Latent Weights Generating for Few Shot Learning Using Information Theory

Yiwei Zhang and **Zongyang Li** [1]

Abstract. Few shot image classification aims at learning a classifier from limited labeled data. Generating the classification weights has been applied in many metalearning approaches for few shot image classification due to its simplicity and effectiveness. However, fixed classification weights for different query samples within one task might be sub-optimal, due to the few shot challenge, and it is difficult to generate the exact and universal classification weights for all the diverse query samples from very few training samples. In this work, we introduce latent weights generating using information theory (LWGIT) for few shot learning which addresses current issues by generating different classification weights for different query samples by letting each of query samples attends to the whole support set. The experiment results demonstrate the effectiveness of LWGIT, thereby contributing to exceed the performances of the existing state-of-the-art models.

1 Introduction

While deep learning methods achieve great success in domains such as computer vision [1], natural language processing [2], reinforcement learning [3], their hunger for large amount of labeled data limits the application scenarios where only a few data are available for training. Humans, in contrast, are able to learn from limited data, which is desirable for deep learning methods. Few shot learning is thus proposed to enable deep models to learn from very few samples.

Meta learning is by far the most popular and promising approach for few shot problems [4]. In meta learning approaches, the model extracts high level knowledge across different tasks so that it can adapt itself quickly to a new-coming task [5]. There are several kinds of meta learning methods for few shot learning, such as gradient-based [4] and metric-based [6]. Weights generation, among these different methods, has shown effectiveness with simple formulation [7]. In general, weights generation methods learn to generate the classification weights for different tasks conditioned on the limited labeled data.

However, fixed classification weights for different query samples within one task might be sub-optimal, due to the few shot challenge, and it is difficult to generate the exact and universal classification weights for all the diverse query samples from very few training samples.

To addresses current issues, we propose latent weights generating using information theory (LWGIT) for few shot learning in this work.The contribution is as followed:

- To overcome issues mentioned above, we propose the LWGIT which generates different classification weights for different query samples by letting each of query samples attends to the whole support set.
- To guarantee the generated weights adaptive to different query sample, we re-formulate the problem to maximize the lower bound of mutual information between generated weights and query as well as support data.
- The experiment results demonstrate the effectiveness of LWGIT, thereby contributing to exceed the performances of the existing state-of-the-art models.

The remaining of this paper is organized as follows. Section 2 includes the related work. Section 3 introduces our proposed latent weights generating using information theory method. In section 4, we evaluate our proposed models and report experimental results on extensive realworld datasets. Section 5 concludes this work.

2 Related Work

2.1 Few Shot Learning

Learning from few labeled training data has received growing attentions recently. Most successful existing methods apply meta learning to solve this problem and can be divided into several categories. In the gradient-based approaches, an optimal initialization for all tasks is learned [4]. Ravi Larochelle [8] learned a meta-learner LSTM directly to optimize the given fewshot classification task. Sun et al. [9] learned the transformation for activations of each layer by gradients to better suit the current task.

In the metric-based methods, a similarity metric between query and support samples is learned [10]. Spatial information or local image descriptors are also considered in some works to compute richer similarities [11].

Generating the classification weights directly has been explored by some works. Gidaris [12] generated classification weights as linear combinations of weights for base and novel classes. Similarly, Qiao et al. [13] generated the classification weights from activations of a trained feature extractor. Graph neural network denoising autoencoders are used in [7]. Munkhdalai [14] proposed to generate fast weights from the loss gradient for each task. All these methods do not consider generating different weights for different query examples, nor maximizing the mutual information.

There are some other methods for few-shot classification. Generative models are used to generate or hallucinate more data in [15] used the closed-form solutions directly for few shot classification. Liu et al. [16] integrated label propagation on a transductive graph to predict the query class label.

[1] Kings college London, United Kingdom, email: yiwei.1.zhang@kcl.ac.uk

2.2 Attention mechanism

Attention mechanism shows great success in computer vision [17] and natural language processing [18]. It is effective in modeling the interaction between queries and key-value pairs from certain context. Based on the fact that keys and queries point to the same entities or not, people refer to attention as self attention or cross attention. In this work, we use both types of attention to encode the task and query-task information.

3 Latent weights generating using information theory

3.1 Background

Suppose that a sequence of tasks $\{\mathcal{T}_1, \ldots, T_{N_t}\}$ are sampled from an environment which is a probability distribution $\mathcal{E}$ on tasks. In each task $\mathcal{T}_i \sim \mathcal{E}$, we have a few examples $\{\mathbf{x}_{i,j}, \mathbf{y}_{i,j}\}_{j=1}^{n^{tr}}$ to constitute the training set $\mathcal{D}_{\mathcal{T}_i}^{tr}$ and the rest as the test set $\mathcal{D}_{\mathcal{T}_i}^{te}$.

Given a base learner f with θ as parameters, the optimal parameters $\theta_{\mathcal{T}_i}$ are learned to make accurate predictions, i.e., $f_{\theta_{\mathcal{T}_i}}(\mathbf{x}_{i,j}) \rightarrow \mathbf{y}_{i,j}$. The effectiveness of such a base learner on $\mathcal{D}_{\mathcal{T}_i}^{tr}$ is evaluated by the loss function $\mathcal{L}\left(f_{\theta_{\mathcal{T}_i}}, \mathcal{D}_{\mathcal{T}_i}^{tr}\right)$, which equals the mean square error for regression problems:

$$\sum_{\left(\mathbf{x}_{i,j}, \mathbf{y}_{i,j}\right) \in \mathcal{D}_{T_i}^{tr}} \left\| f_{\theta_{\mathcal{T}_i}}(\mathbf{x}_{i,j}) - \mathbf{y}_{i,j} \right\|_2^2 \tag{1}$$

or the cross entropy loss :

$$-\sum_{\left(\mathbf{x}_{i,j}, \mathbf{y}_{i,j}\right) \in \mathcal{D}_{\mathcal{T}_i}^{tr}} \log p\left(\mathbf{y}_{i,j} | \mathbf{x}_{i,j}, f_{\theta_{\mathcal{T}_i}}\right) \tag{2}$$

for classification problems.

The goal of meta-learning is to learn from previous tasks a well-generalized meta-learner $\mathcal{M}(\cdot)$ which can facilitate the training of the base learner in a future task with a few examples. In fulfillment of this, meta-learning involves two stages, i.e., meta-training and meta-testing.

During meta- training, the parameters of the base learner for all tasks, i.e., $\{\theta_{\mathcal{T}_i}\}_{i=1}^{N_t}$, and the meta-learner $\mathcal{M}(\cdot)$ are optimized alternatingly. In virtue of M, the parameters $\{\theta_{\mathcal{T}_i}\}_{i=1}^{N_t}$ are learned to minimize the expected empirical loss over training sets of all N_t historical tasks:

$$\min_{\{\theta_{\mathcal{T}_i}\}_{i=1}^{N_t}} \sum_{i=1}^{N_t} \mathcal{L}\left(\mathcal{M}\left(f_{\theta_{\mathcal{T}_i}}\right), \mathcal{D}_{\mathcal{T}_i}^{tr}\right) \tag{3}$$

In turn, a well-generalized M can be obtained by minimizing the expected empirical loss over test sets:

$$\min_{\mathcal{M}} \sum_{i=1}^{N_t} \mathcal{L}\left(\mathcal{M}\left(f_{\theta_{\mathcal{T}_i}}\right), \mathcal{D}_{\mathcal{T}_i}^{te}\right) \tag{4}$$

When it comes to the metatesting phase, provided with a future task $\mathcal{T}_t$, the learning effectiveness and efficiency are improved by applying the meta-learner M and solving

$$\min_{\theta_{\tau_t}} \mathcal{L}\left(\mathcal{M}\left(f_{\theta_{\tau_t}}\right), \mathcal{D}_{\mathcal{T}_t}^{tr}\right) \tag{5}$$

3.2 Problem formulation

Following many popular meta-learning methods for few shot classification, we formulate the problem under episodic training paradigm [4]. One N-way K-shot task sampled from an unknown task distribution $P(T)$ includes support set and query set:

$$\mathcal{T} = (\mathcal{S}, \mathcal{Q}) \tag{6}$$

where $\mathcal{S} = \left\{\left(\mathbf{x}^{c_n;k}, \mathbf{y}^{c_n;k}\right) | k = 1, \ldots, K; n = 1, \ldots, N\right\}$,$\mathcal{Q} = \left\{\left(\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_{[Q]}\right)\right\}$ Support set S contains NK labeled samples. Query set Q includes $\hat{\mathbf{x}}$ and we need to predict label $\hat{\mathbf{y}}$ for $\hat{\mathbf{x}}$ based on S. During meta-training, the meta-loss is estimated on Q to optimize the model. During metatesting, the performance of meta-learning method is evaluated on Q, provided the labeled S. The classes used in meta-training and meta-testing are disjoint so that the meta-learned model needs to learn the knowledge transferable across tasks and adapt itself quickly to novel tasks.

Our proposed approach follows the general framework to generate the classification weights [13]. In this framework, there is a feature extractor to output image feature embeddings. The meta-learner needs to generate the classification weights for different tasks

3.3 Latent embedding optimization

Latent Embedding Optimization (LEO) [19] is one of the weights generation methods that is most related to our work. In LEO, the latent code z is generated by h conditioned on support set S, described as $z = h(\mathcal{S})$. h is instantiated as relation networks [20]. Classification weights w can be decoded from z with l, $w = l(z)$. In the inner loop, we use w to compute the loss (usually cross entropy) on the support set and then update z:

$$z' = z - \eta \nabla_z \mathcal{L}_{\mathcal{S}}(w) \tag{7}$$

where L_S indicates that the loss is evaluated on S only. The updated latent code z' is used to decode new classification weights w' with generating function l. w' is adopted in the outer loop for query set Q and the objective function of LEO then can be written as

$$\min_{\theta} \mathcal{L}_{\mathcal{Q}}\left(w'\right) \tag{8}$$

Here θ stands for the parameters of h and l and we omit the regularization terms for clarity. LEO avoids updating high-dimensional w in the inner loop by learning a lower-dimensional latent space, from which sampled z can be used to generate w. The most significant difference between LEO and LWGIT is that we do not need inner updates to adapt the model. Instead, LWGIT is a feedforward network trained to maximize the mutual information so that it fits to different tasks well. On the other hand, LWGIT learns to generate optimal classification weights for each query sample while LEO generates fixed weights conditioned on the support set within one task.

3.4 Weights Generation Using Information Theory

The framework of our proposed method is shown in Figure 1. Assume that we have a feature extractor, which can be a simple 4-layer Convnet or a deeper Resnet. All the images included in the sampled task T are processed by this feature extractor and represented as d-dimensional vectors afterwards, i.e., $\mathbf{x}^{c_n;k}, \hat{\mathbf{x}} \in R^d$. There are two paths to encode the task context and the individual query sample respectively, which are called contextual path and attentive path. The outputs of both paths are concatenated together as input to the

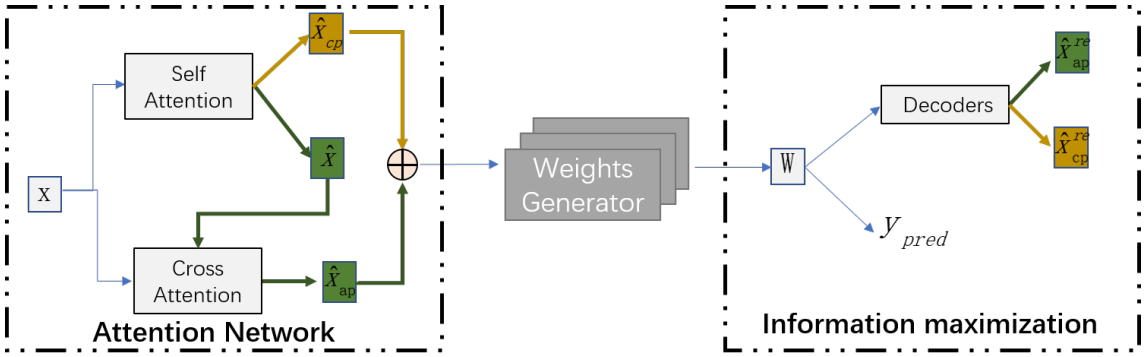

Figure 1. The structure of the proposed LWGIT

generator for classification weights. Generated classification weights are used to not only predict the label of $\hat{\mathbf{x}}$, but also maximize the lower bound of mutual information between itself and other variables, which will be discussed in the following section.

3.4.1 *Attention Network*

The encoding process includes two paths, namely the contextual path and attentive path. The contextual path aims at learning representations for only the support set with a multi-head self-attention network f_{sa}^{cp} [18]. The outputs of contextual path $\mathbf{X}^{cp} \in R^{NK \times d_h}$ thus contain richer information about the task and can be used later for weights generation.

Existing weights generation methods generate the classification weights conditioned on the support set only, which is equivalent to using contextual path. However, the classification weights generated in this way might be sub-optimal. This is because estimating the exact and universal classification weights from very few labeled data in the support set is difficult and sometimes impossible. The generated weights are usually in lack of adaptation to different query samples. We address this issue by introducing attentive path, where the individual query example attends to the task context and then is used to generate the classification weights. Therefore, the classification weights are adaptive to different query samples and aware of the task context as well.

In the attentive path, a new multi-head self-attention network f_{sa}^{ap} on the support set is employed to encode the global task information. f_{sa}^{ap} is different from f_{sa}^{cp} in contextual path because the self-attention network in contextual path emphasizes on generating the classification weights. On the contrary, outputs of self-attention here plays the role of providing the Value context for different query samples to attend in the following cross attention. Sharing the same self-attention networks might limit the expressiveness of learned representations in both paths. The cross attention network f_{ca}^{ap} applied on each query sample and task-aware support set is followed to produce $\hat{\mathbf{X}}^{ap} \in R^{|\mathcal{Q}| \times d_h}$.

We use multi-head attention with h heads in both paths. In one attention block, we produce h different sets of queries, keys and values. Multi-head attention is claimed to be able to learn more comprehensive and expressive representations from h different subspaces [18].

3.4.2 *Weights Generator*

We replicate $\mathbf{X}^{cp} \in R^{NK \times d_h}$ and $\hat{\mathbf{X}}^{ap} \in R|\mathcal{Q}| \times d_h$ for $|\mathcal{Q}|$ and NK times respectively and reshape them afterwards. Then we have $\mathbf{X}^{cp} \in R^{|Q| \times NK \times d_h}$ and $\hat{\mathbf{X}}^{ap} \in R^{|Q| \times NK \times d_h}$. These two tensors are concatenated to become $\mathbf{X}^{cp \oplus ap} \in R|\mathcal{Q}| \times NK \times 2d_h$, $\mathbf{X}^{cp \oplus ap}$ can be interpreted that each query sample has its own latent representations for support set to generate specific classification weights, which are both aware of the task-context and adaptive to individual query sample.

$\mathbf{X}^{cp \oplus ap}$ is decoded by the weights generator g $:R^{2d_h} \rightarrow R^{2d}$. We assume that the classification weights follow Gaussian distribution with diagonal covariance. g outputs the distribution parameters and we sample the weights from learned distribution during meta-training. The sampled classification weights are represented as $\mathbf{W} \in R|\mathcal{Q}| \times NK \times d$. To reduce complexity, we compute the mean value on K classification weights for each class to have $\mathbf{W}^{final} \in R^{|Q| \times N} \times d$. Therefore, ith query sample has its specific classification weight matrix $\mathbf{W}_{i,i,i}^{final} \in R^{N \times d}$. The prediction for query data can be computed by $\hat{\mathbf{X}}\mathbf{W}^{final\mathbf{T}}$. The support data X is replicated for $|Q|$ times and reshaped as $\mathbf{X}_s \in R|\mathbf{Q}| \times NK \times d$. So the prediction for support data can also be computed as $\mathbf{X}_s\mathbf{W}^{final\mathbf{T}}$.

Besides the weights generator g, we have another two decoders $r_1 : R^d \rightarrow R^{d_h}$ and $r_2 : R^d \rightarrow R^{d_h}$. They both take the generated weights W as inputs and learn to reconstruct $\mathbf{X}^{cp}$ and $\mathbf{X}^{ap}$ respectively. The outputs of r_1 and r_2 are denoted as $\mathbf{X}_{re}^{cp}, \hat{\mathbf{X}}_{re}^{ap} \in R^{|Q| \times NK \times d_h}$.

3.5 Information Theory

In this section, we perform the analysis for one query sample without loss of generality. The subscripts for classification weights are omitted for clarity. In general, we use $(\mathbf{x}, \mathbf{y})$ and $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ to represent support and query samples respectively.

Since the classification weights w generated from g are encoded with attentive path and contextual path, it is expected that we can directly have the query-specific weights. However, we show in the experiments that simply doing this does not outperform a weight generator conditioned only on the S significantly, which implies that the generated classification weights from two paths are not sensitive to different query samples. In other words, the information from attentive path is not kept well during the weights generation.

To address this limitation, we propose to maximize the mutual information between generated weights w and support as well as query

data. The objective function can be described as

$$\max I((\hat{\mathbf{x}}, \hat{\mathbf{y}}); \mathbf{w}) + \sum_{(\mathbf{x},y)\in\mathcal{S}} I((\mathbf{x}, \mathbf{y}); \mathbf{w}) \tag{9}$$

According to the chain rule of mutual information, we have

$$I((\hat{\mathbf{x}}, \hat{\mathbf{y}}); \mathbf{w}) = I(\hat{\mathbf{x}}; \mathbf{w}) + I(\hat{\mathbf{y}}; \mathbf{w}|\hat{\mathbf{x}}) \tag{10}$$

Equation 10 stands for both terms in 9. So the objective function can be written as

$$\max I(\hat{\mathbf{x}}; \mathbf{w}) + I(\hat{\mathbf{y}}; \mathbf{w}|\hat{\mathbf{x}}) + \sum_{(\mathbf{x},y)\in\mathcal{S}} [I(\mathbf{x}; \mathbf{w}) + I(\mathbf{y}; \mathbf{w}|\mathbf{x})] \tag{11}$$

Directly computing the mutual information in Equation 11 is intractable since the true posteriori distributions like $p(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w}), p(\hat{\mathbf{x}}|\mathbf{w})$ are still unknown. Therefore, we use Variational Information Maximization [21] to compute the lower bound of Equation 9. We use $p_\theta(\hat{\mathbf{x}}|\mathbf{w})$ to approximate the true posteriori distribution, where θ represents the model parameters. As a result, we have

$$\begin{aligned} I(\hat{\mathbf{x}}; \mathbf{w}) &= H(\hat{\mathbf{x}}) - H(\hat{\mathbf{x}}|\mathbf{w}) \\ &= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\mathbf{x},\mathcal{S})} \left[\mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w})}[\log p(\hat{\mathbf{x}}|\mathbf{w})]\right] \\ &= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\mathbf{x},\mathcal{S})} D_{\mathbf{KL}}\left(p(\hat{\mathbf{x}}|\mathbf{w})||p_\theta(\hat{\mathbf{x}}|\mathbf{w})\right) \\ &\quad + \mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w})}\left[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w})\right] \\ &\geq H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\mathbf{x},\mathcal{S})} \left[\mathbb{E}_{\hat{\mathbf{x}}\sim p(\mathbf{x}|\mathbf{w})}\left[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w})\right]\right] \end{aligned} \tag{12}$$

$H(\cdot)$ is the entropy of a random variable. $H(\hat{\mathbf{x}})$ is a constant value for given data. We can maximize this lower bound as the proxy for the true mutual information. Similar to $I(\hat{\mathbf{x}}; \mathbf{w})$

$$\begin{aligned} I(\hat{\mathbf{y}}; \mathbf{w}|\hat{\mathbf{x}}) &\geq H(\hat{\mathbf{y}}|\hat{\mathbf{x}}) \\ &\quad + \mathbb{E}_{\mathbf{w}\sim p(\mathbf{w}|\hat{\mathbf{x}},\mathcal{S})}\left[\mathbb{E}_{\hat{\mathbf{y}}\sim p(\hat{\mathbf{y}}|\mathbf{x},\mathbf{w})}\left[\log p_\theta(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w})\right]\right] \end{aligned} \tag{13}$$

$$\begin{aligned} \sum_{(\mathbf{x},y)\in\mathcal{S}} I((\mathbf{x}, \mathbf{y}); \mathbf{w}) &\geq \sum_{(\mathbf{x},y)\in\mathcal{S}} H((\mathbf{x}, \mathbf{y})) \\ &\quad + \mathbb{E}_{(\mathbf{x},y)\sim p((\mathbf{x},y)|\mathbf{w})}\left[\log p_\theta(\mathbf{x}|\mathbf{w}) + \log p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{w})\right] \end{aligned} \tag{14}$$

$p_\theta(\hat{\mathbf{x}}|\mathbf{w}), p_\theta(\mathbf{x}, \mathbf{y}|\mathbf{w})$ are used to approximate the true posteriori distribution $p(\hat{\mathbf{x}}|\mathbf{w})$ and $p(\mathbf{x}, \mathbf{y}|\mathbf{w})$

Put the lower bounds back into Equation 11. Omit the constant entropy terms and the expectation subscripts for clarity, we have the new objective function as

$$\begin{aligned} &\max \mathbb{E}_\theta\left[\log p_\theta(\hat{\mathbf{y}}|\hat{\mathbf{x}}, \mathbf{w})\right] \\ &\quad + \mathbb{E}_\theta\left[\log p_\theta(\mathbf{y}|\mathbf{x}, \mathbf{w}) + \log p_\theta(\mathbf{x}|\mathbf{w}) + \log p_\theta(\hat{\mathbf{x}}|\mathbf{w})\right] \end{aligned} \tag{15}$$

The first two terms are maximizing the log likelihood of label for both support and query data with respective to the network parameters, given the generated classification weights. This is equivalent to minimizing the cross entropy between prediction and ground-truth. We assume that $p_\theta(\hat{\mathbf{x}}|\mathbf{w})$ and $p_\theta(\mathbf{x}|\mathbf{w})$ are Gaussian distributions. r_1 and r_2 are used to approximate the mean of these two Gaussian distributions. Therefore maximizing the log likelihood is equivalent to reconstruct $\mathbf{X}^{Cp}$ and $\hat{\mathbf{X}}^{ap}$ with L2 loss. Thus the loss function to train the network can be written as

$$\begin{aligned} L &= \mathrm{CE}\left(\hat{\mathbf{y}}_{pred}, \hat{\mathbf{y}}\right) + \lambda_1 \sum_{\mathbf{y}\in\mathcal{S}} \mathrm{CE}\left(\mathbf{y}_{pred}, \mathbf{y}\right) \\ &\quad + \lambda_2 \sum_{\mathbf{x}^{cp}\in\mathcal{S}} \left\|\mathbf{x}^{cp} - \mathbf{x}^{cp}_{re}\right\|_2 + \lambda_3 \left\|\hat{\mathbf{x}}^{ap} - \hat{\mathbf{x}}^{ap}_{re}\right\|_2 \end{aligned} \tag{16}$$

CE here stands for cross entropy. $\mathbf{x}^{cp}$ and $\hat{\mathbf{x}}^{ap}$ are the inputs to weights generator g. $\mathbf{x}^{cp}_{re} \sim p_\theta(\mathbf{x}|\mathbf{w})$ and $\hat{\mathbf{x}}^{ap}_{re} \sim p_\theta(\hat{\mathbf{x}}|\mathbf{w})$ are the reconstruction of $\mathbf{x}^{cp}$ and $\hat{\mathbf{x}}^{ap}$. Since we convert the log likelihood in Equation 15 to mean square error or cross entropy in Equation 16 to optimize, the value of each term in Equation 16 is not equal to real log likelihood and we have to decide the weightage for each one. $\lambda_1, \lambda_2, \lambda_3$ are thus hyper-parameters for trade-off of different terms. With the help of last three terms, the generated classification weights are forced to carry information about the support data and the specific query sample.

In LEO [19], the inner update loss is computed as cross entropy on support data. If we merge the inner update into outer loop, then the loss becomes the summation of first two terms in Equation 16. However, the weight generation in LEO does not involve specific query samples, thus making reconstructing $\hat{\mathbf{X}}^{ap}$ impossible. In this sense, LEO can be regarded as a special case of our proposed method, where (1) only contextual path exits and (2) $\lambda_2 = \lambda_3 = 0$.

Model	Feature Extractor	5-way 1-shot	5-way 5-shot
MAML [4]	Conv-4	51.67	70.3
Prototypical Nets [22]	Conv-4	53.31	72.69
Relation Nets [6]	Conv-4	54.48	71.32
TPN [16]	Conv-4	59.91	72.85
MetaOptNet [23]	Resnets-12	65.81	81.75
LEO [19]	WRN-28-10	66.33	81.44
LWGIT (ours)	WRN-28-10	67.46	82.57

Table 1. Accuracy comparison with other approaches on tieredImageNet.

4 Experiments

4.1 Datasets And Protocols

We conduct experiments on miniImageNet [24] and tieredImageNet [26], two commonly used benchmark datasets, to compare with other methods and analyze our model. Both datasets are subsets of ILSVRC-12 dataset. miniImageNet contains 100 randomly sampled classes with 600 images per class. We follow the train/test split in [8], where 64 classes are used for meta-training, 16 for meta-validation and 20 for meta-testing. tieredImageNet is a larger dataset compared to miniImageNet. There are 608 classes and 779,165 images in total. They are selected from 34 higher level nodes in ImageNet [27] hierarchy. 351 classes from 20 high level nodes are used for meta-training, 97 from 6 nodes for meta-validation and 160 from 8 nodes for meta-testing.

We use the image features similar to LEO [19]. They trained a 28-layer Wide Residual Network [28] on the meta-training set. Each image then is represented by a 640 dimensional vector, which is used as the input to our model. For N-way K-shot experiments, we randomly sample N classes from meta-training set and each of them contains K samples as the support set and 15 as query set. Similar to other works, we train 5-way 1-shot and 5-shot models on two dataset. During meta-testing, 600 N-way K-shot tasks are sampled from meta-testing set and the average accuracy for query set is reported with 95confidence interval, as done in recent works [4, 19].

4.2 Few shot image classification

We compare the performance of our approach LWGIT on two datasets with several state-of-the-art methods proposed in recent

Model	miniImageNet		tieredImageNet	
	5-way 1-shot	5-way 5-shot	5-way 1-shot	5-way 5-shot
LEO	61.76%	77.59%	66.33%	81.44%
Generator in LEO	60.33%	74.53%	65.17%	78.77%
Generator conditioned on S only	61.02%	74.33%	66.22%	79.66%
Generator conditioned on S with IM	62.04%	77.54%	66.43%	81.73%
MLP encoding, $\lambda_1 = \lambda_2 = \lambda_3 = 0$	58.95%	71.68%	63.92%	75.80%
MLP encoding	62.26%	76.91%	65.84%	79.24%
$\lambda_1 = \lambda_2 = \lambda_3 = 0$	61.61%	74.14%	65.65%	79.93%
$\lambda_1 = \lambda_2 = 0$	62.06%	74.18%	65.85%	80.42%
$\lambda_3 = 0$	62.91%	77.88%	67.27%	81.67%
$\lambda_1 = 0$	62.19%	74.21%	66.82%	80.61%
$\lambda_2 = \lambda_3 = 0$	62.12%	77.65%	66.86%	81.03%
random shuffle in class	62.87%	77.48%	67.52%	82.55%
random shuffle between classes	61.20%	77.48%	66.55%	82.53%
LWGIT (ours)	63.42%	78.44%	67.58%	82.04%

Table 2. Analysis of our proposed LWGIT. In the top half, the attentive path is removed to compare with LEO. In the bottom part, ablation analysis with respective to different components is provided.

Model	Feature Extractor	5-way 1-shot	5-way 5-shot
Matching Networks [24]	Conv-4	46.60%	60.00%
MAML [4]	Conv-4	48.70%	63.11%
Meta LSTM [8]	Conv-4	43.44%	60.60%
Prototypical Nets [22]	Conv-4	49.42%	68.20%
Relation Nets [6]	Conv-4	50.44%	65.32%
SNAIL [25]	Resnets-12	55.71%	68.88%
TPN [16]	Resnets-12	59.46%	0.00%
MTL [9]	Resnets-12	61.20%	75.50%
Dynamic [12]	WRN-28-10	60.06%	76.39%
Prediction [13]	WRN-28-10	59.60%	73.74%
DAE-GNN [7]	WRN-28-10	62.96%	78.85%
LEO [19]	WRN-28-10	61.76%	77.59%
LWGIT (ours)	WRN-28-10	62.27%	78.74%

Table 3. Accuracy comparison with other approaches on miniImageNet

years. The results of MAML, Prototypical Nets, Relation Nets on tieredImageNet are evaluated by [16]. The results of Dynamic on miniImageNet with WRN-28-10 as the feature extractor is reported in [7]. The other results are reported in the corresponding original papers. We also include the backbone network structure of the used feature extractor for reference. The results on miniImageNet and tieredImageNet are shown in Table 1 and 2 respectively.

The top half parts of Table 1 and 2 display the methods belonging with different meta learning categories, such as metric-based(Matching Networks, Prototypical Nets), gradient-based (MAML, MTL), graph-based (TPN). The bottom part shows the classification weights generation approaches including Dynamic, Prediction, DAE-GNN, LEO and our proposed LWGIT.

LWGIT can outperform all the methods in top parts of two table. Comparing with other classification weights generation methods in the bottom part, LWGIT still shows very competitive performance, namely the best on tieredImageNet and close to the state-of-the-art on miniImageNet. We note that all the classification weights generation methods are using WRN-28-10 as backbone network, which makes the comparison fair. In particular, LWGIT can outperform LEO in all settings.

4.3 Analysis

We perform detailed analysis on LWGIT, shown in Table 3. We include the results of LEO Rusu et al. (2019) for reference. Generator in LEO means that there is no inner update in LEO. In the upper part of the table, we first studied the effect of attentive path. We implemented two generators including only the contextual path during encoding. Generator conditioned on S with IM indicates that we add the cross entropy loss and reconstruction loss for support set. It can be observed that Generator conditioned on S only is trained with cross entropy on query set, which is similar to Generator in LEO without inner update. It is able to achieve similar or slightly better results than Generator in LEO, which implies that self-attention is no worse than relation networks used in LEO to model task-context. With information maximization, our generator is able to obtain slightly better performance than LEO.

The effect of attention is investigated by replacing the attention modules with 2-layer MLPs, which is shown as MLP encoding. More specifically, one MLP in contextual path is used for support set and another MLP in attentive path for query samples. We can see that even without attention to encode the task-contextual information, MLP encoding can achieve accuracy close to LEO, for the sake of information maximization. However, if we let $\lambda_1 = \lambda_2 = \lambda_3 = 0$ for MLP encoding, the performance drops significantly, which demonstrates the importance of maximizing the information

We conducted ablation analysis with respective to $\lambda_1, \lambda_2, \lambda_3$ to investigate the effect of information maximization. First,$\lambda_1, \lambda_2, \lambda_3$ are all set to be 0. In this case, the accuracy is similar to generator conditioned on S only, showing that the generated classification weights are not fitted for different query samples, even with the attentive path. It can also be observed that maximizing the mutual information between weights and support is more crucial since $\lambda_1 = \lambda_2 = 0$ degrades accuracy significantly, comparing with $\lambda_3 = 0$. We further investigate the relative importance of the classification on support as well as reconstruction. $\lambda_1 = 0$ affects the performance noticeably. We conjecture that the support label prediction is more critical for information maximization.

The classification weights are generated specifically for each query sample in LWGIT. To this point, we shuffle the classification weights between query samples within the same classes and between

different classes as well to study whether the classification weights are adapted for different query samples. Assume there are T query samples per class in one task. $W^{\text{final}} \in \mathbb{R}|\mathcal{Q}| \times N \times d$ can be reshaped into $W^{\text{final}} \in \mathbb{R}^{N \times T \times N \times d}$. Then we shuffle this weight tensor along the first and second axis randomly. The results are shown as random shuffle between classes and random shuffle in class in Table 3. For 5-way 1-shot experiments, the random shuffle between classes degrades the accuracy noticeably while the random shuffle in class dose not affect too much. This indicates that when the support data are very limited, the generated weights for query samples from the same class are very similar to each other while distinct for different classes. When there are more labeled data in support set, two kinds of random shuffle show very close or even the same results in 5-way 5-shot experiments, which are both worse than the original ones. This implies that the generated classification weights are more diverse and specific for each query sample in 5-way 5-shot setting. The possible reason is that larger support set provides more knowledge to estimate the optimal classification weights for each query example.

5 CONCLUSION

In this work, we introduce latent weights generating using information theory(LWGIT) for few shot learning. LWGIT learns to generate optimal classification weights for each query sample within the task by two encoding paths. To guarantee this, the lower bound of mutual information between generated weights and query, support data is maximized. The effectiveness of LWGIT is demonstrated by state-of-the-art performance on two benchmark datasets.

REFERENCES

[1] 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.

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

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

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

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

[6] 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.

[7] S. Gidaris and N. Komodakis, "Generating classification weights with gnn denoising autoencoders for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 21–30.

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

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

[10] H. Li, D. Eigen, S. Dodge, M. Zeiler, and X. Wang, "Finding task-relevant features for few-shot learning by category traversal," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 1–10.

[11] Y. Lifchitz, Y. Avrithis, S. Picard, and A. Bursuc, "Dense classification and implanting for few-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 9258–9267.

[12] 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.

[13] 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.

[14] T. Munkhdalai and H. Yu, "Meta networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 2554–2563.

[15] Z. Chen, Y. Fu, Y.-X. Wang, L. Ma, W. Liu, and M. Hebert, "Image deformation meta-networks for one-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 8680–8689.

[16] Y. Liu, J. Lee, M. Park, S. Kim, E. Yang, S. J. Hwang, and Y. Yang, "Learning to propagate labels: Transductive propagation network for few-shot learning," *arXiv preprint arXiv:1805.10002*, 2018.

[17] N. Parmar, A. Vaswani, J. Uszkoreit, Ł. Kaiser, N. Shazeer, A. Ku, and D. Tran, "Image transformer," *arXiv preprint arXiv:1802.05751*, 2018.

[18] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[19] 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.

[20] A. Santoro, D. Raposo, D. G. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap, "A simple neural network module for relational reasoning," in *Advances in neural information processing systems*, 2017, pp. 4967–4976.

[21] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, "Infogan: Interpretable representation learning by information maximizing generative adversarial nets," in *Advances in neural information processing systems*, 2016, pp. 2172–2180.

[22] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Advances in neural information processing systems*, 2017, pp. 4077–4087.

[23] K. Lee, S. Maji, A. Ravichandran, and S. Soatto, "Meta-learning with differentiable convex optimization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10 657–10 665.

[24] 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.

[25] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel, "A simple neural attentive meta-learner," *arXiv preprint arXiv:1707.03141*, 2017.

[26] M. Ren, E. Triantafillou, S. Ravi, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel, "Meta-learning for semi-supervised few-shot classification," *arXiv preprint arXiv:1803.00676*, 2018.

[27] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 2009, pp. 248–255.

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