ProtoDiff: Learning to Learn Prototypical Networks by Task-Guided Diffusion

Yingjun Du[1], Zehao Xiao[1], Shencai Liao[2] , Cees G. M. Snoek[1]
[1]AIM Lab, University of Amsterdam [2]Inception Institute of Artificial Intelligence

Abstract

Prototype-based meta-learning has emerged as a powerful technique for addressing few-shot learning challenges. However, estimating a deterministic prototype using a simple average function from a limited number of examples remains a fragile process. To overcome this limitation, we introduce ProtoDiff, a novel framework that leverages a task-guided diffusion model during the meta-training phase to gradually generate prototypes, thereby providing efficient class representations. Specifically, a set of prototypes is optimized to achieve per-task prototype overfitting, enabling accurately obtaining the overfitted prototypes for individual tasks. Furthermore, we introduce a task-guided diffusion process within the prototype space, enabling the meta-learning of a generative process that transitions from a vanilla prototype to an overfitted prototype. ProtoDiff gradually generates task-specific prototypes from random noise during the meta-test stage, conditioned on the limited samples available for the new task. Furthermore, to expedite training and enhance ProtoDiff's performance, we propose the utilization of residual prototype learning, which leverages the sparsity of the residual prototype. We conduct thorough ablation studies to demonstrate its ability to accurately capture the underlying prototype distribution and enhance generalization. The new state-of-the-art performance on within-domain, cross-domain, and few-task few-shot classification further substantiates the benefit of ProtoDiff.

1 Introduction

This paper considers prototype-based meta-learning, where models are trained to swiftly adapt to current tasks and perform classification through metric-based comparisons between examples and newly introduced variables - the prototypes of the classes. This approach, rooted in works by Reed [34] and further developed by Snell *et al.*[42], generalizes deep learning models to scenarios where labeled data is scarce. The fundamental idea is to compute the distances between queried examples and class prototypes and perform classification based on these distances to predict classes for queried examples. Derived from the prototypical network [42], several prototype-based methods have demonstrated their effectiveness in few-shot learning [2, 8, 11, 61]. However, as prototypes are estimated from a limited number of sampled examples, they may not accurately capture the overall distribution [54, 61]. Such biased distributions could lead to biased prototypes and subsequent classification errors, suggesting that the current prototype modeling approach might lack the ability to represent universal class-level information effectively. Rather than relying on points in the prototype embedding space by using a simple average function, we propose to generate the distribution of prototypes for each task.

To overcome the challenges of estimating deterministic prototypes from limited examples in few-shot learning, we propose ProtoDiff, an innovative framework that leverages a task-guided diffusion model during the meta-training phase. ProtoDiff introduces a three-step process to address these limitations comprehensively. Firstly, we optimize a set of prototypes to achieve per-task prototype overfitting,

37th Conference on Neural Information Processing Systems (NeurIPS 2023).

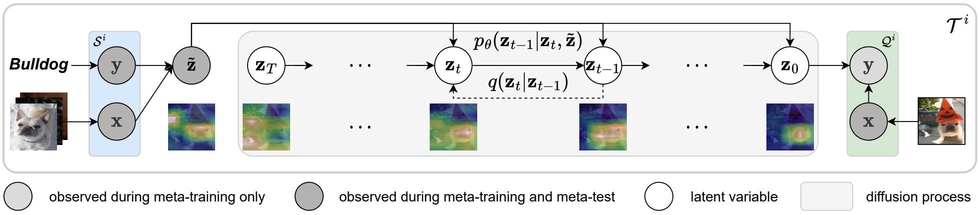

Bulldog
$\mathcal{S}^i$
y
x
$\tilde{\mathbf{z}}$
$\mathbf{z}_T$
$\mathbf{z}_t$
$p_\theta(\mathbf{z}_{t-1}|\mathbf{z}_t, \tilde{\mathbf{z}})$
$q(\mathbf{z}_t|\mathbf{z}_{t-1})$
$\mathbf{z}_{t-1}$
$\mathbf{z}_0$
$\mathcal{Q}^i$
$\mathcal{T}^i$
observed during meta-training only
observed during meta-training and meta-test
latent variable
diffusion process

Figure 1: Computational graph of ProtoDiff. In the initial stage, the computation of the vanilla prototype $\tilde{\mathbf{z}}$ is performed based on the support set $\mathcal{S}^i$. Subsequently, through the process of diffusion sampling, the diffused prototype $\mathbf{z}_{t-1}$ is derived from the combination of $\mathbf{z}_t$ and $\tilde{\mathbf{z}}$. Finally, the prediction $\mathbf{y}$ for the query set $\mathcal{Q}^i$ is generated by utilizing the diffused prototype $\mathbf{z}_0$ in conjunction with the query set $\mathbf{x}$. The diffusion forward and sampling processes are indicated by dashed and solid arrows within the gray rectangular area.

ensuring accurate retrieval of overfitted prototypes specific to individual tasks. These overfitted prototypes serve as the *ground truth* representations for each task. Secondly, a task-guided diffusion process is implemented within the prototype space, enabling meta-learning of a generative process that smoothly transitions prototypes from their vanilla form to the overfitted state. ProtoDiff generates task-specific prototypes during the meta-test stage by conditioning random noise on the limited samples available for the given new task. Finally, we propose residual prototype learning, which significantly accelerates training and further enhances the performance of ProtoDiff by leveraging the sparsity of residual prototypes. The computational graph of ProtoDiff, illustrated in Figure 1, showcases the sequential steps involved in the forward diffusion process on the prototype and the generative diffusion process. The resulting generated prototype, denoted as $\mathbf{z}_0$, is utilized for query prediction. By incorporating probabilistic and task-guided prototypes, ProtoDiff balances adaptability and informativeness, positioning it as a promising approach for augmenting few-shot learning in prototype-based meta-learning models.

To validate the effectiveness of ProtoDiff, we conduct comprehensive experiments on three distinct few-shot learning scenarios: within-domain, cross-domain, and few-task few-shot learning. Our findings reveal that ProtoDiff significantly outperforms state-of-the-art prototype-based meta-learning models, underscoring the potential of task-guided diffusion to boost few-shot learning performance. Furthermore, we provide a detailed analysis of the diffusion mechanism employed in ProtoDiff, showcasing its ability to capture the underlying data structure better and improve generalization. This thorough investigation highlights the strengths of our approach, demonstrating its potential to offer a more effective solution for few-shot learning tasks in various applications and settings.

2 Preliminaries

Before detailing our ProtoDiff methodology, we first present the relevant background on few-shot classification, the prototypical network, and diffusion models.

Few-shot classification. We define the N-way K-shot classification problem, which consists of support sets $\mathcal{S}$ and a query set $\mathcal{Q}$. Each task $\mathcal{T}^i$, also known as an episode, represents a classification problem sampled from a task distribution $p(\mathcal{T})$. The *way* of an episode denotes the number of classes within the support sets, while the *shot* refers to the number of examples per class. Tasks are created from a dataset by randomly selecting a subset of classes, sampling points from these classes, and subsequently dividing the points into support and query sets. The episodic optimization approach [50] trains the model iteratively, performing one episode update at a time.

Prototypical network. We develop our method based on the prototypical network (ProtoNet) by Snell *et al.* [42]. Specifically, the ProtoNet leverages a non-parametric classifier that assigns a query point to the class having the nearest prototype in the learned embedding space. The prototype $\mathbf{z}^c$ of an object class c is obtained by: $\mathbf{z}^c{=}\frac{1}{K}\sum_k f_\phi(\mathbf{x}^{c,k})$, where $f_\phi(\mathbf{x}^{c,k})$ is the feature embedding of the support sample $\mathbf{x}^{c,k}$, which is usually obtained by a convolutional neural network. For each query sample $\mathbf{x}^q$, the distribution over classes is calculated based on the softmax over distances to the prototypes of all classes in the embedding space:

$$p(\mathbf{y}_n^q = c|\mathbf{x}^q) = \frac{\exp(-d(f_\phi(\mathbf{x}^q), \mathbf{z}^c))}{\sum_{c'} \exp(-d(f_\phi(\mathbf{x}^q), \mathbf{z}^{c'}))}, \tag{1}$$

where $\mathbf{y}^q$ denotes a random one-hot vector, with $\mathbf{y}^q_n$ indicating its n-th element, and $d(\cdot, \cdot)$ is some (Euclidean) distance function. Due to its non-parametric nature, the ProtoNet enjoys high flexibility and efficiency, achieving considerable success in few-shot learning. To avoid confusion, we omit the superscript c for the prototype $\mathbf{z}$ in this subsection.

Diffusion model. In denoising diffusion probabilistic models [16], a forward diffusion process, $q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1})$, is characterized as a Markov chain that progressively introduces Gaussian noise at each time step t, beginning with a clean image $\boldsymbol{x}_0 \sim q(\boldsymbol{x}_0)$. The subsequent forward diffusion process is formulated as follows:

$$q(\boldsymbol{x}_T|\boldsymbol{x}_0) := \prod_{t=1}^{T} q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}), \quad \text{where} \quad q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) := \mathcal{N}(\boldsymbol{x}_t; \sqrt{1-\beta_t}\boldsymbol{x}_{t-1}, \beta_t \boldsymbol{I}), \tag{2}$$

where $\{\beta\}_{t=0}^{T}$ is a variance schedule. By defining $\alpha_t{:=}1{-}\beta_t$ and $\bar{\alpha}_t{:=}\prod_{s=1}^{t} \alpha_s$, the forward diffused sample at time step t, denoted as $\boldsymbol{x}_t$, can be generated in a single step as follows:

$$\boldsymbol{x}_t = \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \quad \text{where} \quad \boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}). \tag{3}$$

Since the reverse of the forward step, $q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, is computationally infeasible, the model learns to maximize the variational lower bound using parameterized Gaussian transitions, $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, where the parameter is denoted as θ. Consequently, the reverse process is approximated as a Markov chain with the learned mean and fixed variance, starting from a random noise $\boldsymbol{x}_T \sim \mathcal{N}(\boldsymbol{x}_T; \mathbf{0}, \boldsymbol{I})$:

$$p_\theta(\boldsymbol{x}_{0:T}) := p_\theta(\boldsymbol{x}_T) \prod_{t=1}^{T} p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t), \tag{4}$$

where

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) := \mathcal{N}(\boldsymbol{x}_{t-1}; \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t), \sigma_t^2 \boldsymbol{I}), \quad \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t) := \frac{1}{\sqrt{\alpha_t}}\Big(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\Big). \tag{5}$$

Here, $\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)$ is the diffusion model trained by optimizing the following objective function:

$$\mathcal{L}_\theta = \mathbb{E}_{t,\boldsymbol{x}_0,\boldsymbol{\epsilon}}\Big[\|\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_\theta(\sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)\|^2\Big]. \tag{6}$$

Upon completing the optimization, the learned score function is integrated into the generative (or reverse) diffusion process. To sample from $p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t)$, one can perform the following:

$$\boldsymbol{x}_{t-1} = \boldsymbol{\mu}_\theta(\boldsymbol{x}_t, t) + \sigma_t\boldsymbol{\epsilon} = \frac{1}{\sqrt{\alpha_t}}\Big(\boldsymbol{x}_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}}\boldsymbol{\epsilon}_\theta(\boldsymbol{x}_t, t)\Big) + \sigma_t\boldsymbol{\epsilon}. \tag{7}$$

In the case of conditional diffusion models [22, 38, 39], the diffusion model $\boldsymbol{\epsilon}_\theta$ in equations (6) and (7) is substituted with $\boldsymbol{\epsilon}_\theta(\mathbf{c}, \sqrt{\bar{\alpha}_t}\boldsymbol{x}_0 + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, t)$, where $\mathbf{c}$ represents the corresponding conditions, e.g., other images, languages, and sounds, etc. Consequently, the matched conditions strictly regulate the sample generation in a supervised manner, ensuring minimal changes to the image content. Dhariwal and Nichol [7] suggested classifier-guided image translation to generate the images of the specific classes along with a pre-trained classifier. Taking inspiration from conditional diffusion models that excel at generating specific images using additional information, we introduce the task-guided diffusion model that generates prototypes of specific classes by taking into account contextual information from various few-shot tasks. Note that our task-guided diffusion model operates not on an image $\boldsymbol{x}$, but on the prototype $\mathbf{z}$.

3 Methodology

This section outlines our approach to training prototypical networks via task-guided diffusion. We begin by explaining how to obtain task-specific overfitted prototypes in Section 3.1. Next, we introduce the task-guided diffusion method for obtaining diffused prototypes in Section 3.2. In the same section, we also introduce residual prototype learning, which accelerates training. Figure 2 visualizes the diffusion process of diffused prototypes by our ProtoDiff.

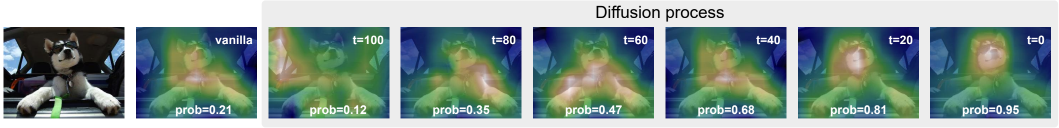


Figure 2: Visualization of the diffusion process. ProtoDiff randomly selects certain areas to predict during the diffusion process, with the lowest probability at the beginning time step. As time progresses, the prototype gradually aggregates towards the *dog*, with the highest probability at t=0.

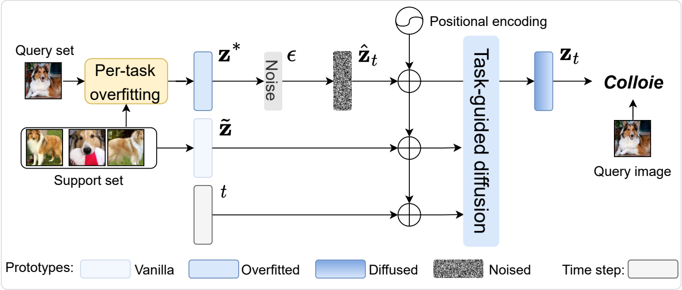


Figure 3: ProtoDiff illustration. We first obtain an overfitted prototype by the per-task prototype overfitting. We then add noise to the overfitted prototype, which inputs the diffusion forward process. The input of task-guided diffusion includes the vanilla prototype $\tilde{\mathbf{z}}$ and a random time step t. The resulting output is the diffused prototype $\mathbf{z}_t$.

3.1 Per-task prototype overfitting

Our approach utilizes a modified diffusion process as its foundation. Firstly, a meta-learner, denoted by $f_\phi(\mathcal{T}^i){=}f(\mathcal{T}^i, \phi)$, is trained on the complete set of tasks, $\mathcal{T}$. Here, ϕ corresponds to the learned weights of the model over the entire task set. Subsequently, fine-tuning is performed on each individual task $\mathcal{T}^i$ to obtain task-specific overfitted prototypes denoted by $\mathbf{z}^{i,*}{=}f\phi^i(\mathcal{S}^i, \mathcal{Q}^i)$. This involves running the meta-learner f_ϕ on the support set $\mathcal{S}^i$ and a query set $\mathcal{Q}^i$ of the task $\mathcal{T}^i$.

$$\phi^i = \phi - \eta \sum_{(\mathbf{x},\mathbf{y})\sim\mathcal{T}_i}^{\mathcal{T}_i} \mathcal{L}_{\mathrm{CE}}(f(\mathcal{S}^i, \mathbf{x}^{q^i}, \phi), \mathbf{y}^{q^i}), \tag{8}$$

where $\mathcal{L}_{\mathrm{CE}}$ is the cross-entropy loss minimized in the meta-learner training. The support set $\mathcal{S}^i$ and query set $\mathcal{Q}^i{=}\{\mathbf{x}^{q^i}, \mathbf{y}^{q^i}\}$ correspond to the data used for fine-tuning task $\mathcal{T}^i$. During fine-tuning, we obtain the ϕ^i through several iterations of gradient descent. We illustrate the overall framework of the per-task prototype learning in the appendix.

Our proposed method, ProtoDiff, for few-shot learning, relies on acquiring task-specific overfitted prototypes that can be considered as the "optima" prototypes due to their high confidence in final predictions, approaching a value of 1. To achieve this, we employ fine-tuning of the meta-learner and extract task-specific information that surpasses the accuracy and reliability of generic prototypes used in the meta-training stage. However, accessing the query set in the meta-test stage is not feasible. Hence, we need to rely on the vanilla prototype to meta-learn the process of generating the overfitted prototype during the meta-training stage. In the forthcoming section, we will introduce the task-guided diffusion model, which facilitates the learning process transitioning from the vanilla prototype to the overfitted prototype.

3.2 Task-guided diffusion

Our generative overfitted prototype is based on diffusion [43], a robust framework for modeling the prototypes instead of images. The length of the diffusion process T determines the number of forward passes needed to generate a new prototype, rather than the dimensionality of the data. Our model uses diffusion to progressively denoise the overfitted prototype $\mathbf{z}^*$.

Meta-training phase. Diffusion models can be configured to predict either the signal or the noise when presented with a noisy input [16, 31]. Previous research in the image domain has suggested that noise prediction is superior to signal prediction. However, we discovered empirically that signal prediction performs better than noise prediction in our experiments, so we parameterized the diffusion model to output the prototype.

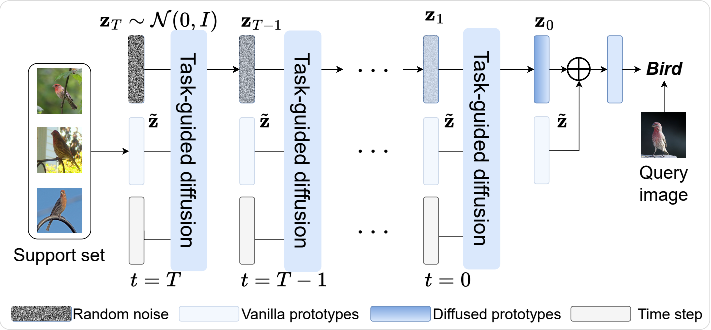


Figure 4: Diffusion sampling during meta-test. Sampling starts from a random noise $\mathbf{z_T}$ and gradually denoises it to the diffused prototypes $\mathbf{z_t}$. At each time step t is sampled by taking the $\mathbf{z_{t+1}}$, vanilla prototypes $\tilde{\mathbf{z}}$, and time step t as inputs. The diffused prototypes $\mathbf{z_0}$ are used to predict the query set.

The generative diffusion process is devised to reconstruct the overfitted prototype $\mathbf{z}^*$ by iteratively operating on a random noise vector $\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, which possesses the same dimensions as $\mathbf{z}^*$. This process continues with $\mathbf{z}_t$, where t denotes the number of diffusion steps. Consequently, the reconstructed $\mathbf{z}_0$ should exhibit proximity to $\mathbf{z}^*$ for a given task.

Specifically, during the forward diffusion process at time step t, we obtain the noised overfitted prototype $\hat{\mathbf{z}}_t$. Subsequently, we input the noised prototype $\hat{\mathbf{z}}_t$, the vanilla prototype $\tilde{\mathbf{z}}$, and the time step t into the task-guided diffusion. This yields the diffused prototype $\mathbf{z}_t$, which then allows us to predict the final results of the query set using Equation (1). For each task, our task-guided diffusion entails two components: the variational lower bound $\mathcal{L}_{\text{diff}}$ for the diffused prototype, and the cross-entropy loss $\mathcal{L}_{\text{CE}}$.

The objective is to minimize the simplified variational lower bound, which involves predicting the denoised overfitted prototype:

$$\mathcal{L}_{\text{diff}} = |\mathbf{z}^* - \mathbf{z}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{z}^* + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \tilde{\mathbf{z}}, t)|^2, \tag{9}$$

Here, $\mathbf{z}_\theta(\cdot, \cdot)$ is implemented by the transformer model [49] operating on prototype tokens from $\mathbf{z}^*$, $\tilde{\mathbf{z}}$, and the time step.

By utilizing equation (1), we derive the final prediction $\hat{\mathbf{y}}^q$ using the diffused prototype $\mathbf{z}_t$. The ultimate objective is expressed as follows:

$$\mathcal{L} = \sum_{(\mathbf{x},\mathbf{y})\sim\mathcal{Q}}^{|Q|} \Big[-\mathbb{E}q(\mathbf{z}_t|\mathbf{z}_{t+1}, \tilde{\mathbf{z}}) \big[\log p(\mathbf{y}^q|\mathbf{x}^q, \mathbf{z}_t) \big] + \beta|\mathbf{z}^* - \mathbf{z}_\theta(\sqrt{\bar{\alpha}_t}\mathbf{z}^* + \sqrt{1-\bar{\alpha}_t}\boldsymbol{\epsilon}, \tilde{\mathbf{z}}, t)|^2 \Big], \tag{10}$$

where β represents a hyperparameter, and $|Q|$ denotes the query size.

To prepare the two input prototypes $\mathbf{z}^*$ and $\tilde{\mathbf{z}}$ for processing by the transformer, we assign each position token inspired by [10]. We also provide the scalar input diffusion timestep t as individual tokens to the transformer. To represent each scalar as a vector, we use a frequency-based encoding scheme [28]. Our transformer architecture is based on GPT-2 [33]. The decoder in the final layer of our ProtoDiff maps the transformer's output to the diffused prototype. Note that only the output tokens for the noised overfitted prototypes $\hat{\mathbf{z}}_t$ are decoded to predictions. The overall meta-training phase of ProtoDiff is shown in Figure 3.

Meta-test phase. We can not obtain the overfitted prototype during the meta-test phase since we can not access the query set. Thus, diffusion sampling begins by feeding-in Gaussian noise $\mathbf{z}_T$ as the $\hat{\mathbf{z}}$ input and gradually denoising it. Specifically, to handle a new task τ=$\{\mathcal{S}, \mathcal{Q}\}$, we first compute the vanilla prototype $\tilde{\mathbf{z}}$ using the support set $\mathcal{S}$. We then randomly sample noise ϵ from $\mathcal{N}(\mathbf{0}, \boldsymbol{I})$. We input both $\tilde{\mathbf{z}}$and ϵ to the learned task-guided diffusion model to obtain the diffused prototype $\mathbf{z}_{T-1}$=$\mathbf{z}_\theta(\mathbf{z}_T, \tilde{\mathbf{z}}, T)$. After T iterations, the final diffused prototype can be obtained as $\mathbf{z}_0 = \mathbf{z}_\theta(\mathbf{z}_1, \mathbf{z}, t_0)$. Once the final diffused prototype $\mathbf{z}_0$ is obtained, we calculate the final prediction $\hat{\mathbf{y}}^q$ using equation (1). Figure 4 illustrates sampling in the meta-test stage. We also provide the detailed algorithm of the meta-training and meta-test phase in the appendix.

Residual prototype learning. To further enhance the performance of ProtoDiff, we introduce a residual prototype learning mechanism in our framework. We observe that the differences between the overfitted prototype $\mathbf{z}^*$ and its vanilla prototype $\tilde{\mathbf{z}}$ are not significant, as the vector $\mathbf{z}^* - \tilde{\mathbf{z}}$ contains many zeros. Therefore, we propose to predict the prototype update $\mathbf{z}^* - \tilde{\mathbf{z}}$ instead of directly predicting the overfitted prototype $\mathbf{z}^*$ itself. This approach also enables us to initialize ProtoDiff to perform the identity function by setting the decoder weights to zero. Moreover, we find that the global residual connection, combined with the identity initialization, significantly speeds up training. By utilizing this mechanism, we improve the performance of ProtoDiff in few-shot learning tasks.

4 Related Work

Prototype-based meta-learning. Prototype-based meta-learning is based on distance metrics and generally learns a shared or adaptive embedding space in which query images are accurately matched to support images for classification. It relies on the assumption that a common metric space is shared across related tasks and usually does not employ an explicit base learner for each task. By extending the matching network [50] to few-shot scenarios, Snell *et al.* [42] constructed a prototype for each class by averaging the feature representations of samples from the class in the metric space. The classification matches the query samples to prototypes by computing their distances. To enhance the prototype representation, Allen et al [2] proposed an infinite mixture of prototypes to adaptively represent data distributions for each class, using multiple clusters instead of a single vector. Oreshkin et al [32] proposed a task-dependent adaptive metric for few-shot learning and established prototype classes conditioned on a task representation encoded by a task embedding network. Yoon et al [60] proposed a few-shot learning algorithm aided by a linear transformer that performs task-specific null-space projection of the network output. Graphical neural network-based models generalize the matching methods by learning the message propagation from the support set and transferring it to the query set [13]. FEAT [58] was proposed to leverages samples from all categories within a task to generate a prototype, capitalizing on the intrinsic inter-class information to derive a more discriminative prototype. In contrast, our ProtoDiff, while using the overfitted prototype as supervision, only employs samples from a single category to generate the new prototype. This means we are not tapping into the potential informative context provided by samples from other categories. Additionally, our ProtoDiff employs the diffusion model to progressively generate the prototype, whereas FEAT does not utilize any generative model for prototype creation. Prototype-based methods have recently been improved in various ways [4, 46, 58, 61]. To the best of our knowledge, we are the first to propose a diffusion method to generate the prototype per task, rather than a deterministic function.

Diffusion models. These models belong to a category of neural generative models that utilize stochastic diffusion processes [43, 44], much like those found in thermodynamics. In this framework, a gradual introduction of noise is applied to a sample of data. A neural model then learns to reverse the process by progressively removing noise from the sample. The model denoises an initial pure noise to obtain samples from the learned data distribution. Ho *et al.* [16] and Song *et al.* [44] contributed to advancements in image generation, while Dhariwal and Nichol [7] introduced classifier-guided diffusion for a conditioned generation. GLIDE later adapted this approach [31], enabling conditioning on textual CLIP representations. Classifier-free guidance [17] allows for conditioning with a balance between fidelity and diversity, resulting in improved performance [31]. Since the guided diffusion model requires a large number of image-annotation pairs for training, Hu *et al.* [19] propose self-guided diffusion models. Recently, Hyperdiffusion [12, 27] was proposed in the weight space for generating implicit neural fields and 3D reconstruction. In this paper, we introduce ProtoDiff, a prototype-based meta-learning approach within a task-guided diffusion model that incrementally improves the prototype's expressiveness by utilizing a limited number of samples.

5 Experiments

In this section, we assess the efficacy of ProtoDiff in the context of three distinct few-shot learning scenarios: within-domain few-shot learning, cross-domain few-shot learning, and few-task few-shot learning. For the within-domain few-shot learning experiments, we apply our method to three specific datasets: *mini*Imagenet [50], *tiered*Imagenet [35], and ImageNet-800 [5]. Regarding cross-domain few-shot learning, we utilize *mini*Imagenet [50] as the training domain, while testing is conducted on four distinct domains: CropDisease [30], EuroSAT [15], ISIC2018 [47], and ChestX [52]. Furthermore, few-task few-shot learning [56] challenges the common assumption of having abundant tasks available during meta-training. To explore this scenario, we perform experiments on four few-task meta-learning challenges: *mini*Imagenet-S [50], ISIC [29], DermNet-S [6], and Tabular

Table 1: Benefit of ProtoDiff on *mini*Imagenet, *tiered*Imagenet and ImageNet-800. The results of the Classifier-Baseline and Meta-Baseline are provided by Chen *et al.* [5]. ProtoDiff consistently achieves better performance than two baselines on all datasets and settings.

	***mini*Imagenet**		***tiered*Imagenet**		**Imagenet-800**	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Classifier-Baseline [5]	58.91±0.23	77.76±0.17	68.07±0.29	83.74±0.18	86.07±0.21	96.14±0.08
Meta-Baseline [5]	63.17±0.23	79.26±0.17	68.62±0.27	83.74±0.18	89.70±0.19	96.14±0.08
ProtoDiff	**66.63**±0.21	**83.48**±0.15	**72.95**±0.24	**85.15**±0.18	**92.13**±0.20	**98.21**±0.08

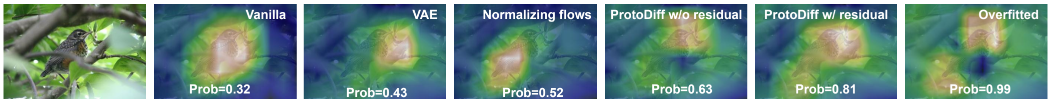


Figure 5: Visualization of the different reconstructed overfitted prototypes using different generative models. The vanilla prototype focuses only on the overall features of the bird and overlooks the finer details of the *robin*. In contrast, the overfitted prototype can highlight specific features such as the tail and beak. While VAE and normalizing flow can generate prototypes of certain parts, the diffusion method can generate prototypes that more closely resemble the overfitted prototype. With residual prototype learning, ProtoDiff achieves better.

Murris [3]. For a comprehensive understanding of the datasets used in each setting, we provide detailed descriptions in the Appendix.

Implementation details. In our within-domain experiments, we utilize a Conv-4 and ResNet-12 backbone for *mini*Imagenet and *tiered*Imagenet. A ResNet-50 is used for ImageNet-800. We follow the approach described in [5] to achieve better performance and initially train a feature extractor on all the meta-training data without episodic training. Standard data augmentation techniques are applied, including random resized crop and horizontal flip. For our cross-domain experiments, we use a ResNet-10 backbone to extract image features, which is a common choice for cross-domain few-shot classification [14, 57]. The training configuration for this experiment is the same as the within-domain training. For few-task few-shot learning, we follow [56] using a network containing four convolutional blocks and a classifier layer. The average within-domain/ cross-domain, few-task few-shot classification accuracy (%, top-1) along with 95% confidence intervals are reported across all test query sets and tasks. Code available at: `https://github.com/YDU-uva/ProtoDiff`.

Benefit of ProtoDiff. Table 1 compares our ProtoDiff with two baselines to demonstrate its effectiveness. The first Classifier-Baseline is a classification model trained on the entire label set using a classification loss and performing few-shot tasks with the cosine nearest-centroid method. The Meta-Baseline [5] consists of two stages. In the first stage, a classifier is trained on all base classes, and the last fully connected layer is removed to obtain the feature encoder. The second stage is meta-learning, where the classifier is optimized using episodic training. The comparison between the Baseline and Meta-Baseline highlights the importance of episodic training for few-shot learning. ProtoDiff consistently outperforms Meta-Baseline [5] by a large margin on all datasets and shots. The task-guided diffusion model employed in our ProtoDiff generates more informative prototypes, leading to improvements over deterministic prototypes.

Benefit of diffusion model. To confirm that the performance gain of our ProtoDiff model can be attributed to the diffusion model, we conducted the experiments only using MLP and transformers as non-generative models. We also compared it with two widely used generative models: the variational autoencoder (VAE) [21] and normalizing flows [36]. VAE learns a low-dimensional representation of input data to generate new samples. Normalizing flows are more recent and learn a sequence of invertible transformations to map a simple prior distribution to the data distribution. We obtained the task-specific prototype in the meta-test stage, conditioned on the support samples and $\epsilon \sim \mathcal{N}(0, I)$, by first acquiring an overfit-

Table 2: Benefit of diffusion model over (non-)generative models on *mini*Imagenet.

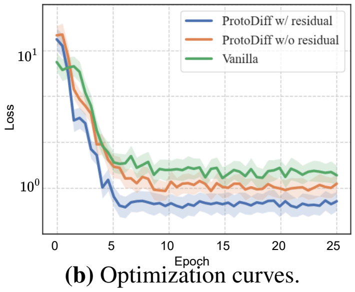
	***mini*Imagenet**	
Method	**1-shot**	**5-shot**
w/o generative model [5]	63.17±0.23	79.26±0.17
w/ MLP	64.15±0.21	80.23±0.15
w/ Transformer	64.97±0.21	81.28±0.14
w/ VAE	64.45±0.22	80.13±0.15
w/ Normalizing flows	65.11±0.22	81.96±0.17
w/ Diffusion	**66.63**±0.21	**83.48**±0.15

ted prototype $\mathbf{z}^*$ and then using it as the ground truth to train VAE and normalizing flows in the meta-training stage. The experimental results reported in Table 2 show that the diffusion model outperforms all variants in terms of accuracy. Specifically, our diffusion model improves accuracy by 2.18% compared to VAE and 1.52% compared to normalizing flows. Furthermore, we reconstructed the overfitted prototype using different generative models in Figure 5. The vanilla prototype focuses solely on the bird's features and overlooks the finer details of the *robin*. In contrast, the overfitted prototype can emphasize specific features such as the tail and beak, resulting in better discrimination of the *robin* from other bird species. While VAE and normalizing flows can generate prototypes of specific parts, our diffusion method can generate prototypes that more closely resemble the overfitted prototype, leading to improved performance. Our findings suggest that diffusion models hold promise as a more practical approach for few-shot learning, as they can model complex distributions and produce informative prototypes.

Effect of the residual prototype. The incorporation of residual prototypes in ProtoDiff presents a viable approach for reducing computational costs. This is attributable to the fact that the residual prototype exclusively encapsulates the disparity between the overfitted prototype and the original prototype, while the latter can be readily reused in each episode. Consequently, the calculation of the overfitted prototype is only necessitated once during the meta-training stage. Notably, the residual prototype often encompasses numerous zero values, thereby further expediting the training process. Table 6a illustrates the superior accuracy achieved by ProtoDiff with residual prototype learning in comparison to both the vanilla model and ProtoDiff without the residual prototype. The training progress of ProtoDiff is visually represented in Figure 6b, demonstrating its accelerated training capabilities when compared to alternative methods. These results indicate the effectiveness of residual prototype learning in capturing task-specific information previously unaccounted for by the vanilla prototypes. Thus, the inclusion of residual prototype learning in ProtoDiff not only expedites the training process but also serves as a straightforward and effective approach to enhance the overall performance of ProtoDiff.

(a) Accuracy (%) on various settings.

	*mini*Imagenet	
Method	**1-shot**	**5-shot**
Vanilla	63.17±0.23	79.26±0.17
ProtoDiff w/o residual	64.75±0.22	80.76±0.16
ProtoDiff w/ residual	**66.63**±0.21	**83.48**±0.15



(b) Optimization curves.

Figure 6: Effect of residual prototype on *mini*Imagenet. Utilizing ProtoDiff with a residual prototype not only accelerates the training process but also surpasses the performance of both the vanilla prototype and the non-residual prototype.

Analysis of uncertainty. In the process of estimating the log-likelihood of input data, the choice of the number of Monte Carlo samples becomes a critical hyperparameter. Typically, a larger number of samples tends to provide a more accurate approximation and better generalization to unforeseen prompts. In our experiments, we employed Meta-Baseline and FEAT [58] with ProtoDiff, incorporating the sampling of multiple prototypes in the final phase of the diffusion process. The results of these experiments are provided in Table 3. Notably, we observe a substantial performance boost with an increase in the number of samples. For instance, a sample size of 50 results in a significant enhancement in 1-shot accuracy, raising it from 66.63 for Meta-Baseline + ProtoDiff to 72.25 for FEAT + ProtoDiff.

Table 3: Analysis of uncertainty. Increasing the number of Monte Carlo samples provides a good improvement on *mini*Imagenet.

	Meta-baseline + ProtoDiff		**FEAT + ProtoDiff**	
	1-shot	**5-shot**	**1-shot**	**5-shot**
1	66.63±0.21	83.48±0.15	68.97±0.25	85.16±0.17
10	68.02±0.17	84.95±0.10	70.18±0.15	86.53±0.12
20	68.91±0.15	85.74±0.10	61.07±0.15	87.14±0.12
50	69.14±0.15	86.12±0.10	72.25±0.15	88.32±0.12
100	69.13±0.15	86.07±0.10	72.21±0.15	88.37±0.12
1000	69.11±0.15	86.11±0.10	72.12±0.15	88.13±0.12

Visualization of the prototype adaptation process. We undertake a qualitative analysis of the adaptation process of our ProtoDiff framework during the meta-test time. We employ a 3-way 5-shot setup on the *mini*ImageNet dataset for more comprehensive visualization of the adaptation process. To achieve this, we depict the vanilla prototypes, the diffused prototypes at various timesteps, and instances from the support and query sets. All examples are first normalized to a length of 1 and

Table 5: Cross-domain few-shot comparison on four benchmarks. ProtoDiff is a consistent top-performer.

	CropDiseases		**EuroSAT**		**ISIC**		**ChestX**	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**
ProtoNet [42]	57.57±0.51	79.72±0.67	54.19±0.57	73.29±0.71	29.62±0.32	39.57±0.57	22.30±0.25	24.05±1.01
GNN [40]	57.19±0.50	83.12±0.40	54.61±0.50	78.69±0.40	30.14±0.30	42.54±0.40	21.94±0.20	23.87±0.20
AFA [20]	67.61±0.50	88.06±0.30	63.12±0.50	85.58±0.40	33.21±0.30	**46.01**±0.40	22.92±0.20	25.02±0.20
ATA [51]	67.47±0.50	**90.59**±0.30	61.35±0.50	83.75±0.40	33.21±0.30	44.91±0.40	22.10±0.20	24.32±0.20
HVM [11]	65.13±0.45	87.65±0.31	61.97±0.34	74.88±0.45	33.87±0.35	42.05±0.34	**22.94**±0.47	27.15±0.45
This paper	**68.93**±0.31	**90.15**±0.31	**65.93**±0.34	**87.25**±0.35	**34.97**±0.33	**45.65**±0.31	**23.01**±0.45	**28.54**±0.41

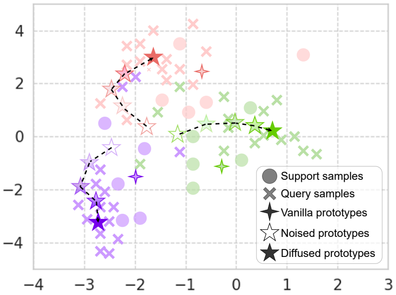


Figure 7: t-SNE visualization of ProtoDiff adaptation process on 3-way 5-shot task.

subsequently projected onto a two-dimensional space using t-SNE [48]. Figure 7 displays the instance representations from both the query and support sets that are cross and circle, the vanilla prototypes that are denoted by a four-point star symbol, an the diffused prototypes that are denoted by stars. Arrows depict the adaptation process of the diffused prototypes, and each color corresponds to each respective class. Initially, the diffused prototypes are randomly allocated based on a Gaussian distribution, positioning them considerably from the optimal prototypes for each class. As the diffusion process continues, these diffused prototypes incrementally approach the optimal prototypes, resulting in more distinctive representations for different classes. In contrast, the vanilla prototypes lack distinguishability as they are computed using a single deterministic function based solely on the support set. This observation underscores the importance of the prototype adaptation process in attaining improved performance in few-shot classification tasks.

Table 4: Within-domain few-shot comparison on *mini*Imagenet and *tiered*Imagenet. ProtoDiff performs well on all datasets.

	***mini*Imagenet**		***tiered*Imagenet**	
Method	**1-shot**	**5-shot**	**1-shot**	**5-shot**
TapNet [59]	61.65±0.15	76.36±0.10	63.08±0.15	80.26±0.12
CTM [26]	62.05±0.55	78.63±0.06	64.78±0.11	81.05±0.52
MetaOptNet [24]	62.64±0.61	78.63±0.46	65.81±0.74	81.75±0.53
Meta-Baseline [5]	63.17±0.23	79.26±0.17	68.62±0.27	83.74±0.18
CAN [18]	63.85±0.48	79.44±0.34	69.89±0.51	84.23±0.37
Meta DeepBDC [53]	67.34±0.43	84.46±0.28	72.34±0.49	87.31±0.32
SUN [9]	67.80±0.45	83.25±0.30	72.99±0.50	86.74±0.33
SetFeat [1]	68.32±0.62	82.71±0.46	73.63±0.88	87.59±0.57
This paper	**71.25**±0.45	**83.95**±0.45	**75.97**±0.75	**88.75**±0.18

Within-domain few-shot. We evaluate our method on few-shot classification within domains, in which the training domain is consistent with the test domain. The results are reported in Table 4. In this comparison, we apply SetFeat [1] with ProtoDiff to experiment since SetFeat is the current state-of-the-art model based on prototype-based meta-learning. Our ProtoDiff consistently achieves state-of-the-art performance on all datasets under various shots. The better performance confirms that ProtoDiff can achieve more informative and higher quality prototypes to perform better for few-shot learning within domains.

Cross-domain few-shot. We also evaluate our method in the cross-domain few-shot classification, where the training domain is different from test domain. Table 5 shows the evaluation of ProtoDiff on four datasets, with 5-way 1-shot and 5-way 5-shot settings. ProtoDiff also achieves competitive performance on all four cross-domain few-shot learning benchmarks for each setting. On EuroSAT [15], our model obtains high recognition accuracy under different shot configurations, outperforming the second best method, AFA [20], by a significant margin of 2.82% for 5-way 1-shot. Even on the most challenging ChestX [52], which has a considerable domain gap with *mini*ImageNet, our model achieves 28.54% accuracy in the 5-way 5-shot setting, surpassing the second-best HVM [11] by 1.39%. The consistent improvement across all benchmarks and settings confirms the effectiveness of ProtoDiff for cross-domain few-shot learning.

Few-task few-shot. We evaluate ProtoDiff on the four different datasets under 5-way 1-shot and 5-way 5-shot in Table 6. In this comparison, we apply MLTI [56] with ProtoDiff to experiment. Our method achieves state-of-the-art performance on all four few-task meta-learning benchmarks under each setting. On *mini*Imagenet-S, our model achieves 44.15% under 1-shot, surpassing the

Table 6: Few-task few-shot comparison on four benchmarks. ProtoDiff consistently ranks among the top-performing models.

	*mini*Imagenet-S		ISIC		Dermnet-S		Tabular Murris	
	1-shot	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**
ProtoNet [42]	36.26±0.70	50.72±0.70	58.56±1.01	66.25±0.96	44.21±0.75	60.33±0.70	80.03±0.90	89.20 ±0.56
Meta-Dropout [23]	38.32±0.75	52.53±0.70	58.40±1.16	67.32±0.95	44.30±0.81	60.86±0.75	78.18±0.90	89.25±0.56
MetaMix [55]	39.80±0.87	53.35±0.88	59.66±1.10	68.97±0.80	46.06±0.85	62.97±0.70	79.56±0.91	88.88±0.60
Meta Interpolation [25]	40.28±0.70	53.06±0.72	-	-	-	-	-	-
MLTI [56]	41.36±0.73	55.34±0.72	62.82±1.09	71.52±0.89	49.38±0.85	65.19±0.73	81.89±0.88	90.12±0.59
MetaModulation [45]	43.21±0.73	57.26±0.72	65.61±1.09	**76.40**±0.89	50.45±0.84	67.05±0.73	83.13±0.89	91.23±0.57
This paper	**44.75**±0.70	**58.18**±0.72	**66.13**±1.04	76.23±0.81	**51.53**±0.85	**67.97**±0.74	**84.03**±0.89	**91.35**±0.57

best method MetaModulation [45], by a margin of 1.54%. Even on the most challenging DermNet-S, which forms the largest dermatology dataset, our model delivers 51.53% on the 5-way 1-shot setting. The consistent improvements on all benchmarks under various configurations confirm that our approach is also effective for few-task meta-learning.

Limitations. Naturally, our proposal also comes with limitations. Firstly, the diffusion model employed in our approach necessitates a substantial number of timesteps to sample the diffused prototype during the meta-test stage. Although we somewhat alleviate this issue by utilizing DDIM [44] sampling, it still requires more computational resources than vanilla prototypes. Secondly, obtaining the overfitted prototype involves fine-tuning, which inevitably leads to increased training time. While this fine-tuning step contributes to the effectiveness of our model, it comes with an additional computational cost. In meta-training and meta-testing ProtoDiff is slower by factors of ProtoNet 5× and 15× in terms of wall-clock times per task. More detailed time numbers are provided in the Appendix. As part of future work, we will investigate and address these limitations to further enhance the applicability and efficiency of our approach.

6 Conclusion

We proposed ProtoDiff, a novel approach to prototype-based meta-learning that utilizes a task-guided diffusion model during the meta-training phase to generate efficient class representations. We addressed the limitations of estimating a deterministic prototype from a limited number of examples by optimizing a set of prototypes to accurately represent individual tasks and training a task-guided diffusion process to model each task's underlying distribution. Our approach considers both probabilistic and task-guided prototypes, enabling efficient adaptation to new tasks while maintaining the informativeness and scalability of prototypical networks. Extensive experiments on three distinct few-shot learning scenarios: within-domain, cross-domain, and few-task few-shot learning validated the effectiveness of ProtoDiff. Our results demonstrated significant improvements in classification accuracy compared to state-of-the-art meta-learning techniques, highlighting the potential of task-guided diffusion in augmenting few-shot learning and advancing meta-learning.

Acknowledgment

This work is financially supported by the Inception Institute of Artificial Intelligence, the University of Amsterdam and the allowance Top consortia for Knowledge and Innovation (TKIs) from the Netherlands Ministry of Economic Affairs and Climate Policy.

References

[1] A. Afrasiyabi, H. Larochelle, J.-F. Lalonde, and C. Gagné. Matching feature sets for few-shot image classification. In *CVPR*, pages 9014–9024, 2022.

[2] K. R. Allen, E. Shelhamer, H. Shin, and J. B. Tenenbaum. Infinite mixture prototypes for few-shot learning. In *ICML*, pages 232–241, 2019.

[3] K. Cao, M. Brbic, and J. Leskovec. Concept learners for few-shot learning. *ICLR*, 2020.

[4] T. Cao, M. Law, and S. Fidler. A theoretical analysis of the number of shots in few-shot learning. *arXiv preprint arXiv:1909.11722*, 2019.

[5] Y. Chen, Z. Liu, H. Xu, T. Darrell, and X. Wang. Meta-baseline: Exploring simple meta-learning for few-shot learning. In *ICCV*, pages 9062–9071, 2021.

[6] Dermnet. Dermnet dataset. https://dermnet.com/. 2016.

[7] P. Dhariwal and A. Nichol. Diffusion models beat gans on image synthesis. *NeurIPS*, 34:8780–8794, 2021.

[8] N. Ding, X. Wang, Y. Fu, G. Xu, R. Wang, P. Xie, Y. Shen, F. Huang, H.-T. Zheng, and R. Zhang. Prototypical representation learning for relation extraction. *arXiv preprint arXiv:2103.11647*, 2021.

[9] B. Dong, P. Zhou, S. Yan, and W. Zuo. Self-promoted supervision for few-shot transformer. In *ECCV*, pages 329–347. Springer, 2022.

[10] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

[11] Y. Du, X. Zhen, L. Shao, and C. G. M. Snoek. Hierarchical variational memory for few-shot learning across domains. In *ICLR*, 2022.

[12] Z. Erkoç, F. Ma, Q. Shan, M. Nießner, and A. Dai. Hyperdiffusion: Generating implicit neural fields with weight-space diffusion. *arXiv preprint arXiv:2303.17015*, 2023.

[13] V. Garcia and J. Bruna. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[14] Y. Guo, N. C. Codella, L. Karlinsky, J. V. Codella, J. R. Smith, K. Saenko, T. Rosing, and R. Feris. A broader study of cross-domain few-shot learning. In *European Conference on Computer Vision*, 2020.

[15] P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 12(7):2217–2226, 2019.

[16] J. Ho, A. Jain, and P. Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 33:6840–6851, 2020.

[17] J. Ho and T. Salimans. Classifier-free diffusion guidance. *arXiv preprint arXiv:2207.12598*, 2022.

[18] R. Hou, H. Chang, M. Bingpeng, S. Shan, and X. Chen. Cross attention network for few-shot classification. In *NeurIPS*, pages 4005–4016, 2019.

[19] V. T. Hu, D. W. Zhang, Y. M. Asano, G. J. Burghouts, and C. G. M. Snoek. Self-guided diffusion models. In *CVPR*, 2023.

[20] Y. Hu and A. J. Ma. Adversarial feature augmentation for cross-domain few-shot classification. In *ECCV*, pages 20–37. Springer, 2022.

[21] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[22] G. Kwon and J. C. Ye. Diffusion-based image translation using disentangled style and content representation. *arXiv preprint arXiv:2209.15264*, 2022.

[23] H. B. Lee, T. Nam, E. Yang, and S. J. Hwang. Meta dropout: Learning to perturb latent features for generalization. In *ICLR*, 2020.

[24] K. Lee, S. Maji, A. Ravichandran, and S. Soatto. Meta-learning with differentiable convex optimization. In *CVPR*, pages 10657–10665, 2019.

[25] S. Lee, B. Andreis, K. Kawaguchi, J. Lee, and S. J. Hwang. Set-based meta-interpolation for few-task meta-learning. In *NeurIPS*, 2022.

[26] H. Li, D. Eigen, S. Dodge, M. Zeiler, and X. Wang. Finding task-relevant features for few-shot learning by category traversal. In *CVPR*, pages 1–10. Computer Vision Foundation / IEEE, 2019.

[27] S. S. Lutati and L. Wolf. Ocd: Learning to overfit with conditional diffusion models. *arXiv preprint arXiv:2210.00471*, 2022.

[28] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng. Nerf: Representing scenes as neural radiance fields for view synthesis. *Communications of the ACM*, 65(1):99–106, 2021.

[29] M. A. A. Milton. Automated skin lesion classification using ensemble of deep neural networks in isic 2018: Skin lesion analysis towards melanoma detection challenge. *arXiv preprint arXiv:1901.10802*, 2019.

[30] S. P. Mohanty, D. P. Hughes, and M. Salathé. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7:1419, 2016.

[31] A. Nichol, P. Dhariwal, A. Ramesh, P. Shyam, P. Mishkin, B. McGrew, I. Sutskever, and M. Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. *arXiv preprint arXiv:2112.10741*, 2021.

[32] B. Oreshkin, P. R. López, and A. Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, pages 721–731, 2018.

[33] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[34] S. K. Reed. Pattern recognition and categorization. *Cognitive psychology*, 3(3):382–407, 1972.

[35] 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.

[36] D. Rezende and S. Mohamed. Variational inference with normalizing flows. In *ICML*, pages 1530–1538. PMLR, 2015.

[37] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. Berg, and L. Fei-Fei. ImageNet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[38] C. Saharia, W. Chan, H. Chang, C. Lee, J. Ho, T. Salimans, D. Fleet, and M. Norouzi. Palette: Image-to-image diffusion models. In *ACM SIGGRAPH*, pages 1–10, 2022.

[39] H. Sasaki, C. G. Willcocks, and T. P. Breckon. Unit-ddpm: Unpaired image translation with denoising diffusion probabilistic models. *arXiv preprint arXiv:2104.05358*, 2021.

[40] V. G. Satorras and J. B. Estrach. Few-shot learning with graph neural networks. In *ICLR*, 2018.

[41] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *ICCV*, pages 618–626, 2017.

[42] J. Snell, K. Swersky, and R. Zemel. Prototypical networks for few-shot learning. In *NeurIPS*, pages 4077–4087, 2017.

[43] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*, pages 2256–2265. PMLR, 2015.

[44] J. Song, C. Meng, and S. Ermon. Denoising diffusion implicit models. *arXiv preprint arXiv:2010.02502*, 2020.

[45] W. Sun, Y. Du, X. Zhen, F. Wang, L. Wang, and S. Cees, G.M. Metamodulation: Learning variational feature hierarchies for few-shot learning with fewer tasks. In *ICML*. PMLR, 2023.

[46] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, and H. Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *arXiv preprint arXiv:1903.03096*, 2019.

[47] P. Tschandl, C. Rosendahl, and H. Kittler. The ham10000 dataset, a large collection of multi-source dermatoscopic images of common pigmented skin lesions. *Scientific data*, 5(1):1–9, 2018.

[48] L. Van der Maaten and G. Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[49] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *NeurIPS*, pages 6000–6010, 2017.

[50] O. Vinyals, C. Blundell, T. Lillicrap, D. Wierstra, et al. Matching networks for one shot learning. In *NeurIPS*, pages 3630–3638, 2016.

[51] H. Wang and Z.-H. Deng. Cross-domain few-shot classification via adversarial task augmentation. *arXiv preprint arXiv:2104.14385*, 2021.

[52] X. Wang, Y. Peng, L. Lu, Z. Lu, M. Bagheri, and R. M. Summers. Chestx-ray8: Hospital-scale chest x-ray database and benchmarks on weakly-supervised classification and localization of common thorax diseases. In *CVPR*, pages 2097–2106, 2017.

[53] J. Xie, F. Long, J. Lv, Q. Wang, and P. Li. Joint distribution matters: Deep brownian distance covariance for few-shot classification. In *CVPR*, pages 7972–7981, 2022.

[54] S. Yang, L. Liu, and M. Xu. Free lunch for few-shot learning: Distribution calibration. *arXiv preprint arXiv:2101.06395*, 2021.

[55] H. Yao, L.-K. Huang, L. Zhang, Y. Wei, L. Tian, J. Zou, J. Huang, et al. Improving generalization in meta-learning via task augmentation. In *ICML*, pages 11887–11897. PMLR, 2021.

[56] H. Yao, L. Zhang, and C. Finn. Meta-learning with fewer tasks through task interpolation. *arXiv preprint arXiv:2106.02695*, 2021.

[57] H. Ye, H. Hu, D. Zhan, and F. Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *CVPR*, 2020.

[58] H.-J. Ye, H. Hu, D.-C. Zhan, and F. Sha. Few-shot learning via embedding adaptation with set-to-set functions. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 8808–8817, 2020.

[59] S. Yoon, J. Seo, and J. Moon. Tapnet: Neural network augmented with task-adaptive projection for few-shot learning. In *ICML*, 2019.

[60] S. W. Yoon, J. Seo, and J. Moon. Tapnet: Neural network augmented with task-adaptive projection for few-shot learning. In *ICML*, pages 7115–7123. PMLR, 2019.

[61] X. Zhen, Y. Du, H. Xiong, Q. Qiu, C. G. M. Snoek, and L. Shao. Learning to learn variational semantic memory. In *NeurIPS*, 2020.

A Algorithms

We describe the detailed algorith ms for meta-training and meta-test of ProtoDiff as following Algorithm 1 and 2, respectively:

Algorithm 1 Meta-training phase of ProtoDiff.
Input: $p(\mathcal{T})$: distribution over tasks; θ: diffusion parameters; ϕ: feature extractor parameters; T: diffusion steps.
Output: $\mathbf{z}_\theta$: diffusion network, f_ϕ: feature extractor.

```
repeat
    Sample batch of tasks $\mathcal{T}^i \sim p(\mathcal{T})$
    for all $\mathcal{T}^i$ do
        Sample support and query set $\{\mathcal{S}^i, \mathcal{Q}^i\}$ from $\mathcal{T}^i$
        Compute the vanilla prototype $\tilde{\mathbf{z}}^i = f(\mathcal{S}^i)$
        repeat
            Take some gradient step on $\nabla \mathcal{L}_{\text{CE}}(\mathcal{S}^i, \mathcal{Q}^i)$, updating $\phi^i$
        until $\mathcal{L}_{\text{CE}}(\mathcal{S}_i, \mathcal{Q}_i)$ converges
        Compute the overfitted prototype by using the updated $\phi^i$, $\mathbf{z}^{i,*} = f_{\phi^i}(\mathcal{S}_i)$
        $t \sim Uniform(1...T)$
        $\epsilon = \mathcal{N}(\mathbf{0}, \mathbf{1})$
        $\beta_t = \frac{10^{-4}(T-t)+10^{-2}(t-1)}{T-1}$,$\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \Pi_{k=0}^{k=t}\alpha_k$
        $\hat{\mathbf{z}}_t^i = \sqrt{\bar{\alpha}_t}(\mathbf{z}^{i,*} - \tilde{\mathbf{z}}^i) + \sqrt{1 - \bar{\alpha}_t{}^2}\epsilon$
        Compute diffusion loss $\mathcal{L}_{\text{diff}}$ with Equation (9) and the final loss $\mathcal{L}_{\mathcal{T}^i}$ with Equation (10)
        Update $\{\phi, \theta\} \leftarrow \{\phi, \theta\} - \beta\nabla_{\{\phi,\theta\}} \sum_{\mathcal{T}^i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}^i}$ using query data of each task.
    end for
until $f_\phi$ and $\mathbf{z}_\theta$ converges
```

Algorithm 2 Meta-test phase of ProtoDiff.
Input: $\tau = \{\mathcal{S}, \mathcal{Q}\}$: meta-test task, $\mathbf{z}_\theta$: trained diffusion network parameters, f_ϕ: trained feature extractor network parameters, T: diffusion steps.

```
$\mathbf{z}_T \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$
Compute vanilla prototype $\tilde{\mathbf{z}}$ with support set $f(\mathcal{S})$
for t=$T, \cdots, 1$ do
    $\epsilon = N(\mathbf{0}, \mathbf{1})$
    $\beta_t = \frac{10^{-4}(T-t)+10^{-2}(t-1)}{T-1}$,$\alpha_t = 1 - \beta_t$, $\bar{\alpha}_t = \Pi_{k=0}^{k=t}\alpha_k$
    $\mathbf{z}_{t-1} = \mathbf{z}_\theta(\mathbf{z}_t, \tilde{\mathbf{z}}, t)$
end for
Compute the final prediction $\mathbf{y}^q$ by with Equation (1) based on $\mathbf{z}_0 + \tilde{\mathbf{z}}$
```

B Per-task prototype overfitting architecture

To enhance our comprehension of the Per-task prototype overfitting part, we propose a succinct architectural representation depicted in Figure 8. The initial step entails the computation of the conventional prototypes $\tilde{\mathbf{z}}$ for a meta-training task. Subsequently, Equation (1) is employed to calculate the predictions for the query sample. The backbone's parameters are subsequently updated through I iterations. Through the utilization of parameters from the final iteration, we ultimately obtain the prototypes $\mathbf{z}^*$ that exhibit overfitting characteristics.

C Residual prototype learning architecture

In order to gain a more comprehensive understanding of our residual prototype learning, we have crafted a succinct architecture diagram illustrated in Figure 9. Our proposition involves the prediction of the prototype update, denoted as $\mathbf{z}^* - \tilde{\mathbf{z}}$, instead of directly predicting the overfitted prototype

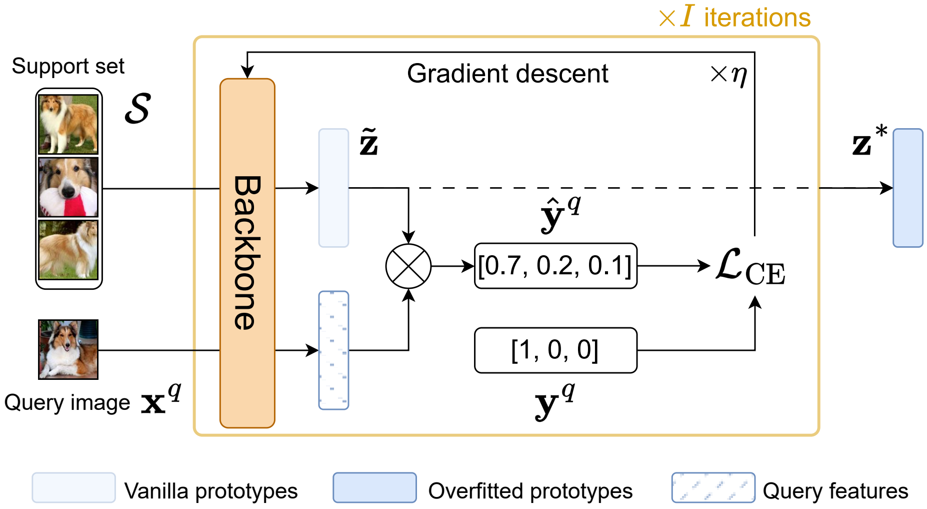


Figure 8: Per-task prototype overfitting.

$\mathbf{z}^*$. This distinctive approach also allows us to initialize ProtoDiff with the capability to perform the identity function, achieved by assigning zero weights to the decoder. Notably, we have discovered that the amalgamation of a global residual connection and the identity initialization substantially expedites the training process. By harnessing this mechanism, we have successfully enhanced the performance of ProtoDiff in the context of few-shot learning tasks.

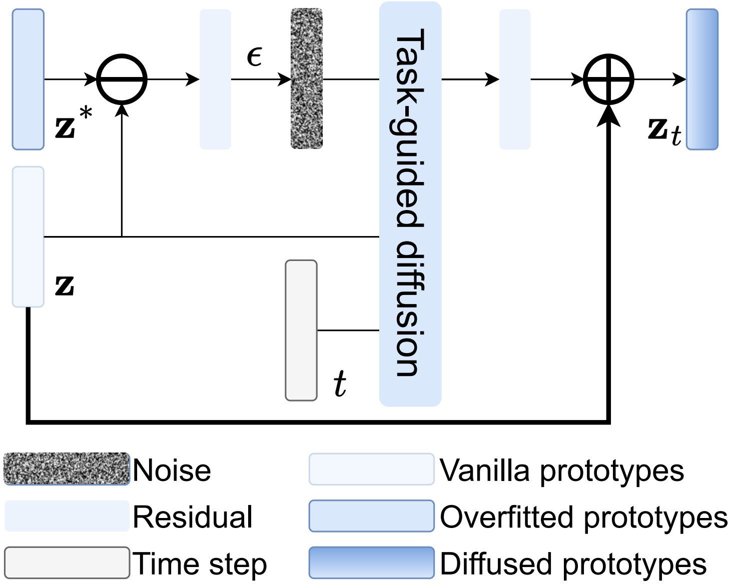


Figure 9: Residual prototype learning.

D Datasets

Within-domain few-shot. For this setting we focus on 5-way 1-shot/5-shot tasks, which aligns with previous research [42]. The within-domain few-shot experiments are performed on three datasets: *mini*Imagenet [50] *tiered*Imagenet [35], and ImageNet-800 [5]. *mini*Imagenet consists of 100 randomly selected classes from ILSVRC-2012 [37], while *tiered*Imagenet is composed of 608 classes that are grouped into 34 high-level categories. We measure the accuracy of 600 tasks randomly sampled from the meta-test set to evaluate the performance. Following [5], we also evaluate our model on ImageNet-800, a dataset obtained by randomly dividing the 1,000 classes of ILSVRC-2012 into 800 base classes and 200 novel classes. The base classes consist of images from the original training set, while the novel classes comprise images from the original validation set.

Cross-domain few-shot. In the 5-way 5-shot cross-domain few-shot classification experiments, the training domain is *mini*Imagenet [50], and the testing is conducted on four different domains. These domains are CropDisease [30], which contains plant disease images; EuroSAT [15], a collection of satellite images; ISIC2018 [47], consisting of dermoscopic images of skin lesions, and ChestX [52], a dataset of X-ray images.

Few-task few-shot. Few-task few-shot learning [56] challenges the common meta-training assumption of having many tasks available. We conductt experiments on four few-task meta-learning challenges, namely *mini*Imagenet-S [50], ISIC [29], DermNet-S [6], and Tabular Murris [3]. To reduce the number of tasks and make it comparable to previous works, we followed [56] by limiting the number of meta-training classes to 12 for miniImagenet-S, with 20 meta-test classes. ISIC [29] involves classifying dermoscopic images across nine diagnostic categories, with 10,015 images available for training in eight different categories, of which we selected four as meta-training classes. DermNet [6], which contains over 23,000 images of skin diseases, was utilized to construct Dermnet-S by selecting 30 diseases as meta-training classes, following [56]. Finally, Tabular Murris [3], which deals with cell type classification across organs and includes nearly 100,000 cells from 20 organs and tissues, was utilized to select 57 base classes as meta-training classes, again following the same approach as [56].

E Implementation details

In our within-domain experiments, we utilize a Conv-4 and ResNet-12 backbone for *mini*Imagenet and *tiered*Imagenet. A ResNet-50 is used for ImageNet-800. We follow the approach described in [5] to achieve better performance and initially train a feature extractor on all the meta-training data without episodic training. We use the SGD optimizer with a momentum of 0.9, a learning rate starting from 0.1, and a decay factor of 0.1. For *mini*Imagenet, we train for 100 epochs with a batch size of 128, where the learning rate decays at epoch 90. For tieredImageNet, we train for 120 epochs with a batch size of 512, where the learning rate decays at epochs 40 and 80. Lastly, for ImageNet-800, we train for 90 epochs with a batch size of 256, where the learning rate decays at epochs 30 and 60. The weight decay is 0.0005 for ResNet-12 and 0.0001 for ResNet-50. Standard data augmentation techniques, including random resized crop and horizontal flip, are applied. For episodic training, we use the SGD optimizer with a momentum of 0.9, a fixed learning rate of 0.001, and a batch size of 4, meaning each training batch consists of 4 few-shot tasks to calculate the average loss. For our cross-domain experiments, we use a ResNet-10 backbone to extract image features, which is a common choice for cross-domain few-shot classification [14, 57]. The training configuration for this experiment is the same as the within-domain *mini*Imagenet training. For few-task few-shot learning, we follow [56] using a network containing four convolutional blocks and a classifier layer. Each block comprises a 32-filter 3×3 convolution, a batch normalization layer, a ReLU nonlinearity, and a 2×2 max pooling layer. All experiments are performed on a single A100 GPU, each taking approximately 20 hours. We will release all our code.

F Visualization of diffusion process

The ProtoDiff method utilizes a task-guided diffusion model to generate prototypes that provide efficient class representations, as discussed in the previous section. To better understand the effectiveness of our proposed approach, we provide a visualization by Grad-Cam [41] in Figure of the diffusion process, demonstrating how ProtoDiff gradually aggregates towards the desired class prototype during meta-training. The vanilla prototype is shown in the first row on the left, which does not exclusively focus on the *guitar*. In contrast, the overfitted prototype in the second row on the left provides the highest probability for the *guitar*. ProtoDiff, with the diffusion process, randomly selects certain areas to add noise and perform diffusion, resulting in a prototype that gradually moves towards the *guitar* with the highest probability at t=0. Moreover, ProtoDiff with residual learning produces a more precise prototype. The comparison between these different prototypes demonstrates

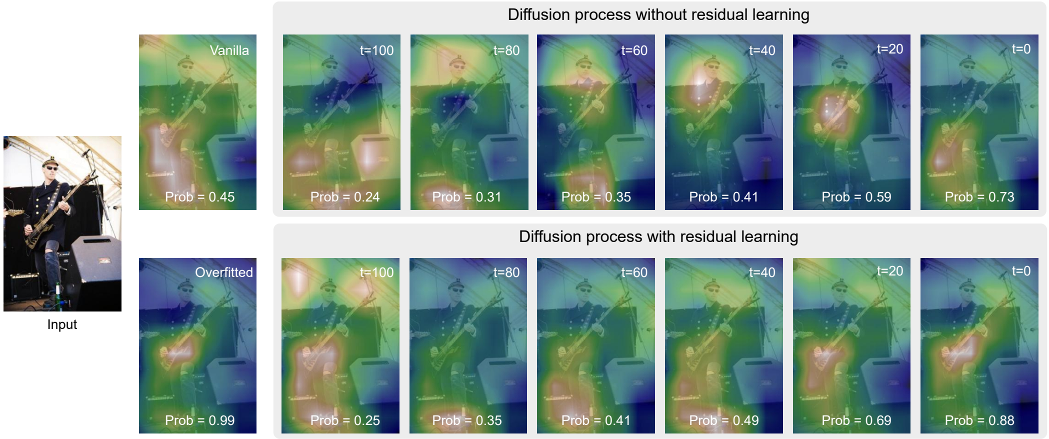


Figure 10: Visualization of the diffusion process. The first row on the right shows the vanilla prototype, which does not exclusively focus on the *guitar*. In contrast, the overfitted prototype in the second row on the right provides the highest probability for the *guitar*. ProtoDiff randomly selects certain areas to predict during the diffusion process, with the lowest probability at the beginning time step. As time progresses, the prototype gradually aggregates towards the *guitar*, with the highest probability at t=0. In comparison, ProtoDiff with residual learning produces a more precise prototype.

the effectiveness of the ProtoDiff diffusion process in generating a more accurate and informative prototype for few-shot learning tasks.

G More results

Effect of transformer structure Our ProtoDiff model is constructed using a design inspired by GPT-2. This includes a 12-layer transformer, a 512-dimensional linear transformation, an attention mechanism with 16 heads, and an MLP with a hidden dimensionality 512. These configurations are in line with GPT-2's default parameters. The configuration files can be accessed in our code repository for more detailed parameter setup. Our experiments in the table 7a and 7b highlight that our model achieves optimal performance with these settings.

Table 7: Effect of transformer structures

(a) Results on the different transformer structures.

	*mini*Imagenet	
Structures	**1-shot**	**5-shot**
3-layers	62.17±0.25	78.93±0.17
6-layers	63.25±0.22	79.63±0.15
9-layers	65.21±0.21	80.13±0.18
12-layers	66.63±0.21	83.48±0.15
15-layers	**64.15**±0.25	**80.93**±0.18

(b) Results on the different heads numbers.

	*mini*Imagenet	
Nodes numbers	**1-shot**	**5-shot**
node = 1	63.28±0.24	79.97±0.15
node = 4	64.43±0.22	80.13±0.14
node = 8	65.91±0.23	81.91±0.16
node = 16	**66.63**±0.21	**83.48**±0.15

Effect of different time steps We have selected the diffusion time step T for the diffusion process to be 100. We've adopted the DDIM sampling strategy to accelerate prediction with . This effectively reduces the total sample requirement to just 10. We've conducted comparative experiments using varying diffusion times and intermediate intervals. As presented in the tables 8 and 9, we observe that as the diffusion timesteps increase, both the performance and the inference time increase. Simultaneously, when is increased, the inference time decreases, making the process more efficient.

Effect of more support images To prove more support images during meta-training to obtain more accurate prototypes, we conducted an experimental comparison using different numbers of support sets during meta-training. The results in the table 10 illustrate that augmenting the number of support images for each class during the meta-training phase enhances performance across various shots.

Table 8: Results with the different diffusion timesteps on ***mini***Imagenets.

dim(τ) = 10, **Timesteps**	**1-shot**	**5-shot**	**1-shot inference time**	**5-shot inference time**
T =10	63.98±0.24	80.12 ±0.15	1 ms	2 ms
T = 50	65.93±0.21	82.93±0.17	5 ms	7 ms
T = 100	66.63±0.21	83.48±0.15	9 ms	14 ms
T = 500	66.65±0.23	83.59±0.14	53 ms	93 ms
T = 1000	66.74±0.20	83.97±0.17	102 ms	192 ms

Table 9: Results with the different diffusion timesteps on ***mini***Imagenets.

dim(τ), T = 100	**1-shot**	**5-shot**	**Speed**
τ =1	66.78±0.21	83.62 ±0.14	99 ms
τ = 5	66.15±0.23	83.44±0.16	45 ms
τ = 10	66.63±0.21	83.48 ±0.15	9 ms
τ = 20	66.12±0.21	82.79 ±0.13	5 ms
τ = 100	63.15±0.23	81.27 ±0.14	1 ms

Meta-training and meta-test wall-clocks times per task We have compiled the wall-clock time for both the meta-training and meta-testing phases of ProtoDiff and compared these against the respective times for ProtoNet in the Tables 11a and 11b. In meta-training and meta-testing wall-clock times per task, our ProtoDiff is slower by factors of ProtoNet 5× and 15 ×, respectively. As part of future work, we will investigate and address this limitation to further enhance the efficiency of our approach.

Rationale for Conditioning We utilize the "vanilla" prototype as a guiding condition, gradually enabling the diffusion model to generate class-specific prototypes. We also explored two alternative strategies for conditioning in the table 12: one without any conditions and another with learned class embeddings. When not conditioned, the performance tends to be arbitrary due to the absence of class-specific cues during diffusion. On the other hand, employing learned class embeddings as a condition yielded subpar results compared to the vanilla prototype, potentially due to outliers in each class's learned embeddings.

Results of our pre-trained model We also give the results of our own pre-trained model before applying ProtoDiff to provide a clear comparison and demonstrate our method's improvements. We have prepared the results of our pre-trained model (trained with CE loss on the whole training set) and will present it in table 13.

Table 10: Results of the different support sets for each class.

	***mini*Imagenet**	
Structures	**1-shot**	**5-shot**
1	66.63±0.21	73.12±0.15
5	70.25±0.22	83.48±0.13
8	73.91±0.21	84.72±0.13
16	82.21±0.23	84.98±0.17

Table 11: Meta-training and meta-test wall-clock times on each task.

(a) Meta-training.

	***mini*Imagenet**	
Meta-training	**1-shot**	**5-shot**
ProtoNet	0.6 ms	1.2 ms
ProtoDiff	3 ms	5.3 ms

(b) Meta-test.

	***mini*Imagenet**	
Meta-test	**1-shot**	**5-shot**
ProtoNet	0.6 ms	1.2 ms
ProtoDiff	9 ms	14 ms

Table 12: Results of different conditions.

	***mini*Imagenet**	
Conditions	**1-shot**	**5-shot**
w/o conditions	20.27±0.25	22.32±0.15
Learnt class embeddings	66.17 ±0.22	81.17±0.16
Vanilla prototype	66.63 ±0.21	83.48±0.15

Table 13: Results of our pre-trained model on different datasets

		***mini*Imagenet**		***tiered*Imagenet**		**Imagenet-800**	
		1-shot	**5-shot**	**1-shot**	**5-shot**	**1-shot**	**5-shot**
Pretrained from Chen *et al.* [5]	Classifier-Baseline	58.91±0.23	77.76±0.17	68.07±0.29	83.74±0.18	86.07±0.21	96.14±0.08
Pretrained from Chen *et al.* [5]	**ProtoDiff**	**66.63**±0.21	**83.48**±0.15	**72.95**±0.24	**85.15**±0.18	**92.13**±0.20	**98.21**±0.08
Pretrained from ours	Classifier-Baseline	58.75±0.21	77.86±0.17	68.15±0.28	83.95±0.17	85.97±0.22	96.03±0.09
Pretrained from ours	**ProtoDiff**	**66.49**±0.25	**83.51**±0.14	**73.01**±0.24	**85.72**±0.17	**92.05**±0.21	**98.11**±0.18