Attentive Weights Generation for Few Shot Learning via Information Maximization

Yiluan Guo, Ngai-Man Cheung
Singapore University of Technology and Design
yiluan_guo@mymail.sutd.edu.sg, ngaiman_cheung@sutd.edu.sg

Abstract

*Few shot image classification aims at learning a classifier from limited labeled data. Generating the classification weights has been applied in many meta-learning methods for few shot image classification due to its simplicity and effectiveness. In this work, we present **A**ttentive **W**eights **G**eneration for few shot learning via **I**nformation **M**aximization (AWGIM), which introduces two novel contributions: i) Mutual information maximization between generated weights and data within the task; this enables the generated weights to retain information of the task and the specific query sample. ii) Self-attention and cross-attention paths to encode the context of the task and individual queries. Both two contributions are shown to be very effective in extensive experiments. Overall, AWGIM is competitive with state-of-the-art. Code is available at `https://github.com/Yiluan/AWGIM`.*

1. Introduction

While deep learning methods achieve great success in computer vision [14], natural language processing [9], reinforcement learning [38], their hunger for large amount of labeled data limits the application scenarios where only 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 [10].

Meta learning is a promising approach for few shot problems [43, 11, 39, 33, 36]. 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 [37, 2]. There are several kinds of meta learning methods for few shot learning, such as gradient-based [11, 33] and metric-based [39, 41]. Weights generation, among these different methods, has shown effectiveness with simple formulation [31, 32, 12, 13]. 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 could be sub-optimal.

In this work, we present **A**ttentive **W**eights **G**eneration for few shot learning via **I**nformation **M**aximization (AWGIM) to address the limitation. AWGIM models the probability distribution of classification weights conditioned on the whole support set and individual query sample. In our model, two paths composed of attention blocks are employed to encode contextual and query-specific information. However, we show in experiments that cross attention between query samples and support set is not adequate to generate classification weights that are adaptive to diverse query data. In particular, some query-specific information is lost during weights generation.

To address this issue, we take inspiration from InfoGAN[6]. In particular, when training GAN, [6] proposes to learn disentangled representation by maximizing the mutual information (MI) between a structured latent code and the generator output. The MI maximization helps retain the information of the structured latent code in the generator output. In a similar spirit, we apply MI maximization in order to retain the information of query/support samples in the generated weights. **Our contributions are**:

- We solve the weights generation problem for few shot classification by maximizing the mutual information between generated weights and support/query data. With MI maximization, the weights generator is able to generate classification weights that adapt to diverse query samples.
- We propose to encode the task context and individual query sample in two separate paths. Attention mechanism is applied in both paths to capture the context information.
- We conduct extensive experiments and show that AWGIM compares favorably to state-of-the-art methods. We also conduct detailed analysis to validate the contribution of each component in AWGIM. The in-

duced computational overhead is minimal due to the nature of few shot problems. Adaptive classification weights generating also results in faster convergence.

2. Related Works

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 [11]. [33] learned a meta-learner LSTM directly to optimize the given few shot classification task. [40] 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. [19, 43, 39, 41, 23]. Spatial information or local image descriptors are also considered in some works to compute richer similarities [25, 24, 45].

Generating the classification weights directly has been explored by some works. [12] generated classification weights as linear combinations of weights for base and novel classes. Similarly, [32] and [31] both generated the classification weights from activations of a trained feature extractor. Graph neural network denoising auto-encoders are used in [13]. [29] 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 MI.

There are some other methods for few shot classification. Generative models are used to generate or hallucinate more data in [50, 44, 7]. [5] and [21] used the closed-form solutions directly. [26] integrated label propagation on a transductive graph to predict the query class label.

2.2. Attention

Attention mechanism shows great success in computer vision [46, 30] and natural language processing [3, 42]. 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. The most similar work is Attentive Neural Processes [17], which also employ self and cross attention. However, we are using attention for few shot image classification via maximizing the MI. In stark contrast, [17] works on regression from the perspective of stochastic processes and the variational objective is optimized. There are some works [47, 16] employing spatial attention to enhance features while we do not rely on spatial cues and focus on modeling the interactions with self/cross attention.

2.3. Mutual Information

Given two random variables x and y, mutual information $I(\mathrm{x};\mathrm{y})$ measures the decrease of uncertainty in one random variable when another is known. It is defined as the Kullback-Leibler divergence between joint distribution $p(\mathrm{x},\mathrm{y})$ and product of marginal distributions $p(\mathrm{x}) \otimes p(\mathrm{y})$,

$$I(\mathrm{x};\mathrm{y}) = D_{\mathrm{KL}}(p(\mathrm{x},\mathrm{y}) \| p(\mathrm{x}) \otimes p(\mathrm{y})). \tag{1}$$

When x and y are independent, $p(\mathrm{x},\mathrm{y}) = p(\mathrm{x}) \otimes p(\mathrm{y})$ so that $I(\mathrm{x},\mathrm{y}) = 0$, indicating that knowing x does not reveal any information about y. When y is a deterministic function of x, $I(\mathrm{x},\mathrm{y})$ achieves its maximum value. MI has been widely applied in Generative Adversarial Networks [6], self-supervised learning [15], visual question generation [20], etc.. Recently, MI is introduced in few shot learning as a regularization for memorization problem [48]. Specifically, MI between query labels and support data is maximized or MI between query labels and meta parameters is minimized. In [22], MI between learned binary codes and labels are maximized with a closed-form solution. Instead, we solve the few shot classification problem directly by generating accurate weights with variational lower bound of MI.

3. Proposed Method

In this section, we provide the problem formulation. We then discuss the most related work and reveal its limitations. We derive our objective function from theoretical analysis in Section 3.3. The overall model is detailed in Section 3.4.

3.1. Problem Formulation

Following many popular meta-learning methods for few shot classification, we formulate the problem under episodic training paradigm [43, 11]. One N-way K-shot task $\mathcal{T}$ sampled from an unknown task distribution $P(\mathcal{T})$ includes support set and query set:

$$\mathcal{T} = (\mathcal{S}, \mathcal{Q}), \tag{2}$$

where $\mathcal{S} = \{(\mathbf{x}_{c_n}^k, \mathbf{y}_{c_n}^k) | k = 1, ..., K; n = 1, ..., N\}$, $\mathcal{Q} = \{(\hat{\mathbf{x}}_1, ..., \hat{\mathbf{x}}_{|\mathcal{Q}|})\}$. Support set $\mathcal{S}$ contains NK labeled samples. Query set $\mathcal{Q}$ includes $\hat{\mathbf{x}}$ and we need to predict label $\hat{\mathbf{y}}$ for $\hat{\mathbf{x}}$ based on $\mathcal{S}$. In the following discussion, we use $(\mathbf{x}_{c_n}, \mathbf{y}_{c_n})$ and $(\hat{\mathbf{x}}, \hat{\mathbf{y}})$ to represent support (from class c_n) and query samples respectively. During meta-testing, the performance of meta-learning method is evaluated on $\mathcal{Q}$, provided the labeled $\mathcal{S}$. The classes used in meta-training and meta-testing are disjoint so that the meta-learned model needs to learn the high level knowledge transferable across tasks and adapt itself quickly to novel tasks.

Our proposed approach follows the general framework to generate the classification weights [31, 32, 36, 12, 13]. In

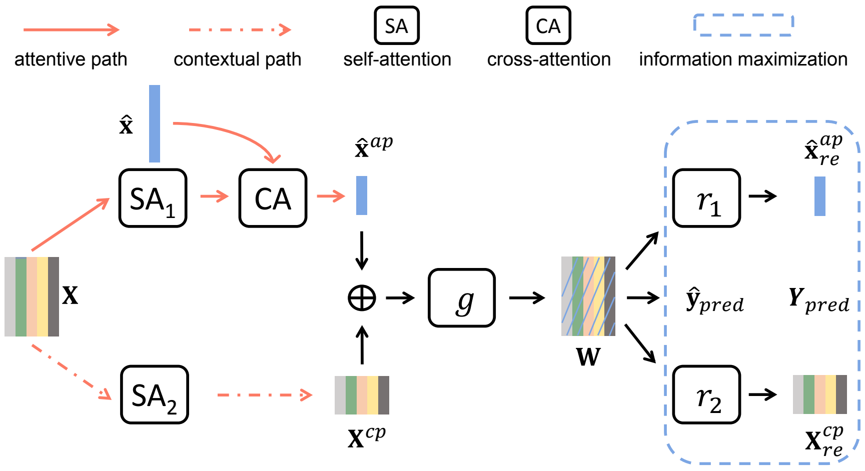

Figure 1. The overview of our proposed AWGIM. The input task is 5-way 1-shot with $\mathbf{X}$ as support set and $\hat{\mathbf{x}}$ as one query example. Different colors of the data in support set indicate different categories. The encoding process in contextual path produces context-aware support representations $\mathbf{X}^{cp}$. Similarly, the attentive path enables the query sample $\hat{\mathbf{x}}$ to be equipped with task knowledge. Both paths are achieved by attention mechanism. $\hat{\mathbf{x}}^{ap}$ is repeated to concatenate with $\mathbf{X}^{cp}$. The weight generator g takes these concatenated representations as input to generate classification weights $\mathbf{W}$ specific for $\hat{\mathbf{x}}$, denoted by the colorful matrix with slash. $\mathbf{W}$ can be used to predict the class label for $\hat{\mathbf{x}}$ and $\mathbf{X}$. $\mathbf{W}$ is also used to reconstruct the inputs of the generator g by two networks r_1 and r_2. In this way, the lower bound of mutual information is maximized and g is forced to generate classification weights sensitive to different query samples. During meta-testing, r_1 and r_2 are discarded.

this framework, there is a feature extractor to output image feature embeddings. The meta-learner needs to generate the classification weights for different tasks.

3.2. Latent Embedding Optimization

Latent Embedding Optimization (LEO) [36] is one of the weights generation methods that is most related to our work. In LEO, a latent code $\boldsymbol{z}$ is generated by u conditioned on $\mathcal{S}$, described as $\boldsymbol{z} = u(\mathcal{S})$. Classification weights $\mathbf{w}$ can be decoded from $\boldsymbol{z}$ with generating function v, $\mathbf{w} = v(\boldsymbol{z})$. In the inner loop, $\mathbf{w}$ is used to compute the loss (usually cross entropy) on the support set and then update $\boldsymbol{z}$:

$$\boldsymbol{z}' = \boldsymbol{z} - \eta \nabla_{\boldsymbol{z}} L_{\mathcal{S}}(\mathbf{w}), \tag{3}$$

where $L_{\mathcal{S}}$ indicates that the loss is evaluated on $\mathcal{S}$ only. The updated latent code $\boldsymbol{z}'$ is used to decode new classification weights $\mathbf{w}'$ with v. $\mathbf{w}'$ is adopted in the outer loop for query set $\mathcal{Q}$ and the objective function of LEO then can be written as

$$\min_{\theta} L_{\mathcal{Q}}(\mathbf{w}'), \tag{4}$$

where θ stands for the parameters of u and v. LEO avoids updating high-dimensional $\mathbf{w}$ in the inner loop by learning a lower-dimensional latent space, from which sampled $\boldsymbol{z}$ can be used to generate $\mathbf{w}$.

There are two significant differences between LEO and AWGIM. First, LEO relies on inner update (Equation 3) to guide v to generate the weights suitable for the input tasks. Instead, AWGIM is a feedforward network trained to maximize the MI so that it fits to different tasks well. Second, AWGIM learns to generate optimal classification weights for each query sample while LEO generates fixed weights conditioned on the support set of one task.

3.3. Information Maximization for Weights Generation

Our goal is to generate classification weights for one sampled task with few labeled training data. In other words, we want to define model $p(\mathbf{w}|\mathcal{T})$ for one task $\mathcal{T}$. It is noted that the classification weights generated in LEO are not sensitive to different query samples, which are also part of the task $\mathcal{T}$. To remedy this problem, we could encode the query-specific information during generation of weights and learn the model $p(\mathbf{w}|\hat{\mathbf{x}}, \mathcal{S})$ instead. However, information on $\hat{\mathbf{x}}$ might be ignored during generation, which has been observed in the experiments.

To address this limitation, we propose to maximize the MI between generated weights $\mathbf{w}$ and query as well as support data. Without loss of generality, we consider classification weight $\mathbf{w}_i$ for class c_i in the following discussion. The objective function can be described as

$$\max I((\hat{\mathbf{x}}, \hat{\mathbf{y}}); \mathbf{w}_i) + \frac{1}{K}\sum_{K} I((\mathbf{x}_{c_i}, \mathbf{y}_{c_i}); \mathbf{w}_i). \tag{5}$$

According to the chain rule of MI, we have

$$I((\hat{\mathbf{x}}, \hat{\mathbf{y}}); \mathbf{w}_i) = I(\hat{\mathbf{x}}; \mathbf{w}_i) + I(\hat{\mathbf{y}}; \mathbf{w}_i|\hat{\mathbf{x}}). \tag{6}$$

Equation 6 holds for both terms in 5. So the objective function can be written as

$$\max I(\hat{\mathbf{x}};\mathbf{w}_i)+I(\hat{\mathbf{y}};\mathbf{w}_i|\hat{\mathbf{x}})+\frac{1}{K}\sum_K[I(\mathbf{x}_{c_i};\mathbf{w}_i)+I(\mathbf{y}_{c_i};\mathbf{w}_i|\mathbf{x}_{c_i})]. \tag{7}$$

Directly computing the MI in Equation 7 is intractable since the true posteriori distributions like $p(\hat{\mathbf{y}}|\hat{\mathbf{x}},\mathbf{w}_i)$, $p(\hat{\mathbf{x}}|\mathbf{w}_i)$ are still unknown. Therefore, we use Variational Information Maximization [4, 6] to compute the lower bound of Equation 5. We use $p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)$ to approximate the true posteriori distribution, where θ represents the model parameters. As a result, we have

$$\begin{aligned}
I(\hat{\mathbf{x}};\mathbf{w}_i) &= H(\hat{\mathbf{x}}) - H(\hat{\mathbf{x}}|\mathbf{w}_i)\\
&= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}_i\sim p(\mathbf{w})}[\mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w}_i)}[\log p(\hat{\mathbf{x}}|\mathbf{w}_i)]]\\
&= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}_i\sim p(\mathbf{w})}[D_{\mathrm{KL}}(p(\hat{\mathbf{x}}|\mathbf{w}_i)\|p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i))\\
&\quad +\mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w}_i)}[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)]]\\
&\geq H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}_i\sim p(\mathbf{w})}[\mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}}|\mathbf{w}_i)}[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)]]\\
&= H(\hat{\mathbf{x}}) + \mathbb{E}_{\mathbf{w}_i,\hat{\mathbf{x}}\sim p(\mathbf{w},\hat{\mathbf{x}})}[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)]\\
&= H(\hat{\mathbf{x}}) + \mathbb{E}_{\hat{\mathbf{x}}\sim p(\hat{\mathbf{x}})}[\mathbb{E}_{\mathbf{w}_i\sim p(\mathbf{w}|\hat{\mathbf{x}})}[\log p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)]]
\end{aligned} \tag{8}$$

$H(\cdot)$ 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 MI.

Similar to $I(\hat{\mathbf{x}};\mathbf{w}_i)$,

$$\begin{aligned}
I(\hat{\mathbf{y}};\mathbf{w}_i|\hat{\mathbf{x}}) \geq & H(\hat{\mathbf{y}}|\hat{\mathbf{x}})+\\
& \mathbb{E}_{\hat{\mathbf{y}}\sim p(\hat{\mathbf{y}}|\hat{\mathbf{x}})}[\mathbb{E}_{\mathbf{w}_i\sim p(\mathbf{w}|\hat{\mathbf{y}},\hat{\mathbf{x}})}[\log p_\theta(\hat{\mathbf{y}}|\hat{\mathbf{x}},\mathbf{w}_i)]].
\end{aligned} \tag{9}$$

We can make the same derivation for the support data $(\mathbf{x}_{c_i},\mathbf{y}_{c_i})$ from class c_i. Put the lower bounds back into Equation 7. Omit the constant entropy terms and the expectation subscripts for clarity, we have the new objective function as

$$\begin{aligned}
\max_\theta \mathbb{E}[& \log p_\theta(\hat{\mathbf{y}}|\hat{\mathbf{x}},\mathbf{w}_i) + \log p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)+\\
& \frac{1}{K}\sum_K \log p_\theta(\mathbf{y}_{c_i}|\mathbf{x}_{c_i},\mathbf{w}_i) + \log p_\theta(\mathbf{x}_{c_i}|\mathbf{w}_i)].
\end{aligned} \tag{10}$$

The first and third 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. Furthermore, we assume that $p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)$ and $p_\theta(\mathbf{x}_{c_i}|\mathbf{w}_i)$ are Gaussian distributions. Therefore maximizing the log likelihood can be achieved by minimizing $L2$ reconstruction loss. Overall, applying MI maximization, we arrive at an objective function with cross entropy loss and reconstruction loss, which will be discussed in Section 3.5.

3.4. Attentive Weights Generation

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 $\mathcal{T}$ are processed by this feature extractor and represented as d-dimensional vectors afterwards, i.e., $\mathbf{x},\hat{\mathbf{x}}\in\mathbb{R}^d$. There are two paths to encode the task context and individual query sample respectively, which are called contextual path and attentive path. The outputs of both paths are concatenated together as input to the 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 MI.

3.4.1 Contextual and Attentive Paths

We use multi-head attention networks for encoding. The use of attention mechanism is to model the interactions/relations between samples within one task as the task-specific property. Previous work has applied relation networks for this purpose[36]. We use more advanced multi-head attention because of its advantage in modeling interactions from different representation subspaces[42]. The multi-head attention with H heads can be described as

$$MultiHead(Q,K,V) = Concat(head_1,...,head_H)W^O, \tag{11}$$

$$head_j(Q^j,K^j,V^j) = Attention(Q^j,K^j,V^j), \tag{12}$$

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}}V), \tag{13}$$

$$Q^j = QW_Q^j, K^j = KW_K^j, V^i = VW_V^j, \tag{14}$$

where Q,K,V are query, key, value matrices. W_Q^j, W_K^j, W_V^j are the weight matrices for jth head. W^O is the weight matrix for output. d_k is the dimension of keys. Original Q is added to the output of Equation 11 to stabilize the training as residual learning.

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 $f_{\theta_{cp}^{sa}}$ parameterized by θ_{cp}^{sa} [1] [42], described as

$$\mathbf{X}^{cp} = f_{\theta_{cp}^{sa}}(Q=\mathbf{X},K=\mathbf{X},V=\mathbf{X}). \tag{15}$$

The outputs of contextual path $\mathbf{X}^{cp}\in\mathbb{R}^{NK\times d_h}$, thus contain richer information about the task and can be used later for weights generation. $d_h < d$ is the hidden dimension.

Existing weights generation methods generate the classification weights conditioned on the support set only, which

[1] cp, sa stand for contextual path, self-attention respectively.

is equivalent to using contextual path. However, the classification weights generated in this way might be sub-optimal, lacking necessary adaptation to different query samples. We address this issue by introducing an 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 other words, our contextual/attentive path models global/local task structure respectively.

In the attentive path, a new multi-head self-attention network $f_{\theta_{ap}^{sa}}$ on the support set is employed to encode the global task information,

$$\mathbf{X}^{ap} = f_{\theta_{ap}^{sa}}(Q = \mathbf{X}, K = \mathbf{X}, V = \mathbf{X}). \tag{16}$$

$f_{\theta_{ap}^{sa}}$ is different from $f_{\theta_{cp}^{sa}}$ in contextual path because the self-attention network in contextual path emphasizes on generating the classification weights. On the contrary, $\mathbf{X}^{ap}$ plays the role of providing the $Value$ context to be attended by different query samples 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 $f_{\theta_{ap}^{ca}}$ applied on each query sample and task-aware support set is followed to produce $\hat{\mathbf{x}}^{ap} \in \mathbb{R}^{d_h}$,

$$\hat{\mathbf{x}}^{ap} = f_{\theta_{ap}^{ca}}(Q = \hat{\mathbf{x}}, K = \mathbf{X}, V = \mathbf{X}^{ap}). \tag{17}$$

3.4.2 Weights Generator

The outputs of contextual path $\mathbf{X}^{cp} \in \mathbb{R}^{NK \times d_h}$ are concatenated with $\hat{\mathbf{x}}^{ap} \in \mathbb{R}^{d_h}$. Then we have $\mathbf{X}^{cp \oplus ap} \in \mathbb{R}^{NK \times 2d_h}$. $\mathbf{X}^{cp \oplus ap}$ is specific for one query to generate classification weights.

$\mathbf{X}^{cp \oplus ap}$ is fed into the weights generator $g : \mathbb{R}^{2d_h} \rightarrow \mathbb{R}^{2d}$, parameterized by θ_g. 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 with reparameterization trick [18], shown in Equation 18 and 19.

$$\boldsymbol{\mu}_{\mathbf{w}_i}, \boldsymbol{\sigma}_{\mathbf{w}_i} = g(\mathbf{x}_{c_i}^{cp \oplus ap}) \tag{18}$$

$$\mathbf{w}_i | \hat{\mathbf{x}}, \mathbf{x}_{c_i} \sim \mathcal{N}(\boldsymbol{\mu}_{\mathbf{w}_i}, \boldsymbol{\Sigma}_{\mathbf{w}_i}) \tag{19}$$

$\boldsymbol{\Sigma}_{\mathbf{w}_i}$ is the covariance matrix with $\boldsymbol{\sigma}_{\mathbf{w}_i}$ as diagonal entries. The sampled classification weights are represented as $\mathbf{W} \in \mathbb{R}^{NK \times d}$. To reduce complexity, we compute the mean value on K classification weights for each class to have $\mathbf{W}^{final} \in \mathbb{R}^{N \times d}$. The prediction for query data can be computed by $\mathbf{W}^{final}\hat{\mathbf{x}}$. The prediction for support data can also be computed as $\mathbf{W}^{final}\mathbf{x}_{c_i}$.

Besides the weights generator g, we have another two decoders $r_1 : \mathbb{R}^d \rightarrow \mathbb{R}^{d_h}$ and $r_2 : \mathbb{R}^d \rightarrow \mathbb{R}^{d_h}$, parameterized by θ_{r_1} and θ_{r_2}. They both take the generated weights $\mathbf{W}$ as inputs and are used to learn $p_\theta(\hat{\mathbf{x}}|\mathbf{w}_i)$ and $p_\theta(\mathbf{x}_{c_i}|\mathbf{w}_i)$ in Equation 10. In other words, r_1 and r_2 learn to reconstruct $\mathbf{x}_{c_i}^{cp}$ and $\hat{\mathbf{x}}^{ap}$ respectively since $\mathbf{x}_{c_i}^{cp}$ and $\hat{\mathbf{x}}^{ap}$ are the direct inputs to g. The outputs of r_1 and r_2 are denoted as $\mathbf{x}_{c_i,re}^{cp}, \hat{\mathbf{x}}_{re}^{ap} \in \mathbb{R}^{d_h}$.

3.5. Training and Inference

The objective function 10 used during meta-training is equivalent to

$$\min_{\theta_{cp}^{sa}, \theta_{ap}^{sa}, \theta_{ap}^{ca}, \theta_g} \mathrm{CE}(\hat{\mathbf{y}}_{pred}, \hat{\mathbf{y}}) + \frac{\lambda_1}{K} \sum_K \mathrm{CE}(\mathbf{y}_{pred,c_i}, \mathbf{y}_{c_i})$$
$$+ \min_{\theta_g, \theta_{r_1}, \theta_{r_2}} \frac{\lambda_2}{K} \sum_K ||\mathbf{x}_{c_i}^{cp} - \mathbf{x}_{ci,re}^{cp}||_2 + \lambda_3 ||\hat{\mathbf{x}}^{ap} - \hat{\mathbf{x}}_{re}^{ap}||_2. \tag{20}$$

CE here stands for cross entropy. Since we convert the log likelihood in Equation 10 to mean square error or cross entropy in Equation 20 to optimize, the value of each term in Equation 20 is different from the corresponding log likelihood in Equation 10 by some constant multiplier. Thus, we have to decide the hyper-parameters $\lambda_1, \lambda_2, \lambda_3$ for trade-off of different terms. The reconstruction loss is used to update r_1, r_2 for reconstruction and g for weights generation. This is because we want the attention modules in two paths to focus on encoding expressive representations for following classification. 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. It should be noted that this loss function is computed for one query example in one task. During meta training, there are certain number of query samples in one task and multiple tasks in one batch, which are used to compute average of Equation 20. In meta-testing, $\boldsymbol{\mu}_{\mathbf{w}_i}$ is used as the classification weight for class c_i without sampling.

3.6. Complexity Analysis

The encoding process in contextual path results in computational complexity $O((NK)^2)$ due to self-attention. Similarly, the computational complexity of attentive path is $O((NK)^2 + |\mathcal{Q}|(NK))$. In total, the complexity is $O((NK)^2 + |\mathcal{Q}|(NK))$. However, because of the nature of few shot learning problem, the value of $(NK)^2$ is usually negligible. The value of $|\mathcal{Q}|$ is task-dependent and the cross attention can be implemented in parallel via matrix multiplication. Besides, AWGIM avoids the inner update without compromising the performance, further reducing inference time. Therefore, when $|\mathcal{Q}|$ is not extremely large, the induced computational overhead will be negligible.

Table 1. Accuracy comparison with other approaches on *mini*ImageNet. Top 3 results are highlighted. We remark that our AWGIM is trained with fixed image features in order to have fair comparison with LEO [36]. On the contrary, MetaOptNet [21] is trained with a feature extractor end-to-end. The same for the results on *tiered*ImageNet.

Model	Feature Extractor	5-way 1-shot	5-way 5-shot
Matching Networks [43]	Conv-4	46.60	60.00
MAML[11]	Conv-4	$48.70 \pm 1.84\%$	$63.11 \pm 0.92\%$
Meta LSTM [33]	Conv-4	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
Prototypical Nets [39]	Conv-4	$49.42 \pm 0.78\%$	$68.20 \pm 0.66\%$
Relation Nets [41]	Conv-4	$50.44 \pm 0.82\%$	$65.32 \pm 0.70\%$
SNAIL [28]	Resnets-12	$55.71 \pm 0.99\%$	$68.88 \pm 0.92\%$
TPN [26]	Resnets-12	59.46	75.65
MTL [40]	Resnets-12	$61.20 \pm 1.80\%$	75.50 ± 0.80
MetaOptNet [21]	Resnets-12	**$64.09 \pm 0.62\%$**	**$80.00 \pm 0.45\%$**
Dynamic [12]	WRN-28-10	$60.06 \pm 0.14\%$	$76.39 \pm 0.11\%$
Prediction [32]	WRN-28-10	$59.60 \pm 0.41\%$	$73.74 \pm 0.19\%$
DAE-GNN [13]	WRN-28-10	**$62.96 \pm 0.15\%$**	**$78.85 \pm 0.10\%$**
LEO [36]	WRN-28-10	$61.76 \pm 0.08\%$	$77.59 \pm 0.12\%$
AWGIM	WRN-28-10	**$63.12 \pm 0.08\%$**	**$78.40 \pm 0.11\%$**

Table 2. Accuracy comparison with other approaches on *tiered*ImageNet. Top 3 results are highlighted.

Model	Feature Extractor	5-way 1-shot	5-way 5-shot
MAML [11]	Conv-4	$51.67 \pm 1.81\%$	$70.30 \pm 1.75\%$
Prototypical Nets [39]	Conv-4	$53.31 \pm 0.89\%$	$72.69 \pm 0.74\%$
Relation Nets [41]	Conv-4	$54.48 \pm 0.93\%$	$71.32 \pm 0.78\%$
TPN [26]	Conv-4	$59.91 \pm 0.96\%$	$72.85 \pm 0.74\%$
MetaOptNet [21]	Resnets-12	$65.81 \pm 0.74\%$	$81.75 \pm 0.53\%$
Dynamic [12]	WRN-28-10	**$67.92 \pm 0.16\%$**	**$83.10 \pm 0.12\%$**
DAE-GNN [13]	WRN-28-10	**$68.18 \pm 0.16\%$**	**$83.09 \pm 0.12\%$**
LEO [36]	WRN-28-10	$66.33 \pm 0.05\%$	$81.44 \pm 0.09\%$
AWGIM	WRN-28-10	**$67.69 \pm 0.11\%$**	**$82.82 \pm 0.13\%$**

4. Experiments

4.1. Datasets and Protocols

We conduct experiments on ***mini*ImageNet** [43] and ***tiered*ImageNet** [34], two commonly used benchmark datasets, to compare with other methods and analyze our model. Both datasets are subsets of ILSVRC-12 dataset [35]. *mini*ImageNet contains 100 randomly sampled classes with 600 images per class. We follow the train/test split in [33], where 64 classes are used for meta-training, 16 for meta-validation and 20 for meta-testing. *tiered*ImageNet is a larger dataset with 608 classes and 779,165 images in total. They are selected from 34 higher level nodes in ImageNet [8] 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.

For fair comparison, we use the same image features in LEO [36] provided by the authors [2]. They trained a 28-layer Wide Residual Network [49] on the meta-training set. Each image then is represented by a 640 dimensional vector, which is used as the input to our network.

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. 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 95% confidence interval, as done in recent works [11, 39, 36].

4.2. Implementation Details

We use TensorFlow [1] to implement our method. $d = 640$ is the dimension of feature embeddings. d_h is set to be 128. The number of heads H in attention module is set to be 4. g, r_1 and r_2 are 2-layer MLPs with 256 hidden units. We use $\lambda_1 = 1$, $\lambda_2 = \lambda_3 = 0.001$ by meta-validation performance.

ADAMW [27] is used to optimize the network with weight decay 1×10^{-6}. The initial learning rate is set to 0.0002 for 5-way 1-shot and 0.001 for 5-way 5-shot, which is decayed by 0.2 for every 15,000 iterations. We train the model for 50,000 iterations. Batch size is 64 for 5-way 1-shot and 32 for 5-way 5-shot. Similar to LEO [36], we first train the model on meta-training set and choose the optimal hyper-parameters by validation results. Then we train the model on meta-training and meta-validation sets together

[2]https://github.com/deepmind/leo

Table 3. Analysis of our proposed AWGIM. 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. We also shuffle the generated classification weights randomly to show that they are indeed optimal for different query samples.

Model	*mini*ImageNet		*tiered*ImageNet	
	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 $\mathcal{S}$ only	61.02%	74.33%	66.22%	79.66%
Generator conditioned on $\mathcal{S}$ with IM	62.04%	77.54%	66.43%	81.73%
MLP encoding (*i.e.* no attention)	62.26%	76.91%	65.84%	79.24%
MLP encoding, $\lambda_1 = \lambda_2 = \lambda_3 = 0$	58.95%	71.68%	63.92%	75.80%
$\lambda_1 = \lambda_2 = \lambda_3 = 0$	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%
AWGIM (ours)	**63.12%**	**78.40%**	**67.69%**	**82.82%**

using fixed hyper-parameters.

4.3. Comparison with Other Methods

We compare the performance of our approach AWGIM on two datasets with several state-of-the-art methods proposed in recent years. The results of MAML, Prototypical Nets, Relation Nets on *tiered*ImageNet are evaluated by [26]. The results of Dynamic on *mini*ImageNet with WRN-28-10 as the feature extractor are reported in [13]. The other results are reported in the corresponding original papers. We also include the backbone network of feature extractor for reference. The results on *mini*ImageNet and *tiered*ImageNet 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 [43], Prototypical Nets [39]), gradient-based (MAML [11], MTL [40]), graph-based (TPN [26]). The bottom part shows the classification weights generation approaches including Dynamic [12], Prediction [32], DAE-GNN [13], LEO [36].

AWGIM achieves top 3 highest accuracy on both datasets and compares favorably to the best results. In particular, AWGIM can outperform LEO in all settings. It should be noted that AWGIM is trained with fixed image features extracted from WRN-28-10 in order to have fair comparison with LEO. On the contrary, MetaOptNet [21] is trained with a feature extractor end-to-end. From Table 4 in the MetaOptNet paper [21], we can see that large parts of performance gains stem from the strong feature extractor. Several training techniques including data augmentation, weight decay, drop block and so on boost the performance significantly.

4.4. Analysis

We perform detailed analysis on AWGIM, shown in Table 3. We include the results of LEO [36] for reference.

The effect of attentive path is shown in the upper part of Table 3. "Generator in LEO" means that there is no inner update in LEO. We implemented two generators *including only the contextual path* during encoding. "Generator conditioned on $\mathcal{S}$ 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 on support set is no worse than relation networks used in LEO to model task-context. "Generator conditioned on $\mathcal{S}$ with IM" indicates that we add the cross entropy loss and reconstruction loss for support set. 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.

The contribution of multi-head attention is further clarified in Table 4. We replace the multi-head attention in the two paths with single-head attention and conduct the 5-way 1-shot and 5-way 5-shot experiments on *mini*ImageNet dataset. We can see clearly that multi-head attention improves the performance. In particular, for 1-shot experiment, single head attention gives results close to MLP encoding, which indicates that single head attention struggles when labeled support data are extremely scarce.

Ablation analysis with respective to λ_1, λ_2 and λ_3 is conducted to study the effect of information maximization.

Table 4. Accuracy results on *mini*ImageNet with 4 heads or single head in attention networks.

Method	5-way 1-shot	5-way 5-shot
4 heads	63.12%	78.40%
single head	62.35%	77.75%

First, λ_1, λ_2 and λ_3 are all set to be 0. In this case, the accuracy is similar to "generator conditioned on $\mathcal{S}$ only", showing that the generated classification weights are not fitted for current tasks, even with the attentive path. It can also be observed that maximizing the MI 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, which implies that the support label prediction is more critical for information maximization.

Whether the classification weights are adapted for different query samples is investigated by shuffling the classification weights. In particular, we shuffle the classification weights between query samples within the same classes and between different classes as well. Assume there are T query samples per class in one task. $\mathbf{W}^{final} \in \mathbb{R}^{|\mathcal{Q}|\times N\times d}$ can be reshaped into $\mathbf{W}^{final} \in \mathbb{R}^{N\times T\times N\times d}$. 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.

4.5. Convergence

We compare AWGIM with LEO in terms of convergence speed. The batch size is set to be 16 for both methods. We use the hyper-parameters tuned by authors to train LEO. The accuracy of meta-validation set during meta-training on 5-way 1-shot *mini*ImageNet is plotted, shown in Figure 2. We can see clearly that AWGIM converges faster than LEO and outperforms LEO except for the first few iterations.

4.6. Inference Time Cost

We measure the inference time of AWGIM to show that it induces minimal computational overhead. One MLP in

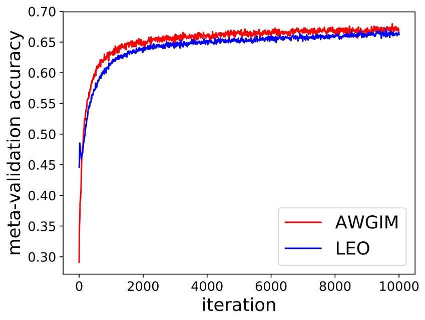

Figure 2. The meta-validation accuracy during meta-training.

both paths and LEO are compared to AWGIM, which have time complexity $O(NK + |\mathcal{Q}|)$ and $O((NK)^2)$. Since the time complexity of AWGIM and MLP depends on $|\mathcal{Q}|$, we test different number of query samples. We use two set-ups on *mini*ImageNet and the batch size is set to be 64. 100 batches are processed and we report the average consumed time for one batch. All these experiments are conducted on the same computing device. The results are shown in Table 5. It can be observed that when $|\mathcal{Q}|$ is small, AWGIM is faster than LEO due to avoiding inner update. When $|\mathcal{Q}|$ is large, both MLP and AWGIM performs slower. Noticeably, the usage of self-attention and cross attention in AWGIM incurs negligible overhead, compared with MLP encoding.

Table 5. Inference time cost of AWGIM and MLP encoding.

Method	5-way 1-shot		5-way 5-shot	
	$\lvert\mathcal{Q}\rvert = 5$	$\lvert\mathcal{Q}\rvert = 50$	$\lvert\mathcal{Q}\rvert = 5$	$\lvert\mathcal{Q}\rvert = 50$
MLP	0.015s	0.031s	0.021s	0.076s
LEO	0.029s	0.032s	0.033s	0.039s
AWGIM	0.019s	0.036s	0.025s	0.079s

5. Conclusion

In this work, we introduce Attentive Weights Generation via Information Maximization for few shot image classification. AWGIM learns to generate optimal classification weights for each query sample within the task by two encoding paths. To this end, the lower bound of mutual information between generated weights and query, support data is maximized. The effectiveness of AWGIM is demonstrated by competitive performance on two benchmark datasets and extensive analysis.

6. Acknowledgments

This work was supported by ST Electronics and the National Research Foundation(NRF), Prime Minister's Office, Singapore under Corporate Laboratory @ University Scheme (Programme Title: STEE Infosec - SUTD Corporate Laboratory), National Research Foundation Singapore under its AI Singapore Programme [Award Number: AISG-100E2018-005] and partially supported by the Energy Market Authority (EP award no. NRF2017EWT-EP003-061).

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Marcin Andrychowicz, Misha Denil, Sergio Gomez, Matthew W Hoffman, David Pfau, Tom Schaul, Brendan Shillingford, and Nando De Freitas. Learning to learn by gradient descent by gradient descent. In *NeurIPS*, 2016.

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In *ICLR*, 2015.

[4] David Barber and Felix V Agakov. The im algorithm: a variational approach to information maximization. In *NeurIPS*, 2003.

[5] Luca Bertinetto, Joao F Henriques, Philip Torr, and Andrea Vedaldi. Meta-learning with differentiable closed-form solvers. In *ICLR*, 2019.

[6] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *NeurIPS*, 2016.

[7] Zitian Chen, Yanwei Fu, Yu-Xiong , Lin Ma, Wei Liu, and Martial Hebert. Image deformation meta-networks for one-shot learning. In *CVPR*, 2019.

[8] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, 2009.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[10] Li Fei-Fei, Rob Fergus, and Pietro Perona. One-shot learning of object categories. *TPAMI*, 2006.

[11] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *ICML*, 2017.

[12] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. In *CVPR*, 2018.

[13] Spyros Gidaris and Nikos Komodakis. Generating classification weights with gnn denoising autoencoders for few-shot learning. In *CVPR*, 2019.

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, 2016.

[15] R Devon Hjelm, Alex Fedorov, Samuel Lavoie-Marchildon, Karan Grewal, Phil Bachman, Adam Trischler, and Yoshua Bengio. Learning deep representations by mutual information estimation and maximization. *ICLR*, 2019.

[16] Ruibing Hou, Hong Chang, MA Bingpeng, Shiguang Shan, and Xilin Chen. Cross attention network for few-shot classification. In *NeurIPS*, 2019.

[17] Hyunjik Kim, Andriy Mnih, Jonathan Schwarz, Marta Garnelo, Ali Eslami, Dan Rosenbaum, Oriol Vinyals, and Yee Whye Teh. Attentive neural processes. In *ICLR*, 2019.

[18] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *ICLR*, 2014.

[19] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, 2015.

[20] Ranjay Krishna, Michael Bernstein, and Li Fei-Fei. Information maximizing visual question generation. In *CVPR*, 2019.

[21] Kwonjoon Lee, Subhransu Maji, Avinash Ravichandran, and Stefano Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, 2019.

[22] Yoonho Lee, Wonjae Kim, and Seungjin Choi. Discrete infomax codes for meta-learning. *arXiv preprint arXiv:1905.11656*, 2019.

[23] Hongyang Li, David Eigen, Samuel Dodge, Matthew Zeiler, and Xiaogang Wang. Finding task-relevant features for few-shot learning by category traversal. In *CVPR*, 2019.

[24] Wenbin Li, Lei Wang, Jinglin Xu, Jing Huo, Yang Gao, and Jiebo Luo. Revisiting local descriptor based image-to-class measure for few-shot learning. In *CVPR*, 2019.

[25] Yann Lifchitz, Yannis Avrithis, Sylvaine Picard, and Andrei Bursuc. Dense classification and implanting for few-shot learning. In *CVPR*, 2019.

[26] Yanbin Liu, Juho Lee, Minseop Park, Saehoon Kim, Eunho Yang, Sung Ju Hwang, and Yi Yang. Learning to propagate labels: Transductive propagation network for few-shot learning. In *ICLR*, 2019.

[27] Ilya Loshchilov and Frank Hutter. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 2017.

[28] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2018.

[29] Tsendsuren Munkhdalai and Hong Yu. Meta networks. In *ICML*, 2017.

[30] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *ICML*, 2018.

[31] Hang Qi, Matthew Brown, and David G Lowe. Low-shot learning with imprinted weights. In *CVPR*, 2018.

[32] Siyuan Qiao, Chenxi Liu, Wei Shen, and Alan L Yuille. Few-shot image recognition by predicting parameters from activations. In *CVPR*, 2018.

[33] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2016.

[34] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. In *ICLR*, 2018.

[35] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *IJCV*, 2015.

[36] Andrei A Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. In *ICLR*, 2019.

[37] Jürgen Schmidhuber. Evolutionary principles in self-referential learning, or on learning how to learn: the meta-meta-... hook. *PhD thesis*, 1987.

[38] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[39] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, 2017.

[40] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. In *CVPR*, 2019.

[41] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip HS Torr, and Timothy M Hospedales. Learning to compare: Relation network for few-shot learning. In *CVPR*, 2018.

[42] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.

[43] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, 2016.

[44] Yu-Xiong Wang, Ross Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. In *CVPR*, 2018.

[45] Davis Wertheimer and Bharath Hariharan. Few-shot learning with localization in realistic settings. In *CVPR*, 2019.

[46] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *ICML*, 2015.

[47] Shipeng Yan, Songyang Zhang, Xuming He, et al. A dual attention network with semantic embedding for few-shot learning. In *AAAI*, 2019.

[48] Mingzhang Yin, George Tucker, Mingyuan Zhou, Sergey Levine, and Chelsea Finn. Meta-learning without memorization. *ICLR*, 2020.

[49] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[50] Ruixiang Zhang, Tong Che, Zoubin Ghahramani, Yoshua Bengio, and Yangqiu Song. Metagan: An adversarial approach to few-shot learning. In *NeurIPS*, 2018.