Decoder Choice Network for Metalearning

Jialin Liu, Fei Chao, *Member, IEEE,* Longzhi Yang, *Senior Member, IEEE,* Chih-Min Lin, *Fellow, IEEE,* Changjing Shang, and Qiang Shen

***Abstract*—Metalearning has been widely applied for implementing few-shot learning and fast model adaptation. Particularly, existing metalearning methods have been exploited to learn the control mechanism for gradient descent processes, in an effort to facilitate gradient-based learning in gaining high speed and generalization ability. This paper presents a novel method that controls the gradient descent process of the model parameters in a neural network, by limiting the model parameters within a low-dimensional latent space. The main challenge for implementing this idea is that a decoder with many parameters may be required. To tackle this problem, the paper provides an alternative design of the decoder with a structure that shares certain weights, thereby reducing the number of required parameters. In addition, this work combines ensemble learning with the proposed approach to improve the overall learning performance. Systematic experimental studies demonstrate that the proposed approach offers results superior to the state-of-the-art in performing the Omniglot classification and miniImageNet classification tasks.**

***Index Terms*—Metalearning, latent variable, decoder, ensemble learning.**

I. Introduction

MACHINE learning has recently demonstrated near-human performance in performing challenging tasks of object recognition, image classification, digital games and scenario generations, etc. [1]–[5]. The key to the success of machine learning is the availability or obtainability of high-quality large-sized datasets. Collecting and labeling data or harvesting labeled data from the literature and historic archives require massive human efforts, and the resulting dataset can usually only be used for one specific task. Yet, humans have the ability to quickly learn new concepts and skills for novel tasks based on prior knowledge and experience. Metalearning is a machine learning technique that imitates this ability by learning parameters fine-tuned from prior datasets and pre-trained models. Consequently, metalearning may significantly extend the boundary of machine learning in terms of the applicable range of tasks and different distributions of data samples. It not only enables the 'reuse' of datasets across different tasks but also prevents overfitting on new and usually small datasets for novel tasks [6]. In this case, each novel learning task is supported by training samples (or shots) and testing samples (or queries) [7].

The most widely researched area within metalearning is few-shot learning, which requires models to predict labels of instances from unseen classes during the testing phase, with the support of only a few labeled samples from each category. Many methods have recently been proposed to implement few-shot learning tasks, which can be categorized into three types [8]: memory-based, optimization-based, and metric-based. Memory-based methods extend a memory space to store key training examples or model-related information [9], which are often achieved by applying attention models [10], [11]. Optimization-based methods learn to control the process of optimization for a given task by learning the initial parameters (e.g., [12], [13]) or the underlying optimizer (e.g., [14]–[16]). Metric-based methods focus on learning similarity metrics that maximize the similarity between members from the same class.

The purpose of this work is to design an optimization-based method that controls the optimization of a network's model parameters, by restricting the parameters within a low-dimensional space. This is inspired by the development of the approach for neural style transfer [17], which updates the input images of a deep network rather than the network's parameters. If the input images are treated as a latent variable and the output as model parameters, the model parameters can be indirectly modified by updating the latent variable. For cross-referencing simplicity, the network that implements the mapping from the latent variable to the model parameters is hereafter termed the decoder, for use in the proposed optimization-based process.

Whilst potentially powerful, the aforementioned idea can be difficult to implement for a high-dimensional model parameter space. If a fully connected network is adopted as the decoder, its complexity is usually the square of the number of model parameters. Instead of a fully connected network, a new structure is introduced in this work, named "group linear transformation (GLT)", which requires less computational time and has lower space complexity. In addition, the ideas of task-dependent adaptive metric (TADAM) [18] and latent embedding optimization (LEO) [8] are also exploited, in an effort to better capture the correlation between the model and a task, such that the model parameters reflect the need of the task. In particular, the correlation between the decoder and the task is enhanced by choosing different decoders in response to the task features with a decoder choice network (DCN). All decoders are designed to be able to share a certain part of their parameters. Thus, the complexity of the choice network

J. Liu and F. Chao are with the Department of Artificial Intelligence, School of Informatics, Xiamen University, China e-mail: (fchao@xmu.edu.cn). L. Yang is with the Department of Computer and Information Sciences, Northumbria University, UK. C.-M. Lin is with the Department of Electrical Engineering, Yuan Ze University, Taiwan. F. Chao, C. Shang, and Q. Shen are with Institute of Mathematics, Physics and Computer Science, Aberystwyth University, UK. Corresponding Author: Fei Chao

This work was supported by the National Natural Science Foundation of China (No. 61673322, 61673326, and 91746103), the Fundamental Research Funds for the Central Universities (No. 20720190142); the Key Project of National Key R & D Project (No. 2017YFC1703303), the European Union's Horizon 2020 research and innovation programme under the Marie Sklodowska-Curie grant agreement No. 663830, and the Strategic Partner Acceleration Award (80761-AU201) under the Sêr Cymru II programme, UK.

Manuscript received; revised.

is lower than that of the decoders, enabling the decoders to be task-dependent with little additional cost. Following this, to further strengthen the generalization ability of the resulting model, the learning mechanism adopts the training protocol of snapshot ensemble [19]. The resulting protocol is capable of selecting models with the highest accuracy on the validation set within a single training process.

The effectiveness of a DCN is evaluated while performing both regression and classification few-shot learning tasks. Systemmatic experimental results demonstrate that the proposed optimization-based method greatly improves the accuracy of the few-shot learning model. It achieves such positive results by reinforcing the dependency between the model and a given task, while having fewer learnable parameters (on a number of application problems). The main contribution of this work is threefold: 1) an efficient structure that enables gradient control for high-dimensional model parameters to fall within a low-dimensional space, 2) implementation of task-dependent gradient control by choosing decoders based on task features, and 3) integration of snapshot ensembles and the proposed DCN model for performance enhancement.

The rest of the paper is organized as follows. Section II outlines the underlying techniques for the development of the present work. Section III details the proposed DCN model. Section IV reports on, and discusses about, the experimental results. Section V concludes the work and points out directions for relevant future research.

II. Background

The theoretical underpinning of the proposed approach is reviewed in this section, including few-shot learning and metalearning.

A. Few-Shot Learning

In supervised learning, the training dataset contains a number of labeled data instances $\mathcal{D} = \{(\mathrm{x}^1, \mathrm{y}^1), (\mathrm{x}^2, \mathrm{y}^2), \cdots, (\mathrm{x}^K, \mathrm{y}^K)\}$, where each $(\mathrm{x}^i, \mathrm{y}^i)$, $1 \leq i \leq K$ is a data instance with given features x^i and labels y^i, and K is the number of data instances. Particularly, with labeled training data, a few-shot learning method learns how to differentiate between tasks, for which the input dataset is represented as $\mathcal{D}^{meta} = \{\mathcal{D}_i^{tr}, \mathcal{D}_i^{test}\}_i$, where $\mathcal{D}_i^{tr} = \{\mathrm{x}_{ij}^{tr}, \mathrm{y}_{ij}^{tr}\}_j$ and $\mathcal{D}_i^{test} = \{\mathrm{x}_{ij}^{test}, \mathrm{y}_{ij}^{test}\}_j$. In other words, together with given features, few-shot learning takes each dataset and regards the decision tasks as instances of training. Note that, in this work, $\mathcal{D}^{meta}$ is divided into three subsets, labeled as: $\mathcal{D}^{meta-tr}$, $\mathcal{D}^{meta-val}$ and $\mathcal{D}^{meta-test}$, standing for meta training data, meta validating data, and meta testing data, respectively.

Running few-shot learning involves three important concepts, namely: N-way, K-shot, and H-query tasks. What these concepts indicate is that there are N classification tasks, each with K training samples and H testing samples [20]. Fig. 1 shows a 5-way, 1-shot, 1-query task of miniImageNet, with each class in a task having one training instance and one testing instance. In the 5-way, 5-shot, 1-query task, there are five training instances and five testing instances in each class.

The objectives of supervised learning in general and few-shot learning in particular can be expressed by the following two optimization tasks, respectively:

$$\hat{\theta} = \arg\min_{\theta} \sum_i \mathcal{L}(f_\theta(\mathrm{x}^i), \mathrm{y}^i), \tag{1}$$

$$\hat{\theta} = \arg\min_{\theta} \sum_i \sum_j \mathcal{L}(f_{\{\mathcal{D}_i^{tr}, \theta\}}(\mathrm{x}_{ij}^{test}), \mathrm{y}_{ij}^{test}). \tag{2}$$

Eq. (1) represents a standard empirical risk minimization task of supervised learning, in which the prediction of y^i only depends on x^i and θ. However, with respect to few-shot learning, as shown in Eq. (2), the prediction of y_{ij}^{test} also depends on the training examples of $\mathcal{D}_i^{tr}$ that reflects the tasks concerned, in addition to the corresponding features x_{ij}^{test} and the parameters θ.

B. Metalearning

Methods for few-shot learning are commonly implemented by a certain metalearning mechanism, which enables the learn-to-learn ability. Generally speaking, metalearning involves two hierarchical learning processes: 1) low-level learning, which is usually termed the "inner loop", learns how to handle general tasks; and 2) high-level learning, which is often referred to as the "outer loop", improves the performance of the low-level learning process.

Deep learning is in turn, typically employed to implement a metalearning process, although other machine learning methods, such as Bayesian learning [21], may also be applied. Indeed, most of the metalearning techniques use the gradient descent method within the "outer loop"; with the gradient of the "outer loop" termed the metagradient. As for the "inner loop" different learning methods may be applied, including the metalearning approaches that can be implemented in either of the following three types of method: 1) memory-based, 2) optimization-based, and 3) metric-based [8].

Metric-based methods can be artificially viewed as applying the K-nearest neighbor (KNN) method or one of its variations to optimize a feature embedding space within the "outer loop". This minimizes the similarity metrics between instances from the same class and maximizes those from different classes. Such a method predicts testing sample labels based on the similarity metrics in the embedding space within the "inner loop". To support this type of metalearning learning, a number of similarity metric approaches have been suggested for use in metalearning, such as cosine distance, squared Euclidean distance, and even relationships learned by a neural network [20], [22], [23].

Optimization-based methods adopt deep learning methods within the "inner loop", and during the "outer loop", to learn the hyperparameters, such as parameter initialization, learning rate, and gradient direction, of the model. Among these methods, model agnostic meta learning (MAML) is the most typical [12]; it tries to learn the initialization of the model parameters. In addition, those methods reported in [14] and [15] attempt to learn the learning rate of the "inner loop".

Memory-based methods memorize and search for key training examples [10] or model-related information within

Fig. 1. Example of few-shot learning data. These are instances from 5-way, 1-shot, 1-query metadata. Each learning task contains ten images from different classes, and each class has one training example and one testing example. Note that images marked with the same number in the upper left corner of each line are from the same image class.

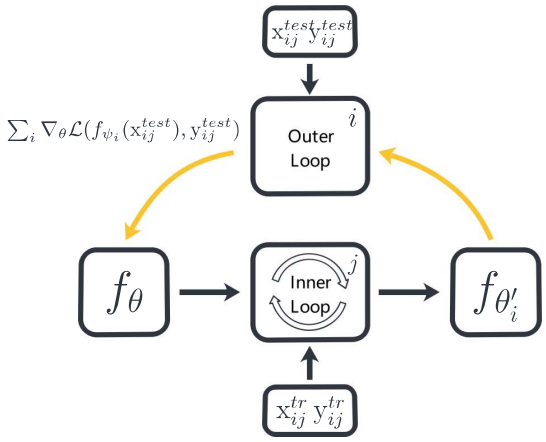

Fig. 2. A typical metalearning model, where θ is the parameters updated within the "outer loop", θ_i' represents the parameters obtained by the training on task i during the "inner loop", and $\{\mathrm{x}_{ij}^{tr}, \mathrm{y}_{ij}^{tr}\}_j$ and $\{\mathrm{x}_{ij}^{test}, \mathrm{y}_{ij}^{test}\}_j$ are training samples and testing samples regarding task i, respectively.

TABLE I
Notations and descriptions. # means "the number of". * means "the parameters of".

Notation	Description	Notation	Description
$\mathcal{D}_i^{tr}$	training dataset i	c_s	weight of s decoder
$\mathcal{D}_i^{test}$	test dataset i	α	"inner loop" learning rate
x_{ij}^{tr}	training feature	η	"outer loop" learning rate
y_{ij}^{tr}	training label	M	# "inner loop" iterations
x_{ij}^{test}	test feature	z	latent variable
y_{ij}^{test}	test label	ϵ	* batch normalization
$\mathcal{T}_i$	task i	θ_i	parameter for task i
g_{ϕ_c}	choice network	$\hat{\theta}_i$	θ_i without normalization
$g_{\phi_d}^s$	decoder s	ξ	mean of $\{\hat{\theta}_i\}_i$
S	# decoders	σ^2	variance of $\{\hat{\theta}_i\}_i$
f_{θ_i}	network with θ_i	h	hidden variable matrix
W_n	weight matrix n	λ	* softshrink
ω	* ELU	F_h	# groups of h
F_g	# groups of z	F_x	# data features
F_f	# firing strengths	μ_A	firing strengths A
γ_n	state variable n	μ_B	firing strengths B
ϕ_{fc}	* last layer		

the "inner loop". Here, model-related information may include any information that is related to the model of the "inner loop", such as the network weights [11] or the activation values [24] within different layers. These methods typically work by extending an external memory, and by reading and writing the memory with attention models.

In summary, a general metalearning model is illustrated in Fig. 2. If a parametric learning method is utilized during the "inner loop", the model will obtain the parameters θ_i' by training on the data of task i; otherwise, if a nonparametric learning method is used, θ_i' is just equal to $\{\theta, \{\mathrm{x}_{ij}^{tr}, \mathrm{y}_{ij}^{tr}\}_j\}$. During the "inner loop", θ is then updated to θ_i', thereby being capable of differentiating between θ and θ_i'. Last, within the "outer loop", θ is updated by the gradient descent method.

III. Proposed Metalearning Model

The proposed approach offers an optimization-based method that controls the gradient descent process within the "inner loop", by limiting the model parameters in a low-dimensional latent space (as shown in Fig. 2). The latent variable in the latent space is decoded by a nonlinear decoder to obtain the parameters of the prediction model. The GLT structure is devised to implement decoder networks. Different decoders can be chosen in response to a given task. Therefore, the low-dimensional latent space used in the decoder choice network is also dependent on the task. For presentational simplicity, Table I lists all notations and summarized descriptions.

A. Decoder Choice Network

Fig. 3 illustrates the structure of DCN, consisting of three parts: a choice network g_{ϕ_c}, a number of decoders $\{g_{\phi_d}^1, g_{\phi_d}^2, \cdots, g_{\phi_d}^S\}$, and a latent variable z. At the start of the "inner loop", the choice network receives the task features and produces the choice of the decoder. To make the decoder choice differentiable, the idea of neural Turing

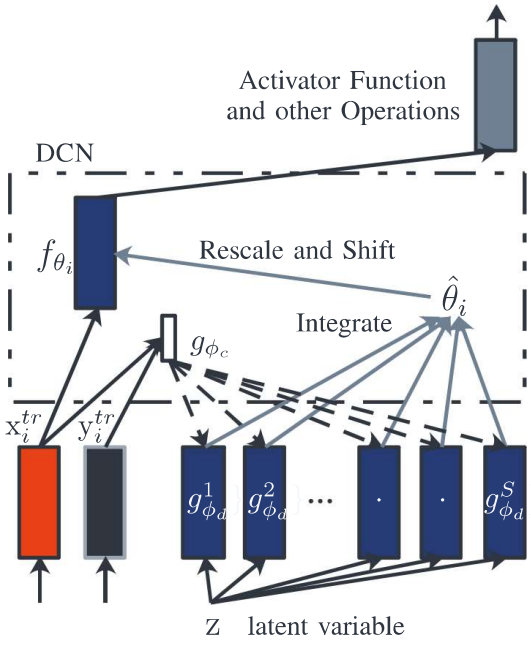

Fig. 3. Typical structure of a neural network layer with DCN, where $\mathrm{x}_i^{tr} = \{\mathrm{x}_{ij}^{tr}\}_j$ and $\mathrm{y}_i^{tr} = \{\mathrm{y}_{ij}^{tr}\}_j$, and the choice network g_{ϕ_c} is illustrated as a white box.

Algorithm 1 Inner Loop of DCN.

Require: Choice network g_{ϕ_c}; Decoders $\{g_{\phi_d}^1, g_{\phi_d}^2, \cdots, g_{\phi_d}^S\}$; Latent variable z; Training samples $\mathcal{D}_i^{tr} = \{\mathrm{x}_{ij}^{tr}, \mathrm{y}_{ij}^{tr}\}_j$; Testing samples $\mathcal{D}_i^{test} = \{\mathrm{x}_{ij}^{test}, \mathrm{y}_{ij}^{test}\}_j$; Learning rate α; Number of steps M; Loss function $\mathcal{L}$.

Initialize $\mathrm{z}' = \mathrm{z}$
$\{c_1, c_2, \cdots, c_S\} \leftarrow g_{\phi_c}(\mathcal{D}_i^{tr})$
for $m = 1, \cdots, M$ **do**
  $\hat{\theta}_i \leftarrow \sum_{s=1}^{S} c_s \cdot g_{\phi_d}^s(\mathrm{z}')$
  $\theta_i \leftarrow$ Normalize $\hat{\theta}_i$ by Eq. (4)
  $\mathcal{L}_i^{tr} = \sum_j \mathcal{L}(f_{\theta_i}(\mathrm{x}_{ij}^{tr}), \mathrm{y}_{ij}^{tr})$
  $\mathrm{z}' \leftarrow \mathrm{z}' - \alpha \nabla_{\mathrm{z}'} \mathcal{L}_i^{tr}$
end for
$\hat{\theta}_i \leftarrow \sum_{s=1}^{S} c_s \cdot g_{\phi_d}^s(\mathrm{z}')$
$\theta_i \leftarrow$ Normalize $\hat{\theta}_i$ by Eq. (4)
$\mathcal{L}_i^{test} = \sum_j \mathcal{L}(f_{\theta_i}(\mathrm{x}_{ij}^{test}), \mathrm{y}_{ij}^{test})$
return $\mathcal{L}_i^{test}$

machine (NTM) [25] is adopted. The choice network, g_{ϕ_c}, selects each decoder with different extents, $\{c_1, c_2, \cdots, c_S\}$. When applying an NTM, the weights of the decoders are not provided by the attention model but flexibly via the use of a fuzzy set (which will be explained in Section III-B).

Given the weights of the decoders, the latent variable of the "inner loop" is initialized with the latent variable $\mathrm{z}' = \mathrm{z}$ and it is decoded to obtain the parameters of a neural network model for the task $\mathcal{T}_i$ under consideration:

$$\hat{\theta}_i \leftarrow \sum_{s=1}^{S} c_s \cdot g_{\phi_d}^s(\mathrm{z}'). \tag{3}$$

Before parameterizing the neural network model with $\hat{\theta}_i$, to prevent vanishing gradient and accelerate convergence, $\hat{\theta}_i$ is normalized with batch normalization [26], which is given by:

$$\theta_i = \rho * \frac{\hat{\theta}_i - \xi}{\sqrt{\sigma^2 + \epsilon}} + \beta, \tag{4}$$

where ξ and σ^2 are the mean and variance of $\{\hat{\theta}_i\}_i$, respectively, ρ and β are learnable parameters, ϵ is a positive value close to zero that is added to the denominator for numerical stability, and $\epsilon = 1e-05$ is chosen following the work [26]. After obtaining θ_i, the model f_{θ_i} is built in relation to the task. f_{θ_i} is used to process each data point from training samples with a general feed-forward mapping. The forward process is depicted in Fig. 3.

After obtaining the prediction and loss functions for all training samples, gradient descent is used to update θ. Note that θ is not directly updated whilst the latent variable z is updated instead:

$$\mathrm{z}' \leftarrow \mathrm{z}' - \alpha \nabla_{\mathrm{z}'} \mathcal{L}_i^{tr}(f_{\theta_i}), \tag{5}$$

where α is the learning rate of gradient descent within the "inner loop". For simplicity, $\sum_j \mathcal{L}(f_{\theta_i}(\mathrm{x}_{ij}^{tr}), \mathrm{y}_{ij}^{tr})$ is rewritten as $\mathcal{L}_i^{tr}$. In the next step, the model parameterized by θ_i is used to calculate the loss on the training samples, and the process $(3) \rightarrow (4) \rightarrow (5) \rightarrow (3)$ loops several times before it is evaluated on the testing samples. The "inner loop" process of DCN is summarized in Algorithm 1.

Finally, the adapted parameters θ_i are used to calculate the testing loss $\sum_j \mathcal{L}_i(f_{\theta_i}(\mathrm{x}_{ij}^{test}), \mathrm{y}_{ij}^{test})$, written as $\mathcal{L}_i^{test}$. Within the "outer loop", the choice network, g_{ϕ_c}; the decoders, $\{g_{\phi_d}^1, g_{\phi_d}^2, \cdots, g_{\phi_d}^S\}$; and the iatent variable, z, are updated to reduce $\mathcal{L}_i^{test}$.

B. Decoders

The main challenge for the implementation of DCN is that if fully connected multilayer networks are used as decoders, a significantly large number of parameters will be required. The decoders receive the latent variable, producing the model parameters of a neural network. Because the dimension of the model parameters is rather large, the complexity of the decoders often becomes unacceptable. To address this problem, methods are introduced to reduce the complexity of the decoders.

1) Group Linear Transformation: Although the main challenge is the output of the decoders being of a high dimensionality, the dimensionality of the latent variable z should also be considered (which may be 10 times smaller than the total of model parameters at most). This is because if the number of the latent variables is too low, it will limit the expression of the metalearning model.

One way to overcome these two challenges, is to divide the input and weight matrices of a certain layer in the decoders into groups, and to reuse each of the weight matrices in all groups of the input. Here, as an example, the first layer of the decoders is utilized for this to illustrate the idea. In particular, the latent variable is divided into a number of groups $\mathrm{z} = [\mathrm{z}_1, \mathrm{z}_2, \cdots, \mathrm{z}_{F_g}]$, where z is a matrix and the elements in each column are from the same group. Each element of the output depends only on the latent variables in

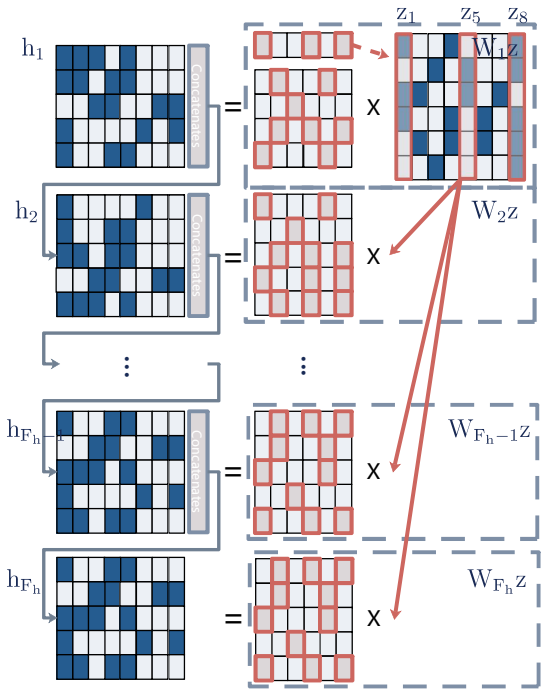

Fig. 4. Group linear transformation structure, where $\{\mathrm{h}_1, \mathrm{h}_2, \cdots, \mathrm{h}_{\mathrm{F_h}}\}$ are concatenated rto h and the length of the vector in each channel of h is increased F_h times, which is the number of the weight matrices.

the group. Then, each weight matrix of $\{\mathrm{W}_1, \mathrm{W}_2, \cdots, \mathrm{W}_{F_h}\}$ is exploited to calculate the hidden variable:

$$\mathrm{h}_n = \mathrm{W}_n \mathrm{z}, \ (n = 1, 2, \cdots, F_h). \tag{6}$$

From this, $\{\mathrm{h}_1, \mathrm{h}_2, \cdots, \mathrm{h}_{F_h}\}$ are concatenated into a matrix h and of course, h has already been divided into F_h. This process is summarized in Fig. 4. The resulting structure is named as GLT, which is then analyzed in order to reduce the number of parameters in the decoder network (see III-B4).

2) Non-linear: Within the hidden layer, the exponential linear unit (ELU) ($\omega = 1$) [27] is chosen as the nonlinear transformation in the decoder network, which is given by:

$$\mathrm{ELU}(x) = \max(0, x) + \min(0, \omega * (\exp(x) - 1)), \tag{7}$$

where ω is for simplicity, set to one in all experimental investigations unless otherwise stated. In the output layer, a double-thresholding strategy is adopted, applying an equal threshold on both the positive and the negative side. Such a function, called "softshrink", is utilized on the basisof PyTorch [28]. It is given by:

$$softshrink(x) = \begin{cases} x - \lambda, & \text{if } x > \lambda \\ x + \lambda, & \text{if } x < -\lambda \\ 0, & \text{otherwise} \end{cases}, \tag{8}$$

where λ denotes the threshold. For all experiments in this work, λ is set to 0.01, as commonly done in the literature.

3) Structure sharing: To further reduce the complexity of the decoders, all the decoder networks are devised to share the same low-level layers. In particular, the last layer of this neural network is set to have multiple heads, with each head representing a decoder and the output dimension of each of the heads being the same to one another. As the functions of each head are related to each other, this type of structure not only helps reduce the time and spatial complexity of the model, but also prevents gradient vanishing while rapidly accelerating convergence. The reason is that the shared shallow layers of the network can obtain the gradients from all heads. This is a classical method adopted in a number of deep learning works (e.g., [29]–[31]), and it has been proven to be efficient supported with empirical evidence.

The choice network and decoders are reused in different layers. However, if the task features are the same for these layers, all layers with a DCN structure are set to have the same $\{c_1, c_2, \cdots, c_S\}$. The features of the training samples of the previous layer are employed as the task features of the current layer. In so doing, each layer chooses its own decoder. Note that in all following experimental studies, a 2-layer neural network is used to implement the decoder, whose outputs of the second layer are S vectors with a size equaling to the number of the model parameters.

Note that in certain circumstances, such as miniImageNet, the number of model parameters may still be too large. Thus, $\hat{\theta}$ is resized with linear interpolation to enlarge its dimensionality. This method effectively reduces the output size of the decoders and makes each model parameter related to the others, which can be viewed as a type of regularization [17]. In addition, the execution of linear interpolation allows DCN to be reused in layers with different model dimension parameters. In implementation, the upsampling bilinear 2D model based on PyTorch [28] is utilized to perform the required linear interpolation.

4) Complexity Analysis One: The number of model parameters in the prediction model is first analyzed. The number of channels (corresponding to the convolution layer or the fully connected layer) is assumed to be F. If the numbers of channels are equal to each other in the hidden layers, they will be proportional to F^2. Denote such a proportion rate as $l\kappa$, with l representing the number of layers and κ the kernel size of the convolutional unit. Thus, most of the convolutional neural networks (CNNs) will meet this condition. Any deviations caused by the first and last layers can therefore be ignored due to their rather minute magnitudes.

Consider the worst case, where each decoder is a single fully connected network. Suppose that both the input and the output of the decoder network are the latent variable and the model parameters, respectively. Then, the fact of $\dim(\mathrm{z}) \propto \dim(\theta_i)$ leads to $\dim(\mathrm{W}) \propto F^4 l^2 \kappa^2$, where W denotes the weight matrix of the decoder network. Since the time and spatial complexities are proportional to the number of weights in a neural network, both the time and spatial complexities are $O(F^4 l^2 \kappa^2)$.

Now that all decoders are shared amongst the different layers, the time and spatial complexities are reduced to $O(F^4 S \kappa^2)$. If the decoders are also shared between the dimensions of the kernel, the complexity will decrease to $O(F^4 S)$. In the general case, $S \ll l^2 \kappa^2$, when replacing the fully connected network with the proposed GLT structure.

The number of parameters in a 2-layer fully connected network and GLT are $[\dim(\mathrm{z}) + \dim(\theta_i) S] \dim(\mathrm{h})$ and $\frac{\dim(\mathrm{h})}{\dim(\mathrm{z})} F_g^2 +$

$\frac{\dim(\theta_i)}{\dim(\mathrm{h})} SF_h^2$, respectively, where $\dim(\mathrm{h})$ and $\dim(\mathrm{z})$ denote the total dimensionality of the latent variables and that of the hidden variables. Note that $\frac{\dim(\mathrm{h})}{\dim(\mathrm{z})}$ and $\frac{\dim(\theta_i)}{\dim(\mathrm{h})}$ should be integers, while $1 \leq F_g \leq \dim(\mathrm{z})$ and $1 \leq F_h \leq \dim(\mathrm{h})$. Smaller F_g and F_h values indicate fewer parameters in GLT. Therefore, the number of parameters can be reduced by decreasing the number of groups. If $F_g = F_h = 1$ and $\dim(\mathrm{h})^2 = \dim(\mathrm{z})\dim(\theta_i)$, the time and spatial complexities are reduced to $O(\frac{F}{\dim(\mathrm{z})})$. In contrast, if $F_g = \dim(\mathrm{z})$ and $F_h = \dim(\mathrm{h})$, GLT is equal to the fully connected network.

C. Choice Network

The choice network, g_{ϕ_c}, receives the task features and produces the choice of the decoders. In order to answer the question of how to choose the task features, questions of what type of neural network to use for processing the task features and how to calculate the weights for each decoder need to be considered.

1) Choice Mechanism: First, the choice network is reused in different layers. To select different decoders for different layers, the choice network receives the input features of all training samples in the current layer as its task features. Thus, each layer chooses its own decoder. Second, the choice network is reused in the different dimensions of the convolutional network kernel. The task features should therefore be organized with respect to the dimension of the kernels; details of which are given in Appendix A-A.

After obtaining the task features, a capsule net [32] is adopted to process such features (with just one capsule layer utilized in this work). The task features, which are input to the capsule layer, are divided into a number of capsules, and each capsule is set o only involve one variable. Thus, the output variables of the capsule layer are all in one capsule. This follows the conventional process of dynamic routing [32] in the literature. For academic completeness, details of this are given in Appendix A-B.

Finally, fuzzy sets are used to represent the weights of each decoder. Suppose that the output variables of the capsule layer lie in $[-1, 1]$ and are denoted as $\{v_1, v_2, \cdots, v_{F_f}\}$. They are transformed to fall within $[0, 1]$ to denote the state variables, $\gamma_n = (v_n + 1)/2,\ n = 1, 2, \cdots, F_f$. The value of each state variable is represented by a pair of fuzzy sets (say, A and B). Thus, there are 2^n (i.e., 2^{F_f}) value combinations, and each such combination of two fuzzy sets determines the weights of a decoder. From this, it can be readily derived that $S = 2^{F_f}$. The relationship between the membership functions of the two fuzzy sets is set to $\mu_A(x) = 1 - \mu_B(x)$, and the value of μ_A are determined by

$$\mu_A(x) = \begin{cases} 1, & x \leq 0 \\ 1 - x, & 0 < x \leq 1 \\ 0, & x > 1 \end{cases}. \tag{9}$$

Without losing generality, denote the membership functions of each pair of fuzzy sets specifying γ_n as $\mu_A(\gamma_n)$ and $1 - \mu_A(\gamma_n)$. The weight of each decoder is then calculated by:

$$c_s = \frac{\prod_n \mu_A(\gamma_n)^{a_s^n} \mu_B(\gamma_n)^{1-a_s^n}}{\sum_{l=1}^{S} \prod_n \mu_A(\gamma_n)^{a_l^n} \mu_B(\gamma_n)^{1-a_l^n}}, \tag{10}$$

where $a_s^n \in \{0, 1\}$ indicates which fuzzy set the s^{th} variable takes as its value. Due to $\sum_{l=1}^{S} \prod_n \mu_A(\gamma_n)^{a_l^n} \mu_B(\gamma_n)^{1-a_l^n} \equiv 1$, Eq. (10) can be written as:

$$c_s = \prod_n \mu_A(\gamma_n)^{a_s^n} \mu_B(\gamma_n)^{1-a_s^n}. \tag{11}$$

Note that S in Eq. 10 is a hyperparameter. In general, a higher value of S normally implies that the model can obtain a better performance since higher values represent more decoders, but there is balance to strike between accuracy and efficiency. Therefore, S is empirically set to four in this work; such a structural specification of the choice network entails a low computational complexity.

2) Complexity Analysis Two: More generally speaking, the complexity of the choice network can be analyzed as follows.

First, the complexity of the capsule net is the same as that of a regular fully connected network and can be obtained by multiplying the input dimension by the output dimension. Since the input and output of the choice network are data features and the membership values of their states, respectively, its complexity is $O(F_{\mathrm{x}} F_f)$, where F_{x} is the number of the data features. For this work, F_{x} is not greater than the dimensionality of the set of original features. As $F_f = \log 2S$, the complexity of the choice network can be rewritten as $O(F_{\mathrm{x}} \log 2S)$. If however, no fuzzy set representation is used in the choice network, the network's complexity is $O(F_{\mathrm{x}} S)$. Note that feature embedding is herein used to process data in order to obtain low-dimensional, highly abstract features (see Section III-D next). In addition, unless otherwise stated, $F_f = \log S$ and the largest S are both set to 16 to facilitate fair comparison in the experimental studies. As F_f is rather small, the size of the capsule net is fairly small relative to that of the decoders.

Second, the complexity of calculating the membership values is $O(S \log S)$. Obviously, this complexity can also be ignored relative to that of the decoders. As a result, the overall time and spatial complexities of DCN are similar to those of the decoders with little effect from the inclusion of the choice network.

D. Metatraining Strategies

A successful application of metalearning requires an appropriate introduction of the underlying metatraining strategies, which are discussed below.

1) Feature embedding: It is necessary to use a much deeper network (e.g., ResNet [33] or dense net [34]) to obtain a higher classification accuracy on a dataset involving complicated image contents, such as miniImageNet. However, deeper networks require a great amount of GPU memory to learn all the parameters. This is independent of whether the learning is done by an optimization-based method or a memory-based method, whilst a metric-based method may be unstable in such use [18]. Therefore, recently, cotraining or supervised pretraining methods have been proposed to ease this problem. For example, TADAM [18] trains ResNet with auxiliary cotraining of 64 classifications [18] to improve system convergence, and LEO [8] uses all classes from the training and validation sets to perform 80 pretraining steps to

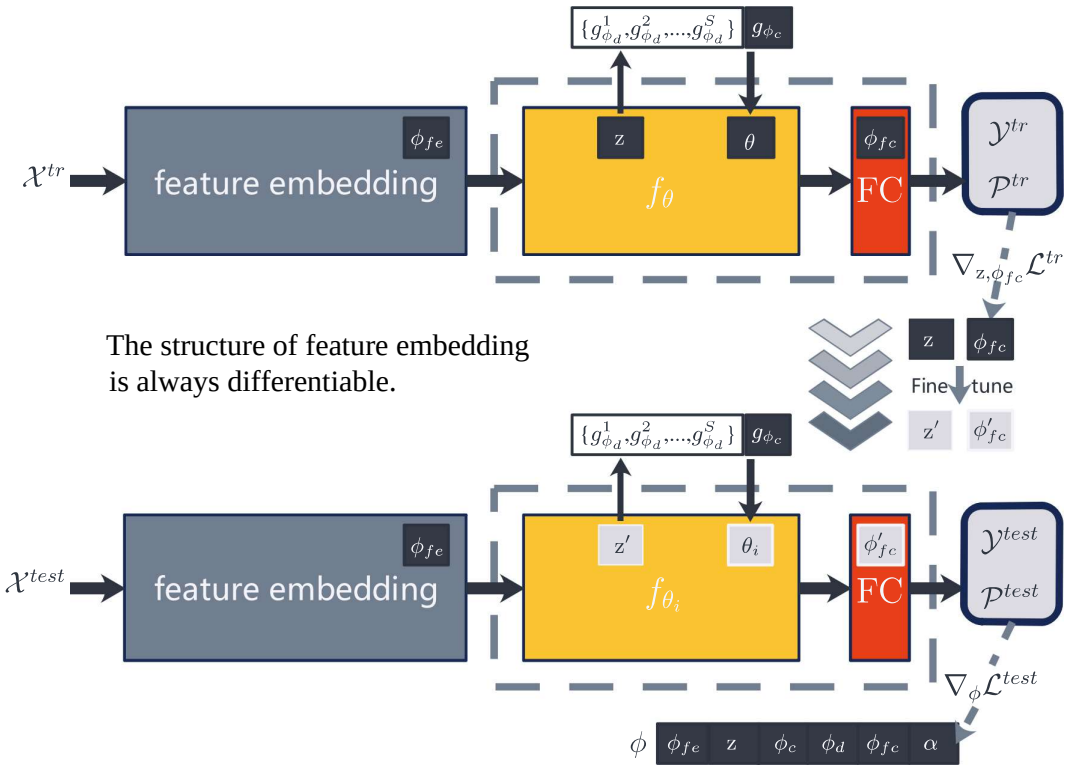

Fig. 5. A typical training process, where $\phi = \{\phi_{fe}, \mathrm{z}, \phi_c, \phi_d, \phi_{fc}, \alpha\}$; FC, denotes the last layer of the model; $\mathcal{X}^{tr} = \{x_{ij}^{tr}\}_j$ and $\mathcal{Y}^{tr} = \{y_{ij}^{tr}\}_j$ are the feature values and their corresponding labels in the training samples, $\mathcal{X}^{test} = \{x_{ij}^{test}\}_j$ and $\mathcal{Y}^{test} = \{y_{ij}^{test}\}_j$ are those of the testing samples, and $\mathcal{P}^{tr}$ and $\mathcal{P}^{test}$ are the model predictions for the training and testing samples. Note that z denotes the latent variable and is transformed into the model parameter θ by the decoder network. During the inner training step, the training loss $\mathcal{L}_i^{tr}$ is obtained and (z, ϕ_{fc}) is fine-tuned to $(\mathrm{z}', \phi'_{fc})$ by $\nabla_{\mathrm{z},\phi'_{fc}} \mathcal{L}_i^{tr}$ within the "inner loop". Then, the choice network and the decoders receive z′ and product θ_i. Last, the testing loss $\mathcal{L}_i^{test}$ is calculated to update ϕ during the "outer loop".

obtain a feature embedding, with other similar approaches as reported in [35]–[37].

In contrast, DCN is herein trained in an end-to-end manner, as illustrated in Fig. 5. Particularly, the job of feature embedding is implemented with a standard CNN (or a much deeper network, such as ResNet [33] or dense net [34]). As indicated in Fig. 5, the system parameters required by feature embedding, ϕ_{fe}, remain constant within the "inner loop" and are updated during the "outer loop". Because the fine tuning process (see next) in the last few layers is differentiable, the feature embedding mechanism can be updated by gradient back propagation from the last few layers directly. This makes metatraining a very deep network simpler and more efficient.

2) Fine tuning: DCN works based on the use of an optimization-based method and the latent parameters are updated by the method explained in III-A. Thus, it is required to implement gradient descent in the "inner loop". Indeed, the other parameters, such as ϕ_{fc} in the last fully connected layer, are updated by a gradient descent algorithm (e.g., MAML [12]), as shown in Fig. 5. Note that the hyperparameters controlling the "inner loop", such as the learning rate α, the choice network g_{ϕ_c} and the decoders $\{g_{\phi_d}^1, g_{\phi_d}^2, \cdots, g_{\phi_d}^S\}$, remain constant within the "inner loop" and updated during the "outer loop" by stochastic gradient descent (SGD).

The "outer loop" process of DCN is summarized in Algorithm 2. As can be seen from this algorithm, ϕ_{fc} is also updated during the "inner loop", which is slightly different from the process in Algorithm 1. After model initialization, the approach randomly samples a batch of tasks $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{test}\}_i$ with regard to the given task distribution $\mathcal{T}$ and extracts the features of all samples in $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{test}\}_i$ by feature embedding.

Then, forward propagation of training samples is performed for each task, and the backward gradient method is used to update the parameters ϕ_{fc} and z to obtain ϕ'_{fc} and z′ (as detailed in the "inner loop"), respectively. After training, a unique model is generated per task, and the model is tested on the testing samples of each task. Finally, the loss of all tasks based on the testing samples is used to update ϕ within the "outer loop". Since all operations within the "inner loop" process are derivable, the parameters can be updated by stochastic gradient descent based on the training result achieved within the "inner loop" process.

E. Ensemble Learning

The standard training protocol followed by most previous few-shot learning methods is to decrease the learning rate during training and to choose a resulting model with the validation set. However, a recent study has showed that a training protocol with a snapshot ensemble [19] may be more suitable for model training on certain datasets, such as miniImageNet. Such a protocol makes better use of a model obtained through a single training process.

Algorithm 2 Outer Loop of DCN.

Require: Parameters of the choice network and decoders, ϕ_c and ϕ_d; Learning rate of the "inner loop" α; Number of steps of the "inner loop" M; Loss function $\mathcal{L}$; Parameters of fully connected layer ϕ_{fc} and latent variable z; Distribution over the tasks $p(\mathcal{T})$; Parameters of the feature embedding ϕ_{fe}; Learning rate of the "outer loop" η;

Initialise $\phi = \{\phi_{fe}, \mathrm{z}, \phi_c, \phi_d, \phi_{fc}, \alpha\}$
while ϕ has not converged **do** \\ Outer Loop
  Sample batch of the tasks $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{test}\}_i \sim p(\mathcal{T})$
  for all $\{\mathcal{D}_i^{tr}, \mathcal{D}_i^{test}\}$ **do** \\ Inner Loop
    Use the feature embedding to extract features:

$$\hat{\mathcal{D}}_i^{tr} = \{\hat{\mathrm{x}}_{ij}^{tr}, \mathrm{y}_{ij}^{tr}\}_j,\ \hat{\mathcal{D}}_i^{test} = \{\hat{\mathrm{x}}_{ij}^{test}, \mathrm{y}_{ij}^{test}\}_j$$

    Initialize $\mathrm{z}' = \mathrm{z},\ \phi'_{fc} = \phi_{fc}$
    $\{c_1, c_2, \cdots, c_S\} \leftarrow g_{\phi_c}(\mathcal{D}_i^{tr})$
    for $m = 1, \cdots, M$ **do**
      $\hat{\theta}_i \leftarrow \sum_{s=1}^{S} c_s \cdot g_{\phi_d}^s(\mathrm{z}')$
      $\theta_i \leftarrow$ Normalize $\hat{\theta}_i$ by Eq. (4)
      $\hat{\mathrm{x}}_{ij}^{tr} \leftarrow f_{\theta_i}(\hat{\mathrm{x}}_{ij}^{tr})$
      $\mathcal{L}_i^{tr} = \sum_j \mathcal{L}(f_{\phi_{fc}}(\hat{\mathrm{x}}_{ij}^{tr}), \mathrm{y}_{ij}^{tr})$
      $\mathrm{z}' \leftarrow \mathrm{z}' - \alpha \nabla_{\mathrm{z}'} \mathcal{L}_i^{tr}$
      $\phi'_{fc} \leftarrow \phi'_{fc} - \alpha \nabla_{\phi'_{fc}} \mathcal{L}_i^{tr}$
    end for
    $\hat{\theta}_i \leftarrow \sum_{s=1}^{S} c_s \cdot g_{\phi_d}^s(\mathrm{z}')$
    $\theta_i \leftarrow$ Normalize $\hat{\theta}_i$ by Eq. (4)
    $\hat{\mathrm{x}}_{ij}^{test} \leftarrow f_{\theta_i}(\hat{\mathrm{x}}_{ij}^{test})$
    $\mathcal{L}_i^{test} = \sum_j \mathcal{L}(f_{\phi'_{fc}}(\hat{\mathrm{x}}_{ij}^{test}), \mathrm{y}_{ij}^{test})$
  end for \\ Inner Loop
  $\phi \leftarrow \phi - \eta \nabla_{\phi} \sum_i \mathcal{L}_i^{test}$
end while \\ Outer Loop

Note that when a model is trained on the basis of miniImageNet, the training and validation sets do not have overlapping classes and typically, the number of classes in the training set is rather small. Therefore, a larger generalization gap exists between the training and validation loss. When the training loss reduces rapidly, the validation loss barely changes. In this case, choosing models from different numbers of iterations may result in models with similar validation losses and completely different training losses. This means that certain models may have good generalization performance during training, but their performance can be rather different on various classes or datasets, showing the diversity and quality of models with ensemble learning.

Having recognized above, in this work, instead of using the best model returned by a training process, all models with strong performance from the training process are used. This approach has been adopted in "support nets" [38], with its effectiveness shown and supported by empirical results. Following this approach, top n models of the best performance on the validation set are herein chosen to construct an ensemble model.

Particularly, the models are selected over certain iteration intervals and are sorted according to their accuracy. These models are then added to the ensemble model in the sequence with respect to their accuracy on the validation set. If such an added model improves the performance of the ensemble model, it is retained; otherwise, it is dropped. The model with the highest validation accuracy is tested for the ensemble prior.

Ensemble learning is integrated with the proposed optimization-based DCN method, with the resultant system referred to as DCN with ensemble learning, and abbreviated as DCN-E. Such an ensemble learning approach exploits all information contained within the given all selected models that have been obtained in a single training process. As to be shown later, this method greatly improves the generalization performance of the resulting system.

F. Comparison between DCN and LEO

In principle, the proposed network has a similar manner aim to that of LEO [8]. That is to encode latent variables in response to the task and to decode latent variables as the model parameters, by the use of a certain decoder. Both LEO and DCN perform gradient descent within a low-dimensional latent space during the "inner loop". However, significant differences exist between them.

First, LEO and DCN address the issue of task dependence in different places. Whilst they both try to learn the latent variables and the decoders, via decoding the latent variables into model parameters, LEO attempts to build task dependence on the latent variables, while DCN does this on the decoder. Besides, the decoders of LEO and the latent variables of DCN are set for different tasks. Second, they build task dependence in a different way. LEO uses a network to convert the task features into the latent variables, while DCN chooses the decoders from the set of candidate decoders according to the given task features. Third, the model parameters generated by LEO and DCN are different. The former creates the model parameters for the last layer, while DCN generates parameters for the hidden layer. Higher time and space complexities are required for generating the model parameters of the hidden layer.

Note that as the model parameters in the last layer can be separately generated by the classes within a given task, the features of each class may be used as its model parameters. However, the model parameters of the hidden layer cannot be separated from classes. For this reason, the output size of a decoder increases from h_s to h_s^2, and the number of parameters in the decoder increases from h_s^2 to h_s^4. This is the problem analyzed in Section III-B4. Therefore, LEO does not considering the use of a multilayer nonlinear decoder within any large-scale model. Nonetheless, since DCN can reduce the parameter size of the decoders to an acceptable scale, DCN can use multilayer nonlinear decoders.

IV. Experimentation

A comparative experimental study in reference to the state-of-the-art approaches is reported in this section for model evaluation. It particularly addresses these three questions: (1) Can optimization control of DCN learn useful information for metalearning? (2) Can the proposed training strategy train a much deeper network without co-training or pretraining?

(3) Can snapshot ensemble [19] improve the performance of metalearning? In answering these questions, all experimental investigations are run on "Pytorch" [28].

Regression and classification tasks are used in the experiments. The regression task is regarded as a sinusoid curve fitting, with the results compared against those obtained by MAML [12]. A range of classification tasks are run on the Omniglot [39] and miniImageNet datasets [20], which are common benchmarks for few-shot learning in the literature. The code for all programs implemented is available for open access on github[1].

A. Data Description

The **sinusoidal** curve fitting task was originally reported in [12]. The data samples for each task are obtained from the input and output of a sine wave, where the amplitudes and phases of the task $p(\mathcal{T})$ are different. The amplitudes and random phases are sampled from uniform distributions over the ranges of 0.1 to 5.0 and 0 to π, respectively. Both training and testing features are sampled uniformly from $[-5.0, 5.0]$. The mean-squared error between the output of the network and the corresponding sine function value is used for both evaluation and computing the loss functions.

The **Omniglot** dataset consists of samples from 50 international languages, each character has 20 instances, and there are a total of 1,623 characters. The 20 instances of each character are written by a different person. Following [20], the Omniglot dataset is divided into 1,200 and 423 characters for training and evaluation, respectively. All images are resized to 28×28, and samples are augmented by rotating 90, 180, and 270 degrees. The model is evaluated on 1-shot and 5-shot, 5-way and 20-way tasks. Each task contains the same number of shots and queries.

The **miniImageNet** consists of 100 classes, each of which involves 600 natural images. All images are resized into those of 84×84 to ensure a fair comparison with the prior work. The miniImageNet dataset is sampled from the ILSVRC-12 dataset [40] (which was first proposed in [20] by Vinyals et al.). Following the split of the miniImageNet proposed by Ravi and Larochelle [14] and most previous work, the dataset is herein split into 64, 16 and 20 classes for training, validation, and testing, respectively.

B. Experimental Setup

Prior to the presentation and discussion of experimental results, it is helpful to give an overview of the experimental methodology used.

1) Sinusoid Curve Fitting: Three hidden layers of sizes, $[40, 40, 35]$, are used with rectified linear unit (ReLU) nonlinearities, and the two middle layers are used with DCN. This is different from the work of [12] because if two hidden layers were used in the model, there would be only one layer with DCN. In that case, it would be difficult to reflect the advantages of DCN since it would not be able to illustrate the idea of choosing the decoders that are shared between layers. Except for the hidden layers with DCN, the other layers are standard fully connected ones trained by MAML. There is no feature embedding involved as no fine tuning is required in this sinusoid curve fitting experiment.

During the training process, two-step updates are applied with the "inner loop" having a fixed learning rate of $\alpha = 0.01$ and the "outer loop" having a fixed learning rate of $\beta = 10^{-3}$. These rates are set following the common practice in the literature. Both models are trained for 60,000 iterations by Adam with AMSGrad [41]. Note that the inner learning and weight decay rate are not updated in this experiment. During testing, 10-, 20- and 30-step updates for the 5-shot, 10-shot, and 20-shot images are used, respectively. The popular mean-squared loss function is used, with the form of the loss being given by:

$$\mathcal{L}_i^{test} = \sum_j \| f_\theta(\mathrm{x}_{ij}^{test}) - \mathrm{y}_{ij}^{test} \|_2^2, \tag{12}$$

where x_{ij}^{test} and y_{ij}^{test} are the input and output sampled from each sinusoid curve, respectively, and f_θ denotes a model with parameters θ.

Since the sizes of the hidden layers are not equal to each other, linear interpolation is adopted to resize the output of the decoders. There are 3,116 learnable parameters in MAML and 2,020 learnable parameters in DCN. Their forwarding structures built in the "inner loop" are the same. A total of 600 minibatches of tasks are randomly sampled for evaluation, and each batch has 25 tasks.

2) Omniglot Classification: The layers with DCN are applied to replace two convolution layers before the fully connected layer in a standard 4-layer embedding CNN (which was proposed in [20]), and mean pooling is used to replace max pooling. Such a model is denoted by DCN4 hereafter. Similarly, DCN6 denotes a model implemented by four convolution layers and two layers with DCN being employed before the fully connected layer. Layers used before those layers with DCN realize feature embedding, which are not fine-tuned, and their parameters are updated by gradient descent during the "outer loop". The fully connected layer with a size of 64 and softmax is exploited to calculate the classification probability.

As with the case study on sinusoid curve fitting, the proposed model has fewer parameters than a standard 4-layer convolution embedding with a fully connected layer and softmax. In running the 5-way task, there are 112,005 learnable parameters in a standard 4-layer convolution embedding network and 76,553 learnable parameters in DCN4. The models are trained for 60,000 iterations, and the initial learning rate is set to 10^{-3} and is decayed by 0.5 for every 10K episodes. Inner learning and the weight decay rate are updated, and their learning rates are 0.1 times those of the other parameters. The common cross-entropy loss is used for classification, which takes the following form:

$$\mathcal{L}_i^{test} = \sum_j \sum_k \mathrm{y}_{ij,k}^{test} \log f_{\theta,k}(\mathrm{x}_{ij}^{test}) \tag{13}$$

where x_{ij}^{test} denotes the features of a testing example, $\{\mathrm{y}_{ij,k}^{test}\}_k$ denotes the labels of a testing example, and f_θ denotes a model with parameters θ.

[1] www.github.com/AceChuse/DCN

Given the similar aim between DCN is similar to LEO [8], for fair comparison, LEO is used to replace DCN in DCN4, DCN4-E, DCN6, and DCN6-E in places. The coefficients of entropy penalty and stopgrad penalty are set to 0.1 and 1e-8, which are of a similar scale as those obtained by random search (see [8]), without using the orthogonality penalty (because it would require too much GPU memory). As explained earlier, LEO requires substantially more parameters, making it unfair to directly compare with other models. In running the 5-way task, there are 5,125,525 parameters in LEO4, and the other three models also contain many more parameters than the DCN model. However, DCN has better performance in most cases. Since DCN4 is of the same size as the model used in prior works, it is compared with those methods too.

3) MiniImageNet Classification: The feature embedding network is trained by DenseNet-161 [34] with 96 initial channels and 16 growth rates (but not fine-tuned during the "inner loop"). After DenseNet-161, one 1×1 convolution layer is used to change the number of channels to 256 without pooling as feature embedding. Standard data augmentation from ImageNet is employed, involving: 1) randomly flipping an image horizontally; 2) resizing of each image into a 100×100 frame while cropping it with a random size and aspect ratio and then, resizing to 84×84; and 3) randomly jittering the brightness contrast and saturation of a given image.

After feature embedding, the structure of the ResNet $(3, 3)$ block will include a layer with DCN, a convolution layer with batch normalization and the nonlinear function ReLU. Although there is only one layer within DCN, it is yet effective. This is because the decoders can be reused within the layer. The features after global average pooling are fed into a fully connected layer with softmax. Models are trained for 40K iterations by Adam with AMSGrad [41]. The initial learning rate is set to 0.1 and decay rate to 0.5 for every 10K episodes. After 20k episodes, learning rate cyclic annealing is used, which is defined by:

$$\upsilon(t) = \frac{\upsilon_0}{2}\left(\cos\left(\frac{\pi \mathrm{mod}(t-1, T)}{T}\right) + 1\right), \qquad (14)$$

where υ_0 denotes the initial learning rate that is set to $\upsilon_0 = 0.001$, and T is the period of cyclic annealing which is empirically set to $T = 2000$.

Similar to Omniglot, the classification system for miniImageNet is trained using cross-entropy loss, learnable inner learning and a weight decay rate. The learning rate is 0.1 times that of the other parameters. Clips of the gradient norm of an adaptable parameter are added during the "inner loop". After testing on the validation set, the model is retrained on the dataset involving the training set and validation set, with the same hyperparameters. A resulting model trained with the same number of iterations to that used for training the ensemble is chosen and tested on the testing set. Note that the computing power available in conducting the present research is not sufficient to train WRN-28-10 [42] using the population-based training (PBT) [43] (as done in [8]). Thus, DCN is replaced with LEO in dealing with the miniImageNet model. There are more than 80 million parameters in the minImageNet model with LEO, which is much larger than the number of parameters in the DCN model (that has 4 million parameters).

The proposed model is evaluated on 1-shot, 5-shot, and 5-way tasks. All tasks have eight sub-tasks in a mini-batch, each containing 15 samples from a query. One thousand tasks are randomly generated through the use of the validation and testing sets after training.

TABLE II
Number of parameters in the models which are used in few-shot classification on the miniImageNet dataset.

Model	**5-way space**	
	1-shot	5-shot
LEO (mini)	81.74M	81.74M
DCN (mini)	4.07M	4.07M

TABLE III
Average time consumed by 10 iterations on the miniImageNet dataset. After $\pm$ is the 95% confidence intervals over 10 iterations. The units are seconds.

Model	**5-way time**	
	1-shot	5-shot
LEO(mini)	3.322±0.015	3.640±0.004
DCN(mini)	2.757±0.010	3.052±0.036

C. Time and Space Consumption

To demonstrate the efficiency of DCN, both time and space consumptions respectively incurred by DCN and LEO are compared. The number of parameters in the models that work on the miniImageNet and Omniglot datasets are listed in Tables II, V, and IV, respectively.

The average time consumption and confidence intervals of 10 iterations on the miniImageNet datasets are shown in III, respectively. Clearly, the parameter number of the DCN model is much lower than that of the corresponding LEO model. The time consumption of the DCN model is lower than that of LEO on the miniImageNet dataset. However, on the Omniglot dataset, which is of a moderate size, the time consumption of the DCN model is higher than that of LEO. This is likely due to the fact that LEO uses a linear decoder mechanism, which has a faster parallel calculation speed than a multilayer nonlinear decoder. Nonetheless, once the DCN model is used to solve problems involving massive data, DCN will have a faster processing speed than LEO. All compared results are listed in Tables II, III, IV, V, VI, and VII.

D. Model Accuracy

The results on the accuracy of sinusoidal curve fitting are summarized in Table X, which shows that the proposed model has better performance than MAML. Together with the observation on the model efficiency as discussed above, this implies that sinusoid function can be learned rapidly by DCN. Note that the performance improvement is more significant when the number of shots is smaller. Comparing 5-shot to 20-shot images, the loss of MAML increases 28 fold, but the

TABLE IV
Number of parameters of the models that are used in 5-way few-shot classification on the Omniglot dataset.

Model	5-way space	
	1-shot	5-shot
LEO4 [23]	5.126M	5.126M
LEO6 [23]	5.200M	5.200M
DCN4	0.077M	0.077M
DCN6	0.151M	0.151M

TABLE V
Number of parameters of the models that are used in 20-way few-shot classification on the Omniglot dataset.

Model	20-way space	
	1-shot	5-shot
LEO4 [23]	5.127M	5.127M
LEO6 [23]	5.200M	5.200M
DCN4	0.078M	0.078M
DCN6	0.151M	0.151M

TABLE VI
Average time consumed by 10 iterations on the Omniglot 5-way few-shot classification task. After $\pm$ is the 95% confidence interval over 10 iterations. The units are seconds.

Model	5-way time	
	1-shot	5-shot
LEO4 [23]	0.059$\pm$0.006	0.130$\pm$0.008
LEO6 [23]	0.067$\pm$0.004	0.168$\pm$0.003
DCN4	0.234$\pm$0.003	0.644$\pm$0.003
DCN6	0.219$\pm$0.001	0.570$\pm$0.004

TABLE VII
Average time consumed by 10 iterations on the Omniglot 20-way few-shot classification task. After $\pm$ is the 95% confidence interval over 10 iterations. The units are seconds.

Model	20-way time	
	1-shot	5-shot
LEO4 [23]	0.056$\pm$0.002	0.216$\pm$0.001
LEO6 [23]	0.076$\pm$0.003	0.301$\pm$0.001
DCN4	0.277$\pm$0.002	1.082$\pm$0.003
DCN6	0.240$\pm$0.001	0.955$\pm$0.003

loss of DCN only increases 6 fold. This demonstrates that the approach introduced in this work has made better use of a small amount of data in performing the regression task.

The results of model accuracy regarding the Omniglot dataset are listed in Tables IX and VIII. The "Support nets6" model in Table VIII involves 6-layer embedding CNNs, , which has more convolutional layers than, and hence is different from, the others. It is included here to have a more complete comparison with the existing literature. Except for the 5-way 1-shot task, all implemented models following the proposed approach obtain higher accuracy than the state-of-the-art ones. The results of DCN4 on the 20-way, 1-shot and 5-shot tasks are even better than those of "Support nets6". In addition, the results in Table IX show that DCN has better performance on the Omniglot dataset than LEO.

The results on the miniImageNet dataset are shown in Table XI. Whilst most of the large-scale models require co-training or pretraining, the proposed method can lead to state-of-art results on 5-way 5-shot classification, or comparable to those of state-of-art 5-way 1-shot classification, without co-training or pretraining. DCN and LEO are shown to have similar accuracies on the miniImageNet, but DCN has much fewer parameters than LEO.

Furthermore, the improvements brought forward by ensemble learning can be observed in Tables IX and XI: it enhances most of the results. In particular, ensemble learning has the most significant positive impact upon the model learned for the miniImageNet dataset, which has higher generation gaps.

E. Ablation Study

The effects of GLT, choice networks and ensemble learning are studied here, supported with the use of additional validation datasets, in terms of generalization accuracy. The results on miniImageNet are shown in Table XII. The results illustrate that the training protocol with a snapshot ensemble is more suitable for few-shot learning tasks. In addition, the improvement gained from the snapshot ensemble is more significant in miniImageNet than in Omniglot, which validates the hypothesis taken in this work in that a larger generalization gap will cause more improvement.

As for the impact of the choice network as introduced in Section III-C, a significant improvement is observed. Since LEO has provided task-related information by the encoder and relation network, the results of LEO are better than those of DCN without the choice network. However, in most cases, DCN with a choice network obtains better results. Possible reason for this outcome is that the learning process under such a severe experimentation condition is too complicated for a linear decoder, such as LEO, to perform, thereby unable to return an optimal learning system. In contrast, DCN can use multilayer nonlinear decoders to handle these complex nonlinear situations.

F. Discussions

The proposed approach benefits from an integrated use of DCN, feature embedding, and ensemble learning. First, DCN improves the performance of the model by updating

TABLE VIII
Averages of the accuracy values over 1800 tasks generated from the testing set, of few-shot classification on the Omniglot dataset. After ± is the 95% confidence interval over the testing tasks. The best-performing methods are highlighted, and '-' means no report.

Model	**5-way Acc**		**20-way Acc**	
	1-shot	5-shot	1-shot	5-shot
MANN [10]	82.8%	94.9%	-	-
Siamese nets [44]	97.3%	98.4%	88.1%	97.0%
Matching nets [20]	98.1%	98.9%	93.8%	98.5%
Neural statistician [45]	98.1%	99.5%	93.2%	98.1%
Prototypical nets [46]	98.8%	99.7%	96.0%	98.9%
MAML [12]	98.7±0.4%	99.9±0.1%	95.8±0.3%	98.9±0.2%
Meta-SGD [15]	99.53±0.26%	**99.93±0.09%**	95.93±0.38%	98.97±0.19%
Relation nets [22]	**99.6±0.2%**	99.8±0.1%	97.6±0.2%	99.1±0.1%
SNAIL [23]	99.07±0.16%	99.78±0.09%	97.64±0.30%	99.36±0.18%
Support nets4 [38]	99.24±0.14%	99.75±0.15%	97.79±0.06%	99.27±0.15%
Support nets6 [38]	99.37±0.09%	99.80±0.03%	**98.58±0.07%**	**99.45±0.04%**
DCN4	**99.800±0.050%**	99.891±0.008%	**98.825±0.025%**	**99.505±0.004%**

TABLE IX
Averages of the accuracy values over 1,800 tasks generated from the testing set, of few-shot classification on the Omniglot dataset. After ± is the 95% confidence interval over the testing tasks. The higher accuracies of DCN and LEO with same structures are highlighted.

Model	**Choice**	**Ens**	**5-way Acc**		**20-way Acc**	
			1-shot	5-shot	1-shot	5-shot
LEO4			99.433±0.074%	99.727±0.011%	98.358±0.030%	99.178±0.005%
LEO4		√	99.444±0.075%	99.727±0.011%	98.361±0.030%	99.184±0.005%
LEO6			99.422±0.073%	99.751±0.012%	98.692±0.026%	99.355±0.005%
LEO6		√	99.478±0.070%	99.771±0.010%	98.736±0.026%	99.370±0.004%
DCN4			99.722±0.050%	99.860±0.009%	94.556±0.052%	97.719±0.009%
DCN4		√	99.733±0.055%	99.851±0.009%	94.775±0.050%	97.838±0.009%
DCN6			99.711±0.058%	99.913±0.006%	96.933±0.040%	98.974±0.006%
DCN6		√	99.733±0.055%	**99.924±0.006%**	97.236±0.038%	99.061±0.006%
DCN4	√		**99.800±0.050%**	**99.891±0.008%**	**98.825±0.025%**	**99.505±0.004%**
DCN4	√	√	**99.833±0.042%**	**99.909±0.007%**	**98.842±0.025%**	**99.522±0.004%**
DCN6	√		**99.856±0.040%**	**99.924±0.007%**	**99.183±0.021%**	**99.593±0.004%**
DCN6	√	√	**99.922±0.032%**	**99.924±0.007%**	**99.108±0.022%**	**99.633±0.003%**

TABLE X
Mean-squared error of sinusoid curve fitting. The 95% confidence intervals over given tasks is shown after ±.

Models	5-shot	10-shot	20-shot
MAML	0.1564±0.0052	0.0360±0.0011	0.0055±0.00014
DCN	0.0176±0.0011	0.0051±0.0001	0.0028±0.00005

parameters in a low-dimensional space. Importantly, adopting DCN does not increase the size of the model, but reduces it. The reasons are that the model parameters of the layer with DCN are replaced by a latent variable, which has a much lower dimension, and DCN has fewer parameters itself.

Second, feature embedding is also proven to be effective and efficient. Fine tuning is only required on the last few layers, while the feature embedding process can be updated during the "outer loop" to enhance the performance of the "outer loop" itself. This enables the model to learn useful features for metalearning and to achieve higher accuracy than many models with pretraining feature embedding, while making the training of a large-scale metamodel end-to-end.

TABLE XI
Averages of the accuracy values over 1000 tasks generated from the test dataset, of few-shot classification on the miniImageNet dataset. After ± is the 95% confidence interval over the testing tasks. The first set shows the results of the models that use convolutional networks, and the results of the models that use much deeper networks with ResNet or dense net blocks are shown in the second set. The best-performing methods are highlighted.

Model	**Fine Tune**	**Co- or Pretraining**	**5-way Acc**	
			1-shot	5-shot
Matching nets [20]	N	N	43.56±0.84%	55.31±0.73%
Meta-learner LSTM [14]	N	N	43.44±0.77%	60.60±0.71%
MAML [12]	Y	N	48.70±1.84%	63.11±0.92%
Prototypical nets [46]	N	N	49.42±0.78%	68.20±0.66%
Meta-SGD [15]	Y	N	50.47±1.87%	64.03±0.94%
Reptile [13]	Y	N	49.97±0.32%	65.99±0.58%
Relation nets [22]	N	N	50.33±0.82%	65.32±0.70%
Support nets [38]	N	N	56.32±0.47%	71.94±0.37%
SNAIL [23]	N	N	55.71±0.99%	68.88±0.92%
Dynamic Few-Shot Visual Learning [37]	Y	Y	56.20±0.86%	73.00±0.64%
Discriminative k-shot learning [36]	Y	Y	56.30±0.40%	73.90±0.30%
Predicting Parameters from Activations [35]	Y	Y	59.60±0.41%	73.74±0.19%
TADAM [18]	N	Y	58.50±0.30%	76.70±0.30%
LEO [8]	Y	Y	**61.76±0.08%**	**77.59±0.12%**
LEO (best)	Y	N	60.22±0.08%	77.67±0.06%
DCN (best)	Y	N	**61.64±0.08%**	**77.74±0.06%**

TABLE XII
TAverages of the accuracy values over 1000 tasks generated from the test dataset, of few-shot classification on the miniImageNet dataset. After ± is the 95% confidence interval over the testing tasks. The best-performing methods are highlighted.

Model	**Choice**	**Ens**	**+Val**	**5-way Acc**	
				1-shot	5-shot
LEO				57.90±0.09%	73.31±0.07%
LEO		√		59.57±0.08%	77.02±0.06%
LEO			√	59.06±0.09%	73.51±0.06%
LEO		√	√	60.22±0.08%	77.67±0.06%
DCN				55.67±0.09%	72.51±0.07%
DCN		√		57.69±0.08%	76.38±0.06%
DCN			√	56.48±0.09%	72.50±0.07%
DCN		√	√	58.44±0.08%	77.07±0.06%
DCN	√			59.11±0.09%	72.22±0.07%
DCN	√	√		61.11±0.09%	76.62±0.07%
DCN	√		√	59.72±0.08%	73.81±0.07%
DCN	√	√	√	**61.64±0.08%**	**77.74±0.06%**

Finally, the results show that the performance improvement of ensemble learning on miniImageNet is more significant than that on Omniglot. This validates the hypothesis that a task with a larger generation gap between the training set and the validation set is more suitable for snapshot ensembles. In performing such tasks, the proposed approach is capable of obtaining models with higher quality and diversity.

V. Conclusion

This paper has proposed a metamodel with a decoder choice network and applied a decoder to control the learning process. Additionally, an innovative GLT structure has been included to reduce the computational time and space complexities. The results have shown that DCN is able to denote task-general information. Also, ensemble learning is integrated with DCN to improve the generalization ability of the learned model. Further experimental results have demonstrated the effectiveness of the approach in solving few-shot learning problems.

For future work, despite promising experimental results, it would be interesting to investigate whether, and how, this work may be improved by replacing the decoder networks with other types of neural network. Also, introducing an attention method into DCN may help to improve the network's efficiency; this forms a worthy piece of further research. Additionally, DCN may be applied to performing reinforcement learning tasks for quick adaptations. This is potentially very useful since learning environment information from a low-dimensional parameter space may enable the resulting deep reinforcement learning models to become more stable during training.

References

[1] Y. Huang, Y. Cheng, A. Bapna, O. Firat, D. Chen, M. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu *et al.*, "Gpipe: Efficient training of giant neural networks using pipeline parallelism," in *Advances in neural information processing systems*, 2019, pp. 103–112.

[2] C. Choy, J. Gwak, and S. Savarese, "4d spatio-temporal convnets: Minkowski convolutional neural networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 3075–3084.

[3] B. Singh, M. Najibi, and L. S. Davis, "Sniper: Efficient multi-scale training," in *Advances in neural information processing systems*, 2018, pp. 9310–9320.

[4] B. Zoph, G. Ghiasi, T.-Y. Lin, Y. Cui, H. Liu, E. D. Cubuk, and Q. V. Le, "Rethinking pre-training and self-training," *arXiv preprint arXiv:2006.06882*, 2020.

[5] H. Zhang, C. Wu, Z. Zhang, Y. Zhu, Z. Zhang, H. Lin, Y. Sun, T. He, J. Mueller, R. Manmatha *et al.*, "Resnest: Split-attention networks," *arXiv preprint arXiv:2004.08955*, 2020.

[6] L. Fei-Fei, R. Fergus, and P. Perona, "One-shot learning of object categories," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 594–611, 2006.

[7] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel, "Continuous adaptation via meta-learning in nonstationary and competitive environments," *arXiv preprint arXiv:1710.03641*, 2017.

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

[9] Z. Xu, X. Chen, and L. Cao, "Fast task adaptation based on the combination of model-based and gradient-based meta learning," *IEEE Transactions on Cybernetics*, pp. 1–10, 2020.

[10] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap, "Meta-learning with memory-augmented neural networks," in *International Conference on Machine Learning*, 2016, pp. 1842–1850.

[11] T. Munkhdalai and H. Yu, "Meta networks," *arXiv preprint arXiv:1703.00837*, 2017.

[12] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," *arXiv preprint arXiv:1703.03400*, 2017.

[13] A. Nichol and J. Schulman, "Reptile: a scalable metalearning algorithm," *arXiv preprint arXiv:1803.02999*, 2018.

[14] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," 2016.

[15] Z. Li, F. Zhou, F. Chen, and H. Li, "Meta-sgd: Learning to learn quickly for few shot learning," *arXiv preprint arXiv:1707.09835*, 2017.

[16] H. Zhu, L. Li, J. Wu, S. Zhao, G. Ding, and G. Shi, "Personalized image aesthetics assessment via meta-learning with bilevel gradient optimization," *IEEE Transactions on Cybernetics*, pp. 1–14, 2020.

[17] N. Bansal, X. Chen, and Z. Wang, "Can we gain more from orthogonality regularizations in training deep cnns?" in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. Curran Associates Inc., 2018, pp. 4266–4276.

[18] B. N. Oreshkin, A. Lacoste, and P. Rodriguez, "Tadam: Task dependent adaptive metric for improved few-shot learning," *arXiv preprint arXiv:1805.10123*, 2018.

[19] G. Huang, Y. Li, G. Pleiss, Z. Liu, J. E. Hopcroft, and K. Q. Weinberger, "Snapshot ensembles: Train 1, get m for free," *arXiv preprint arXiv:1704.00109*, 2017.

[20] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra *et al.*, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 3630–3638.

[21] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[22] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 1199–1208.

[23] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," *arXiv preprint arXiv:1707.03141*, 2018.

[24] T. Munkhdalai, X. Yuan, S. Mehri, and A. Trischler, "Rapid adaptation with conditionally shifted neurons," in *International Conference on Machine Learning*, 2018, pp. 3661–3670.

[25] A. Graves, G. Wayne, and I. Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

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

[27] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," *arXiv preprint arXiv:1511.07289*, 2015.

[28] N. Ketkar, "Introduction to pytorch," in *Deep Learning with Python*. Springer, 2017, pp. 195–208.

[29] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[30] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[31] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[32] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in Neural Information Processing Systems*, 2017, pp. 3856–3866.

[33] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[34] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 4700–4708.

[35] S. Qiao, C. Liu, W. Shen, and A. L. Yuille, "Few-shot image recognition by predicting parameters from activations," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7229–7238.

[36] M. Bauer, M. Rojas-Carulla, J. B. Świątkowski, B. Schölkopf, and R. E. Turner, "Discriminative k-shot learning using probabilistic models," *arXiv preprint arXiv:1706.00326*, 2017.

[37] S. Gidaris and N. Komodakis, "Dynamic few-shot visual learning without forgetting," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 4367–4375.

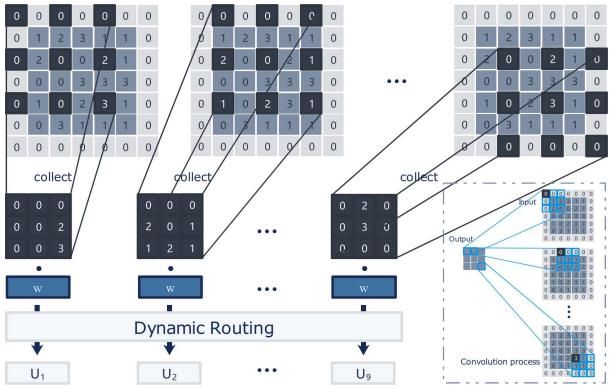

Fig. 6. An example of a 9×9 kernel with $C_{in} = 1$, where U_l is the output capsule. The kernel size, stride, zero-padding and channel size of the layer are 9×9, 2×2, 1×1 and 1, respectively. When the channel size is not equal to 1, w is not a vector but a matrix, and the number of U_l remains unchanged. The lower right corner of the graph is the operation process after the generated parameters. The convolution operation is consistent with standard a CNN.

[38] J. Liu, S. J. Gibson, and M. Osadchy, "Learning to support: Exploiting structure information in support sets for one-shot learning," *arXiv preprint arXiv:1808.07270*, 2018.

[39] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 33, no. 33, 2011.

[40] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[41] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

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

[43] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan *et al.*, "Population based training of neural networks," *arXiv preprint arXiv:1711.09846*, 2017.

[44] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *ICML Deep Learning Workshop*, vol. 2, 2015.

[45] H. Edwards and A. Storkey, "Towards a neural statistician," *arXiv preprint arXiv:1606.02185*, 2016.

[46] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4077–4087.

APPENDIX A

A. Task Feature

Suppose that the size of the input u of the network layer is $(N_d, C_{in}, W_{in}, H_{in})$, where N_d and C_{in} are the number of examples and that of channels, respectively, and W_{in} and H_{in} are the width and height, respectively. The output size is $(N_d, C_{ou}, W_{ou}, H_{ou})$. An example of the feature choice is shown in Fig. 6, where the kernel size is 9×9 and $C_{in} = 1$. The mathematical form of the channel e is given by,

$$\mathrm{u}_k^{ne} = \begin{bmatrix} u^{ne}_{h_1^k, w_1^k} & u^{ne}_{h_1^k, w_2^k} & \cdots & u^{ne}_{h_1^k, w_{W_{ou}}^k} \\ u^{ne}_{h_2^k, w_1^k} & u^{ne}_{h_2^k, w_2^k} & \cdots & u^{ne}_{h_2^k, w_{W_{ou}}^k} \\ \vdots & \vdots & \ddots & \vdots \\ u^{ne}_{h_{H_{ou}}^k, w_1^k} & u^{ne}_{h_{H_{ou}}^k, w_2^k} & \cdots & u^{ne}_{h_{H_{ou}}^k, w_{W_{ou}}^k} \end{bmatrix}, \quad (15)$$
$$(n = 1, 2, \cdots, N_d), (e = 1, 2, \cdots, C_{in}),$$

$$\mathrm{u}_k^e = \mathrm{vec}\left(\left[\mathrm{u}_k^{1e}, \mathrm{u}_k^{2e}, \cdots, \mathrm{u}_k^{Ne}\right]\right), \quad (16)$$

where e and k are the indexes of channel and kernel dimensions, respectively; $h_i^k = h_0^k + (i-1)str_1$, $(i = 1, 2, \cdots, H_{ou})$ and $w_i^k = w_0^k + (i-1)str_2$, $(i = 1, 2, \cdots, W_{ou})$; h_0^k and w_0^k are the starting coordinates of scanning; str_1 and str_2 are the strides in rows and columns, respectively; and $\mathrm{vec}()$ is the operation of pulling the matrix into a row vector. The weights of a capsule layer are shared by the same channel across different samples. Finally, the task features received by the capsule layer are given by:

$$\begin{aligned} \hat{\mathrm{u}}_k &= \mathrm{cat}\left(\mathrm{w}_k^1 \mathrm{u}_k^1, \mathrm{w}_k^2 \mathrm{u}_k^2, \cdots, \mathrm{w}_k^{C_{in}} \mathrm{u}_k^{C_{in}}\right) \\ &= \begin{bmatrix} \hat{u}_{1|1} & \hat{u}_{1|2} & \cdots & \hat{u}_{1|J} \\ \hat{u}_{2|1} & \hat{u}_{2|2} & \cdots & \hat{u}_{2|J} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{u}_{F_f|1} & \hat{u}_{F_f|2} & \cdots & \hat{u}_{F_f|J} \end{bmatrix}, \end{aligned} \quad (17)$$

where $\mathrm{cat}()$ denotes the concatenation of the matrix, F_f is the number of state variables, $J = H_{ou} W_{ou} N_d C_{in}$, and w_k^e is a column vector with a length of F_f.

B. Dynamic Routing

After obtaining the features, dynamic routing [32] is adopted to calculate the output capsule. The "prediction vector" [32] is given by:

$$\hat{\mathrm{u}}_{\cdot|j} = \left[\hat{u}_{1|j}, \hat{u}_{2|j}, \cdots, \hat{u}_{F_f|j}\right]^{\mathrm{T}}, \quad (18)$$

where $j = 1, 2, \cdots, J$. Each $\hat{\mathrm{u}}_{\cdot|j}$ accumulates its value according to the coupling coefficients, and nonlinear squashing is used to shrink the vector module to $0 \sim 1$:

$$\hat{\mathrm{s}} = \sum_j \mathrm{L}_j \circ \hat{\mathrm{u}}_{\cdot|j}, \quad (19)$$

$$\mathrm{v} = \frac{\|\hat{\mathrm{s}}\|^2}{1 + \|\hat{\mathrm{s}}\|^2} \frac{\hat{\mathrm{s}}}{\|\hat{\mathrm{s}}\|}, \quad (20)$$

where $\circ$ denotes element-wise multiplication. In this case, the input includes a J capsule, the length of which is 1, and the output includes the 1 capsule, the length of which is F_f; the squashing operation is applied to the entire output $\hat{\mathrm{s}}$. L_j is the coupling coefficient that is calculated by iterative dynamic routing [32] through:

$$L_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{kj})}, \quad (21)$$

where $\mathrm{L}_j = \{L_{ij}\}_i$ and $\mathrm{b} = \{b_{ij}\}_{ij}$ denotes the correlation between $\hat{\mathrm{u}}_l$ and v. The initial value of b is a zero matrix. After obtaining the output v, b can be recalculated as follows:

$$\mathrm{b}_{n\cdot} = \hat{\mathrm{u}}_{n|\cdot} v_n, \ (n = 1, 2, \cdots, F_f), \quad (22)$$

where $\mathrm{b}_{n\cdot}$ and $\hat{\mathrm{u}}_{n|\cdot}$ are the row vector of b and that of $\hat{\mathrm{u}}_k$, respectively, and v_n is an element of v. The process is repeated on the basis of Eq. (21) $\rightarrow$ (19) $\rightarrow$ (20) $\rightarrow$ (22) r times after initializing b. Following the original work of [32], $r = 3$ is applied in all the experiments.