APPENDIX

A MIP PARAMETER INITIALIZATION

Parameter initialization strategies for regular neural networks have remained fairly stable over the past half decade. The prevalent method consists of using Glorot or He initializations that are designed to preserve the magnitude of activations during the forward pass and maintain the magnitude of gradients during the backward pass (Glorot & Bengio, 2010; He et al., 2015).

Typically, the assumptions of these initialization schemes do not hold when applied to hypernetwork settings. This has prompted the development of several specialized initialization schemes tailored for hypernetwork formulations (Beck et al., 2023; Chang et al., 2019; Knyazev et al., 2021). Crucially, these techniques require incorporating the knowledge of the primary network when performing the initialization, and are designed for categorical inputs represented as embedding vectors. The guarantees these schemes provide do not hold when using magnitude-encoded inputs such as scalars.

We propose a simple yet effective initialization scheme based on the recommendations from the neural network literature. First, the hypernetwork weights ω are initialized using common initialization methods for fully connected layers. Then, the independent parameters θ^0 are initialized taking into consideration their role in the primary network.

We illustrate our initialization rules using examples with the Kaiming He *fan-out* scheme. Using He fan-out init, the weights and biases of a neural network layer with n_{in} input neurons and n_{out} output neurons are sampled

$$W \sim \mathcal{N}\left(0, \frac{G}{\sqrt{n_{\text{out}}}}\right) \qquad b = 0, \tag{4}$$

where G is the relative *gain* of the non-linearity function $\phi(x)$. For ReLU, we have $G = 2$ whereas linear or sigmoid, we have $G = 1$.

We differentiate the following cases:

- **Intermediate Hypernetwork Layer** - We initialize a $(n_{\text{in}}, n_{\text{out}})$ layer in the hypernetwork following the scheme we just outlined, i.e., $W \sim \mathcal{N}(0, G/(\sqrt{n_{\text{out}}})$ and $b = 0$.
- **Final Hypernetwork Layer** – We consider two cases, but we do not consider biases because they are redundant with θ_0.
  1. Layer predicting a primary network weight of shape $(n_{\text{in}}, n_{\text{out}})$:
  $$W \sim \mathcal{N}\left(0, \frac{1}{\sqrt{n_{\text{in}} n_{\text{out}}}}\right) \tag{5}$$
  2. Layer predicting a primary network bias: $W = 0$
- **Independent Weights** θ_0 – For fully connected layer with (n_{in}, n_{out}) neurons, we initialize $W \sim \mathcal{N}(0, G/(\sqrt{n_{\text{out}}})$, and $b = 0$.

From this initialization, we can observe that the set of independent weights θ_0 is initialized as if they were the weights of a regular neural network. Alternatively, if the primary network corresponds to a pretrained model, the independent weights θ_0 are initialized using the pretrained values, and can be optionally frozen during training.

A.1 IMPLEMENTATION CONSIDERATIONS

Since the θ_0 weights are redundant with the bias parameters of the final hypernetwork layer, we remove bias parameters from the final hypernetwork layer. In our implementation, we use a single final layer, but we initialize its weights as if it were the multiple smaller layers, since otherwise the initialization would not follow the recommendations outlined in the previous section.

Under some hypernetwork configurations, all the primary network parameters are predicted in a single forward pass of the hypernetwork. In this scenario, we implement the parameters θ^0 as the

bias vector terms of the last layer $b^{(n)}$, which proves to be as efficient as the default formulation. This is correct because we do not have $b^{(n)}$ in our formulation, and $b^{(n)}$ is equivalent to θ^0 in the computational graph, receiving the same gradients. Hence, we initialize $b^{(n)}$ as a one-dimensional representation of the primary network parameters, subsequently reshaping it to construct θ.

B Additional Experimental Details

B.1 Datasets

MNIST. We train models on the MNIST digit classification task. We use the official MNIST database of handwritten digits. The MNIST database of handwritten digits comprises a training set of 60,000 examples, and a test set of 10,000 examples. We use the official train-test split for training data, and further divide the training split into training and validation using a stratified 80%-20% split. We use the digit labels and consider the 10-way classification problem.

OxfordFlowers-102. We use the OxfordFlowers-102 dataset, a fine-grained vision classification dataset with 8,189 examples from 102 flower categories (Nilsback & Zisserman, 2006). We utilize this dataset as it poses a non-trivial learning task that does not quickly converge, and allows us to better study learning dynamics. We use the official train-test split for training data, and further divide the training split into training and validation using a stratified 80%-20% split. We perform data augmentation by considering random square crops of between 25% and 100% of the original image area and resizing images to 256 by 256 pixels. Additionally, we perform random horizontal flips and color jitter (brightness 25%, contrast 50%, saturation 50%). For evaluation we take the central square crop of each image and resize to 256 by 256 pixels.

OASIS We use a version of the open-access OASIS Brains dataset (Hoopes et al., 2022; Marcus et al., 2007), a medical imaging dataset containing 414 MRI scans from separate individuals, comprised of skull-stripped and bias-corrected images that are resampled into an affinely-aligned, common template space. For each scan, segmentation labels for 24 brain substructures in a 2D coronal slice are available. We use 64%, 16% and 20% splits for training, validation and test.

B.2 Bayesian Neural Networks

Primary Network. For the MNIST task, we use a LeNet architecture variant that uses ReLU activations as they have become more prevalent in modern deep learning models. Moreover, we replace the first fully-connected layer with two convolutional layers of 32 and 64 features. We found this change did not impact test accuracy in non-hypernetwork models, but it lead to more stable initializations for the default hypernetworks.

For the OxfordFlowers-102 task, the primary network f features a ResNet-like architecture with five downsampling stages with (16, 32, 64, 128, 128) feature channels respectively. For experiments including normalization layers, such as BatchNorm and LayerNorm, the learnable affine parameters of the normalization layers are not predicted by the hypernetworks and are optimized like in regular neural networks via backpropagation.

Training. We train using a categorical cross entropy loss. For both optimizers we use learning rate $\eta = 3 \times 10^{-4}$. Nevertheless, we found consistent results with the ones we report using learning rates in the range $\eta = [10^{-4}, 3 \times 10^{-3}]$. We sample γ from the uniform distribution $\mathcal{U}[0, 1]$.

Evaluation. For evaluation we use top-1 accuracy on the classification labels. In order to get a more fine-grained evolution of the test accuracy, we evaluate on test set at 0.25 epoch increments during training. We report results with five model replicas with different random seeds.

B.3 HyperMorph

HyperMorph, a learning based strategy for deformable image registration learns models with different loss functions in an amortized manner. In image registration, the γ hypernetwork input controls the trade-off between the reconstruction and regularization terms of the loss.

Primary Network. For our primary network f we use a U-Net architecture (Ronneberger et al., 2015) with a convolutional encoder with five downsampling stages with two convolutional layers

per stage of 32 channels each. Similarly, the convolutional decoder is composed of four stages with two convolutional layers per stage of 32 channels each. We found that models with more convolutional filters performed no better than the described architecture.

Training. We train using the setup described in HyperMorph (Hoopes et al., 2022) using mean squared error for the reconstruction loss and total variation for the regularization of the predicted flow field. For the Adam optimizer we use $\beta_1 = 0.9$ and $\beta_2 = 0.999$ with decoupled decay Loshchilov & Hutter (2017) and $\eta = 10^{-4}$, but we found that learning rates $[10^{-4}, 3 \times 10^{-3}]$ lead to similar convergence results. For SGD with momentum, we tested learning rates $\eta = \{3 \times 10^{-2}, 10^{-2}, 3 \times 10^{-3}, 10^{-3}, 3 \times 10^{-4}, 10^{-4}, 3 \times 10^{-5}, 10^{-5}, \}$. In all cases the default hypernetwork formulation failed to meaningfully train. We train for 3000 epochs, and sample γ uniformly in the range $[0, 1]$ like in the original work.

Evaluation. Like Hoopes et al. (2022), we use segmentation labels as the main means of evaluation and use the predicted flow field to warp the segmentation label maps and measure the overlap to the ground truth using the Dice score (Dice, 1945), a popular metric for measuring segmentation quality. Dice score quantifies the overlap between two regions, with a score of 1 indicating perfect overlap and 0 indicating no overlap. For multiple segmentation labels, we compute the overall Dice coefficient as the average of Dice coefficients for each label. We report results with five model replicas with different random seeds.

B.4 SCALE-SPACE HYPERNETWORKS

We evaluate on a task where the hypernetwork input γ controls architectural properties of the primary network. We use γ to determine the amount of downsampling in the pooling layers. Instead of using pooling layers that rescale by a fixed factor of two, we replace these operations by a fractional bilinear sampling operation that rescales the input by a factor of γ.

Primary Network. For classification tasks, our primary network f features a ResNet-like architecture with five downsampling stages with (16, 32, 64, 128, 128) feature channels respectively. For experiments including normalization layers, such as BatchNorm and LayerNorm, the learnable affine parameters of the normalization layers are not predicted by the hypernetworks and are optimized like in regular neural networks via backpropagation.

For segmentation tasks, we model the primary network f using a U-Net architecture (Ronneberger et al., 2015) with a convolutional encoder with five downsampling stages with two convolutional layers per stage of 32 channels each. Similarly, the convolutional decoder is composed of four stages with two convolutional layers per stage of 32 channels each.

Training. We sample the hypernetwork input γ uniformly in the range $[0, 0.5]$ where $\gamma = 0.5$ corresponds to downsampling by 2. We train the multi-class classification task using a categorical cross-entropy loss, and train with a weight decay factor of 10^{-3}, and with label smoothing Goodfellow et al. (2016); Szegedy et al. (2016) the ground truth labels with a uniform distribution of amplitude $\epsilon = 0.1$. For the segmentation tasks we train using a cross-entropy loss and then finetune using a soft-Dice loss term, as in Ortiz et al. (2023). For both optimizers we use learning rate $\eta = 1 \times 10^{-4}$. Nevertheless, we found consistent results with the ones we report using learning rates in the range $\eta = [1 \times 10^{-4}, 3 \times 10^{-3}]$.

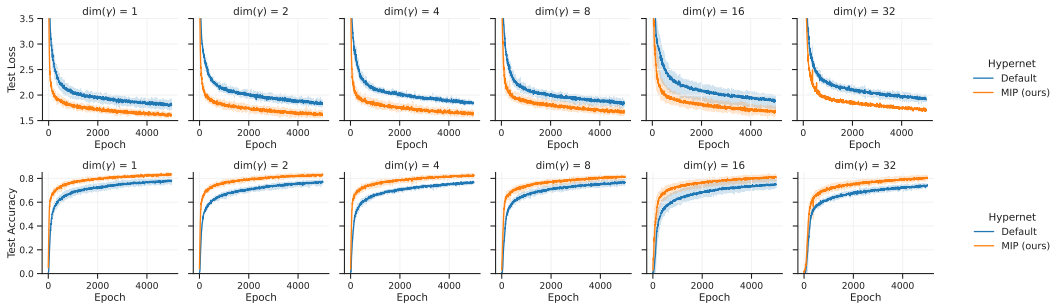
C Additional Experimental Results

C.1 Number of Input Dimensions

In this experiment, we study the effect of the number of dimensions of the input to the hypernetwork model on the hypernetwork training process, both for the default parametrization and for our MIP parametrization. We evaluate using the Bayesian Hypernetworks, since we can vary the number of dimensions of the input prior without having to define new tasks. We train models with geometrically increasing number of input dimensions, $\dim(\gamma) = 1, 2, \ldots, 32$. We apply the input encoding to each dimension independently. We study two types of input distribution: uniform $\mathcal{U}(0, 1)$ and Gaussian $\mathcal{N}(0, 1)$. For MIP, we apply a sigmoid to the Gaussian inputs to constrain them to the [0,1] range as specified by our method. We evaluate on the Bayesian hypernetworks task on the OxfordFlowers-102 dataset with a primary convolutional network optimized with Adam.

Figure 6 shows the convergence curves during training. Results indicate that the proposed MIP parametrization leads to improvements in model convergence and final model accuracy for all number of input dimensions to the hypernetwork and for both choices of input distribution. Moreover, we observe that the gap between MIP and the default parametrization does not diminish as the number of input dimensions grows.

(a) Uniform Inputs ($\Gamma_i = \mathcal{U}(0, 1)$)



(b) Gaussian Inputs ($\Gamma_i = \mathcal{N}(0, 1)$)

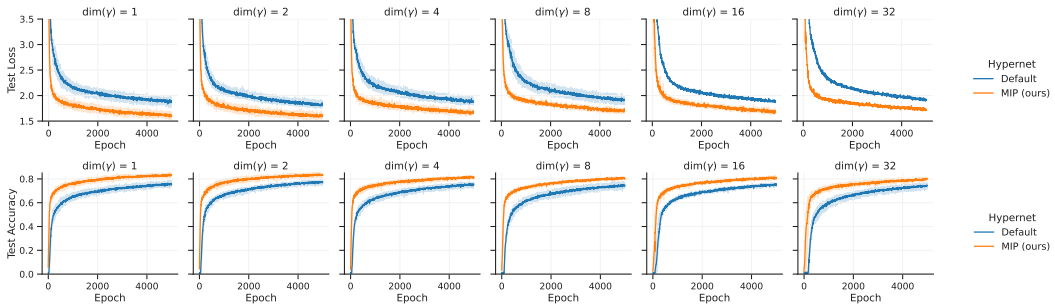


Figure 6: **Number of dimensions of hypernetwork input**. Test loss (top row) and test accuracy (bottom row) for Bayesian hypernetworks trained on the OxfordFlowers classification task for increasing number of dimensions of the hypernetwork input γ. We report results for different prior input distributions: Uniform (a) and Gaussian (b). For each setting, we train 3 independent replicas with different random initialization and report the mean (solid line) and the standard deviation (shaded region). We see significant improvements in model training convergence when the hypernetwork uses the proposed MIP parametrization.

C.2 Choice of Hypernetwork Architecture

In this experiment we test whether increasing the choice of hypernetwork architecture size has an effect on the improvements achieved by incorporating Magnitude Invariant Parametrizations (MIP). We study varying the width (the number of neurons per hidden layer) and the depth (the number of hidden layers) independently as well as jointly. For the depth, we consider networks with 3, 4 and 5 layers. For width, we consider having 16 neurons per layer, 128 neurons per layer, or having an exponentially growing number of neurons per layer (exp), following the expression $\text{Dim}(x^n) = 16 \cdot 2^n$.

We compare training networks using the default hypernetwork parametrization and MIP for the HyperMorph task. Figure 7 shows convergence curves for the evaluated settings, for several random initializations. Additionally, Figure 8 shows the distribution of final model performances for the range of inputs $\gamma \in [0, 1]$. We find that MIP models converge faster without sacrificing final model accuracy.

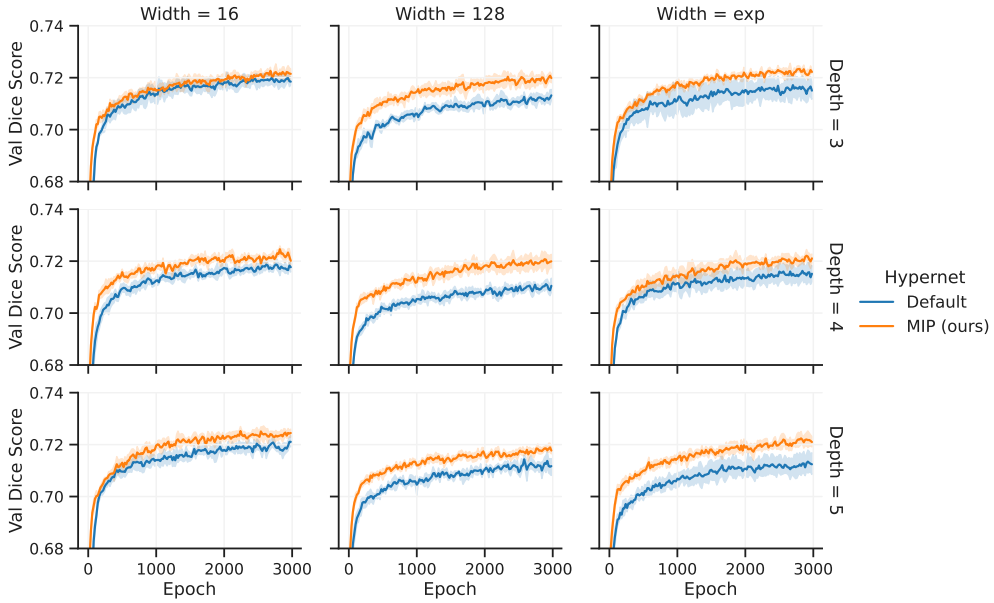


Figure 7: Model convergence for several configurations of depth and width of the hypernetwork architecture for default and MIP hypernetworks. Results are for HyperMorph on OASIS. Shaded regions measure standard deviation across hypernetwork initializations.

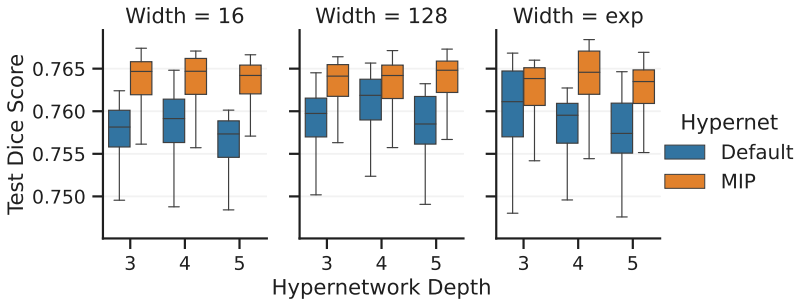


Figure 8: Test dice score for several configurations of depth and width of the hypernetwork architecture for default and MIP hypernetworks. Results are for HyperMorph on OASIS. Box-plots are reported over the range of hypernetwork inputs γ. For all hypernetwork architectures, MIP parametrizations consistently lead to more accurate models.

C.3 Choice of Nonlinear Activation Function

While our method is motivated by the training instability present in hypernetworks with (Leaky)-ReLU nonlinear activation functions, we explored applying it to other popular choices of activation functions. We consider popular activation functions GELU and SiLU (also known as Swish) that are close to the ReLU formulation, as well as the Tanh nonlinear function Hendrycks & Gimpel (2016); Ramachandran et al. (2017).

We evaluate on the Bayesian hypernetworks task on the OxfordFlowers-102 dataset with a primary convolutional network trained optimized with Adam. Figure 9 shows the convergence curves for Bayesian hypernetworks with a primary convolutional network trained on the OxfordFlowers classification task optimized with Adam. We see that MIP consistently helps for all choices of nonlinear activation function, and the improvements are similar to those of the LeakyReLU models.

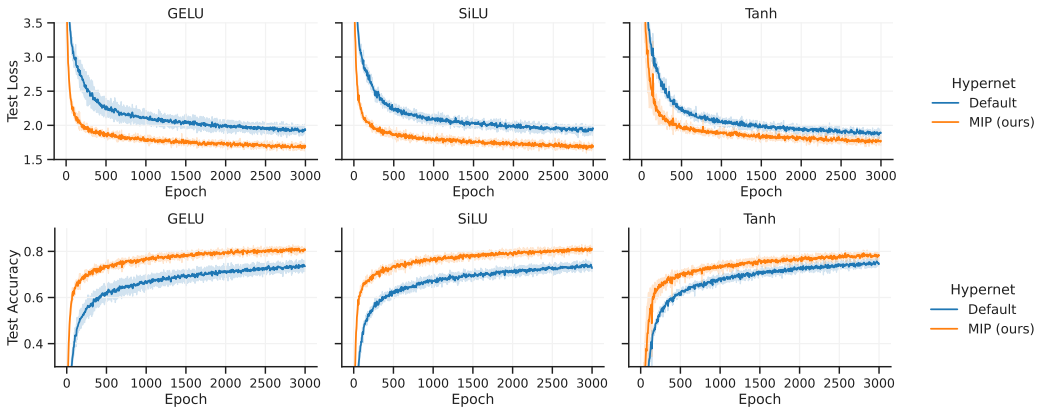


Figure 9: **MIP on alternative nonlinear activation functions** Test loss (top row) and test accuracy (bottom row) for Bayesian hypernetworks trained on the OxfordFlowers classification task for various choices of nonlinear activation function in the hypernetwork architecture: GELU, SiLU and Tanh. For each setting, we train 3 independent replicas with different random initialization and report the mean (solid line) and the standard deviation (shaded region). We see significant improvements in model training convergence when the hypernetwork uses the proposed MIP parametrization.

C.4 FINAL MODEL PERFORMANCE

Table 1: **Final model results on the test set for the considered tasks and models.** We report the average performance averaged across the range of γ inputs. We find MIP does not decrease model performance in any setting, while providing substantial improvements in several of them, especially when using the SGD optimizer. Standard deviation across random initializations is included in parentheses.

		Adam		**SGD**	
Task	**Data**	Default	MIP	Default	MIP
Bayesian NN	MNIST	98.1 (1.1)	99.1 (0.3)	99.2 (0.2)	99.0 (0.2)
	OxfordFlowers-102	78.1 (1.9)	83.2 (0.3)	1.4 (0.1)	75.4 (0.5)
HyperMorph	OASIS	71.0 (0.3)	72.1 (0.3)	54.3 (0.4)	70.5 (0.2)
Scale-Space HN	OASIS	81.4 (0.3)	84.4 (0.6)	75.3 (2.7)	78.8 (1.4)

C.5 NORMALIZATION STRATEGIES

Before developing MIP parametrizations we tested the viability of existing normalization strategies (such as Layer or Weight normalization) to deal with the identified proportionality phenomenon. While normalizing inputs and activations is a common practice in neural network training, hypernetworks present different challenges, and applying these techniques can actually be detrimental to the training process. Hypernetworks predict network parameters, and many of the assumptions behind parameter initialization and activation distribution do not easily translate between classical networks and hypernetworks.

An important distinction is that the main goal of our formulation is to ensure that the hypernetwork input has constant magnitude, not that is normalized (i.e., zero mean, unit variance). A normalized variable $z \sim \mathcal{N}(0, 1)$ does not have constant magnitude (i.e., L2 norm), over its support, so normalization techniques do not solve the identified magnitude dependency and can actually lead to undesirable formulations. To show this, let $x \in \mathbb{R}^k$ be a hypernetwork activation vector, and $\gamma \in [0, 1]$ the hypernetwork input. Then, according to the identified proportionality in Section 3.2, we know that $x = \gamma z$. Here x is the activation when the input is γ and z is a vector independent of γ. The normalization output will be

$$\text{Norm}(x) = \frac{x - \mathbb{E}[x]}{\text{Stdev}[x]} = \frac{\gamma z - \mathbb{E}[\gamma z]}{\text{Stdev}[\gamma z]} = \frac{\gamma z - \gamma \mathbb{E}[z]}{|\gamma| \text{Stdev}[z]} = \frac{z - \mathbb{E}[z]}{\text{Stdev}[z]},$$

making the output independent of the hypernetwork input γ. Following this reasoning, strategies like layer norm, instance norm or group norm in the hypernetwork will make the output of the model independent of the hypernetwork input, rendering the hypernetwork unusable for scalar inputs. For batch normalization cases it depends upon whether different hypernetwork inputs are used for each element in the minibatch. If not, the same logic applies as in the feature normalization strategies. Otherwise, the proportionality will still hold as the batch mean and standard deviation will be the same for all entries in the minibatch. Our experimental results confirm this. Hypernetworks with layer normalization fail to train in most settings. In contrast, we found consistently that training substantially improves when using our MIP formulation. See Figure 5a in the main body which shows that none of the tested normalization strategies is competitive with MIP in terms of model convergence or final model accuracy.

Batch Normalization - Applying batch normalization fails to deal with the proportionality phenomenon because it normalizes statistics that are independent of the magnitude of γ keeping the proportionality (Ioffe, 2017). In our experiments, batch normalization performed similar to the default formulation when included in either the hypernetwork or the primary network, failing to address the proportionality relationship. For instance, all of the results in Figure 6 use batch normalization layers, as recommended for ResNet-like architectures. In this case, MIP still provides a substantial improvement in terms of model convergence and training stability.

Feature Normalization - Feature normalization techniques such as layer normalization, instance normalization or group normalization do remove the proportionality phenomenon we identify (Ba et al., 2016; Ulyanov et al., 2016). However, by doing so they make the predicted weights independent of the input hyperparameter, limiting the modeling capacity of the hypernetwork architecture. Moreover, in our empirical analysis, networks with layer normalization in the hypernetwork layers failed to train entirely, with the loss diverging early in training.

Weight Normalization - We also considered techniques that decouple the gradient magnitude and direction such as weight normalization (Qiao et al., 2019). Performing weight normalization on the hypernetwork predictions effectively decouples the gradient magnitude and direction. We find that convergence is substantially lower compared to the default parametrization. Moreover, final model performance does not match the default parametrization.