldsymbol{\theta}^{(0)}_{\mathrm{PT}}, \boldsymbol{\psi}^{(0)}_{\mathrm{PT}}; \mathcal{D}_{\mathrm{PT}}, \boldsymbol{\phi}),$$
$$\boldsymbol{\theta}^*_{\mathrm{FT}}(\boldsymbol{\phi}), \psi^*_{\mathrm{FT}}(\boldsymbol{\phi}) = \mathrm{Alg}_{\mathrm{FT}}(\boldsymbol{\theta}^*_{\mathrm{PT}}(\boldsymbol{\phi}), \boldsymbol{\psi}^{(0)}_{\mathrm{FT}}; \mathcal{D}_{\mathrm{FT}}).$$

We do not explicitly include the dependence of the best responses on the initialization values for notational convenience.

With these defined, we now consider the desired gradient term, $\frac{\partial g}{\partial \boldsymbol{\phi}}$. Under our definitions, the direct partial derivatives $\frac{\partial L_{\mathrm{FT}}}{\partial \boldsymbol{\phi}}$ and $\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \boldsymbol{\phi}}$ are zero, so $\frac{\partial g}{\partial \boldsymbol{\phi}}$ reduces to a simple expression of the chain rule:

$$\left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}'} = \underbrace{\left.\frac{\partial L_{\mathrm{FT}}}{\partial \left[\boldsymbol{\theta}_{\mathrm{FT}}, \quad \boldsymbol{\psi}_{\mathrm{FT}}\right]}\right|_{\boldsymbol{\theta}^*_{\mathrm{FT}}(\boldsymbol{\phi}'), \boldsymbol{\psi}^*_{\mathrm{FT}}(\boldsymbol{\phi}')}}_{\text{FT Loss Gradient}} \times \overbrace{\left.\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \boldsymbol{\theta}_{\mathrm{PT}}}\right|_{\boldsymbol{\theta}^*_{\mathrm{PT}}(\boldsymbol{\phi}')}}^{\text{FT Best Response Jacobian}} \times \underbrace{\left.\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}'}}_{\text{PT Best Response Jacobian}}. \tag{4.2}$$

The FT Loss Gradient term on the RHS of (4.2) is easily computed using backpropagation. Computing the other two terms is more involved, and we detail each below, beginning with the PT best response Jacobian. The full algorithm with both gradient estimation terms is provided in Algorithm 2.

PT Best Response Jacobian $\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \boldsymbol{\phi}}$ Using recent work in hyperparameter optimization with implicit differentiation [113], we re-express this term using the implicit function theorem (IFT). If we assume that $\boldsymbol{\theta}^*_{\mathrm{PT}}(\boldsymbol{\phi}) = \mathrm{Alg}_{\mathrm{PT}}\left(\boldsymbol{\theta}^{(0)}_{\mathrm{PT}}; \mathcal{D}_{\mathrm{PT}}, \boldsymbol{\phi}\right)$ is a good approximation of $\mathrm{argmin}_{\boldsymbol{\theta}\in\Theta} L_{\mathrm{PT}}\left(\boldsymbol{\theta}; \mathcal{D}_{\mathrm{PT}}, \boldsymbol{\phi}\right)$ (i.e., the PT model converges to $\mathcal{L}_{\mathrm{PT}}$-optimal parameters), then under certain smoothness and regularity assumptions on the PT parameters and meta-parameters, the IFT allows us to re-express $\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \boldsymbol{\phi}}$ as:

$$\left.\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}'} = -\left[\frac{\partial^2 L_{\mathrm{PT}}}{\partial \boldsymbol{\theta}_{\mathrm{PT}}\, \partial \boldsymbol{\theta}^\top_{\mathrm{PT}}}\right]^{-1} \times \left.\frac{\partial^2 L_{\mathrm{PT}}}{\partial \boldsymbol{\theta}_{\mathrm{PT}}\, \partial \boldsymbol{\phi}^\top}\right|_{\boldsymbol{\theta}^*_{\mathrm{PT}}(\boldsymbol{\phi}'), \boldsymbol{\phi}'}, \tag{4.3}$$

which is the product of the inverse Hessian and a matrix of mixed partial derivatives. Following [113], the inverse can be efficiently approximated using a truncated Neumann series.

FT Best Response Jacobian $\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}$ First, note that without additional constraints on Alg_{FT}, the FT best response Jacobian may be zero. This is because L_{FT} has no functional dependence on the variable $\boldsymbol{\theta}_{\text{PT}}$ and, if we assume the convergence point $\boldsymbol{\theta}^*_{\text{FT}}$ is stable (as we did for the PT best response Jacobian), this implies that the gradient of $\boldsymbol{\theta}^*_{\text{FT}}$ with respect to $\boldsymbol{\theta}_{\text{PT}}$ would be zero. To enable effective learning, we must therefore either (1) impose restrictions on Alg_{FT} to ensure there is a dependence between the initialization point and the final loss value (e.g., proximal regularization [153]) or (2) leverage methods that do not differentiate through Alg_{FT} through convergence, since at non-converged points we will still observe nonzero L_{FT}-gradients [85, 137]. Given that the FT phase often involves shorter optimization horizons than PT, we take approach 2 here, and iteratively update $\boldsymbol{\theta}_{\text{FT}}$ for K steps. We first initialize the FT head $\boldsymbol{\psi}^{(0)}_{\text{FT}}$ and then compute:

$$
\begin{aligned}
\boldsymbol{\theta}^{(0)}_{\text{FT}} &= \texttt{copy}(\boldsymbol{\theta}^*_{\text{PT}}) \qquad \text{(init with PT solution, implicitly performing stop gradient)} \\
\boldsymbol{\theta}^{(k)}_{\text{FT}}, \boldsymbol{\psi}^{(k)}_{\text{FT}} &= \left[\boldsymbol{\theta}^{(k-1)}_{\text{FT}}, \quad \boldsymbol{\psi}^{(k-1)}_{\text{FT}}\right] - \eta_{\text{FT}} \left.\frac{\partial L_{\text{FT}}}{\partial \left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}\right|_{\boldsymbol{\theta}^{(k-1)}_{\text{FT}}, \boldsymbol{\psi}^{(k-1)}_{\text{FT}}} \qquad k = 1, \ldots, K \\
\boldsymbol{\theta}^*_{\text{FT}}, \boldsymbol{\psi}^*_{\text{FT}} &\approx \boldsymbol{\theta}^{(K)}_{\text{FT}}, \boldsymbol{\psi}^{(K)}_{\text{FT}},
\end{aligned}
\tag{4.4}
$$

and compute the gradient $\left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}^*_{\text{PT}}(\phi')}$ by differentiating through this optimization.[3]

It is also possible to freeze the feature extractor parameters $\boldsymbol{\theta}_{\text{FT}}$ and update only the head parameters $\boldsymbol{\psi}_{\text{FT}}$ during truncated FT and to obtain meta-parameter gradients. This resembles *linear evaluation*, where a linear classifier is trained on top of fixed, pre-trained feature extractors [134, 5, 177].

[3] While Equation 4.4 uses standard gradient descent, we could use other differentiable optimizers (e.g., Adam).

Together, these two approximations allow for efficient computation of meta-parameter gradients.

4.4.2 Our Algorithm and Practical Considerations

By leveraging the above approximations, we obtain Algorithm 2 to optimize meta-parameters $\boldsymbol{\phi}$ online during PT & FT of the base model. Note that Alg_{PT} is explicitly written out as a sequence of gradient updates (lines 4-6 in Algorithm 2). By using data from the FT set during pre-training (to optimize meta-parameters), our algorithm couples the PT and FT problems, helping to mitigate issues with negative transfer. There are certain important implementation details and practical considerations for this algorithm, which we now discuss. Further details are given in Appendix C.2.

Algorithm 2 Gradient-based algorithm to learn meta-parameters. Notation defined in Table C.1. Note that vector-Jacobian products (VJPs) can be efficiently computed by standard autodifferentiation.

1: Initialize PT parameters $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})}, \boldsymbol{\psi}_{\text{PT}}^{(\text{init})}, \boldsymbol{\psi}_{\text{FT}}^{(\text{init})}$ and meta-parameters $\boldsymbol{\phi}^{(0)}$
2: **for** $n = 1, \ldots, N$ iterations **do**
3: Initialize $\boldsymbol{\theta}_{\text{PT}}^{(0)} = \boldsymbol{\theta}_{\text{PT}}^{(\text{init})}$ and $\boldsymbol{\psi}_{\text{PT}}^{(0)} = \boldsymbol{\psi}_{\text{PT}}^{(\text{init})}$.
4: **for** $p = 1, \ldots, P$ PT iterations **do**
5: $\left[\boldsymbol{\theta}_{\text{PT}}^{(p)}, \boldsymbol{\psi}_{\text{PT}}^{(p)}\right] = \left[\boldsymbol{\theta}_{\text{PT}}^{(p-1)}, \boldsymbol{\psi}_{\text{PT}}^{(p-1)}\right] - \eta_{\text{PT}} \left.\frac{\partial L_{\text{PT}}}{\partial \left[\boldsymbol{\theta}_{\text{PT}}, \boldsymbol{\psi}_{\text{PT}}\right]}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(p-1)}, \boldsymbol{\psi}_{\text{PT}}^{(p-1)}}$ `# Unrolled step of` Alg_{PT}
6: **end for**
7: **if** $n < N_{\text{warmup}}$ **then**
8: Update PT initialization by setting: $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})} = \boldsymbol{\theta}_{\text{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\text{PT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{PT}}^{(P)}$
9: Skip meta-parameter update and continue
10: **end if**
11: Initialize FT parameters $\boldsymbol{\psi}_{\text{FT}}^{(0)} = \boldsymbol{\psi}_{\text{FT}}^{(\text{init})}$ and $\boldsymbol{\theta}_{\text{FT}}^{(0)} = \texttt{copy}(\boldsymbol{\theta}_{\text{PT}}^{(P)})$.
12: Approximate $\boldsymbol{\theta}_{\text{FT}}^*, \boldsymbol{\psi}_{\text{FT}}^*$ using (4.4), with $\mathcal{D}_{\text{FT}}^{(\text{tr})}$.
13: Compute $g_1 = \left.\frac{\partial L_{\text{FT}}}{\partial \left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}\right|_{\boldsymbol{\theta}_{\text{FT}}^*, \boldsymbol{\psi}_{\text{FT}}^*}$, using $\mathcal{D}_{\text{FT}}^{(\text{val})}$. `# FT Loss gradient`
14: Compute VJP $g_2 = g_1 \left.\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\right|_{\boldsymbol{\theta}_{\text{PT}}^{(P)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}}$ using the unrolled learning step from line 12, and $\mathcal{D}_{\text{FT}}^{(\text{tr})}$.
15: Approximate VJP $\left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}} = g_2 \left.\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ using IFT (4.3).
16: $\boldsymbol{\phi}^{(n)} = \boldsymbol{\phi}^{(n-1)} - \eta_{\text{V}} \left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ `# Update meta-parameters`
17: Update PT initialization by setting: $\boldsymbol{\theta}_{\text{PT}}^{(\text{init})} = \boldsymbol{\theta}_{\text{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\text{PT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{PT}}^{(P)}$.
18: Update FT initialization by setting: $\boldsymbol{\psi}_{\text{FT}}^{(\text{init})} = \boldsymbol{\psi}_{\text{FT}}^*$.
19: **end for**

(1) Notational clarification: vector concatenation: We use the notation: $\left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]$ to represent concatenation of the two vectors $\boldsymbol{\theta}_{\text{FT}}$ and $\boldsymbol{\psi}_{\text{FT}}$. The output of Alg_{FT} contains two parameter vectors, and these are implicitly concatenated to make sure that dimensionalities agree in the algorithm.

(2) Access to $\mathcal{D}_{\text{FT}}$ and generalizing to new FT tasks: Solving the meta-PT problem requires availability of the model $f_{\bullet}$, the PT data $\mathcal{D}_{\text{PT}}$, and the FT data $\mathcal{D}_{\text{FT}}$. In this work, we assume availability of the model and PT dataset, but since assuming access to the complete FT dataset at meta-PT time is more restrictive, we study two scenarios: *Full FT Access*, where all FT data that we expect to encounter is available at meta-PT time, and *Partial FT Access*, where the FT data available at meta-PT time is only a sample from the full distribution of FT data.

Full FT Access occurs in settings like semi-supervised learning, where we are given a large unlabelled PT dataset and a small labelled FT dataset and our goal is to achieve the best possible performance by leveraging these two fixed datasets [199, 206, 78, 76, 30, 31].

Partial FT Access occurs when our goal is to learn transferable representations: at meta-PT time, we might have limited knowledge of FT tasks or data. In evaluating this scenario, we examine generalizability to new FT tasks, given only small amounts of FT data/task availability at meta-PT time, demonstrating that even very limited FT access can be sufficient for effective meta-parameter optimization [46, 119, 154, 84].

(3) $\mathcal{D}_{\text{FT}}$ splits: In practice, we have access to finite datasets and use minibatches, rather than true data-generating processes. Following standard convention, we split $\mathcal{D}_{\text{FT}}$ into two subsets for meta-learning: $\mathcal{D}_{\text{FT}}^{(\text{tr})}$ and $\mathcal{D}_{\text{FT}}^{(\text{val})}$ (independent of any held-out $\mathcal{D}_{\text{FT}}$ testing split), and define the FT data available at meta-PT time as $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}^{(\text{tr})} \cup \mathcal{D}_{\text{FT}}^{(\text{val})}$. We use $\mathcal{D}_{\text{FT}}^{(\text{tr})}$ for the computation of $\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}\Big|_{\boldsymbol{\theta}_{\text{PT}}^{(P)}, \boldsymbol{\psi}_{\text{FT}}^{(0)}}$ and $\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}\Big|_{\boldsymbol{\phi}^{(n-1)}}$ and $\mathcal{D}_{\text{FT}}^{(\text{val})}$ for the computation of $\frac{\partial L_{\text{FT}}}{\partial \left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}\Big|_{\boldsymbol{\theta}_{\text{FT}}^{*}, \boldsymbol{\psi}_{\text{FT}}^{*}}$ in Algorithm 2.

(4) Online updates: Given that PT phases often involve long optimization horizons, for computational efficiency, we update $\boldsymbol{\theta}_{\text{PT}}$ and $\boldsymbol{\psi}_{\text{PT}}$ online rather than re-initializing them at every meta-iteration (see Algorithm 2). FT phases are often shorter so we could in theory re-initialize $\boldsymbol{\psi}_{\text{FT}}$ at each meta-iteration. However, for computational and memory efficiency, in our experiments, we also optimize these parameters online. For $\boldsymbol{\psi}_{\text{FT}}$, this makes each meta-iteration resemble a "warm-start" to the FT problem. Algorithm 2, reflects this implementational detail.

Note that prior work [198] has suggested that online optimization of certain hyperparameters (e.g., learning rates) using short horizons may yield suboptimal solutions. We comment on this in Appendix C.2, study this effect for our algorithm in synthetic experiments in Appendix C.4, and in real-world experiments on self-supervised learning in Appendix C.6, revealing it is not a significant concern.

(5) Warmup iterations: We can optionally include *warmup iterations* where we optimize the PT parameters and do not perform updates to the meta-parameters. This is to ensure that the PT parameters are a reasonable approximation of $\mathcal{L}_{\text{PT}}$-optimal parameters. The description in Algorithm 2 reflects this detail, with the N_{warmup} reflecting the number of warmup iterations.

(6) Computational tractability: Our method can scale to large encoder models and high-dimensional meta-parameters, despite the complexity of the two-stage PT & FT process. This is because: (i) meta-parameters are optimized jointly with the base model parameters; (ii) using the IFT to obtain gradients has similar time and memory complexity to one iteration of training [113]; (iii) the FT best response Jacobian can be approximated efficiently using a small number of unrolled optimization steps K, and by only unrolling the FT head of the network. In our real-world experiments (Sections 4.6 and 4.7), meta-parameterized PT has less than twice the time cost of standard PT. Further details on time and memory cost are provided in Appendices C.5 and C.6.

(7) Setting optimizer parameters: Learning rates and momentum values can impact the efficacy of the algorithm. A discussion on how to set them in practice is provided in Appendix C.3.

4.4.3 Connections to Other Algorithms in this Thesis

The algorithm presented here has two clear connections to the other nested optimization work in this thesis.

Firstly, computing gradients wrt the fine-tuning initialization point (i.e., the pre-trained parameters) via differentiating through optimization, mirrors what is done in the MAML algorithm [56] in Chapter 2 for few-shot learning. Our new algorithms in Chapter 2, ANIL and NIL, also utilize this same strategy during the meta-training phase. Recall that we employ differentiating through optimization for this gradient term rather than implicit differentiation since the initialization point does not directly affect the optimized loss, as discussed in Section 4.4.1.

Secondly, the use of implicit differentiation techniques (through the implicit function theorem) to approximately compute gradients through long optimization trajectories is used in Chapter 3 when optimizing the parameters of our flexible data augmentation pipeline, TaskAug. Implicit differentiation is a sensible choice in both cases since the parameters we wish to optimize (pre-training meta-parameters in this chapter, and augmentation hyperparameters in Chapter 3) directly impact the optimal value of the optimized model parameters.

Overall, this algorithm can be viewed as extending traditional one-stage gradient-based hyperparameter optimization strategies to the two-stage pre-training & fine-tuning problem.

4.5 Synthetic Experiments

We validate that our algorithm recovers optimal low and high dimensional meta-parameters in two synthetic MNIST experiments with *Full FT Access*. Further details and results are provided in Appendix C.4, including a study of how our method

performs comparably to differentiating exactly through the entire learning process of PT & FT, without approximations.

First, we optimize low dimensional meta-parameters characterizing a data augmentation scheme. We tune a 1-D meta-parameter ϕ representing the mean of a normal distribution $\mathcal{N}(\phi, 1^2)$ from which we sample rotation augmentations to apply to PT images. FT images undergo rotations from a normal distribution $\mathcal{N}(\mu_{\text{FT}}, 1^2)$ with $\mu_{\text{FT}} = 90°$; we therefore expect that ϕ should converge to near μ_{FT}. Using Algorithm 2 to optimize ϕ, we find that the mean error in the optimized meta-parameter over 10 different initializations is small: $7.2 \pm 1.5°$, indicating efficacy of the algorithm.

Next, we consider learning high dimensional meta-parameters that characterize a PT per-example weighting scheme. The PT dataset contains some examples that have noisy labels, and FT examples all have clean labels. The meta-parameters are the parameters of a neural network that assigns importance weights to each PT example, which is used to weight the loss on that example during PT. We use Algorithm 2 again to optimize ϕ, over 10 random initializations, finding the ratio of assigned importance weights between clean label PT examples and noisy label PT examples is greater than 10^2. This is expected since the noisy label classes may worsen the quality of the PT model and so should be down-weighted.

4.6 Meta-Parameterized Multitask Pre-Training for Graph Neural Networks

We consider optimizing PT task weights for a multitask PT & FT problem of predicting the presence of protein functions (multitask binary classification) given graph-structured biological data as input.

We have two experimental goals: first, in the *Full FT Access* setting, where methods are given access to all FT data at PT time, we evaluate whether optimizing task weighting meta-parameters can improve predictive performance on the FT tasks. Second, motivated by how in typical transfer learning problems, new tasks or labels not

available at PT time may become available at FT time, we study the *Partial FT Access* setting, investigating how our method performs when it only sees *limited* FT tasks at PT time.

4.6.1 Problem Setup

Dataset and Task. We consider the transfer learning benchmark introduced in [84], where the prediction problem at both PT and FT is multitask binary classification: predicting the presence/absence of specific protein functions (y) given a Protein-Protein Interaction (PPI) network as input (represented as a graph $\boldsymbol{x}$). The PT dataset has pairs $\mathcal{D}_{\text{PT}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{PT}}|}$, where $y \in \{0,1\}^{5000}$ characterizes the presence/absence of 5000 particular protein functions. The FT dataset has pairs $\mathcal{D}_{\text{FT}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{FT}}|}$, where $y \in \{0,1\}^{40}$ now characterizes the presence/absence of 40 different protein functions. Further dataset details in Appendix C.5.

Meta-Parameterized Multitask PT. To define a meta-parameterized PT scheme, we let meta-parameters $\phi \in \mathbb{R}^{5000}$ be weights for the binary PT tasks. Then, we define a PT loss incorporating the weights:

$$\mathcal{L}_{\text{PT}} = \frac{1}{5000} \sum_{i=1}^{5000} 2\ \sigma(\phi_i)\ \mathcal{L}_{\text{CE}}(f_{\text{PT}}(\boldsymbol{x}; \boldsymbol{\theta}_{\text{PT}}, \boldsymbol{\psi}_{\text{PT}})_i, y_i),$$

with i indexing the tasks, $\sigma(\cdot)$ representing the sigmoid function (to ensure non-negativity and clamp the range of the weights), and $\mathcal{L}_{\text{CE}}$ denoting the binary cross-entropy loss. With this loss defined, we use Algorithm 2 (with $P = 10$ PT steps and $K = 1$ truncated FT steps) to jointly learn ϕ and the feature extractor parameters $\boldsymbol{\theta}_{\text{PT}}$. For computational efficiency, we only update the FT head when computing the FT best response Jacobian and keep the feature extractor of the model fixed. We use the training and validation splits of the FT dataset $\mathcal{D}_{\text{FT}}$ proposed by the dataset creators [84] for computing the relevant gradient terms.

Baselines. Motivated by our goals, we compare with the following PT baselines:

- **No PT:** Do not perform PT (i.e., feature extractor parameters are randomly initialized).

- **Graph Supervised PT:** As explored in prior work on this domain [84], perform multitask supervised PT with $\mathcal{D}_{\text{PT}}$. This corresponds to setting all task weights to 1: $\phi_i = 1, i = 1, \ldots, 5000$.

- **CoTrain:** A common baseline that makes use of the FT data available during PT [201] (like meta-parameterized PT). We PT a model with $5000 + 40$ outputs (covering the space of PT and FT labels) jointly on both $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$. We do so by alternating gradient updates on batches sampled from each dataset in turn. Further details are in Appendix C.5.

- **CoTrain + PCGrad:** An extension of CoTrain, where we leverage the method PCGrad [205] to perform gradient projection and prevent destructive gradient interference between updates from $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$. Further details and variants we tried are in Appendix C.5.

Experimental Details. We use a standardized setup to facilitate comparisons. Following [84], all methods use the Graph Isomorphism Network architecture [200], undergo PT for 100 epochs, and FT for 50 epochs, over 5 random seeds. At FT time, we employ early stopping based on validation set performance (so models are trained for a maximum of 50 epochs). During FT, we initialize a new FT network head and either FT the whole network or freeze the PT feature extractor and learn the FT head alone (Linear Evaluation [134]). We report results for the strategy that performed best (full results in the appendix). We consider two experimental scenarios: (1) *Full FT Access:* Provide methods full access to $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$ at PT time ($\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$) and evaluate on the full set of 40 FT tasks; (2) *Partial FT Access:* Limit the number of FT tasks seen at PT time, by letting $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ include only 30 of the 40 FT tasks. At FT time, models are fine-tuned on the held-out 10 tasks not in $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$. We use a 4-fold approach where we leave out 10 of the 40 FT tasks in turn, and examine performance across these 10 held-out tasks, over the folds.

Method	AUC ($\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$)	AUC ($\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ excludes tasks)
No PT	66.6 ± 0.7	65.8 ± 2.5
Graph Supervised PT	74.7 ± 0.1	74.8 ± 1.8
CoTrain	70.2 ± 0.3	69.3 ± 1.8
CoTrain + PCGrad	69.4 ± 0.2	68.1 ± 2.3
Meta-Parameterized PT	**78.6 ± 0.1**	**77.0 ± 1.3**

Table 4.1: **Meta-Parameterized PT improves predictive performance over baselines.** Table showing mean AUC and standard error for two evaluation settings. When provided all FT data at PT time (first results column), meta-parameterized PT significantly improves predictive performance. In a more challenging setting when $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ excludes FT tasks (10 of the 40 available tasks are held-out), evaluating mean AUC/standard error across four folds with each set of 10 FT tasks held out in turn, meta-parameterized PT again obtains the best performance: it is effective even with partial information about the downstream FT tasks.

4.6.2 Results

Key Findings. By optimizing PT task weights, meta-parameterized multitask PT improves performance on the FT problem of predicting presence/absence of protein functions given a protein-protein interaction graph as input. Performance improvements are also seen when generalizing to new FT tasks (protein functions), unseen at meta-PT time.

Commentary on results. Table 4.1 presents quantitative results for the two experimental settings described. For the No PT and Graph Supervised PT baselines, we re-implement the methods from [84], obtaining improved results (full comparison in Appendix Table C.3). In both full and partial FT access settings, meta-parameterized PT improves on other methods, indicating that optimizing meta-parameters can improve predictive performance generally, and be effective even when new, related tasks are considered at evaluation time. Interestingly, we observe that CoTrain and CoTrain + PCGrad obtain relatively poor performance compared to other baselines; this could be because the methods overfit to the FT data during PT. Further analysis of this is presented in Appendix C.5.

Further experiments. In Appendix C.5, we study another partial FT access scenario with smaller $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$, setting $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right| = 0.5\,|\mathcal{D}_{\text{FT}}|$, and find that meta-parameterized PT again outperforms other methods. (Table C.5). We also examine another meta-parameter learning baseline, namely a version of CoTrain where we optimize task weights using a traditional hyperparameter optimization algorithm [113] jointly with the main model. We find that our method outperforms this baseline also (Table C.3).

Analysis of learned structures. In Appendix C.5, we conduct further analysis and study the effect of various PT strategies on the pre-trained representations (Figure C-2), finding intuitive patterns of similarity between different methods. We also examine the learned task weights (Figure C-3), and examine performance on a per-FT task basis with/without meta-parameterized PT (Figure C-4), finding little evidence of negative transfer.

4.7 Meta-Parameterized SimCLR for Semi-Supervised Learning with ECGs

We now explore a second real-world application of our method: optimizing a data augmentation policy for self-supervised PT with SimCLR [30, 31] on electrocardiograms (ECGs). SimCLR is a popular self-supervised PT method that leverages data augmentations to define a contrastive PT objective (details in Appendix C.6.1). The choice/strength of the augmentations used significantly impacts the effectiveness of the algorithm [30]. In settings where relevant augmentations are known (e.g., natural images), SimCLR is readily applicable; however, for ECGs, effective augmentations are less clear, motivating the use of our algorithm to optimize the augmentation pipeline. This question of finding effective data augmentations for ECG data is also the focus of Chapter 3 of this thesis; however, in that contribution, we explored fully supervised learning, whereas here, we study a self-supervised PT scenario where we have unlabelled data.

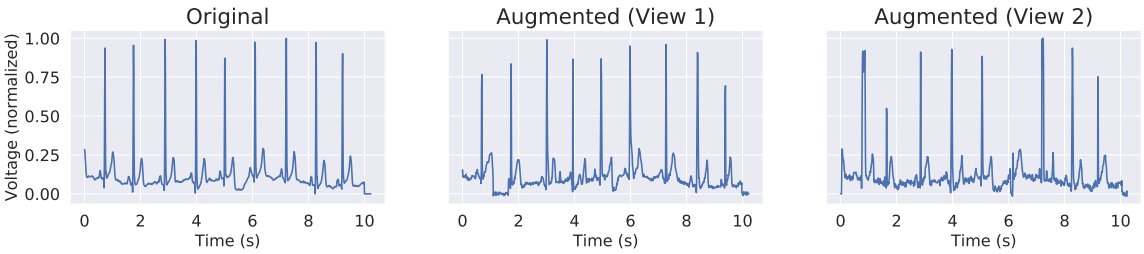


Figure 4-2: A single lead (or channel) of the 12 lead ECG signal and two augmented views (following cropping, jittering, and temporal warping) that are used in contrastive learning.

We have two experimental goals. Firstly, we examine the typical semi-supervised learning setting of *Full FT Access*: we explore whether optimizing the augmentations in SimCLR PT can improve performance on the supervised FT task of detecting pathologies from ECGs, given access to all FT data at meta-PT time. Secondly, to study the data efficiency of our method, we consider the *Partial FT Access* setting and explore performance given access to limited FT data at meta-PT time.

4.7.1 Problem Setup

Dataset and Task. We construct a semi-supervised learning (SSL) problem using PTB-XL [191, 61], an open-source dataset of electrocardiogram (ECG) data. We also use this dataset in our experimental evaluation in Chapter 3.

Let the model input at both PT and FT time be denoted by $\boldsymbol{x}$, which represents a 12-lead (or channel) ECG sampled at 100 Hz for 10 seconds resulting in a 1000×12 signal. Our goal is to pre-train a model f_{PT} on an unlabeled PT dataset of ECGs $\mathcal{D}_{\text{PT}} = \{\boldsymbol{x}_i\}_{i=1}^{|\mathcal{D}_{\text{PT}}|}$ using SimCLR PT [30], and then fine-tune it on the labeled FT dataset $\mathcal{D}_{\text{FT}} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{|\mathcal{D}_{\text{FT}}|}$, where the FT labels $y \in \{0,1\}^5$ encode whether the signal contains certain features indicative of particular diseases/pathologies. Further dataset details in Appendix C.6.

ECG Data Augmentations. To augment each ECG for SimCLR, we apply three transformations in turn (based on prior work in time series augmentation [87, 197]):

1. **Random cropping:** A randomly selected portion of the signal is zeroed out.

2. **Random jittering:** IID Gaussian noise is added to the signal.

3. **Random temporal warping:** The signal is warped with a random, diffeomorphic temporal transformation. This is formed by sampling from a zero mean, fixed variance Gaussian at each temporal location in the signal to obtain a velocity field, and then integrating and smoothing (following [9, 10]) to generate a temporal displacement field, which is applied to the signal.

An example of applying these augmentations is shown in Figure 4-2.

Meta-Parameterized SimCLR. To construct a meta-parameterized SimCLR PT scheme, we instantiate meta-parameters $\boldsymbol{\phi}$ as the weights of a neural network $w(\boldsymbol{x}; \boldsymbol{\phi})$ that takes in an input signal and outputs the warp strength: the variance of the Gaussian that is used to obtain the velocity field for temporal warping. This parameterization permits signals to be warped more/less aggressively depending on their individual structure. With this definition, the SimCLR PT loss is directly a function of the meta-parameters, and we can use Algorithm 2 (with $P = 10$ PT steps and $K = 1$ truncated FT steps) to jointly learn ϕ and the feature extractor parameters $\boldsymbol{\theta}_{\mathrm{PT}}$. For computational efficiency, we only update the FT head when computing the FT best response Jacobian and keep the feature extractor of the model fixed. We use the training and validation splits of the FT dataset $\mathcal{D}_{\mathrm{FT}}$ proposed by the dataset creators [191] for computing the relevant gradient terms.

Baselines. Our experimental goals suggest the following PT baselines:

- **No PT:** Do not perform PT (i.e., feature extractor parameters are randomly initialized).

- **SimCLR:** Pre-train a model using SimCLR with the above three augmentations *without* learning per-example temporal warping strengths.

Experimental Details. We standardize the experimental setup to facilitate comparisons. All methods use a 1D CNN based on a ResNet-18 [77] architecture. The

	Test AUC at different FT dataset sizes $\lvert\mathcal{D}_{\text{FT}}\rvert$				
FT dataset size $\lvert\mathcal{D}_{\text{FT}}\rvert$	100	250	500	1000	2500
No PT	71.5 ± 0.7	76.1 ± 0.3	78.7 ± 0.3	82.0 ± 0.2	84.5 ± 0.2
SimCLR	74.6 ± 0.4	76.5 ± 0.3	79.8 ± 0.3	82.2 ± 0.3	85.8 ± 0.1
Meta-Parameterized SimCLR	**76.1 ± 0.5**	**77.8 ± 0.4**	**81.7 ± 0.2**	**84.0 ± 0.3**	**86.7 ± 0.1**

Table 4.2: **Meta-Parameterized SimCLR obtains improved semi-supervised learning performance.** Table showing mean AUC/standard error over seeds across 5 FT binary classification tasks for baselines and meta-parameterized SimCLR at different sizes of $\mathcal{D}_{\text{FT}}$, with $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$. We observe improvements in performance with meta-parameterized SimCLR, which optimizes the augmentation pipeline.

temporal warping network $w(\boldsymbol{x}; \phi)$ is a four layer 1D CNN. SimCLR PT takes place for 50 epochs for all methods (further PT did not result in improved performance after FT), over three PT seeds. At evaluation time, for all methods, we initialize a new FT network head over the PT network feature extractor and FT the whole network for up to 200 epochs (with early stopping based on validation set AUC), over five FT seeds. We consider two experimental settings: (1) *Full FT Access*, standard SSL: consider different sizes of the labelled FT dataset $\mathcal{D}_{\text{FT}}$ and make all the FT data available at meta-PT time, $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$; and (2) *Partial FT Access*, examining data efficiency of our algorithm: SSL when only limited FT data is available at meta-PT time: $\mathcal{D}_{\text{FT}}^{(\text{Meta})} \subseteq \mathcal{D}_{\text{FT}}$. We evaluate performance across the 5 binary classification tasks in both settings. Further details are provided in Appendix C.6.

4.7.2 Results

Key Findings. By optimizing the data augmentation policy used in SimCLR PT, meta-parameterized SimCLR improves performance on the FT problem of detecting pathologies from ECG data. Even a small amount of FT data provided at meta-PT time can lead to improved FT performance.

Commentary on results. Table 4.2 shows results for the *Full FT Access* setting, $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$: mean AUC/standard error over seeds across the 5 FT binary classification tasks at different sizes of $\mathcal{D}_{\text{FT}}$. We observe that meta-parameterized SimCLR

improves on other baselines in all settings. Note that these gains are obtained with simple augmentation policies; our method might yield further improvements if applied to policies with more scope to specialize the augmentations. Leveraging some of the insights from the augmentations developed in Chapter 3, and generalizing them to become task agnostic, might be a promising direction of exploration.

Next, we consider the *Partial FT Access* scenario where $\mathcal{D}_{\text{FT}}^{(\text{Meta})} \subseteq \mathcal{D}_{\text{FT}}$, which is relevant when we only have a small amount of FT data at meta-PT time. Fixing $|\mathcal{D}_{\text{FT}}| = 500$, we find that with $|\mathcal{D}_{\text{FT}}^{(\text{Meta})}|$ as small as 50, we obtain test AUC of 81.3 ± 0.5, compared to 79.8 ± 0.3 with no optimization of augmentations: this shows that even small $|\mathcal{D}_{\text{FT}}^{(\text{Meta})}|$ appear to be sufficient for meta-parameter learning. Further results showing performance curves varying $|\mathcal{D}_{\text{FT}}^{(\text{Meta})}|$ are in Appendix C.6.

Further experiments. In Appendix C.6, we study other aspects of our method on this domain, including: (1) Exploring different values of K, the number of FT steps differentiated through when obtaining meta-parameter gradients; and (2) Examining a meta-parameter learning baseline where augmentations are optimized for supervised learning, using the method in [113], and then applied to semi-supervised learning (to compare how optimizing augmentations for supervised learning compares to optimizing them for semi-supervised learning). We find that our method is not very sensitive to the value of K (provided $K > 0$), and that it outperforms this additional baseline.

4.8 Scope and Limitations

Intended use-case. Our gradient-based algorithm applies in situations where we want to optimize PT hyperparameters, or *meta-parameters*, and have access to a model, PT data, and FT data. We demonstrated that even limited FT data availability can be sufficient to guide meta-parameter learning; however, our method would not apply when no FT data at all is available at meta-PT time, or if the model and/or PT data were not available at meta-PT time. In general, our method is most useful when the PT process is long and/or the meta-parameter space is high-dimensional,

and so it is not feasible to use methods such as random/grid search and Bayesian optimization (which involve multiple runs of PT and FT). In a setting like that described in Chapter 5, where self-supervised PT and FT are relatively quick to run and there meta-parameters are low-dimensional, a grid search is more efficient than using this nested optimization algorithm and optimizing its learning rate hyperparameters.

Requirements on meta-parameters. Our algorithm requires meta-parameters to be differentiable, and cannot directly be used to optimize meta-parameters that do not affect the PT optimization landscape (e.g., PT learning rates). Similarly to what was discussed in Chapter 3, the initialization point of meta-parameters is important. If meta-parameters are initialized poorly, it will be difficult for our algorithm to recover from that setting and be effective. One strategy to address this in practice could be to conduct a small random search to find a promising meta-parameter initialization, and then applying our algorithm afterwards.

4.9 Conclusion

In this chapter, we investigated the popular pre-training (PT) followed by fine-tuning (FT) paradigm. In this setting, our goal is to train a model on a task of interest that only has limited labelled data (the FT task). To address this data scarcity challenge, we employ a two stage training process: we first use a large related dataset (the PT dataset) to learn an effective initialization for the model parameters, and then we further optimize these model parameters for the task of interest. This two-stage training can result in improved performance compared to training directly on the downstream task.

We began by identifying that the PT & FT paradigm, although effective, introduces high-dimensional design choices, or *meta-parameters* during the PT phase. To pre-train a model effectively, these meta-parameters must be optimized well. We outlined a new PT algorithm that jointly optimizes meta-parameters and model parameters by efficiently approximating gradients through the two-stage PT & FT optimiza-

tion. In experiments, we found that using our PT algorithm improves performance over baselines (following FT) on two real-world tasks: transfer learning on graph structured biological data [84], and semi-supervised learning on electrocardiograms [191].

Chapter 5

Self-Supervised Learning for Complex Clinical Time-Series

5.1 Introduction

In the previous chapter, we studied the data-efficient learning paradigm of pre-training (PT) followed by fine-tuning (FT), and in particular explored how to pre-train models more effectively using tools from nested optimization. In this chapter, we continue our investigation of this paradigm, but instead of exploring how to improve PT algorithms more generally (the goal of Chapter 4), we focus on developing a new PT algorithm for complex, unlabelled clinical time series data, using tools from self-supervised learning (SSL).

There have been many efforts in the literature to develop self-supervised PT algorithms for clinical time series data, with the goal of using these pre-trained models for various downstream predictive tasks with small labelled datasets [120, 195, 182, 204, 179]. In particular, developing PT algorithms for clinical time series data from the intensive care unit (ICU) has been well-studied, since patients in the ICU are closely monitored and consequently generate a profusion of unlabeled time series data, which contains significant physiological information about a patient's state and progression over time [93].

Although prior works develop effective strategies to model clinical time series data,

they typically focus only on unimodal time series, such as a sequence of structured features alone, or an individual high dimensional physiological signal. In reality however, data originating from a patient's encounter is significantly more complex, containing multimodal data recorded at regular intervals. As an example, a given patient might have two very different types of data recorded hourly: (1) high-frequency physiological signals (e.g., an electrocardiogram recorded at 240 Hz); and (2) structured data from labs and vitals signs. These modalities provide complementary information about a patient's physiological state. Extending existing SSL methods to operate on these time series is challenging, since they do not deal with sequences of high-dimensional data, and do not contend with the multimodal data stream.

In this chapter, we take steps towards addressing this gap and outline an approach for self-supervised pre-training on these complex clinical time series. We propose a SSL strategy where we jointly optimize two SSL losses to better capture structure in the data. Our contributions are as follows:

1. We formalize the problem of self-supervised learning (SSL) on *trajectories*, our abstraction of a multimodal time series that contains complex, high-dimensional data recorded at each timestep in the sequence.

2. We outline a new SSL method, *Sequential Multi-Dimensional Self-Supervised Learning* (SMD SSL), for trajectories. Motivated by the structure of trajectories, SMD SSL incorporates two losses: (1) a component SSL loss on the level of individual high dimensional data points in the sequence; and (2) a global SSL loss on the level of the overall sequence. SMD SSL can be instantiated with contrastive losses, as in SimCLR [30] or non-contrastive losses, as in VICReg [14]. This is beneficial since different loss functions may be effective in different applications.

3. We evaluate SMD SSL on two real-world clinical datasets where the time series contains sequences of (1) high-frequency electrocardiograms and (2) structured data from labs and vitals signs. On both datasets and on two downstream tasks — (1) detecting elevated pulmonary pressures and (2) predicting 24 hour

mortality — we find SMD SSL improves performance over baselines. In several settings, we observe performance boosts using both SimCLR and VICReg objective functions.

5.2 Related Work

Self-supervised learning (SSL). SSL methods are used to pre-train models and/or learn generalizable representations using unlabeled data. Many existing methods take either a multiview perspective [30, 33, 14, 76, 33, 67] or an autoregressive denoising approach [188, 75]. Here, we focus on multiview approaches since they have been effective in improving predictive performance on clinical tasks [195, 182].

SSL for clinical data. Existing applications of multiview SSL to medical data have been focused either on physiological signals, such as electrocardiograms (ECGs) [35, 98, 63, 47, 133], on sequences of tabular data, such as laboratory tests [204, 107], or on medical imaging [156]. In contrast, we consider SSL on time series where individual timesteps contain both high-dimensional data (such as ECGs) and structured features. Prior studies exploring SSL on multimodal medical data typically infer one modality of data from the other at test time, such as predicting radiologist comments from chest X-ray images [180], rather than modeling sequences of multimodal data where the modalities present non-overlapping sources of information, as we do here.

Multilevel SSL loss functions. Our method uses a two-level loss function that is motivated by the complex structure of the data stream we consider: sequences in which individual elements are themselves high-dimensional. A related approach in computer vision, VICRegL [15], applies multilevel self-supervision to images where patch-level similarity is defined using spatial transformations. In contrast to VICRegL, which formulates a component level loss using patches, our method formulates a component level loss on entire signals, and so operates on a different level of abstraction. Another recent method decouples local and global representation learning for a single time series [183]. This work also operates on a different level of abstraction

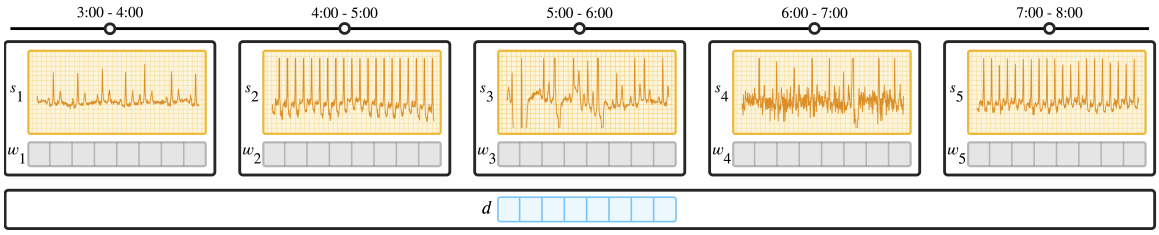


Figure 5-1: **An example of a multimodal clinical time-series or trajectory'** The trajectory τ contains an static vector d consisting of measurements that remain constant over the time period, and a sequence of high-dimensional physiological signals s_t and structured data w_t measured at each time step. Here, each time step is a 1 hour window.

than ours, since it does not consider sequences of time series. Finally, [156] demonstrate that multiscale SSL for neuroimaging offers improvement on downstream tasks. However, their techniques are tailored for neuroimaging and do not generalize readily to the data we consider.

5.3 Methods

In this section, we describe our approach for self-supervised learning (SSL) on multimodal clinical time series: *Sequential Multi-Dimensional SSL*. We first outline our problem setup, describing the multimodal data stream that we consider. We then detail our SSL scheme, specifying the loss functions used to learn representations on both an individual timestep (component) level and on a overall sequence (global) level. We conclude with a discussion of other applications of the method.

5.3.1 Problem Setup

Defining *trajectories*. We use the term *trajectory* to refer to a sequence of physiological signals and structured data collected over time for a patient. This definition is motivated by an important use-case in cardiovascular medicine, where patients may be monitored with telemetry devices that regularly record physiological waveforms in addition to having lab tests and vitals signs periodically measured. The concept of a trajectory could be expanded to include other information, such as imaging or medications, depending on the context.

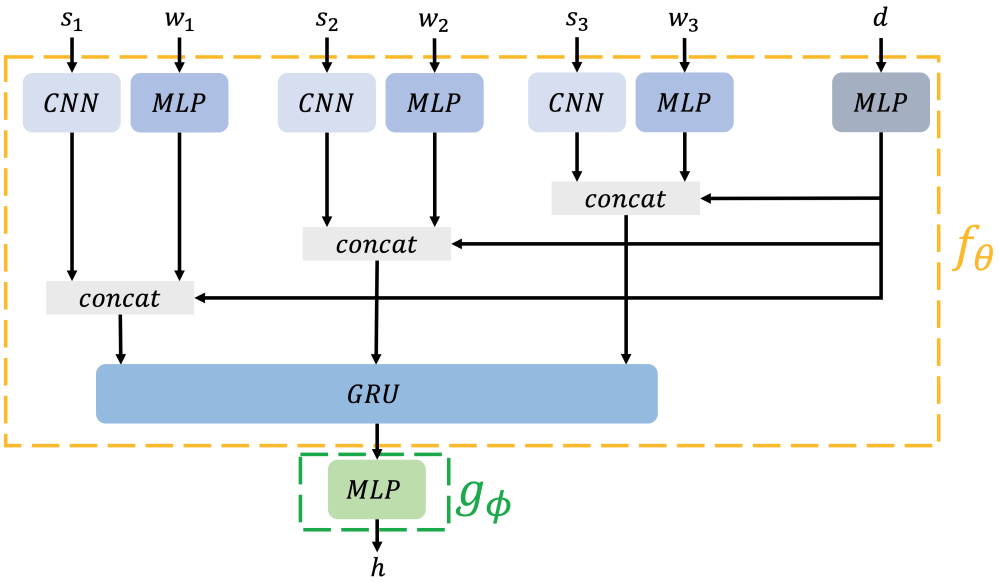


Figure 5-2: **Model architecture used in our experiments.** We show the architecture used to model trajectories in a scenario where the input trajectory has 3 timesteps.

Formally, a trajectory τ of length T has the structure:

$$\tau = (d, \{(w_t, s_t)\}_{t=1}^{T}).$$

Here, $d \in \mathbb{R}^L$ represents a set of static features that do not change over the trajectory (such as demographic information or infrequently measured lab values). The sequence

$$\{(w_t, s_t)\}_{t=1}^{T}$$

contains a vector of structured data $w \in \mathbb{R}^M$, and a high-dimensional signal $s \in \mathbb{R}^{C \times P}$, where C is the number of signal channels and P is the number of samples in the signal (typically on the order of a few thousand). Note that under this definition, a trajectory is a time-series where the data at each time step also contains a time series (the signal). A visualization of a trajectory is shown in Figure 5-1.

Trajectory neural network. Trajectories are mapped into vector representations using a neural network f_θ with three components (Figure 5-2): (1) a static and structured features encoder $f_\theta^{w,d}$; (2) a signals encoder f_θ^s; and (3) a sequence module f_θ^τ.

At each timestep t, the modalities are embedded and concatenated into timestep representations:

$$z_t = \text{concat}\left(f_\theta^{w,d}(w_t, d), f_\theta^s(s_t)\right).$$

The sequence module maps these representations into a vector:

$$z = f_\theta^\tau(z_1, z_2, \ldots, z_T).$$

In a supervised setting, a classifier c_ψ maps z to a predicted label $\hat{y}$.

Also shown in Figure 5-2 is the projection head g_ϕ, which is used during self-supervised learning (as described in Section 5.3.2). More implementation details about the architecture are presented in Appendix D.2.2.

Trajectory self-supervised learning (SSL). Inspired by recent work in SSL [30, 31, 14, 208], we consider a two-stage learning problem: pre-training (PT) followed by fine-tuning (FT). We first pre-train the model f_θ on an unlabelled dataset of trajectories using some SSL algorithm, and then evaluate the SSL method by FT this pre-trained model on a set of downstream tasks and measuring performance on these tasks. At FT time, different paradigms could be used – we could initialize a classification head and then fine-tune the whole model, or train a linear classifier on the frozen model.

Note that this paradigm of PT followed by FT was also studied in Chapter 4 – as discussed previously in this thesis, it is a general framework for building powerful neural network models in situations where a FT task of interest is data-scarce. Here, our innovation is to develop a new PT algorithm for multimodal, multidimensional trajectory data, rather than use nested optimization to improve existing PT algorithms (as in Chapter 4). Given our use-case of PT on unlabelled data, we also focus exclusively on self-supervised PT, rather than supervised PT.

To pre-train the model, we assume access to an unlabelled PT dataset of N_{PT} patient trajectories, $\mathcal{D}_{\text{PT}} = \{\tau^{(n)}\}_{n=1}^{N_{\text{PT}}}$. To fine-tune the model, we assume access to

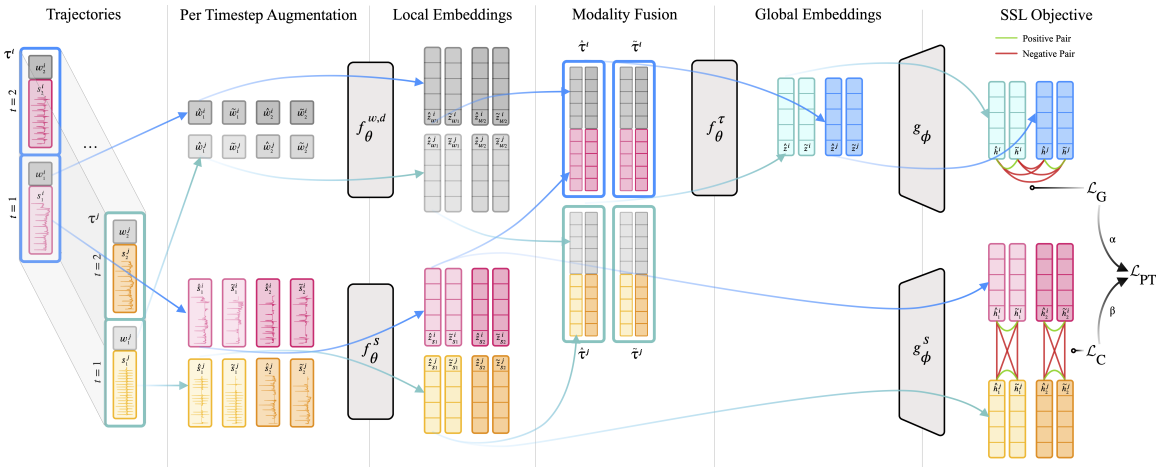


Figure 5-3: **Overview of Sequential Multi-Dimensional Self-Supervised Learning (SMD SSL),** which uses losses at two levels to encourage effective pre-training on complex time series. We start with a batch of trajectories, each denoted τ, consisting of a static vector d (not shown for clarity) and a sequence of signals s_t and structured data w_t (sequence of length 2 here). These data are augmented on a per-modality and per-timestep basis (arrows show flow for the data at a single timestep) and passed through encoders f_θ^s and $f_\theta^{w,d}$ to generate local embeddings of the signals and structured data at each timestep. The signal embeddings pass through a projection head g_ϕ^s, after which we compute a component SSL loss $\mathcal{L}_{\text{C}}$. Separately, the embedding of the entire trajectory (obtained by concatenating the per-modality embeddings) is passed through a sequence model f_θ^τ and a global projection head g_ϕ, on which we compute the global SSL loss $\mathcal{L}_{\text{G}}$. The total loss $\mathcal{L}_{\text{PT}}$ is a weighted sum of the component and global losses. SMD SSL can be instantiated with both contrastive and non-contrastive losses – shown here is a contrastive framing (as in SimCLR) with explicit negative pairs.

a set of labelled FT datasets – given K FT tasks indexed by k, we denote each FT dataset as $\mathcal{D}_{\text{FT}}^{(k)} = \{(\tau^{(n)}, y^{(n)}\}_{n=1}^{N_{\text{FT}}^{(k)}}$, where $y^{(n)}$ denotes the label for a trajectory $\tau^{(n)}$.

5.3.2 Sequential Multi-Dimensional SSL

We propose a new method for SSL on trajectories – Sequential Multi-Dimensional SSL (SMD SSL), depicted in Figure 5-3. Our approach builds on multi-view SSL like SimCLR [30] and VICReg [14], since prior work has successfully used these strategies on clinical data [47, 98, 63, 190, 133].

SMD SSL uses a loss function with two terms – a global loss, computed at the trajectory level, and a component loss, computed at the individual signal level. We now describe these two losses, and then present the overall objective.

Global Loss

The global loss $\mathcal{L}_{\text{G}}$ encourages the encoding model f_θ to embed similar trajectories to similar points in the representation space. We follow related work [30] and define a similar (or positive) pair of trajectories to be those that are augmentations of the same base trajectory, and negatives to be other trajectories in the sampled batch.

Given a trajectory-level augmentation function, the computation of the global loss proceeds as follows:

1. Sample a batch of trajectories from the PT dataset: $\{\tau^{(n)}\}_{n=1}^{B}$, where B is the batch size.

2. For each trajectory $\tau^{(n)}$, generate two augmented views of it: $\tilde{\tau}^{(n)}$ and $\hat{\tau}^{(n)}$.

3. Pass the augmented views through the representation model f_θ and a projection head g_ϕ generating two sets of projections: $\tilde{h}^{(n)}$ and $\hat{h}^{(n)}$.

4. Assemble projected pairs into two sets of matrices: $\hat{H} = [\hat{h}^{(1)}, \ldots, \hat{h}^{(B)}]$, $\tilde{H} = [\tilde{h}^{(1)}, \ldots, \tilde{h}^{(B)}]$.

5. The global loss is equal to the trajectory self-supervised loss $\mathcal{L}^{\tau}_{\text{SSL}}$ computed on these projections:

$$\mathcal{L}_{\text{G}} = \mathcal{L}^{\tau}_{\text{SSL}}(\hat{H}, \tilde{H}). \tag{5.1}$$

The choice of the trajectory self-supervised loss $\mathcal{L}^{\tau}_{\text{SSL}}$ is a design decision; one choice is the normalized temperature-scaled cross-entropy loss (NT-Xent) as in SimCLR [30, 31]. Given all $2B$ positive pairs (h_i, h_j) as the rows of the two matrices $[\hat{H}, \tilde{H}]$ and $[\tilde{H}, \hat{H}]$, we compute:

$$\mathcal{L}^{\tau}_{\text{NT-Xent}} = \frac{1}{2B} \sum_{i=1}^{2B} -\log \frac{\exp(\text{sim}(h_i, h_j)/\gamma)}{\sum_{k=1}^{2B} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(h_i, h_k)/\gamma)}, \tag{5.2}$$

where $\text{sim}(a, b)$ is cosine similarity and γ is the temperature hyperparameter. Another choice is the VICReg loss, minimizing mean squared error between positive pairs, with

variance and covariance regularizers [14]:

$$\mathcal{L}^{\tau}_{\text{VICReg}} = \lambda I(\hat{H}, \tilde{H}) + \mu \text{Var}(\hat{H}, \tilde{H}) + \nu \text{Cov}(\hat{H}, \tilde{H}), \tag{5.3}$$

where $I()$ is the mean squared error, Var() is the variance regularizer, Cov() is the covariance regularizer, and λ, μ, and ν are hyperparameters. Flexibility in the form of the loss function is beneficial since different applications might benefit from different losses. In our experiments, we use the NT-Xent and VICReg loss functions.

Component Loss

Pre-training with the global loss is a straightforward application of SSL to trajectories. However, each trajectory contains complex substructures (the high-frequency signals s_t) and the global loss alone may not be sufficient to guide the model to learn useful representations of these substructures. Incorporating a second loss term on the individual signal level, the *component loss* $\mathcal{L}_{\text{C}}$, leads to learning richer representations of the signals.

Given a signal augmentation function, we compute the component loss as follows:

1. Sample a batch of trajectories from the PT dataset: $\{\tau^{(n)}\}_{n=1}^{B}$, where B is the batch size.

2. Generate two augmented views of each signal in each trajectory. For a given trajectory $\tau^{(n)}$, let the two augmented sets of signals be: $\{\tilde{s}_t^{(n)}\}_{t=1}^{T}$ and $\{\hat{s}_t^{(n)}\}_{t=1}^{T}$.

3. Pass the augmented views through the signal encoder model f_θ^s and a signal projection head g_ϕ^s generating two sets of projections: $\{\tilde{h}_t^{(n)}\}_{t=1}^{T}$ and $\{\hat{h}_t^{(n)}\}_{t=1}^{T}$.

4. Assemble these pairs of projections into pairs on a per-timestep basis: $\hat{S}_t = [\hat{h}_t^{(1)}, \ldots, \hat{h}_t^{(B)}]$, $\tilde{S}_t = [\tilde{h}_t^{(1)}, \ldots, \tilde{h}_t^{(B)}], \forall t = 1, \ldots, T$.

5. The component loss is equal to the signal self-supervised loss $\mathcal{L}^{s}_{\text{SSL}}$ averaged

over timesteps:

$$\mathcal{L}_{\mathrm{C}} = \frac{1}{T}\sum_{t=1}^{T} \mathcal{L}^{s}_{\mathrm{SSL}}(\hat{S}_t, \tilde{S}_t). \tag{5.4}$$

As with the global loss, the form of the signal SSL loss used is a design decision. The intuition for computing the signal SSL loss separately at each timestep is that nearby timesteps in a trajectory can be very similar. Particularly if we use a contrastive loss such as NT-Xent, we do not want these nearby timesteps to serve as negative examples in the contrastive loss. Separating out the computation over timesteps addresses this issue.

Overall Objective

The overall objective used at PT is:

$$\mathcal{L}_{\mathrm{PT}} = \alpha\mathcal{L}_{\mathrm{G}} + \beta\mathcal{L}_{\mathrm{C}}. \tag{5.5}$$

The hyperparameters α and β control the contributions of the global and component losses. Fixing $\alpha = 0$ is SSL on a signal-level alone (only PT the signal encoder) and fixing $\beta = 0$ is SSL on the overall trajectory level alone; we evaluate both in our experiments, finding that combining the two losses is beneficial to performance. In Appendix D.2.3, we study the evolution of the two losses over SMD SSL training, which provides intuition as to the effect of each term during pre-training. We observe that the two terms are not independent (minimizing one also contributes to the other being reduced).

Connections to multi-scale learning. The overall objective can be interpreted as a multi-scale SSL loss, capturing the fact that trajectories contain predictive information at two different scales: individual physiological signals (component) and overall time series (global).

5.3.3 Augmentation Functions

SMD SSL requires augmentations for trajectories and signals in order to compute the global and component losses respectively. We now describe these.

Trajectory augmentation. We form an augmented trajectory by separately augmenting each of the data modalities within the trajectory, using the following approach for each data type (further details in Appendix D.1):

- **High-frequency signal s:** For each twenty second long signal in the trajectory of length T, we form a pair of augmented views by first splitting the signal into two disjoint segments (e.g., taking the first 10 seconds as one view, and the second 10 seconds as the second view), and then applying random masking and noise addition as augmentations to each view independently. This is similar to the approach used in CLOCS [98]. The intuition is that two segments of a signal that are close in time should encode similar physiology, and can therefore be considered paired views. Random masking and noise addition are commonly used as time-series augmentations [63, 208, 148, 88].

- **Structured-time series data w:** The tabular data sequence forms a $T \times M$ matrix over all timesteps of the trajectory. Following prior work [204], we apply two data augmentation strategies to this matrix: Gaussian noise addition and history cutout.

- **Static features d:** Following [204], we use random dropout and noise addition. Other corruption strategies (e.g., [7]), were found to be less effective, potentially because of being too strong (also seen in [106]). Dropout in particular is a sensible augmentation here since static features are sometimes missing, and are imputed with the mean in that case. Dropout therefore resembles a situation where the feature was not measured in the data.

We note that our approach of forming augmented trajectories by independently transforming each individual data type is straightforward, but not necessarily optimal.

Exploring other strategies for generating multiple views of trajectory data is an important direction of future work.

Signal augmentation. For computational efficiency, we re-use the augmented signals already generated during the trajectory-level augmentation when computing the component loss. However, additional/different augmentations could be applied on the signal level.

5.3.4 Broader Applications of SMD SSL

We have instantiated SMD SSL for a setting in which multimodal trajectories consists of a sequence of structured data and high-dimensional signals. Our approach is potentially valuable in any setting where we have sequences of high-dimensional data – the two-level loss function encourages representation learning on both an individual signal level and an overall sequence level. For example, SMD SSL could be useful in modeling sequence of medical images for a patient taken over time. The component loss encourages learning rich embeddings of individual images, and the global loss encourages learning temporal trends.

5.4 Experiments

In this section, we evaluate Sequential Multi-Dimensional Self-Supervised Learning (SMD SSL) on two clinical datasets. We begin by describing the datasets, tasks, and experimental setup. We then evaluate SMD SSL and baselines in two settings:

1. *Unimodal*: with trajectories that contain only a sequence of physiological signals.

2. *Multimodal*, with trajectories containing both signals and structured data.

We find that SMD SSL performs strongly in both settings. We also analyze SMD SSL's sensitivity to the component loss weight and its learned representations.

Table 5.1: Dataset Statistics.

	Dataset 1			Dataset 2		
Task	# Patients	# Trajectories	Prevalence	# Patients	# Trajectories	Prevalence
Pre-training	8888	43858	N/A	5022	26615	N/A
Elevated mPAP	2025	48511	77.5%	500	14957	87.9%
24hr Mortality	9605	57758	1.4%	5689	318306	2.3%

5.4.1 Datasets and Tasks

We consider two clinical datasets (Table 5.1):

- **Dataset 1** is a private dataset derived from the electronic health record (EHR) of the Massachusetts General Hospital (MGH), consisting of a cohort of patients with a prior diagnosis of heart failure. For each patient, we have structured data from the EHR and physiological signals measured by a bedside telemetry monitor. These signals include vitals signs such as heart rate (HR) and oxygen saturation (SpO2), measured at a low frequency (0.5 Hz), and waveforms such as the electrocardiogram (ECG), measured at a high frequency (240 Hz).

- **Dataset 2** is a public dataset derived from the commonly used MIMIC-III clinical database [93, 61] and its associated database of physiological signals [123]. The clinical database contains structured data over a patient's stay, and the physiological signals database contains vitals signs (HR, SpO2) and waveforms (ECG) measured by a bedside telemetry monitor.

  We use the widely adopted preprocessing pipeline introduced in [72] to form the specific cohort and extract the structured data features used in modeling. This pipeline also provides the functionality to create development and testing sets for the different downstream tasks we consider.

  The clinical database is available on PhysioNet [61] to credentialed users. The database of physiological signals is open-access on PhysioNet.

Constructing PT and FT sets. Each dataset consists of a number of patient hospital visits. We resample each patient's hospital visit at hourly resolution, and

each hour of a patient's stay represents a single timestep in our trajectory abstraction. For simplicity, we fix the length of trajectories to be 8 elements (letting a trajectory correspond to a common shift length of 8 hours).

To generate PT trajectories from these resampled visits, we first split each visit into non-overlapping contiguous 12 hour blocks. A PT trajectory is formed by first sampling a 12 hour block, and then selecting 8 contiguous timesteps from that block (with the starting timestep selected randomly). We discuss the implications of this in Appendix D.2.1. At a high level, this strategy ensures that negative samples never overlap in time with an anchor trajectory.

To generate FT trajectories, we use a sliding window to select contiguous 8 hour blocks at 1 hour increments from each visit. Each of these contiguous 8 hour blocks is a trajectory in the FT dataset. The trajectory labels are formed based on the specific task, as described below.

Fine-tuning tasks. We consider two predictive tasks:

- **Elevated mPAP**: Each hour, detect whether a patient's mean Pulmonary Arterial Pressure (mPAP) is abnormally high. This task is of clinical interest since the mPAP is typically measured via an invasive study, and so inferring whether it is abnormal using minimally invasive information (i.e., the ECG, labs, and vitals signs) is valuable. Prior work [164, 147] studied similar tasks of predicting hemodynamic variables from the 12-lead ECG, but not in the context of online trajectory data, as we do here.

- **24hr Mortality:** Each hour, predict whether the patient is going to die in the next 24 hours. This task is commonly used to evaluate predictive models for ICU time series data [120, 204] – our goal with studying this task is to understand how our approach performs when compared to other established methods. In Dataset 2, this task is named 'Decompensation' in the preprocessing pipeline from [72]; we refer to it at 24hr mortality here.

Studying these two predictive tasks, particularly the former, connects back to the

broader motivating goal of constructing data-efficient ML models for applications in cardiovascular medicine (discussed in Chapter 1).

Trajectory features and preprocessing. The trajectories in PT and FT sets consist of static features d and a time-series of structured data and physiological signals $\{(w_t, s_t)\}_{t=1}^{T}$, as presented in Section 5.3.1. The static features d contain information on infrequently measured vitals signs and lab values; $d \in \mathbb{R}^9$ in Dataset 1 and $d \in \mathbb{R}^{38}$ in Dataset 2. At each timestep, w_t captures summary statistics related to regularly measured vitals signs within a 1 hour window; $w_t \in \mathbb{R}^{30}$ in Dataset 1 and $w_t \in \mathbb{R}^{13}$ in Dataset 2. s_t is a 10 second electrocardiogram (ECG) signal extracted from a longer signal measured within each 1 hour window; in Dataset 1, $s_t \in \mathbb{R}^{4\times 2400}$ is a 240 Hz 4-channel ECG, and in Dataset 2, $s_t \in \mathbb{R}^{1\times 1250}$ is a 125 Hz 1-channel ECG.

Missing structured data are forward-fill imputed if part of a time series, and otherwise imputed with the mean over the training dataset. Missing signals are represented with zeros. Any trajectories that have more than 1 timestep with a missing signal are excluded. Further dataset and preprocessing details are in the appendix.

5.4.2 Experimental Setup

Dataset splits. We split Dataset 1 on a per-patient level into 80/20 development/test sets and use 20% of the development set as a validation set. For Dataset 2, we use the predefined development/test split defined in the preprocessing pipeline [72], and use 20% of the development set (splitting on a per-patient basis) as a validation set.

Model architecture. Recall that the encoder f_θ has three components: we implement the structured features encoder as a 2-layer MLP, the signals encoder as a 1-D ResNet18 CNN [77], and the sequence model as a 4-layer GRU. We use a 2-layer MLP for the projection head g_ϕ. The model architecture is described more fully in

Appendix D.2.2, and is shown pictorially in Figure 5-2.

Training setup. We conduct pre-training for 15 epochs, using a batch size of 128, with the Adam optimizer [96]. We found that model performance did not improve with longer pre-training times (Appendix D.2.3).

At fine-tuning time, we consider two evaluation strategies:

1. Linear evaluation: train a linear classifier on the frozen representations from f_θ [30].
2. Full FT: initialize a new linear layer after f_θ and fine-tune the entire model for a maximum of 10 epochs with Adam (performance did not improve after this point), with early stopping based on validation AUROC.

For each method and task, we report the test set AUROC from the evaluation strategy that obtains the best validation set AUROC. We adopt this approach since our goal is to determine which self-supervised pre-training approach is best – in order to do so fairly, we compare results under the evaluation strategy that obtains the highest performance. To obtain error bars, we use bootstrapping: we sample with replacement 100 bootstraps from the testing dataset, and report the 95% confidence interval in AUROC over these bootstraps. Since the mortality task has low prevalence, we additionally report trends in AUPRC in Appendix D.2.3.

Unimodal and Multimodal evaluation. SMD SSL is generally applicable when we have sequence-structured data where at least one elemtn of the sequence is itself high-dimensional. As mentioned earlier, we consider two instantiations of such sequences here: (1) the *unimodal setting*, where the input trajectory only contains the signals sequence of the input, $\tau = \{s_t\}_{t=1}^T$; (2) the full *multimodal setting*, where the input trajectory contains the full input sequence of structured data and signals, $\tau = (d, \{(w_t, s_t)\}_{t=1}^T)$.

Baselines and SMD SSL variations. Existing methods for SSL on clinical data are not exactly applicable, since they do not study pre-training pipelines for multimodal and multi-dimensional time series. For example, Neighbourhood Contrastive

Learning (NCL) [204] is primarily designed for structured data time series alone, and CLOCS [98] and SACL [35] operate on single physiological waveforms (rather than sequences of waveforms).

As a result, we focus here on evaluating general SSL methods as baselines (with further baselines in Appendix D.2.3), varying whether we use the component and/or global loss, in both unimodal and multimodal settings. Our goal is to understand whether the two-level loss formulation boosts performance. The full set of baselines is as follows (further details in Appendix D.2.2):

- **RandInit**: A standard baseline: train a model from random initialization on each FT task.
- **SimCLR (global) [30]:** Pre-train using the NT-Xent global loss alone. This is SimCLR PT on the trajectory level, setting $\alpha = 1, \beta = 0$ in Eqn. 5.5.
- **VICReg (global) [14]:** Pre-train using the VICReg global loss alone. This is VICReg PT on the trajectory level, setting $\alpha = 1, \beta = 0$ in Eqn. 5.5.
- **SimSiam (global) [34]:** Pre-train using SimSiam at a global level. This is SimSiam PT on the trajectory level.
- **SimCLR (component):** Pre-train using the NT-Xent component loss alone ($\alpha = 0, \beta = 1$ in Eqn. 5.5).
- **VICReg (component):** Pre-train using the VICReg component loss alone ($\alpha = 0, \beta = 1$ in Eqn. 5.5).
- **SimSiam (component):** Pre-train using SimSiam at a component level.

We consider two variations of SMD SSL:

- **SMD SSL (SimCLR):** Pre-train using SMD SSL with the NT-Xent loss (Eqns. 5.2 and 5.5), fixing the global loss weight $\alpha = 1$ and tuning the component loss weight β.

Table 5.2: **Pre-training with the SMD SSL objective improves performance on both datasets in the unimodal setting.** Mean and 95% confidence interval of AUROC on FT tasks. We observe that PT methods outperform not doing PT, and training with the SMD SSL (component and global) losses boosts performance the most.

(a) Results on Dataset 1.

	Elevated mPAP	24hr Mortality
RandInit	65.0 ± 0.1	56.1 ± 0.6
SimCLR (global)	68.1 ± 0.1	66.7 ± 0.5
VICReg (global)	66.6 ± 0.1	64.9 ± 0.5
SimSiam (global)	63.8 ± 0.1	50.6 ± 0.6
SimCLR (component)	67.5 ± 0.1	71.7 ± 0.5
VICReg (component)	68.7 ± 0.1	63.5 ± 0.4
SimSiam (component)	64.2 ± 0.1	54.3 ± 0.5
SMD SSL (SimCLR)	**69.9 ± 0.1**	72.3 ± 0.4
SMD SSL (VICReg)	67.6 ± 0.1	**74.6 ± 0.5**

(b) Results on Dataset 2.

	Elevated mPAP	24hr Mortality
RandInit	63.4 ± 0.4	54.6 ± 0.2
SimCLR (global)	65.9 ± 0.4	56.6 ± 0.2
VICReg (global)	66.7 ± 0.4	53.6 ± 0.2
SimSiam (global)	61.9 ± 0.4	61.1 ± 0.2
SimCLR (component)	65.7 ± 0.4	61.1 ± 0.2
VICReg (component)	66.7 ± 0.4	63.1 ± 0.2
SimSiam (component)	65.9 ± 0.4	50.3 ± 0.2
SMD SSL (SimCLR)	**67.0 ± 0.4**	**65.9 ± 0.2**
SMD SSL (VICReg)	66.6 ± 0.4	58.5 ± 0.2

- **SMD SSL (VICReg):** Pre-train using SMD SSL with the VICReg loss (Eqns. 5.3 and 5.5), fixing the global loss weight $\alpha = 1$ and tuning the component loss weight β.

Hyperparameters. There are various hyperparameters to tune, such as learning rates and loss weighting for VICReg and SMD SSL. Evaluating many hyperparameters is computationally expensive (involves doing both PT and FT runs), so we conduct a reduced search on a subset of the hyperparameters focusing only on the Elevated mPAP task in the unimodal setting, optimizing validation AUROC. We include full details in Appendix D.2.2.

5.4.3 Results

Unimodal Evaluation

Table 5.2 shows results in the unimodal setting, where the input trajectories consist only of the signals. We highlight three key takeaways from these results:

1. **Pre-training (PT) helps performance.** In the unimodal setting, PT (particularly SimCLR or VICReg variants in Tables 5.2a and 5.2b) almost always improve performance over not doing any PT (RandInit in Tables 5.2a and 5.2b). This result is expected, since we would expect that given the complex input space, a PT phase for the highly-parameterized CNN encoder and GRU should condition the model better for the downstream tasks, given the limited amount of labelled data on these tasks.

2. **SMD SSL obtains the best performance.** On both datasets, we observe that a SMD SSL method does best, suggesting the utility of a two-level loss, which encourages the learning of informative representations on both a signal-level and a sequence-level.

3. **SMD SSL vs single-level SSL.** When using SimCLR, we find that SMD SSL consistently improves on both component-only and global-only SimCLR models. With VICReg, improvements are less consistent, and the component-only VICReg variation often performs the best. This may be because VICReg has many loss weighting terms, and these were not jointly tuned with the component loss weight in SMD SSL. A more thorough hyperparameter search, perhaps using efficient gradient-based methods [143], might improve performance of SMD SSL (VICReg).

Unimodal experiments with structured data. Since our main contribution is the strategy to conduct SSL on the sequence of high-dimensional signals, our *unimodal evaluation* focuses on the setting where the only modality in the trajectory is the signals. For completeness however, we also show in Table 5.3 the results for the other

Table 5.3: Mean and 95% confidence interval of AUROC on fine-tuning tasks in the unimodal setting when only using structured data, comparing no PT (RandInit) to PT with the SimCLR and VICReg global losses. We find that when considering structured data alone, PT does not offer much benefit to performance; however, there are improvements seen in the Elevated mPAP task.

(a) Results on Dataset 1.

	Elevated mPAP	24hr Mortality
RandInit	65.3 ± 0.1	**79.0 ± 0.4**
SimCLR	**66.8 ± 0.1**	**79.0 ± 0.4**
VICReg	66.0 ± 0.0	77.9 ± 0.4

(b) Results on Dataset 2.

	Elevated mPAP	24hr Mortality
RandInit	65.0 ± 0.3	**90.1 ± 0.1**
SimCLR	66.8 ± 0.3	88.1 ± 0.1
VICReg	**68.1 ± 0.3**	89.3 ± 0.1

unimodal setting, when training on only the structured data (structured time-series and static vector) in the trajectory. The key finding is that the Elevated mPAP task can benefit from pre-training, but the 24hr Mortality task performance is not boosted by pre-training. This is likely because the structured data are relatively simple and low-dimensional, and there is enough data to learn useful predictive information from these data as-is, without pre-training.

Multimodal Evaluation

Table 5.4 shows results in the multimodal setting, where the input trajectories consist of both signals and structured data. In addition to the aforementioned baselines, we also include results from RandInit and SSL methods trained on only signals and on only structured data. We highlight some key takeaways from these results:

1. **The effect of multimodal data is task-specific.** Incorporating both the signals and structured data leads to improvements in the Elevated mPAP task on both datasets (particularly with SMD SSL), but has less significant effects in the 24hr Mortality task. This is likely because the structured data in their raw (relatively low-dimensional) form are highly predictive of mortality, and so there

is little benefit to be gained from PT. This is seen clearly when comparing the performance of RandInit (structured) and SSL (structured). This phenomenon has been observed in prior work [120].

2. **SMD SSL performs effectively.** On the Elevated mPAP task (both datasets) and 24hr mortality task (Dataset 2), SMD SSL, either with the SimCLR or VICReg loss function, obtains the best performance among all methods.

3. **SMD SSL vs single-level SSL.** When compared to single-level SSL, we observe improvements when using SMD SSL on Elevated mPAP for both SimCLR and VICReg. However, this is not the case for 24hr Mortality – for example, component-only SSL with SimCLR on this task performs better than SMD SSL. This may arise because structured data PT does not improve performance significantly, and so it is preferable to only pre-train the signals encoder rather than the entire model.

Table 5.4: **Pre-training with the SMD SSL objective improves performance in three settings in a multimodal evaluation.** Mean and 95% confidence interval of AUROC on FT tasks. In all settings except 24hr Mortality on Dataset 1, we observe that SMD SSL obtains the best performance. The 24 hour mortality task on Dataset 1 appears to benefit little from incorporating the high-dimensional signals, which could explain why SMD SSL worsens performance here.

(a) Results on Dataset 1.

	Elevated mPAP	24hr Mortality
RandInit (signals)	65.0 ± 0.1	56.1 ± 0.6
SSL (signals)	69.9 ± 0.1	74.6 ± 0.5
RandInit (structured)	65.3 ± 0.1	**79.1 ± 0.4**
SSL (structured)	66.7 ± 0.1	79.0 ± 0.4
RandInit	69.1 ± 0.1	79.0 ± 0.4
SimCLR (global)	69.8 ± 0.1	76.4 ± 0.5
VICReg (global)	69.4 ± 0.1	78.0 ± 0.4
SimSiam (global)	69.6 ± 0.1	78.4 ± 0.4
SimCLR (component)	71.4 ± 0.1	78.6 ± 0.4
VICReg (component)	64.0 ± 0.1	74.0 ± 0.5
SimSiam (component)	68.4 ± 0.1	79.0 ± 0.4
SMD SSL (SimCLR)	**72.3 ± 0.1**	77.4 ± 0.4
SMD SSL (VICReg)	70.3 ± 0.1	77.0 ± 0.4

(b) Results on Dataset 2.

	Elevated mPAP	24hr Mortality
RandInit (signals)	63.4 ± 0.4	54.6 ± 0.2
SSL (signals)	67.0 ± 0.4	65.9 ± 0.2
RandInit (structured)	65.3 ± 0.3	90.0 ± 0.1
SSL (structured)	68.3 ± 0.3	89.3 ± 0.1
RandInit	65.3 ± 0.3	87.8 ± 0.1
SimCLR (global)	63.7 ± 0.4	86.6 ± 0.1
VICReg (global)	70.4 ± 0.4	87.8 ± 0.1
SimSiam (global)	60.6 ± 0.3	90.4 ± 0.1
SimCLR (component)	59.7 ± 0.4	89.8 ± 0.1
VICReg (component)	67.1 ± 0.4	84.4 ± 0.1
SimSiam (component)	67.4 ± 0.4	90.6 ± 0.1
SMD SSL (SimCLR)	69.9 ± 0.3	88.1 ± 0.1
SMD SSL (VICReg)	**71.6 ± 0.3**	**90.7 ± 0.1**

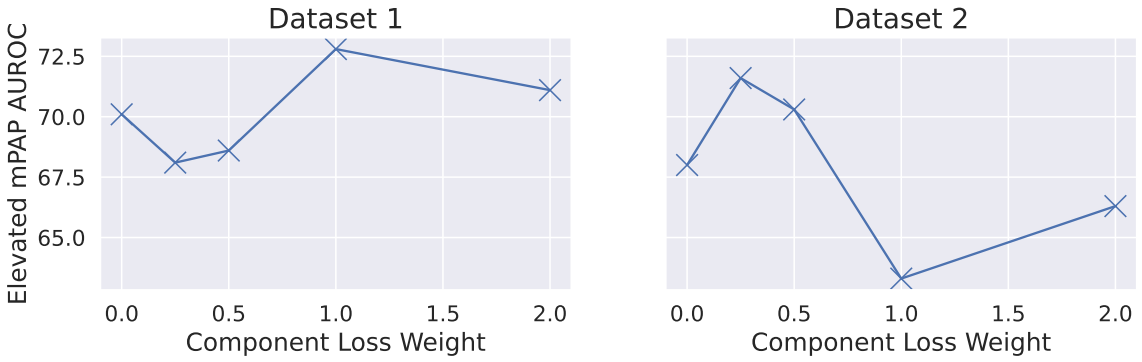


Figure 5-4: **Studying sensitivity to the component loss weight.** We find a higher optimal component loss weight on Dataset 1 compared to Dataset 2, possibly due to Dataset 1 having more complex signals.

Further Results

Additional evaluation. In Appendix D.2.3, we present three additional experiments. First, we compare SMD SSL to other global-only baselines using different loss functions (NCL, CLOCS, and SACL), finding that SMD SSL improves on these methods. Second, we compare linear evaluation to full FT for different methods, finding that full FT improves on linear evaluation. Third, we investigate the effect of longer PT times, finding that performance does not improve.

Component loss weight sensitivity. Considering SMD SSL (SimCLR) in the unimodal setting, we fix the global loss weight $\alpha = 1.0$ and vary the component loss weight β for the Elevated mPAP task. The validation set AUROC results are shown in Figure 5-4. The optimal value of the component loss weight is higher for Dataset 1, perhaps because the signals in Dataset 1 are more complex than the signals in Dataset 2 (higher sampling rate, multiple channels).

Representational similarity analysis. Using Centered Kernel Alignment (CKA) [101], we study the representations learned by the signal encoder on Dataset 2 under different SSL methods. Our findings are: (1) representations from SMD SSL encode aspects of both component-only SSL and global-only SSL (illustrated in Figure 5-5); (2) the component loss appears to have more effect in the earlier layers of the signal

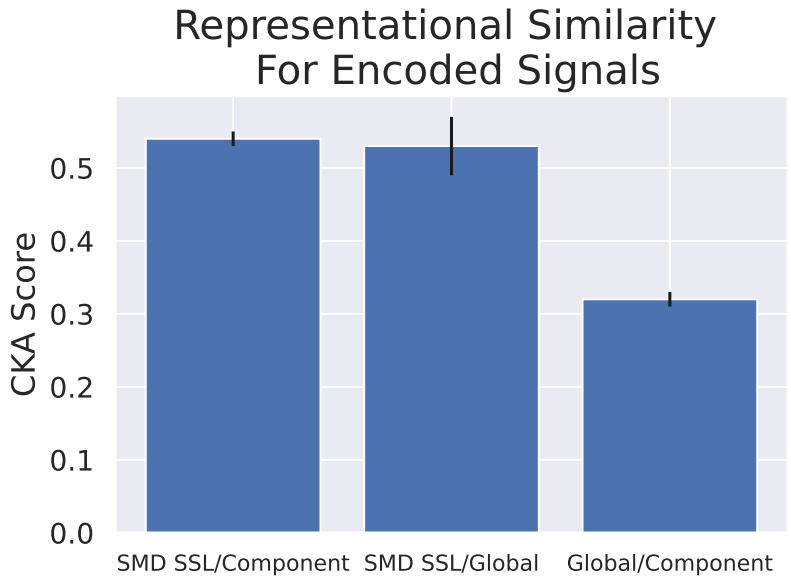


Figure 5-5: **Learned representations from SMD SSL are similar to both component-only and global-only SSL.** Comparing learned representations of the signals using different SimCLR-based SSL strategies – SMD SSL, Global, and Component, using Centered Kernel Alignment (CKA). Global and Component SSL show low representational similarity (right-most bar), but SMD SSL shows higher similarity with both individually, suggesting that learned representations in SMD SSL encode aspects of both component and global SSL.

encoder, whereas the global loss has more effect in later layers (details in Appendix D.2.3).

5.5 Scope and Limitations

Intended use-case. Our method is most appropriate in settings where we have multimodal trajectory data for patients, i.e., both structured data and ECGs are available. This use case is driven by our target application of monitoring patients with cardiovascular disease (Dataset 1), where the majority of patients have both structured data and ECGs. In datasets where a small proportion of patients have multimodal data (such as in Dataset 2), it may be preferable to use other unimodal SSL approaches so that patients without multimodal data can still be included in model development. Although Dataset 2 does not exactly match our intended use case, we still considered it because it is publicly available and well-studied in related

work.

Choice of data augmentations. SMD SSL uses data augmentations to generate different views. Our main contribution is in our two-level loss function, so we did not investigate novel augmentation strategies, instead leveraging existing effective data augmentations for clinical data. Our framework could equally apply with different augmentations or multiview generation approaches. However, in a setting where effective augmentations (or methods to generate different views) do not exist, our method may not be effective. It may be preferable to use other pre-training schemes in those scenarios, such as supervised pre-training or masked imputation [119].

Additional data modalities. Our experiments focused on clinical time series consisting of a sequence of structured data and high-dimensional physiological signals; we did not include predictive information that may be available from other modalities, such as medical imaging. SMD SSL could be extended to this scenario, and it could be a valuable direction to explore.

5.6 Conclusion

In this chapter, we continued our exploration of the pre-training (PT) followed by fine-tuning (FT) paradigm, and focused specifically on self-supervised pre-training for complex, unlabelled clinical time series data. We observed that real-world clinical time series are often multimodal, and individual timesteps in a time series may contain high-dimensional information, such as physiological signals. Existing self-supervised learning (SSL) methods and pipelines cannot readily be used for pre-training on data of this form.

To address this gap, we first introduced an abstraction of these multimodal and multidimensional time-series, which we called *trajectories*. We then outlined a new method for pre-training on trajectories, Sequential Multi-Dimensional SSL (SMD SSL), whose design is motivated by the structure of these complex time-series. SMD SSL encourages effective pre-training on both a component level (level of individual

signals in a sequence) and a global level (level of the entire sequence). In experiments on two clinical datasets, we showed that pre-training with SMD SSL and then fine-tuning improves performance on downstream tasks compared to baselines.

Chapter 6

Summary and Conclusion

6.1 Thesis Summary

This thesis presented four contributions to data-efficient machine learning (ML), where the goal is to build effective ML models given only limited (labeled) training data. Our contributions span different paradigms of data-efficient ML, namely:

1. Few-shot learning, where we have many related prediction tasks (often where each task is data-scarce), and we wish to learn a predictive model that generalizes effectively to new tasks.

2. Supervised Learning on small datasets, where we have a task with a small labelled dataset, and wish to learn an effective model for that task.

3. Transfer and Semi-Supervised Learning, where we have a small labeled dataset of interest (the downstream dataset) and a large related dataset (labelled or unlabelled) that we can leverage to build more effective models for the downstream dataset.

Our contributions include both (1) analysis of existing strategies for data-efficient ML, revealing potential areas for improvement; and (2) new methods to build data-efficient models that improve on the current state of the art.

Aside from the theme of data-efficient ML, two other common themes are explored. From a methodological standpoint, three of the contributions in the thesis draw on ideas from nested optimization. From an application standpoint, three of the contributions are motivated by challenges in applying ML to cardiovascular medicine, a setting in which it is not always possible to obtain large labelled datasets for model training.

In more detail, the contributions of each chapter are as follows:

1. In Chapter 2, we studied few-shot learning, and analyze why MAML [56], a popular few-shot learning method, works as well as it does: a question that had previously not been explored. Through our investigation, we understood why MAML is effective, and this led us to a method for simplifying it. We formalized and benchmarked this simplified version of MAML, finding that it performs almost equivalently to the original algorithm while offering substantial computational benefits.

2. In Chapter 3, we studied the use of data augmentation to improve model performance on data-scarce prediction problems from electrocardiogram (ECG) data. Through a detailed empirical study, we showed that existing task-agnostic data augmentation strategies for ECG data are not consistently beneficial. We outlined a learnable task-aware data augmentation strategy for ECG data that outperforms (or performs as well as) other augmentation strategies on a range of ECG predictive problems.

3. In Chapter 4, we studied the learning paradigm of pre-training (PT) followed by fine-tuning (FT). This paradigm is relevant when a FT task of interest is data-scarce, but we have abundant related data that can be used in PT to learn an effective model initialization. PT algorithms often have complex design choices, which we term meta-parameters, that can significantly affect the quality of pre-trained models and their suitability for FT. We introduced a new, scalable gradient-based PT algorithm that jointly learns these meta-parameters and base model parameters over a single PT run. This algorithm

uses a novel gradient estimator for computing gradients through the two-stage PT followed by FT process. In two experimental domains — multitask PT on protein-protein interaction graphs and self-supervised PT on ECG data — we found our PT algorithm results in pre-trained models that perform better than baselines after undergoing FT.

4. In Chapter 5, we studied a specific instantiation of the pre-training then fine-tuning paradigm. We explored using self-supervised learning (SSL) algorithms for pre-training models on unlabeled multimodal multidimensional time-series data from the intensive care unit, focusing on a cohort of patients with a diagnosis of cardiovascular disease (heart failure). We outlined a new SSL algorithm designed for the structure of this data stream, utilizing a multiscale loss function, and demonstrated that our method outperforms baselines on two clinical datasets.

6.2 Limitations and Future Work

There are several areas of promising future work that address limitations of this work. We present selected topics below.

Chapter 3: Data Augmentation for ECG Supervised Learning.

- **Local transformations.** Our approach for ECG augmentation, TaskAug, does not incorporate transformations that affect only local structures (such as specific parts of the cardiac cycle). Doing so could improve performance, potentially leveraging mathematical techniques from [138].

- **Latent space augmentation.** We study augmentations in the raw signal space, in the time-domain. A promising investigation area is to augment ECG data in a model's latent space [36] in addition to/instead of the signal space – these latent space transformations might promote additional invariance in the models, leading to improved performance. A further advantage of latent space

transformations is that they can be readily extended to multimodal data input (e.g., ECG + tabular data), whereas the feature space transformations we use require transforming each modality independently.

Chapter 4: Nested Optimization and Pre-Training.

- **Broader applications.** We applied our algorithm to optimize two types of meta-parameters in two domains: task weights for multitask PT on graph data and augmentations for self-supervised PT on ECGs. An extension to this work could apply the method to richer augmentation strategies, such as those explored in Chapter 3. Augmentations are a crucial component of self-supervised PT methods, and thus optimizing them further could yield significant benefits. This could be especially valuable in under-explored domains where the best augmentations are unknown – one could parameterize a highly general transformation space, and optimize augmentation meta-parameters accordingly. Applications of our method to other domains (e.g., computer vision, natural language understanding), and to other PT meta-parameters (such as those discussed in [145]) are also interesting directions of investigation.

- **Algorithmic investigations.** Our gradient estimator uses backpropagation through training to compute derivatives through the FT stage. We used this strategy since it does not requiring tuning a second implicit differentiation approximation. However, this design choice is not necessarily optimal – for example, would a proximal regularization approach [153] be more appropriate for this stage? In general, a more thorough benchmarking of our algorithm, analogously to what was explored for standard hyperparameter optimization in [38], is an important direction of inquiry.

Chapter 5: Self-Supervised Learning on Multimodal Clinical Time Series.

- **Long time series.** Our method operates on sequences of short ECG segments (10 seconds). Often, much longer ECGs are recorded, and incorporating these

could be advantageous. Utilizing newer time-series encoding architectures such as structured state space models [68, 170] could allow us to efficiently model these longer time series. Doing so would likely require adaptations to our SSL pipeline.

- **Information content overlap.** We focus on a scenario in which the data modalities measured have partial overlap in information. That is, the vitals signs/labs and ECGs recorded for a given patient capture both shared and unshared aspects of a patient's physiology. However, our SSL pipeline treats the two data modalities independently, in contrast to other work in multimodal self-supervised learning, e.g., [139]. Designing a new loss function that captures the partial overlap between the information contained in the different modalities, could lead to a more effective pre-training scheme. Methods such as [62] that have both a intra-modality and inter-modality objective could also be interesting to explore.

- **Other data modalities.** Many other data modalities are recorded for a given patient, such as imaging, omics data, etc. Extending our approach to these data modalities that are more sparsely measured is a valuable investigation.

6.3 Final Takeaways

This dissertation outlined four contributions to data-efficient machine learning (ML), spanning different paradigms: few-shot learning, traditional supervised learning, transfer learning, and semi-supervised learning. We both analyze existing approaches for data-efficient ML within these paradigms, and propose new methods that improve on the existing state of the art.

Many of our contributions are motivated by applying ML to cardiovascular medicine. This application area has inspired both the choice of problems studied and the data modalities for which we develop methods. An important message is that effective modelling of specialized data modalities (e.g., electrocardiograms) often requires new

methodological insights (such as new data augmentations and self-supervised learning algorithms).

We hope that the methods and insights from this thesis inform future work on data-efficient ML both in the domain of cardiovascular medicine and more broadly.

Appendix A

Additional Information and Results for Chapter 2

A.1 Few-Shot Image Classification Datasets and Experimental Setups

We consider the few-shot learning paradigm for image classification to evaluate MAML and ANIL. We evaluate using two datasets often used for few-shot multiclass classification – the Omniglot dataset and the MiniImageNet dataset.

Omniglot: The Omniglot dataset consists of over 1600 different handwritten character classes from 23 alphabets. The dataset is split on a character-level, so that certain characters are in the training set, and others in the validation set. We consider the 20-way 1-shot and 20-way 5-shot tasks on this dataset, where at test time, we wish our classifier to discriminate between 20 randomly chosen character classes from the held-out set, given only 1 or 5 labelled example(s) from each class from this set of 20 testing classes respectively. The model architecture used is identical to that in the original MAML paper, namely: 4 modules with a 3 x 3 convolutions and 64 filters with a stride of 2, followed by batch normalization, and a ReLU nonlinearity. The Omniglot images are downsampled to 28 x 28, so the dimensionality of the last

hidden layer is 64. The last layer is fed into a 20-way softmax. Our models are trained using a batch size of 16, 5 inner loop updates, and an inner learning rate of 0.1.

MiniImageNet: The MiniImagenet dataset was proposed by [155], and consists of 64 training classes, 12 validation classes, and 24 test classes. We consider the 5-way 1-shot and 5-way 5-shot tasks on this dataset, where the test-time task is to classify among 5 different randomly chosen validation classes, given only 1 and 5 labelled examples respectively. The model architecture is again identical to that in the original paper: 4 modules with a 3 x 3 convolutions and 32 filters, followed by batch normalization, ReLU nonlinearity, and 2 x 2 max pooling. Our models are trained using a batch size of 4. 5 inner loop update steps, and an inner learning rate of 0.01 are used. 10 inner gradient steps are used for evaluation at test time.

A.2 Additional Details and Results: Freezing and Representational Similarity

In this section, we provide further experimental details and results from freezing and representational similarity experiments.

A.2.1 Experimental Details

We concentrate on MiniImageNet for our experiments in Section 2.3.2, as it is more complex than Omniglot.

The model architecture used for our experiments is identical to that in the original paper: 4 modules with a 3×3 convolutions and 32 filters, followed by batch normalization, ReLU nonlinearity, and 2×2 max pooling. Our models are trained using a batch size of 4, 5 inner loop update steps, and an inner learning rate of 0.01. 10 inner gradient steps are used for evaluation at test time. We train models 3 times with different random seeds. Models were trained for 30000 iterations.

A.2.2 Details of Representational Similarity

CCA takes in as inputs $L_1 = \{z_1^{(1)}, z_2^{(1)}, ..., z_m^{(1)}\}$ and $L_2 = \{z_1^{(2)}, z_2^{(1)}, ..., z_n^{(2)}\}$, where L_1, L_2 are layers, and $z_i^{(j)}$ is a neuron activation vector: the vector of outputs of neuron i (of layer L_j) over a set of inputs X. It then finds linear combinations of the neurons in L_1 and neurons in L_2 so that the resulting activation vectors are maximally correlated, which is summarized in the canonical correlation coefficient. Iteratively repeating this process gives a similarity score (in $[0, 1]$ with 1 identical and 0 completely different) between the representations of L_1 and L_2.

We apply this to compare corresponding layers of two networks, net1 and net2, where net1 and net2 might differ due to training step, training method (ANIL vs MAML) or the random seed. When comparing convolutional layers, as described in [149], we perform the comparison over channels, flattening out over all of the spatial dimensions, and then taking the mean CCA coefficient. We average over three random repeats.

A.2.3 Similarity Before and After Inner Loop with Euclidean Distance

In addition to assessing representational similarity with CCA/CKA, we also consider the simpler measure of Euclidean distance, capturing how much weights of the network change during the inner loop update (task-specific finetuning). We note that this experiment does not assess functional changes on inner loop updates as well as the CCA experiments do; however, they serve to provide useful intuition.

We plot the per-layer average Euclidean distance between the initialization θ and the finetuned weights $\theta_m^{(b)}$ across different tasks T_b, i.e.

$$\frac{1}{N}\sum_{b=1}^{N} ||(\theta_l) - (\theta_l)_m^{(b)}||$$

across different layers l, for MiniImageNet in Figure A-1. We observe that very quickly after the start of training, all layers except for the last layer have small Euclidean

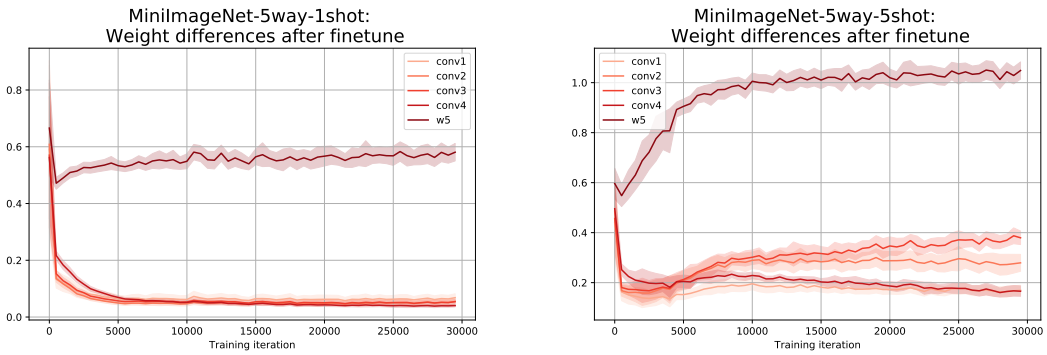


Figure A-1: **Euclidean distance before and after finetuning for MiniImageNet**. We compute the average (across tasks) Euclidean distance between the weights before and after inner loop adaptation, separately for different layers. We observe that all layers except for the final layer show very little difference before and after inner loop adaptation, suggesting significant feature reuse.

distance difference before and after finetuning, suggesting significant feature reuse. (Note that this is despite the fact that these layers have more parameters than the final layer.)

A.2.4 CCA Similarity Across Random Seeds

The experiment in Section 2.3.2 compared representational similarity of L_1 and L_2 at different points in training (before/after inner loop adaptation) but corresponding to the same random seed. To complete the picture, it is useful to study whether representational similarity across *different* random seeds is also mostly unaffected by the inner loop adaptation. This motivates four natural comparisons: assume layer L_1 is from the first seed, and layer L_2 is from the second seed. Then we can compute the representational similarity between (L_1 pre, L_2 pre), (L_1 pre, L_2 post), (L_1 post, L_2 pre) and (L_1 post, L_2 post), where pre/post signify whether we take the representation before or after adaptation.

Prior work has shown that neural network representations may vary across different random seeds [150, 124, 108, 193], organically resulting in CCA similarity scores much less than 1. So to identify the effect of the inner loop on the representation, we plot the CCA similarities of (i) (L_1 pre, L_2 pre) against (L_1 pre, L_2 post) and (ii) (L_1 pre, L_2 pre) against (L_1 post, L_2 pre) and (iii) (L_1 pre, L_2 pre) against (L_1

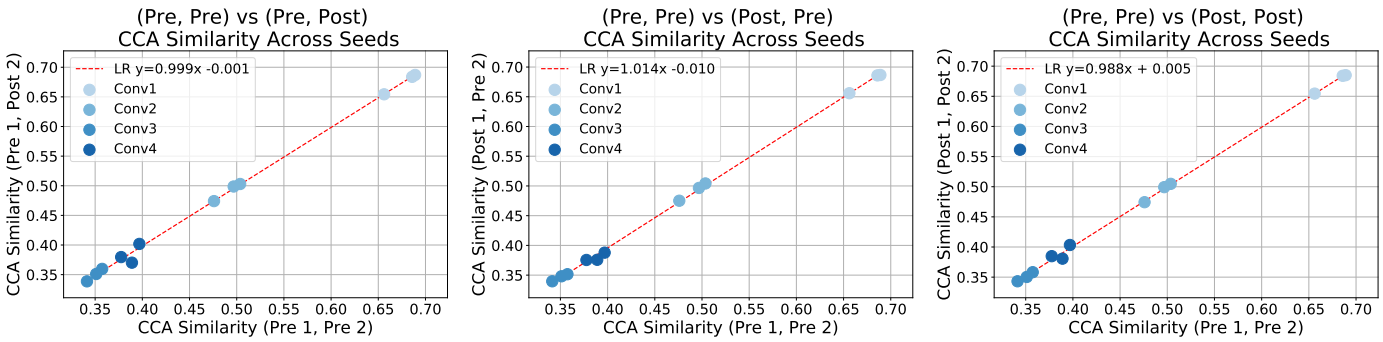


Figure A-2: **Computing CCA similarity pre/post adaptation across different random seeds further demonstrates that the inner loop doesn't change representations significantly.** We compute CCA similarity of L_1 from seed 1 and L_2 from seed 2, varying whether we take the representation *pre* (before) adaptation or *post* (after) adaptation. To isolate the effect of adaptation from inherent variation in the network representation across seeds, we plot CCA similarity of of the representations before adaptation against representations after adaptation in three different combinations: (i) (L_1 pre, L_2 pre) against (L_1 pre, L_1 post), (ii) (L_1 pre, L_2 pre) against (L_1 pre, L_1 post) (iii) (L_1 pre, L_2 pre) against (L_1 post, L_2 post). We do this separately across different random seeds and different layers. Then, we compute a line of best fit, finding that in all three plots, it is almost identical to $y = x$, demonstrating that the representation does not change significantly pre/post adaptation. Furthermore a computation of the coefficient of determination R^2 gives $R^2 \approx 1$, illustrating that the data is well explained by this relation. In Figure A-3, we perform this comparison with CKA, observing the same high level conclusions.

post, L_2 post) separately across the different random seeds and different layers. We then compute the line of best fit for each plot. If the line of best fit fits the data and is close to $y = x$, this suggests that the inner loop adaptation doesn't affect the features much – the similarity before adaptation is very close to the similarity after adaptation.

The results are shown in Figure A-2. In all of the plots, we see that the line of best fit is almost exactly $y = x$ (even for the pre/pre vs post/post plot, which could conceivably be more different as both seeds change) and a computation of the coefficient of determination R^2 gives $R^2 \approx 1$ for all three plots. Putting this together with Figure 2-2, we can conclude that the inner loop adaptation step doesn't affect the representation learned by any layer except the head, and that the learned representations and features are mostly reused as is for the different tasks.

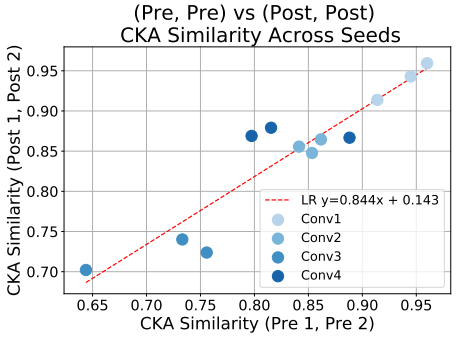


Figure A-3: We perform the same comparison as in Figure A-2, but with CKA instead. There is more variation in the similarity scores, but we still see a strong correlation between (Pre, Pre) and (Post, Post) comparisons, showing that representations do not change significantly over the inner loop.

A.2.5 MiniImageNet-5way-1shot Freezing and CCA Over Training

Figure A-4 shows that from early on in training, on MiniImageNet-5way-1shot, that the CCA similarity between activations pre and post inner loop update is very high for all layers but the head. We further see that the validation set accuracy suffers almost no decrease if we remove the inner loop updates and freeze all layers but the head. This shows that even early on in training, the inner loop appears to have minimal effect on learned representations and features. This supplements the results seen in Figure 2-3 on MiniImageNet-5way-5shot.

A.3 ANIL Algorithm: More Details

In this section, we provide more details about the ANIL algorithm, including an example of the ANIL update, implementation details, and further experimental results.

A.3.1 An Example of the ANIL Update

Consider a simple, two layer linear network with a single hidden unit in each layer: $\hat{y}(x; \boldsymbol{\theta}) = \theta_2(\theta_1 x)$. In this example, θ_2 is the *head*. Consider the 1-shot regression problem, where we have access to examples $\left\{(x_1^{(t)}, y_1^{(t)}), (x_2^{(t)}, y_2^{(t)})\right\}$ for tasks $t =$

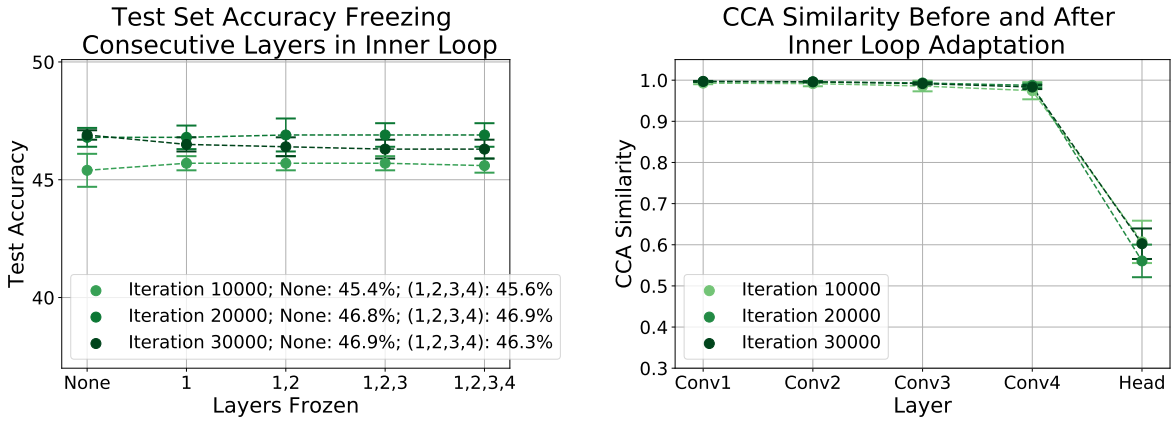


Figure A-4: **Inner loop updates have little effect on learned representations from early on in learning.** We consider freezing and representational similarity experiments for MiniImageNet-5way-1shot. We see that early on in training (from as few as 10k iterations in), the inner loop updates have little effect on the learned representations and features, and that removing the inner loop updates for all layers but the head have little-to-no impact on the validation set accuracy.

$1, \ldots, T$. Note that $(x_1^{(t)}, y_1^{(t)})$ is the (example, label) pair in the meta-training set (used for inner loop adaptation – support set), and $(x_2^{(t)}, y_2^{(t)})$ is the pair in the meta-validation set (used for the outer loop update – target set).

In the few-shot learning setting, we firstly draw a set of N tasks and labelled examples from our meta-training set: $\left\{(x_1^{(1)}, y_1^{(1)}), \ldots, (x_1^{(N)}, y_1^{(N)})\right\}$. Assume for simplicity that we only apply one gradient step in the inner loop. The inner loop updates for each task are thus defined as follows:

$$\theta_1^{(t)} \leftarrow \theta_1 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_1} \tag{A.1}$$

$$\theta_2^{(t)} \leftarrow \theta_2 - \frac{\partial L(\hat{y}(x_1^{(t)}; \boldsymbol{\theta}), y_1^{(t)})}{\partial \theta_2} \tag{A.2}$$

where $L(\cdot, \cdot)$ is the loss function, (e.g. mean squared error) and $\theta_i^{(t)}$ refers to a parameter after inner loop update for task t.

The task-adapted parameters for MAML and ANIL are as follows. Note how only

the head parameters change per-task in ANIL:

$$\boldsymbol{\theta}^{(t)}_{\text{MAML}} = \left[\theta^{(t)}_1, \theta^{(t)}_2\right] \tag{A.3}$$

$$\boldsymbol{\theta}^{(t)}_{\text{ANIL}} = \left[\theta_1, \theta^{(t)}_2\right] \tag{A.4}$$

In the outer loop update, we then perform the following operations using the data from the meta-validation set:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x^{(t)}_2; \boldsymbol{\theta}^{(t)}), y^{(t)}_2)}{\partial \theta_1} \tag{A.5}$$

$$\theta_2 \leftarrow \theta_2 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x^{(t)}_2; \boldsymbol{\theta}^{(t)}), y^{(t)}_2)}{\partial \theta_2} \tag{A.6}$$

Considering the update for θ_1 in more detail for our simple, two layer, linear network (the case for θ_2 is analogous), we have the following update for MAML:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x^{(t)}_2; \boldsymbol{\theta}^{(t)}_{\text{MAML}}), y^{(t)}_2)}{\partial \theta_1} \tag{A.7}$$

$$\hat{y}(x^{(t)}_2; \boldsymbol{\theta}^{(t)}_{\text{MAML}}) = \left(\left[\theta_2 - \frac{\partial L(\hat{y}(x^{(t)}_1; \boldsymbol{\theta}), y^{(t)}_1)}{\partial \theta_2} \right] \cdot \left[\theta_1 - \frac{\partial L(\hat{y}(x^{(t)}_1; \boldsymbol{\theta}), y^{(t)}_1)}{\partial \theta_1} \right] \cdot x_2 \right) \tag{A.8}$$

For ANIL, on the other hand, the update will be:

$$\theta_1 \leftarrow \theta_1 - \sum_{t=1}^{N} \frac{\partial L(\hat{y}(x^{(t)}_2; \boldsymbol{\theta}^{(t)}_{\text{ANIL}}), y^{(t)}_2)}{\partial \theta_1} \tag{A.9}$$

$$\hat{y}(x^{(t)}_2; \boldsymbol{\theta}^{(t)}_{\text{ANIL}}) = \left(\left[\theta_2 - \frac{\partial L(\hat{y}(x^{(t)}_1; \boldsymbol{\theta}), y^{(t)}_1)}{\partial \theta_2} \right] \cdot \theta_1 \cdot x_2 \right) \tag{A.10}$$

Note the lack of inner loop update for θ_1, and how we do not remove second order terms in ANIL (unlike in first-order MAML); second order terms still persist through

the derivative of the inner loop update for the head parameters.

A.3.2 ANIL Learns Almost Identically to MAML

We implement ANIL on MiniImageNet and Omniglot, and generate learning curves for both algorithms in Figure A-5. We find that learning proceeds almost identically for ANIL and MAML, showing that removing the inner loop has little effect on the learning dynamics.

A.3.3 ANIL and MAML Learn Similar Representations

We compute CCA similarities across representations in a MAML seed and an ANIL seed, and then plot these against the same MAML seed representation compared to a different MAML seed (and similarly for ANIL). We find a strong correlation between these similarities (Figure A-6), which suggests that MAML and ANIL are learning similar representations, despite their algorithmic differences. (ANIL and MAML are about as similar to each other as two ANILs are to each other, or two MAMLs are to each other.)

A.3.4 ANIL Implementation Details

Supervised Learning Implementation: We used the TensorFlow MAML implementation open-sourced by the original authors [55]. We used the same model architectures as in the original MAML paper for our experiments, and train models 3 times with different random seeds. All models were trained for 30000 iterations, with a batch size of 4, 5 inner loop update steps, and an inner learning rate of 0.01. 10 inner gradient steps were used for evaluation at test time.

Reinforcement Learning Implementation: We used the open source PyTorch implementation of MAML for RL [1], due to challenges encountered when running the open-sourced TensorFlow implementation from the original authors. We note

[1] `https://github.com/tristandeleu/pytorch-maml-rl`

that the results for MAML in these RL domains do not exactly match those in the original paper; this may be due to large variance in results, depending on the random initialization. We used the same model architecture as the original paper (two layer MLP with 100 hidden units in each layer), a batch size of 40, 1 inner loop update step with an inner learning rate of 0.1 and 20 trajectories for inner loop adaptation. We trained three MAML and ANIL models with different random initialization, and quote the mean and standard deviation of the results. As in the original MAML paper, for RL experiments, we select the best performing model over 500 iterations of training and evaluate this model at test time on a new set of tasks.

A.4 Further Results on the Network Head and Body

A.4.1 Training Regimes for the Network Body

We add to the results of Section 2.5.2 in the main text by seeing if training a head and applying that to the representations at test time (instead of the NIL algorithm) gives in any change in the results. As might be predicted by Section 2.5.1, we find no change the results.

More specifically, we do the following:

- We train MAML/ANIL networks as standard, and do standard test time adaptation.
- For multiclass training, we first (pre)train with multiclass classification, then throw away the head and freeze the body. We initialize a new e.g. 5-class head, and train that (on top of the frozen multiclass pretrained features) with MAML. At test time we perform standard adaptation.
- The same process is applied to multitask training.
- A similar process is applied to random features, except the network is initialized and then frozen.

Table A.1: **Test time performance is dominated by features learned, with no difference between NIL/MAML heads.** We see identical performances of MAML/NIL heads at test time, indicating that MAML/ANIL training leads to better learned features.

Method	MiniImageNet-5way-1shot	MiniImageNet-5way-5shot
MAML training-MAML head	46.9 ± 0.2	63.1 ± 0.4
MAML training-NIL head	48.4 ± 0.3	61.5 ± 0.8
ANIL training-ANIL head	46.7 ± 0.4	61.5 ± 0.5
ANIL training-NIL head	48.0 ± 0.7	62.2 ± 0.5
Multiclass pretrain-MAML head	38.4 ± 0.8	54.6 ± 0.4
Multiclass pretrain-NIL head	39.7 ± 0.3	54.4 ± 0.5
Multitask pretrain-MAML head	26.5 ± 0.8	32.8 ± 0.6
Multitask pretrain-NIL head	26.5 ± 1.1	34.2 ± 3.5
Random features-MAML head	32.1 ± 0.5	43.1 ± 0.3
Random features-NIL head	32.9 ± 0.6	43.2 ± 0.5

The results of this, along with the results from Table 2.6 in the main text is shown in Table A.1. We observe very little performance difference between using a MAML/ANIL head and a NIL head for each training regime. Specifically, task performance is purely determined by the quality of the features and representations learned during training, with task-specific alignment at test time being (i) unnecessary (ii) unable to influence the final performance of the model (e.g. multitask training performance is equally with a MAML head as it is with a NIL-head.)

A.4.2 Representational Analysis of Different Training Regimes

In Table A.2 we include results on using CCA and CKA on the representations learned by the different training methods. Specifically, we studied how similar representations of different training methods were to MAML training, finding a direct correlation with performance – training schemes learning representations most similar to MAML also performed the best. We computed similarity scores by averaging the scores over the first three conv layers in the body of the network.

Table A.2: **MAML training most closely resembles multiclass pretraining, as illustrated by CCA and CKA similarities**. On analyzing the CCA and CKA similarities between different baseline models and MAML (comparing across different tasks and seeds), we see that multiclass pretraining results in features most similar to MAML training. Multitask pretraining differs quite significantly from MAML-learned features, potentially due to the alignment problem.

Feature pair	CCA Similarity	CKA Similarity
(MAML, MAML)	0.51	0.83
(Multiclass pretrain, MAML)	0.48	0.79
(Random features, MAML)	0.40	0.72
(Multitask pretrain, MAML)	0.28	0.65

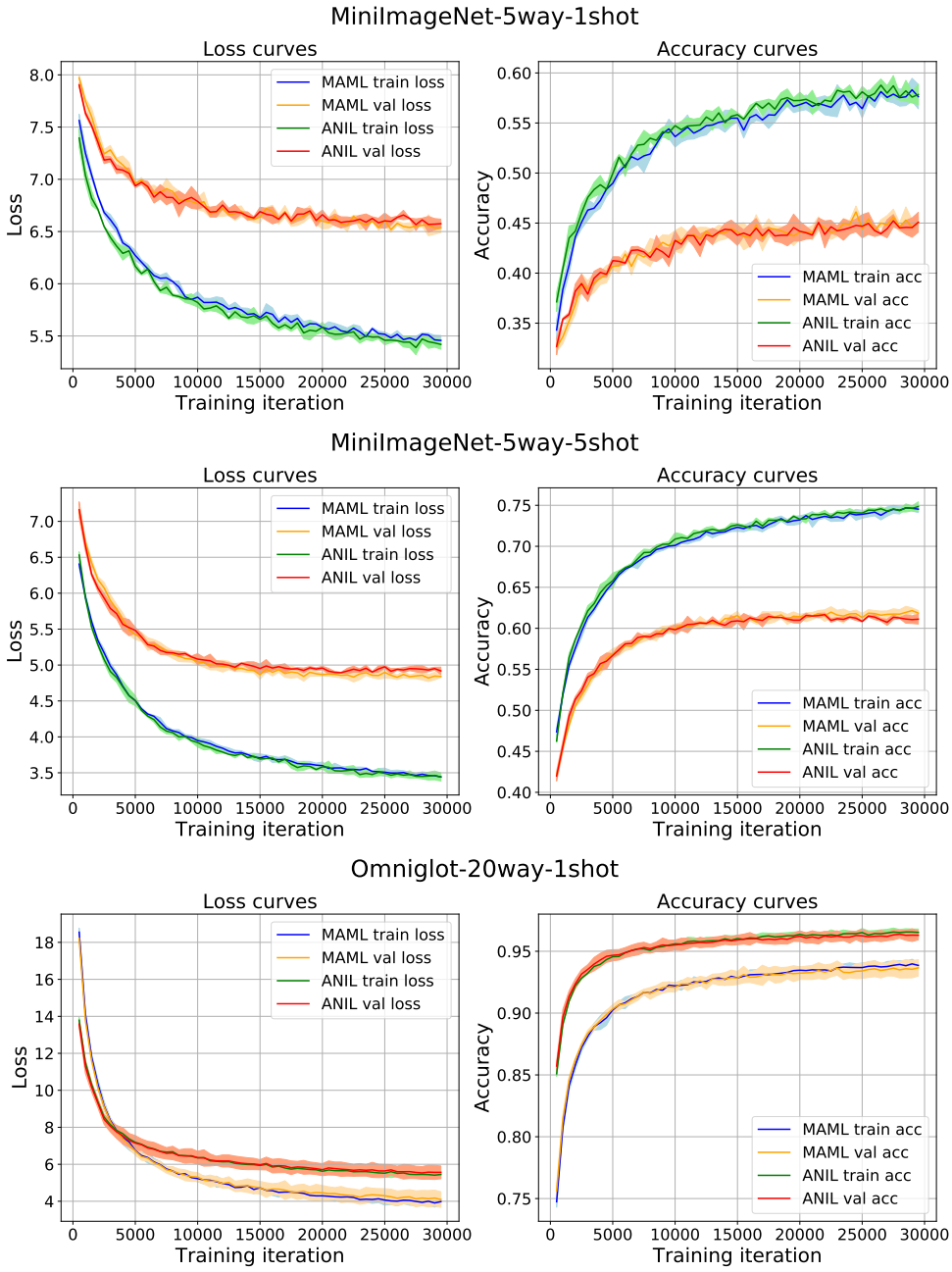


Figure A-5: **ANIL and MAML on MiniImageNet and Omniglot**. Loss and accuracy curves for ANIL and MAML on (i) MiniImageNet-5way-1shot (ii) MiniImageNet-5way-5shot (iii) Omniglot-20way-1shot. These illustrate how both algorithms learn very similarly over training.

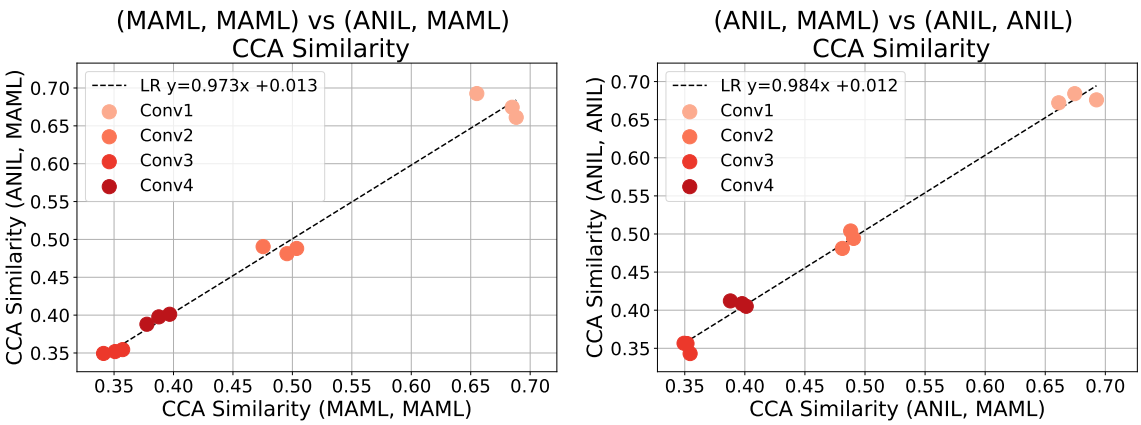


Figure A-6: **Computing CCA similarity across different seeds of MAML and ANIL networks suggests these representations are similar.** We plot the CCA similarity between an ANIL seed and a MAML seed, plotted against (i) the MAML seed compared to a different MAML seed (ii) the ANIL seed compared to a different ANIL seed. We observe a strong correlation of similarity scores in both (i) and (ii). This tells us that (i) two MAML representations vary about as much as MAML and ANIL representations (ii) two ANIL representations vary about as much as MAML and ANIL representations. In particular, this suggests that MAML and ANIL learn similar features, despite having significant algorithmic differences.

Appendix B

Additional Information and Results for Chapter 3

B.1 Augmentation Methods

In this section, we provide further details on the different augmentation strategies explored (existing and TaskAug), and visualize their operation.

B.1.1 Existing methods

Figures B-1-B-4 present examples following augmentation using the existing methods. We show only one lead for clarity; however, these operations will be applied to each lead.

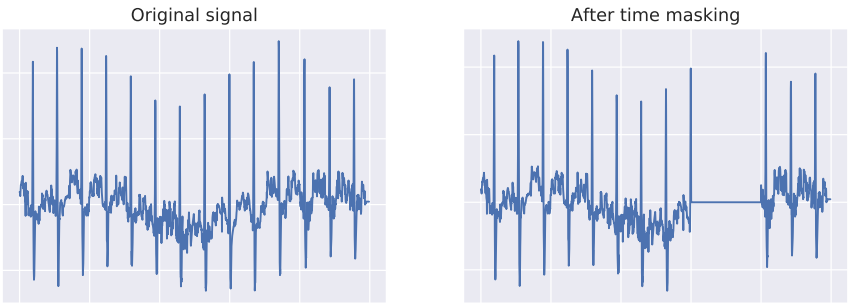


Figure B-1: Time Masking.

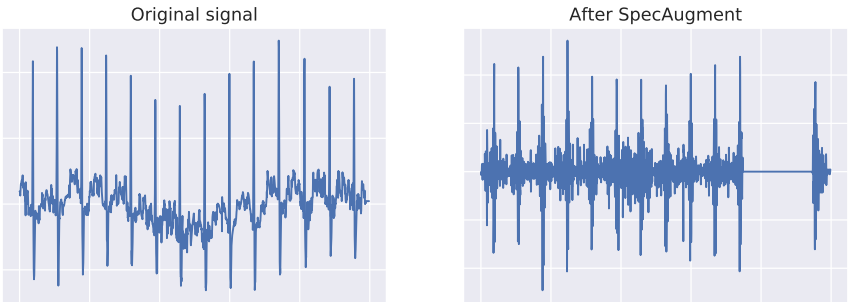


Figure B-2: SpecAugment.

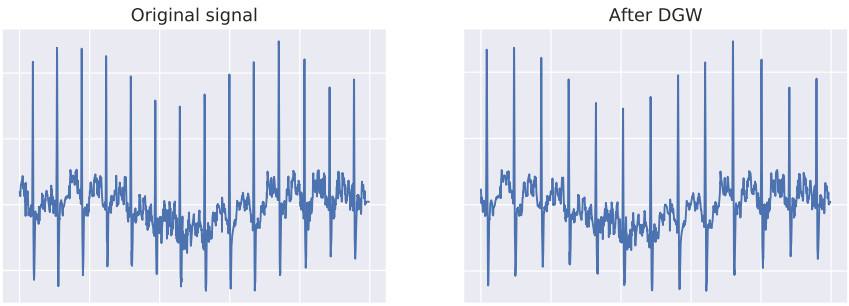


Figure B-3: Discriminative Guided Warping (DGW).

B.1.2 Further Details on TaskAug

Example Application of TaskAug. Suppose we have a one-stage TaskAug policy, $K = 1$, our augmentation set has two operations $\mathcal{S} = \{A_1, A_2\}$ which are $A_1 = \text{TimeMask}(x, y; \mu_0 = 0.2, \mu_1 = 0.1)$ and $A_2 = \text{Noise}(x, y; \mu_0 = 2.1, \mu_1 = 5.3)$, and the operation selection probability vector is $\pi = [0.9, 0.1]$ (that is, we select TimeMask with probability 0.9, and noise with probability 0.1). Now consider applying TaskAug to a (data, label) pair $(x, 1)$, i.e., the label is 1. We follow these steps:

1. Obtain a reparameterizable sample u from Categorical($[0.9, 0.1]$): let this be $u = [0.75, 0.25]$.
2. Find $i = \arg\max u$; in this case, $i = 1$.
3. Select the operation A_1, i.e. TimeMask.
4. Compute the masking strength based on the label. Recall this is defined as $s = y\mu_1 + (1 - y)\mu_0$, so $s = 1 \times 0.1 + (1 - 1) \times 0.2 = 0.1$.
5. Apply time-masking with strength 0.1 to x, generating $\hat{x}$.
6. Scale this by $\frac{u_1}{\texttt{stop\_grad}(u_1)}$ to generate $\tilde{x}$.

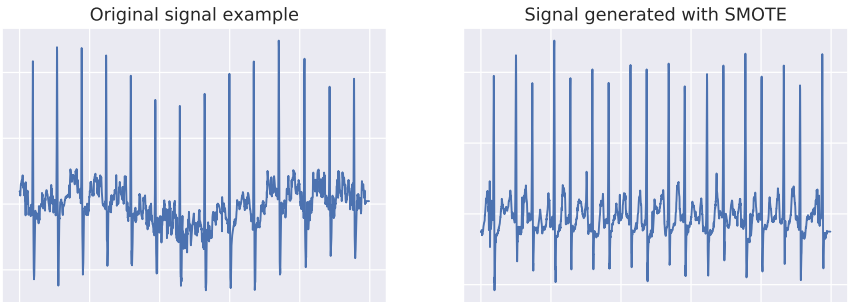


Figure B-4: SMOTE.

Algorithm 3 Optimizing TaskAug parameters.

Initialize base model parameters θ and TaskAug parameters ϕ
for $t = 1, \ldots, T$ **do**
  Compute training loss, $\mathcal{L}_T(\theta)$
  Compute $\frac{\partial \mathcal{L}_T}{\partial \theta}$
  Update $\theta \leftarrow \theta - \eta_\theta \frac{\partial \mathcal{L}_T}{\partial \theta}$
  if $t \ \% \ P == 0$ **then**
    Set $\hat{\theta} = \theta$
    Compute the validation loss, $\mathcal{L}_V(\hat{\theta})$
    Compute $\frac{\partial \mathcal{L}_V}{\partial \hat{\theta}}$
    Approximate $\frac{\partial \hat{\theta}}{\partial \phi}$ using Equation 3.5.
    Compute the derivative $\frac{\partial \mathcal{L}_V}{\partial \phi} = \frac{\partial \mathcal{L}_V}{\partial \hat{\theta}} \times \frac{\partial \hat{\theta}}{\partial \phi}$ using the previous two steps.
    Update $\phi \leftarrow \phi - \eta_\phi \frac{\partial \mathcal{L}_V}{\partial \phi}$
  end if
end for

Choice of P in optimization algorithm. As detailed in the main text, there are many learnable parameters in TaskAug, and we use gradient-based optimization to learn these jointly with the base model parameters – the algorithm to do so is reproduced here, Algorithm 3.

An important hyperparameter in this algorithm is P. This influences how many 'inner' gradient steps (to the base model) we perform before an 'outer' gradient step (to the TaskAug parameters). There is a tradeoff here: if P is too small, then applying the IFT to approximate $\frac{\partial \hat{\theta}}{\partial \phi}$ will result in a poor approximation [113]; if P is too large, then updates to the policy parameters will have little effect on model parameters since the base model has already reached minimal training loss (and may start to overfit). In our experiments, we find that $P > 5$ suffered from this second problem, and $P = 1$

was sometimes unstable due to the first problem. In general, $P = 1$ worked well at small sample sizes ($N = 1000$), and $P = 5$ worked better at $N = 2500$ and $N = 5000$.

Augmentation operations

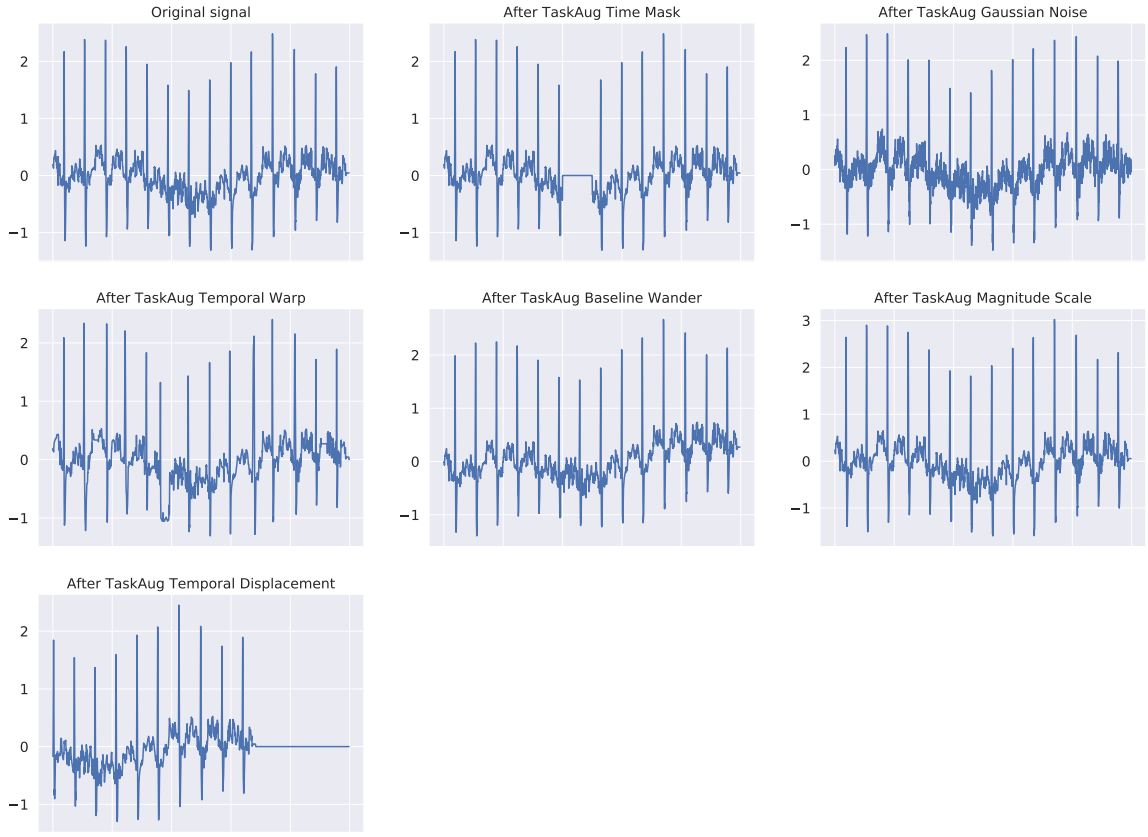


Figure B-5: Examples of the different operations used in TaskAug.

Figure B-5 shows the different operations used in TaskAug. We show only one lead for clarity; however, these operations will be applied to each lead. We now provide more details on the implementation of these operations in our experiments.

- **TimeMask.** As with the existing TimeMask strategies, we randomly select a contiguous portion of the signal to set to zero. We set 10% of the signal to zero in our implementation. This parameter is not optimized.
- **Gaussian Noise.** IID Gaussian noise is added to the signal. This is formed as follows. We first compute the standard deviation of each lead of the signal: let us denote this as σ. Then, the noise added to each sample of the signal is

expressed as: $\epsilon = 0.25 \times \sigma \times \texttt{sigmoid}(s) \times \mathcal{N}(0, 1)$, where s is the learnable strength parameter, initialized to 0. The coefficient 0.25 was found by visual inspection of some augmented examples, and observing that this allowed flexible augmentations to be generated without overwhelming the signal with noise.

- **Temporal warping.** The signal is warped with a random, diffeomorphic temporal transformation. To form this, we sample from a Gaussian with zero mean, and a fixed variance $100 \times s^2$, where s is the learnable strength parameter (initialized to 1), at each temporal location, to generate a length T dimensional random velocity field. This velocity field is then integrated (following the scaling and squaring numerical integration routine used by [9, 10]). This resulting displacement field is then smoothed with a Gaussian filter to generate the smoothed temporal displacement field. This field represents the number of samples each point in the original signal is translated in time. The field is then used to transform the signal, translating each channel in the same way (i.e., the field is the same across channels).
- **Baseline wander.** We firstly form a wander amplitude by computing: $A = 0.25 \times \texttt{sigmoid}(s) \times \texttt{Uniform}(0, 1)$, where again s is a learnable strength parameter. Then, we compute the frequency and phase of the sinusoidal offset. The frequency is computed as: $f = \frac{20 \times \texttt{Uniform}(0,1) + 10}{60}$, based on the approximate number of breaths per minute for an adult. The phase is: $\phi = 2\pi \times \texttt{Uniform}(0, 1)$. Then, the sinusoidal offset is computed as: $A \sin(ft + \phi)$.
- **Magnitude scaling.** We scale the entire signal by a random magnitude given by $\texttt{sigmoid}(s) \times \texttt{Uniform}(0.75, 1.25)$, where s is a learnable strength parameter, initialized to 0.
- **Temporal displacement.** We shift the entire signal in time, padding with zeros where required. Our implementation directly generates a displacement field (as with temporal warping) and uses the spatial transformation from [9, 10] to transform the signal. This allows the operation to be differentiable, and for us to learn the displacement strength s. The displacement magnitude is a Uniform distribution on $[-100 \times s^2, 100 \times s^2]$, with the strength being initialized to 0.5.

B.2 Dataset Details

We provide more details about the three datasets.

B.2.1 Dataset A

The labels for RVH and AFib were assigned to each example based on whether relevant diagnostic statements were present in either a clinician's read of the ECG, or a machine read of the ECG.

For RVH, there were six diagnostic statements that led to a positive label being assigned: "right ventricular hypertrophy", "biventricular hypertrophy", "combined ventricular hypertrophy", "right ventricular enlargement", "rightventricular hypertrophy", "biventriclar hypertrophy".

For AFib, there were nine such statements: "atrial fibrillation with rapid ventricular response", "atrial fibrillation with moderate ventricular response", "fibrillation/flutter", "atrial fibrillation with controlled ventricular response", "afib", "atrial fib", "afibrillation", "atrial fibrillation", "atrialfibrillation".

Preprocessing. ECGs were sampled at 250 Hz for 10 seconds, resulting in a 2500×12 tensor for all 12 leads, per-ECG. We normalized the signals by dividing by 1000. Other forms of normalization for this dataset (e.g., z-scoring) resulted in some abnormally large/small values.

B.2.2 Dataset B

The four labels are obtained by aggregating relevant sets of diagnostic statements – we refer the reader to the PTB-XL paper [191] for further details. Of relevance here is that certain labels, such as MI, contain a small number of distinct diagnostic statements (3), potentially suggesting why many augmentation strategies can help – it is a fine-grained task. Others (such as CD) are much broader, covering many more diagnostic statements.

Preprocessing. ECGs in the dataset are sampled at 500 Hz for 10 seconds; we downsample these by a factor of 2 for consistency with Dataset A and C, resulting in a 2500×12 tensor for all 12 leads, per-ECG. Normalization involved z-scoring, following the code provided with the dataset.

B.2.3 Dataset C

The hemodynamics prediction cohort consists of patients who had an ECG and right heart catheterization procedure on the same day. The catheterization procedure measures hemodynamics variables including the pulmonary capillary wedge pressure (PCWP) and cardiac output (CO), and these are used to form the prediction targets. We consider inferring abnormally low Cardiac Output (less than 2.5 L/min), and abnormally high Pulmonary Capillary Wedge Pressure (greater than 20 mmHg).

Preprocessing. ECGs were sampled at 250 Hz for 10 seconds, resulting in a 2500×12 tensor for all 12 leads, per-ECG. We normalized the signals by dividing by 1000. Other forms of normalization for this dataset (e.g., z-scoring) resulted in some abnormally large/small values, so we opted for the division-based normalization.

B.3 Experiments

In this section, we provide further experimental details. We first provide implementation details, and then outline additional experimental results including: Results for AUPRC in the low-sample ($N = 1000$) regime, performance on Datasets A and B in the high-sample regime, performance on Dataset B in an additional low sample regime ($N = 500$ data points), interpretation of the TaskAug policy for RVH, a study of the impact of optimizing policy parameters across different sample size regimes, and a study of the impact of class-specific magnitudes across different sample size regimes.

B.3.1 Implementation details

Network architecture. In all experiments, we use a 1D CNN based on a ResNet-18 [77] architecture. This model has convolutions with a kernel size of 15, and stride 2 (informed by the temporal window we want the convolutions to operate over). The blocks in the ResNet architecture have convolutional layers with 32, 64, 128, and 256 channels respectively. The output after the final block is average pooled in the temporal dimension, and then a linear layer is applied to predict the probability of the positive class.

Optimization settings. As discussed, we used Adam with a learning rate of 1e-3 for all methods, given that this resulted in stable training across all settings. When optimizing the TaskAug policy parameters, we used RMSprop with a learning rate of 1e-2, following [113].

Computational information. All models and training were implemented in PyTorch and run on a single NVIDIA V100 GPU.

B.3.2 Additional results

AUPRC results at 1000 samples. As discussed in Section 3.5.2, the improvements in AUROC are not always statistically significant. Given that some of the labels are very low prevalence (RVH: 1%, AFib: 5%, low CO: 4%), we evaluate the AUPRC in the low-sample regime, which provides additional information about model performance. Results are shown in Tables B.1, B.2, and B.3. We observe that for the low prevalence RVH, AFib, and Low CO tasks, TaskAug obtains statistically significant improvements in performance. On Dataset A tasks (RVH and AFib), it is the only method to do so.

	RVH	AFib
NoAugs	7.4 ± 1.3	<u>21.2 ± 2.0</u>
TaskAug	**10.8 ± 0.8***	**27.3 ± 1.8***
SMOTE	9.7 ± 1.2	21.0 ± 2.2
DGW	7.1 ± 0.9	19.4 ± 2.3
SpecAug	<u>10.6 ± 1.2</u>	21.1 ± 2.0
TimeMask	10.1 ± 1.5	20.3 ± 2.3

Table B.1: Mean and standard error of AUPRC for various data augmentation strategies when detecting cardiac abnormalities on Dataset A. We consider a low-sample regime with a development set of 1000 data points. The best-performing method is bolded, and the second best is underlined, and * indicates statistically significant improvement at the $p < 0.05$ level. TaskAug is the only method to obtain significant improvements in performance on both tasks.

	MI	HYP	STTC	CD
NoAugs	59.2±2.1	53.1±1.7	66.9±2.5	**67.3±1.1**
TaskAug	**63.1±1.7**	**55.2±0.9**	68.7±1.3	66.8±1.2
SMOTE	<u>62.0±1.6</u>	41.2±2.9	65.9±1.0	62.7±1.1
DGW	61.1±1.2	53.9±1.6	67.9±1.1	64.7±2.6
SpecAug	61.7±1.6	<u>54.5±1.5</u>	<u>68.8±1.5</u>	65.8±1.4
TimeMask	60.3±1.3	52.8±1.8	**68.8±1.2**	**70.1±1.3**

Table B.2: Mean and standard error of AUPRC for various data augmentation strategies on detecting cardiac abnormalities on Dataset B. We consider a low-sample regime with a development set of 1000 data points. The best-performing method is bolded, and the second best is underlined, and * indicates statistically significant improvement at the $p < 0.05$ level.

	Low CO	High PCWP: $N = 1000$	High PCWP: All Data
NoAugs	7.2 ± 0.4	<u>42.5 ± 0.8</u>	49.7 ± 0.8
TaskAug	**8.8 ± 0.6***	**43.5 ± 0.9**	**50.8 ± 0.8**
SMOTE	**8.8 ± 0.6***	41.9 ± 0.7	46.9 ± 0.7
DGW	<u>8.1 ± 0.7</u>	41.2 ± 0.7	49.7 ± 1.0
SpecAug	7.8 ± 0.4	42.3 ± 1.1	<u>50.3 ± 0.8</u>
TimeMask	8.0 ± 0.5	42.4 ± 0.7	50.1 ± 0.9

Table B.3: Mean and standard error of AUPRC for various data augmentation strategies for the hemodynamics inference task in Dataset C. We consider a low-sample regime with a development set of 1000 data points. The best-performing method is bolded, and the second best is underlined, and * indicates statistically significant improvement at the $p < 0.05$ level. TaskAug is the one of only two methods to obtain significant improvements in performance on the low CO detection task.

Results at higher sample regimes. Tables B.4-B.7 show AUROC for the different augmentation methods on the tasks from Datasets A and B. We observe that augmentations are less effective at higher samples. Particularly when the development set sizes are 2500 and 5000 datapoints, we observe that the improvement with using augmentations (over the NoAugs baseline) with any of the methods is quite small, and nearly always less than 1% AUROC. This suggests that in general, augmentations are less useful at these higher data regimes.

	RVH	AFib
NoAugs	86.1±0.9	89.0 ± 0.4
TaskAug	**86.9±0.9**	89.1 ± 0.4
SMOTE	85.5±1.3	89.1 ± 0.5
DGW	84.8±1.3	88.4 ± 0.5
SpecAug	83.3±1.8	**89.1 ± 0.3**
TimeMask	85.8±1.1	88.2 ± 0.4

Table B.4: Mean and standard error of AUROC for augmentation methods on Dataset A tasks with a development set of 2500 data points. The best performing method is bolded, and the second best is underlined.

	RVH	AFib
NoAugs	90.6±0.6	92.6±0.2
TaskAug	90.6±0.4	**92.8±0.1**
SMOTE	89.8±0.6	92.6±0.2
DGW	**90.8±0.5**	92.5±0.2
SpecAug	90.5±0.8	92.7±0.1
TimeMask	89.4±0.7	92.6±0.2

Table B.5: Mean and standard error of AUROC for augmentation methods on Dataset A tasks with a development set of 5000 data points. The best performing method is bolded, and the second best is underlined.

	MI	HYP	STTC	CD
NoAugs	84.5±0.5	86.4±0.4	89.7±0.3	85.8±0.3
TaskAug	**86.1±0.5**	86.2±0.4	89.7±0.3	86.6±0.4
SMOTE	84.7±0.7	81.9±1.3	88.7±0.4	85.5±0.6
DGW	84.1±0.5	85.9±0.6	89.5±0.3	86.2±0.3
SpecAug	84.6±0.8	86.2±0.6	**90.2±0.3**	86.8±0.6
TimeMask	85.7±0.4	**86.6±0.3**	90.1±0.1	**87.0±0.7**

Table B.6: Mean and standard error of AUROC for augmentation methods on Dataset B tasks with a development set of 2500 data points. The best performing method is bolded, and the second best is underlined.

	MI	HYP	STTC	CD
NoAugs	89.4±0.3	88.2±0.2	91.0±0.3	89.3±0.4
TaskAug	89.4±0.3	88.3±0.2	**91.6±0.2**	**90.0±0.2**
SMOTE	86.6±0.7	86.7±0.4	90.6±0.3	88.0±0.3
DGW	88.6±0.3	88.0±0.2	91.3±0.1	89.3±0.2
SpecAug	**89.5±0.2**	88.4±0.4	**91.6±0.2**	89.9±0.2
TimeMask	89.3±0.3	**88.6±0.2**	**91.6±0.2**	89.8±0.2

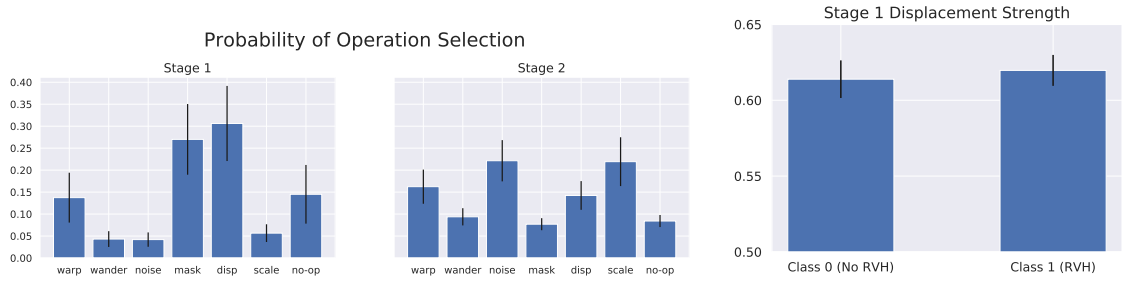
Table B.7: Mean and standard error of AUROC for augmentation methods on Dataset B tasks with a development set of 5000 data points. The best performing method is bolded, and the second best is underlined.

	MI	HYP	STTC	CD
NoAugs	74.4 ± 0.9	**81.9 ± 0.8**	85.2 ± 0.5	78.9 ± 1.2
TaskAug	**78.4 ± 0.5**	81.5 ± 1.2	86.2 ± 0.4	**80.7 ± 0.6**
SMOTE	75.7 ± 1.2	79.2 ± 1.5	85.5 ± 0.3	78.6 ± 1.5
DGW	78.2 ± 0.6	78.7 ± 1.2	82.0 ± 1.3	79.0 ± 0.9
SpecAug	77.8 ± 0.7	81.0 ± 0.6	86.3 ± 0.4	79.3 ± 1.1
TimeMask	77.8 ± 1.0	80.9 ± 1.3	**86.6 ± 0.5**	80.3 ± 0.8

Table B.8: Mean and standard error of AUROC for augmentation methods on Dataset B tasks with a development set of 500 data points. The best performing method is bolded, and the second best is underlined.

Results on Dataset B at $N = 500$. Table B.8 shows AUROC for the different augmentation methods in an additional low sample regime, with $N = 500$. We see that the maximum improvement over the NoAugs baseline by any augmentation strategy is greater in this regime than it was at $N = 1000$ (see Table 3.2). Given that the prevalence of these tasks is relatively high, we see more significant performance improvements in the $N = 500$ regime.

Figure B-6: **TaskAug policy for detecting Right Ventricular Hypertrophy.** The learned TaskAug policy: probability of selecting each transformation in both augmentation stages and the optimized displacement strengths in the first stage. We show the mean/standard error of the learned parameter values over 15 runs. Temporal operations (masking and displacement) have a high probability of selection in Stage 1, which is sensible since these operations are likely to be label preserving (RVH is typically detected based on the relative magnitudes of portions of beats in the ECG). We see that both positive and negative classes have similar optimized displacement augmentation strengths – we do not expect displacement to impact the class label differently for the two classes, so this is sensible.



(a) Operation selection probabilities

(b) Displacement strengths

Interpreting the RVH policy. We visualize the TaskAug policy for RVH in Figure B-6. We observe high probability assigned to selecting two temporal operations in stage 1, namely masking and displacement. Relative magnitudes of different portions of the ECG affect the RVH label, so temporal operations having higher probability of selection is sensible since they are more likely to be label preserving than operations that change the relative magnitudes of different parts of the ECG. We examine the learned strengths for the displacement operation in Stage 1, Figure B-6b, and we see that there is little differentiation on a per-class basis. This is sensible, since we do not expect displacement of the signal in time to affect the RVH label for differently for the positive and negative classes.

Further study on the impact of optimizing augmentations. As shown in the main text, Figure 3-5, optimizing the policy parameters improves performance over keeping them fixed at their initial values. In Figure B-7, we study this effect across different dataset sizes and find that the optimization has the most impact in the low sample regime, but still results in improvements even at higher samples. This could be due to the fact that at higher samples, augmentations boost performance less in general, so the specific parameter settings in TaskAug also have less impact.

Further study on the impact of class-specific magnitudes. As shown in the main text, Figure 3-6, optimizing class-specific magnitudes improves over learning one magnitude parameter for each class. Figure B-8 studies this effect across different dataset sizes and we see that the class-specific parameters improve performance at all dataset sizes, but the improvement is most clearly seen at low samples. Similarly with the optimization of augmentation parameters, this could be due to the fact that at higher samples, augmentations boost performance less in general, so the class-specific parameterization in TaskAug has less impact.

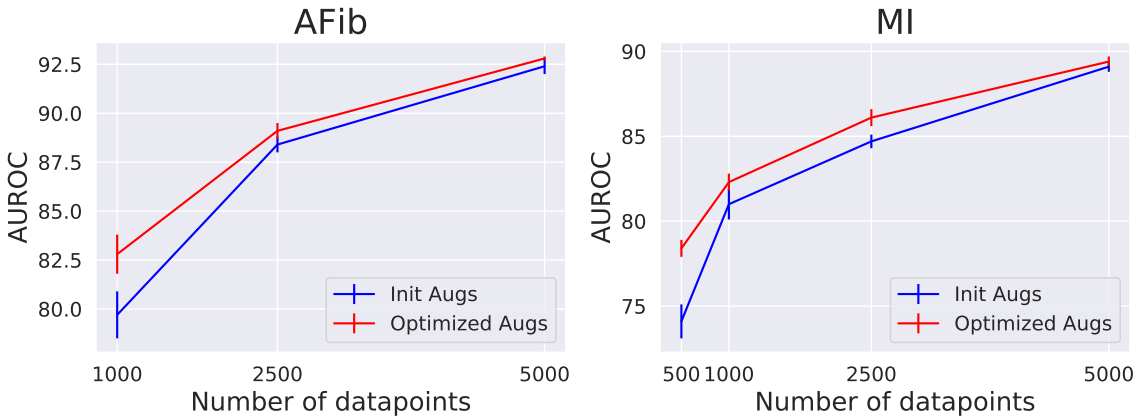


Figure B-7: **Studying performance when we do not optimize the policy parameters in TaskAug.** We show the mean/standard error of AUROC over 15 runs for AFib and over 5 runs for MI. We see that optimizing the policy parameters results in noticeable improvements in performance over keeping the policy parameters at their initial values (InitAugs). However, the impact of optimizing the parameters is reduced at larger dataset sizes, possibly due to the fact that augmentations are inherently less useful at higher sample regimes.

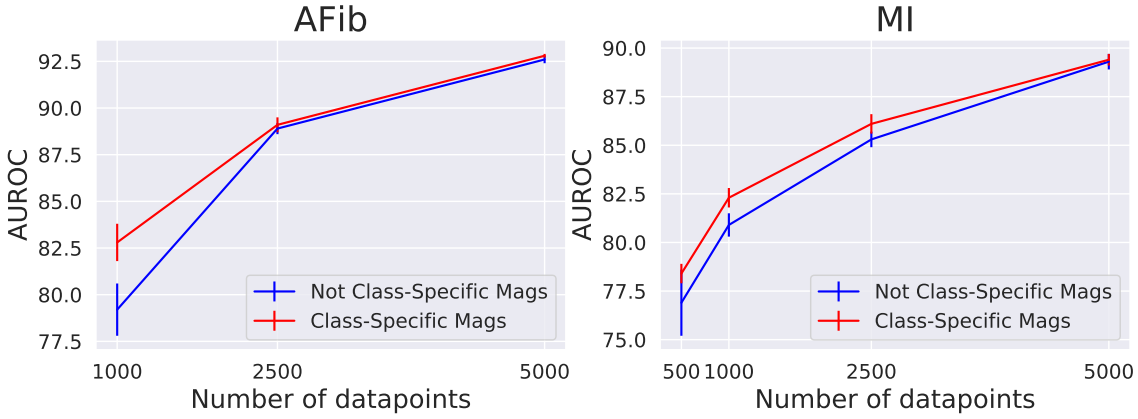


Figure B-8: **Studying performance when we do not have class-specific magnitude parameters in TaskAug.** We show the mean/standard error of AUROC over 15 runs for AFib and over 5 runs for MI. Class-specific magnitude parameters improve performance most in the low sample regime. At higher samples, this impact is reduced, possibly due to the fact that augmentations are inherently less useful at higher sample regimes.

Appendix C

Additional Information and Results for Chapter 4

C.1 Notation and Acronyms

PT	Pre-training
FT	Fine-tuning
AUROC (AUC)	Area Under Receiver-Operator Characteristic
$\bullet$	Placeholder for either PT or FT
$\mathcal{X}$	Input domain to models
$\boldsymbol{x}$	Model input
$\mathcal{Y}_\bullet$	True output space (e.g., space of labels)
y	Label
$\hat{\mathcal{Y}}_\bullet$	Prediction output space of a model
$\hat{y}$	Predicted Label
Θ	Parameter space for feature extractors of models
$\boldsymbol{\theta}$	Feature extractor parameters
$\Psi_\bullet$	Parameter space for prediction head of model (output layer)
$\boldsymbol{\psi}_\bullet$	Head parameters
Φ	Space of meta-parameters
ϕ	Meta-parameters
$f_\bullet : \mathcal{X}; \Theta, \Psi_\bullet \to \hat{\mathcal{Y}}_\bullet$	General parameteric model satisfying compositional structure $f_\bullet = f_\bullet^{(\text{head})} \circ f^{(\text{feat})}$
$f^{(\text{feat})}(\cdot; \boldsymbol{\theta} \in \Theta)$	Feature extractor that is transferable across learning stages (e.g., PT to FT)
$f_\bullet^{(\text{head})}(\cdot; \psi \in \Psi_\bullet)$	Output 'head' of a model that is stage-specific and not transferable.
$\mathcal{D}_\bullet$	General data distribution or dataset
$\mathcal{L}_\bullet : \hat{\mathcal{Y}}_\bullet \times \mathcal{Y}_\bullet \to \mathbb{R}$	Loss function
$\mathcal{L}_{\text{CE}}$	Example loss function: cross-entropy
$L_\bullet(\boldsymbol{\theta}, \boldsymbol{\psi}_\bullet; \mathcal{D}_\bullet)$	Expected loss over a data distribution $\mathbb{E}_{\mathcal{D}_\bullet}\left[\mathcal{L}_\bullet(f_\bullet(\boldsymbol{x}_\bullet; \boldsymbol{\theta}, \boldsymbol{\psi}_\bullet), y_\bullet)\right]$.
$\text{Alg}_\bullet$	Learning algorithm used for optimization (e.g., stochastic gradient descent)
$g(\phi)$	Meta-parameter optimization objective $L_{\text{FT}}\Big(\text{Alg}_{\text{FT}}\big(\text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi}), \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}}\big); \mathcal{D}_{\text{FT}}\Big)$
$\phi^{(\text{opt})}$	Optimal meta-parameters satisfying $\boldsymbol{\phi}^{(\text{opt})} = \text{argmin}_{\boldsymbol{\phi} \in \Phi}\, g(\boldsymbol{\phi})$
$\boldsymbol{\theta}_{\text{PT}}^*(\boldsymbol{\phi})$	PT best response values satisfying $\boldsymbol{\theta}_{\text{PT}}^*(\boldsymbol{\phi}) = \text{Alg}_{\text{PT}}(\boldsymbol{\theta}_{\text{PT}}^{(0)}, \boldsymbol{\psi}_{\text{PT}}^{(0)}; \mathcal{D}_{\text{PT}}, \boldsymbol{\phi})$
$\boldsymbol{\theta}_{\text{FT}}^*(\boldsymbol{\phi}), \psi_{\text{FT}}^*(\boldsymbol{\phi})$	FT best response values satisfying $\boldsymbol{\theta}_{\text{FT}}^*(\boldsymbol{\phi}), \psi_{\text{FT}}^*(\boldsymbol{\phi}) = \text{Alg}_{\text{FT}}(\boldsymbol{\theta}_{\text{PT}}^*(\boldsymbol{\phi}), \boldsymbol{\psi}_{\text{FT}}^{(0)}; \mathcal{D}_{\text{FT}})$
$\frac{\partial g}{\partial \phi}$	Gradient w.r.t. meta-parameters, which we compute for gradient-based optimization of ϕ
$\left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]$	Shorthand to representation concatenation of parameter vectors.
$\frac{\partial L_{\text{FT}}}{\partial \left[\boldsymbol{\theta}_{\text{FT}}, \quad \boldsymbol{\psi}_{\text{FT}}\right]}$	FT loss gradient: first term in meta-parameter gradient.
$\frac{\partial \text{Alg}_{\text{FT}}}{\partial \boldsymbol{\theta}_{\text{PT}}}$	FT best response Jacobian: second term in meta-parameter gradient.
$\frac{\partial \text{Alg}_{\text{PT}}}{\partial \boldsymbol{\phi}}$	PT best response Jacobian: third term in meta-parameter gradient.
K	Number of steps we unroll in FT to compute FT best response Jacobian.
P	Number of PT steps before each meta-parameter update.
$\texttt{copy}(\theta)$	Make a copy of the parameters θ such that gradients do not flow through (like a stop-gradient).
$\mathcal{D}_{\text{FT}}^{(\text{tr})}$	Training split of the FT data set, used during meta-parameter learning for updating the FT parameters.
$\mathcal{D}_{\text{FT}}^{(\text{val})}$	Validation split of the FT data set, used during meta-parameter learning for optimizing meta-parameters.
$\mathcal{D}_{\text{FT}}^{(\text{Meta})}$	FT data available at PT time for meta-parameter learning. We have that $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}^{(\text{tr})} \cup \mathcal{D}_{\text{FT}}^{(\text{val})} \subseteq \mathcal{D}_{\text{FT}}^{(\text{all})}$.
IFT	Implicit Function Theorem
GIN	Graph Isomorphism Network
ECG	Electrocardiogram
η_{PT}	learning rate for PT
η_{FT}	learning rate for FT
η_{V}	learning rate for meta parameters

Table C.1: Notation

C.2 Our Algorithm: Further Details

Algorithm 4 Gradient-based algorithm to learn meta-parameters, incorporating other practical details not present in the main paper description. Notation defined in Table C.1. Note that vector-Jacobian products (VJPs) can be efficiently computed by standard autodifferentiation.

1: Initialize PT parameters $\boldsymbol{\theta}_{\mathrm{PT}}^{(\mathrm{init})}, \boldsymbol{\psi}_{\mathrm{PT}}^{(\mathrm{init})}, \boldsymbol{\psi}_{\mathrm{FT}}^{(\mathrm{init})}$ and meta-parameters $\boldsymbol{\phi}^{(0)}$
2: **for** $n = 1, \ldots, N$ iterations **do**
3: Initialize $\boldsymbol{\theta}_{\mathrm{PT}}^{(0)} = \boldsymbol{\theta}_{\mathrm{PT}}^{(\mathrm{init})}$ and $\boldsymbol{\psi}_{\mathrm{PT}}^{(0)} = \boldsymbol{\psi}_{\mathrm{PT}}^{(\mathrm{init})}$.
4: **for** $p = 1, \ldots, P$ PT iterations **do**
5: $\left[\boldsymbol{\theta}_{\mathrm{PT}}^{(p)}, \boldsymbol{\psi}_{\mathrm{PT}}^{(p)}\right] = \left[\boldsymbol{\theta}_{\mathrm{PT}}^{(p-1)}, \boldsymbol{\psi}_{\mathrm{PT}}^{(p-1)}\right] - \eta_{\mathrm{PT}} \left.\frac{\partial L_{\mathrm{PT}}}{\partial \left[\boldsymbol{\theta}_{\mathrm{PT}}, \boldsymbol{\psi}_{\mathrm{PT}}\right]}\right|_{\boldsymbol{\theta}_{\mathrm{PT}}^{(p-1)}, \boldsymbol{\psi}_{\mathrm{PT}}^{(p-1)}}$ `# Unrolled step of` $\mathrm{Alg}_{\mathrm{PT}}$
6: **end for**
7: **if** $n < N_{\mathrm{warmup}}$ **then**
8: Update PT initialization by setting: $\boldsymbol{\theta}_{\mathrm{PT}}^{(\mathrm{init})} = \boldsymbol{\theta}_{\mathrm{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\mathrm{PT}}^{(\mathrm{init})} = \boldsymbol{\psi}_{\mathrm{PT}}^{(P)}$
9: Skip meta-parameter update and continue
10: **end if**
11: Initialize FT parameters $\boldsymbol{\psi}_{\mathrm{FT}}^{(0)} = \boldsymbol{\psi}_{\mathrm{FT}}^{(\mathrm{init})}$ and $\boldsymbol{\theta}_{\mathrm{FT}}^{(0)} = \mathtt{copy}(\boldsymbol{\theta}_{\mathrm{PT}}^{(P)})$.
12: Approximate $\boldsymbol{\theta}_{\mathrm{FT}}^{*}, \boldsymbol{\psi}_{\mathrm{FT}}^{*}$ using (4.4), with $\mathcal{D}_{\mathrm{FT}}^{(\mathrm{tr})}$.
13: Compute $g_1 = \left.\frac{\partial L_{\mathrm{FT}}}{\partial \left[\boldsymbol{\theta}_{\mathrm{FT}}, \quad \boldsymbol{\psi}_{\mathrm{FT}}\right]}\right|_{\boldsymbol{\theta}_{\mathrm{FT}}^{*}, \boldsymbol{\psi}_{\mathrm{FT}}^{*}}$, using $\mathcal{D}_{\mathrm{FT}}^{(\mathrm{val})}$. `# FT Loss gradient`
14: Compute VJP $g_2 = g_1 \left.\frac{\partial \mathrm{Alg}_{\mathrm{FT}}}{\partial \boldsymbol{\theta}_{\mathrm{PT}}}\right|_{\boldsymbol{\theta}_{\mathrm{PT}}^{(P)}, \boldsymbol{\psi}_{\mathrm{FT}}^{(0)}}$ using the unrolled learning step from line 12, and $\mathcal{D}_{\mathrm{FT}}^{(\mathrm{tr})}$.
15: Approximate VJP $\left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}} = g_2 \left.\frac{\partial \mathrm{Alg}_{\mathrm{PT}}}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ using IFT (4.3).
16: $\boldsymbol{\phi}^{(n)} = \boldsymbol{\phi}^{(n-1)} - \eta_{\mathrm{V}} \left.\frac{\partial g}{\partial \boldsymbol{\phi}}\right|_{\boldsymbol{\phi}^{(n-1)}}$ `# Update meta-parameters`
17: Update PT initialization by setting: $\boldsymbol{\theta}_{\mathrm{PT}}^{(\mathrm{init})} = \boldsymbol{\theta}_{\mathrm{PT}}^{(P)}$ and $\boldsymbol{\psi}_{\mathrm{PT}}^{(\mathrm{init})} = \boldsymbol{\psi}_{\mathrm{PT}}^{(P)}$.
18: Update FT initialization by setting: $\boldsymbol{\psi}_{\mathrm{FT}}^{(\mathrm{init})} = \boldsymbol{\psi}_{\mathrm{FT}}^{*}$.
19: **end for**

On optimization horizons. For reference, our algorithm to optimize meta-parameters online is reproduced here, in Algorithm 4. Prior work [198] has suggested that online optimization of certain hyperparameters (such as learning rates) using short horizons may yield suboptimal solutions. This is known as the short-horizon bias (SHB) problem. We now discuss this concern further in the context of our algorithm.

- **What is the short-horizon bias (SHB) problem?** SHB is understood to be a special case of the bias induced by truncating telescoping sums for

optimization parameters. The effects of the truncation can be pronounced with optimization parameters [198], but there exist methods like [19] to deal with these if they occur.

- **Do we expect this to be a concern in our setting?** There are two hypergradients in our system that could suffer from bias: the PT hypergradients and the FT hypergradients. In both cases, the impact from biased hypergradients appears to be minimal. We will argue for this claim through each hypergradient term separately.

  PT Hypergradient: The PT hypergradient does not suffer from the short-horizon bias because the PT model is expected to have approximately converged at each hyperparameter update. This is not only a requirement of the implicit function theorem and the algorithm from [113] to apply, but also is directly enforced in our system through the use of online-updates and a warmup period (see Algorithm 4).

  FT Hypergradient: For the gradient through FT, we acknowledge that differentiating through only one step could, in theory, produce biased hypergradients. However, several prior works on meta-learning various structures similar to what we consider [137, 145, 113, 129, 85] did not observe significant bias. Therefore, from an empirical standpoint, this bias is not necessarily expected to be a significant issue.

  As seen in our experimental results, we also observe improved experimental results by setting $K = 1$ in our algorithm, suggesting minimal SHB impact. To study this issue further, we include experiments comparing to full back-propagation through PT and FT in synthetic experiments (Appendix C.4), and compare different values of K in our semi-supervised learning experiments (Appendix C.6).

- **Why might SHB not be a concern with the hyperparameters we consider?** As stated, the SHB issue has mainly been observed in the context of optimization hyperparameters such as the learning rate. This could be because

the learning rate directly affects the rate at which we approach the critical point, but it does not directly change the critical point. As seen in the analysis in [198], optimizing the LR with short rollouts results in (far too aggressively) decaying the step size to decrease variance and converge faster. In contrast, other hyperparameters, like weight decay or augmentations, directly change the fixed point that we are converging to (as opposed to just the rate). In setups where the hyperparameters directly affect the fixed point, SHB has not been observed — for example, see [157, 114]. These works do online, limited horizon optimization of hyperparameters directly affecting the critical point.

C.3 Practical Heuristics for Tuning Optimizer Parameters

The optimization parameters used in nested optimization can be crucial for success. In our synthetic experiments in Section 4.5 we were able to use default optimizer selections; however these settings may not work in practice for all domains (as seen in our real-world experiments). Here, we list some basic guidelines a practitioner can iterate through to debug meta-parameter optimization.

Step 1: What to do if the meta-parameters are changing wildly? First, decrease the learning rate for the meta-parameters. Momentum parameters can be dangerous – see [59]; using an optimizer without momentum may work better in some situations. If the meta-parameters begin oscillating later into training, try decreasing momentum.

Step 2: What to do if the pre-training parameters are changing wildly? First, make sure the meta-parameters are not moving around rapidly. Once the meta-parameters are stable, you should be able to decrease the learning rate of the pre-training optimizer until convergence.

Step 3: What to do if the meta-parameters are not changing? First, make sure that your pre-training parameters are finding good solutions by examining the pre-training optimization and optimizer settings. Next, make sure that the IFT is giving a good approximation for the pre-training response. You should begin with 1 Neumann term (or an identity inverse-Hessian approximation), because this often works well; see [113]. If 1 Neumann term works, you can try adding more until they offer no benefit. Next, make sure that differentiation through optimization is giving a reasonable gradient. If the unrolled optimizer is diverging, this will not give us useful gradients, so we must make sure these FT parameters converge. After you verify these components, try increasing the meta-parameter learning rate.

C.4 Synthetic Experiments: Further Details

We discuss further details on the synthetic experiments. All experiments in this section were run on Google Colab, using the default GPU backend.

C.4.1 Meta-Parameterized Data Augmentation

Here, we have additional details for the data augmentation synthetic experiments in Section 4.5.

Dataset. Both PT and FT tasks are supervised MNIST digit classification (i.e., a 10-class classification problem). Our pre-training dataset is 3000 randomly-sampled MNIST data points. The fine-tuning training and validation sets are a distinct set of 3000 randomly-sampled MNIST data points augmented with a rotational degree drawn from $\mathcal{N}(\mu, \sigma^2)$ for some mean μ and standard deviation σ.

In the main text (Section 4.5) we studied the situation where the FT rotation distribution was $\mathcal{N}(\mu, 1^2)$, $\mu = 90°$; note that the standard deviation is fixed at 1. We also examine here a situation where we try to learn both the mean and standard deviation (results in Figure C-1a), where the FT rotation distribution is $\mathcal{N}(45, 15^2)$.

Defining meta-parameters. Our meta-parameters parameterize a rotational augmentation distribution that we apply to the PT images, $\mathcal{N}(\mu_{\text{PT}}, \sigma^2_{\text{PT}})$. We consider two scenarios. First, where we only optimize the mean: $\phi = \{\mu_{\text{PT}}\}$, and σ_{PT} is fixed to 1, which is the situation in Section 4.5. In this case, the initialization of ϕ is sampled uniformly from $[45, 135]$. Second, we optimize both the mean and the standard deviation of the rotation distribution: $\phi = \{\mu_{\text{PT}}, \sigma_{\text{PT}}\}$ (results in Figure C-1a). In both settings, we expect the optimal PT rotation distribution for augmentations to be equal to what is used at FT time.

Model architectures. Our model is a fully-connected feedforward network, with 1 hidden layer with 64 hidden units and a ReLU activation.

Algorithm and Implementation details. We are able to use implicit differentiation with 1 Neumann term for the pre-training, and 1 step of differentiation through optimization for the fine-tuning training step. We use an Adam optimizer with a LR of 0.01 for pre-training and 0.3 for the meta-parameters. For fine-tuning, we use SGD (to match exactly the methods description in (4.4)) with the default learning rate of 0.01. We train with a batch size of 64 for each optimizer for 5 epochs. We alternate between taking 1 step of optimization for each set of parameters: $N_{\text{warmup}} = 0, K = 1, P = 1$. These hyperparameters were chosen based on a simple strategy discussed in Section C.3, without particular tuning.

Experimental setup. For the main experiments where we learn only the mean, we consider 10 different sampled mean initializations from $[45, 135]$. For the additional experiments where we learn the mean and the standard deviation, we fix the target distribution at $\mathcal{N}(45, 15^2)$ and examine two initializations: $\phi^{(0)} = \{0, 1\}$ and $\phi^{(0)} = \{90, 1\}$.

Results. When learning the mean, we are able to approximately recover the true rotation distribution after training with a final difference mean and standard error of $7.2 \pm 1.5°$, over 10 sampled mean rotations, indicating efficacy of the algorithm.

Next, we examine the results for learning the mean and standard deviation from different initializations, in Figure C-1a, and observe that we can approximately recover the true augmentation distribution from both initializations.

C.4.2 Meta-Parameterized Per-Example Weighting

Here, we have additional details for the example weighting synthetic experiments in Section 4.5.

Dataset. PT and FT tasks are again based on supervised MNIST image classification. The PT task is adjusted to be a 1000-class problem, where MNIST digits in classes 0-4 keep their original labels, and MNIST digits in classes 5-9 are now

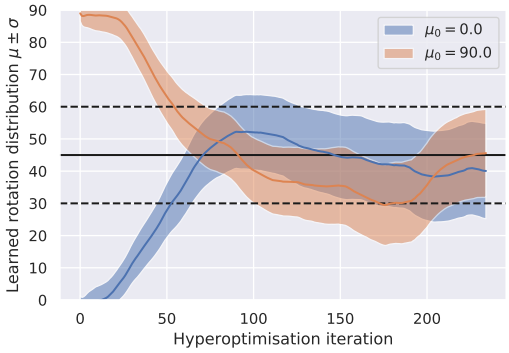
assigned a noisy label: a random label between 5 and 1000. Our PT set is 3000 randomly-sampled MNIST data points. We use the standard MNIST training set for pre-training, and if any data point is in class 5-9 we noise it, by assigning a random label between 5 and 1000. We use the standard MNIST testing set for FT, split into a FT training and validation set. The FT set contains only images with classes 0-4.

Defining meta-parameters. Our meta-parameters ϕ are the parameters of a *weighting CNN* that assigns an importance weight to each PT data point, which is then used to weight the loss on that data point during PT. We expect the optimal weighting strategy to assign maximal weight to PT images in classes 0-4, since these are not noisy and are seen at FT time, and minimal weight to the other images, since these have noisy labels.

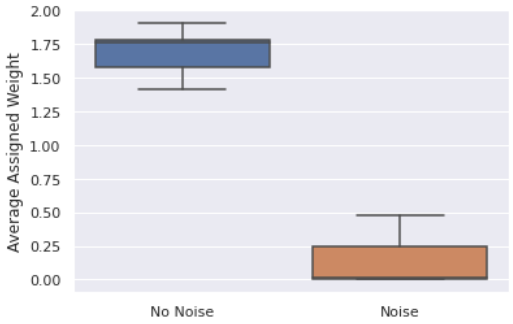
Model architectures. Our weighting network has an architecture of two convolutional layers, then a fully-connected layer. The first layer has 32 filters with a kernel size of 5, followed by batch-norm, with a ReLU activation and max pooling. The second convolutional layer is the same as the first, except with 64 filters and a kernel size of 3. The fully-connected layer has a 1-dimensional output with an activation of 2σ applied, so the output is in $(0, 2)$.

Implementation details. As with the MNIST augmentation experiments, we are able to use implicit differentiation with 1 Neumann terms for the PT, and 1 step of differentiation through optimization for the FT training. Again, we use an Adam optimizer with default parameters for PT and the meta-parameters, and SGD with default learning rate of 0.01 for FT. We use a batch-size of 100 for each optimizer and train each seed for 100 epochs. We alternate between taking 1 step of optimization for each set of parameters: $N_{\text{warmup}} = 0, K = 1, P = 1$. These hyperparameters were chosen based on a simple strategy discussed in Section C.3, without particular tuning.

Results. Using Algorithm 2 once again, we find that PT images from class 0-4 are assigned high weight, and those from classes 5-1000 are assigned low weight. This is



(a) Visualizing the optimization of meta-parameters, which parameterize PT rotation augmentation distributions. We consider two different meta-parameter initializations of mean/standard deviation, $\phi = \{90, 1\}$ (orange) and $\phi = \{0, 1\}$ (blue), showing the mean (solid line) and standard deviation (shaded region) over learning. The rotation distribution for the FT validation set is $\mathcal{N}(45, 15^2)$, shown with a solid black line (mean) and dashed lines (plus/minus standard deviation). In both cases, we observe approximate recovery of the optimal meta-parameters, namely the FT mean and standard deviation. The final mean and standard deviation for the 90 initialization and 0 initialization are 42.4 ± 13.9 and 45.9 ± 15.5 respectively.



(b) The distribution of importance weights assigned to examples with/without noisy labels, over 10 random seeds of weights, produced by a weighting CNN. We show the average weight applied to non-noised and noised examples, normalized by dividing by the sum of the data weights. The weighted CNN has recovered the desired solution of down-weighting examples with noisy labels, indicating successful learning of high-dimensional meta-parameters.

Figure C-1: **Results for learning pre-training augmentation meta-parameters.**

an expected result: since the PT classes 0-4 are also the FT classes, we expect images from these classes to be upweighted. PT images not from these classes do not appear at FT and have noisy labels, hence are downweighted. This result is visualized in Figure C-1b.

C.4.3 The Impact of Approximating Meta-Parameter Gradients

We now study how using the two gradient approximations in our algorithm compare to storing the entire PT and FT process in GPU memory and differentiating through the whole process to obtain meta-parameter gradients. We consider our first synthetic

setting, where we aim to learn rotation augmentations for MNIST PT, given that the FT set is augmented in a specific way. In the following experiments, the FT set is augmented with rotations drawn from $\mathcal{N}(90, 1)$.

Experimental setup. We compare the following methods to study the impact of the gradient approximations.

- Backpropagation through training (BPTT): The PT augmentation distribution is initialized to $\mathcal{N}(45, 1)$. We do 500 steps of PT and 500 steps of FT steps, and use BPTT (through these 1000 optimization steps) to optimize the augmentations. This is near the limit of what we could fit into our GPU memory. This process is then repeated for 500 hyperparameter optimization steps.

- Meta-parameterized PT: We run our algorithm. The PT augmentation distribution is initialized to $\mathcal{N}(45, 1)$. We set $P = 1$, $K = 1$, running for 500 PT and FT steps overall (for a fair comparison with BPTT).

- Optimal augmentations: We set the PT augmentation distribution to be the optimal setting (i.e., identical to that used for FT): $\mathcal{N}(90, 1)$. This is also run for 500 PT and 500 FT steps.

- Initialization augmentations: We set the PT augmentation distribution to be: $\mathcal{N}(45, 1)$ as a baseline. This is also run for 500 PT and 500 FT steps.

Results.

- BPTT without compute limitations: Running BPTT for 500 hyperparameter optimization steps takes about 20 hours. Doing so, it achieves a test accuracy of 88.0%.

- Meta-parameterized PT: Running our method takes about 30 minutes. This achieves a test accuracy of 87.6%.

- BPTT with compute limitations: Limiting the compute budget of BPTT to be similar to our method, it obtains a test accuracy 83.4%.

- Optimal augmentations: This achieves a test accuracy of 88.3%.
- Initialization augmentations: This achieves a test accuracy of 80.1%.

Analysis. As seen, our method, with about 2-3% of the compute time and significantly lower memory cost than BPTT, obtains very comparable performance in this toy domain, and almost matches the performance with the optimal hyperparameter setting. This indicates effective optimization of the hyperparameters.This performance is achieved even when differentiating through a short FT optimization of 1 step.

Conclusions. In this toy domain, our method obtains performance very comparable to BPTT and the optimal hyperparameter setting, and has a fraction of the compute and memory cost of BPTT. This suggests that optimizing the augmentations online is not incurring significant short horizon/truncation bias.

C.5 Meta-Parameterized Multitask PT: Further Details

We provide further dataset details, experimental details, and results for the multitask PT experiments. All experiments in this section were run on a single NVIDIA V100 GPU.

C.5.1 Further dataset details

The transfer learning benchmark we consider is the biological data benchmark from [84] where the prediction problem at both PT and FT is multitask binary classification: predicting the presence/absence of specific protein functions (y) given a Protein-Protein Interaction (PPI) network as input (represented as a graph $\boldsymbol{x}$). The PT and FT datasets both contain 88K graphs.

[84] provide open-source code in their paper to download the raw dataset and then pre-process it. The important steps are extracting subgraphs of the PPI networks centered at particular proteins, and then using the Gene Ontology to identify the set of protein functions associated with each of the proteins.

Importantly, the set of protein functions that we predict at PT time and FT time are different. The PT targets represent coarse-grained biological functions, and the FT targets are fine-grained biological functions, which are harder to obtain experimentally and therefore there is interest in predicting them having pre-trained a model on predicting the targets that are more readily obtained. The PT dataset has labels $y \in \{0,1\}^{5000}$, and the FT dataset has labels $y \in \{0,1\}^{40}$.

[84] discuss the importance of appropriate train/validation/test set splitting for this domain. We follow their suggestion and use the *species split*, where the test set involves predicting biological functions for proteins from new species, not encountered at training/validation time.

We refer the reader to [84] for full details on the pre-processing and construction of subgraphs, the nature of the labels, and the splitting strategy for training, validation,

and testing.

C.5.2 Further experimental details

Baselines

We include most important details for baselines in Section 4.6. Here, for the CoTrain + PCGrad baseline we provide further details, and we also include information about another baseline, CoTrain + Learned Task Weights.

CoTrain + PCGrad details: In our implementation, we computed gradient updates using a batch of data from $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$ separately, averaging the losses across the set of binary tasks in each dataset (5000 for $\mathcal{D}_{\text{PT}}$ and 40 for $\mathcal{D}_{\text{FT}}$). PCGrad [205] was then used to compute the final gradient update given these two averaged losses. We also experimented with: (1) computing the overall update using all 5040 tasks (rather than averaging), but this was too memory expensive; and (2) computing the overall update using an average over the 5000 PT tasks and each of the 40 FT tasks individually, but this was unstable and did not converge.

A further baseline: CoTrain + Learned Task Weights: We also tried a variant of CoTrain where we learn task weights for each of the 5040 tasks (from $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$), along with training the base model. We treat the task weights as high-dimensional supervised learning hyperparameters and optimize these task weights using traditional gradient-based hyperparameter optimization, following the work from [113]. These weights are optimized based on the model's loss on the validation set split of $\mathcal{D}_{\text{FT}}$.

Implementation details

General details for all methods. For all methods, we use the Graph Isomorphism Network (GIN) architecture [200], which was found to be effective on this domain [84].

All methods first undergo PT for 100 epochs with Adam, with a batch size of 32. We used LR=1e-3 for Graph Supervised PT, CoTrain and CoTrain + PCGrad, which is the default LR in the prior work [84]. For the two nested optimization methods

that jointly pre-train and learn weights, CoTrain + Learned Task Weights and Meta-Parameterized PT, we used LR=1e-4; we originally tried LR=1e-3, but this led to unstable nested optimization.

After PT, all methods are then fine-tuned for 50 epochs using Adam, with a batch size of 32, over 5 random seeds, using early stopping based on validation set AUC (following [84]). We used 5 seeds rather than 10 ([84] used 10) for computational reasons. For all models, we initialize a new FT network head on top of the PT network body. At FT time, we either FT the whole network (Full transfer) or freeze the PT encoder and learn the FT head alone (Linear Evaluation [134]). We report results here for both FT policies for all methods.

When fine-tuning models using the Full Transfer paradigm, we found that methods were sensitive to LR choices and a FT LR of 1e-3 used in [84] was unstable. The Adam optimizer FT LRs of 1e-5, 3e-5, and 1e-4 were tried for different methods, with FT validation set AUC used to choose the best LR. For Meta-Parameterized PT, we used a full transfer FT LR of 1e-5, and for the other methods, we used 3e-5. For linear evaluation, we used Adam with an LR of 1e-4 for all methods, which was stable.

Further details for Meta-Parameterized PT. For meta-parameterized PT, during the meta-PT phase, we use the Adam optimizer with a learning rate of 1e-4 for both PT and FT parameters, and use Adam with a LR of 1 for meta-parameters. These values were set based on the methodology in Appendix C.3. In Algorithm 4, we use a Neumann series with 1 step in evaluating the inverse Hessian for PT, 1 warmup epoch, $P = 10$ PT steps, $K = 1$ FT steps; we did not search over values for these, and these choices were partly influenced by compute considerations (e.g., large K is more memory expensive).

With these settings, meta-parameterized PT on this task takes about 8-9 GB of GPU memory (about twice the memory cost of normal PT, which is 4-5 GB), and takes about 5 hours to run (as compared to about 2.5 hours for standard PT).

Further details for CoTrain + Learned Task Weights. Following a similar process to the above, we used Adam with LR of 1e-4 for the base parameters and LR of 1 for the task weights. We use a Neumann series with 1 step when using the method from [113] for the fairest comparison with meta-parameterized PT.

Experimental Setup

We re-state the two settings considered, and provide more details about an additional scenario in the Partial FT Access setting.

(1) *Full FT Access:* Provide methods full access to $\mathcal{D}_{\text{PT}}$ and $\mathcal{D}_{\text{FT}}$ at PT time ($\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$) and evaluate on the full set of 40 FT tasks.

(2) *Partial FT Access:* Consider two situations. First, construct a scenario where we limit the FT data available at PT time directly: $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right| = 0.5\,|\mathcal{D}_{\text{FT}}|$. We assess performance on the full FT dataset, as before. Results for this were not presented in the main text due to space constraints.

Second, limit the number of FT tasks seen at PT time, by letting $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$ include only 30 of the 40 FT tasks. At FT time, models are fine-tuned on the held-out 10 tasks not in $\mathcal{D}_{\text{FT}}^{(\text{Meta})}$. We use a 4-fold approach where we leave out 10 of the 40 FT tasks in turn, and examine performance across these 10 held-out tasks, over the folds.

C.5.3 Further results

Quantitative Results

Summary of main quantitative results. Table C.2 summarizes the main results across full and limited data/task regimes, reporting the better of Full Transfer and Linear Evaluation. We observe consistent improvements with the meta-parameterized PT strategy over the baselines on the three different experimental evaluation settings.

In the remainder of this section, we discuss these quantitative results further, showing both full transfer and linear evaluation results, and other analysis.

Method	AUC ($\left\|\mathcal{D}_{\mathrm{FT}}^{(\mathrm{Meta})}\right\| = \|\mathcal{D}_{\mathrm{FT}}\|$)	AUC ($\left\|\mathcal{D}_{\mathrm{FT}}^{(\mathrm{Meta})}\right\| = 0.5\,\|\mathcal{D}_{\mathrm{FT}}\|$)	AUC ($\mathcal{D}_{\mathrm{FT}}^{(\mathrm{Meta})}$ excludes tasks)
No PT	66.6 ± 0.7	66.6 ± 0.7	65.8 ± 2.5
Graph Sup PT	74.7 ± 0.1	74.7 ± 0.1	74.8 ± 1.8
CoTrain	70.2 ± 0.3	71.0 ± 0.2	69.3 ± 1.8
CoTrain + PCGrad	69.4 ± 0.2	71.1 ± 0.2	68.1 ± 2.3
Meta-Parameterized PT	**78.6 ± 0.1**	**78.2 ± 0.1**	**77.0 ± 1.3**

Table C.2: **Meta-Parameterized PT improves predictive performance in three evaluation settings.** Table showing mean AUC and standard error on mean for three different evaluation settings. **First results column: Full FT Access, with evaluation on all tasks, with all FT data provided at PT time. Second results column: Partial FT Access, evaluation with limited FT data at PT time.** When only 50% of the FT dataset is provided at PT time, Meta-Parameterized PT can again improve on other methods in mean test AUC over 40 FT tasks, demonstrating sample efficiency. **Third results column: Partial FT Access, evaluation on new, unseen tasks at FT time.** When 10 of the 40 available FT tasks are held-out at PT, over four folds (each set of 10 FT tasks held out in turn), considering mean test AUC across tasks and folds (and standard error on the mean), meta-parameterized PT obtains the best performance: it is effective even with partial information about the downstream FT tasks.

Further results for Full FT Access setting. Table C.3 presents results for all methods across 40 FT tasks, considering both full transfer and linear evaluation. We observe that meta-parameterized PT improves on other baselines in both settings, but most noticeably so in linear evaluation. We also present the results for No PT and Graph Supervised PT from [84]. We observe improvements with our re-implementation, which uses lower FT LRs.

Studying potential overfitting in CoTrain strategies. For methods leveraging the FT dataset during PT, the process of performing FT might worsen performance if the model overfits the FT training set. We evaluate FT test performance 'online' during the PT phase, with results in Table C.4, and observe that meta-parameterized PT outperforms other methods here also. We do observe some of this overfitting behaviour: note the improved performance on the test set with the learned weights strategy.

Further results for Partial FT Access setting. Table C.5 shows improved performance even with smaller meta-FT datasets, and Table C.6 shows improved per-

Method	Full Transfer	Linear Evaluation
Rand Init (from [84])	64.8 ± 0.3	N/A
Rand Init (reimplement, lower FT LR)	66.6 ± 0.7	N/A
Graph Sup PT (from [84])	69.0 ± 0.8	N/A
Graph Sup PT (reimplement, lower FT LR)	73.9 ± 0.2	74.7 ± 0.1
CoTrain	70.2 ± 0.3	65.9 ± 0.1
CoTrain + PCGrad	69.4 ± 0.2	62.4 ± 0.3
CoTrain + Learned Task Weights	67.7 ± 0.2	64.4 ± 0.1
Meta-Parameterized PT	74.7 ± 0.3	78.6 ± 0.1

Table C.3: **Meta-Parameterized PT results in improved predictive performance.** Table showing mean AUC and standard error on mean across 40 FT tasks on the held-out test set, over 5 random FT seeds. We observe that Meta-Parameterized PT outperforms other baselines in both Full Transfer and Linear Evaluation settings, with significant improvement with Linear Evaluation. Note that with a lower FT LR, baselines from [84] are improved relative to previously reported performance.

Method	Test AUC	Validation AUC
Meta-Parameterized PT	76.1	88.2
CoTrain	67.3	83.1
CoTrain + PCGrad	69.0	84.0
CoTrain + Learned Task Weights	70.7	84.6

Table C.4: Mean AUC across FT tasks evaluated during PT, for methods that use the FT set at PT time. The separate FT stage may worsen performance of some of the methods, and evaluating in this manner helps account for that. In this setting also, meta-parameterized PT improves on other baselines, in both test and validation set performance.

formance even with limited tasks at meta-FT time.

Qualitative Results

We now analyze other aspects of meta-parameterized PT.

Analyzing learned representations. To understand the impact of meta-parameterized PT on what the model learns, we compare the learned representations on the FT data across the different PT strategies using Centered Kernel Alignment (CKA) [101, 41] in Figure C-2. We observe that Meta-Parameterized PT most closely resembles a combination of CoTrain + Learned Weights and Supervised PT, which is sensible given that it blends aspects of both approaches.

Method	Full Transfer	Linear Evaluation
Rand Init (from [84])	64.8 ± 0.3	N/A
Rand Init (reimplement, lower FT LR)	66.6 ± 0.7	N/A
Graph Sup PT (from [84])	69.0 ± 0.8	N/A
Graph Sup PT (reimplement, lower FT LR)	73.9 ± 0.2	74.7 ± 0.1
CoTrain	71.0 ± 0.2	64.4 ± 0.1
CoTrain + PCGrad	71.1 ± 0.2	64.4 ± 0.1
CoTrain + Learned Task Weights	66.0 ± 0.3	64.6 ± 0.3
Meta-Parameterized PT	74.3 ± 0.2	78.2 ± 0.1

Table C.5: **Meta-Parameterized PT also improves predictive performance with smaller MetaFT datasets.** In a setting where only 50% of the FT dataset is provided at PT time, Meta-Parameterized PT can again improve on other methods in mean test AUC over 40 FT tasks, indicating that it is effective even with limited amounts of FT data available at PT time.

Analyzing learned weights. Figure C-3 compares learned weights for meta-parameterized PT and the CoTrain+Learned Weights strategies. We observe differences in the histogram of weights, and also the specific values on a per-task basis for these two strategies, indicating that they learn different structures.

Analyzing negative transfer. Figure C-4 assesses potential negative transfer on a per-task basis, comparing performance with PT to performance after supervised PT and meta-parameterized PT. Both PT strategies have little negative transfer, and meta-parameterized PT obtains a small extra reduction in negative transfer over standard supervised PT.

Method	Full Transfer	Linear Evaluation
Rand Init	65.8 ± 2.5	N/A
Graph Sup PT	71.5 ± 1.6	74.8 ± 1.8
CoTrain	69.3 ± 1.8	67.0 ± 2.0
CoTrain + PCGrad	67.1 ± 1.5	68.1 ± 2.3
CoTrain + Learned Weights	65.4 ± 2.0	69.1 ± 2.6
Meta-Parameterized PT	71.3 ± 2.5	77.0 ± 1.3

Table C.6: **When evaluating on new, unseen tasks at FT time, meta-parameterized PT again improves on other methods.** We consider a setting where 10 of the 40 available FT tasks are held-out at PT, and only provided at FT time. Over four folds (where different sets of 10 FT tasks are held out in turn), considering mean test AUC across tasks and folds (and standard error on the mean over folds), meta-parameterized PT obtains the best performance. This suggests that the method can perform well even with partial information about the downstream FT tasks.

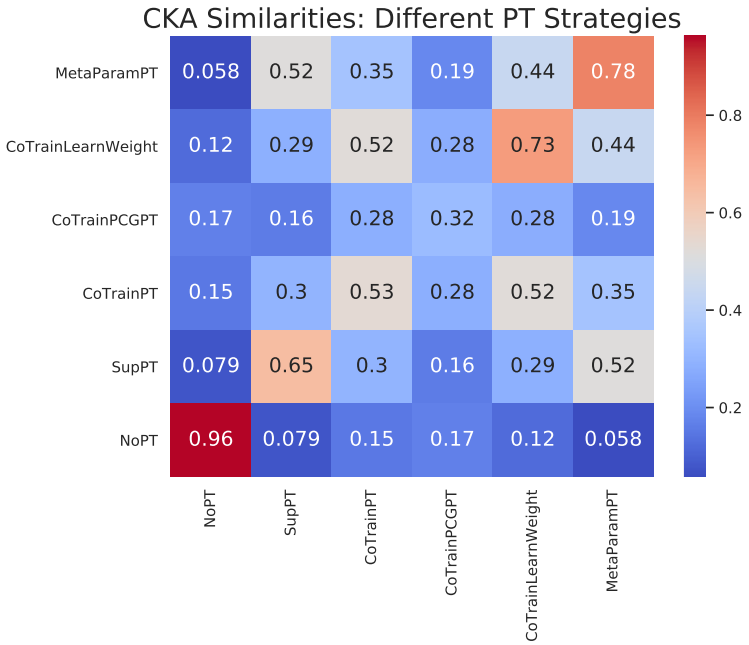


Figure C-2: Comparing learned representations with different PT strategies using CKA [101]. We obtain model representations before the final linear layer across 6400 FT data points, and then compute CKA between pairs of models (averaging over different random initialisations). We observe that Meta-Parameterized PT most closely resembles a combination of CoTrain + Learned Weights and Supervised PT, which is sensible given that it blends aspects of both approaches: meta-parameterized PT learns task weights to modulate the learned representations (as in CoTrain + Learned Weights), and representations are adapted using the PT task alone (as in supervised PT). Interestingly, CoTrain + PCGrad has comparatively little similarity to most other methods in terms of its learned representations.

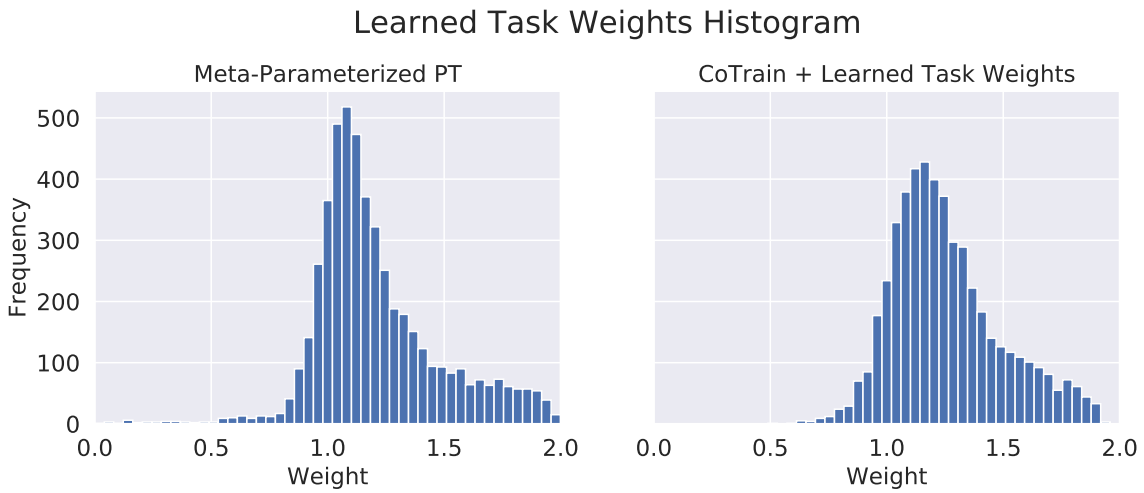


Figure C-3: Comparing learned weights on 5000 PT tasks for meta parameterized PT and with CoTrain + Learned Weights. We observe that different structures appear to be learned by these approaches; the median/half IQR in absolute difference in learned weights is 0.13 ± 0.09. Meta-Parameterized PT appears to have more tasks downweighted (weights below 0.5) than the CoTrain approach.

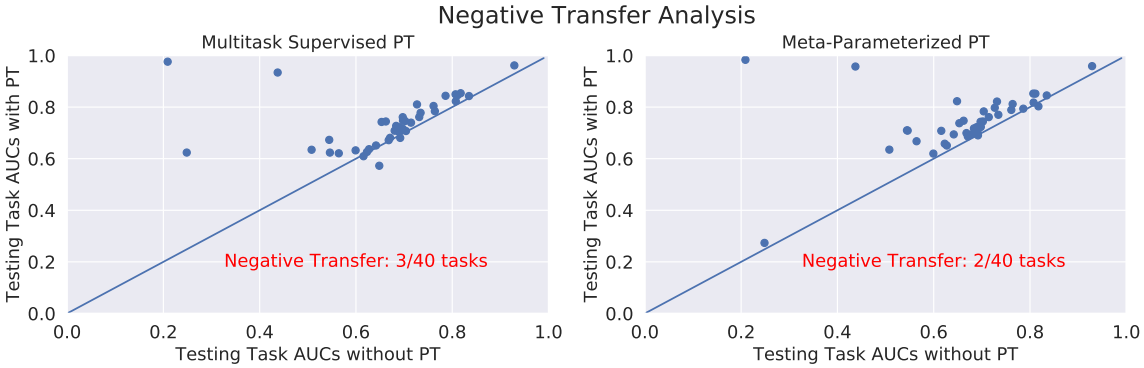


Figure C-4: Comparing performance on FT tasks with and without two different PT strategies: standard supervised PT on the left, and meta-parameterized PT on the right. We show the mean performance over 5 seeds on each of the 40 FT tasks without PT (x axis) and with PT (y axis). A small improvement is observed in reduced negative transfer with Meta-Parameterized PT.

C.6 Meta-Parameterized SimCLR PT: Further Details

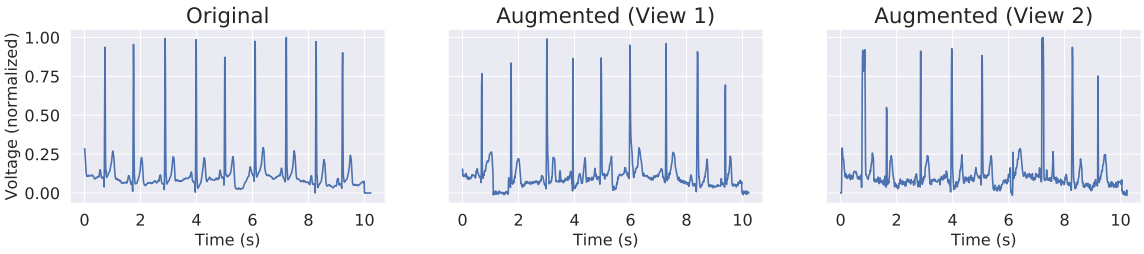


Figure C-5: A single lead (or channel) of the 12 lead ECG signal and two augmented views (following cropping, jittering, and temporal warping) that are used in contrastive learning.

We provide further SimCLR details, dataset details, experimental details, and results for the SimCLR ECG experiments. All experiments in this section were run on a single NVIDIA V100 GPU.

C.6.1 SimCLR summary

SimCLR is a variant of contrastive self-supervised learning [194, 199, 134, 81]. During training, examples are augmented in two different ways to create two views $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, each of which are encoded independently to produce representations $f^{(\text{enc})}(\boldsymbol{x}_i) = \boldsymbol{h}_i$ and $f^{(\text{enc})}(\boldsymbol{x}_j) = \boldsymbol{h}_j$. These representations are further transformed using a multi-layer decoder ("projection head") to produce vectors $f^{(\text{dec})}(\boldsymbol{h}_i) = \boldsymbol{z}_i$ and $f^{(\text{dec})}(\boldsymbol{h}_j) = \boldsymbol{z}_j$. Models are trained to minimize the normalized temperature-scaled cross-entropy loss (NT-Xent), which contrasts the similarity between pairs of views derived from the same example against the other $2N-2$ views in a minibatch of size N:

$$\mathcal{L}_{\text{PT}}(\boldsymbol{z}_i, \boldsymbol{z}_j) = -\log \frac{\exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(\boldsymbol{z}_i, \boldsymbol{z}_k)/\tau)} \tag{C.1}$$

where $\text{sim}(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{a}^\top \boldsymbol{b}/(\|\boldsymbol{a}\|\|\boldsymbol{b}\|)$ is cosine similarity and τ is the temperature hyperparameter.

C.6.2 Further dataset details

We construct our semi-supervised learning (SSL) problem using PTB-XL [191, 61], an open-source dataset of electrocardiogram (ECG) data. Let the model input at both PT and FT time be denoted by $\boldsymbol{x}$, which represents a 12-lead (or channel) ECG sampled at 100 Hz for 10 seconds resulting in a 1000×12 signal. An example signal is in Figure C-5. The PTB-XL dataset contains 21837 ECGs from 18885 unique patients. Each ECG has a 5-dimensional label $y \in \{0, 1\}^5$, where each dimension indicates whether the signal contains certain features indicative of particular diseases/pathologies, namely: Normal ECG, Myocardial Infarction, ST/T Change, Conduction Disturbance, and Hypertrophy. The dataset is split in 10 folds on a patient-level (ECGs from the same patient are all in the same fold), with a suggested train-validation-testing split.

To form an SSL problem from this dataset, we take the training and validation folds, remove the labels, and use only the unlabelled ECGs as the PT dataset. This PT dataset has 19634 unique ECGs. For the FT dataset, we take a random sample of $|\mathcal{D}_{\text{FT}}|$ ECG-label pairs from the training and validation folds. As is common in prior SSL work, we consider different sizes of $\mathcal{D}_{\text{FT}}$ to understand performance given different amounts of labelled data. The FT testing set is the testing fold of the original dataset, which has 2203 ECG-label pairs.

At both PT and FT time, ECGs are normalized before input to the model using zero mean-unit variance normalization, following [191].

We refer the reader to the open-source data repository on PhysioNet [61], and the paper introducing the dataset [191] for further details.

C.6.3 Further experimental details

ECG Data Augmentations

To augment each ECG for SimCLR, we apply three transformations in turn (based on prior work in time series augmentation [87, 197]):

1. **Random cropping:** A randomly selected portion of the signal is zeroed out. We

randomly mask up to 50% of the input signal.

2. **Random jittering:** IID Gaussian noise is added to the signal. The noise is zero mean and has standard deviation equal to 10% of the standard deviation of the original signal.
3. **Random temporal warping:** The signal is warped with a random, diffeomorphic temporal transformation. To form this, we sample from a Gaussian with zero mean, and a fixed variance at each temporal location, to generate a 1000 dimensional random velocity field. This velocity field is then integrated (following the scaling and squaring numerical integration routine used by [9, 10]). This resulting displacement field is then smoothed with a Gaussian filter to generate the smoothed temporal displacement field, which is 1000 dimensional. This field represents the number of samples each point in the original signal is translated in time. The field is then used to transform the signal, translating each channel in the same way (i.e., the field is the same across channels).

Two augmented views of an ECG are shown in Figure C-5.

Implementation details

General details for all methods. For all methods, we use a 1D CNN based on a ResNet-18 [77] architecture as the base model that undergoes PT & FT. This model has convolutions with a kernel size of 15, and stride 2 (set based on the rough temporal window we wish to capture in the signal). The convolutional blocks have 32, 64, 128, and 256 channels respectively. The output of these layers is average pooled in the temporal dimension, resulting in a 256 dimensional feature vector. For SimCLR PT, the projection head takes this 256 dimensional vector as input and is a fully connected network with 1 hidden layer of size 256, and output size of 128, with ReLU activation. These hyperparameters were not tuned.

The SimCLR methods are first pre-trained on the PT dataset using SimCLR PT, with a temperature of 0.5 in the NT-Xent loss. We use Adam with an LR of 1e-4 for SimCLR PT, with a batch size of 256, and pre-train for 50 epochs. We consider 3 PT seeds. The methods are then fine-tuned on the FT dataset, replacing the projection

head with a new linear FT network head. This whole network is fine-tuned for 200 epochs with Adam, learning rate of 1e-3, batch size of 256. We used an 80%-20% split of the labelled data to form training and validation sets, and validation set AUC was used for early stopping. We consider 5 FT seeds, resulting in a total of 15 runs for each method at each setting.

Further details for Meta-Parameterized PT. Meta-parameterized SimCLR incorporates a learned per-example temporal warping strength. We form this by instantiating a four-layer 1D CNN $w(\boldsymbol{x}; \phi)$ that takes in the input ECG $\boldsymbol{x}$ and outputs the variance (1-D output) of the velocity field used to generate the random velocity field. This network has four blocks of convolution, batch norm, and ReLU activation with a kernel size of 15, stride of 2, and 32 channels. We also a optimize a global warping strength scale that multiplies the network output to adjust the overall scale of the warping. The network weights and the global scale are optimized using Adam, with LR=1e-4 and LR=1 respectively. These values were set based on the methodology in Section C.3. In Algorithm 4, we use a Neumann series with 1 step when evaluating the inverse Hessian for PT, 1 warmup epoch, $P = 10$ PT steps, $K = 1$ FT steps; we did not search over these, and chose these values based on compute considerations. However, we do conduct a comparison with running for other values of K in Appendix C.6.4.

With these settings, meta-parameterized PT on this task takes about 8-9 GB of GPU memory (about twice the memory cost of normal PT, which is 4-5 GB), and takes about 3 hours to run (as compared to about 1.5 hours for standard PT).

When running meta-parameterized PT with very small meta-FT datasets, of size 10 or 25, the 80%-20% split is not as practical. When $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right| = 10$, we use a 50-50 split in training and validation, and when it is 25, we use a 60-40 split.

An additional baseline: SimCLR + Optimized Supervised Learning Augmentations (SimCLR + OptSLA): We also investigated a baseline in this domain where the same parametric augmentation policy used for meta-parameterized

PT above is: (1) optimized for supervised learning on the labelled FT set, $\mathcal{D}_{\text{FT}}$, following the method from [113]; (2) used as is in SimCLR PT to learn representations; (3) evaluated in a standard FT setting. When using the algorithm from [113], the augmentation meta-parameters are optimized based on the model's loss on the validation set split of $\mathcal{D}_{\text{FT}}$, as is typical in hyperparameter optimization. This baseline compares how optimizing augmentations over the two stage PT and FT compares to optimizing for supervised learning alone. We use Adam for optimization, and use 1 Neumann step in the algorithm from [113].

Experimental Setup

We re-state the two experimental settings considered. In both settings, we evaluate performance as average AUC across the 5 binary classification tasks, reporting mean and standard error over the 15 runs.

(1) *Full FT Access*, standard SSL: consider different sizes of the labelled FT dataset $\mathcal{D}_{\text{FT}}$ and make all the FT data available at meta-PT time, $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$.

(2) *Partial FT Access*, examining data efficiency of our algorithm: SSL when only limited FT data is available at meta-PT time: $\mathcal{D}_{\text{FT}}^{(\text{Meta})} \subseteq \mathcal{D}_{\text{FT}}$.

C.6.4 Further results

We now present additional results in the semi-supervised learning domain.

Further Full PT Access results with new baseline. We first present results in the Full PT Access setting, varying $|\mathcal{D}_{\text{FT}}|$ and setting $\mathcal{D}_{\text{FT}} = \mathcal{D}_{\text{FT}}^{(\text{Meta})}$, shown in Table C.7. As seen, meta-parameterized SimCLR obtains improvements over the one-stage hyperparameter learning baseline, SimCLR + OptSLA, suggesting that learning augmentations for the two-stage PT & FT process is advantageous.

Impact of K. We now study the impact of different values of K, the number of unrolled differentation steps when computing the gradient through FT. In our main

	Test AUC at different FT dataset sizes $\lvert\mathcal{D}_{\text{FT}}\rvert$			
	100	250	500	1000
No PT	71.5 ± 0.7	76.1 ± 0.3	78.7 ± 0.3	82.0 ± 0.2
SimCLR	74.6 ± 0.4	76.5 ± 0.3	79.8 ± 0.3	82.2 ± 0.3
SimCLR + OptSLA	74.6 ± 0.6	77.0 ± 0.3	79.6 ± 0.4	82.8 ± 0.2
Meta-Parameterized SimCLR	**76.1 ± 0.5**	**77.8 ± 0.4**	**81.7 ± 0.2**	**84.0 ± 0.3**

Table C.7: **Meta-Parameterized SimCLR obtains improved semi-supervised learning performance.** Table showing mean AUC/standard error over seeds across 5 FT binary classification tasks for baselines and meta-parameterized SimCLR at different sizes of $\mathcal{D}_{\text{FT}}$, with $\mathcal{D}_{\text{FT}}^{(\text{Meta})} = \mathcal{D}_{\text{FT}}$. We observe improvements in performance with meta-parameterized SimCLR over other baselines, including SimCLR + OptSLA, which optimizes the augmentations purely for one-stage supervised learning (rather than two-stage PT and FT).

experiments, we set $K = 1$ for simplicity and computational efficiency. We now seek to understand the following alternative choices:

- $K = 0$: In this setting, we perform **no** FT when optimizing the meta-parameters; that is, we use a randomly initialized linear classifier on top of the PT representations when we compute the FT loss. The meta-learning problem here corresponds to learning PT meta-parameters that optimize the performance of a randomly initialized linear classifier. This experiment tests what happens when the gradient through FT is noisy, but the component through PT is informative.

- $K > 1$: This setting tests whether unrolling more steps during FT can improve the gradient signal received when optimizing meta-parameters.

Results are shown in Table C.8. As can be seen, unrolling through one step of FT ($K = 1$) improves upon using a noisy FT gradient ($K = 0$) in all cases. Using a noisy FT gradient but an informative PT component ($K = 0$) improves on not optimizing the augmentations at all (SimCLR). This implies that both the PT and FT components inform the optimization of the augmentations. When $K > 1$, we do observe some improvement with more steps, but diminishing returns as K increases further.

	Test AUC at different FT dataset sizes $\lvert\mathcal{D}_{\text{FT}}\rvert$			
	100	250	500	1000
SimCLR	74.6 ± 0.4	76.5 ± 0.3	79.8 ± 0.3	82.2 ± 0.3
$K = 0$	75.3 ± 0.5	77.1 ± 0.5	80.5 ± 0.4	83.7 ± 0.3
$K = 1$	76.1 ± 0.5	77.8 ± 0.4	81.7 ± 0.2	84.0 ± 0.3
$K = 5$	76.6 ± 0.2	78.3 ± 0.3	81.9 ± 0.4	84.2 ± 0.2
$K = 10$	76.3 ± 0.5	78.1 ± 0.4	81.7 ± 0.4	84.3 ± 0.3

Table C.8: **Examining how the number of unrolled FT steps affects semi-supervised learning performance.** Table showing mean AUC/standard error over seeds across 5 FT binary classification tasks for meta-parameterized SimCLR when we vary the number of unrolled FT steps used to compute the meta-parameter gradient. We observe that using a noisy FT gradient ($K = 0$) improves on not optimizing augmentations at all, but is worse than using a single step ($K = 1$). Using more unrolled steps can lead to small improvements.

Further Partial PT Access results. We now consider the Partial FT Access setting. Firstly, Figure C-6 shows the performance of meta-PT when we fix $|\mathcal{D}_{\text{FT}}| = 500$ and vary $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$. We find that meta-PT can be effective even with very small validation sets (consider the sharp improvement at small MetaFT data points, with 0 MetaFT points representing no optimization of the augmentations). This result was just considering the $|\mathcal{D}_{\text{FT}}| = 500$ setting; in Figure C-7, we consider other FT dataset sizes and analyze performance. We see that in all regimes, there is a noticeable increase in performance at small meta-FT dataset sizes, which is a desirable result since it shows that our algorithm can be effective even with very limited labelled data available at meta-PT time.

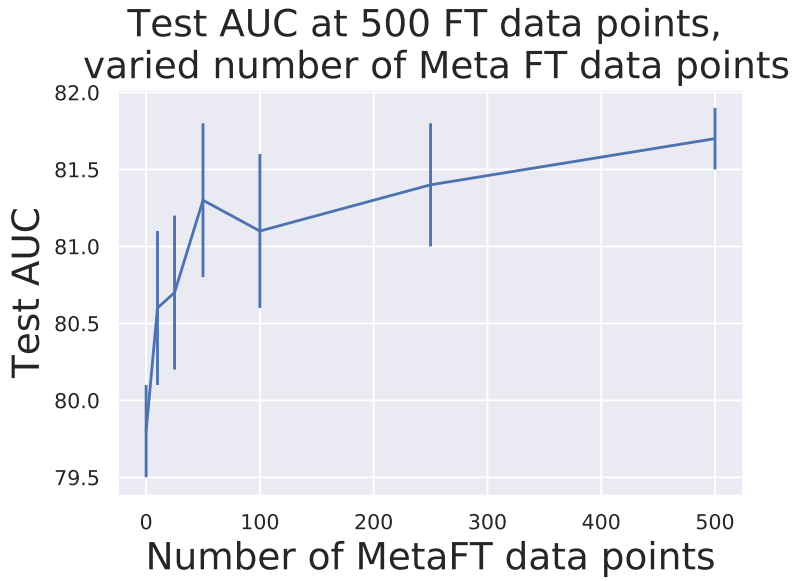


Figure C-6: **Meta-Parameterized SimCLR is effective when only small amounts of FT data are available at PT time.** Test set AUC when varying $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$: the number of FT data points provided at PT time, considering $|\mathcal{D}_{\text{FT}}| = 500$. We see that meta-parameter learning is effective even at small $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$ (sharp improvement in performance at small $\left|\mathcal{D}_{\text{FT}}^{(\text{Meta})}\right|$)

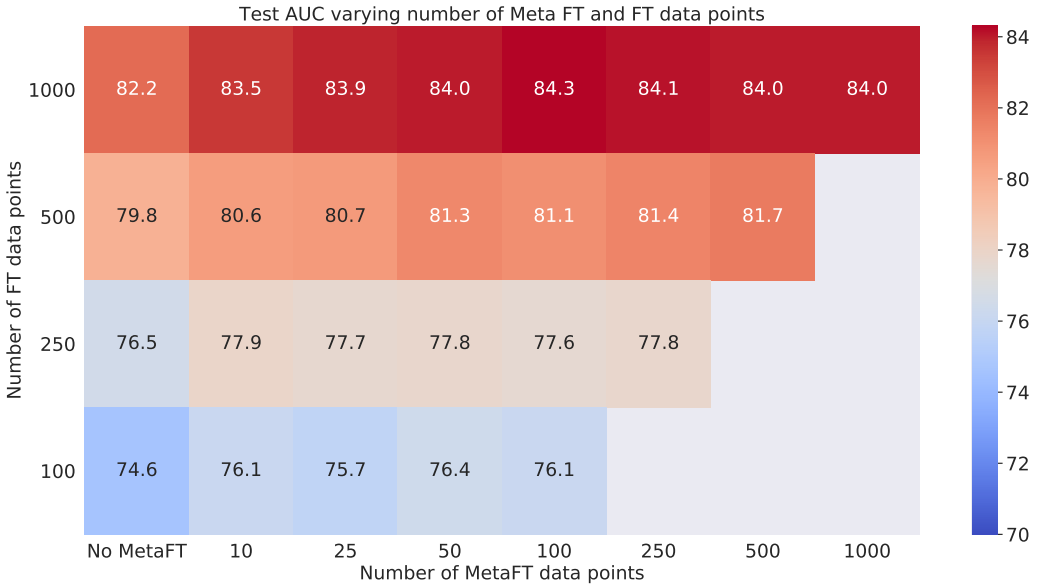


Figure C-7: Sweeping over meta FT/FT data points and analyzing performance trends. We observe that across various settings of FT data availability, a small amount of MetaFT data can lead to significant performance improvements.

Appendix D

Additional Information and Results for Chapter 5

D.1 Augmentation Functions for Sequential Multi-Dimensional Self-Supervised Learning

In this section, we specify more details about augmentation functions used in our method.

D.1.1 Augmentation Details

We form an augmented trajectory by separately augmenting each of the data modalities within the trajectory, using the following approach for each data type.

High-frequency signal s: For each signal in the trajectory of length T, we form a pair of augmented views by first splitting the signal into two disjoint segments and then applying random masking and noise addition as augmentations to each view independently, similar to the approach used in CLOCS [98]. The intuition is that two segments of a signal that are close in time should encode similar physiology, and can therefore be considered paired views. Random masking and noise addition are commonly used as time-series augmentations [63, 208, 148, 88]. In more detail:

- Signal splitting: We split a raw signal of 30 seconds into two disjoint 10 second segments for the first phase of the augmentation process.

- Random signal masking: we choose a random 25% of the signal to set to zero – this was found to overall be more effective than masking proportions of 10% and 50%.

- Noise addition: we add Gaussian noise with standard deviation 0.25 to the signal.

Structured-time series data w: The tabular data sequence forms a $T \times M$ matrix over all timesteps of the trajectory. Following prior work [204], we apply two data augmentation strategies to this matrix: history cutout and noise addition. In more detail:

- History cutout: For each feature, with probability 0.25, randomly set 25% of the timesteps in that timeseries to be missing. Forward fill impute this value. This mirrors the imputation strategy used in our raw data. Unlike in [204], we use forward filling rather than replacing with zeros, because our time series are much shorter (8 timesteps rather than 48) and therefore replacing with zeros destroyed too much information. Using more aggressive cutout augmentations was found to worsen performance, likely because they destroyed too much information in the data. This is in general a challenge when using relatively short time series.
- Noise addition: For each feature, add Gaussian noise with standard deviation equal to 10% of the standard deviation of that feature's values in the training set.

Static features d: Following [204], we use random dropout and noise addition. Other corruption strategies (e.g., [7]), were found to be less effective, potentially due being too strong (also seen in [106]). We randomly drop out 25% of the features (impute with the mean value) and add add Gaussian noise with standard deviation equal to 10% of the standard deviation of that feature's values in the training set.

D.1.2 Other Augmentations

Early on in our experiments, we investigated other augmentation functions such as channel dropout for the structured time-series data [204] and more complex signal augmentations, such as random lead masking [133]. However, we found the improvements from these to be inconsistent and the hyperparameters to be difficult to tune, so we opted for this more focused set of augmentations.

D.2 Further Experimental Details

In this section, we provide further details about our experiments. We first describe more about the datasets and data preprocessing. We then provide further information on the model architecture for our method and baselines, and discuss our hyperparameter search and settings. We then provide additional quantitative results and representational similarity analysis.

D.2.1 Dataset Details

As discussed in the main text, we consider two clinical datasets in our experiments:

- **Dataset 1** is a private dataset derived from the electronic health record (EHR) of the Massachusetts General Hospital (MGH), consisting of a cohort of patients with a prior diagnosis of heart failure. For each patient, we have structured data from the EHR and physiological signals measured by a bedside telemetry monitor. These signals include vitals signs such as heart rate (HR) and oxygen saturation (SpO2), measured at a low frequency (0.5 Hz), and waveforms such as the electrocardiogram (ECG), measured at a high frequency (240 Hz).

  This dataset was obtained with IRB approval (protocol number 2020P003053). Since the dataset has some identifiable information, all computations are performed on a server that sits behind the hospital firewall. Due to restrictions surrounding its use, this dataset cannot be released at this stage.

Table D.1: Dataset statistics.

	Dataset 1		Dataset 2	
Task	# Patients	# Trajectories	# Patients	# Trajectories
Pre-Training	8888	43858	5022	26615
Elevated mPAP	2025	48511	500	14957
24hr mortality	9605	57758	5689	318306

- **Dataset 2** is a public dataset derived from the commonly used MIMIC-III clinical database [93, 61] and its associated database of physiological signals [123]. The clinical database contains structured data over a patient's stay, and the physiological signals database contains vitals signs (HR, SpO2) and waveforms (ECG) measured by a bedside telemetry monitor.

  We use the widely adopted preprocessing pipeline introduced in [72] to form the specific cohort and extract the structured data features used in modeling. This pipeline also provides the functionality to create development and testing sets for the different downstream tasks we consider.

  The clinical database is available on PhysioNet [61] to credentialed users. The database of physiological signals is open-access on PhysioNet.

Constructing PT and FT sets. As outlined in Section 5.4.1, both datasets consist of a number of hospital visits, which we resample at hourly resolution. We extract 30 seconds of the high-dimensional physiological signals at each hour marker from the raw data store.

To generate PT trajectories from these resampled visits, we first split each visit into non-overlapping contiguous 12 hour blocks. A PT trajectory is formed by first sampling a 12 hour block from all the extracted blocks, and then selecting 8 contiguous timesteps from the sampled block (with the starting timestep selected randomly). This trajectory construction strategy has implications in terms of the negative samples in both losses:

- **Global loss.** Consider a sampled anchor trajectory from a given patient i, timesteps $1 - 8$. When computing the global loss, the negative pairs for that

anchor trajectory are either: (1) a trajectory from a different patient $j \neq i$; or (2) or a trajectory from that same patient i starting after timestep 8.

- **Component loss.** When computing the component loss, recall that for an anchor signal, other signals from the same trajectory are *not* used as negatives, in order to minimize correlation between the anchor and negatives. Therefore, negative pairs for an anchor signal are either signals from a different patient, or signals from that same patient from further off in time.

This strategy of sampling trajectories that do not overlap was applied to ensure that we do not use highly correlated trajectories/signals as negatives. We note that this strategy is not necessarily optimal, and that different approaches could be used for both the component and global loss terms. Our framework could easily be used with these other sampling strategies and loss formulations.

To generate FT sets, we first use a sliding window to select contiguous 8 hour blocks at 1 hour increments from each visit. Each of these contiguous 8 hour blocks becomes a trajectory in the FT dataset. The trajectory labels are formed based on the nature of the specific task. For example, for the 24 hour mortality task, the label is based on whether the patient dies within 24 hours of the ending time of that trajectory.

Trajectory Features and Preprocessing. The trajectories in PT and FT sets consist of static features d and a time-series of structured data and physiological signals $\{(w_t, s_t)\}_{t=1}^{T}$, as presented in Section 5.3.1.

In Dataset 1, $d \in \mathbb{R}^9$ contains the following features from the EHR: BUN, Chloride, CO2, Creatinine, Glucose, Potassium, Sodium, Systolic Blood Pressure, Diastolic Blood Pressure. We take the average if multiple values are recorded in each time window. $w_t \in \mathbb{R}^{30}$ has mean, standard deviation, maximum, and minimum of heart rate and SpO2 recorded within each hour window (this is sourced from the telemetry monitor), and also 22 heart rate variability features from the ECG recorded by the telemetry monitor during each time window. $s_t \in \mathbb{R}^{4\times 2400}$ is a 10 second 4-channel ECG measured at 240 Hz, containing leads I, II, III, and V1, extracted from a longer

ECG measured by the telemetry monitor during that hour window.

In Dataset 2, $d \in \mathbb{R}^{38}$ contains the following features from the EHR: FiO2, Glucose, Temperature, pH, and one-hot encoded Glasgow Coma Scale measures, following [72]. $w_t \in \mathbb{R}^{13}$ contains the following information from the physiological signals database: mean, standard deviation, maximum, and minimum of heart rate and SpO2 recorded within each hour window from the telemetry monitor, diastolic blood pressure, systolic blood pressure, mean blood pressure, heart rate, and SpO2 from the EHR. $s_t \in \mathbb{R}^{1\times 1250}$ is a 10 second 1-channel ECG measured at 125 Hz, containing lead II, extracted from a longer ECG measured by the telemetry monitor during that hour window.

Missing structured data are forward-fill imputed where possible (for example, if part of a time series) and otherwise imputed with the mean over the training dataset. Missing signals are represented with zeros. We drop any trajectories that have more than 1 timestep with a missing signal. For Dataset 2, we note that by dropping any trajectory with more than 1 timestep with a missing signal, we have a smaller dataset than in [72].

Forming labels. The FT labels for the Elevated mPAP task are formed based on the PA pressure waveform recorded for patients (when available) – if the mean pressure is over 20 mmHg over a 1 minute period at the final timestep of the trajectory, we assign a binary label of 1, and else 0. For the 24 hour mortality task, we use the recorded time of death recorded in the EHR and if it is less than 24 hours from the end time of the trajectory, we assign a label of 1, and otherwise 0. This is as was done in [72].

D.2.2 Experimental Setup

Model Architecture

We use the following architecture for the encoder and projection head for all methods:

- **Encoder**: Each signal s_t in the trajectory is passed through a ResNet-styled

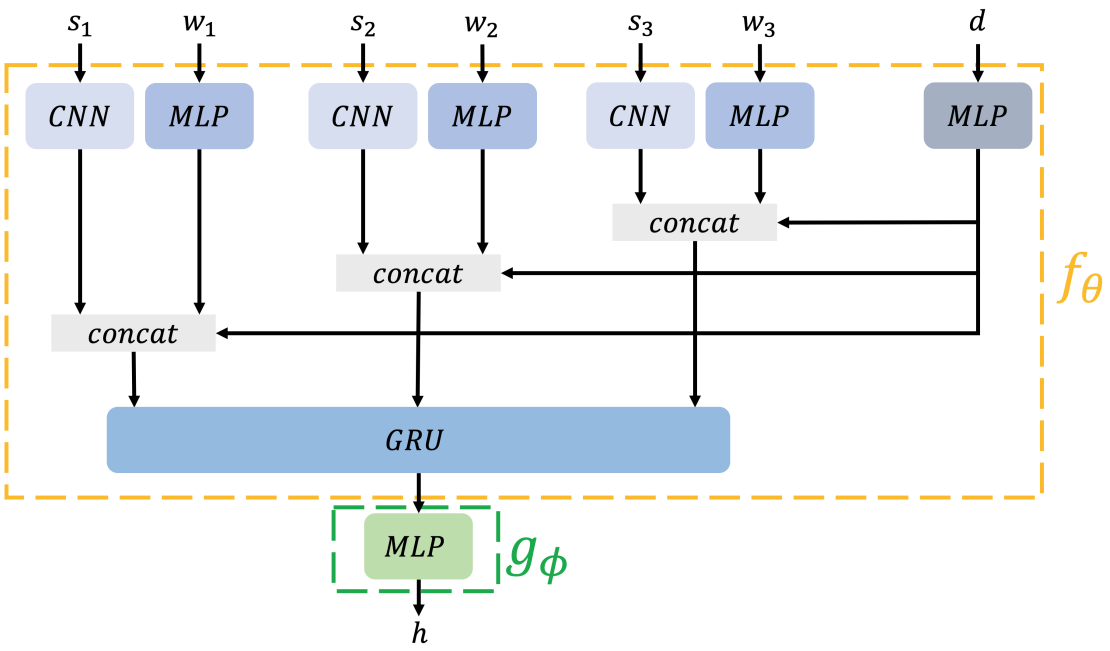


Figure D-1: **Model architecture used in our experiments.** We show the architecture used to model trajectories in a scenario where the input trajectory has 3 timesteps.

1-D CNN encoder with global average pooling. We base our CNN encoder model off a ResNet-18 architecture with kernel size of 15. Following global average pooling over the temporal dimension, each signal is projected into 128 dimensions with a linear layer.

The structured data w_t at each timestep is embedded with a 2-layer fully-connected network with 128 hidden units and ReLU activation at each layer, and this embedding is then concatenated with the signal embedding. The static features d are passed through a different 2-layer fully-connected network with 128 hidden units and ReLU activation at each layer, and then concatenated with the embeddings of the signal and structured data timeseries at each timestep. The resulting sequence of vectors is passed into a 4-hidden layer GRU with hidden size of 384, with the last hidden state of the GRU being used as the overall trajectory embedding vector.

- **Projection heads**: The two projection heads for the signal and the trajectory are both 2-layer fully connected networks with batch normalization and ReLU

activation with 2048 hidden units. The trajectory projection head takes the last hidden state of the GRU as input, and the signal projection head takes the output of the signals encoder as the input. When using the NT-Xent loss, the resulting projection is normalized [30] before computing the NT-Xent loss over the batch.

Figure D-1 shows the model architecture in a scenario in which the input trajectory has 3 timesteps.

SSL Methods: Implementation

We describe the implementation of the SimCLR and VICReg methods in the main paper, Section 5.3. For SimSiam, we follow the setup in [34] and use a predictor network in the trainable branch, and minimize cosine distance between the output of the predictor network in the trainable branch and the output of the projection head in the stop-gradient branch.

For simplicity, we let this predictor network have the same architecture as the projection head – a two layer fully connected network with batch normalization and ReLU activation, with 2048 hidden units. We did not find a bottleneck structure to improve performance in initial investigations, but further experiments may be warranted here.

Loss, Architecture, and Optimization Hyperparameters

There are various hyperparameters to tune, such as learning rates, loss weighting for VICReg loss terms, and loss weighting for SMD SSL. Evaluating many hyperparameter settings is very computationally expensive (since it entails doing both a PT and FT run), so we conduct a reduced search on a subset of the hyperparameters focusing only on the Elevated mPAP task in the unimodal setting, optimizing validation AUROC.

In our hyperparameter search, we use the following setup:

- Learning rate: tune on a randomly initialized model for the elevated mPAP

task on each dataset, and then use this learning rate for all other experiments. We compared Adam with a learning rate of 1e-4, 3e-4, 1e-3, and 3e-3. We found 1e-3 to be the most stable and best performing.

- VICReg loss weights: Tune these for the VICReg (global) model only, and use the best hyperparameters for all other uses of the VICReg loss, including SMD SSL (VICReg). Following the original paper, we set the covariance weight $\nu = 1$ and then tune the invariance weight λ and variance weight μ. We found in early experiments that the variance weight did not have much impact on performance, and so focused on the invariance weight, studying $\lambda = 1, 2, 5$. We found $\lambda = 1$ to perform the best on both datasets.

- SMD SSL (SimCLR) component loss weight: Set the global weight $\alpha = 1$ in Eqn. 5.5, and tune the component weight β, on both datasets separately, comparing $\beta = 0.25, 0.5, 1.0, 2.0$. We found $\beta = 1.0$ to perform the best on Dataset 1, and $\beta = 0.25$ to perform the best on Dataset 2.

- SMD SSL (VICReg) component loss weight: Use the best VICReg loss weights found above, set the global weight $\alpha = 1$ in Eqn. 5.5, and tune the component loss weight β, comparing $\beta = 0.1, 0.25, 0.5, 1.0, 2.0$. We found $\beta = 1.0$ to perform the best on Dataset 1, and $\beta = 0.1$ to perform the best on Dataset 2.

We fixed the temperature of the NT-Xent loss to 0.1, following [204].

We did not conduct tuning of the architecture hyperparameters, and instead opted to use architectural choices that were found to be effective in previous works, such as a ResNet signal encoder [143, 148], a wide projection head [31, 14], and a GRU sequence model [120]. Similar to [120], we did not find a transformer model to be beneficial as the sequence model, though perhaps architectural tuning could improve its performance.

Compute Details

All models were trained on either a single NVIDIA Quadro RTX 8000 or a single NVIDIA RTX A6000 GPU. Pre-training takes about 8 hours on Dataset 1 and about 2 hours on Dataset 2. Fine-tuning on Dataset 1 tasks takes about 4 hours. Fine-tuning in Dataset 2 on Elevated mPAP takes about 30 minutes, and about 4 hours on 24hr Mortality. Pre-training uses approximately 20 GB of GPU memory, and fine-tuning uses approximately 10 GB of GPU memory.

D.2.3 Additional Results

Studying loss curves. Figure D-2 shows training loss curves for SimCLR-based models on Dataset 2. We observe that SMD SSL effectively minimizes both component and global losses over training. Considering the component loss alone (left plot), we see that this loss naturally reduces during training of the SimCLR (global) model even though this is not explicitly enforced during model training – we compute the component loss in this case with a randomly initialized signal level projection head that is not updated, and the network parameters are also not updated to minimize this component loss. An analogous situation with SimCLR (component) and the global loss is seen in the right plot. This suggests that training with one of the losses does encourage structure in both representation spaces, even with random projections, but this structure is more clearly defined when the loss is explicitly minimized (as in SMD SSL).

Trends in AUPRC. Since the mortality task has low prevalence, we study trends in AUPRC among methods as they compare to AUROC. We find that in the unimodal setting, on both datasets, our objective improves AUPRC by 1-2% over baselines, but AUPRCs are all relatively low (<5%) due to the limited predictive signal in the ECGs alone for the mortality prediction task. In the multimodal setting, on Dataset 1, SMD SSL worsens AUPRC over the single-level approach by 2.9% AUPRC (10.8% vs 7.9%) – this is consistent with what was seen with AUROC. On Dataset 2, the AUPRC with

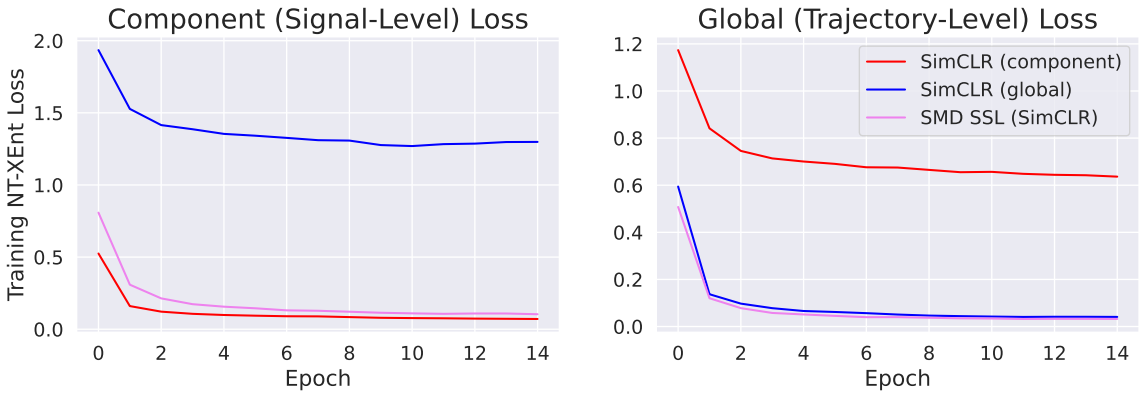


Figure D-2: **Studying training loss curves for SMD SSL and variations.** We observe that SMD SSL effectively minimizes both the component and global NT-XEnt losses during training. Interestingly, we observe that the NT-Xent loss computed on the signal level and trajectory level reduces somewhat even when it is not explicitly minimized. To see this, consider the SimCLR (global) method and the component Loss – the component loss reduces over training even with a random projection head, without adding this term to the objective. This indicates that the global loss and component loss are not entirely independent (as is expected).

our approach is 28.4%, an improvement of about 4% over the best baseline.

Additional baselines. As discussed in the main text, related SSL strategies for time series data are not exactly applicable in our setting since they formulate pipelines for structured data-only time series (rather than multimodal time series), or are concerned with individual physiological waveforms (rather than sequences of waveforms).

Despite these differences, for completeness, we study here the performance of adapted versions of three related methods from the literature for Dataset 2 (MIMIC-III), focusing on the multimodal setting. Specifically, we evaluate the SSL objective functions proposed in NCL [204], SACL [35], and CLOCS (specifically the CMSC formulation) [98]. We use each of these losses in a global-only SSL setup in order to compare how they perform to our proposed two-level loss function. We additionally evaluate structured data-only NCL.

We evaluate these methods using the same experimental setup (augmentation pipeline, optimization hyperparameters, model architecture, etc) as what was used when evaluating our method. For NCL, we considered two values of α (0.3 and 0.5), and fixed $w = 16$ as in the original paper's configuration on MIMIC. We select the

Table D.2: Test AUROC of different SSL algorithms on Dataset 2 (MIMIC) in the multimodal setting. We observe that SMD SSL (VICReg) improves on these additional SSL baselines.

	Elevated mPAP	24hr Mortality
SMD SSL (VICReg)	71.6	90.7
NCL (Structured data only)	67.6	87.7
NCL (Multimodal)	65.5	89.9
SACL	64.7	89.8
CLOCS	64.3	89.9

value of α that obtained the best validation AUROC on each downstream task.

Results are shown in Table D.2. As seen, the best two-level approach, SMD SSL (VICReg), outperforms the different baselines. This indicates that a two-level loss is not easily outperformed by other global-only loss functions. An important investigation is to conduct a more thorough hyperparameter search for these alternative loss functions, and also evaluate whether two-level versions of these other objectives could improve on SMD SSL (VICReg).

Linear Evaluation vs Full Fine-tuning. As discussed in the main paper, our goal is to develop a self-supervised pre-training algorithm that finds an effective model initialization for adaptation to downstream tasks (i.e., a transfer learning setting). As a result, we evaluate both full FT and linear evaluation, reporting the evaluation strategy that obtains the best validation AUROC on a per-method and per-task basis. It is important to consider full FT since it almost always outperforms linear evaluation – we observed this in our results, and a similar finding was see in the evaluation from [120]. Table D.3 highlights this finding for a subset of the methods on Dataset 2 (MIMIC), in the multimodal setting.

Studying longer Pre-training. On Dataset 2 (MIMIC), we pre-trained methods for longer (50 epochs) and compared performance after 15 and 50 epochs, following the best of full FT and linear evaluation (following our standard experimental setup). Results are in Table D.4, indicating that performance did not improve following longer PT.

Table D.3: Comparing test AUROC of the two evaluation paradigms — Linear Evaluation and Full Fine-tuning (FT) — with selected SSL algorithms on Dataset 2 (MIMIC) in the multimodal setting. We find that Full FT routinely performs better than linear evaluation.

	Elevated mPAP		24hr Mortality	
	Full FT	Linear Evaluation	Full FT	Linear Evaluation
RandInit	65.3	N/A	87.8	N/A
SMD SSL (VICReg)	71.6	65.0	90.7	72.2
VICReg (Global)	70.4	64.4	87.8	71.7
SimCLR (Global)	63.7	63.1	86.8	71.7
SimSiam (Component)	67.4	61.6	90.6	71.9
SimSiam (Global)	60.6	50.6	90.4	50.0

Table D.4: Comparing test AUROC after different amounts of PT with selected SSL algorithms on Dataset 2 (MIMIC) in the multimodal setting. Performance does not appear to improve after more PT.

	Elevated mPAP		24hr Mortality	
	15 epochs PT	50 epochs PT	15 epochs PT	50 epochs PT
SMD SSL (VICReg)	71.6	66.6	90.7	89.3
VICReg (Global)	70.4	63.2	87.8	78.8
SimSiam (Component)	67.4	59.6	90.6	78.8
SimSiam (Global)	60.6	51.1	90.4	88.2

Additional representational similarity experiments. In the main text, we presented a simple representational similarity study examining the how the learned representations by the CNN signals encoder compared in SMD SSL (SimCLR) vs. SimCLR (component only) and SimCLR (global only) pre-training. We found that SMD SSL representations had reasonable Centered Kernel Alignment (CKA) similarity [101] with both component-only and global-only PT. On the other hand, global-only and component-only PT were quite dissimilar.

To further understand the effect of training with the component and global losses in SMD SSL, we conduct a finer-grained CKA study. We take the output of each residual block of the CNN signals encoder and compare the CKA similarity in these representations (following pooling over the sequence dimension) to the CKA similarity of component-only and global-only PT models, on a per-block basis. That is, we average the CKA similarity between SMD SSL (block i) and component-only PT

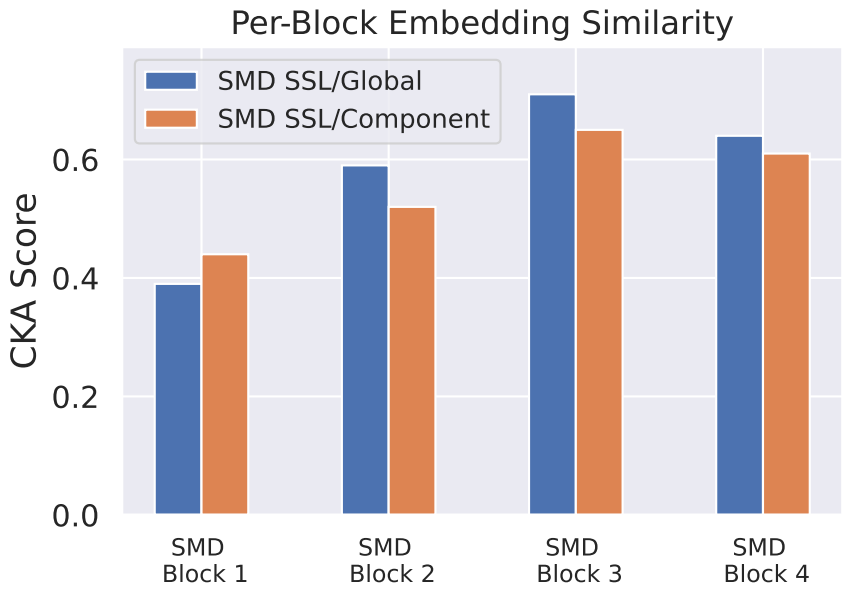


Figure D-3: **Studying per-block representational similarity in the CNN encoder between SMD SSL pre-training and component-only and global-only pre-training.** SMD SSL representations are more similar to component-only PT early on in the CNN signals encoder, and more similar to global-only PT deeper in the network.

(blocks $1, 2, 3, 4$), and similarly for global-only PT. The results are shown in Figure D-3. We see that SMD SSL PT has more similarity with component-only PT in the first block, and greater similarity with global-PT in the remaining blocks. This suggests that the component loss is having the most impact on SMD SSL representations in the earlier CNN layers, indicating that these low-level features are particularly relevant for minimizing the component loss – this makes sense, since we would expect lower-level features to matter more in a per-signal embedding, and global-level features to matter more in a sequence-level embedding.

Appendix E

ECG-guided Non-invasive Estimation of Pulmonary Congestion in Patients with Heart Failure

E.1 Introduction

In the main thesis, we outlined contributions to data-efficient machine learning (ML): strategies to build effective ML models given only limited labelled training data. One application area that motivated several of these methodological contributions was using ML models for diagnosis in cardiovascular medicine, particularly using rich data modalities such as the electrocardiogram (ECG). Here, we describe a more application-focused contribution we make to the area of ML for cardiovascular medicine: building deep learning models to assist in non-invasive monitoring of patients with heart failure.

As some background, heart failure (HF) is a global public health burden with a prevalence of approximately 12% in individuals over 60 years of age [187]. Despite advances in diagnosis and therapy, patients with HF remain at increased risk for a variety of adverse outcomes including repeat hospitalization and death [28].

The clinical evaluation of patients with HF involves an assessment of their un-

derlying hemodynamics, with the cardiac output and mPCWP – an estimate of the left atrial pressure – being quantities that are used to assess the hemodynamic severity [48]. Although originally developed for patients who present with an acute myocardial infarction, the Forrester classification scheme has been validated in patients with heart failure and continues to provide prognostic information in these patients [58, 37, 17, 18]. Forrester identified four distinct hemodynamic subsets based on two hemodynamic parameters: the mPCWP and the cardiac index (CI). Patients with a mPCWP $\geq$ 18 and a CI $\geq$ 2.2L/min/m2 ("dry-warm") had the best prognosis, while patients with a mPCWP $>$ 18mmHg and a CI $<$ 2.2L/min/m2 ("wet-cold") had the worst prognosis [58, 131]. In addition, the mPCWP has particular importance in clinical assessment as a mPCWP$>$18mmHg is an independent predictor of adverse outcomes, while the CI is not [1, 69].

Unfortunately, both the mPCWP and CI are challenging to estimate from the clinical exam alone and are best are obtained via insertion of a pulmonary-artery catheter, which cannot always be performed safely and expeditiously in many clinical settings [51]. Current methods for the non-invasive estimation of mPCWP rely on analyses of mitral inflow velocities, obtained from a cardiac ultrasound [128, 127, 181]. However, this approach necessarily requires the acquisition of spectral Doppler images from an experienced sonographer and therefore may not be routinely available.

Deep learning (DL) holds the promise of leveraging non-invasive measurements that are easily obtained in a many clinical venues to estimate quantities that are typically acquired via an invasive study. Recent work developed DL models to estimate when the mPCWP is elevated using CXR images [80]. This approach, however, was not specifically developed for, nor tested in, patients with known HF. As ECG parameters have been shown to be correlated with abnormal hemodynamic profiles in some patient populations [185], the 12-lead ECG serves as a potential data source that can be leveraged to estimate central pressures. Recently, we developed a deep learning model to estimate when the mPCWP is greater than 15mmHg from ECG data using a heterogeneous cohort of patients who were referred for right heart catherization [164]. While the discriminatory ability of the model was good in a subset of patients who

Table E.1: Model performance (AUROC) on test data. HFNet significantly outperforms the baseline logistic regression (LR) model. Asterisk (*) indicates p-value < 1e-10.

Model	AUROC	
	Internal Test Set	External Holdout Set
LR	0.71 ± 0.01	0.67 ± 0.01
HFNet	0.82 ± 0.01 *	0.81 ± 0.01 *

were referred for an evaluation of heart failure, a mPCWP cutoff of 15mmHg at rest has not been shown to risk stratify patients with chronic heart failure [1]. Indeed, our previous model was intended to be a screening tool to rule out an elevated mPCWP in all patients over 60 years old. [164].

Our contribution here is to use an independent cohort of HF patients to develop and validate a model that uses the 12-lead ECG and simple demographic features (age, sex) to detect whether mPCWP > 18mmHg. As in our previous work, our underlying hypothesis is that subtle changes in the ECG, which are difficult to detect by visual inspection alone, have diagnostic significance. We also developed a measure of model uncertainty using the predictive entropy of the model, which provides insight into when a patient-specific model prediction is likely to be untrustworthy.

E.2 Results

E.2.1 HFNet Discriminatory Performance

Table E.1 shows the AUROC of our method, HFNet, and a baseline logistic regression model that uses ECG intervals, age, and sex to detect an elevated mPCWP > 18mmHg. HFNet achieves an AUROC of 0.82 ± 0.01 on the internal holdout set, and significantly outperforms a logistic regression model trained using age/sex and ECG intervals. The associated receiver operator characteristic curves for each model are shown in Figure E-1.

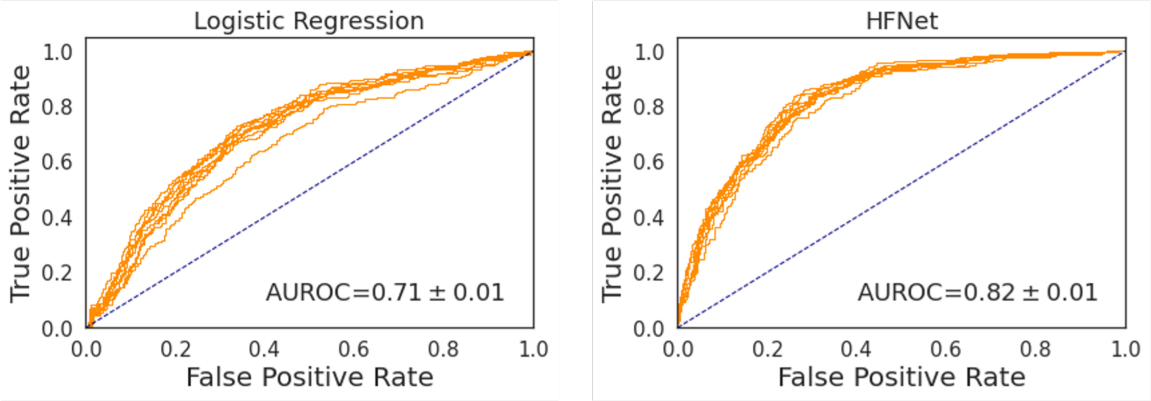


Figure E-1: Receiver Operator Characteristic on the internal holdout set for (a) Logistic Regression baseline and (b) HFNet, showing 10 bootstraps. HFNet obtains significant improvements over the logistic regression baseline (paired t-test, p < 1e-10).

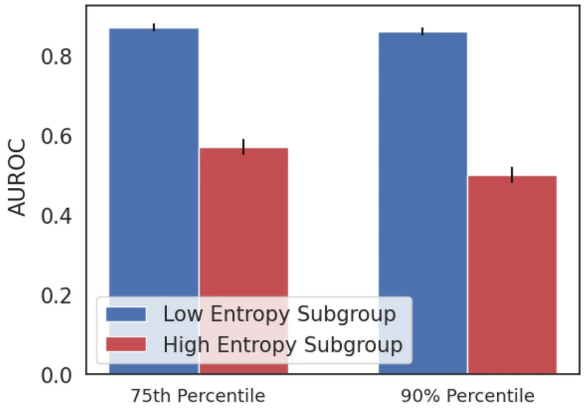


Figure E-2: Performance stratified by entropy-based uncertainty score, showing mean and standard deviation over 10 bootstraps. HFNet performance is improved in cohorts with low uncertainty.

E.2.2 A Prediction-Specific Uncertainty Score

We developed a prediction-specific score, based on the prediction entropy, to identify instances in which model performance is likely to be poor. We hypothesized that the set of predictions with high prediction entropies correspond to a subset where model performance is poor. The discriminatory ability of the model is reduced in cohorts that have high predictive entropies (Figure E-2). We compute entropy percentiles on the "dev" set, and find that the cohort consisting of patients with entropies below the 90th percentile has a testing set AUROC of 0.86 ± 0.01; on predictions within the top entropy decile, the AUROC is 0.50 ± 0.02. At the 75th percentile entropy

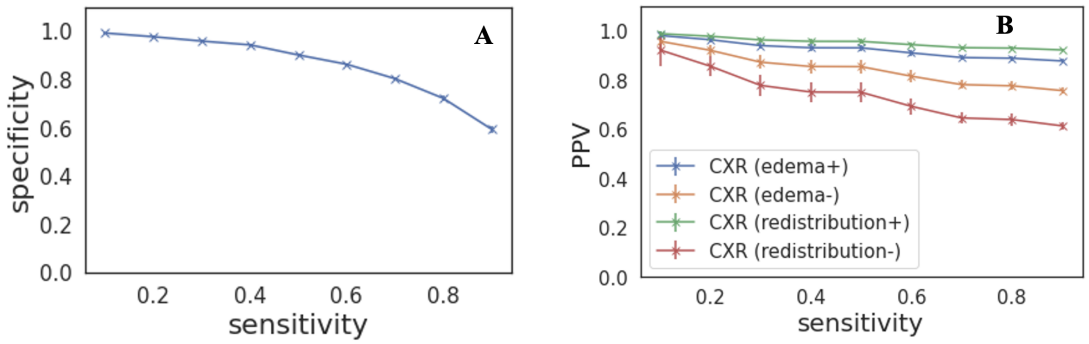


Figure E-3: **HFNet predictive value.** (A) Specificity vs Sensitivity on the entire internal holdout set; (B) PPVs on the subcohort of the internal holdout set with reduced Ejection Fraction (EF $<$ 40) and as a function of CXR findings on present on presentation. CXR(edema+) = evidence of interstitial edema; CXR(edema-) = interstitial edema absent; CXR(redistribution+) = evidence of pulmonary vascular redistribution; CXR(redistribution-)=no pulmonary vascular redistribution noted. Prevalence of different CXR findings in patients with reduced LV function taken from [26]. Plots show mean and standard deviation over 10 bootstraps.

threshold, the AUROC is 0.87 $\pm$ 0.01, and within the top quartile of entropy values, the AUROC is 0.57 $\pm$ 0.02.

E.2.3 HFNet Predictive Value

Sensitivity and Specificity plots for HFNet for the entire cohort are shown in Figure E-3. Using a threshold that achieves a sensitivity of 80% on the "dev" set, the associated specificity is 0.72 $\pm$ 0.01. Using our prediction-specific uncertainty score at the 90th percentile entropy threshold, and again an 80% sensitivity threshold, the model specificity increases to 0.83 $\pm$ 0.01. At the 75th percentile entropy threshold, the specificity increases to 0.91 $\pm$ 0.01.

To determine the predictive value of the model in the presence/absence of clinical findings consistent with volume overload, knowledge of the sensitivity, specificity and prevalence of an elevated mPCWP are needed (see equation 1). Since patients in our cohort do not reliably have a chest X-ray (CXR) in close proximity to when the RHC is performed, we rely on previously published data to estimate the prevalence of mPCWP $>$ 18mmHg in HF patients with reduced Ejection Fraction (HFrEF) in the setting of different CXR findings (see Table E.3) [26]. Figure E-3 shows the calcu-

lated PPVs as function of sensitivity. In patients who have CXR findings consistent with interstitial edema, the PPV of the model is 0.89 ± 0.01, using a threshold corresponding to an 80% sensitivity. In patients who do not have evidence of interstitial edema on CXR, the PPV of the model 0.78 ± 0.02, again using a threshold that captures 80% of the true positive values. For the isolated CXR findings of vascular redistribution, the PPV of the model is 0.93 ± 0.01, and when there are no signs of vascular redistribution the PPV is 0.64 ± 0.03. Negative predictive values for the model are not as informative; i.e., in the presence of the CXR consistent with volume overload the NPV is 0.34 ± 0.02, and in the absence of such findings the NPV is 0.54 ± 0.03.

E.2.4 HFNet Tracks Patient Specific Changes in mPCWP

We assessed the ability of the HFNet to predict patient specific changes in hemodynamics. For each patient in the internal test set who had two or more right heart catheterizations, we computed the average accuracy of the model for detecting elevated mPCWP on that patient's sequence of catheterizations; e.g., an accuracy of 100% means that the model reproduces the trend of mPCWPs measured across of that patient's invasively measured mPCWPs. Figure E-4 summarizes the results. For patients with multiple mPCWP measurements, the model achieves an average accuracy of 78.7 ± 2.4%, and the lower the entropy, the greater the accuracy. In the cohort with entropy values below the 75th percentile, the accuracy is approximately 87%.

Figure E-4 also plots model predictions and mPCWP for two patients in the internal test set (additional examples are shown in Figures E-9 and E-10). These data demonstrate that the model's output can reproduce trends in the mPCWP over time and captures both reductions and increases in mPCWP over time.

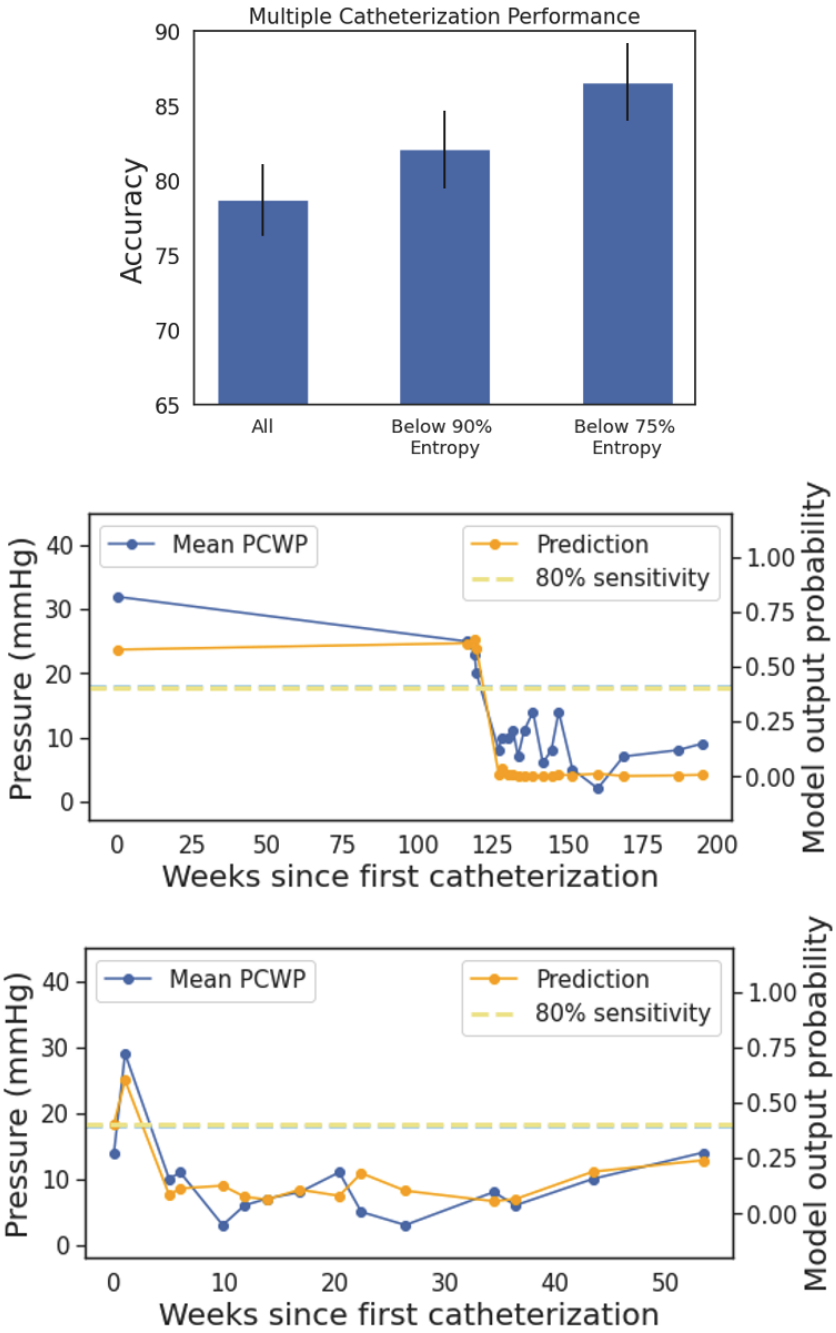


Figure E-4: **Performance on patients with multiple catheterizations.** (A): Showing mean accuracy and standard error on sequences of catheterizations for patients at varying degrees of prediction uncertainty (N=136). (B,C): Two examples of patients who had multiple catheterizations, showing measured mPCWP and model predictions. The blue dashed line corresponds to 18mmHg mPCWP (left axis), and the dashed green line indicates the model output threshold corresponding to an 80% sensitivity level.

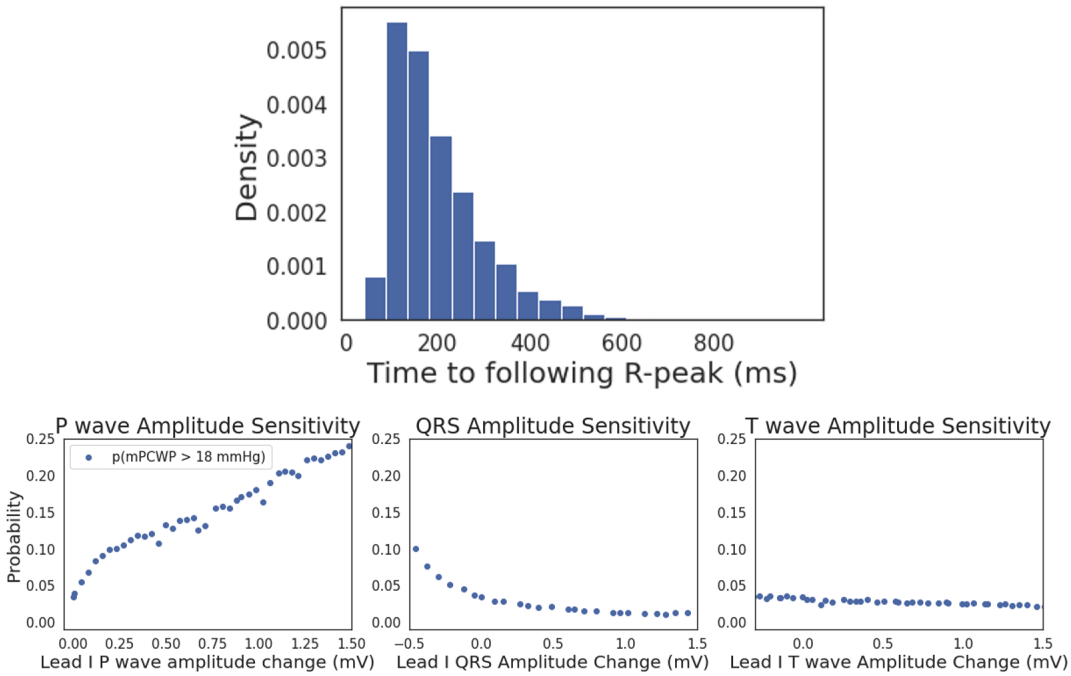


Figure E-5: **Model sensitivity and saliency map analysis.** Top: A saliency-map analysis showing that over 95% of the samples in the test set ECGs with the highest saliency score occur within 400ms of the subsequent R-peak. Bottom: Model output as a function of changes to the ECG, indicating that the model output is most sensitive to changes in the P-wave amplitude.

E.2.5 Saliency Analysis and Model Sensitivity

Saliency maps constitute one often-used method for identifying what portions of an ECG are most important for model decision making [164]. The method produces a "saliency value" for each sample in the ECG signal that quantifies the importance of that sample for the model's prediction. For each ECG in the test set, we computed a saliency map for the input ECG, take the 100 highest saliency points in each ECG and compute the average time between the highest saliency point and the subsequent R-peak. Figure E-5 presents the results. Over 95% of the highest saliency points are within 400ms of the subsequent R-peak, indicating that the diastolic portion of the cardiac cycle has the greatest influence on model predictions.

To determine what specific ECG features (e.g., P-wave, QRS complex, T-wave) most influence model predictions, we generated synthetic ECG data using a previously described ECG generation tool that defines a dynamical system that can be

used to produce synthetic 12-lead ECG waveforms [163]. The method enables us to systematically vary portions of the ECG signal to determine how sensitive the model output is to modifying the amplitude of different regions of the cardiac cycle. Results are shown in Figure E-5. Of the three perturbations, increasing the P wave amplitude most affects the model output probability. Reducing the QRS amplitude elevates the predicted probability, though not as much as increased P wave amplitude. Changes to the T wave amplitude have little effect on the model output probability.

E.2.6 Validation on an external dataset

We assessed HFNet performance using data from a second institution (See Table E.1); Figure E-6 summarizes our key findings. HFNet significantly outperforms ($p < 1e\text{-}10$) the baseline logistic regression model on this dataset – HFNet achieves an AUROC of 0.81 $\pm$ 0.01, compared to an AUROC of 0.67 $\pm$ 0.01 for the baseline model (Table E.1, Figure E-6).

Using our prediction-specific uncertainty score, we find that at the 90th percentile entropy threshold, the AUROC increases to 0.82 $\pm$ 0.01, and on the subgroup with high uncertainty it is 0.56 $\pm$ 0.04. Using the 75th percentile entropy threshold, the AUROC is 0.81 $\pm$ 0.01, and in the cohort with high uncertainty, the AUROC is 0.60 $\pm$ 0.02. This suggests that the uncertainty score also is effective on the validation dataset. Figure E-6 shows the specificity vs sensitivity plot for the external holdout set, and it is similar to what was observed on our internal test set. Using a threshold that achieves a sensitivity of 80% on the dev set, the specificity on the external holdout set is 0.82 $\pm$ 0.01; i.e., similar to what we obtained with the internal test set. Using our prediction-specific uncertainty score and the 90th percentile entropy threshold, at an 80% sensitivity threshold, the model specificity increases to 0.89 $\pm$ 0.01. At the 75th percentile entropy threshold, the specificity is 0.92 $\pm$ 0.01. We do not have ejection fraction (EF) data for Hospital 2, so we could not calculate metrics in the reduced EF cohort with different clinical findings. Considering patients with multiple catheterizations, HFNet achieves an average accuracy of 76.2 $\pm$ 1.8% across multiple catherizations per patient. The accuracy increases to above 80% in the cohort for

predictions that are in the lower 75th percentile of entropy.

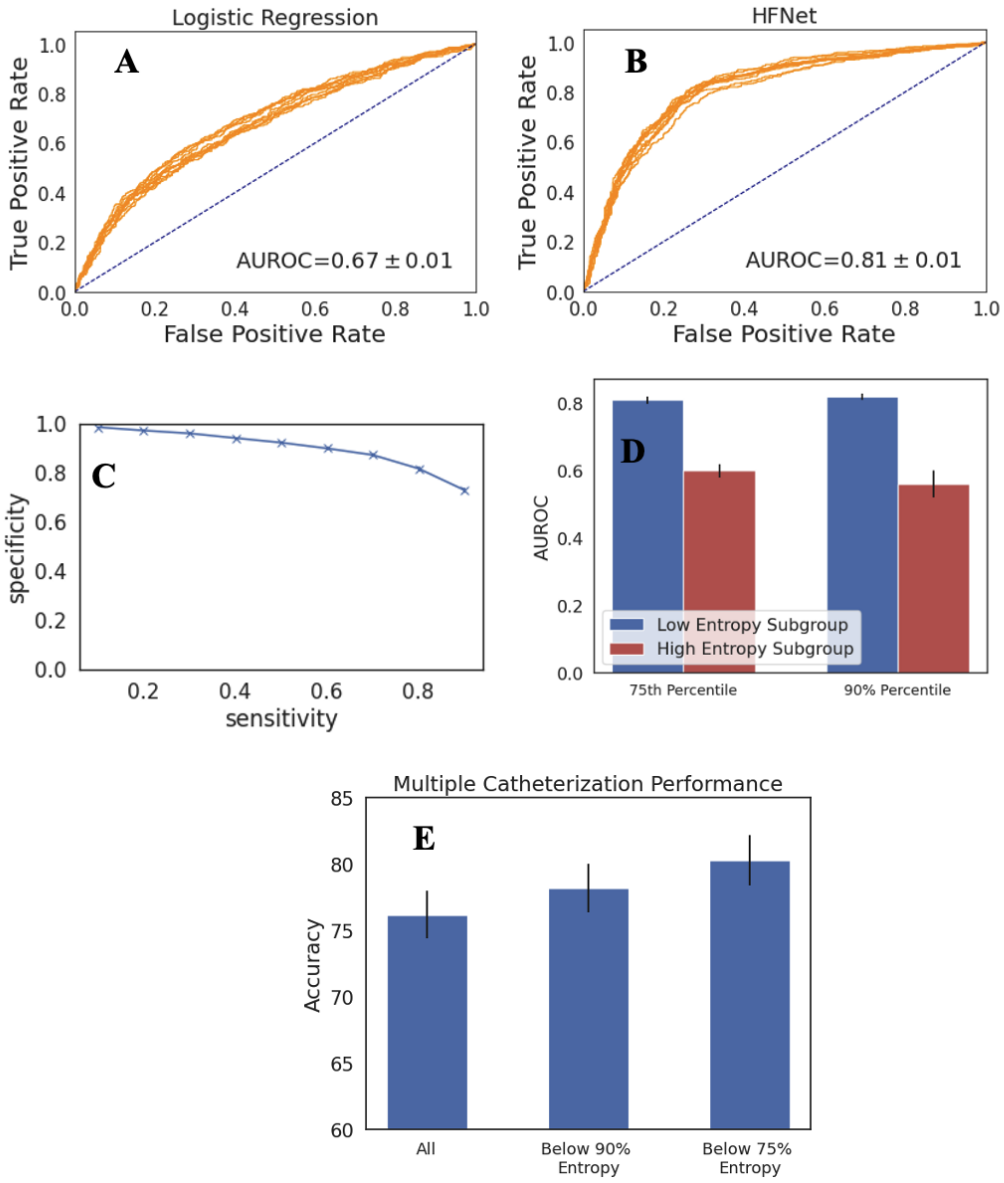


Figure E-6: **Validation on external dataset.** On a dataset from a second institution, showing: (a),(b) Receiver Operator Characteristics for baseline logistic regression and HFNet over 10 bootstraps; (c) Performance stratified by prediction uncertainty score showing mean and standard deviation over 10 bootstraps; (d) Sensitivity and Specificity and PPV/NPV on the external dataset showing mean and standard deviation over 10 boostraps; (e) Sequence accuracy for patients with multiple catheterizations, showing mean and standard error (N=281). The performance trends demonstrated on the internal test set are consistent in the external validation set.

E.3 Discussion

In this study we developed a deep learning model to estimate the probability that a patient's mPCWP is elevated, using a threshold of 18mmHg in patients who have a prior diagnosis of HF. The resulting model has good discriminatory ability across a cohort of HF patients from two different hospitals – the AUROC scores on the internal and external holdout datasets were 0.82 ± 0.01 and 0.81 ± 0.01 respectively.

While good discriminatory ability is a necessary condition for a clinical useful model, this, by itself, is far from sufficient [165]. For example, given that no model is ever, in practice, accurate 100% of the time, understanding failure modes of any model is important for high stakes decision making. We therefore developed a prediction-specific score, based on the entropy of the model output, which quantifies how certain the model is when it makes a prediction. We demonstrate that model discriminatory ability is significantly reduced when the associated entropy is high. Overall, the degree to which a specific model prediction should affect clinical decision making should be weighted by the associated prediction-specific reliability score.

When studying our model's performance for patients with multiple catheterizations, we find that its predictions can approximately track the trends in the patient's ground truth mPCWP measurements through catheterization. This is most evident when considering the model's most confident predictions (with low entropy), and this trend is observed in both the internal and external datasets.

To gauge how HFNet could be integrated into clinical practice, we computed predictive values for patients with HF and reduced left ventricular ejection fraction (LVEF) in different clinical scenarios of interest. As the CXR forms a routine part of the evaluation of HF patients in the emergency room and in inpatients with a HF exacerbation, we focus on the added value of model predictions given different CXR findings. When signs of pulmonary congestion are evident on CXR, then the diagnosis of HF is very likely. In this setting HFNet provides incremental benefit, as its role would be to confirm the diagnosis of cardiogenic pulmonary edema. However, the CXR is not always a reliable indicator of cardiogenic pulmonary edema in patients

with HF. For example, several studies suggest that approximately 20% of patients with acute decompensated HF present with normal CXRs and up to 50% of patients with HF have a mPCWP $>$ 18mmHg and CXR findings that are not consistent with vascular redistribution or interstitial edema [26, 192]. Hence, arriving at the right diagnosis is challenging in HF patients with suspected decompensated heart failure who have unremarkable chest x-ray findings. Plasma levels of B-type natriuretic peptide (BNP) and N-Terminal pro-BNP (NT-proBNP) have high negative predictive value for ruling out acute heart failure in patients who present with dyspnea [118]. However, since BNP and NT-proBNP levels can be persistently elevated in patients with chronic HF, the positive predictive value of natriuretic peptides in the diagnosis of acute decompensated HF in these patients is not as clear [45].

Using estimates of the prevalence of mPCWP $>$ 18mmHg in HFrEF patients, given different radiographic pulmonary findings, and a decision threshold corresponding to a sensitivity of 80%, we estimate that the PPV is 0.89 $\pm$ 0.01 when the CXR is consistent with interstitial edema. When signs of interstitial edema are absent the PPV is 0.78 $\pm$ 0.02, again at a sensitivity of 80%. HFNet can therefore provide information that complements CXR findings in patients with HF. Indeed, an HFNet prediction consistent with an elevated mPCWP is meaningful even if the CXR does not show signs of pulmonary edema; i.e., a diagnosis of acute HF is likely in patients with a positive HFNet prediction and negative CXR findings. Moreover, given that a persistently elevated mPCWP after directed therapy for a HF is associated with adverse outcomes [40], HFNet could be part of the evaluation of HF patients before discharge, even when the CXR may be unremarkable. When considering whether a given inpatient with HF is ready for hospital discharge, a positive prediction should raise the suspicion that their filling pressures remain elevated and that additional inpatient therapy is required. Lastly, we note that while we focus on a decision threshold corresponding to a sensitivity of 80% – a common cutoff used in the medical literature – we note that higher PPVs are achieved at lower sensitivity values (as shown in Figure E-3).

Deep learning models suffer from the criticism that they are opaque algorithms

because it is challenging to understand what these models have learned in practice [165]. In order to probe what HFNet has learned, we used two complementary approaches. The first method, saliency map analysis, is a widely used approach for understanding what portions of the input data have the greatest influence on model predictions. The second approach used synthetic ECG sequences to systematically vary the amplitude of different portions of the ECG to study how the model output varies as different portions of the ECG are modified. Both approaches suggest that HFNet preferentially focuses on the diastolic portion of the cardiac cycle, with the amplitude of the P-wave having the greatest influence on model output. These data are consistent with the longstanding clinical intuition that the mPCWP is a reasonable estimate for left sided pressures at end diastole in many patients [196]. While this does not constitute a comprehensive explanation of what the model has learned, it does suggest that HFNet has garnered information that is consistent with our prior understanding of human pathophysiology.

While we believe these data suggest that HFNet has a role in the evaluation and management of patients with HF, our study has limitations. Our approach estimates whether the mPCWP is above a threshold or below it, rather than predicting the precise value of the mPCWP itself. Although a finer grained prediction would be more useful, the method here identifies patients who have pulmonary congestion and helps to prioritize who should have additional investigative studies. More data are needed to develop a robust method that can precisely estimate the mPCWP. Also, when curating our dataset of paired ECGs and mPCWP measurements, we ensured that the ECG and catheterization were performed on the same day, however, the specific time of catheterization relative to the ECG recording is not known and this introduces some uncertainty in our results. Obtaining the ECG immediately prior to catheterization could improve predictive performance. In addition, our ECG dataset was not restricted to patients with normal sinus rhythm – as we hope to develop a method that would work in the setting of different cardiac arrythmias. However, assessing the performance in cohorts with significant arrhythmias are challenging as our set of ECGs with significant arrhythmias is small; e.g., less than 10% of our ECGs

Table E.2: Dataset summary statistics.

Dataset	Institution	# RHCs	# patients	Age	% Female	% with mPCWP >18mmHg
Development	Hospital 1	5680	3014	63± 15	34%	37.7%
Internal Test	Hospital 1	1441	753	63 ±15	38%	35.5%
External Validation	Hospital 2	2725	1249	56 ± 15	32%	34.3%

have atrial fibrillation. Additional work is needed to fully assess the performance of our model on patients with significant arrhythmias. Furthermore, our estimates for the PPV and NPV rely on estimates of the prevalence of elevated mPCWP that were derived from prior studies of patients with HF and reduced LVEF. Since we have limited data on the demographics of this population (i.e., only age, sex, filling pressures, LVEF), it is difficult to make definitive statements about how these results generalize to other populations that may have very different characteristics. Further prospective studies are needed to verify that these estimates are applicable to the wide swath of patients that are seen in modern day practice; for example, to evaluate how the model performs on patients who had an ECG but who would not have undergone catheterization as part of their care.

Overall, our study suggests that there is a role for deep learning models in the estimation of central cardiac pressures in patients with heart failure. The method has the potential to provide clinically useful information when invasive assessment of central hemodynamics is either not possible or feasible.

E.4 Methods

E.4.1 Datasets

We develop and validate our model using datasets from two different hospitals. Our first dataset consists of 7121 records from 3767 unique patients who underwent cardiac catheterization at Massachusetts General Hospital (Hospital 1). All patients had a diagnosis of HF (according to ICD 9/10 codes in their medical record) within the 1 year prior to their catheterization date.

This dataset is split into an 80% development set, used to train predictive models,

and a 20% internal holdout test set, used for model evaluation. Datasets are constructed such that no data from a single patient appears in different data splits; i.e., all data splits are done on a per-patient basis. We further split the development set on a per-patient level using an 80-20 split into training and "dev" sets. The training set is used to train the model and the dev set is used to determine when training is completed.

Our second dataset consists of 2725 records from 1249 unique patients who underwent cardiac catheterization at the Brigham and Women's Hospital (Hospital 2). As with data from MGH, these patients all had a diagnosis of heart failure (according to ICD 9/10 codes in their medical record) within the 1 year prior to their catheterization date. We used this entire dataset as an external validation set for model evaluation.

Each record in the datasets consists of: the mean Pulmonary Capillary Wedge Pressure (as measured by cardiac catheterization), a 10-second, 12-lead ECG recorded by the same system (GE Healthcare MUSE) on the same day as the catheterization procedure, and basic demographic information (age/sex). Dataset details are summarized in Table E.2.

E.4.2 Data pre-processing

We resampled all 12-lead ECGs at 250 Hz (the raw ECG signals were recorded at either 250 Hz or 500 Hz) and removed any ECGs containing non-physical voltage values ($>$ 5mV in magnitude) in any lead. We additionally removed ECGs that had any leads that were zero-valued for more than 5 seconds. We normalized the continuous age feature using Z-scoring based on the mean and standard deviation of age in the training set. We binarized the sex feature and then subtracted 0.5 to center it on zero; i.e., 0.5 denotes male and -0.5 denotes female. The continuous mPCWP measurements were binarized using the threshold of 18 mmHg.

E.4.3 Baseline Model

We constructed a Logistic Ridge Regression model to predict elevated mPCWP from age, sex, and intervals extracted from the ECG (heart rate and PR, QRS, and QT intervals) to predict whether the mPCWP is above 18 mmHg. The regularization strength was chosen to be the value that gave the best AUROC on the dev set. The intervals were normalized using z-scoring based on training set statistics, and the age and sex features were normalized as described above.

E.4.4 Model development

We developed a deep learning model, HFNet, which estimates whether a patient's mPCWP is over 18mmHg, using the 10-second 12-lead ECG, age, and sex as input features. We included age and sex in the model since we hypothesized that that these features encode some prior information about filling pressures; e.g., older age and male sex are associated with a higher prevalence of cardiac disease. The model combines a convolutional encoder for the ECG and a fully connected network encoder for the demographic features. The model's weights and biases are first randomly initialized. The model is then trained to minimize the binary cross-entropy loss between its output, corresponding to mPCWP $>$ 18mmHg, and the binary label of whether the pressure measured during catheterization was elevated or not. Training proceeds for at most 50 epochs using the Adam optimizer with a learning rate of 1e-3, however, we use early stopping based on the dev set AUROC score to prevent model overfitting. Full architectural details of the model are in given in Section E.5.

E.4.5 Statistics

We obtain measures of uncertainty in performance using empirical bootstrapping; we sample data points at random, with replacement, obtaining 10 bootstrapped sets that have the same size as the original set. Performance metrics are computed on each of the bootstraps, and then the mean/standard deviation of performance across these bootstraps is reported. Paired t-tests were used to calculate p-values to assess

statistical significance.

Positive and Negative Predictive Values as a function of sensitivity, specificity, and prevalence were computed using the following relationships:

$$\text{PPV} = \frac{\text{sensitivity} \times \text{prevalence}}{\text{sensitivity} \times \text{prevalence} + (1 - \text{specificity}) * (1 - \text{prevalence})}$$

$$\text{NPV} = \frac{\text{specificity} \times (1 - \text{prevalence})}{\text{specificity} \times (1 - \text{prevalence}) + (1 - \text{sensitivity}) * \text{prevalence}}$$

E.4.6 Identifying Untrustworthy Predictions

Assessing whether a given prediction made by the model is trustworthy or not is important for practical use, since it helps clinicians decide in which cases to trust a given model prediction. We hypothesized that we could use the predictive entropy of the model's output as a way of determining whether the model's output is trustworthy or not on a per-ECG basis. The predictive entropy is defined as:

$$\text{Entropy}(x) = -\left(\hat{y} \log \hat{y} + (1 - \hat{y}) \log(\hat{y})\right),$$

where x denotes the ECG and age, sex (inputs to the model); and $\hat{y}$ is the model output (the prediction). We hypothesize that large entropy values (which correspond to the predicted probability being near 0.5) arise in situations where the model is least reliable, and the lowest entropy values (which correspond probabilities near 0 or 1) arise for inputs where the model's prediction is most reliable.

E.5 Supplementary Information

E.5.1 Model Architecture

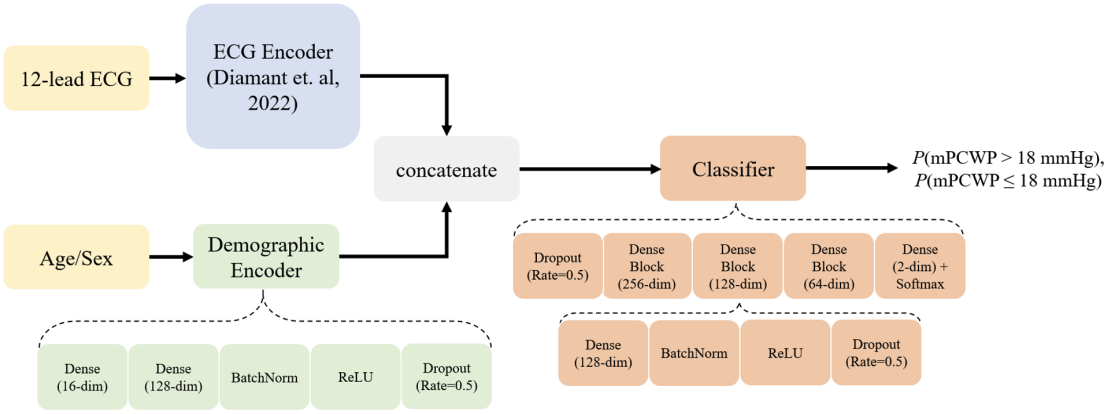


Figure E-7: **Overall HFNet Model Architecture.**

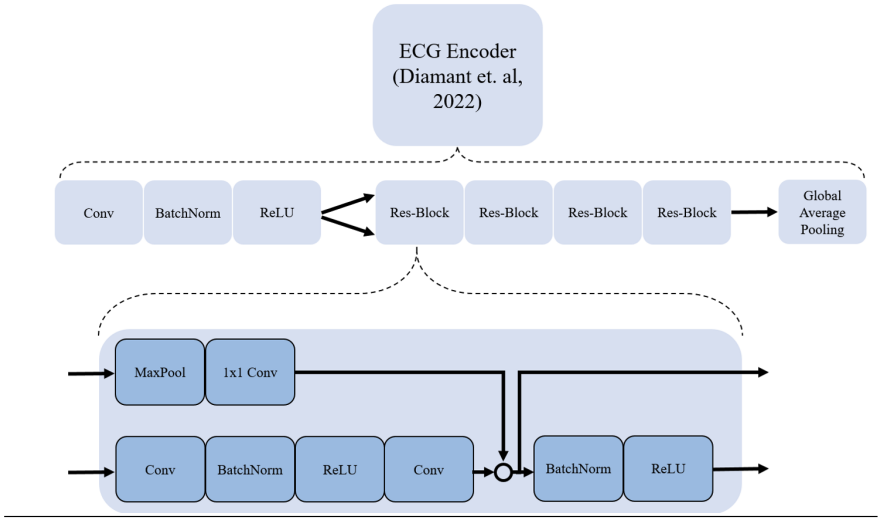


Figure E-8: **ECG Encoder Architecture.**

Our model has three components: an ECG encoder, a demographic features encoder, and a classifier, shown in Figure E-7.

ECG encoder. Our ECG encoder is a convolutional neural network (CNN). We adopt the encoder architecture from [47] for the encoder, Figure E-8. This model consists of a residual architecture followed by a Global Average Pooling layer to produce

a 320-dimensional representation of the 12-lead ECG signal.

Demographic features encoder. The demographic features encoder is a fully connected neural network. The two-dimensional vector with normalized age and sex is first projected into a 16-dimensional space with a Dense layer. This is then passed through a fully connected block with a Dense layer mapping into 128-dimensions, followed by Batch Normalization, ReLU activation and Dropout (rate 0.5).

Classifier. The classifier first concatenates the output from the ECG encoder and demographic features encoder, yielding a 448-dimensional vector, and then applies Dropout (rate 0.5). Then, 3 blocks of Dense, BatchNorm, ReLU, Dropout (rate 0.5) are applied, with Dense layer output dimensionalities being 256, 128, and 64 dimensions. This is followed by a final Dense layer with 2 dimensional output, which corresponds to the probability of mPCWP being above 18 mmHg and below 18 mmHg respectively.

E.5.2 Further Patient Characteristic and Prevalence Details

Table E.3: Prevalence of mPCWP>18mmHg as a function of different CXR findings. Taken from [26].

CXR Finding	Prevalence of mPCWP>18mmHg in patients with HFrEF	Sensitivity of positive CXR for detecting mPCWP>18mmHg
Interstitial edema	0.83	27%
No interstitial	0.68	-
Pulmonary Vascular Redistribution	0.89	65%
No Pulmonary Vascular Distribution	0.52	-

Table E.4: Characteristics of patients with reduced ejection fraction (EF).

Dataset	# RHCs	# patients	Age	% Female	% with mPCWP >18mmHg	Mean and standard deviation of EF
Development	1535	1053	63± 15	25%	59.2%	24 ± 8
Internal Test	358	245	64 ±14	30%	62.2%	24 ± 8

E.5.3 Further multiple catheterization results

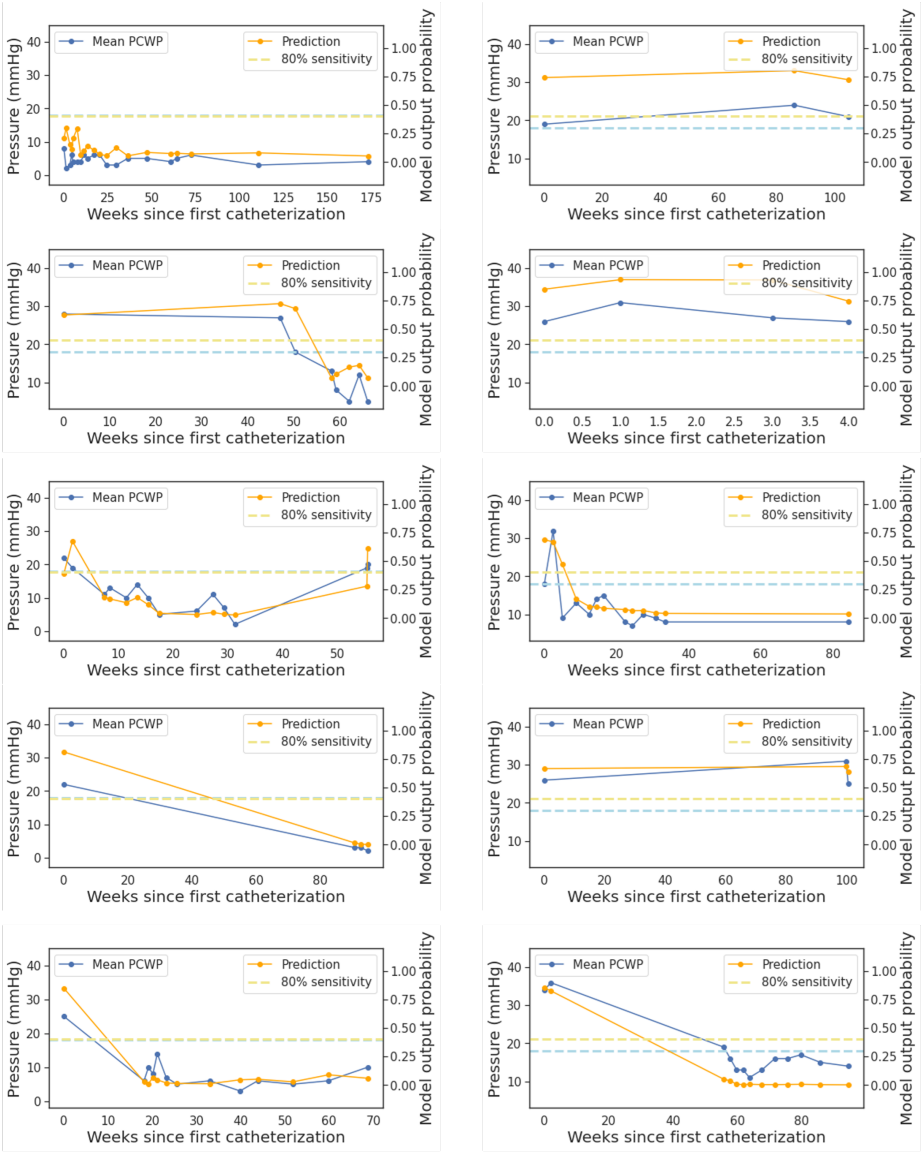


Figure E-9: Multiple catheterization: further examples from the internal test set. The blue dashed line corresponds to 18mmHg mPCWP (left axis), and the dashed green line indicates the model output threshold corresponding to an 80% sensitivity level.

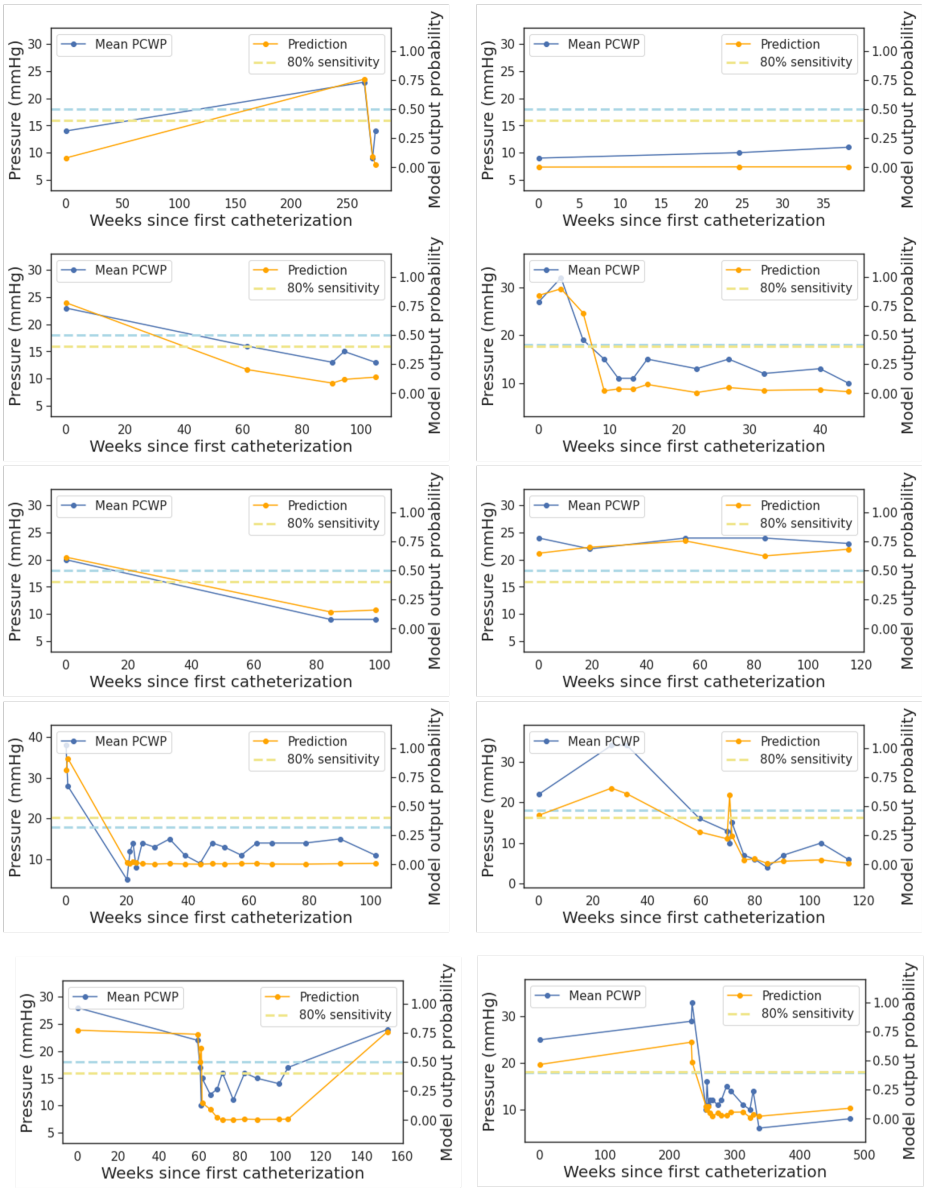


Figure E-10: Multiple catheterization: further examples from the external validation set. The blue dashed line corresponds to 18mmHg mPCWP (left axis), and the dashed green line indicates the model output threshold corresponding to an 80% sensitivity level.

Bibliography

[1] Margot Aalders and Wouter Kok. Comparison of hemodynamic factors predicting prognosis in heart failure: a systematic review. *Journal of Clinical Medicine*, 8(10):1757, 2019.

[2] Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katherine Millican, Malcolm Reynolds, et al. Flamingo: a visual language model for few-shot learning. *Advances in Neural Information Processing Systems*, 35:23716–23736, 2022.

[3] Antreas Antoniou, Harrison Edwards, and Amos Storkey. How to train your MAML. *arXiv preprint arXiv:1810.09502*, 2018.

[4] Shekoofeh Azizi, Basil Mustafa, Fiona Ryan, Zachary Beaver, Jan Freyberg, Jonathan Deaton, Aaron Loh, Alan Karthikesalingam, Simon Kornblith, Ting Chen, et al. Big self-supervised models advance medical image classification. *arXiv preprint arXiv:2101.05224*, 2021.

[5] Philip Bachman, R Devon Hjelm, and William Buchwalter. Learning representations by maximizing mutual information across views. *arXiv preprint arXiv:1906.00910*, 2019.

[6] Alexei Baevski, Yuhao Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A framework for self-supervised learning of speech representations. *Advances in neural information processing systems*, 33:12449–12460, 2020.

[7] Dara Bahri, Heinrich Jiang, Yi Tay, and Donald Metzler. Scarf: Self-supervised contrastive learning using random feature corruption. In *International Conference on Learning Representations*, 2021.

[8] J Bajorat, R Hofmockel, DA Vagts, M Janda, B Pohl, C Beck, and G Noeldge-Schomburg. Comparison of invasive and less-invasive techniques of cardiac output measurement under different haemodynamic conditions in a pig model. *European journal of anaesthesiology*, 23(1):23–30, 2006.

[9] Guha Balakrishnan, Amy Zhao, Mert Sabuncu, John Guttag, and Adrian V. Dalca. An unsupervised learning model for deformable medical image registration. *CVPR: Computer Vision and Pattern Recognition*, pages 9252–9260, 2018.

[10] Guha Balakrishnan, Amy Zhao, Mert Sabuncu, John Guttag, and Adrian V. Dalca. Voxelmorph: A learning framework for deformable medical image registration. *IEEE TMI: Transactions on Medical Imaging*, 38:1788–1800, 2019.

[11] Kasun Bandara, Hansika Hewamalage, Yuan-Hao Liu, Yanfei Kang, and Christoph Bergmeir. Improving the accuracy of global forecasting models using time series data augmentation. *Pattern Recognition*, 120:108148, 2021.

[12] Rohan Banerjee and Avik Ghose. Synthesis of realistic ecg waveforms using a composite generative adversarial network for classification of atrial fibrillation. In *2021 29th European Signal Processing Conference (EUSIPCO)*, pages 1145–1149, 2021.

[13] Yujia Bao, Menghua Wu, Shiyu Chang, and Regina Barzilay. Few-shot text classification with distributional signatures. *arXiv preprint arXiv:1908.06039*, 2019.

[14] Adrien Bardes, Jean Ponce, and Yann LeCun. Vicreg: Variance-invariance-covariance regularization for self-supervised learning. In *ICLR*, 2022.

[15] Adrien Bardes, Jean Ponce, and Yann LeCun. VICRegL: Self-Supervised Learning of Local Visual Features. In *NeurIPS*, 2022.

[16] Anthony Bau, Yonatan Belinkov, Hassan Sajjad, Nadir Durrani, Fahim Dalvi, and James Glass. Identifying and controlling important neurons in neural machine translation. *arXiv preprint arXiv:1811.01157*, 2018.

[17] Guillaume Baudry, Juliette Bourdin, Raluca Mocan, Elisabeth Hugon-Vallet, Matteo Pozzi, Antoine Jobbé-Duval, Nicolas Paulo, Patrick Rossignol, Laurent Sebbag, and Nicolas Girerd. Prognosis of advanced heart failure patients according to their hemodynamic profile based on the modified forrester classification. *Journal of Clinical Medicine*, 11(13):3663, 2022.

[18] Guillaume Baudry, Guillaume Coutance, Richard Dorent, Fabrice Bauer, Katrien Blanchart, Aude Boignard, Céline Chabanne, Clément Delmas, Nicolas d'Ostrevy, Eric Epailly, et al. Prognosis value of forrester's classification in advanced heart failure patients awaiting heart transplantation. *ESC Heart Failure*, 9(5):3287–3297, 2022.

[19] Alex Beatson and Ryan P Adams. Efficient optimization of loops and limits with randomized telescoping sums. In *International Conference on Machine Learning*, pages 534–543. PMLR, 2019.

[20] Yoshua Bengio. Gradient-based optimization of hyperparameters. *Neural Computation*, 12(8):1889–1900, 2000.

[21] Emelia J Benjamin, Paul Muntner, Alvaro Alonso, Marcio S Bittencourt, Clifton W Callaway, April P Carson, Alanna M Chamberlain, Alexander R

Chang, Susan Cheng, Sandeep R Das, et al. Heart disease and stroke statistics—2019 update: a report from the american heart association. *Circulation*, 139(10):e56–e528, 2019.

[22] Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, USA:, 1994.

[23] Luca Bertinetto, Joao F Henriques, Philip HS Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. *arXiv preprint arXiv:1805.08136*, 2018.

[24] Henry Blackburn, Ancel Keys, Ernst Simonson, Pentti Rautaharju, and Sven Punsar. The electrocardiogram in population studies: a classification system. *Circulation*, 21(6):1160–1175, 1960.

[25] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[26] Samuel M Butman, Gordon A Ewy, James R Standen, Karl B Kern, and Elizabeth Hahn. Bedside cardiovascular examination in patients with severe chronic heart failure: importance of rest or inducible jugular venous distension. *Journal of the American College of Cardiology*, 22(4):968–974, 1993.

[27] Victor Ion Butoi, Jose Javier Gonzalez Ortiz, Tianyu Ma, Mert R Sabuncu, John Guttag, and Adrian V Dalca. Universeg: Universal medical image segmentation. *arXiv preprint arXiv:2304.06131*, 2023.

[28] Ibadete Bytyçi and Gani Bajraktari. Mortality in heart failure patients. *Anatolian journal of cardiology*, 15(1):63, 2015.

[29] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[30] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. *arXiv preprint arXiv:2002.05709*, 2020.

[31] Ting Chen, Simon Kornblith, Kevin Swersky, Mohammad Norouzi, and Geoffrey Hinton. Big self-supervised models are strong semi-supervised learners. *arXiv preprint arXiv:2006.10029*, 2020.

[32] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

[33] Xinlei Chen, Haoqi Fan, Ross Girshick, and Kaiming He. Improved baselines with momentum contrastive learning. *arXiv preprint arXiv:2003.04297*, 2020.

[34] Xinlei Chen and Kaiming He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[35] Joseph Y Cheng, Hanlin Goh, Kaan Dogrusoz, Oncel Tuzel, and Erdrin Azemi. Subject-aware contrastive learning for biosignals. *arXiv preprint arXiv:2007.04871*, 2020.

[36] Tsz-Him Cheung and Dit-Yan Yeung. Modals: Modality-agnostic automated data augmentation in the latent space. In *International Conference on Learning Representations*, 2021.

[37] Ovidiu Chioncel, Alexandre Mebazaa, Aldo P Maggioni, Veli-Pekka Harjola, Giuseppe Rosano, Cecile Laroche, Massimo F Piepoli, Maria G Crespo-Leiro, Mitja Lainscak, Piotr Ponikowski, et al. Acute heart failure congestion and perfusion status–impact of the clinical classification on in-hospital and long-term outcomes; insights from the esc-eorp-hfa heart failure long-term registry. *European journal of heart failure*, 21(11):1338–1352, 2019.

[38] Ross M Clarke, Elre Talea Oldewage, and José Miguel Hernández-Lobato. Scalable one-pass optimisation of high-dimensional weight-update hyperparameters by implicit differentiation. In *International Conference on Learning Representations*.

[39] Liam Collins, Aryan Mokhtari, Sewoong Oh, and Sanjay Shakkottai. Maml and anil provably learn representations. In *International Conference on Machine Learning*, pages 4238–4310. PMLR, 2022.

[40] Lauren B Cooper, Robert J Mentz, Susanna R Stevens, G Michael Felker, Carlo Lombardi, Marco Metra, Lynne W Stevenson, Christopher M O'Connor, Carmelo A Milano, Chetan B Patel, et al. Hemodynamic predictors of heart failure morbidity and mortality: fluid or flow? *Journal of cardiac failure*, 22(3):182–189, 2016.

[41] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Algorithms for learning kernels based on centered alignment. *The Journal of Machine Learning Research*, 13(1):795–828, 2012.

[42] Ricardo Couceiro, Paulo Carvalho, Jorge Henriques, Manuel Antunes, Matthew Harris, and Jörg Habetha. Detection of atrial fibrillation using model-based ecg analysis. In *2008 19th International Conference on Pattern Recognition*, pages 1–5. IEEE, 2008.

[43] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation strategies from data. In *Proceedings*

of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 113–123, 2019.

[44] Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. Randaugment: Practical automated data augmentation with a reduced search space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pages 702–703, 2020.

[45] James A de Lemos, Darren K McGuire, and Mark H Drazner. B-type natriuretic peptide in cardiovascular disease. *The Lancet*, 362(9380):316–322, 2003.

[46] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding, 2018.

[47] Nathaniel Diamant, Erik Reinertsen, Steven Song, Aaron Aguirre, Collin Stultz, and Puneet Batra. Patient contrastive learning: a performant, expressive, and practical approach to ecg modeling. 2021.

[48] K Dickstein. Esc committee for practice guidelines (cpg): Esc guidelines for the diagnosis and treatment of acute and chronic heart failure 2008: The task force for the diagnosis and treatment of acute and chronic heart failure 2008 of the european society of cardiology: Developed in collaboration with the heart failure association of the esc (hfa) and endorsed by the european society of intensive care medicine (esicm). *Eur Heart J*, 29:2388–2442, 2008.

[49] Justin Domke. Generic methods for optimization-based modeling. In *Proceedings of the Fifteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2012*, JMLR Proceedings. JMLR.org, 2012.

[50] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International conference on machine learning*, pages 647–655. PMLR, 2014.

[51] Mark H Drazner, Anne S Hellkamp, Carl V Leier, Monica R Shah, Leslie W Miller, Stuart D Russell, James B Young, Robert M Califf, and Anju Nohria. Value of clinician assessment of hemodynamics in advanced heart failure: the escape trial. *Circulation: Heart Failure*, 1(3):170–177, 2008.

[52] Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. *arXiv preprint arXiv:1805.03643*, 2018.

[53] Yang Fan, Yingce Xia, Lijun Wu, Shufang Xie, Weiqing Liu, Jiang Bian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach with deep interactions. *arXiv preprint arXiv:2007.04649*, 2020.

[54] Francis M Fesmire, Robert F Percy, Jim B Bardoner, David R Wharton, and Frank B Calhoun. Usefulness of automated serial 12-lead ecg monitoring during the initial emergency department evaluation of patients with chest pain. *Annals of emergency medicine*, 31(1):3–11, 1998.

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

[56] Chelsea Finn and Sergey Levine. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. *arXiv preprint arXiv:1710.11622*, 2017.

[57] Chelsea Finn, Kelvin Xu, and Sergey Levine. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*, pages 9516–9527, 2018.

[58] James S Forrester, George A Diamond, and HJC Swan. Correlative classification of clinical and hemodynamic function after acute myocardial infarction. *The American journal of cardiology*, 39(2):137–145, 1977.

[59] Gauthier Gidel, Reyhane Askari Hemmat, Mohammad Pezeshki, Rémi Le Priol, Gabriel Huang, Simon Lacoste-Julien, and Ioannis Mitliagkas. Negative momentum for improved game dynamics. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1802–1811. PMLR, 2019.

[60] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[61] A. Goldberger, L. A. Amaral, L. Glass, Jeffrey M. Hausdorff, P. Ivanov, R. Mark, J. Mietus, G. Moody, C. Peng, and H. Stanley. PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation*, 101 23:E215–20, 2000.

[62] Yuan Gong, Andrew Rouditchenko, Alexander H. Liu, David Harwath, Leonid Karlinsky, Hilde Kuehne, and James R. Glass. Contrastive audio-visual masked autoencoder. In *The Eleventh International Conference on Learning Representations*, 2023.

[63] Bryan Gopal, Ryan W. Han, Gautham Raghupathi, Andrew Y. Ng, Geoffrey H. Tison, and Pranav Rajpurkar. 3kg: Contrastive learning of 12-lead electrocardiograms using physiologically-inspired augmentations. 2021.

[64] Jonathan Gordon, John Bronskill, Matthias Bauer, Sebastian Nowozin, and Richard E Turner. Meta-Learning probabilistic inference for prediction. *arXiv preprint arXiv:1805.09921*, 2018.

[65] Akhilesh Gotmare, Nitish Shirish Keskar, Caiming Xiong, and Richard Socher. A closer look at deep learning heuristics: Learning rate restarts, warmup and distillation. *arXiv preprint arXiv:1810.13243*, 2018.

[66] Erin Grant, Chelsea Finn, Sergey Levine, Trevor Darrell, and Thomas Griffiths. Recasting gradient-based meta-learning as hierarchical bayes. *arXiv preprint arXiv:1801.08930*, 2018.

[67] Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Guo, Mohammad Gheshlaghi Azar, et al. Bootstrap your own latent-a new approach to self-supervised learning. *Advances in neural information processing systems*, 33:21271–21284, 2020.

[68] Albert Gu, Karan Goel, and Christopher Re. Efficiently modeling long sequences with structured state spaces. In *International Conference on Learning Representations*.

[69] Stefano Guzzetti, Maria Teresa La Rovere, Gian Domenico Pinna, Roberto Maestri, Ester Borroni, Alberto Porta, Andrea Mortara, and Alberto Malliani. Different spectral components of 24 h heart rate variability are related to different modes of death in chronic heart failure. *European heart journal*, 26(4):357–362, 2005.

[70] Awni Y Hannun, Pranav Rajpurkar, Masoumeh Haghpanahi, Geoffrey H Tison, Codie Bourn, Mintu P Turakhia, and Andrew Y Ng. Cardiologist-level arrhythmia detection and classification in ambulatory electrocardiograms using a deep neural network. *Nature medicine*, 25(1):65–69, 2019.

[71] James Harrison, Apoorva Sharma, and Marco Pavone. Meta-learning priors for efficient online bayesian regression. *arXiv preprint arXiv:1807.08912*, 2018.

[72] Hrayr Harutyunyan, Hrant Khachatrian, David C Kale, Greg Ver Steeg, and Aram Galstyan. Multitask learning and benchmarking with clinical time series data. *Scientific data*, 6(1):1–18, 2019.

[73] Faezeh Nejati Hatamian, Nishant Ravikumar, Sulaiman Vesal, Felix P Kemeth, Matthias Struck, and Andreas Maier. The effect of data augmentation on classification of atrial fibrillation in short single-lead ecg signals using deep neural networks. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1264–1268. IEEE, 2020.

[74] Ryuichiro Hataya, Jan Zdenek, Kazuki Yoshizoe, and Hideki Nakayama. Meta approach to data augmentation optimization. *arXiv preprint arXiv:2006.07965*, 2020.

[75] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of*

the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 16000–16009, 2022.

[76] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[77] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[78] Olivier Henaff. Data-efficient image recognition with contrastive predictive coding. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 4182–4192. PMLR, 13–18 Jul 2020.

[79] Bart Hiemstra, Geert Koster, Renske Wiersema, Yoran M Hummel, Pim van der Harst, Harold Snieder, Ruben J Eck, Thomas Kaufmann, Thomas WL Scheeren, Anders Perner, et al. The diagnostic accuracy of clinical examination for estimating cardiac index in critically ill patients: the simple intensive care studies-i. *Intensive care medicine*, 45(2):190–200, 2019.

[80] Yukina Hirata, Kenya Kusunose, Takumasa Tsuji, Kohei Fujimori, Jun'ichi Kotoku, and Masataka Sata. Deep learning for detection of elevated pulmonary artery wedge pressure using standard chest x-ray. *Canadian Journal of Cardiology*, 37(8):1198–1206, 2021.

[81] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. In *International Conference on Learning Representations*, 2019.

[82] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. *arXiv preprint arXiv:1801.06146*, 2018.

[83] Kyle Hsu, Sergey Levine, and Chelsea Finn. Unsupervised learning via meta-learning. *arXiv preprint arXiv:1810.02334*, 2018.

[84] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *International Conference on Learning Representations*, 2020.

[85] Zhiting Hu, Bowen Tan, Russ R Salakhutdinov, Tom M Mitchell, and Eric P Xing. Learning data manipulation for augmentation and weighting. In *Advances in Neural Information Processing Systems*, volume 32, 2019.

[86] J Hurst, C Rackley, E Sonnenblick, and N Wenger. *The heart, arteries and veins*, volume 1. McGraw-Hill, 1990.

[87] Brian Kenji Iwana and Seiichi Uchida. An empirical survey of data augmentation for time series classification with neural networks. *arXiv preprint arXiv:2007.15951*, 2020.

[88] Brian Kenji Iwana and Seiichi Uchida. An empirical survey of data augmentation for time series classification with neural networks. *Plos one*, 16(7):e0254841, 2021.

[89] Brian Kenji Iwana and Seiichi Uchida. Time series data augmentation for neural networks by time warping with a discriminative teacher. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 3558–3565. IEEE, 2021.

[90] Eric Jang, Shixiang Gu, and Ben Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[91] Khurram Javed and Martha White. Meta-learning representations for continual learning. *arXiv preprint arXiv:1905.12588*, 2019.

[92] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *International Conference on Machine Learning*, pages 2304–2313, 2018.

[93] Alistair EW Johnson, Tom J Pollard, Lu Shen, Li-wei H Lehman, Mengling Feng, Mohammad Ghassemi, Benjamin Moody, Peter Szolovits, Leo Anthony Celi, and Roger G Mark. Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9, 2016.

[94] Kirthevasan Kandasamy, Karun Raju Vysyaraju, Willie Neiswanger, Biswajit Paria, Christopher R Collins, Jeff Schneider, Barnabas Poczos, and Eric P Xing. Tuning hyperparameters without grad students: Scalable and robust Bayesian optimisation with Dragonfly. *arXiv preprint arXiv:1903.06694*, 2019.

[95] Minki Kang, Moonsu Han, and Sung Ju Hwang. Neural mask generator: Learning to generate adaptive word maskings for language model adaptation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6102–6120, 2020.

[96] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *International Conference on Learning Representations*, 2014.

[97] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023.

[98] Dani Kiyasseh, Tingting Zhu, and David A Clifton. Clocs: Contrastive learning of cardiac signals across space, time, and patients. In *International Conference on Machine Learning*, pages 5606–5615. PMLR, 2021.

[99] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[100] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. *arXiv preprint arXiv:1912.11370*, 6(2):8, 2019.

[101] Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

[102] Simon Kornblith, Jonathon Shlens, and Quoc V Le. Do better ImageNet models transfer better? In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2661–2671, 2019.

[103] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[104] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 10657–10665, 2019.

[105] Yoonho Lee and Seungjin Choi. Gradient-based meta-learning with learned layerwise metric and subspace. *arXiv preprint arXiv:1801.05558*, 2018.

[106] Roman Levin, Valeriia Cherepanova, Avi Schwarzschild, Arpit Bansal, C Bayan Bruss, Tom Goldstein, Andrew Gordon Wilson, and Micah Goldblum. Transfer learning with deep tabular models. *arXiv preprint arXiv:2206.15306*, 2022.

[107] Yikuan Li, Mohammad Mamouei, Gholamreza Salimi-Khorshidi, Shishir Rao, Abdelaali Hassaine, Dexter Canoy, Thomas Lukasiewicz, and Kazem Rahimi. Hi-behrt: Hierarchical transformer-based model for accurate prediction of clinical events using multimodal longitudinal electronic health records. *arXiv preprint arXiv:2106.11360*, 2021.

[108] Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *FE@ NIPS*, pages 196–212, 2015.

[109] Gen-Min Lin and Henry Horng-Shing Lu. A 12-lead ecg-based system with physiological parameters and machine learning to identify right ventricular hypertrophy in young adults. *IEEE Journal of Translational Engineering in Health and Medicine*, 8:1–10, 2020.

[110] Shikun Liu, Andrew Davison, and Edward Johns. Self-supervised generalisation with meta auxiliary learning. In *Advances in Neural Information Processing Systems*, pages 1679–1689, 2019.

[111] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[112] Jonathan Lorraine and David Duvenaud. Stochastic hyperparameter optimization through hypernetworks. *arXiv preprint arXiv:1802.09419*, 2018.

[113] Jonathan Lorraine, Paul Vicol, and David Duvenaud. Optimizing millions of hyperparameters by implicit differentiation. In *International Conference on Artificial Intelligence and Statistics*, pages 1540–1552. PMLR, 2020.

[114] Matthew MacKay, Paul Vicol, Jon Lorraine, David Duvenaud, and Roger Grosse. Self-tuning networks: Bilevel optimization of hyperparameters using structured best-response functions. In *International Conference on Learning Representations*, 2019.

[115] Dougal Maclaurin, David Duvenaud, and Ryan Adams. Gradient-based hyperparameter optimization through reversible learning. In *International Conference on Machine Learning*, pages 2113–2122, 2015.

[116] Chris J Maddison, Andriy Mnih, and Yee Whye Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[117] Niru Maheswaranathan, Alex H. Willams, Matthew D. Golub, Surya Ganguli, and David Sussillo. Universality and individuality in neural dynamics across large populations of recurrent networks. *arXiv preprint arXiv:1907.08549*, 2019.

[118] Alan Maisel. B-type natriuretic peptide levels: diagnostic and prognostic in congestive heart failure: what's next?, 2002.

[119] Matthew McDermott, Bret Nestor, Evan Kim, Wancong Zhang, Anna Goldenberg, Peter Szolovits, and Marzyeh Ghassemi. A comprehensive evaluation of multi-task learning and multi-task pre-training on ehr time-series data. *arXiv preprint arXiv:2007.10185*, 2020.

[120] Matthew McDermott, Bret Nestor, Evan Kim, Wancong Zhang, Anna Goldenberg, Peter Szolovits, and Marzyeh Ghassemi. A comprehensive ehr timeseries pre-training benchmark. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 257–278, 2021.

[121] Temesgen Mehari and Nils Strodthoff. Self-supervised representation learning from 12-lead ecg data. *arXiv preprint 2103.12676*, 2021.

[122] Jonas Močkus. On Bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404, 1975.

[123] Benjamin Moody, George Moody, Mauricio Villarroel, Gari Clifford, and Ikaro Silva. MIMIC-III waveform database, 2020.

[124] Ari S Morcos, Maithra Raghu, and Samy Bengio. Insights on representational similarity in neural networks with canonical correlation. *arXiv preprint arXiv:1806.05759*, 2018.

[125] Meinard Müller. Dynamic time warping. *Information retrieval for music and motion*, pages 69–84, 2007.

[126] Basil Mustafa, Aaron Loh, Jan Freyberg, Patricia MacWilliams, Alan Karthikesalingam, Neil Houlsby, and Vivek Natarajan. Supervised transfer learning at scale for medical imaging. *arXiv preprint arXiv:2101.05913*, 2021.

[127] SF Nagueh, OA Smiseth, CP Appleton, B Byrd, H Dokainish, T Edvardsen, et al. Ase/eacvi guidelines and standards. recommendations for the evaluation of left ventricular diastolic function by echocardiography: An update from the american society of echocardiography and the european association of cardiovascular imaging. *J Am Soc Echocardiogr*, 29(4):277–314, 2016.

[128] Sherif F Nagueh, Katherine J Middleton, Helen A Kopelen, William A Zoghbi, and Miguel A Quiñones. Doppler tissue imaging: a noninvasive technique for evaluation of left ventricular relaxation and estimation of filling pressures. *Journal of the American College of Cardiology*, 30(6):1527–1533, 1997.

[129] Aviv Navon, Idan Achituve, Haggai Maron, Gal Chechik, and Ethan Fetaya. Auxiliary learning by implicit differentiation. In *International Conference on Learning Representations*, 2021.

[130] Alex Nichol and John Schulman. Reptile: A scalable metalearning algorithm. *arXiv preprint arXiv:1803.02999*, 2, 2018.

[131] Anju Nohria, Sui W Tsang, James C Fang, Eldrin F Lewis, John A Jarcho, Gilbert H Mudge, and Lynne W Stevenson. Clinical assessment identifies hemodynamic profiles that predict outcomes in patients admitted with heart failure. *Journal of the American College of Cardiology*, 41(10):1797–1804, 2003.

[132] Jaehoon Oh, Hyungjun Yoo, ChangHwan Kim, and Se-Young Yun. BOIL: Towards Representation Change for Few-shot Learning. In *International Conference on Learning Representations*, 2021.

[133] Jungwoo Oh, Hyunseung Chung, Joon-myoung Kwon, Dong-gyun Hong, and Edward Choi. Lead-agnostic self-supervised learning for local and global representations of electrocardiogram. In *Conference on Health, Inference, and Learning*, pages 338–353. PMLR, 2022.

[134] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[135] Daniel S Park, William Chan, Yu Zhang, Chung-Cheng Chiu, Barret Zoph, Ekin D Cubuk, and Quoc V Le. Specaugment: A simple data augmentation method for automatic speech recognition. *arXiv preprint arXiv:1904.08779*, 2019.

[136] Daniel S Park, Yu Zhang, Chung-Cheng Chiu, Youzheng Chen, Bo Li, William Chan, Quoc V Le, and Yonghui Wu. Specaugment on large scale datasets. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6879–6883. IEEE, 2020.

[137] Hieu Pham, Zihang Dai, Qizhe Xie, Minh-Thang Luong, and Quoc V Le. Meta pseudo labels. *arXiv preprint arXiv:2003.10580*, 2020.

[138] Jielin Qiu, Jiacheng Zhu, Mengdi Xu, Peide Huang, Michael Rosenberg, Douglas Weber, Emerson Liu, and Ding Zhao. Cardiac disease diagnosis on imbalanced electrocardiography data through optimal transport augmentation. In *ICASSP 2023-2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 1–5. IEEE, 2023.

[139] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021.

[140] Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine McLeavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. *arXiv preprint arXiv:2212.04356*, 2022.

[141] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *OpenAI blog*, 2018.

[142] Aniruddh Raghu, John Guttag, Katherine Young, Eugene Pomerantsev, Adrian V Dalca, and Collin M Stultz. Learning to predict with supporting evidence: applications to clinical risk prediction. In *Proceedings of the Conference on Health, Inference, and Learning*, pages 95–104, 2021.

[143] Aniruddh Raghu, Jonathan Lorraine, Simon Kornblith, Matthew McDermott, and David K Duvenaud. Meta-learning to improve pre-training. *Advances in Neural Information Processing Systems*, 34, 2021.

[144] Aniruddh Raghu, Maithra Raghu, Samy Bengio, and Oriol Vinyals. Rapid learning or feature reuse? towards understanding the effectiveness of maml. In *International Conference on Learning Representations*, 2019.

[145] Aniruddh Raghu, Maithra Raghu, Simon Kornblith, David Duvenaud, and Geoffrey Hinton. Teaching with commentaries. *arXiv preprint arXiv:2011.03037*, 2020.

[146] Aniruddh Raghu, Maithra Raghu, Simon Kornblith, David Duvenaud, and Geoffrey Hinton. Teaching with commentaries. In *International Conference on Learning Representations*, 2021.

[147] Aniruddh Raghu, Daphne Schlesinger, Eugene Pomerantsev, Srikanth Devireddy, Pinak Shah, Joseph Garasic, John Guttag, and Collin M Stultz. Ecg-guided non-invasive estimation of pulmonary congestion in patients with heart failure. *Scientific Reports*, 13(1):3923, 2023.

[148] Aniruddh Raghu, Divya Shanmugam, Eugene Pomerantsev, John Guttag, and Collin M Stultz. Data augmentation for electrocardiograms. In *Conference on Health, Inference, and Learning*, pages 282–310. PMLR, 2022.

[149] Maithra Raghu. SVCCA Code and Tutorials. `https://github.com/google/svcca`.

[150] Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability. In *Advances in Neural Information Processing Systems*, pages 6076–6085, 2017.

[151] Maithra Raghu, Chiyuan Zhang, Jon Kleinberg, and Samy Bengio. Transfusion: Understanding transfer learning with applications to medical imaging. *arXiv preprint arXiv:1902.07208*, 2019.

[152] Sushravya Raghunath, Alvaro E Ulloa Cerna, Linyuan Jing, Joshua Stough, Dustin N Hartzel, Joseph B Leader, H Lester Kirchner, Martin C Stumpe, Ashraf Hafez, Arun Nemani, et al. Prediction of mortality from 12-lead electrocardiogram voltage data using a deep neural network. *Nature medicine*, 26(6):886–891, 2020.

[153] Aravind Rajeswaran, Chelsea Finn, Sham Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 2019.

[154] Roshan Rao, Nicholas Bhattacharya, Neil Thomas, Yan Duan, Xi Chen, John Canny, Pieter Abbeel, and Yun S Song. Evaluating protein transfer learning with tape. *Advances in Neural Information Processing Systems*, 32:9689, 2019.

[155] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. 2016.

[156] Mengwei Ren, Neel Dey, Martin A. Styner, Kelly Botteron, and Guido Gerig. Local spatiotemporal representation learning for longitudinally-consistent neuroimage analysis, 2022.

[157] Mengye Ren, Wenyuan Zeng, Bin Yang, and Raquel Urtasun. Learning to reweight examples for robust deep learning. In *International Conference on Machine Learning*, pages 4331–4340, 2018.

[158] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10684–10695, 2022.

[159] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

[160] Stephen M Salerno, Patrick C Alguire, and Herbert S Waxman. Competency in interpretation of 12-lead electrocardiograms: a summary and appraisal of published evidence. *Annals of Internal Medicine*, 138(9):751–760, 2003.

[161] Adam Santoro, Sergey Bartunov, Matthew Botvinick, Daan Wierstra, and Timothy Lillicrap. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*, pages 1842–1850, 2016.

[162] Naomi Saphra and Adam Lopez. Understanding learning dynamics of language models with SVCCA. *arXiv preprint arXiv:1811.00225*, 2018.

[163] Omid Sayadi, Mohammad B Shamsollahi, and Gari D Clifford. Synthetic ecg generation and bayesian filtering using a gaussian wave-based dynamical model. *Physiological measurement*, 31(10):1309, 2010.

[164] Daphne Schlesinger, Nathaniel Diamant, Aniruddh Raghu, Erik Reinertsen, Katherine Young, Puneet Batra, Eugene Pomerantsev, and Collin M. Stultz. A deep learning model for inferring elevated pulmonary capillary wedge pressures from the 12-lead electrocardiogram. 2021.

[165] Daphne E Schlesinger and Collin M Stultz. Deep learning for cardiovascular risk stratification. *Current Treatment Options in Cardiovascular Medicine*, 22:1–14, 2020.

[166] Divya Shanmugam, Davis Blalock, Guha Balakrishnan, and John Guttag. Better aggregation in test-time augmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 1214–1223, 2021.

[167] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[168] Connor Shorten and Taghi M Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6(1):1–48, 2019.

[169] Jun Shu, Qi Xie, Lixuan Yi, Qian Zhao, Sanping Zhou, Zongben Xu, and Deyu Meng. Meta-weight-net: Learning an explicit mapping for sample weighting. In *Advances in Neural Information Processing Systems*, pages 1919–1930, 2019.

[170] Jimmy TH Smith, Andrew Warrington, and Scott W Linderman. Simplified state space layers for sequence modeling. *arXiv preprint arXiv:2208.04933*, 2022.

[171] Slawek Smyl and Karthik Kuber. Data preprocessing and augmentation for multiple short time series forecasting with recurrent neural networks. In *36th International Symposium on Forecasting*, 2016.

[172] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems*, pages 4077–4087, 2017.

[173] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical Bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[174] Peter Solin, Peter Bergin, Meroula Richardson, David M Kaye, E Haydn Walters, and Matthew T Naughton. Influence of pulmonary capillary wedge pressure on central apnea in heart failure. *Circulation*, 99(12):1574–1579, 1999.

[175] Felipe Petroski Such, Aditya Rawal, Joel Lehman, Kenneth Stanley, and Jeffrey Clune. Generative teaching networks: Accelerating neural architecture search by learning to generate synthetic training data. In *International Conference on Machine Learning*, pages 9206–9216. PMLR, 2020.

[176] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[177] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[178] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need? In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XIV 16*, pages 266–282. Springer, 2020.

[179] Sindhu Tipirneni and Chandan K Reddy. Self-supervised transformer for sparse and irregularly sampled multivariate clinical time-series. *ACM Trans. Knowl. Discov. Data*, 1(1), 2022.

[180] Ekin Tiu, Ellie Talius, Pujan Patel, Curtis P. Langlotz, Andrew Y. Ng, and Pranav Rajpurkar. Expert-level detection of pathologies from unannotated chest x-ray images via self-supervised learning. *Nature Biomedical Engineering*, 6(12):1399–1406, September 2022.

[181] Sunit Tolia, Zubair Khan, Gunjan Gholkar, and Marcel Zughaib. Validating left ventricular filling pressure measurements in patients with congestive heart failure: Cardiomems™ pulmonary arterial diastolic pressure versus left atrial pressure measurement by transthoracic echocardiography. *Cardiology Research and Practice*, 2018, 2018.

[182] Sana Tonekaboni, Danny Eytan, and Anna Goldenberg. Unsupervised representation learning for time series with temporal neighborhood coding. *arXiv preprint arXiv:2106.00750*, 2021.

[183] Sana Tonekaboni, Chun-Liang Li, Sercan O. Arik, Anna Goldenberg, and Tomas Pfister. Decoupling local and global representations of time series. In Gustau Camps-Valls, Francisco J. R. Ruiz, and Isabel Valera, editors, *Proceedings of The 25th International Conference on Artificial Intelligence and Statistics*, volume 151 of *Proceedings of Machine Learning Research*, pages 8700–8714. PMLR, 28–30 Mar 2022.

[184] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-Dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[185] Maja Tschumper, Lukas Weber, Hans Rickli, Sebastian Seidl, Roman Brenner, Marc Buser, Niklas F Ehl, Franziska Jäger-Rhomberg, Peter Ammann, and Micha T Maeder. Corrected qt interval in severe aortic stenosis: Clinical and hemodynamic correlates and prognostic impact. *The American journal of medicine*, 134(2):267–277, 2021.

[186] Terry T Um, Franz MJ Pfister, Daniel Pichler, Satoshi Endo, Muriel Lang, Sandra Hirche, Urban Fietzek, and Dana Kulić. Data augmentation of wearable sensor data for parkinson's disease monitoring using convolutional neural networks. In *Proceedings of the 19th ACM International Conference on Multimodal Interaction*, pages 216–220, 2017.

[187] Evelien ES van Riet, Arno W Hoes, Kim P Wagenaar, Alexander Limburg, Marcel AJ Landman, and Frans H Rutten. Epidemiology of heart failure: the prevalence of heart failure and ventricular dysfunction in older adults over time. a systematic review. *European journal of heart failure*, 18(3):242–252, 2016.

[188] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. *Proceedings of the 25th international conference on Machine learning*, 2008.

[189] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching Networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[190] Yen Nhi Truong Vu, Richard Wang, Niranjan Balachandar, Can Liu, Andrew Y. Ng, and Pranav Rajpurkar. MedAug: Contrastive learning leveraging patient metadata improves representations for chest X-ray interpretation. 149:755–769, 06–07 Aug 2021.

[191] Patrick Wagner, Nils Strodthoff, Ralf-Dieter Bousseljot, Dieter Kreiseler, Fatima I Lunze, Wojciech Samek, and Tobias Schaeffter. PTB-XL, a large publicly available electrocardiography dataset. *Scientific data*, 7(1):1–15, 2020.

[192] Charlie S Wang, J Mark FitzGerald, Michael Schulzer, Edwin Mak, and Najib T Ayas. Does this dyspneic patient in the emergency department have congestive heart failure? *Jama*, 294(15):1944–1956, 2005.

[193] Liwei Wang, Lunjia Hu, Jiayuan Gu, Zhiqiang Hu, Yue Wu, Kun He, and John E. Hopcroft. To what extent do different neural networks learn the same representation: A Neuron Activation Subspace Match Approach. In *NeurIPS 2018*, 2018.

[194] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. In *Proceedings of the IEEE international conference on computer vision*, pages 2794–2802, 2015.

[195] Addison Weatherhead, Robert Greer, Michael-Alice Moga, Mjaye Mazwi, Danny Eytan, Anna Goldenberg, and Sana Tonekaboni. Learning unsupervised representations for icu timeseries. In *Proceedings of the Conference on Health, Inference, and Learning*, volume 174 of *Proceedings of Machine Learning Research*, pages 152–168. PMLR, 07–08 Apr 2022.

[196] Harrison G Weed. Pulmonary "capillary" wedge pressure not the pressure in the pulmonary capillaries. *Chest*, 100(4):1138–1140, 1991.

[197] Qingsong Wen, Liang Sun, Fan Yang, Xiaomin Song, Jingkun Gao, Xue Wang, and Huan Xu. Time series data augmentation for deep learning: A survey. *arXiv preprint arXiv:2002.12478*, 2020.

[198] Yuhuai Wu, Mengye Ren, Renjie Liao, and Roger Grosse. Understanding short-horizon bias in stochastic meta-optimization. *arXiv preprint arXiv:1803.02021*, 2018.

[199] Zhirong Wu, Yuanjun Xiong, Stella X Yu, and Dahua Lin. Unsupervised feature learning via non-parametric instance discrimination. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3733–3742, 2018.

[200] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *International Conference on Learning Representations*, 2019.

[201] Yuan Xue, Nan Du, Anne Mottram, Martin Seneviratne, and Andrew M Dai. Learning to select best forecast tasks for clinical outcome prediction. *Advances in Neural Information Processing Systems*, 33, 2020.

[202] Clyde W Yancy, Mariell Jessup, Biykem Bozkurt, Javed Butler, Donald E Casey, Mark H Drazner, Gregg C Fonarow, Stephen A Geraci, Tamara Horwich, James L Januzzi, et al. 2013 accf/aha guideline for the management of heart failure: executive summary: a report of the american college of cardiology foundation/american heart association task force on practice guidelines. *Journal of the American College of Cardiology*, 62(16):1495–1539, 2013.

[203] Qinyuan Ye, Belinda Z. Li, Sinong Wang, Benjamin Bolte, Hao Ma, Wen tau Yih, Xiang Ren, and Madian Khabsa. On the influence of masking policies in intermediate pre-training. *arXiv preprint arXiv:1801.06146*, 2021.

[204] Hugo Yèche, Gideon Dresdner, Francesco Locatello, Matthias Hüser, and Gunnar Rätsch. Neighborhood contrastive learning applied to online patient monitoring. In *International Conference on Machine Learning*, pages 11964–11974. PMLR, 2021.

[205] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. Gradient surgery for multi-task learning. *arXiv preprint arXiv:2001.06782*, 2020.

[206] Xiaohua Zhai, Avital Oliver, Alexander Kolesnikov, and Lucas Beyer. S4l: Self-supervised semi-supervised learning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1476–1485, 2019.

[207] Xiaohua Zhai, Joan Puigcerver, Alexander Kolesnikov, Pierre Ruyssen, Carlos Riquelme, Mario Lucic, Josip Djolonga, Andre Susano Pinto, Maxim Neumann, Alexey Dosovitskiy, et al. A large-scale study of representation learning with the visual task adaptation benchmark. *arXiv preprint arXiv:1910.04867*, 2019.

[208] Xiang Zhang, Ziyuan Zhao, Theodoros Tsiligkaridis, and Marinka Zitnik. Self-supervised contrastive pre-training for time series via time-frequency consistency. In *Proceedings of Neural Information Processing Systems, NeurIPS*, 2022.

[209] Yu Zhang, Daniel S Park, Wei Han, James Qin, Anmol Gulati, Joel Shor, Aren Jansen, Yuanzhong Xu, Yanping Huang, Shibo Wang, et al. BigSSL: Exploring the frontier of large-scale semi-supervised learning for automatic speech recognition. *IEEE Journal of Selected Topics in Signal Processing*, 16(6):1519–1532, 2022.

[210] Fengwei Zhou, Bin Wu, and Zhenguo Li. Deep meta-learning: learning to learn in the concept space. *arXiv preprint arXiv:1802.03596*, 2018.

[211] Luisa M Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. *arXiv preprint arXiv:1810.03642*, 2018.