HyperNetwork Designs for Improved Classification and Robust Meta-Learning

Sudarshan Babu [1] **Pedro Savarese** [1] **Michael Maire** [2]

Abstract

We propose design and training schema to enhance capabilities of hypernetworks, extending their practical impact in both classification and meta-learning settings. Our improvements stem from identifying optimization issues with current hypernetwork architectures: incorrect scale of gradients, lack of effective regularization, and insufficient momentum control. We craft solutions for each of these issues, substantially boosting hypernetwork performance. On image classification, our hypernetwork variants of standard residual networks achieve improved generalization accuracy without incurring any overhead at test time. In a Model-Agnostic Meta-Learning (MAML) setting, our hypernetworks outperform standard residual networks when training and testing tasks are sampled from different datasets.

1. Introduction

The introduction of AlexNet brought with it a rapid exploration of CNN architectures leading to improved convergence and generalization (Krizhevsky et al., 2012; Simonyan & Zisserman, 2014; He et al., 2016; Zhu et al., 2018). These improved results have been achieved due to the introduction of network components such as: rectifier activations, batch normalization, and skip connections (Glorot et al., 2011; Ioffe & Szegedy, 2015; He et al., 2016). In this work, we use hypernetworks as components towards building improved neural architectures.

Hypernetworks are meta networks that generate weights for a target network to solve a given task. The target network contains an embedding vector, which is input to the hypernetwork, the hypernetwork generates a set of weights for the target network. This coupling of the target network with the hypernetwork enables joint end to end training of both these networks. Ha et al. (2016) use a linear fully connected network as a hypernetwork to generate weights for resnets. The models explored in the paper, while offering significant compression, suffer from poor generalization and slow convergence when compared to standard resnets.

Hypernetwork-based architectures have found applications in continual learning (von Oswald et al., 2019), neural architecture search (Brock et al., 2017; Zhang et al., 2018), style transfer (Karras et al., 2019; Shen et al., 2017), natural language processing, (Suarez, 2017), pruning (Liu et al., 2019), multi-task learning (Pan et al., 2018; Klocek et al., 2019). So, improving on current hypernetwork architectures would have immediate benefits and is of independent interest.

Surprisingly, Savarese & Maire (2019) show that using a simple hypernetwork that only performs a single linear combination actually yields superior performance than Ha et al. (2016)'s model. While adding non-linearities or more parameters to hypernetwork increases its capacity, the fact that it leads to worse performance suggests that optimization issues exist. This observation is akin to the early days of deep learning where increased model complexity lead to performance degradation. Extending the analogy, we believe that enabling training of deeper hypernetworks could lead to significant gains in machine learning.

Towards enabling training of more complex hypernetworks, we identify fundamental optimization issues: incorrect scale of gradients, lack of effective regularization, and insufficient momentum control. We craft solutions to ameliorate these issues and observe that they enable successful training of convolutional hypernetworks: when generating weights for a target resnet, we improve on the performance of conventionally trained resnets on standard vision tasks. Henceforth, we refer to our hypernetwork variants of resnets as hyperresnets.

Further, we find an interesting use case for hyperresnets in the meta-learning setting. Model-Agnostic Meta-Learning (MAML) is a training routine that is widely used in tackling various problems: few-shot learning, reinforcement learning, neural machine translation, (Finn et al., 2017; Gu et al., 2018). In this work we focus mainly on the few-shot learning problem, where each class contains only a few training examples (1 to 5 examples). In particular, when there is a shift in training and testing distributions, resnets trained with MAML for few-shot learning exhibit a performance drop (Chen et al., 2019). We show that we can ameliorate this issue by simply swapping resnets with hyperresnets in the MAML routine.

To summarise, the contributions of our paper are :

- Equipped with our designed solutions, we are able to increase the complexity of hypernetworks – from linear combinations to convolutions – while improving over standard resnets.

- When training with MAML, we observe that hyperresnets are more robust to shifts in training and testing distributions in comparison to standard resnets.

2. Related Work

2.1. HyperNetworks

The idea of one neural network generating the weights of another neural network was introduced by Schmidhuber (1992). This setup was used to deal with temporal sequences, where the first network provided context dependent weights for the second network. Further Gomez & Schmidhuber (2005) provide applications for this setup.

Ha et al. (2016) use a linear two layer fully connected feed forward network to generate weights for a target resnet, with the goal of reducing parameters by sharing the hypernetwork across layers. Each layer in the main network has a low dimensional embedding, which is mapped by the hypernetwork to ambient weight space. Although this model provide significant parameter reduction, it suffers from performance degradation and slower convergence when compared to standard resnets.

Chang et al. (2019) identify that hypernetwork-generated weights are in an incorrect scale, rendering these models harder for optimization. This results in slower learning and convergence. They appropriately scale weights of the hypernetwork at initialization to alleviate these issues, resulting in lower training loss and better convergence. They report that their initialization scheme does not handle resnets as the target networks, and provide results only on hypernetworks generating weights for CNNs. However, as we will see, our hypernetwork strategies generalize to resnets as target networks, while providing faster convergence.

Savarese & Maire (2019) suggest a simple linear combination as a hypernetwork. Each layer contains a trainable vector that linearly combines a set of trainable parameters that are shared across layers. As we will see, there are applications where we are interested in maintaining an hypernetwork while re-training only the embeddings. In this scenario, the hypernetwork of Savarese & Maire (2019) is constrained to only generate weights that are linear combinations of the previously generated weights, while the hypernetwork architecture we propose can output weights of arbitrary degree of freedom.

2.2. Meta learning

Schmidhuber (1987) introduces an algorithm that learns a learning rule via a genetic algorithm. Since then, various strategies for learning to learn (meta learning) have been proposed, that use gradient based optimization to learn an update rule for the model (Hochreiter et al., 2001; Andrychowicz et al., 2016).

Proposed by Finn et al. (2017), MAML is currently one of the most prevalent meta-learning algorithms. MAML is a routine used to learn an initialization with the goal of quickly converging to a good solution for any given task. MAML consists of two nested loops: the outer loop finds a meta-initialization, while the inner loop learns to rapidly adapt the meta-initialization for the task at hand. A limitation of MAML is that it does not benefit from increasing the depth of the network that is being trained, and only uses 4 layer networks in its experiments.

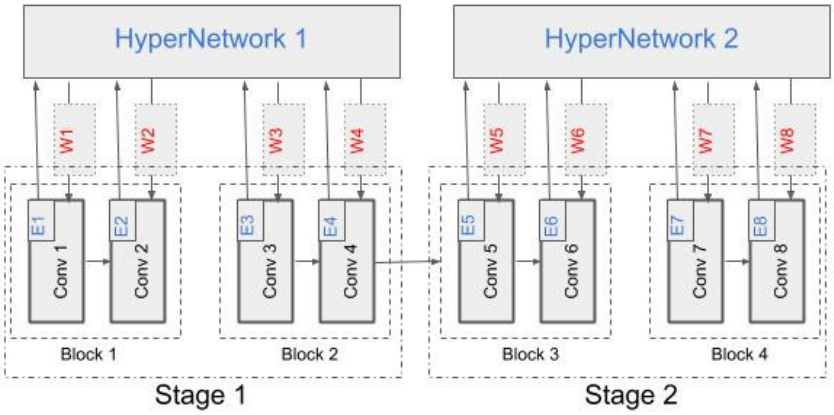
Sun et al. (2019) present a variant of MAML that employs pretraining on a large dataset to enable the use of deeper networks. Once pretrained, the network is frozen, and during training with MAML, it only learns a small set of parameters that scale and shift the frozen pretrained weights.

Rusu et al. (2018) propose a method that learns an encoder that maps training data into a small embedding vector which, in turn, is mapped to model parameters via a decoder. The encoder and decoder parameters are learnt in the outer loop, while the embedding vector is updated during the inner loop to enable adaptation for the current task.

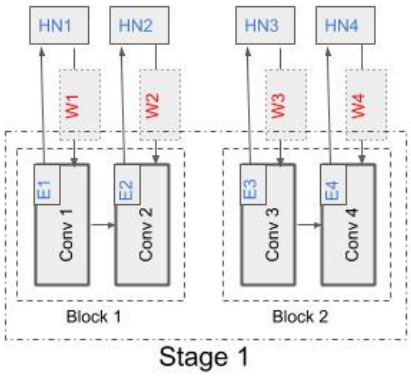
Chen et al. (2019) discover that MAML-based algorithms experience performance drop in few-shot-learning tasks when training and testing tasks are sampled from different datasets.

Raghu et al. (2019) asks a fundamental question on MAML: is its success due to its ability to find initializations that generalize to new tasks, or is it that the class-wise features learnt from the training set are incidentally suitable for classes in the test set? Their analysis leads to the conclusion that it is the latter more than the former. This discovery explains the degradation in performance of MAML models as observed by Chen et al. (2019): once the distribution of test tasks change, the features learnt from the training tasks no longer generalize, yielding a drop in performance.

The ability to adapt to new tasks, or rapid task adaptation, is essential if we aim to generalize to new tasks that are drawn from a distribution that differs from the training distribution. Later in the paper we will see that hyperresnets, when trained by MAML, are better suited for rapid task adaptation when compared to resnets. (Li et al., 2018) modifies MAML to get better task adaptation – we note that theirs is an algorithmic contribution, while ours is an architectural



(a) Variant 1: each hypernetwork is shared across all layers in a stage of the resnet. Hypernetwork 1 generates weights for all the convolutional layers in stage 1, while hypernetwork 2 does the same for stage 2.



(b) Variant 2: there is an independent hypernetwork (HN) associated to each layer in the model.

Figure 1. Illustration of the two hypernetwork sharing variants. E1 to E8 are the embeddings of each layer, and W1 to W8 are the weights generated by the respective hypernetworks. Model parameters are depicted in blue, while generated weights are in red. Skip connection between layers are omitted for clarity.

one.

3. Improving HyperNetworks

In what follows, we identify and describe shortcomings in both design and training of existing hypernetwork-based architectures.

HyperNetwork Sharing Scheme: in Ha et al. (2016), the hypernetwork is shared across all the layers of the target resnet. This sharing scheme leads to the hypernetwork receiving gradient contributions from all layers of the network, making optimization harder as gradients from different layers are likely in different scales. This calls for a need to use Adam (Kingma & Ba, 2014) to optimize the hypernetwork parameters, as it enables appropriate scaling of gradients.

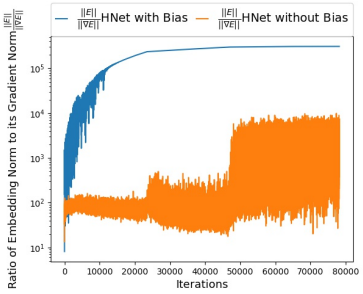
We propose two hypernetwork sharing schemes: sharing the hypernetwork across all layers in each stage of the target resnet (variant 1), and not sharing the hypernetwork at all (variant 2) i.e. each layer of the target resnet has an independent hypernetwork. Figure 1 illustrates the two variants. These hypernetwork sharing schemes have crucial implications: in variant 1, the number of layers whose error contribute to the gradient of the hypernetwork is reduced by a factor of 3; in variant 2, only a single layer contributes to the gradient of each hypernetwork. This balances the scale of the gradient updates, enabling us to use SGD, which helps our cause as it is believed to yield better generalization (Wilson et al., 2017).

HyperNetwork and Embedding Design: using a linear feed-forward network as a hypernetwork does not allow for large embeddings, as it would significantly increase the number of parameters in the hypernetwork. Ha et al. (2016) use only an embedding of size 64 to generate weights for a layer of shape $16 \times 16 \times 3 \times 3$. This essentially constraints the generated weights to have 64 degrees of freedom. On the other hand, when conventionally trained, the layer would have 2304 degrees of freedom. Here, by degrees of freedom we refer to the notion defined in Ha et al. (2016).

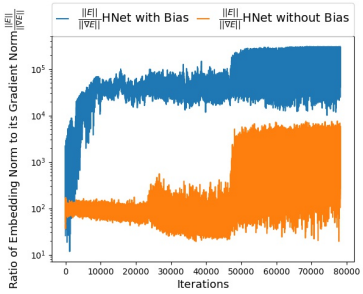
In light of this, we turn to convolutional layers as they are more parameter efficient than fully-connected counterparts. In particular, the parameter count of convolutional layers is independent of the spatial resolution of the input, enabling us to use embeddings with arbitrarily large spatial resolution. Therefore, in this work we consider a hypernetwork that is a 1-layer convolutional network.

To generate weights for a layer L_i in the target network, we associate L_i with an embedding of shape $C_{out} \times d \times k \times k$. Further, the hypernetwork connected to L_i will consist of a convolutional layer with a filter of shape $C_{in} \times d \times k \times k$. In this setup the output of the hypernetwork will have the shape $C_{out} \times C_{in} \times k \times k$. Here, C_{in} and C_{out} are the input and output number of channels of the layer L_i, and k is the size of the kernel. While d can be any integer, for most of our experiments we set $d = C_{in}$. We note that this design removes the constraint on the degrees of freedom on the generated weights.

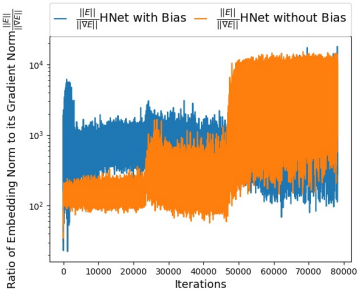
Converting HyperResNets to ResNets: our design choice doubles the number of trainable parameters with respect to standard networks. However, once the hyperresnet has been trained, we can convert it to a standard resnet to decrease the number of parameters and fasten its inference time. To do this, it suffices to collect and store the weights generated



(a) E is the embedding belonging to conv2 in stage 1 block 1 of the hyperresnet.



(b) E is the embedding belonging to conv2 in stage 1 block 1 of the hyperresnet.



(c) E is the embedding belonging to conv2 in stage 1 block 1 of the hyperresnet.

Figure 2. Plots showing ratio of embedding norm to its gradient norm for embeddings in different stages of the hyperresnet.

for each layer by its associated hypernetwork. Once this is done, maintaining the hypernetworks and the embeddings is no longer necessary. If model compression is desired, we use a scheme that is discussed in Section 5.3.

Correcting the Scale of HyperNetwork's Gradients: we investigate the slow convergence of hypernetworks by measuring the ratio between a parameter's norm and its gradient's norm. The plots of these ratios are given in Figure 2. The plots show that for a given parameter P, $\frac{||P||}{||\nabla P||}$ is large when compared to ratios reported in You et al. (2017) for convolutional layers of AlexNet – this implies that parameters don't move significantly from initialization. Moreover, we observe that this ratio is large even in the early phases of training, possibly explaining the poor convergence and generalization we observed.

We discover that this issue can be easily avoided by restricting the convolution to only perform affine operations. This can be done by simply removing the bias term from the convolutional layers. In Figure 2, we see that the ratio is decreased during the beginning of training, implying that the parameter updates are properly scaled. We further observe that the ratio increases when the learning rate is decayed: a desired behavior as we want to network parameters to change less towards the end of training.

Regularizing HyperResNets: for standard networks, weight decay on the model parameters has been the typical choice of regularization. In our experiments we find that this approach of regularization does not lead to improved generalization. Instead, we regularizing the 2-norm of the generated weights, or the output of the hypernetworks, which are analogous to the weights in the standard resnet. As we will later see, this improves classification performance of hyperresets. Further, we do not regularize the embeddings and the hypernetwork. **Momentum Control:** standard networks benefit from Stochastic Weight Averaging (SWA), where the current iterate depends on the current gradient as well as an exponential average maintained across all the iterates (Izmailov et al., 2018). The exponential average acts as a momentum term, and prevents the current iterate from changing drastically due to noisy gradients.

The predicted weights of the hyperresnets depend on both the embedding and the hypernetwork weights. Hence, the ill effects of noisy gradients could possibly get compounded. We apply SWA training to embeddings and the hypernetwork weights, and we observe that hyperresnets benefit more from SWA than standard resnets.

4. HyperNetwork Training via MAML

4.1. Background: Model Agnostic Meta Learning for Few-shot learning

The main objective of MAML is to learn a meta initialization capable of rapid task adaptation. In other words, learn an initialization such that given a new task, the network can rapidly converge to a good solution.

While training for few-shot learning, the algorithm samples a batch of tasks T_1 to T_n, from a distribution of tasks $\mathcal{P}(T)$. This batch is called a meta-batch. Each task T_i consists of two splits $T_i^{tr} = \{X_i^{tr}, Y_i^{tr}\}$ and $T_i^{te} = \{X_i^{te}, Y_i^{te}\}$. X_i^{tr} and X_i^{te} are a collection of samples, while Y_i^{tr} and Y_i^{te} are the corresponding labels. T_i^{tr} and T_i^{te} are called the train and test split of task T_i.

MAML Stage 1: For each task T_i in the meta batch, the model θ is updated to θ_i by minimizing on the loss incurred on T_i^{tr}, as shown in equation 1 . This gives us a set of models S = $\{\theta_1, \theta_2, ..., \theta_n\}$. The loss on T_i^{tr} is given by$L_{T_i^{tr}}(f_{\theta_i}) = L(f_{\theta_i}(T_i^{tr}), Y_i^{tr})$, where L is a standard loss such as cross entropy loss, and f_{θ_i} is a model with parameters θ_i.

$$\theta_i = \theta - \alpha \nabla_\theta L_{T_i^{tr}}(f_\theta) \tag{1}$$

α is the learning rate of the inner loop.

MAML Stage 2: For each $\theta_i \in$ S, we compute the corresponding loss $L_{T_i^{te}}(f_\theta) = L(f_{\theta_i}(T_i^{te}), Y_i^{te})$. Then we optimize on these losses to update θ to θ_{next} as given by equation 2.

$$\theta_{next} = \theta - \beta \nabla_\theta \sum_{i=1}^{n} L_{T_i^{te}}(f_{\theta_i}) \tag{2}$$

β is the learning rate in the outer loop. Training continues by sampling another meta batch from the same distribution of tasks $\mathcal{P}(T)$, and the above explained procedure is applied to θ_{next}. Here, θ is updated so as to minimize $\sum_{T_i^{te}} \mathcal{L}_{T_i^{te}}(f_{\theta^i})$, thereby learning an initialization that can rapidly converge to a good solution. The routine is summarized in algorithm 1

Algorithm 1 MAML Routine

Require: Collection of tasks $T_1, T_2 \cdots T_n$.
Require: Hyperparameters α and β.
Initialize network θ
while not done **do**
  Sample a batch of tasks $\mathcal{B} = \{T_1, T_2, \cdots, T_k\}$
  for all $T_i \in B$ **do**
    Compute $\mathcal{L}_{T_i^{tr}}(f_\theta)$
    Update $\theta_i = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i^{tr}}(f_\theta)$
  end for
  Update $\theta = \theta - \beta \nabla_\theta \sum_{T_i^{te}} \mathcal{L}_{T_i^{te}}(f_{\theta^i})$
end while
return θ

Algorithm 2 Training HyperResNets via MAML Routine

Require: Collection of tasks $T_1, T_2 \cdots T_n$
Require: Pre-trained network $\theta = \{\theta_E, \theta_H, \theta_C\}$. Where $\theta_E, \theta_H, \theta_C$ are embedding, hypernetwork, and classifier parameters.
Require: Hyperparameters α and β.
while not done **do**
  Sample a batch of tasks $\mathcal{B} = \{T_1, T_2, \cdots, T_k\}$
  for all $T_i \in B$ **do**
    Compute $\mathcal{L}_{T_i^{tr}}(f_\theta)$
    Update $\theta_H^i = \theta_H - \alpha \nabla_{\theta_H} \mathcal{L}_{T_i^{tr}}(f_\theta)$
    Update $\theta_C^i = \theta_C - \alpha \nabla_{\theta_C} \mathcal{L}_{T_i^{tr}}(f_\theta)$
    $\theta^i = \{\theta_E^i, \theta_H, \theta_C^i\}$
  end for
  Update $\theta_{H'} = \theta_H - \beta \nabla_{\theta_H} \sum_{T_i^{te}} \mathcal{L}_{T_i^{te}}(f_{\theta^i})$
  Update $\theta_{C'} = \theta_C - \beta \nabla_{\theta_C} \sum_{T_i^{te}} \mathcal{L}_{T_i^{te}}(f_{\theta^i})$
  Update $\theta = \{\theta_E, \theta_{H'}, \theta_{C'}\}$
end while
return θ

Model	C10	C100
Hyper-WRN-40-2 (2) W Bias	91.89	68.15
Hyper-WRN-40-2 (2) W/O Bias	**93.99**	**73.42**
Hyper-WRN-28-4 (2) W Bias	92.96	70.73
Hyper-WRN-28-4 (2) W/O Bias	**94.72**	**76.03**

Table 1. CIFAR classification accuracies. Improved generalization in hyperwideresnets of variant 2 after removing the bias in the hypernetwork.

4.2. MAML based training for HyperNetworks

Pre-training Phase: training deep networks with standard initialization via the MAML routine leads to near chance performance. The standard practice to overcome this issue is to use a model pre-trained on large data as an initialization for the meta-training phase (Sun et al., 2019). Following standard practice we pre-train hyperresnets on a large dataset as well. After pre-training, the classifier layer of the pre-trained network is replaced by a randomly initialized linear layer.

Meta-training Phase: Algorithm 2 provides a routine to train hyperresnet variant 1 via MAML. This is similar to resnet training via MAML, the essential change is that instead of training all the model parameters in both the loops we do the following: the embedding parameters are trained in the inner loop, while the hypernetwork parameters are trained in the outer loop. The classifier is updated in both the loops.

The training of hypernetworks in the outer loop results in the hypernetwork getting its weights updated by gradients from all the tasks in the meta-batch. Enabling the hypernetworks to generate weights that are useful for a collection of tasks, ultimately regularizing the model. Further, in hyperesnet variant 1 there are more embedding parameters than hypernetwork parameters. So, by updating the embeddings in the inner loop, we ensure that the network is tuned for the task at hand.

5. Experiments

5.1. Image Classification on CIFAR

CIFAR Dataset: CIFAR10 (C10) and CIFAR100 (C100) are standard image classification datasets that are frequently used to benchmark classification capabilities of neural networks (Krizhevsky et al., 2009). C10 consists of a 10-way classification problem where each class contains 5000 training and 1000 testing images. C100 is a 100-way classification problem, where each class containing 500 training and 100 testing images. The images in C10 and C100 are of shape 32×32 pixels.

Results: Table 1 shows that the removal of bias in the

Model	C10	C100
Hyper-WRN-40-2 (2) W/O Reg	93.99	73.42
Hyper-WRN-40-2 (2) W Reg	**95.20**	**76.77**
Hyper-WRN-28-4 (2) W/O Reg	94.72	76.03
Hyper-WRN-28-4 (2) W Reg	**95.9**	**79.20**

Table 2. CIFAR Classification accuracies. Improved generalization in hyperwideresnets of variant 2 after regularizing the output of the hypernetwork.

hypernetwork improves classification accuracy significantly across all architectures. Further, in Figure 3, we see that removing the bias also improves convergence when training hyperwideresnets. In Table 2, we see that performance gains are obtained by regularizing the generated weights of the hypernetwork. These improvements in classification accuracy over the vanilla hypernetwork architectures clearly underline the efficacy of our design and training strategies.

From Tables 3 and 4, we see that variant 2 of hyperresnets and hyperwideresnets outperform resnets and wideresnets on both CIFAR datasets across varying depths and widths, while variant 1 performs comparably. On C100, variant 2 improves over the baseline by around absolute 0.60% for most architectures. While the improvements on C10 are small in magnitude, they have significance when compared to the performance of their deeper counterparts. For instance, on C10, HyperResnet-18 (variant 2) outperforms ResNet34. Similarly, HyperWideResNet28-4 (variant 2) performs comparably with WideResNets100-4. This suggests that, at least on CIFAR, allocating parameters to hypernetworks is more beneficial than increasing the depth of the network.

From Table 5, on C10 we observe that HyperResNet-28-4 improves its performance gap over ResNet-28-4 from 0.25% to 0.40% with SWA training. Further, on C100, it improves from 0.60% to 0.85%.

Experimental details: we train all the models with the same standard hyperparameters as done in Zagoruyko & Komodakis (2016). We train with SGD for 200 epochs, using 0.9 as momentum and an initial learning rate of 0.1. We decay the learning rate by a factor of 5 at epochs 60, 120, and 160. For the standard wideresnets and resnets we apply weight decay of 5×10^{-4}, and for the corresponding hypernetwork variants we regularize the activations with a regularization penalty of 6.25×10^{-5}, found through minimal tuning. All results are averages of 3 runs, and are accuracies on the validation set. The wideresnet architecture is the same as Zagoruyko & Komodakis (2016), while the resnet architecture is the same as (DeVries & Taylor, 2017).

Model	C10	C100	Train Params	Test Params
WRN-40-2	94.80	76.06	2.2 M	2.2 M
Hyper-WRNet-40-2 (1)	95	75.70	2.4 M	2.2 M
Hyper-WRN-40-2 (2)	**95.20**	**76.77**	4.17 M	2.2 M
WRN-28-4	95.65	78.62	5.8 M	5.8 M
Hyper-WRN-28-4 (1)	95.65	78.49	6.6 M	5.8 M
Hyper-WRN-28-4 (2)	**95.9**	**79.20**	10.3 M	5.8 M
WRN-100-4	95.9	79.23	24.4 M	24.4 M

Table 3. CIFAR classification accuracies. Hyperwideresnet Variant 2 improves on standard wideresnets. Hyperwideresnet variant 1 provides comparable performance with wideresnets. After training, we can convert hyperwideresnets into standard wideresnets allowing these higher-accuracy versions to be deployed without any parameter overhead.

Model	C10	C100	ImageNet	Train Params	Test Params
ResNet-18	95.27	77.92	70.05	11.1 M	11.1 M
HyperResNet-18 (2)	**95.52**	**78.81**	70.01	21.4 M	11.1 M
ResNet-34	95.34	79	73.58	21.3 M	21.3 M
HyperResNet-34 (2)	**95.62**	**79.30**	**73.87**	41.6 M	21.3 M

Table 4. Identical as Table 3, but hypernetwork re-parameterizations are applied to resnets.

Model	C10	C100	Train params	Test params
WRN-28-4 + SWA	95.86	80.76	5.8 M	5.8 M
Hyper-WRN-28-4 +SWA (2)	**96.35**	**81.59**	10.3 M	5.8 M

Table 5. Classification accuracies on CIFAR. With 300 epochs of SWA training, variant 2 of HyperWideResNet-28-4 benefits more from SWA than WideResNet-28-4.

5.2. Image classification on ImageNet

ImageNet is a challenging large scale dataset that consists of 1.2M training images and 50,000 validation images sampled from 1,000 different classes.

The ImageNet results on Table 4 show minor improvements on resnet-34, while showing no improvement on resnet-18. We observe that the hyperresnet models converge quickly, for instance with 70 % training budget hyperresnet-34 achieves the similar performance to resnet-34. Similar behaviour is seen with hyperresnet-18 as well.

Experimental Details: We train our networks on ImageNet following (Savarese & Maire, 2019). We train the network with SGD for 100 epochs, using 0.9 as momentum, and an initial learning rate of 0.1. We decay the learning rate by with a factor of 10 every 30 epochs. We use a weight decay of $\times 10^{-4}$ for the resnets, while we use a regularization constant of 6.25×10^{-5} for hyperresnets. The resnet architecture used is the same as He et al. (2016).

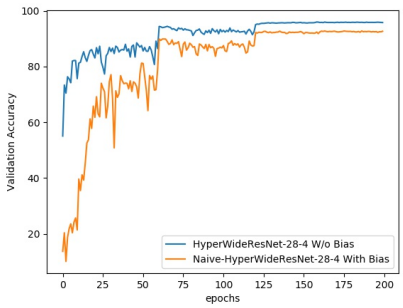


Figure 3. Validation curves for different models trained on CIFAR-10. HyperWRN-28-4 converges significantly faster and achieves better performance when the hypernetwork has its outputs regularized and when it has no bias.

Model	C10	C100	Params
WideResNet-28-4	95.65	78.62	5.8 M
WideResNet-10-4	92.32	71.49	1.22M
HyperWideResNet-28-4 (C)	94.71	74.91	1.22 M

Table 6. Classification accuracies on CIFAR. Our compression scheme offers significant parameter reduction, while outperforming baseline models with comparable number of parameters. (C) here stands for model compressed via our scheme.

5.3. Network Compression:

In variant 1, we observe that training only the hypernetworks while keeping the embeddings constant enables hyperwideresnets to achieve classification performance close to the setup where both embeddings and hypernetworks are trained. This observation offers a natural way to compress these neural models. We generate the embeddings using standard initialization schemes and save the random seed that generated it. Then, we only train the hypernetwork, and save only the hypernetwork parameters. This results in significant parameter reduction as given in Table 6. Moreover, we observe that our compressed versions of deep networks perform significantly better than smaller networks with comparable number of parameters.

5.4. Meta Learning Experiments

Our meta learning experiments are designed to measure the task adaptation capability of models when there is a distribution shift between the training and testing data. We use performance in few-shot learning as the candidate task for our Meta-Learning experiments.

Datasets:

5.4.1. MiniImageNet

It is a subset of the popular ImageNet dataset. Originally proposed by Vinyals et al. (2016), the dataset consists of 100 classes in total, with 64 train classes, 16 val classes and 20 test classes. Each class consists of 600 color images with each image being 84 x 84 in size.

Model	1-shot 5-way	5-shot 5-way
ResNet-12	57.65	**74.33**
HyperResNet-12	**58**	73

Table 7. 1-shot and 5-shot accuracies on 5-way MiniImageNet tasks, when pre-training, meta training and meta testing are done on appropriate splits of MiniImageNet. HyperResNet-12 performs comparably to ResNet-12.

5.4.2. Fewshot-CIFAR100

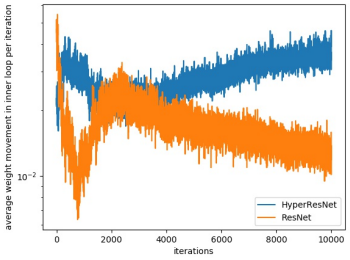
It is one of the most standard few-shot learning benchmarks. It is based off of the CIFAR100 dataset. The train/val/test splits are proposed by Oreshkin et al. (2018). For our experiments we only use the train split of Fewshot-CIFAR100, and it contains 60 classes.

Experimental Details: We evaluate our models on 1-shot and 5-shot 5-way classification problems. In 1-shot classification, we have 1 training example per class. Following Sun et al. (2019), we use batch gradient descent as the optimizer in the inner loop, with a learning rate of 0.01, and the outer loop is optimized by Adam with an initial learning rate of 0.001. We decay this learning rate by half every 1000 iterations to 0.0001. We use a meta-batch size of 4. Each task contains 15 test samples per class.

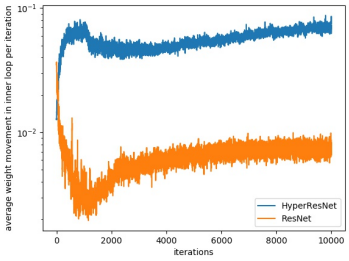
Experiments on the standard Setting: Few-shot learning models are evaluated by training on the 64-class train split of MiniImageNet and are evaluated on the 20 class split of MiniImageNet. Table 7, evaluates models in this scenario, we see that in 1-shot tasks hyperresnets do marginally better than resnets and in 5-shot tasks resnets do better than hyperresnets. In this canonical case there is little distribution shift between the train and test split as both are part of the same dataset. Hence it is not a good measure of the rapid adaptation capabilities of meta learning models.

Realistic and Robust Setting: In order to force a significant distribution shift between the training and testing tasks, we train our meta-learning algorithm by sampling few-shot tasks from the train split of few-shot CIFAR, and evaluate on the test split of MiniImageNet. We believe this a more realistic case as finding training data whose distribution is similar to the testing data is not always likely in few-shot learning scenarios as we are already suffering from lack of data.

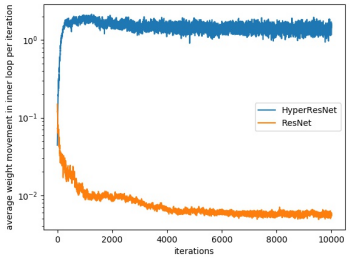
Table 8 shows that hyperwideresnets outperform wideresnets significantly on both 1-shot and 5-shot tasks. Clearly showing that hyperwideresnets are more capable of rapid adaptation as compared to wideresnets. In the following section we provide further experimental evidence suggest-



(a) Average inner loop movement for the first convolution in block 2 of stage 1



(b) Average inner loop movement for the second convolution in block 2 of stage 2



(c) Average inner loop movement for the second convolution in block 4 of stage 3

Figure 4. Blue line measures the average inner loop movement of the predicted weights by the hypernetwork in the inner loop, while the orange line measures the same for the weights of the resnet model in the inner loop. Observe that the hypernetwork predicted weights are analogous to the weight of the resnet model. We observe more movement in the hypernetwork predicted weights by orders of magnitude than the resnet weights, with increasing depth.

Model	1-shot	5-shot
WideResNet-16-4	40.75	53.2
HyperWideResNet-16-4 (1)	**43.17**	**58.66**
WideResNet-28-4	39.65	49.8
HyperWideResNet-28-4 (1)	**43.56**	**58.3**

Table 8. 1-shot and 5-shot accuracies on 5-way MiniImageNet tasks, when pre-training and meta training are done on train split of few-shot CIFAR, and meta testing is done on MiniImageNet test split. Hyperwideresnets outperform their wideresnet counterparts significantly.

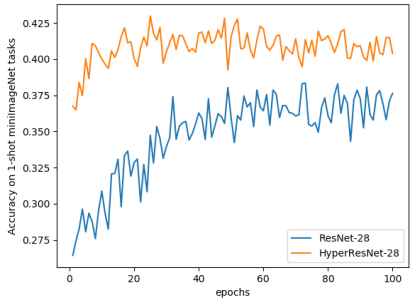


Figure 5. Validation curves on 1-shot and 5-shot tasks, with experimental setup similar to Table 8. HyperNetwork-28-4 trained with MAML shows superior convergence.

ing that hyperesnets are capable of rapid task adaptation. Further in table 8, we notice that deeper networks do poorly than their shallow counterparts, this is observed by several other works as well (Chen et al., 2019). But, we observe that the performance degradation with increased depth is more significant with the baseline wideresnets than the hyperwideresnets. Finally, in Figure 5, we observe that hyperresnets achieve superior convergence to resnets on these cross domain tasks.

Efficient Weight Movement: Task specific information is used only in the inner loop of MAML, hence the inner loop is essential for adaptation to a new task. For each meta-batch, we compute the average weight movement of resnet parameters in the inner loop in the following way. For each task in the meta batch, we compute norm of the difference between the weights at the beginning of the task and end of the task, finally we average this term across all the tasks in that meta-batch.

For hyperwideresnets, instead of calculating the difference in model parameters, we calculate the difference between the generated weight before and after the inner loop. We observe from figure 4, that the generated weights of the hyperwideresnet move more than the resnets by orders of magnitude. We believe that movement of weights in the inner loop, enables the network to get task specific information, and consequently be capable of rapid task adaptation.

6. Discussion

The early times of deep learning were marked with numerous obstacles that hindered the training of deep networks. Design and modeling choices such as rectifier activations, batch normalization, and skip connections, have since then enabled the training of powerful neural models. In a similar vein, we believe hypernetworks are a design choice towards superior neural models. This belief is based on our improved classification and robust meta learning results. However, akin to the earlier days of deep learning, complex hypernetworks cannot be properly trained yet. Here, we make useful contributions by proposing strategies for successfully training a one convolutional hypernetwork. With this work, we hope to increase interest in hypernetworks as a fruitful object of study, since arbitrarily complex hypernetworks can further advance progress towards superior neural network models.

References

Andrychowicz, M., Denil, M., Gomez, S., Hoffman, M. W., Pfau, D., Schaul, T., Shillingford, B., and De Freitas, N. Learning to learn by gradient descent by gradient descent. In *Advances in neural information processing systems*, pp. 3981–3989, 2016.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*, 2017.

Chang, O., Flokas, L., and Lipson, H. Principled weight initialization for hypernetworks. In *International Conference on Learning Representations*, 2019.

Chen, W.-Y., Liu, Y.-C., Kira, Z., Wang, Y.-C. F., and Huang, J.-B. A closer look at few-shot classification. *arXiv preprint arXiv:1904.04232*, 2019.

DeVries, T. and Taylor, G. W. Improved regularization of convolutional neural networks with cutout. *arXiv preprint arXiv:1708.04552*, 2017.

Finn, C., Abbeel, P., and Levine, S. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1126–1135. JMLR. org, 2017.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 315–323, 2011.

Gomez, F. and Schmidhuber, J. Evolving modular fast-weight networks for control. In *International Conference on Artificial Neural Networks*, pp. 383–389. Springer, 2005.

Gu, J., Wang, Y., Chen, Y., Cho, K., and Li, V. O. Meta-learning for low-resource neural machine translation. *arXiv preprint arXiv:1808.08437*, 2018.

Ha, D., Dai, A., and Le, Q. V. Hypernetworks. *arXiv preprint arXiv:1609.09106*, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Hochreiter, S., Younger, A. S., and Conwell, P. R. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, pp. 87–94. Springer, 2001.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Izmailov, P., Podoprikhin, D., Garipov, T., Vetrov, D., and Wilson, A. G. Averaging weights leads to wider optima and better generalization. *arXiv preprint arXiv:1803.05407*, 2018.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J., and Aila, T. Analyzing and improving the image quality of stylegan. *arXiv preprint arXiv:1912.04958*, 2019.

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

Klocek, S., Maziarka, Ł., Wołczyk, M., Tabor, J., Nowak, J., and Śmieja, M. Hypernetwork functional image representation. In *International Conference on Artificial Neural Networks*, pp. 496–510. Springer, 2019.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Li, D., Yang, Y., Song, Y.-Z., and Hospedales, T. M. Learning to generalize: Meta-learning for domain generalization. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

Liu, Z., Mu, H., Zhang, X., Guo, Z., Yang, X., Cheng, K.-T., and Sun, J. Metapruning: Meta learning for automatic neural network channel pruning. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3296–3305, 2019.

Oreshkin, B., López, P. R., and Lacoste, A. Tadam: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*, pp. 721–731, 2018.

Pan, Z., Liang, Y., Zhang, J., Yi, X., Yu, Y., and Zheng, Y. Hyperst-net: Hypernetworks for spatio-temporal forecasting. *arXiv preprint arXiv:1809.10889*, 2018.

Raghu, A., Raghu, M., Bengio, S., and Vinyals, O. Rapid learning or feature reuse? towards understanding the effectiveness of maml. *arXiv preprint arXiv:1909.09157*, 2019.

Rusu, A. A., Rao, D., Sygnowski, J., Vinyals, O., Pascanu, R., Osindero, S., and Hadsell, R. Meta-learning with latent embedding optimization. *arXiv preprint arXiv:1807.05960*, 2018.

Savarese, P. and Maire, M. Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*, 2019.

Schmidhuber, J. *Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook*. PhD thesis, Technische Universität München, 1987.

Schmidhuber, J. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Shen, F., Yan, S., and Zeng, G. Meta networks for neural style transfer. *arXiv preprint arXiv:1709.04111*, 2017.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Suarez, J. Language modeling with recurrent highway hypernetworks. In *Advances in neural information processing systems*, pp. 3267–3276, 2017.

Sun, Q., Liu, Y., Chua, T.-S., and Schiele, B. Meta-transfer learning for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 403–412, 2019.

Vinyals, O., Blundell, C., Lillicrap, T., Wierstra, D., et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pp. 3630–3638, 2016.

von Oswald, J., Henning, C., Sacramento, J., and Grewe, B. F. Continual learning with hypernetworks. *arXiv preprint arXiv:1906.00695*, 2019.

Wilson, A. C., Roelofs, R., Stern, M., Srebro, N., and Recht, B. The marginal value of adaptive gradient methods in machine learning. In *Advances in Neural Information Processing Systems*, pp. 4148–4158, 2017.

You, Y., Gitman, I., and Ginsburg, B. Large batch training of convolutional networks. *arXiv preprint arXiv:1708.03888*, 2017.

Zagoruyko, S. and Komodakis, N. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, C., Ren, M., and Urtasun, R. Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*, 2018.

Zhu, L., Deng, R., Maire, M., Deng, Z., Mori, G., and Tan, P. Sparsely aggregated convolutional networks. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 186–201, 2018.