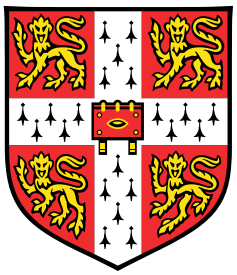
Pre-training Meta-models for Interpretability



Emilija Dordevic

Department of Engineering
University of Cambridge

This dissertation is submitted for the degree of
Master of Philosophy in Machine Learning and Machine Intelligence

King's College August 2023

Dedicated to my loving family and all who cheered me on through my Cambridge journey.

Declaration

I, Emilija Dordevic of King's College, being a candidate for the MPhil in Machine Learning and Machine Intelligence, hereby declare that this report and the work described in it are my own work, unaided except as may be specified below, and that the report does not contain material that has already been used to any substantial extent for a comparable purpose.

Code. The code used in this thesis is split into three GitHub repositories:

1. model zoo creation (original code):
   https://github.com/Emilija2000/model-zoo-jax
2. permutation augmentations (adapted for jax models based on the code available at: https://github.com/HSG-AIML/NeurIPS_2021-Weight_Space_Learning):
   https://github.com/Emilija2000/neural-net-augmentation
3. main experiments (original code):
   https://github.com/Emilija2000/meta-models-for-interpretability

Datasets. I use two publically available datasets (model zoos):

1. MNIST-HYP-10-RAND model zoo by Schürholt et al. (2022b) published under CC-BY 4.0 licence, along with the code to load it available at:
   https://github.com/ModelZoos/ModelZooDataset
2. MNIST part of Small DNN Zoo dataset published by Unterthiner et al. (2020) under CC-BY 4.0 licence at:
   https://github.com/google-research/google-research/tree/master/dnn_predict_accuracy

Libraries. Apart from standard Python libraries, the code utilises: Jax (Google, 2021), Haiku (DeepMind, 2021a), Optax (DeepMind, 2021b), WandB (Biases, 2021). Additionally, PyTorch (Paszke et al., 2019) is used to load the above-mentioned datasets.

Word count: 14996

Emilija Dordevic
August 2023

Acknowledgements

I am profoundly grateful to my supervisors Lauro Langosco and Dr David Krueger, for their thoughtful insights and steadfast support throughout this project.

I would also like to acknowledge the Cambridge Trust and the Pexim Foundation, as without their financial help I would not be able to take up the opportunity to study at the University of Cambridge.

I would like to thank Kristina Nikolić and Andrej Jakovljević for our endless study nights, and all the memories we forged throughout the year. I extend my gratitude to Rebecca, Anna, my coursemates, and all other amazing people I have met this year, for making this experience special.

Finally, this would not be possible without the unwavering encouragement from my loving parents, Violeta and Danijel, my dear sister Isidora, and my boyfriend, Nikola Mrdja. Their faith has been my beacon during the most challenging times.

Abstract

Mechanistic interpretability is a field that aims to explain the behaviour of trained neural networks by studying their learnt parameters. As most of this line of work is laborious and difficult to scale to deep networks, we aim to investigate an automatic approach to interpretability using *meta-models* - networks designed to process the weights of other neural networks, referred to as *base-models*, as input. Because collecting labelled data for interpretability is expensive, the goal of this thesis is to test a self-supervised pre-training procedure and evaluate if it is useful for downstream tasks related to interpretability. To gauge the potential of meta-models for interpretability, we instead explore their performance on easier tasks related to predicting base-models' properties.

Towards this end, we train a transformer-based meta-model on tasks related to predicting hyperparameters and the performance of base-models, as well as on a newly introduced task titled *'dropped class classification'*, related to predicting properties of training data distributions, based solely on neural network weights. It was shown that the meta-model can be used to predict these tasks, with different levels of success.

Finally, we develop a self-supervised pre-training procedure titled *'masked weight modelling'*, a task similar to masked language modelling, adapted for neural network weights as inputs. The usefulness of pre-training for downstream performance was found to be task- and data-dependent. A performance improvement was observed for almost all hyperparameter prediction and performance prediction tasks, as well as for one version of the newly-introduced dropped class classification task, but the performance declined for the easier version of dropped class classification task, which relies strongly on inductive biases in the last base-model layer. The benefits of pre-training are the most apparent when using smaller training datasets for downstream task prediction, as well as for tasks prone to overfitting. In that sense, the pre-training procedure was found to have a regularisation effect.

Table of contents

Chapter 1

Introduction

Mechanistic interpretability is a field that aims to explain the behaviour of trained neural networks by studying their learnt parameters to uncover human-interpretable algorithms they are executing. The research in this field seeks to explain the decisions neural networks make in terms of their internal components, often by specifically focusing on individual features and circuits - computational subgraphs that perform a certain function in a network. (Meng et al., 2022; Olah et al., 2020; Olsson et al., 2022; Wang et al., 2022).

Because most of this line of work relies on human labour (Räuker et al., 2023), it is not easily scalable to deep models. Instead, most current work focuses on *toy models* (Elhage et al., 2022) - networks with smaller feature spaces that are easier to probe and analyse manually. The exception to this is a recent work by Lieberum et al. (2023), but they only study a single simple mechanism (attributing letters to known answers in multiple choice question answering). Some progress regarding the automation of the circuit extraction process has recently been made by Conmy et al. (2023). However, their approach still requires the researcher to observe a specific behaviour the model is displaying and to choose the level of granularity at which it should be investigated. Multiple works propose a search for task-related subnetworks on the level of individual weights[1], with algorithms that rely on weight masking to uncover which parts are important for pre-defined tasks of interest (Csordás et al., 2020; Wortsman et al., 2020).

The motivation for this project is to test a deep-learning approach to interpretability using ***meta-models*** - networks that take the weights of other networks as input. These input networks are referred to as ***base-models*** throughout this thesis. A direct way to train a meta-model for interpretability is a supervised approach, using data from human interpretability researchers. For example, the network could learn to predict an interpretable computational graph from a circuit extracted from the trained base-model.

[1]Note that through this work we use 'weights' to refer to both weights and biases of the neural network.

As collecting the data for a direct approach is expensive, this project will test if it is useful to pre-train the meta-model on a weight-modelling task. We hypothesise that weight modelling is a hard enough problem, and can capture the information about the base-model that is needed for interpretability.

We propose a pre-training method based on weight masking, inspired by the above-mentioned techniques used to search for sub-networks that perform specific functions. Specifically, we use a pre-training task similar to masked language modelling (Devlin et al., 2018), applied on a chunked and tokenized vector of neural network weights.

As the data needed for interpretability is not accessible, this work will focus on a set of easier tasks. Specifically, we highlight the meta-model's performance on: hyperparameter prediction (Eilertsen et al., 2020; Schürholt et al., 2021), predicting base-model performance metrics (Schürholt et al., 2021; Unterthiner et al., 2020), and predicting properties of the training data. This is meant to indicate the potential of the meta-model in understanding neural network weights. Hyperparameter prediction implies having latent knowledge about the training procedure while predicting parameters of the training data involves understanding features - a necessary condition for interpretability. In that sense, if the meta-model did not perform well on these tasks, it would have been unlikely for it to be capable of harder interpretability tasks. Otherwise, their success hints at potential applications in real-world interpretability, which are left to be explored in future work.

1.1 Contributions

The main contributions of this project are as follows:

1. **Performance Analysis of the Meta-Model:** We showcase the proposed meta-model's performance when trained directly for predicting trained base-models' properties. These tasks are used as easier benchmarks, to gauge the potential of the meta-model for interpretability.

    - We establish new baselines in predicting base-model performance metrics and hyperparameters on existing datasets (Section 3.3).
    - We introduce a novel task, *dropped class classification*, meant to showcase the ability of the meta-model to predict the properties of the base-model's training data (Section 3.4).
    - We are first to investigate the effect of permutation augmentations (Schürholt et al., 2021) in direct, supervised meta-model training (Section 3.6).

2. **Introduction of Masked Weight Modelling:** A self-supervised pre-training procedure similar to masked language modelling but tailored for neural network weights as model input.

    - We show that the learning process is non-trivial, first collapsing into a simplified prediction close to data mean, before generalising to a more useful prediction. We measure the retention in the performance of the base-model after the reconstruction of its masked parts. Although the base-model's performance noticeably declines, it is better retained than that of simple baselines based on replacing masked chunks with zero vectors, mean values or random predictions (Section 4.2).
    - We conduct an initial investigation into some components of the pre-training procedure: permutation augmentations, the way the base-model weights are split into input tokens for the meta-model, and the effect of using different masking probabilities (Section 4.4).

3. **Evaluation of Pre-Training Efficacy:** We assess whether our masked weight modelling pre-training technique improves the meta-model's performance on different downstream tasks. The results vary depending on the task, and are especially promissing for small datasets prone to overfitting in direct meta-model training (Section 4.3).

1.2 Thesis Overview

This thesis is organised into five chapters, starting from this introduction.

Chapter 2: We provide essential background on meta-models, tasks and datasets they are trained on in related literature, as well as the self-supervised training methods pertinent to our pre-training objective.

Chapter 3: The first main experimental section. We detail the meta-model's architecture and explain how we train it in a supervised fashion to predict different parameters of the input base-models. We include results from two known datasets and introduce a new task to assess the meta-model's ability to recognize attributes of the base-models' training data.

Chapter 4 The second main experimental section. This chapter introduces the 'masked weight modelling' pre-training method. The focus is on evaluating its performance and determining if this pre-training can improve the meta-model's performance on downstream tasks, in comparison to the results of Chapter 3.

Chapter 5: Concludes the thesis by summarizing our findings and suggesting future research directions.

Chapter 2

Background

This chapter presents the necessary theoretical background and key conclusions from related work utilised in this thesis. In Section 2.1 we discuss the related work on meta-models with transformer architecture, which we adopt in this thesis, along with the details about processing neural weights as meta-model inputs. In Section 2.2 we introduce model zoos as datasets on which meta-models are trained. In Section 2.3 we categorise potential downstream tasks, derived from related work, that are used to evaluate the meta-models. Section 2.4 introduces self-supervised training methods related to the pre-training procedure used in this thesis. Additionally, Section 2.5 contains an overview of applications of meta-models within the literature, as well as mentions of similar models which may inspire future research directions.

2.1 Meta-Models

We define ***meta-models*** as networks that process weights of other networks as their input. We refer to these input networks as ***base-models***. Optionally, the meta-model can also access the architecture $\mathscr{A}$ of the base-model, rendering it a function of both $M(W, \mathscr{A})$. Recent literature has shown that meta-models can be successfully employed to process the weights of base-models to achieve various tasks. In this section, we outline the related work that utilises transformer-based meta-models, along with special consideration for weight-space encodings. These insights will guide the implementation of the meta-model adopted in this thesis.

2.1.1 Transformer Meta-Models

In this work we utilise a meta-model of a transformer architecture, driven by the success of transformer-based meta-models in recent literature. In particular, such models have been

shown to work well in representation learning tasks on sets of base-model weights. Schürholt et al. (2021) show that a transformer-based autoencoder outperforms a fully-connected feed-forward network. In all cases, they use a symmetric encoder and decoder and project the hyper-representations into a lower dimensional space, in comparison to NN weight-space dimensions. Similarly, Peebles et al. (2022) use a version of GPT-2 architecture as a diffusion model.

The suspected advantage of transformers lies in their potential to process long-range connections, making them ideal candidates for managing the weights of deep neural networks. Another important feature of these networks is the potential to separately learn spatial information. Both Schürholt et al. (2021) and Peebles et al. (2022) employ learnt positional encodings (Dosovitskiy et al., 2020). This feature can aid the meta-model in learning the structure of the base-model even without smarter weight-space encodings that consider network architecture.

One important aspect to note is that this model architecture does not inherently accommodate the architecture of base-models. Therefore, the meta-model itself is reduced to the function of weights only $M = M(W)$. Thus, some of the important information about the architecture $\mathscr{A}$ can be encoded through input weight-space representations.

2.1.2 Weight Space Representation

Since transformer-based models do not natively handle graph inputs, base-models need to be represented differently. Related work predominantly forsakes the information about base-model architecture and uses flattened vectors of weights.

Following Eilertsen et al. (2020), neural weight spaces of convolutional neural networks are encoded in the following way:

- For each convolutional layer and its kernels $\mathbf{H}_i$, for $i = 1..K$, and a bias vector $\mathbf{b}$: $\theta = vec(\mathbf{H}_1)||vec(\mathbf{H}_2)||...||vec(\mathbf{H}_K)||\mathbf{b}$.
- For each linear layer with weights $\mathbf{H}$ and bias vector $\mathbf{b}$: $\theta = vec(\mathbf{H})||\mathbf{b}$.
- For N layers in the CNN: $W = \theta_1||\theta_2||...||\theta_N$.

Here, $||$ is a concatenation operator, while $vec(\dot{})$ function flattens the matrix into a vector. Similar generalisations can be made for other types of networks. For example, the encoding of a multi-head attention module would be a concatenation of query, key, value and output projection matrices.

Importantly, this simple representation includes limited spatial information about neural weight space, as it only preserves the ordering between the layers. Transformer-based

approaches improve this through previously mentioned positional encodings and smarter use of input tokenisation.

Schürholt et al. (2021) compare individual weight tokenisation to joining all weights connected to a single neuron or kernel into a single token, concluding that better performance and reduced memory overhead could be achieved with joint tokenisations.

Peebles et al. (2022) employ layer-wise tokenisation, reporting that it was beneficial to *chunk* bigger layers into multiple tokens. The layer-wise tokenisation approach can be connected to the observations made by both Unterthiner et al. (2020) and Eilertsen et al. (2020) who reported that strong performance on their respective prediction tasks can be achieved using methods that harness per-layer statistics.

Although improved, layer-wise weight representations still do not encompass important information about base-model architecture and neural weight space symmetries. Therefore, this is left to be introduced by other means.

2.1.3 Weight Space Permutation Equivariance

Because of intrinsic representations of fully connected neural networks and weight-space symmetries, the same function can be represented with multiple permutations of the same set of weights. Namely, for a single layer of neurons, permuting the rows of the input weight matrix and columns of the output weight matrix in the same way, preserves the exact function of the network (Hecht-Nielsen, 1990). For permutation matrix $\mathbf{P}$, weight matrices $\mathbf{W}$, bias vectors $\mathbf{b}$, activations $\mathbf{a}$ and activation function σ in layer l:

$$\mathbf{z}^{l+1} = \mathbf{W}^{l+1}\sigma(\mathbf{W}^{l}\mathbf{a}^{l-1} + \mathbf{b}^{l}) + \mathbf{b}^{l+1} = \mathbf{W}^{l+1}(\mathbf{P}^{l})^{T}\sigma(\mathbf{P}^{l}\mathbf{W}^{l}\mathbf{a}^{l-1} + \mathbf{P}^{l}\mathbf{b}^{l}) + \mathbf{b}^{l+1} \tag{2.1}$$

The same equation holds for the channel dimensions in convolutional neural networks.

Navon et al. (2023) propose a specialised new architecture that relies on symmetries in NN weight space, implemented through pooling, broadcasting and fully connected layers. It is shown to outperform previous approaches on a simple downstream task, but it is currently adapted only for MLP networks. Instead, since we are interested in models that can be easily generalised to different architectures, we employ the same approach as other papers which use transformer-based meta-models.

If neither the meta-model, nor the encoding of the base-model weights handle this permutation equivariance, it needs to be represented in the dataset for meta-model training. For that reason, Schürholt et al. (2021) introduce permutation augmentation of neural-network weight representations.

The same augmentation is used by Peebles et al. (2022) and Schürholt et al. (2022a), who both report that it is crucial in preventing overfitting in their weight modelling tasks.

A different approach is taken by Navon et al. (2023) who design DWSNets, networks that natively handle weight-space symmetries of MLPs. Although they show that their architecture outperforms augmentation-based approaches on examples of representation learning tasks, it is tailored to specific MLP architecture, and the authors report that the DWSNets are sensitive to the initialisation scheme.

We also use permutation augmentations to embed weight-space symmetries into the dataset in sections 3.6 and 4.4.1. The architecture-based approach is outside of the scope of this thesis.

2.2 Model Zoos

A ***model zoo*** refers to a collection of independently trained neural networks. Every model in the zoo M_i is a network with architecture $\mathscr{A}_i$, which is trained using hyperparameters λ_i to model data from a dataset D_i (sampled from some data-generating distribution $D(x,y)$). Formally, neural network weights originate from a stochastic training process which depends on a radnom seed r:

$$W_i = P(\mathscr{A}_i, D_i, \lambda_i, r_i) \tag{2.2}$$

In a standard training procedure, randomness is reflected in weight initialisation, ordering of batches during training, and application of dropout accross different training iterations. Once trained, the model represents a function $\mathscr{A}_i(\cdot|W_i)$. Therefore, the model zoo can be represented as a set of tuples $\{(W_i, \mathscr{A}_i, D_i, \lambda_i) \mid i = 1, 2, \ldots, N\}$. Generally, all arguments of the training process, including random seeds, can be varied in one zoo.

This definition includes large libraries of pre-trained state-of-the-art models. However, this thesis focuses on model zoos generated specifically for neural weight space analysis, designed with special consideration for diversity and correlation between networks. Also note that this definition does not include datasets of neural network *arhitectures*, like DeepNets dataset used by Knyazev et al. (2021), but only sets of pre-trained neural network *weights* with parameters used for their generation.

In this section, we discuss properties of model zoos used in work mentioned in sections 2.1 and 2.3. This discussion is important for interpreting meta-model training results and recognising potential disadvantages of used model zoos.

The majority of model zoos in relevant papers use contain MLP and CNN architectures. Some larger zoos include models of ResNet architecture (Jiang et al., 2018; Schürholt et al.,

2022b). We are unaware of zoos with transformer models in existing literature. All zoos utilised in this work consist of small convolutional neural networks.

Parameters used to generate a model zoo directly impact the diversity and correlation among models, thus influencing what can be predicted from their weights. The use of different random seeds in model zoo generation is a particularly important characteristic discussed in literature. Unterthiner et al. (2020) purposefully create model zoos with a fixed random seed. They argue that using the same hyperparameter configurations with different random seeds results in models that are similar to each other, and warn that using the same hyperparameters in training and test splits, may lead to data leakage and unrelible evaluation. Schürholt et al. (2021) confirm that varying only the random seed while fixing hyperparameters results in high correlation between simple statistics of the weights presented by Unterthiner et al. (2020) and model performance.

However, by fixing a random seed, along with a fixed dataset D and architecture $\mathscr{A}$ used throughtout the model zoo, the mapping $\lambda \xrightarrow{P} W$ becomes deterministic, and is therefore an easier task to consider. Indeed, Schürholt et al. (2021) show that sharing random seeds between configurations reduces the variance between the models using the same activation function and the same initialization method, resulting in easier-to-predict cathegorical hyperparameters (even with linear classification directly from the weight-space). Moreover, fixing random seeds corresponds to all models in the zoo being optimised from the same point in the weight-space, which is not reflective of real-world networks.

Additionally, while considering correlations within the model zoo, it should be noted that Schürholt et al. (2021) use multiple available checkpoints from the same base-model training run. This introduces correlations in the dataset even using a single random seed per hyperparameter configuration.

We use two of the publicly available zoos trained on MNIST dataset (LeCun et al. (2010)) :

1. **MNIST-HYP zoo** was introduced by Unterthiner et al. (2020), and used by Schürholt et al. (2021) for their meta-model evaluation. It contains 9 checkpoints from 30000 separate training runs, each performed with a unique set of hyperparameters λ. The networks are of fixed architecture $\mathscr{A}$ - a convolutional neural network with 4970 parameters in total. This zoo does not vary random seeds r used for initialisation of their training procedure, which makes the hyperparameter prediction task easier than in real-world scenario. We use this zoo in Section 3.3.1 in order to contrast our training and evaluation approach with results presented by Schürholt et al. (2021) who use a similar meta-model architecture.

2. **MNIST-HYP-10-RAND zoo** was introduced by Schürholt et al. (2022b). It also contains models of a fixed architecture $\mathscr{A}$ - with 2464 parameters. It varies both hyperparameters and initialisation seeds, with 10 seeds being randomly sampled for each hyperparameter configuration. Since the training process is stochastic in this case, hyperparameter prediction tasks should be harder to perform (Schürholt et al., 2021). We use this dataset in the main pre-training experiments, and evaluation of its effectiveness. Additionally, we use it to provide baselines for pre-training in Section 3.3. The only benchmark that exists for this dataset are the predictions with linear models, on top of weights themselves or layer-wise weight statistics.

Besides these zoos, we introduce a new model zoo in Section 3.4, which varies random seeds and uses different subsets of classes from a training image dataset. While some existing works explore the connection between model weights W and hyperparameters λ, this new model zoo will also allow us to consider the connection between W and the data generating process $D(x,y)$.

We leave the exploration of model zoos that vary all four components of the data generating process for further research.

2.3 Predicting Neural Network Properties from Weights

As mentioned in Chapter 1, instead of using the inaccessible and more expensive interpretability data, we focus on a set of easier tasks related to predicting the properties of neural networks and their training procedures. These tasks are *related* to interpretability in a broader sense of the word, as they aim to reveal some aspects of the learning process behind the base-models, but not the actual algorithms they are executing. However, within the context of this thesis, they showcase the ability of the meta-models to predict some underlying characteristics of the base-models, by 'understanding' their weights. This can present a predictor of the potential meta-models have for real interpretability tasks.

This section is meant to give an overview of types of tasks in relevant literature which are discussed in terms of predicting model properties from weights. Formally, we discuss these tasks in terms of learning different mappings from a pre-defined model zoo $\{(W_i, \mathscr{A}_i, D_i, \lambda_i) \mid i = 1,2,\ldots,N\}$, as introduced in Section 2.2.

2.3.1 Model performance

Using a single instance from a trained model zoo, $(W, \mathscr{A}, D, \lambda)$, a function $\mathscr{A}(\cdot, W) : X \to Y$ models the training dataset $D = (X_1, Y_1), \ldots (X_K, Y_K)$.

Test accuracy. The expected accuracy of the model trained for classification is given by $E_{acc} = \mathbb{E}_{(x,y)\sim D(X,Y)}[\mathbb{1}(A(x,W) = y)]$. The mapping $(W, \mathscr{A}) \rightarrow E_{acc}$ is unique and can be estimated using a meta-model (Unterthiner et al., 2020), by estimating the expected accuracy with the test set accuracy. Unterthiner et al. (2020) show that the test accuracy of a trained NN can be predicted with high accuracy using simple statistics of NN weights. The predictions based on the statistics of the last layer achieve an R^2 score of more than 0.98 for CNNs trained on multiple different image datasets. Schürholt et al. (2021) confirm that good performance can be achieved even with simple linear probing of weight statistics.

Generalisation gap. The difference between expected accuracy and the accuracy estimate calculated on the training set defines a generalisation gap: $GGap = E_{acc} - \frac{1}{K}\sum_{i=1}^{K}[\mathbb{1}(A(X_i,W) = Y_i)]$. This task is again estimated by replacing E_{acc} with the test set accuracy. Both Yak et al. (2019) and Jiang et al. (2018) predict generalisation gap from NN activations, extracted from forward passes of the training set. According to the results presented by Schürholt et al. (2021), the generalisation gap is consistently a harder task to predict than the test accuracy alone, across all of their model-zoos.

OOD predictions. By fixing a different data-generating process $D^*(X,Y)$, we can consider the same metrics in terms of out-of-distribution (OOD) performance. Because of implicit biases learnt from the training data D, we can test if the meta-model can *order* the networks in the model zoo according to their performance on $D^*(X,Y)$. Unterthiner et al. (2020) also show that the predictors are able to rank networks based on OOD accuracy. Schürholt et al. (2021) expands on this by predicting the OOD generalisation gap, and shows that it is a harder task.

2.3.2 Hyperparameters

Given a model zoo as defined in Section 2.2, meta-models can also be trained to perform a reversed mapping: $(W|\mathscr{A}) \rightarrow \lambda$.

As already mentioned in Section 2.2, with a fixed dataset used for base-model training, fixed architecture, and a fixed random seed, mapping $(\lambda|\mathscr{A}) \rightarrow W$ is deterministic. However, reverse mapping is not necessarily easy to learn because training with different sets of hyper-parameters can converge to the same minimum during the optimisation process, rendering similar final weights.

Eilertsen et al. (2020) show that NN weights could be used to successfully classify some of the properties of model training, including batch size, initialisation method, optimiser and activation function used. Schürholt et al. (2021) use neural-network representations extracted

from their meta-model to predict: training epoch, learning rate, dropout, l2 regularisation coefficient, training data fraction[1], optimiser, activation function and initialisation scheme. The last three have been shown to be easier tasks in the deterministic version of the dataset.

2.3.3 Training dataset properties

There are not many relevant works that try to predict properties of the data generating distribution $D(X,Y)$ from weights. In one example, Eilertsen et al. (2020) successfully classify which image dataset was used for base-model training, as well as if the training included augmentations (with an accuracy of around 80%).

In terms of research related to interpretability, we argue that this group of tasks is the most interesting. While previously presented tasks uncover characteristics of the training process itself, predicting training data properties is more closely related to the function the base-model is performing. Intuitively, some tasks related to the training dataset can require finding and understanding *features* that the base-model is searching for.

With this in mind, we define a new, relatively simple task, and present it in Section 3.4. Furthermore, this branch of tasks stays largely unexplored.

2.4 Self-supervised learning

Self-supervised learning is a learning paradigm which leverages the data itself, or its parts, to create labels for a supervised training objective. This allows the network to learn the structure of the data in a task-agnostic manner.

In this section, we describe methods related to our pre-training procedure adopted in Section 4. For a more comprehensive view of self-supervised learning, we refer the reader to recent work by Balestriero et al. (2023).

2.4.1 Masked Data Modelling

Masked Language Modelling (MLM) is a learning objective introduced by Devlin et al. (2018), who applied it to a large transformer language model named BERT. It is a successful pre-training procedure in natural language processing and has been shown to generate useful representations for different downstream tasks.

In language modelling, the input sequence can be represented as a sequence of discrete tokens $S = \{t_1, ...t_N\}$. In each step of the MLM training procedure, a random subset of

[1] Changing training data fraction does not modify the data-generating distribution $D(X,Y)$, which is why we classify it under hyperparameter prediction tasks.

tokens $T \subset S$ is chosen to be masked, and replaced by a special token [MASK]. The training objective is to reconstruct the masked tokens based on remaining unmasked part of the input:

$$\mathscr{L}(\theta) = \sum_{i \in T} \log P(t_i | S \setminus T; \theta) \tag{2.3}$$

where θ represents model parameters, and $S \setminus T$ the set of unmasked tokens. Contrastively to traditional sequence modelling methods, the predictions are made based on the whole context vector.

Although language modelling relies on a finite vocabulary size that allows to transform the input sentences into vectors of discrete tokens, similar approaches have been applied in other domains. Bao et al. (2021) apply BERT pretraining on a Masked Image Modelling task by using an autoencoder to learn discrete representatins of image paches, which they use as tokens. On the other hand, Xie et al. (2022) and He et al. (2022) show a simplified versions of the algorithm which outperforms previously proposed complex tokenisation. They frame masked image patch reconstruction as a regression task and train it directly with MSE loss.

While BERT usually uses masked ratios of $10-20\%$, He et al. (2022) found that high ratios improve representation learning capabilities in masked image modelling. Optimal performance on downstream fine-tuning and linear probing was reached when masking 75% of patches.

2.4.2 Contrastive Learning

Contrastive learning is an SSL training procedure which originates from similarity learning - a supervised training approach to learning distances between data samples by projecting similar examples to vectors that are close to each other in representation space. This is reflected in the ideas of contrastive loss (Hadsell et al. (2006)) and triplet loss (Schroff et al. (2015)) as objective functions. Using a contrastive loss entails training a network to minimise the distance between similar pairs of images, and maximise it for dissimilar pairs. On the other hand, triplet loss requires the distance of negative pairs to be greater than that of positive pairs by a pre-defined margin.

While supervised similarity learning uses labels or fixed transformations to arive at positive and negative examples, contrastive learning entails using different data augmentations as positive examples, and treating all other pairs as negative examples, regardless of the class membership (Balestriero et al., 2023; Sohn, 2016).

Schürholt et al. (2021) have shown that representation learning of NN weights using an autoencoder benefits from a contrastive component in the loss. A similar approach would represent a valuable extension to the work presented in this thesis.

2.5 Related Work

In this section, we provide additional information about meta-model applications in the literature and relate them to related neural networks that are not classified as meta-models according to our definition.

2.5.1 Applications of Meta-Models

The proposed applications of models capable of processing NN weight spaces are diverse. While some of the related work comments on the understanding of neural weight spaces (Eilertsen et al., 2020), other works apply meta-models for optimisation (Peebles et al., 2022; Schürholt et al., 2022a). Moreover, wider interest in Implicit Neural Representations (INRs, Sitzmann et al. (2020)), neural networks trained to be effective parametrisations of different types of input signals (for example images as in work by Sitzmann et al. (2020), or 3D structures as Park et al. (2019)), motivated additional research in processing NNs as inputs to another model.

The focus of existing bodies of work can be roughly classified into two classes:

1. supervised approach, i.e. directly predicting base-model properties from weights, and
2. self-supervised approach, i.e. learning weight space representations.

The first set of applications represents meta-models that are directly trained for regression or classification tasks of interest, in a supervised fashion, given trained base-model weights as their input. For example, Eilertsen et al. (2020) train a meta-model to classify base-models based on hyperparameters used for their training. They demonstrate that their 1D-CNN-based model outperforms predictions based on simple weight statistics. Unterthiner et al. (2020) use a simple deep NN to regress base-model accuracy. Navon et al. (2023) primarily test their model on MLPs trained as INR networks. They show that they can recover properties of functions and images that the input MLP base-models represent, using their novel meta-model. In one of their experiments, they show good performance of their model on the task of predicting the generalisation gap, using a small set of MLPs trained for classification on MNIST (LeCun et al. (2010)).

Proposed methods for learning weight-space representations are more diverse and cover different generative approaches. Schürholt et al. (2021) use a bottleneck-autoencoder and train it using a combination of reconstruction and contrastive learning. Similarly, De Luigi et al. (2023) use encoder-decoder architecture to retrieve the compressed representations of input INRs trained to represent 3D shapes. Peebles et al. (2022) train a diffusion model

to predict the next model checkpoint given previous weights, previous loss, and loss at the target checkpoint. Navon et al. (2023) also show that they can learn useful representations of INR models using contrastive learning.

These generative methods are utilised for different tasks. The extracted weight space representations of Schürholt et al. (2021) are evaluated using a hyperparameter prediction task, using simple linear probing. Using the same meta-model, Schürholt et al. (2022a) suggest using weight-space modelling capabilities for weight initialisation for transfer learning. De Luigi et al. (2023) evaluate the retrieved representations on a series of tasks related to underlying 3D shapes their base-networks represented. Peebles et al. (2022) utilise their meta-model as an optimiser.

In this thesis, we combine the self-supervised and fully-supervised pipelines by investigating if the self-supervised method for learning weight representations can be used as pre-training for the set of downstream tasks of interest. We first set the baseline performance by training the meta-model on downstream tasks directly, and then compare it with the performance of the meta-model initialised from the pre-training procedure.

2.5.2 Networks related to Meta-Models

The defining characteristics of a network defined in this work as a meta-model is the fact that they are processing pre-trained base-model weights. This means that a meta-model is supposed to discern certain characteristics of the base-model without accessing its training data. This important property differentiates meta-models from similar neural networks mentioned in the literature.

In contrast, hypernetworks are neural networks trained to predict the parameters of another model, given the base-network architecture and its training data. These models are trained together with the base-network to optimise the downstream task of interest (Ha et al., 2016; Zhmoginov et al., 2022).

Recent advancements in hypernetworks could propose a solution to the problem of meta-models handling different base-models architectures. Knyazev et al. (2021) utilise a graph-hypernetwork to predict the weights of different network types, including more complicated transformer and ResNet architectures. The input network is represented as a computational graph, with nodes encoding operation type (convolution, attention etc.) and weight dimensions, while edges are represented as an adjacency matrix of layer connections, allowing them to encode residual layers. They expand on this in the following work (Knyazev et al., 2023) by using a Graphormer architecture - a transformer with a modified attention mechanism which processes edges in a computational graph, making it possible to generalise

to different architectures. Although outside of the scope of this thesis, we believe that adopting a similar architecture as a meta-model is a valuable step in future research.

Another important distinction should be made between meta-modelling approaches and methods that utilise neuron activations (Dalal et al., 2023; Yak et al., 2019). While these are valuable, this thesis focuses on a dataset-agnostic view of interpreting neural networks and their parameters.

Chapter 3

Experiments: Direct Meta-Model Training

Given our interest in exploring the meta-model's capabilities for interpretability-related tasks, establishing a baseline is crucial. In this chapter, we present the results of direct meta-model training-Later, in Section 4.3, we investigate whether the self-supervised pre-training procedure we introduce enhances this performance.

We describe the meta-model architecture in Section 3.1, and define the direct training objective in Section 3.2. In Section 3.3, we set the baselines on a series of hyperparameter and performance prediction tasks. Additionally, in Section 3.4 we introduce a new task meant to demonstrate the meta-model's ability to predict properties of the base model's training data.

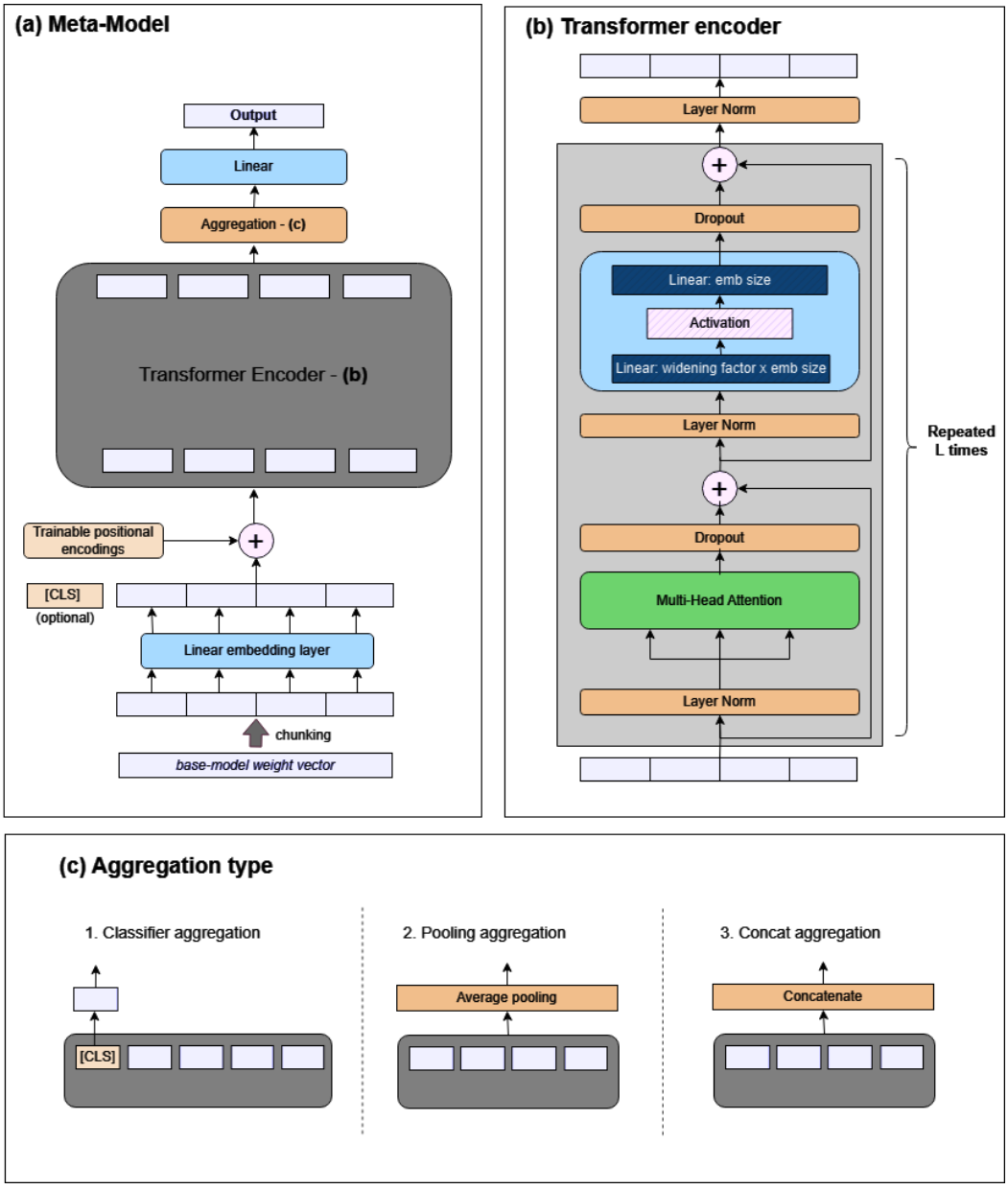
In Section 3.5 we explore one component of the meta-model architecture in more detail.

Unless the opposite is stated, we utilise permutation augmentations proposed by Schürholt et al. (2021), and defined in Section 2.1.3. This is especially significant with small amounts of training data. In Section 3.6, we show the effect of using this augmentation in direct meta-model training.

3.1 Meta-Model Architecture

Motivated by the success of transformer-based models on neural weight modelling tasks, as described in Section 2.1, we adopted a transformer-encoder architecture for our meta-model. This design is similar to the Vision Transformer (ViT) by Dosovitskiy et al. (2020). The full meta-model architecture is shown in figure 3.1.

Fig. 3.1 Meta-model architecture used in supervised training. The base-model weights are divided into equal-sized chunks, which are embedded using a linear layer. Transformer encoder architecture is the same as in ViT (Dosovitskiy et al. (2020)), with layer normalisation before MHA; (a) main meta-model architecture, additional [CLS] token is used only with classifier aggregation; (b) transformer encoder with L layers; (c) types of 'aggregation' that join the output sequence into a single output vector



As presented in the figure 3.1, the input weight vector is divided into chunks, which are used as input to the meta-model. Similarly to Peebles et al. (2022), we implement layer-wise chunking. Weights from each layer of the base-model are flattened separately,

and subsequently divided into pieces of some pre-defined maximal size. Smaller chunks are padded with zeros. Also relying on work by Peebles et al. (2022), we use a simple linear layer to project input weight chunks into tokens of the attention-module hidden size.

We use the same transformer block as Dosovitskiy et al. (2020), with additional layer normalisation at the output - as used originally by Vaswani et al. (2017). We optionally apply dropout after both multi-head attention and the dense block. In all experiments except in Section 3.3.1, we use a linear layer with a widening factor 4, and GeLU activation function (Hendrycks and Gimpel, 2016) in the dense block. The architecture in Section 3.3.1 is modified to match the original baseline, as discussed within that section.

To perform downstream classification or regression, we project the output from the transformer network with three different aggregations:

1. classifier aggregation takes the output of the first token, as in the Vision Transformer (Dosovitskiy et al. (2020)); note that we use an additional classification token in this case, although we found that the performance does not degrade without it,

2. average pooling aggregation,

3. concatenation of all sequence outputs.

In Section 3.3.1, to match the baseline architecture, we use the classifier aggregation with an additional linear layer. In other experiments, we use concatenation as a default option. We discuss these aggregation choices and their implications further in 3.5.

The weights of the transformer module are initialised with a variant of variance scaling with a scheme $scale = \frac{2}{L}$, where L is the number of transformer blocks. This initialisation is introduced by Radford et al. (2019) in the GPT-2 architecture and is meant to account for the effect of accumulation on the residual path.

We use learnt positional embeddings, initialised from a truncated normal distribution with a standard deviation of 0.02. This potentially provides a way of preserving structural information about the base-model architecture, especially for single-architecture model-zoos which are explored in this thesis.

3.2 Direct training objective

As defined in Section 2.2, a model zoo represents a collection of trained neural network models characterized by their weights, architecture, dataset, and hyperparameters used in their training. Given a model zoo $\{(W_i, \mathscr{A}_i, D_i, \lambda_i) \mid i = 1, 2, \ldots, N\}$ our goal is to train the meta-model to infer base-model properties using W_i and $\mathscr{A}_i$.

Following the categorisation of property prediction tasks described in Section 2.3, our tasks of interest include:

- performance metrics - accuracy and generalisation gap,
- a subset of varied hyperparameters λ_i,
- varied properties of the image datasets D_i,

The exact set of tasks is presented together with the experiments for each model zoo. The meta-model is trained separately for each task.

Given base-model weights W and known model architecture $\mathscr{A}$ (optionally incorporated through weight chunking), we train the meta-model using:

- Cross-entropy loss for classification-centric predictions:

$$\mathscr{L}_{\text{cls}} = -\sum_{i=1}^{N} y_i \log(p(y_i|x_i, W_i, \mathscr{A}i)) + (1-y_i)\log(1-p(y_i|x_i, W_i, \mathscr{A}i)) \tag{3.1}$$

- Mean squared error (MSE) for regression-based tasks:

$$\mathscr{L}_{\text{reg}} = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{3.2}$$

where y_i is the target label, $p(y_i|x_i, W_i, \mathscr{A}i)$ is predicted probability (for classification), $\hat{y}_i$ is predicted value (for regression), and N is the number of models in the training part of the model zoo split.

Furthermore, regression is evaluated through R^2 score on the test set, while we report on test accuracies for classification tasks. The R^2 score measures the proportion of variance explained by the model, i.e. mathematically:

$$R^2 = 1 - \frac{SS_{pred}}{SS_{tot}} \tag{3.3}$$

where SS_{pred} is a sum of squares of residuals, and SS_{tot} is a sum of square differences between observed variables and the mean value.

3.3 Outperforming Downstream Task Prediction Benchmarks

In this section, we present the results of direct meta-model training on existing model zoos. We show that direct training yields good performance on downstream tasks of interest and outperforms previously established simple benchmarks. Part of the results presented here is used as a baseline in section 4, to evaluate how the pre-training affects the meta-models's performance.

3.3.1 Comparison With Other Natural Baselines: MNIST-HYP Dataset

We demonstrate that direct meta-model training outperforms the benchmarks based on probing pre-learnt weight representations. This evidence supports our direct training approach and the subsequent pre-training-fine-tuning pipeline, contrasting with Schürholt et al. (2021), who evaluate their self-supervised training procedure based on the quality of extracted weight representations. Although insightful, their evaluation does not directly address the implications of pre-training on meta-models' capabilities for downstream tasks of interest, which is our central aim. Hence, our analysis adopts a task-oriented evaluation, aligning with our research question.

To underscore the superiority of direct training over mere probing of learnt representations — justifying its use as our baseline — we benchmark the outcomes of direct meta-model training for hyperparameter and model-performance prediction tasks on the MNIST-HYP dataset (mentioned in Section 2.2) against the referent results provided by Schürholt et al. (2021). To offer a fair comparison with their work, we train the meta-model of the same architecture as the encoder part of the network used in their paper. Therefore, as mentioned in Section 3.1, we use an additional classification token, classifier sequence aggregation method, and an additional linear layer that projects transformer output to latent dimension 1200. This projection is followed by a single linear layer for classification, which is trained together with the rest of the meta-model.

We match all architecture parameters to their official implementation, using 4 attention layers, with 4 attention heads in each, the embedding size in attention layers of 512, with a widening to size 1000 in the dense portion of the transformer block. Like in their implementation, we use the Leaky ReLU activation function in these experiments. We also use a maximum chunk size of 800 weights, but chunk the weights layer-wise rather than neuron-wise.

Following Schürholt et al. (2021), we use all checkpoints, with 70% of them used for training, and 15% each for validation and testing, ensuring that all checkpoints from the same training run belong to the same part of the split.

We use batch size 500 for all downstream tasks and train for 75 epochs. We use Adam (Kingma and Ba (2014)) optimiser, with a learning rate 0.00003 and weight decay with coefficient 0.0002. Note that the hyperparameters were not tuned separately for optimal performance for each task.

In these experiments, because of the large dataset size, we do not use permutation augmentations, conversely to other experiments in this thesis. It should be noted that this could lead to more noticeable overfitting, and results that do not depict the maximum meta-model performance on the chosen set of tasks. However, because of the very large dataset size and limited variability in the data stemming from the use of fixed random seeds during model zoo creation, this augmentation is not crucial for attaining satisfactory downstream prediction outcomes.

We present the test results in table 3.1.

Table 3.1 Direct meta-model training for hyperparameter and base-model-performance prediction tasks on MNIST-HYP dataset (Unterthiner et al. (2020)). This is juxtaposed with three baseline methods from Schürholt et al. (2021): direct linear probing of model weights W; linear probing of weight space statistics $S(W)$ described by Unterthiner et al. (2020); and linear probing of weight space representations extracted from an autoencoder. Notably, the architecture of this autoencoder mirrors that of our meta-model.

	Direct meta-model training		Baseline results*		
Task	Test R^2 score	Test accuracy	W	$S(W)$	Representations
Epoch	0.634	-	0.258	0.332	0.500
Accuracy	0.9918	-	0.747	0.815	0.949
Learning Rate	0.574	-	0.293	0.343	0.371
Dropout	0.601	-	0.125	0.165	0.201
L2 Regularisation	0.526	-	0.285	0.192	0.358
Training Data Fraction	0.371	-	0.038	0.078	0.159
Activation Function	-	0.899	0.886	0.811	0.887
Initialisation Method	-	0.954	0.946	0.720	0.887
Optimiser	-	0.997	0.767	0.654	0.664

*As baselines, we cite the results of Schürholt et al. (2021), without running their experiments

We observe improved results in all downstream tasks. The most significant improvement can be seen for hyperparameter regression tasks. The easier, classification tasks, which perform well even with linear classification from the weight space display less drastic

improvement. This is a feature of a model zoo which uses a single random seed for base-model training, which makes these hyperparameters easy to predict even by probing weight vectors directly.

While we do consider these results valuable, showing the improvement that arises from direct meta-model training, this comparison should be viewed with caution. The differences between the two implementations include the use of augmentations in their work, and layer-wise instead of neuron-wise weight tokenisation. Because we expect these modifications to only further improve our training results, we believe that the conclusions still hold.

These results demonstrate the capacity of our meta-model for downstream task prediction, and justify using direct meta-model training as a baseline. However, this model-zoo lacking the variability in initialisation seeds, makes some of the presented problems easier to predict. For this reason, we do not use this zoo in pre-training evaluation, but utilise the baselines from the next section.

3.3.2 Model Zoo With Variable Seed: MNIST-HYP-10-RAND Dataset

As discussed in Section 2.2, introducing randomness in base-model initialisation seeds yields a stochastic data-generating process, and potentially makes the classification tasks harder to predict on the resulting dataset. To consider this, we present the results of our meta-model on MNIST-HYP-10-RAND made available by Schürholt et al. (2022b).

In this thesis, we use two distinct subsets of the MNIST-HYP-10-RAND dataset. The larger subset comprises 6000 training samples, 1000 validation samples, and 500 test samples. The smaller subset is non-overlapping and contains 500 training samples, 500 validation samples and 500 test samples. Because of the way it is used, we will refer to the smaller dataset as the fine-tuning dataset. This data selection is meant to be emblematic of real-world situations: with large amounts of unlabelled data accompanied by smaller, labelled training sets. We retain the original training-validation-test split of the dataset, and split each into two parts. Because one of the downstream tasks is epoch prediction, we chose to include 5 uniformly sampled checkpoints from each training run. The checkpoints from the same training run are always in the same part of the split.

We have devised two distinct training and evaluation pipelines used later in chapter 4. In this section, we provide the baseline performance of meta-model training for both, but the purpose of this experiment design is to evaluate the pre-training procedure more thoroughly.

- In the first pipeline, we integrate both datasets, resulting in a combined set with 6500 training examples (retaining the split from the original subsets). The purpose of this pipeline is to assess the benefits of pre-training, where we make sure that both pre-

trained and randomly initialised meta-models are exposed to the same amount of data. Therefore, these experiments can illuminate the intrinsic advantages of the pre-training process.

- The second pipeline involves self-supervised pre-training on the larger dataset followed by direct meta-model training using only the fine-tuning dataset. This is employed to evaluate if the potential benefits of pre-training are more noticeable for small datasets. The choice to not include the samples already used in pre-training in the subsequent fine-tuning experiments is made to eliminate any effects of data leakage and to derive more generalised conclusions.

We train the meta-model with 5 transformer blocks, each with 16 attention heads, and attention size 256. The chunk size is fixed to 8. We use batch size 32 in all experiments. We fine-tune the training hyperparameters for each of the tasks, by selecting the ones which yield the best validation performance (minimal loss). The resulting hyperparameter list is available in appendix A.1.

The experiments on the fine-tuning dataset are always run for 150 epochs, while we use 50 epochs for the full dataset. Accounting for possible overfitting, we evaluate the test set performance using the epoch with minimal validation loss.

It should be noted that the best validation performance for the activation function prediction task was obtained for a set of hyperparameters that does lead to noticeable overfitting. However, we stress once again that the hyperparameters were chosen to minimise the validation loss at its best epoch, and further experiments with stronger regularisation could not outperform the presented results.

The test results on both subsets are shown in table 3.2.

Table 3.2 Direct meta-model training for hyperparameter and base-model-performance prediction tasks on MNIST-HYP-10-RAND dataset (Schürholt et al. (2022b)), for two dataset sizes: full dataset with 6500 training samples, and a fine-tuning dataset with 500 training samples.

	6500-samples training data		500-samples training data	
Task	Test R^2 score	Test accuracy	Test R^2 score	Test accuracy
Epoch	0.2348	-	0.1009	-
Accuracy	0.9597	-	0.8809	-
Generalisation Gap	0.7981	-	0.3934	-
Activation Function	-	0.8281	-	0.7299
Initialisation Method	-	0.6321	-	0.6562

The accuracy prediction task yielded the best results, whereas epoch prediction proved the most challenging for our meta-model in this setting. Noticeably, the results for activation function and initialisation method prediction are lower than on the MNIST-HYP dataset discussed earlier. This is expected not only because of the dataset size but also the usage of variable initialisation seeds, which resulted in a more challenging model zoo.

We use the presented results as a baseline for pre-training evaluation in Section 4.3.2.

3.4 Dropped Class Classification Task

We introduce a new task titled *dropped class classification*, aiming to demonstrate the ability of a meta-model to form conclusions about the training data distributions based on the resulting weights of the trained models. Arguably, this line of work is more related to interpretability than hyperparameter prediction, as it requires insight into the logic behind the base-model predictions, as opposed to understanding the training procedure itself. We show the capabilities of the proposed meta-model in tackling this objective.

3.4.1 Task definition

We train a model zoo $\mathscr{M}$ encompassing M convolutional neural networks. This zoo is trained on an image dataset I consisting of N distinct classes $\{c_1, c_2, ...c_N\}$.

Each network in the zoo, CNN_i, $i \in 1..M$ is trained to perform image classification into $N-1$ classes. Specifically, for each network CNN_i, we randomly (uniformly across all classes) select a class $c_{i_k} \in \{c_1, c_2, ...c_N\}$ to be excluded during training. We therefore train CNN_i on a subset of I containing remaining $N-1$ classes, $I_k = I \setminus c_{i_k}$.

Given the resulting set of neural network weights, the goal of the 'dropped class classification task' is to predict, for each CNN_i, which class c_{i_k} was omitted during training. We experiment with two settings of the task: the easier experiment, where the classes are always ordered in the same way (with a single class being dropped from this ordering), and the harder setting - where we randomly permute the last layer of the network, so that 9 remaining classes cannot be distinguished based on the ordering of its inductive biases.

3.4.2 Model Zoo Generation

We train a model zoo on images from MNIST LeCun et al. (2010) dataset. In each iteration of a zoo-generating process, we uniformly sample a class to be dropped from training images. We scale loaded images to a range of $[0, 1]$ and do not use additional augmentations.

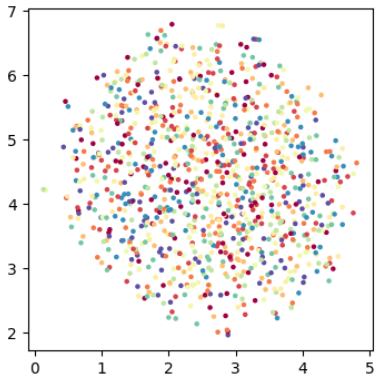
The resulting model zoo contains 7500 small CNNs. Out of these, 23 models are discarded because of failed training or poor classification accuracy, leaving 7477 models in the zoo. The base-model architecture consists of three convolutional layers followed by average pooling and two linear layers and has 2043 parameters in total. The exact architecture is presented in table 3.3.

Table 3.3 Model architecture used in dropped class classification model zoo

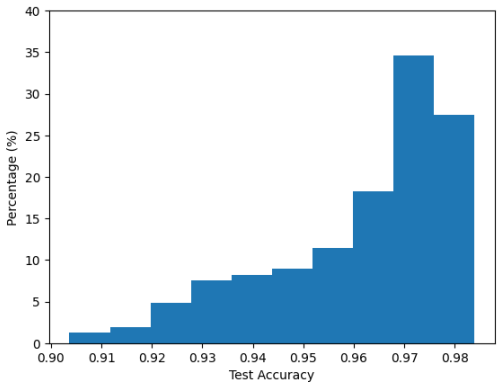
Layer Component	**Parameter**	**Value**
Conv 1	input channels	1
	output channels	8
	kernel size	5
	stride	1
	padding	0
	Max Pooling kernel size	2
	Max Pooling stride	2
Conv 2	input channels	8
	output channels	6
	kernel size	5
	stride	1
	padding	0
	Max Pooling kernel size	2
	Max Pooling stride	2
Conv 3	input channels	6
	output channels	4
	kernel size	2
	stride	1
	padding	0
	Max Pooling kernel size	2
	Max Pooling stride	1
Linear 1	input channels	16
	output channels	20
Linear 2	input channels	20
	output channels	9
Total Parameters		2043

For simplicity, we keep the majority of hyperparameters fixed and vary only initialisation seeds, weight decay, dropout and batch size. Categorical hyperparameters are always kept the same: we use *Leaky ReLU* activation function, Adam optimiser (Kingma and Ba (2014)) and default weight initialisation schemes in Haiku (Xavier initialisation (Glorot and Bengio, 2010) for linear and convolutional layers). The batch size was either 32 or 64. Weight decay was sampled logarithmically in range $[10^{-4}, 10^{-2}]$ and dropout uniformly in range $[0, 0.5]$.

Fig. 3.2 Visualisation of characteristics of generated dropped-class classification model zoo for 1000 data trained networks, each trained on 9 out of 10 MNIST (LeCun et al. (2010)) classes. (a) Two-dimensional UMAP dimensionality reduction (McInnes et al. (2018)); different colours represent different subsets of MNIST classes; (b) Test accuracy distribution in the last training epoch.



(a) UMAP dimensionality reduction



(b) Test accuracy distribution

It should be noted that the chosen method of model-zoo generation leaves space for further improvements. As mentioned in Section 2.2, the setting with varying random seeds and fixed hyperparameters might result in a simplified version of the problem in which simple weight statistics become highly correlated with the resulting performance of the model. However, since the target variable is categorical, the reasoning behind this decision was spreading out potential clusters that might exist in the data. To show this, in figure 3.2a we visualise the UMAP dimensionality reduction of 1000 samples from the trained model zoo. We leave the exploration of the effect of varied hyperparameters for this task for further research.

We only use a single checkpoint per training run, making sure that the training set for dropped class classification consists only of well-trained networks. We visualise the distribution of test set accuracies of our trained zoo in figure 3.2b.

3.4.3 Experimental Details

Similarly to Section 3.3.2, we divided the created model zoo into two subsets. The larger subset, referenced later in chapter 4, comprises 4500 training samples, 500 validation samples, and 477 test samples. Meanwhile, the smaller, non-overlapping fine-tuning set has 500 samples in each of these categories (note that we do not decrease validation and test sizes to reduce the noise in evaluation metrics).

We provide baseline results for the full dataset (preserving the same training-validation-test split) and the fine-tuning dataset. Both will be used in pre-training evaluation in Section 4.3.1. On both datasets, we train the meta-model on both variants of the task described in the previous section: the easier variant with MNIST classes always retaining the same order in the base-model classifier output layer, and the harder task with randomly permuted ordering of classes.

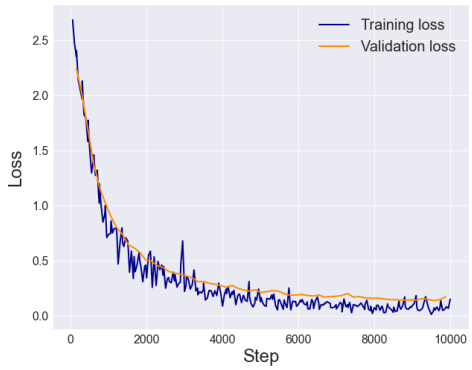
We train a meta-model with 3 transformer blocks, with 16 attention heads in each multi-head attention, and attention size 256. Weights are chunked layer-wise with chunk size 8 and we apply permutation augmentations on the training set, with a new permutation used in each training epoch.

The meta-model is trained with an Adam optimiser, using a learning rate of 0.0002 and batch size 32. For the easier task, we use weight decay of 0.0001 and no dropout, while a weight decay of 0.002 and dropout of 0.05 are applied for the harder-task training. We evaluate the results in the epoch with the best validation loss.

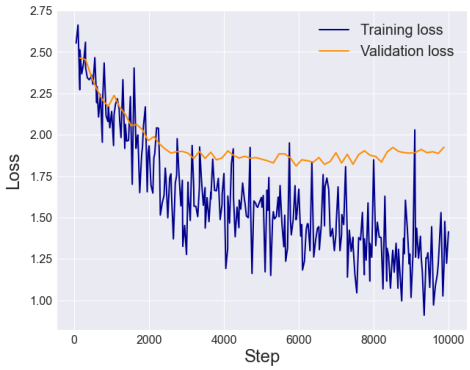
3.4.4 Results

We provide baseline results of direct meta-model training for dropped class classification tasks. In the first set of experiments, we train the meta-model on the full model zoo - with a training set consisting of 5000 models. The training curves are presented in figure 3.3, while table 3.4 shows the final test accuracies for both versions of the task.

Fig. 3.3 Learning curves for direct meta-model training for dropped-class classification tasks. (a) The easier version with image classes in a fixed order in the base-models; (b) The harder version with randomly permuted class order



(a) Easy dropped class classification task



(b) Hard dropped class classification task

The meta-model easily identifies the missing class in the example with the fixed class order, achieving the accuracy of 93.75% on the test set. However, the performance on the

Table 3.4 Test results for dropped class classification tasks. An easy task involves using a fixed class order in the base-model output layer, while this layer is randomly permuted in the hard task.

Task	Test accuracy
Easy dropped class classification	0.9375
Hard dropped class classification	0.3780

more realistic problem, with permuted class order, drastically declines. The stark difference between the results achieved on these two tasks signalises that the meta-model likely leverages the predictable pattern of inductive biases in the last base-model layer to infer the missing class in the easier setup. Randomly permuting the last layer, therefore, makes the task more challenging, as it is perhaps forcing the meta-model to discern more complex relationships within the weights.

Additionally, we provide results of meta-model training for dropped class classification tasks on the fine-tuning dataset, with 500 samples in the training set. These results are provided purely as a baseline for the pre-training procedure, utilised in Section 4.3.1, as they will be used to evaluate if the pre-training improves the meta-model performance on a smaller dataset.

Learning curves are shown in figure 3.4.

Fig. 3.4 Learning curves for direct meta-model training for dropped-class classification tasks, on a small fine-tuning dataset with 500 base-models in the training dataset. (a) The easier version with image classes in a fixed order in the base-models; (b) The harder version with randomly permuted class order

(a) Easy dropped class classification task

(b) Hard dropped class classification task

While the results show clear overfitting on this smaller dataset, further experimentation with higher regularisation hyperparameters led to lower validation performance. The test

accuracy is evaluated using a checkpoint from the epoch with the best validation loss, and the results are presented in table 3.5.

Table 3.5 Baseline test results of direct meta-model training for dropped class classification task, using a small dataset with 500 training samples. The easy task involves using a fixed class order in the base-model output layer, while this layer is randomly permuted in the hard task.

Task	Test accuracy
Easy dropped class classification	0.6354
Hard dropped class classification	0.2083

3.5 Types of aggregation

We explore three different ways to combine the sequence output from the transformer-encoder part of the meta-model, into a single vector, before applying the last classification or regression layer to it. As in figure 3.1, the three types of operations used here are:

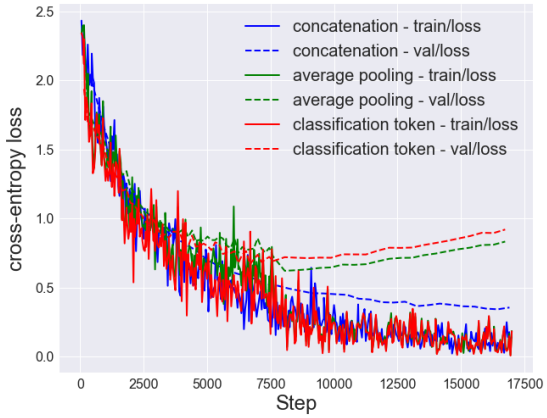
1. classifier aggregation - we add a classification token and only use its output for downstream task prediction;
2. average pooling aggregation - we pool the output sequence across the sequence dimension, to retrieve the output of equal size as a single token;
3. concatenation aggregation - we concatenate all output tokens and use a linear layer to produce the output.

We train a meta-model with 4 transformer blocks, with 16 attention heads and attention size 256, on the full Dropped-Class-Classification model zoo introduced in Section 3.4. The chunk size is set to 64, a small enough number to retain the expressivity of the meta-model, but large enough to not represent a considerable information bottleneck for average pooling or classification token. With this chunk size, we apply layer-wise weight chunking resulting in 38 input chunks. The hyperparameters for models with the three aggregation types are tuned separately for the best validation performance. We use a learning rate of 0.00002 and weight decay of 0.0001 for concatenation aggregation, and a learning rate of 0.0005 with weight decay of 0.0002 for the other two methods.

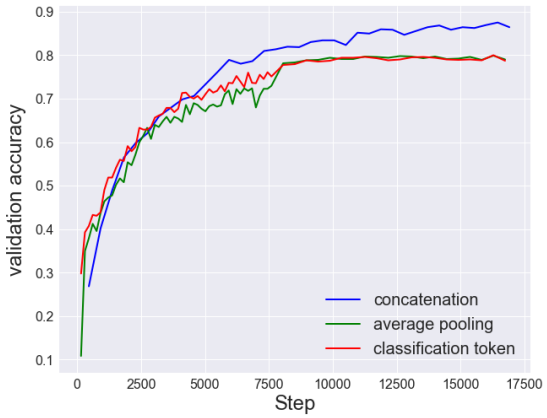
The resulting learning curves are shown in figure 3.5.

In this setting, concatenation noticeably outperforms the other two methods, which exhibit overfitting even for the best hyperparameters found. Further increase in regularisation

Fig. 3.5 Comparison of meta-model training results with different types of operations that aggregate output sequence from a transformer block into a single vector. Concatenation outperforms the other methods.



(a) Training and validation losses



(b) Validation accuracy

hyperparameters resulted in lower validation accuracy. This result is not surprising, considering the information bottleneck that the limited chunk size creates for the other two methods.

We should note that using concatenation has two negative implications: it results in a significantly larger number of parameters and is hard to adapt to multi-architecture zoos. We find that the first condition is not limiting in our case because we use only small base-models in this thesis. However, in order to apply the meta-model to a real-world case, this factor is very restrictive. On the other hand, average pooling and concatenation appear more difficult to train well, especially with smaller chunk sizes.

Despite the disadvantages, we chose to perform experiments in other sections of the thesis using concatenation aggregation. The reason for that is twofold: empirically easier hyperparameter tuning that avoids overfitting on smaller model zoos, and the possibility to explore small chunk sizes in the pre-training procedure without creating an information bottleneck in downstream training.

To extend this method to model zoos with multiple base-model architectures, we would need to adopt one of the approaches which produces output of a fixed size. As a potential improvement, a learned aggregation method, such as a 1D convolution, can be applied. A similar method could potentially mitigate the deficiencies of the presented aggregations. However, further analysis of this issue is not analysed in this thesis.

3.6 The effect of permutation augmentation

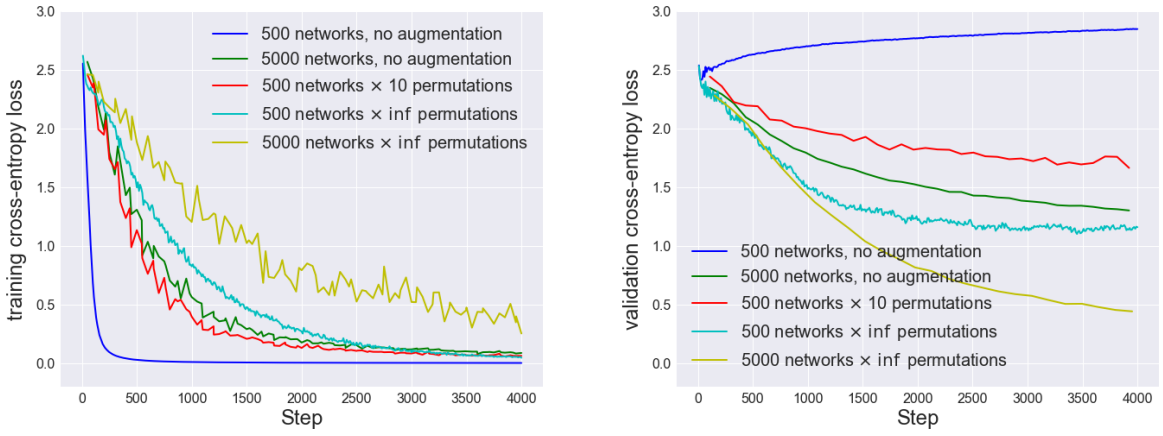
In this section, we show the effect of using permutation augmentations (Section 2.1.3) on the direct meta-model training process. We demonstrate this by comparing five configurations for meta-model training, namely training on a dataset:

1. with a fixed number of trained base-networks $\frac{N}{10} = 500$, without augmentations
2. with a fixed number of trained base-networks $N = 5000$, without augmentations
3. with $\frac{N}{10} = 500$ trained base-networks, and its 10 augmentations
4. with $\frac{N}{10} = 500$ trained base-networks, and using new permutation in each epoch (i.e. the meta-model does not see the same input twice)
5. with $N = 5000$ trained base-networks, and using new permutation in each epoch.

For all examples, we use the Dropped-Class-Classification model zoo introduced in Section 3.4.

The meta-model used in this section has 4 transformer layers, with 16 attention heads, and attention size 256. Weight chunking is done with a chunk size of 8. The models are trained with Adam optimiser, learning rate of 0.0001 and weight decay coefficient of 0.0001. We use batch size 32 and train all configurations for the fixed number of steps (batches). The resulting learning curves are shown in figure 3.6

Fig. 3.6 Learning curves showing the impact of permutation augmentation on direct meta-model training



(a) Training loss

(b) Validation loss

The meta-model is prone to overfitting on small training datasets. This is especially apparent with the small dataset of 500 networks, in comparison with augmented versions of it.

On the other hand, while introducing augmentations improves overfitting, by comparing the results of training on datasets of the same total size (two datasets of size 5000, and two 'infinite' datasets), we can conclude that adding new networks contributes more to the meta-model performance than the equal number of augmented networks. Since the augmentation process is significantly cheaper than training new base-models, it is still beneficial and allows us to experiment with relatively small model zoos. The training with the model zoo of 5000 networks and their permutations generalises well on the validation set.

3.7 Summary

In this chapter, we have tested the performance of meta-models trained directly for downstream tasks of interest. We have shown that direct meta-model training achieves superior results compared to probing pre-learnt representations of neural-network weights, thus motivating our task-oriented pre-training evaluation approach. We have set the baseline results for meta-model training on two datasets: MNIST-HYP-10-RAND model zoo, and a model zoo that we train for dropped class classification tasks. Additionally, we have shown that concatenation works the best as the method for aggregating sequence outputs from a transformer to train it for downstream tasks. Finally, we have demonstrated the effectiveness of permutation augmentations to ensure good generalisation in direct meta-model training, although they are less effective than using the same number of independent data samples.

Chapter 4

Experiments: Masked Weight Modelling Pre-Training Procedure

In this chapter, we introduce the masked weight modelling pre-training procedure and explore its effectiveness in enhancing the meta-model's capability in downstream task prediction.

Firstly, in Section 4.1 we explain the masked weight modelling task, the way we adapt the meta-model architecture for it (Section 4.1.1), and how the meta-model is later fine-tuned for downstream tasks (Section 4.1.2). Some experimental details are presented in Section 4.1.3.

We perform experiments in this section on two model zoos: our Dropped-Class-Classification zoo, and the MNIST-HYP-10-RAND zoo. In Section 4.2, we evaluate the pre-training procedure itself for both zoos. Then, in Section 4.3, using the pre-training procedure to initialise the meta-models, we train them for relevant downstream tasks on both model zoos and compare the resulting performance with the baselines set in Chapter 3.

Furthermore, in Section 4.4 we investigate how different components affect the pre-training process.

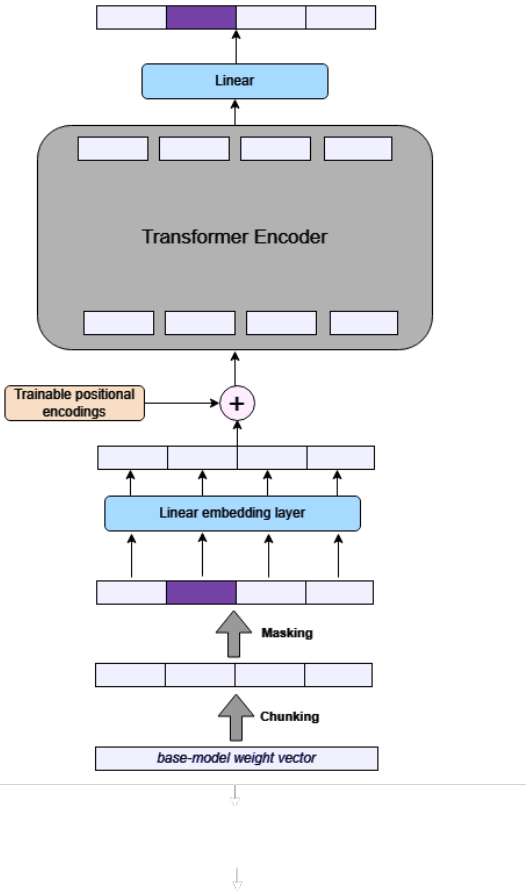
4.1 Method

In this section, we present the masked weight modelling pre-training task, including the meta-model architecture, weight tokenisation and training objective. Furthermore, we explain how the pre-trained weights are utilised in fine-tuning for the downstream tasks of interest.

4.1.1 Masked Weight Modelling Task

The pre-training method is illustrated in figure 4.1. Similarly to the method explained in Section 3, we divide the base-model weight vector into chunks, creating the input sequence. As a pre-training procedure, we randomly mask some of the input chunks, and train the meta-model to reconstruct the masked parts.

Fig. 4.1 An illustration of a masked weight modelling pre-training procedure, using a transformer-encoder meta-model



Weight chunking. We compare three types of weight chunking:

- Flattening the whole vector of weights together and dividing it into equal-size weight chunks, while only the last chunk is padded to match the chosen chunk size;
- Layer-wise weight chunking, with each layer being flattened and divided into chunks separately, as explained in section 3. This is used as a default weight chunking option for downstream training;

- Layer-wise weight chunking with additional one-hot encoded indicators for layer type (in our case linear or convolutional). This method adds information about the base-model architecture.

More detailed discussion and experiments are presented in section 4.4.2,

Masking. We mask weight chunks randomly and uniformly, by masking each chunk with some pre-defined probability p_{mask}. We replace masked portions of the network with zero vectors, and add a binary indicator to each input chunk, to signify the masking - with 1 for masked parts, and 0 for others. Since each chunk is tokenized with a linear layer, this is equivalent to a learnt masked embedding.

Architecture. Same as in section 3.1, we use the transformer encoder meta-model architecture with learnt positional encodings. The model takes in both the masked and the unmasked input weight chunks and outputs the sequence of the same length. We apply a linear layer to the output sequence to project the tokens back to the original weight vector dimension and weight scale.

Pre-training learning objective. We train the meta-model to directly reconstruct the masked chunks, given the whole masked input. The loss is calculated only on the masked parts of the output. We frame this as a regression task and use MSE loss over masked parts of the prediction:

$$\mathscr{L}_{\text{masked MSE}} = \frac{\|\mathbf{M} \odot (\mathbf{Y} - \hat{\mathbf{Y}})\|_2^2}{\sum \mathbf{M}} \tag{4.1}$$

where $\mathbf{Y}$ is a 3D target vector of dimensions $(\text{batch size}, \text{number of chunks}, \text{chunk size}$, and $\hat{\mathbf{Y}}$ is the meta-model output prediction. Matrix M is a binary matrix of the same dimension, with ones for all weights in masked chunks, excluding the added mask indicator. Symbol $\odot$ represents the Hademard product, while $\sum \mathbf{M}$ returns the total number of masked weights. In our experiments, the loss is instead normalised by the number of masked chunks. Since the chunk size is equal between layers and different experiments, this is treated as equivalent.

Layer-wise loss normalisation (LWLN) correction. In contrast to other sources of data (text or images), the noise is not uniformly distributed across the base-model weight vector. According to the observations made by Schürholt et al. (2022a) in a similar weight-modelling task, this could cause reconstruction performance in some layers to drop significantly. Following their solution, to mitigate this, we implement layer-wise loss normalisation:

$$\mathscr{L}_{\text{LWLN}} = \frac{1}{C}\sum_{l=1}^{L}\frac{\|\mathbf{M}_l \odot (\mathbf{Y}_l - \hat{\mathbf{Y}}_l)\|_2^2}{\sigma_l^2} \tag{4.2}$$

where σ_l^2 represents the average variance of layer l, $\mathbf{M}_l$ represents a binary mask for weights, $\mathbf{Y}_l$ targets, and $\hat{\mathbf{Y}}_l$ meta-model predictions for the same layer of the base-model. Constant $C = \sum_{l=1}^{L}\frac{1}{\sigma_l^2}\sum\mathbf{M}_l$ scales the loss to match the order of the original $\mathscr{L}_{\text{masked MSE}}$.

It should be noted that if the weight chunking method contains added indicators, they are excluded from the binary mask, and the loss is calculated only for the reconstruction of the original base-model weights.

4.1.2 Fine-tuning for downstream tasks

To test the effect the pre-training procedure has on downstream performance, we replace the effective 'unembedding layer' of the meta-model, with a new classification/regression head. This new part of the network matches aggregation methods introduced in 3.1, and tested in section 3.5. Here, we chose to replace not only the linear output layer but also the last transformer block.

To match dimensions and utilise the weights of the linear encoding layer of the meta-model, input weight chunks are preprocessed in the same way as in the respective training procedure, and an additional zero indicator is added to show that no chunks are masked.

We fine-tune the whole meta-model, without freezing input layers. Note that since we fine-tune on a model-zoo with models of the same architecture as in the pre-training dataset, the learnt positional encodings are also transferred successfully between these two training stages.

4.1.3 Experimental details

We perform experiments on two model zoos: our Dropped-Class-Classification zoo, and MNIST-HYP-10-RAND dataset from Schürholt et al. (2022b), and compare them to baselines set in Section 3, matching the baseline architecture:

- In experiments on the Dropped-Class-Classification zoo, we train a meta-model with 4 transformer blocks, each with 16 attention heads and an attention size of 256. In fine-tuning experiments on the same zoo, we use a meta-model with 3 transformer blocks, as the last transformer block and linear layer are replaced by concatenation and a joint linear output layer.

- In experiments on MNIST-HYP-10-RAND model zoo, we pre-train the meta-model with 5 transformer blocks, 4 of which are kept when fine-tuning for downstream tasks. One transformer block has the same dimensions as for the Dropped-Class-Classification zoo, with 16 attention heads and an attention size of 256.

The input weights are chunked layer-wise (except in section 4.4.2) with chunk size 8 and we use a masking probability of 0.2 (except in section 4.4.3).

The pre-training for both model zoos is performed with batch size 32, and no regularisation. We use a learning rate of 0.0001 for experiments on Dropped-Class-Classification zoo, and 0.0005 for experiments on MNIST-HYP-10-RAND.

In all sections except for 4.4.1, we apply permutation augmentations (introduced in section 2.1.3) in each training epoch. Because the number of possible permutations is large, it is unlikely that the meta-model ever sees the same training example twice. We show that this ability is crucial for pre-training performance.

4.2 Pre-training results

In this section, we evaluate the pre-training performance of meta-models trained on both model zoos, without delving into its impact on downstream tasks.

4.2.1 Dropped-Class-Classification Zoo

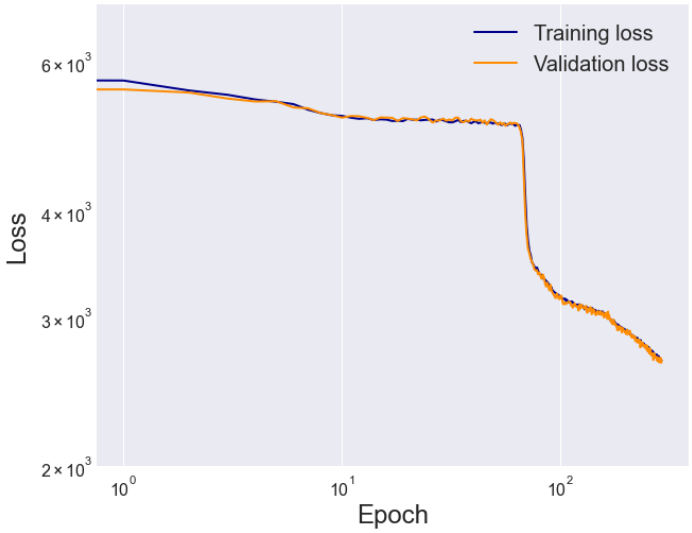
This section focuses on assessing the masked weight modelling pre-training procedure's performance for the Dropped-Class-Classification model zoo. Figure 4.2 displays the retrieved learning curves.

We calculate the R^2 score for the predicted masked chunks. After training presented in figure 4.2, the test set reconstruction R^2 score was 0.5266, signifying a meaningful prediction, better than the mean. It should be noted the training does not appear to converge at this point. However, due to the 24-hour training duration, we did not explore potential performance improvements from extended training.

Noticeably, the loss curves display two training regiments: in the first phase the meta-model learns to predict small values, close to the expected mean. After around 80 training epochs, a notable 'drop' in the loss function occurs, signifying that the meta-model predictions began to align more with the data's variability. Extensive hyperparameter sweeps have not managed to recover the same performance without the appearance of this 'drop' in the loss.

During our preliminary experiments, with larger weight chunk sizes, we noticed that the scale of larger layers is learnt more easily, while the predictions for smaller layers stay close

Fig. 4.2 MSE loss with layer-wise loss normalisation for masked weight modelling pre-training procedure on Dropped-Class-Classification model zoo



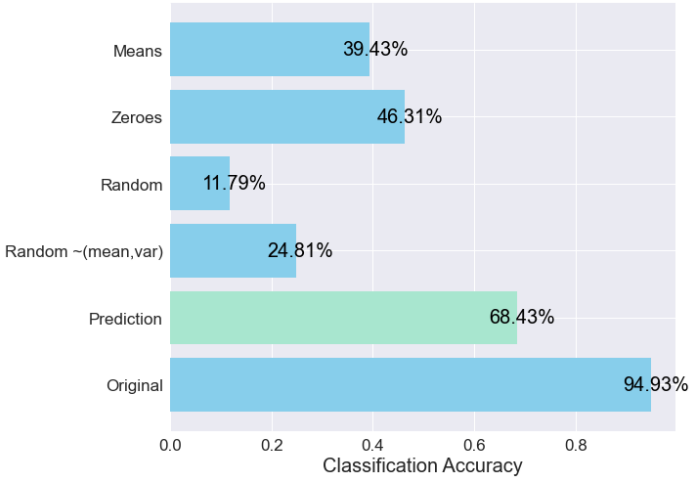
to the mean value longer. This was one of the observations that motivated the usage of small chunk sizes in further work.

Another important observation is the lack of overfitting, despite not using any regularisation methods. This suggests that the input data is diverse, and the meta-model's current capacity might be the bottleneck for the pre-training performance.

To quantify the pre-training performance in a more meaningful way, we compare the performance of the base-models after their masked weights are reconstructed using the meta-model's predictions, on its original image set (MNIST without one of the classes). In figure 4.3 we visualise the average test set classification accuracy of the reconstructed model, and contrast it with the average accuracy achieved by the original models before masking, and with the accuracy achieved by replacing masked parts of the model with some simple baselines:

- zero vectors,
- mean values of weights in that weight chunk for all base-models in the training split of the model zoo,
- randomly generated values (from a Gaussian distribution), and
- vectors sampled from a multivariate Gaussian distribution knowing the mean and the variance of the weights in the same position in the model zoo.

Fig. 4.3 Average base-model performance on its training data after masked weights are reconstructed, for Dropped-Class-Classification zoo. The results are benchmarked against different baselines: (a) substituting masked weights with zero vectors, (b) replacing with mean values from the model zoo, (c) using random Gaussian values, (d) sampling from a multivariate Gaussian based on model zoo weight metrics, (e) reconstructing masked weights using meta-model predictions, and (f) using original weights before masking.



Although the base-models reconstructed from meta-model predictions display significantly better performance than any of the networks reconstructed from relevant simple baselines, it should be noted that the performance of the original base-model achieves significantly higher values. The classification accuracy on the original image data decreases from 94% on average to only 68.4% for classification into 9 MNIST classes.

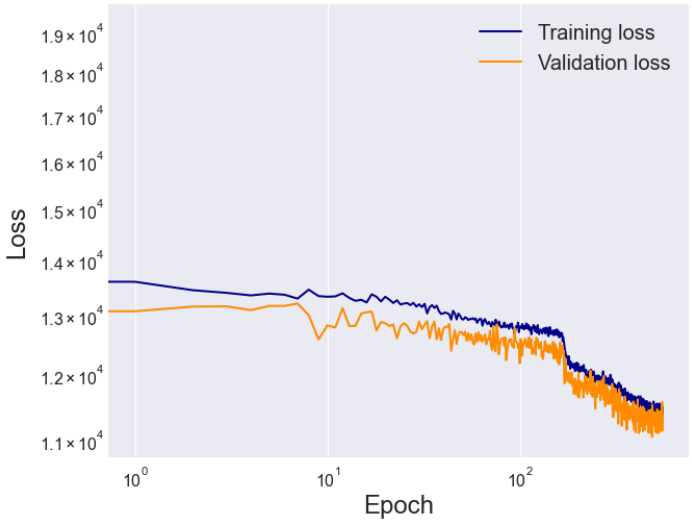
Before the 'drop' in the loss curves, the meta-model's performance closely resembled a simple mean prediction, while it drastically improves afterwards. Based on the loss curves, we suspect that it would improve further with longer training.

4.2.2 MNIST-HYP-10-RAND Model Zoo

In this section, we evaluate the pre-training performance of a meta-model trained on the MNIST-HYP-10-RAND model zoo. The loss curves are presented in figure 4.4. Although it is not clear if the meta-model has converged at this point in the training process, because this training run lasted 48h, we refrained from further exploration.

Similarly to the previous section, the meta-model first converges to a regiment akin to predicting the mean of the masked chunks, and starts modelling variance in the data only after the 'drop' in the loss function. With extensive hyperparameter tuning, the same performance could not be recovered without this 'drop'.

Fig. 4.4 MSE loss with layer-wise loss normalisation for masked weight modelling pre-training procedure on MNIST-HYP-10-RAND zoo (Schürholt et al., 2022b)



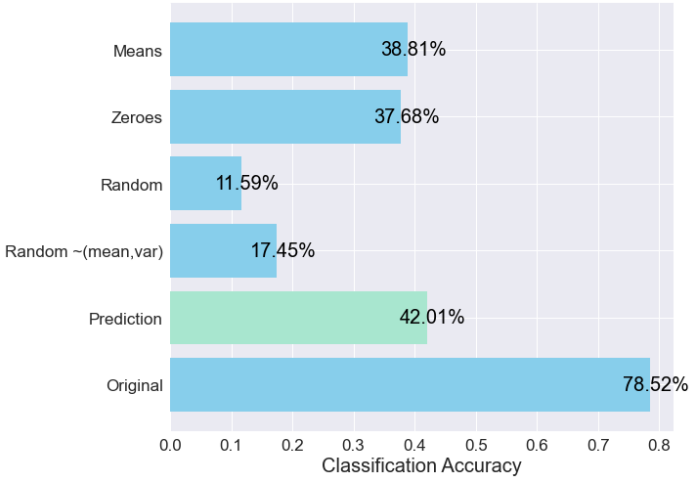
The reconstruction performance of the meta-model is significantly lower on this model zoo, with R^2 score of 0.1986 on the training, and only 0.1302 on the test set. Additionally, figure 4.5 contrasts the average classification accuracy of base-models on MNIST data post-reconstruction by the meta-model, against its original performance and that derived from simple baselines. A similar decrease in pre-training performance is observed.

We suspect that the decline in pre-training performance in comparison with the Dropped-Class-Classification zoo stems from higher variability in the data, resulting from multiple variable hyperparameters used in model zoo creation. As reported by Schürholt et al. (2022b), this leads to a more diverse distribution of neural network weights, which could render the pre-training task harder.

Another potential reason for worse pre-training performance on the MNIST-HYP-10-RAND dataset is the utilisation of poorly trained networks. As the mean accuracy of the original base-models, prior to masking, on MNIST classification is only 78.51%, we suspect that the meta-model is struggling to extract useful information from the poorly trained networks. In particular, the model zoo contains a non-negligible number of base-models with an accuracy of just above 10%, which are a prominent source of the noise. Because one of the target downstream tasks we consider is accuracy prediction, these networks were not filtered out from the zoo.

In comparison with other neural weight modelling tasks, such as representation learning explored by Schürholt et al. (2021), we believe that this kind of variability has a higher effect on the masked weight modelling task. Intuitively, in this instance, the meta-model needs to

Fig. 4.5 Average base-model performance on its training data after masked weights are reconstructed, for MNIST-HYP-1-RAND zoo (Schürholt et al., 2022b), benchmarked against different baselines: (a) substituting masked weights with zero vectors, (b) replacing with mean values from the model zoo, (c) using random Gaussian values, (d) sampling from a multivariate Gaussian based on model zoo weight metrics, (e) reconstructing masked weights using meta-model predictions, and (f) using original weights before masking.



learn to predict if the input base-model is well-trained, from partial input, and to propose a set of weights to replace the masked parts accordingly.

4.3 The effect on downstream tasks prediction

In this section, we test whether pre-trained meta-models outperform the baselines set in Chapter 3 on chosen downstream tasks. We find that the results vary depending on the task, but some improvement is evident for the majority of the tasks, especially when trained on small datasets.

4.3.1 Dropped-Class-Classification Zoo

Here, we test whether a pre-trained meta-model performs better at the dropped class classification tasks, in comparison to the baselines presented in section 3.4.4. The results are evaluated for both tasks (the easier variant with fixed MNIST class order, and the harder variant with randomly permuted class order), and with both dataset sizes as presented baselines.

We adopt the same meta-model architecture and fine-tune the pre-trained meta-model using the same hyperparameters as in direct meta-model training.

Figure 4.6 shows the validation accuracies for all experimental settings. Full loss comparison is available in appendix B.1

Fig. 4.6 A comparison of meta-models initialised from scratch and initialised from the pre-training procedure, trained on dropped class classification tasks. The experiments are done on the full dataset, and a small fine-tuning dataset containing only examples unseen in pre-training. The results are shown for: 1. easier task - with fixed class order in base-model output layer; 2. harder task - with randomly permuted class order in base-model output layer

(a) Easy task variant, full dataset

(b) Easy task variant, fine-tuning dataset

(c) Hard task variant, full dataset

(d) Hard task variant, fine-tuning dataset

We evaluate the test set performance using the checkpoint with the best validation loss and present the results in table 4.1.

The effect of pre-training is not the same for these two tasks: it diminishes performance on the easier task and improves it on the harder one. This can signalise that the pre-training method does not uniformly capture relevant features for different downstream tasks.

Table 4.1 The comparison of performances of the meta-model trained on Dropped-Class-Classification model zoo with and without the masked weight modelling pre-training procedure. The table presents test set accuracy for two versions of the task: the easy version with fixed order of classes in base-models' output layers and the harder version with randomly permuted class order. Both are presented for two subsets of the model zoo: the full dataset which includes samples used in pre-training, and a separate small fine-tuning dataset.

	Full dataset		Small fine-tuning set	
Task	Baseline	Pre-trained	Baseline	Pre-trained
Easy dropped class classification	**0.9375**	0.8447	**0.6354**	0.4583
Hard dropped class classification	0.3780	**0.4004**	0.2083	0.3020

- It was already discussed that the good performance on the easier task likely stems from the meta-model leveraging the inductive biases from the last layer of the base-model. If the pre-training procedure enhances the meta-model's generalisation capabilities, it might reduce the reliance on these biases - thus leading to a decrease in performance on a task that finds them useful.
- As the baseline meta-model training for the harder task showed overfitting, it benefits from better generalisation. In this case, pre-training acts as a form of regularisation.

These results show that pre-training efficacy is highly task-dependent, and indicate that it might have a regularisation effect.

4.3.2 MNIST-HYP-10-RAND Model Zoo

We assess whether pre-training improves meta-model performance on downstream tasks on the MNIST-HYP-10-RAND dataset, in comparison with baselines set in Section 3.3.2.

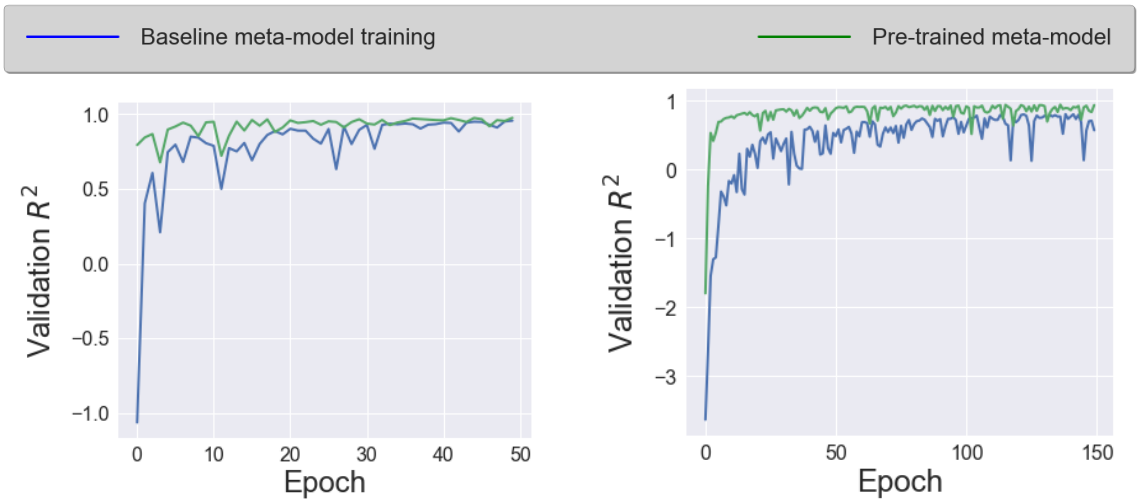
We adopt the hyperparameters optimised in Section 3.3.2, without adjusting them for fine-tuning, and run the experiments for two subsets of MNIST-HYP-10-RAND dataset: the full dataset, and a small fine-tuning dataset without models used in the pre-training stage.

Figure 4.7 presents the validation R^2 score for regression tasks, and validation accuracy for classification tasks. The full comparison of training and validation losses is available in appendix A.2.

We retrieve the meta-model weights in the epoch with the lowest validation loss and evaluate its performance on the test set. The resulting comparison of meta-model performance with and without pre-training is shown in table 4.2.

Across all experiments, the pre-training procedure improves performance, especially in early supervised training. Test set performance evaluation confirms this observation, showing

Fig. 4.7 A comparison of meta-models initialised from scratch and from the pre-training procedure, trained for downstream tasks on MNIST-HYP-10-RAND zoo (Schürholt et al., 2022b). The experiments are done on the full dataset, and a small fine-tuning dataset containing only examples unseen in pre-training. The results are presented for 5 different downstream tasks. We plot validation R^2 score for regression tasks and accuracy for classification tasks.

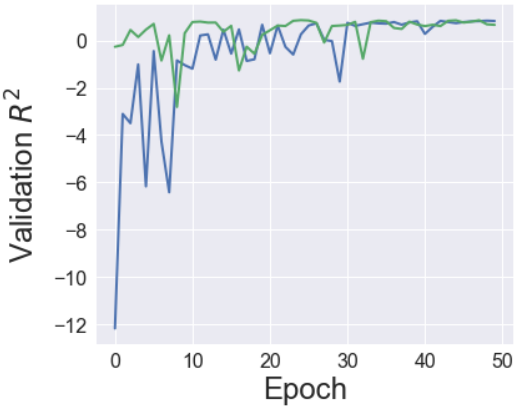


(a) Test accuracy prediction, full dataset

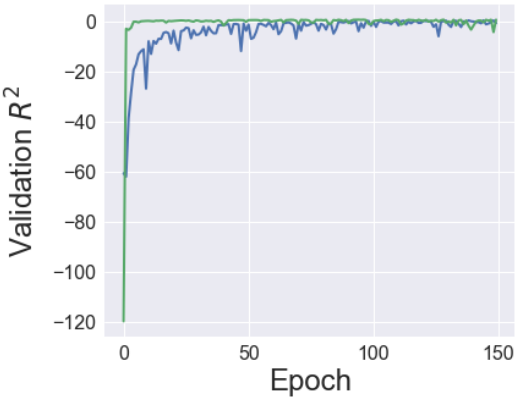
(b) Test accuracy prediction, fine-tuning dataset

Table 4.2 The comparison of performances of the meta-model trained on MNIST-HYP-10-RAND model zoo (Schürholt et al. (2022b)) with and without the masked weight modelling pre-training procedure. The table presents the test set R^2 score for regression tasks (epoch, accuracy and generalisation gap) and test set accuracy for classification tasks (activation function and initialisation method). Both are presented for two subsets of the model zoo: the full dataset, and a small fine-tuning dataset containing only examples unseen in pre-training.

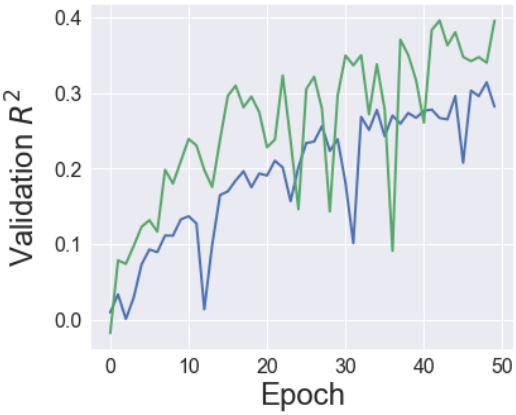
	Full dataset		Small fine-tuning set	
Task	Baseline	Pre-trained	Baseline	Pre-trained
Epoch	0.2348	**0.3574**	0.1009	**0.1041**
Accuracy	0.9597	**0.9780**	0.8809	**0.9519**
Generalisation Gap	0.7981	**0.8286**	0.3934	**0.6878**
Activation Function	0.8281	**0.8374**	0.7299	**0.7812**
Initialisation Method	0.6321	**0.7216**	**0.6562**	0.6292



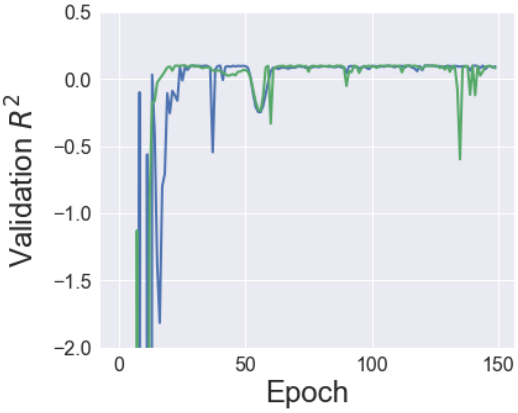
(c) Generalisation gap, full dataset



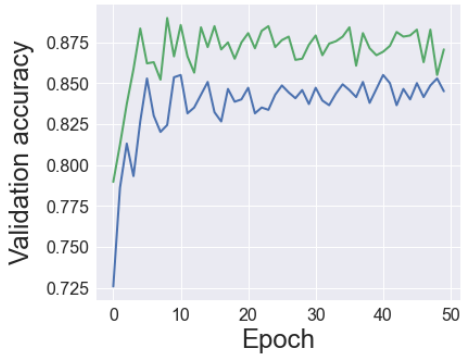
(d) Generalisation gap, fine-tuning dataset



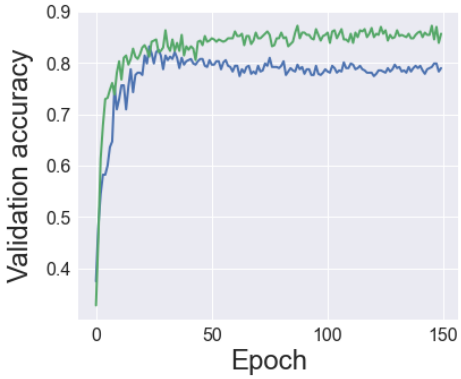
(e) Training epoch, full dataset



(f) Training epoch, fine-tuning dataset



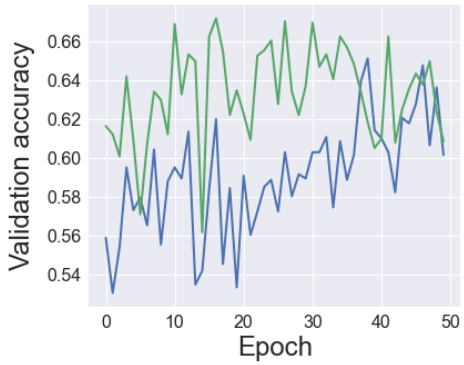
(g) Activation function, full dataset



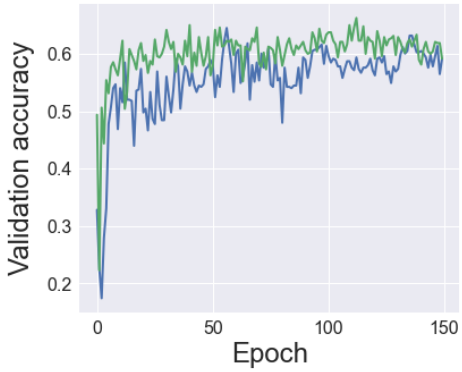
(h) Activation function, fine-tuning dataset

higher test set metrics for the pre-trained meta-model, for all tasks but the initialisation method prediction on the fine-tuning dataset.

However, although the pre-trained models display initially better prediction, the trends observed in convergence vary between tasks and dataset sizes:



(i) Initialisation method, full dataset



(j) Initialisation method, fine-tuning dataset

- **Test accuracy** and **generalisation gap** predictions display a similar behaviour. While the training on the fine-tuning datasets displays long-term benefits from pre-training, the performance on the full dataset attenuates over time. This potentially indicates that although the pre-training provides a useful basis for the downstream tasks, the biggest advantage for well-tuned downstream tasks arises from the amount of processed data.

- The results on **training epoch** prediction on the full dataset manifests the same trend of initially better performance of the pre-trained meta-model. However, since the training has not converged in the presented 50 epochs, it is not clear if the same convergence to a similar level would arise with longer training. The experiments on the fine-tuning dataset show convergence to a similar value, however, the meta-model exhibits particularly poor performance on this task.

- In **activation function** prediction the pre-trained meta-model consistently outperforms the model initialised from scratch. Recall that the meta-model initially overfits to the training data for this downstream task. This is a compelling indicator that the pre-training might have a regularisation effect on downstream performance, similar to what was noticed in section 4.3.1.

- The **initialisation method** also displays initially better performance with pre-training, but predictions for the pre-trained and meta-model initialised from scratch reach a similar validation set performance over time.

There are two main sources of limitations in these results:

- Notably, the pre-training on this dataset did not reach impressive performance, as presented in section 4.2.2. Despite this, meta-models initialised from it do display performance improvement on downstream tasks. This signalises that the metrics used

in the pre-training might not capture the broader utility or latent features the model has learnt, which are useful for downstream tasks. In this light, it is unclear how further improvement in pre-training performance might influence its efficacy for downstream tasks of interest.

- The hyper-parameters were not separately adjusted for fine-tuning, with this specific meta-model initialisation. Because of this, future experiments could further optimise given results, potentially leading to improved performance of the pre-trained meta-model.

These results indicate that while the pre-training could enhance the meta-model's performance in some downstream tasks, especially in the initial phase of the supervised training, its efficacy is very data- and task-dependent. Most notably, test accuracy and generalisation gap prediction show improvement from exposure to more data through pre-training, while the benefits fade if pre-training and fine-tuning are conducted with the same dataset. Similar to the results presented in the previous section, the results on activation function prediction indicate that pre-training acts as a form of regularisation if the original direct meta-model training overfits to the training data.

4.4 Pre-training ablations

In this section, we explore the effect of different components of the pre-training pipeline. All experiments performed on MNIST-HYP-10-RAND zoo (Schürholt et al. (2022b)). We train the meta-model with 5 transformer layers, each with 16 attention heads and attention size 256. In all training runs in this section, we use the learning rate of 0.0005, batch size 32, and no regularisation. Unless stated otherwise, we use layer-wise chunking and masking probability of 0.2.

4.4.1 Permutation Augmentations in Pre-Training

In this section, we test how the permutation augmentations, introduced in section 2.1.3, affect the effectiveness of the pre-training procedure. All other parameters of the training are fixed between the two runs, set to default values from section 4.2.2. The resulting learning curves are shown in figure 4.8.

Similarly to previous work by Schürholt et al. (2021) and Peebles et al. (2022), we observe that permutation augmentations enable generalisation. In contrast, the pre-training easily overfits to training dataset without it.

Fig. 4.8 Learning curves for masked weight modelling pre-training with and without permutation augmentations

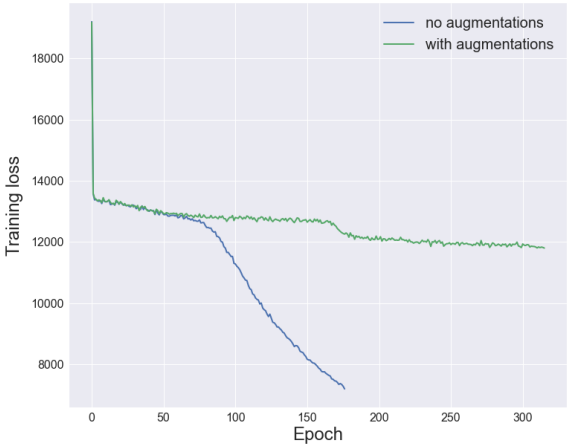


Fig. 4.9 Training set loss

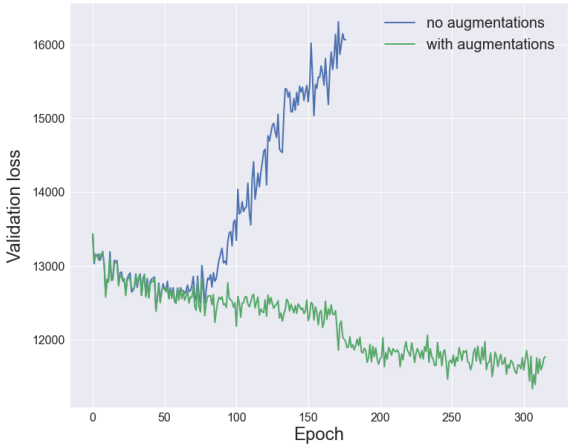


Fig. 4.10 Validation set loss

Importantly, both training procedures, with and without permutation augmentations, learn a similar mapping at first. However, this simple prediction has limited capacity to capture the variance in the data. This ability emerges after the 'drop' in the loss function, manifested in figure 4.8 around epoch 170. The training run without augmentations diverges significantly earlier.

It should be noted, however, that the presented training run was not tuned for learning without augmentations. In a less rigorous investigation, we could not find the hyperparameters that lead to the same 'drop' in the loss function without the use of augmentations.

4.4.2 Weight encodings

We investigate if adding information about base-model architecture through weight vector chunking and encoding benefits the pre-training. We compare three types of weight chunking:

- **Flattening.** All weights of the base-model are flattened into a single vector. The whole vector is divided into equal-sized weight chunks, and only the last token is padded to match this dimension. The benefit of this method is its simplicity. However, this does allow a single token to contain information about weights from multiple layers, which may yield the task more challenging for the meta-model.

- **Layer-wise chunking.** We apply layer-wise base-model weights flattening. Here, we process weights $\mathbf{W}_l$ and biases $\mathbf{b}_l$ of layer l separately. Each flattened vector is padded

to the appropriate dimension and divided into equal-sized chunks. We hypothesise that the separation of the layers makes learning to extract patterns from base-model weights through masked weight modelling an easier feat. One potential disadvantage of this approach, which becomes especially prominent with larger weight chunks, is the addition of large zero paddings, particularly in chunks that contain bias vectors.

- **Layer-wise chunking with indicators.** This method is almost the same as the second one, with the addition of one-hot encodings representing layer type. These encodings are added to each chunk. In our work, since all model zoos we use contain only linear and convolutional layers, these encodings are two-dimensional. However, this is easily extendable to other architectures (such as attention layers). Additionally, we add another binary indicator to the end of each chunk to signalize the end of a layer. Adding these encodings allows the meta-model to process different types of layers in different ways.

  Since we do not mask these indicators with the rest of the token during the pre-training procedure, this method is equivalent to using four different learnable masked tokens: two for linear layer chunks and two for convolutional layer chunks, depending if the chunk is the last one in that layer.

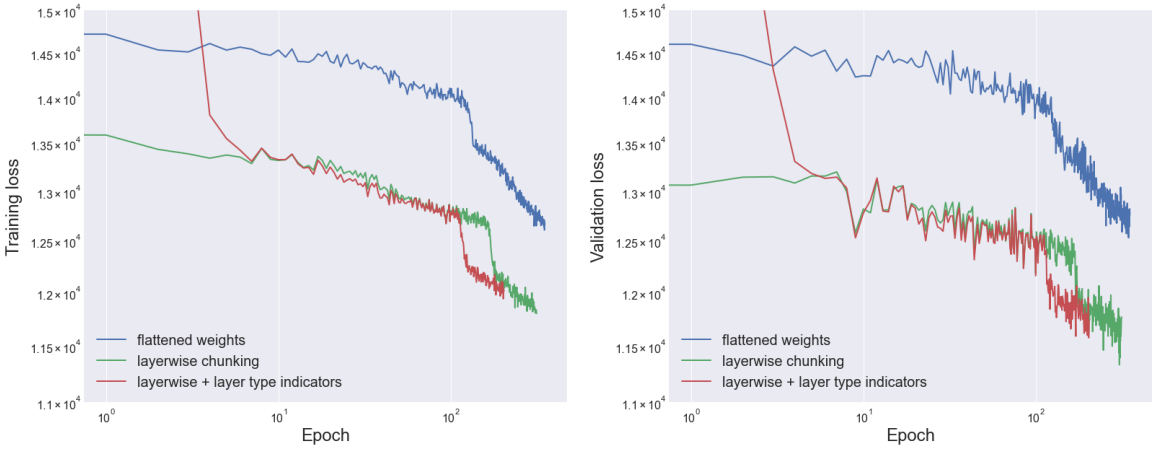
All parameters of the training run except chunking type are kept on the default values from Section 4.2.2. Input chunk size is fixed, and consists of 8 base-model weights. This dimension is low enough to not produce large zero paddings for any of the weight encoding configurations. It additionally allows the meta-model to learn more dense positional information. Intuitively, this should enhance its capabilities to utilise permutation augmentations and architecture information further.

In figure 4.11 we show learning curves for all three weight encoding methods.

In these experiments, flattening the whole set of weights together before dividing them into tokens yielded the highest loss. This could indicate that the simplistic nature of this encoding might make it harder for the meta-model to capture the trends in base-model weights.

On the other hand, the two layer-wise methods arrive at a similar validation performance, although using indicators appears to lead to reaching the 'drop' in the loss values faster. Since these training runs are constrained to a single-architecture model zoo, we could explain the similarities between them by the meta-model being able to learn the same information through learnt positional encodings, even without additional indicators. However, we expect this change to be more important in training on model zoos with multiple different architectures.

Fig. 4.11 Comparing different weight encoding configurations for pre-training on MNIST-HYP-10-RAND model zoo: 1. chunking a full flattened vector of base-model weights, 2. layer-wise weight chunking, and 3. layer-wise chunking with layer-type indicators



(a) Training set losses

(b) Validation set losses

It is important to acknowledge that the training has not yet converged fully. Because of this, it is challenging to predict which method would ultimately achieve the best results, especially considering the steeper trend in the run with the flattened weight vector.

Another limitation of the current experiment is the fact that hyperparameters were not individually tuned for each weight encoding method, but are fixed on the best values for layer-wise chunking without indicators. However, because no regularization was applied during these experiments, and yet, no overfitting was observed across all three encoding methods, and we use the same chunk size and masking probability in all experiments, we do not expect to get a significant improvement in performance from more careful hyperparameter adjustment.

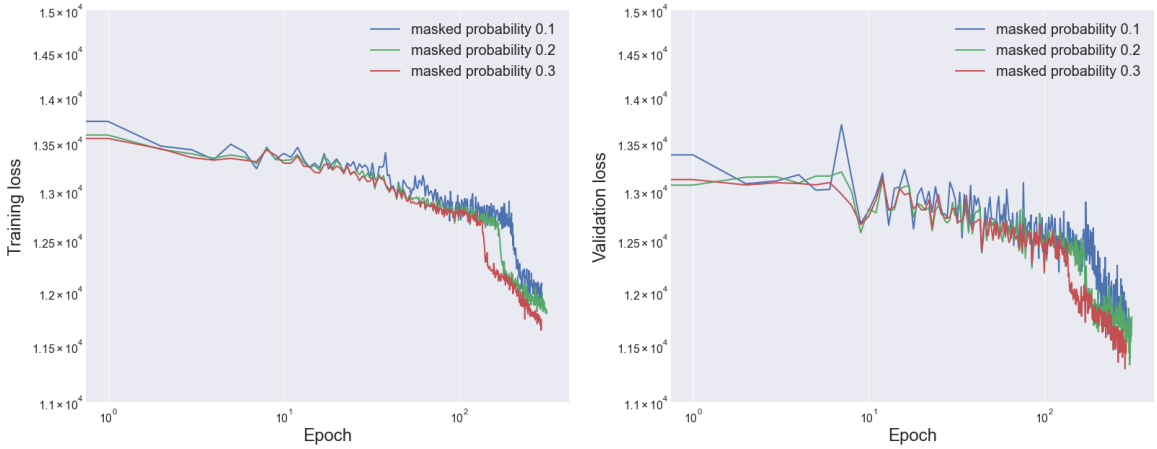
A more thorough investigation, possibly on multi-architecture model zoos, might reveal the full advantages of layer-wise encoding and indicators.

4.4.3 Masking probabilities

We test how changing masking probabilities the pre-training performance. To test this we train a meta-model using the same hyperparameter configuration as in section 4.2.2, with layer-wise weight-chunking, and permutation augmentations used in each experiment.

Figure 4.12 shows the pre-training losses for different masking probabilities. The losses are, as previously mentioned in section 4.1, normalised by the number of masked chunks, which brings them to the same scale and makes them comparable.

Fig. 4.12 MSE loss with layer-wise loss normalisation for different masking probabilities used in masked weight modelling pre-training on MNIST-HYP-10-RAND zoo (Schürholt et al., 2022b)



(a) Training set losses

(b) Validation set losses

In the initial experiments, both training and validation losses are lower for higher masking probabilities, and the losses exhibit an earlier 'drop'. A noticeable difference in loss values emerges only after the 'drop', which could be explained by all models converging into the simplified 'mean prediction regiment' first, as explained in section 4.2.

Although the reason for this behaviour cannot be inferred from these experiments alone, a potential explanation could be that higher masking is acting like an augmentation method, forcing the meta-model to learn more robust and generalisable features, similarly to erasing augmentation applied in work by Schürholt et al. (2021), which was shown to improve the representation learning performance in their work.

Further experimentation is necessary to determine if this trend continues for higher masking probabilities or plateaus. Additionally, it would be beneficial to evaluate the downstream performance of meta-models pre-trained with different masking probabilities.

Another limitation of these results that should be considered is the fact that hyperparameters were fixed during all experiments, and their interaction with masking probability was not explored.

4.5 Summary

In this chapter, we have investigated the masked weight modelling pre-training procedure on two model zoos: MNIST-HYP-10-RAND zoo, and our zoo trained for dropped class classification. We have shown that pre-training exhibits two modes, initially collapsing into

a simplistic prediction close to the mean, and later suddenly starting to generalise better to variance in the data, displaying a 'drop' in the loss function.

While the pre-training performance is much better on the Dropped-Class-Classification zoo, this is not the only indicator for downstream capabilities. We have found that the efficacy of pre-training is very task-dependent. The performance in the harder variant of the dropped class classification task improved, while the easier task exhibited a noticeable decline in performance after pre-training. We suspect that the reason for this is a significant reliance on the inductive biases in the last layer of the base-model, which decreases with higher generalisation through pre-training. Although the pre-training itself does not exhibit high performance, in most experiments on MNIST-HYP-10-RAND zoo, the meta-model initialised from pre-training demonstrates faster convergence, and improved performance on downstream tasks, especially with small downstream-task datasets. The benefits from pre-training decrease for larger amounts of fine-tuning data.

Some observations point towards a regularisation-like nature of this pre-training, with the meta-model showing a significant improvement for tasks which exhibit overfitting in the original direct-training experiments.

A preliminary investigation confirms that using permutation augmentations is crucial for achieving generalisation in pre-training, and indicates potential benefits from higher masking probabilities and layer-wise chunking.

Chapter 5

Conclusions

In this thesis, we have tested a transformer-based meta-model on a series of tasks related to predicting base-models' performance metrics, hyperparameters used in their training, and properties of their training-data, to gauge the potential the meta-models have for interpretability-related tasks. Subsequently, we have developed a self-supervised pre-training procedure titled 'masked weight modelling', a pandan to masked language modelling task applied on vectors of neural network weights. The main research objective addressed in this thesis is to investigate if this self-supervised pre-training can enhance the performance of meta-models in downstream training.

In Chapter 3 we have set the baselines against which we compare the meta-model's performance with pre-training. We have trained the meta-model to predict chosen hyperparameters, as well as performance metrics (test accuracy and generalisation gap) from input weights. Additionally, we train a new model zoo for a task that allows us to explore the potential of the meta-model to infer properties of the base-model's training data. We argue that this line of tasks is more connected to interpretability, as it requires insight into the logic behind the base-model predictions, as opposed to only understanding the parameters of the training procedure itself. The new task called *dropped class classification* denotes training the meta-model to predict which of 10 known classes was not used for training of input 9-fold classifier networks. When the ordering of classes in the last base-model layer is not fixed, this proves to be a challenging problem.

In Chapter 4, we have introduced the masked weight modelling task and evaluated its performance on two model zoos: MNIST-HYP-10-RAND model zoo (Schürholt et al., 2022b), and our model zoo focusing on dropped class classification. Achieving reasonable pre-training performance required long training times, as training displays an intriguing two-phase behaviour, where it is struggling to capture variance in the data at first. We have found the efficacy of pre-training to be task-dependent. A performance improvement is observed

for almost all tasks on MNIST-HYP-10-RAND zoo, while it hinders the performance on a simple version of dropped class classification task, with a fixed order of classes in the last base-model layer. There are indications that pre-training has a regularisation effect on tasks that originally displayed overfitting for direct meta-model training, and it is more useful for small datasets.

Additionally, we have confirmed that using permutation augmentations is crucial for achieving generalisation in pre-training, and it improves the performance of direct meta-model training, although displaying lesser effect than new, original data. Initial experiments show that masked weight modelling pre-training might benefit from higher masking probabilities, and layer-wise weight chunking.

5.1 Future Work

Although our results show the potential of the masked weight modelling pre-training procedure, a more in-depth investigation is warranted:

- Because of time constrictions, pre-training was not conducted until convergence. As the relationship between pre-training performance and its effectiveness for downstream performance is not predictable from our experiments, the potential that a longer training would bring is not completely clear.
- As preliminary results showed that the pre-training might benefit from higher masking probabilities, this research direction can be further explored, joint with the evaluation of the downstream performance of meta-models trained in such a way. It was already established that optimal masking probabilities in masked data modelling tasks vary depending on the type of data - with a probability of 0.15 used in BERT (Devlin et al., 2018), and probabilities as high as 0.75 being the optimal choice images (He et al., 2022). This result would give a useful insight into the masked weight modelling pre-training. Additionally, it should be verified that our observation holds for other model zoos.

Apart from improving the current pre-training, another research direction includes incorporating a contrastive component into the pre-training objective. This is motivated by the observations of Schürholt et al. (2021), who reported that incorporating the contrastive component in their representation learning objective led to significant benefits in downstream performance. Since our pre-training objective is different, we cannot predict if a similar conclusion would hold. We conducted some initial experiments in this direction by adding an auxiliary output from an earlier transformer block and using it to calculate an additional

NT_Xent loss (Chen et al., 2020) component. However, our initial experiments did not achieve a satisfactory weight reconstruction performance, or improve the fine-tuning results. A more rigorous hyperparameter tuning and detailed experimental setup are needed to arrive at meaningful conclusions.

Importantly, we only consider a simplified task with fixed base-model architecture. Since the ultimate motivation for this method is using meta-models for interpretability, an important step of future work would include applying a similar pre-training procedure to a multi-architecture model zoo. In this case, we expect the pre-training performance would benefit from using layer-type encodings, already investigated in section 4.4.2, or a more nuanced way of representing base-model architecture.

Additionally, it could be useful to expand the list of downstream tasks used to assess the potential of meta-models. This could include other tasks that focus on training data properties, and tasks with higher relation to interpretability, such as LessWrong (2023).

References

Balestriero, R., Ibrahim, M., Sobal, V., Morcos, A., Shekhar, S., Goldstein, T., Bordes, F., Bardes, A., Mialon, G., Tian, Y., et al. (2023). A cookbook of self-supervised learning. *arXiv preprint arXiv:2304.12210*.

Bao, H., Dong, L., Piao, S., and Wei, F. (2021). Beit: Bert pre-training of image transformers. *arXiv preprint arXiv:2106.08254*.

Biases, W. . (2021). Weights & biases: Machine learning experiment tracking. https://wandb.ai/site.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. (2020). A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR.

Conmy, A., Mavor-Parker, A. N., Lynch, A., Heimersheim, S., and Garriga-Alonso, A. (2023). Towards automated circuit discovery for mechanistic interpretability. *arXiv preprint arXiv:2304.14997*.

Csordás, R., van Steenkiste, S., and Schmidhuber, J. (2020). Are neural nets modular? inspecting functional modularity through differentiable weight masks. *arXiv preprint arXiv:2010.02066*.

Dalal, A., Sarker, M. K., Barua, A., and Hitzler, P. (2023). Explaining deep learning hidden neuron activations using concept induction. *arXiv preprint arXiv:2301.09611*.

De Luigi, L., Cardace, A., Spezialetti, R., Ramirez, P. Z., Salti, S., and Di Stefano, L. (2023). Deep learning on implicit neural representations of shapes. *arXiv preprint arXiv:2302.05438*.

DeepMind (2021a). Haiku: Sonnet for jax. https://github.com/deepmind/dm-haiku.

DeepMind (2021b). Optax: Gradient processing and optimization library for jax. https://github.com/deepmind/optax.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., et al. (2020). An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*.

Eilertsen, G., Jönsson, D., Ropinski, T., Unger, J., and Ynnerman, A. (2020). Classifying the classifier: dissecting the weight space of neural networks. *arXiv preprint arXiv:2002.05688*.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., Hatfield-Dodds, Z., Lasenby, R., Drain, D., Chen, C., et al. (2022). Toy models of superposition. *arXiv preprint arXiv:2209.10652*.

Glorot, X. and Bengio, Y. (2010). Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256. JMLR Workshop and Conference Proceedings.

Google (2021). Jax: Autograd and xla. https://github.com/google/jax.

Ha, D., Dai, A. M., and Quoc, V. L. (2016). Hypernetworks. corr abs/1609.09106 (2016). *arXiv preprint arXiv:1609.09106*.

Hadsell, R., Chopra, S., and LeCun, Y. (2006). Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE.

He, K., Chen, X., Xie, S., Li, Y., Dollár, P., and Girshick, R. (2022). Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009.

Hecht-Nielsen, R. (1990). On the algebraic structure of feedforward network weight spaces. In *Advanced Neural Computers*, pages 129–135. Elsevier.

Hendrycks, D. and Gimpel, K. (2016). Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*.

Jiang, Y., Krishnan, D., Mobahi, H., and Bengio, S. (2018). Predicting the generalization gap in deep networks with margin distributions. *arXiv preprint arXiv:1810.00113*.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Knyazev, B., Drozdzal, M., Taylor, G. W., and Romero Soriano, A. (2021). Parameter prediction for unseen deep architectures. *Advances in Neural Information Processing Systems*, 34:29433–29448.

Knyazev, B., Hwang, D., and Lacoste-Julien, S. (2023). Can we scale transformers to predict parameters of diverse imagenet models? *arXiv preprint arXiv:2303.04143*.

LeCun, Y., Cortes, C., and Burges, C. J. (2010). Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2.

LessWrong (2023). Auditing games.

Lieberum, T., Rahtz, M., Kramár, J., Irving, G., Shah, R., and Mikulik, V. (2023). Does circuit analysis interpretability scale? evidence from multiple choice capabilities in chinchilla. *arXiv preprint arXiv:2307.09458*.

McInnes, L., Healy, J., and Melville, J. (2018). Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv preprint arXiv:1802.03426*.

Meng, K., Bau, D., Andonian, A., and Belinkov, Y. (2022). Locating and editing factual knowledge in gpt. *arXiv preprint arXiv:2202.05262*.

Navon, A., Shamsian, A., Achituve, I., Fetaya, E., Chechik, G., and Maron, H. (2023). Equivariant architectures for learning in deep weight spaces. *arXiv preprint arXiv:2301.12780*.

Olah, C., Cammarata, N., Schubert, L., Goh, G., Petrov, M., and Carter, S. (2020). Zoom in: An introduction to circuits. *Distill*, 5(3):e00024–001.

Olsson, C., Elhage, N., Nanda, N., Joseph, N., DasSarma, N., Henighan, T., Mann, B., Askell, A., Bai, Y., Chen, A., et al. (2022). In-context learning and induction heads. *arXiv preprint arXiv:2209.11895*.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 165–174.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. https://pytorch.org.

Peebles, W., Radosavovic, I., Brooks, T., Efros, A. A., and Malik, J. (2022). Learning to learn with generative models of neural network checkpoints. *arXiv preprint arXiv:2209.12892*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

Räuker, T., Ho, A., Casper, S., and Hadfield-Menell, D. (2023). Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. In *2023 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, pages 464–483. IEEE.

Schroff, F., Kalenichenko, D., and Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823.

Schürholt, K., Knyazev, B., Giró-i Nieto, X., and Borth, D. (2022a). Hyper-representations for pre-training and transfer learning. *arXiv preprint arXiv:2207.10951*.

Schürholt, K., Kostadinov, D., and Borth, D. (2021). Self-supervised representation learning on neural network weights for model characteristic prediction. *Advances in Neural Information Processing Systems*, 34:16481–16493.

Schürholt, K., Taskiran, D., Knyazev, B., Giró-i Nieto, X., and Borth, D. (2022b). Model zoos: A dataset of diverse populations of neural network models. In *Thirty-Sixth Conference on Neural Information Processing Systems (NeurIPS) Track on Datasets and Benchmarks*.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. *Advances in neural information processing systems*, 33:7462–7473.

Sohn, K. (2016). Improved deep metric learning with multi-class n-pair loss objective. *Advances in neural information processing systems*, 29.

Unterthiner, T., Keysers, D., Gelly, S., Bousquet, O., and Tolstikhin, I. (2020). Predicting neural network accuracy from weights. *arXiv preprint arXiv:2002.11448*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., and Steinhardt, J. (2022). Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

Wortsman, M., Ramanujan, V., Liu, R., Kembhavi, A., Rastegari, M., Yosinski, J., and Farhadi, A. (2020). Supermasks in superposition. *Advances in Neural Information Processing Systems*, 33:15173–15184.

Xie, Z., Zhang, Z., Cao, Y., Lin, Y., Bao, J., Yao, Z., Dai, Q., and Hu, H. (2022). Simmim: A simple framework for masked image modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9653–9663.

Yak, S., Gonzalvo, J., and Mazzawi, H. (2019). Towards task and architecture-independent generalization gap predictors. *arXiv preprint arXiv:1906.01550*.

Zhmoginov, A., Sandler, M., and Vladymyrov, M. (2022). Hypertransformer: Model generation for supervised and semi-supervised few-shot learning. In *International Conference on Machine Learning*, pages 27075–27098. PMLR.

Appendix A

Supplementary Data for experiments on MNIST-HYP-10-RAND zoo

A.1 Hyperparameters For Direct Meta-Model Training

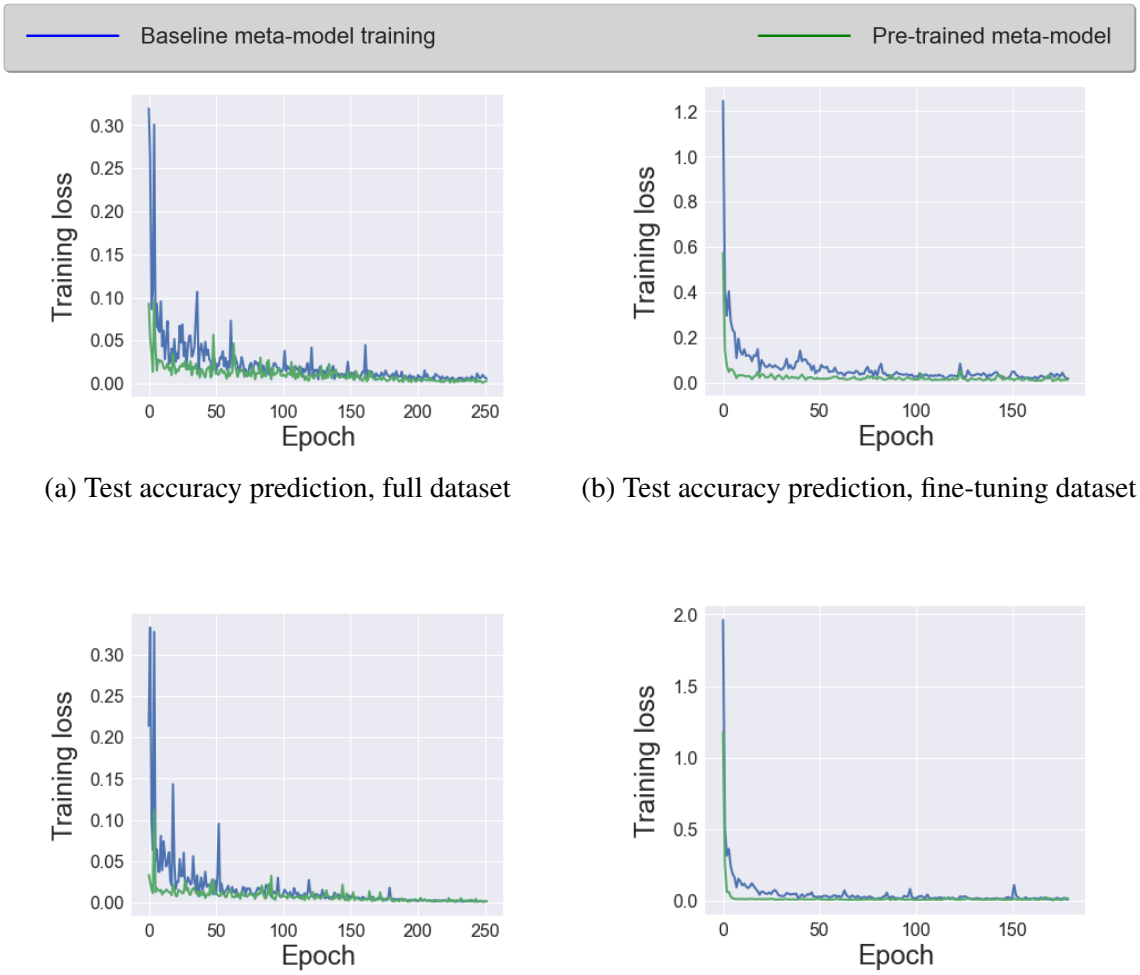
We present hyperparameters used in meta-model training on MNIST-HYP-10-RAND zoo in sections 3.3.2 and 4.3.2. Hyperparameters are tuned for the best validation performance.

Table A.1 Best hyperparameters used for direct meta-model training on MNIST-HYP-10-RAND model zoo, for different downstream tasks

Task	**lr**	**wd**	**dropout**	**bs**
Test accuracy	4.6596×10^{-5}	6.1577×10^{-4}	0.0429	32
Generalisation gap	5.2358×10^{-5}	1.1228×10^{-6}	0.0422	32
Training epoch	0.3580	0.0324	0.0410	64
Activation function	4.7481×10^{-5}	0.032	0.19	64
Initialisation method	0.0033	9.0006×10^{-6}	0.0083	32

A.2 Direct training losses

Fig. A.1 Training losses for experiments on MNIST-HYP-10-RAND zoo (Schürholt et al., 2022b), presented in section 4.3.2

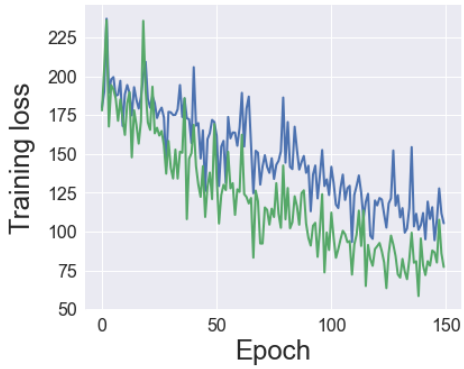


(a) Test accuracy prediction, full dataset

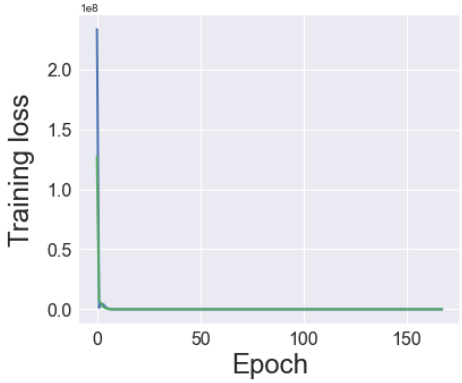
(b) Test accuracy prediction, fine-tuning dataset

(c) Generalisation gap, full dataset

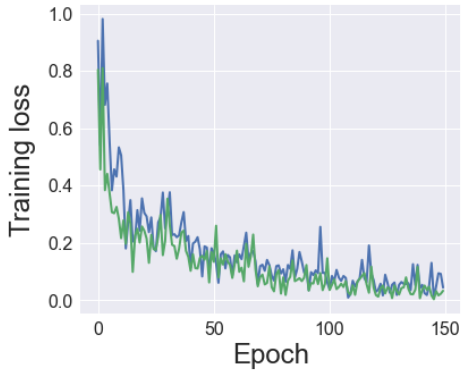
(d) Generalisation gap, fine-tuning dataset



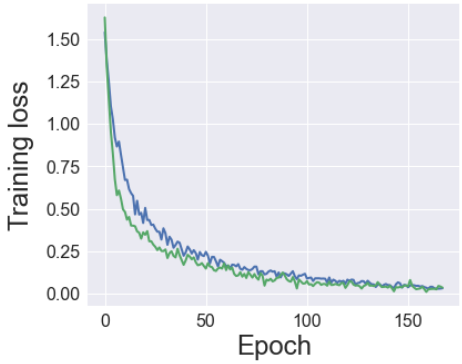
(e) Training iteration, full dataset



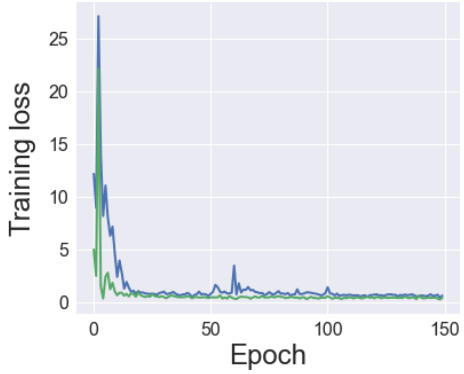
(f) Training iteration, fine-tuning dataset



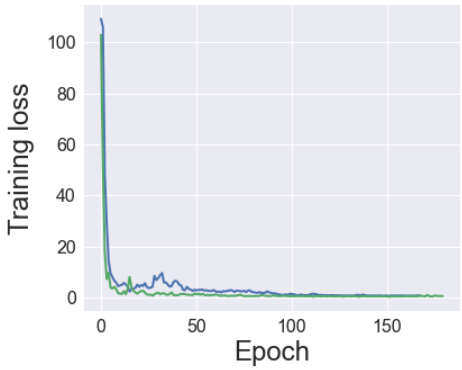
(g) Activation function, full dataset



(h) Activation function, fine-tuning dataset

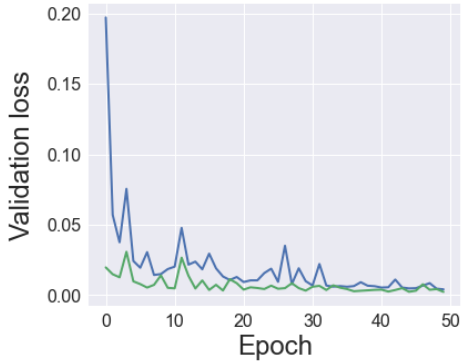


(i) Initialisation method, full dataset

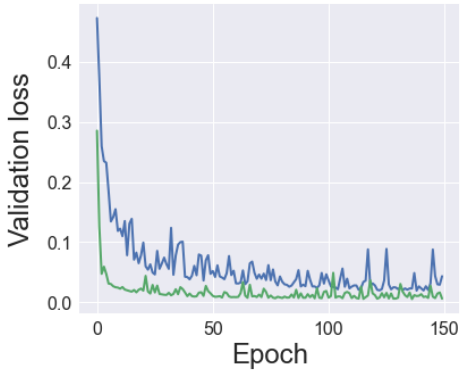


(j) Initialisation method, fine-tuning dataset

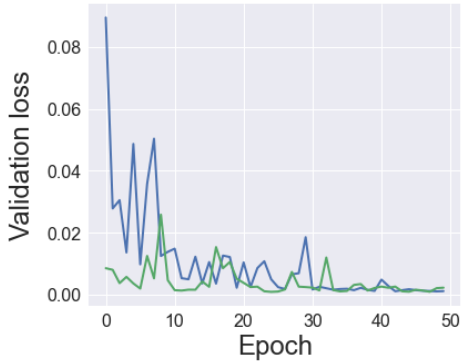
Fig. A.2 Validation losses for experiments on MNIST-HYP-10-RAND zoo (Schürholt et al., 2022b), presented in section 4.3.2



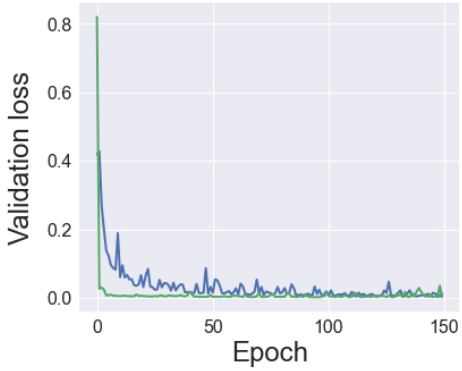
(a) Test accuracy prediction, full dataset



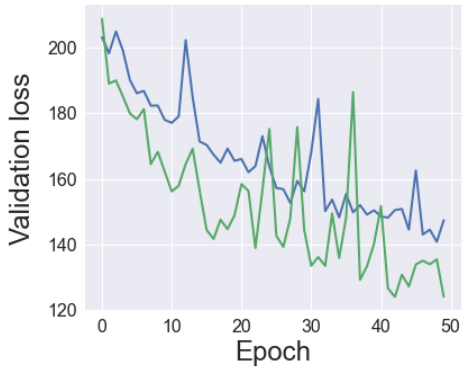
(b) Test accuracy prediction, fine-tuning dataset



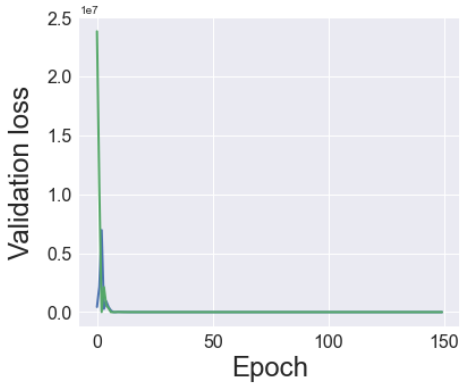
(c) Generalisation gap, full dataset



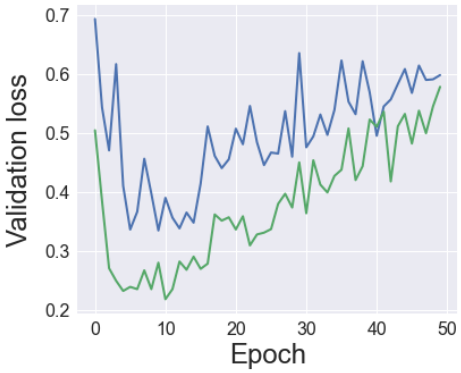
(d) Generalisation gap, fine-tuning dataset



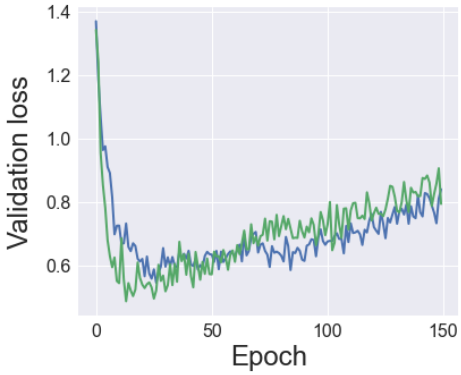
(e) Training iteration, full dataset



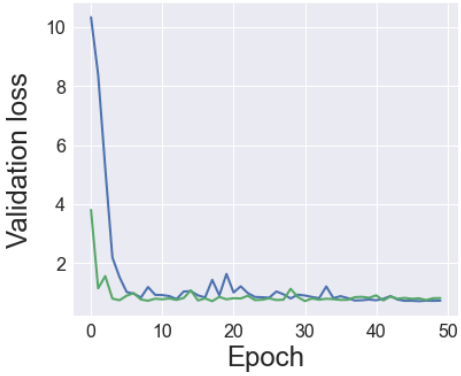
(f) Training iteration, fine-tuning dataset



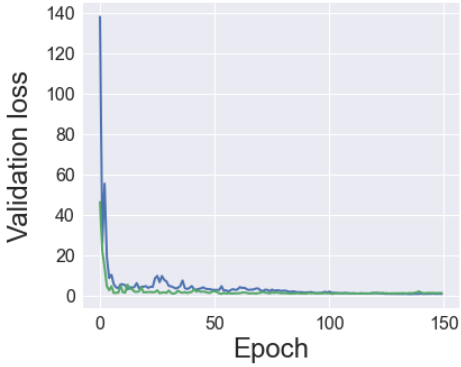
(g) Activation function, full dataset



(h) Activation function, fine-tuning dataset



(i) Initialisation method, full dataset



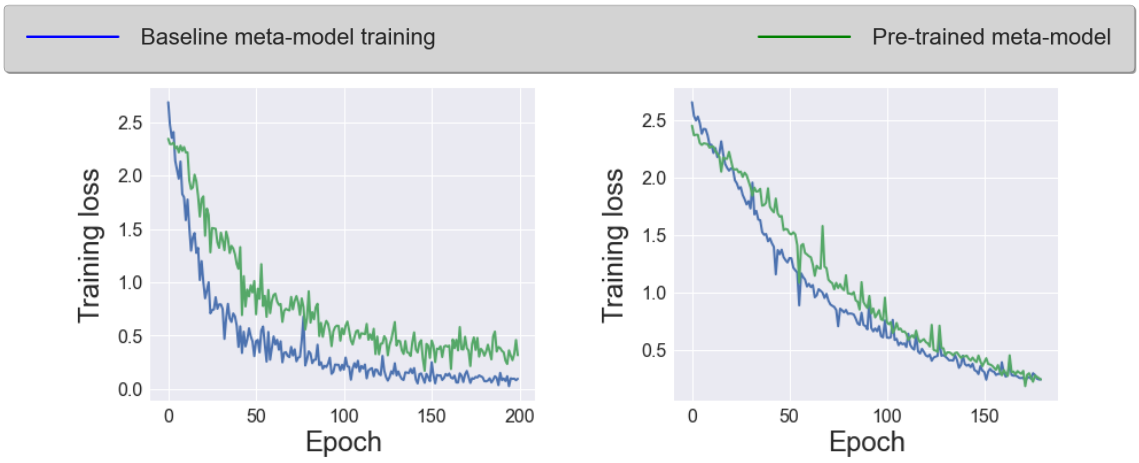
(j) Initialisation method, fine-tuning dataset

Appendix B

Supplementary Data for experiments on Dropped-Class-Classification zoo

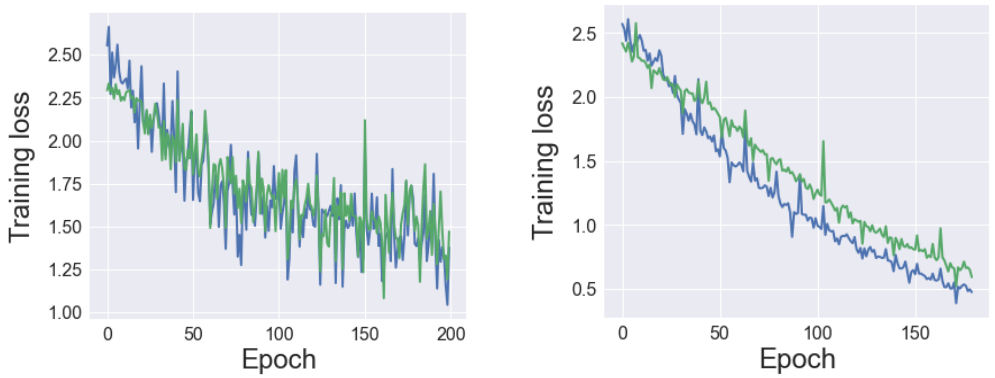
B.1 Direct training losses

Fig. B.1 Training losses for experiments on our Dropped-Class-Classification zoo, presented in section 4.3.1



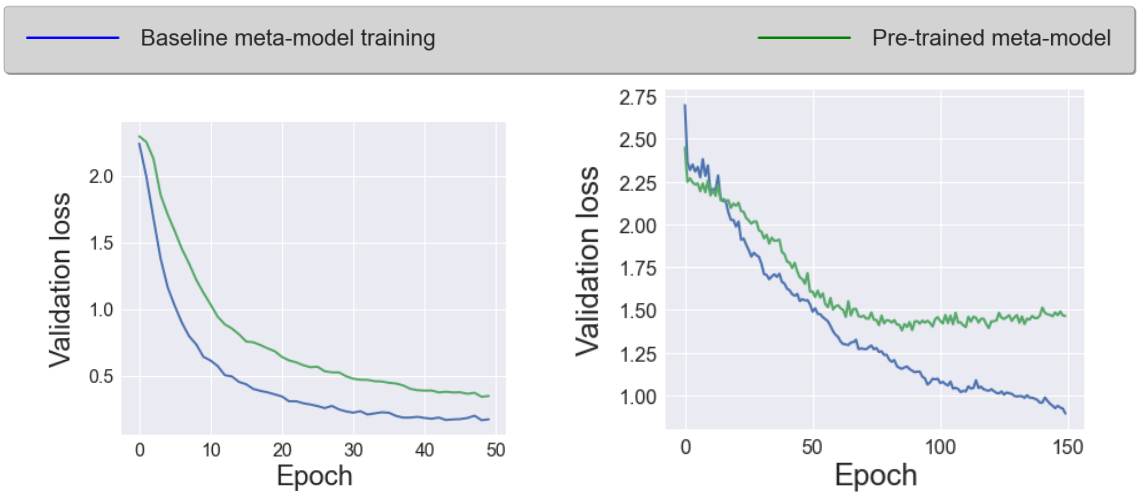
(a) Easy task variant, full dataset

(b) Easy task variant, fine-tuning dataset

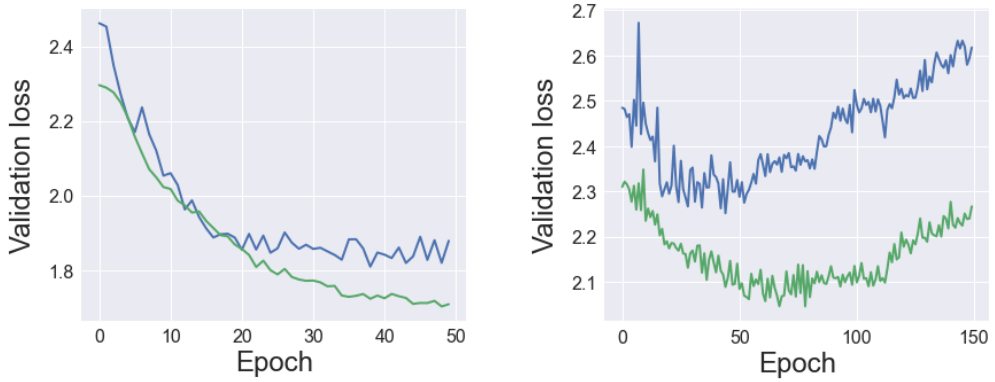


(c) Hard task variant, full dataset (d) Hard task variant, fine-tuning dataset

Fig. B.2 Training losses for experiments on our Dropped-Class-Classification zoo, presented in section 4.3.1



(a) Easy task variant, full dataset (b) Easy task variant, fine-tuning dataset



(c) Hard task variant, full dataset (d) Hard task variant, fine-tuning dataset