Generalizing from a Few Examples: A Survey on Few-Shot Learning

YAQING WANG, Hong Kong University of Science and Technology and 4Paradigm Inc
QUANMING YAO, 4Paradigm Inc
JAMES T. KWOK, Hong Kong University of Science and Technology
LIONEL M. NI, Hong Kong University of Science and Technology

Artificial intelligence succeeds in data-intensive applications, but it lacks the ability of learning from a limited number of examples. To tackle this problem, Few-Shot Learning (FSL) is proposed. It can rapidly generalize from new tasks of limited supervised experience using prior knowledge. To fully understand FSL, we conduct a survey study. We first clarify a formal definition for FSL. Then we figure out that the unreliable empirical risk minimizer is the core issue of FSL. Based on how prior knowledge is used to deal with the core issue, we categorize different FSL methods into three perspectives: data uses the prior knowledge to augment the supervised experience, model constrains the hypothesis space by prior knowledge, and algorithm uses prior knowledge to alter the search for the parameter of the best hypothesis in the hypothesis space. Under this unified taxonomy, we provide thorough discussion of pros and cons across different categories. Finally, we propose possible directions for FSL in terms of problem setup, techniques, applications and theories, in hope of providing insights to following research.

Additional Key Words and Phrases: Few-shot learning, One-shot learning, Low-shot learning, Small sample learning, Meta-learning, Prior knowledge

1 INTRODUCTION

"Can machines think [122]? " This is the question raised in Alan Turing's seminal paper entitled "Computing Machinery and Intelligence" in 1950. He made the statement that, "The idea behind digital computers may be explained by saying that these machines are intended to carry out any operations which could be done by a human computer". In other words, the ultimate goal of machines is to be as intelligent as humans. Recent years, due to the appearance of powerful computing devices such as GPU, large-scale data sets such as ImageNet [26], advanced models and algorithms such as CNN [64], AI speeds up its pace to be like humans and defeats humans in many fields. To name a few, AlphaGo [106] has defeated human champions in playing the ancient game of go, and ResNet [50] has a higher classification accuracy than humans on ImageNet data set of 1000 classes. While in other fields, AI involves in human's daily life as highly intelligent tools, such as voice assistants, search engines, autonomous driving cars , and industrial robots.

Albeit its prosperity, current AI cannot rapidly generalize from a few examples to perform the task. The aforementioned successful applications of AI rely on exhaustive learning from large-scale data. In contrast, humans are capable of learning new task scenarios rapidly by utilizing what they learned in the past. For example, a child who learned how to add can rapidly transfer his knowledge to learn how to multiply given a few examples, e.g., $2 \times 3 = 2 + 2 + 2$ and $1 \times 3 = 1 + 1 + 1$. Another example is that given a few photos of a stranger, a child can easily identify the same person from a large number of photos.

Bridging this gap between AI and human-like learning is an important direction. This can be tackled by turning to *machine learning*, a sub-field of AI which supports its scientific study bases such as models, algorithms and theories. Specifically, machine learning is concerned with the question of how to construct computer programs that automatically improve with experience [80]. For the thirst of learning from limited supervised information to get the hang of the task, a new machine learning problem called *Few-Shot Learning* (FSL) [32, 33] is proposed. When there is only one example to learn, FSL is also called one-shot learning problem. FSL can learn a new task of limited supervised information by incorporating prior knowledge.

FSL acts as a test-bed for AI. Therefore, it is tested on whether it learns like human. A typical example is character recognition [66], where computer programs are asked to classify, parse and generate new handwritten characters given a few examples. To deal with this task, one can decompose the characters into smaller parts transferable across characters, then aggregate these smaller components into new characters. This is a way of learning like human [67]. Naturally, FSL advances the development of robotics [24] which targets at developing machines that can replicate human actions so as to replace humans in some scenarios. Examples are one-shot imitation [30], multi-armed bandits [30], visual navigation [30], continuous control in locomotion [34].

Apart from testing for AI, FSL can also help to relieve the burden of collecting large-scale supervised data for industrial needs. For example, ResNet [50] obtains a higher classification accuracy than humans on ImageNet data of 1000 classes. However, this is under the circumstances where each class has sufficient labeled images. In contrast, human can recognize around 30,000 classes [16], where collecting sufficient images of each class for machines to learn is very laborious. Instead, FSL can help reduce the data gathering effort for these data intensive applications, such as image classification [127], image retrieval [118], object tracking [15], gesture recognition [88], image captioning and visual question answering [28], video event detection [137], and language modeling [127]. Moreover, being able to perform FSL can reduce the cost of those computationally expensive applications such as one-shot architecture search [21]. When the models and algorithms succeed for FSL, they naturally apply to data sets of many samples which are easier to learn.

Another classic scenario for FSL is tasks where supervised information is hard or impossible to acquire due to some reason, such as privacy, safety or ethic issues. For example, drug discovery is a process of discovering the properties of new molecules so as to identify useful ones as new drugs [3]. However, due to possible toxicity, low activity, and low solubility, these new molecules do not have many real biological records on clinical candidates. This makes the drug discovery task a FSL problem. Similar rare case learning applications can be FSL translation [56], cold-start item recommendation [126], where the target tasks do not have many examples. It is through FSL that learning suitable models for these rare cases becomes possible.

With both academic dream of AI and industrial needs for cheap learning, FSL draws much attention and becomes a hot topic. As a learning paradigm, many methods endeavor to solve it, such as meta-learning method [100], embedding learning method [127] and generative modeling method [31]. However, there is no existing work that provides an organized taxonomy to connect FSL methods, explains why some methods work while others fail, nor discusses the pros and cons of different works. Therefore, we conduct a survey on FSL problem. Contributions of this survey are summarized as follows [1].

- We give the formal definition for FSL. It can naturally link to the classic machine learning definition proposed in [80]. The definition is not only general enough to include all existing FSL

[1]In comparison to existing survey [104], they focus on concept learning and experience learning for small sample. In contrast, our survey focuses on FSL.

works, but also specific enough to clarify what is the goal of FSL and how we can solve it. Such a definition is helpful for setting future research target in the FSL area.
- We point out the core issue of FSL based on error decomposition [19] in machine learning. We figure out that it is the unreliable empirical risk minimizer that makes FSL hard to learn. This can be relieved by satisfying or reducing the sample complexity of learning. More importantly, this provides insights to improve FSL methods in a more organized and systematic way.
- We perform extensive literature review from the birth of FSL to the most recent published ones, and categorize them in a unified taxonomy in terms of data, model and algorithm. The pros and cons of different categories are thoroughly discussed. We also present a summary of the insights under each category. These can help establish a better understanding of FSL methods.
- We propose four promising future directions for FSL in terms of problem setup, techniques, applications and theories. These insights are based on the weakness of the current development of FSL, with possible improvements to make in the future. We hope they can provide some insights.

1.1 Organization of the Survey

The remainder of this survey is organized as follows. Section 2 provides an overview of the survey, including FSL's formal definition, core issue, relevant learning problems and a taxonomy of existing works in terms of data, model and algorithm. Section 3 is for methods that augment data to solve FSL problem. Section 4 is for methods that constrain the model so as to make FSL feasible. Section 5 is for methods that alter the search strategy of algorithm to deal with FSL problem. In Section 6, we propose future directions for FSL in terms of problem setup, techniques, applications and theories. Finally, the survey closes with conclusions in Section 7.

2 OVERVIEW

In this section, we first provide the notation used throughout the paper in Section 2.1. A formal definition of FSL problem is given in Section 2.2 with concrete examples. As FSL problem relates to many machine learning problems, we discuss their relatedness and difference in Section 2.3. In Section 2.4, we reveal the cores issue that makes FSL problem hard. Then according to how existing works deal with the core issue, we present a unified taxonomy in Section 2.5.

2.1 Notation

Consider a supervised learning task T, FSL deals with a data set $D = \{D^{\text{train}}, D^{\text{test}}\}$ consisting of training set $D^{\text{train}} = \{(x^{(i)}, y^{(i)})\}_{i=1}^{I}$ where I is small and test set $D^{\text{test}} = \{x^{\text{test}}\}$. Usually, people consider the N-way-K-shot classification task [34, 127] where D^{train} contains $I = KN$ examples from N classes each with K examples. Let $p(x, y)$ be the ground truth joint probability distribution of input x and output y, and $\hat{h}$ be the optimal hypothesis from x to y. FSL learns to discover $\hat{h}$ by fitting D^{train} and testing on D^{test}. To approximate $\hat{h}$, model determines a hypothesis space $\mathcal{H}$ of hypotheses $h(\cdot; \theta)$ parameterized by θ [2]. Algorithm is optimization strategy to search through $\mathcal{H}$ in order to find the θ that parameterizes the optimal $h \in \mathcal{H}$ for D^{train}. The performance is measured by a loss function $\ell(\hat{y}, y)$ defined over the prediction $\hat{y}$ (e.g., $\hat{y} = h(x; \theta)$) and the real output y.

[2]Parametric h is used, as the non-parametric ones count on large-scale data to fit the shape, therefore they are not suitable for FSL.

2.2 Problem Definition

As FSL is naturally a sub-area in machine learning, before giving the definition of FSL, let us recall how machine learning is defined literately. We adopt Mitchell's definition [80] here, which is shown in Definition 2.1.

*Definition 2.1 (**Machine learning** [80]).* A computer program is said to learn from experience E with respect to some classes of task T and performance measure P if its performance can improve with E on T measured by P.

As we can see, a machine learning problem is specified by E, T and P. For example, consider image classification task (T), machine learning programs can improve its classification accuracy (P) through E obtained by training with large-scale labeled images, e.g., ImageNet data set [64]. Another example is the recent computer program, AlphaGo [106], which has defeated the human champion in playing the ancient game of go (T). It improves its winning rate (P) against opponents by E of training using a database of around 30 million recorded moves of human experts as well as playing against itself repeatedly.

The above-mentioned typical applications of machine learning require a lot of supervised information for the given tasks. However, as mentioned in the introduction, this may be difficult or even not possible. FSL is a special case of machine learning, which exactly targets at getting good learning performance with limited supervised information provided by data set D. The supervised information refers to training data set D^{train} comprises examples of the inputs $x^{(i)}$'s along with their corresponding output $y^{(i)}$'s [17]. Formally, we define FSL in Definition 2.2.

Definition 2.2. ***Few-Shot Learning*** (FSL) is a type of machine learning problems (specified by E, T and P) where E contains a little supervised information for the target T.

To understand this definition better, let us show three typical scenarios of FSL (Table 1):

- *Act as a test bed for human-like learning:* To move towards human intelligence, the ability of computer programs to solve FSL problem is vital. A popular task (T) is to generate samples of a new character given only a few examples [66]. Inspired by how humans learn, the computer programs learn with the E consisting of both the given examples as supervised information and pre-trained concepts such as parts and relations as prior knowledge. The generated characters are evaluated through the pass rate of visual Turing test (P), which discriminates whether the images are generated by humans or machines. With this prior knowledge, computer programs can also learn to classify, parse and generate new handwritten characters of a few examples like humans.
- *Reduce data gathering effort and computation cost:* FSL can also help to relieve the burden of collecting large-scale supervised information. Consider classifying classes of a few examples through FSL [32]. The image classification accuracy (P) improves with the E obtained by a few labeled images for each class of the target T, and the prior knowledge extracted from other classes, such as raw images to co-training. Methods succeed in this task usually have higher generality, therefore they can be easily applied for tasks of many samples.
- *Learn for rare cases:* Finally, it is through FSL that one can learn suitable models for rare cases of limited supervised data. For example, consider a common drug discovery task (T) which is to predict whether the new molecule brings in toxic effects [3]. The percent of molecules correctly assigned to toxic or not toxic (P) improves with E obtained by both the new molecule's limited assay, and many similar molecules' assays as prior knowledge.

As only a little supervised information directly related to T is contained in E, it is natural that common supervised machine learning approaches fail on FSL problems. Therefore, FSL methods

Table 1. Illustrations of three FSL examples based on Definition 2.2.

T	E		P
	supervised information	prior knowledge	
character generation [66]	a few examples of new character	pre-learned knowledge of parts and relations	pass rate of visual Turing test
image classification [61]	supervised few labeled images for each class of the target T	raw images of other classes, or pre-trained models.	classification accuracy
drug toxicity discovery [3]	new molecule's limited assay	similar molecules' assays	classification accuracy

make the learning of the target T feasible by combining the available supervised information in E with some prior knowledge, which is "any information the learner has about the unknown function before seeing the examples" [75].

2.3 Relevant Learning Problems

In this section, we discuss the relevant learning problems of FSL. The relatedness and difference with respect to FSL are specially clarified.

- *Semi-supervised learning* [147] learns the optimal hypothesis $\hat{h}$ by experience E consisting of both labeled and unlabeled samples. *Positive-unlabeled learning* [71] is a special case of semi-supervised learning, where only positive and unlabeled samples are given. Another related semi-supervised learning problem is *active learning* [103], which selects informative unlabeled data to query an oracle for output y. By definition, FSL can be supervised learning, semi-supervised learning and reinforcement learning, depending on what kind of data is available apart from the limited supervised information. It neither requires the existing of unlabeled samples nor an oracle.
- *Imbalanced learning* [49] learns from experience E with a severely skewed distribution for y. It trains and tests to choose among all possible y. In contrast, FSL trains and tests for y with a few examples, while possibly taking the other y as prior knowledge for learning.
- *Transfer learning* [87] transfers knowledge learned from the source domain and source task where sufficient training data is available, to the target domain and target task where training data is limited. *Domain adaptation* [11] is a type of transfer learning problem, where the tasks are the same but the domains are different. Another related transfer learning problem called *zero-shot learning* [68] recognizes a new class with no supervised training examples by linking them to existing classes, which usually count on external data sources such as text corpus and lexical database [135]. FSL does not need to be a transfer learning problem. However, when the given supervised information is limited to learn directly, FSL needs to transfer prior knowledge to the current task. Then this kind of FSL problem becomes a transfer learning problem.
- *Meta-learning* or learning-to-learn [51] improves P of the new task T by the provided data set and the meta knowledge extracted across tasks by a meta-learner. Specifically, meta learner gradually learns generic information (meta knowledge) across tasks, and learner rapidly generalizes meta-learner for a new task T using task-specific information. Many FSL methods are meta-learning methods, using the meta-learner as prior knowledge. For later reference, a formal definition of meta-learning is in Appendix A.

2.4 Core Issue

Usually, we cannot get perfect predictions for a machine learning problem, i.e., there are some prediction errors. In this section, we illustrate the core issue under FSL based on error decomposition in machine learning [19, 20].

Recall that machine learning is about improving with E on T measured by P. In terms of our notation, this can be written as

$$\min_{\theta} \sum_{(x^{(i)}, y^{(i)}) \in D^{\text{train}}} l(h(x^{(i)}; \theta), y^{(i)}). \tag{1}$$

Here, θ parameterizes the hypothesis $h \in \mathcal{H}$ chosen by the model. And learning is about the algorithm searching for the θ of the best hypothesis h in $\mathcal{H}$ that fits the data D^{train}.

2.4.1 Empirical Risk Minimization. In essence, we want to minimize the *expected risk* R, which is the losses measured with respect to $p(x, y)$. For some hypothesis h, R is defined as

$$R(h) = \int \ell(h(x), y)\, dp(x, y) = \mathbb{E}[\ell(h(x), y)].$$

However, $p(x, y)$ is unknown. Hence *empirical risk* $R_I(h)$ is used to estimate the expected risk $R(h)$. It is defined as the average of the sample losses over the training data set (D^{train} of I samples):

$$R_I(h) = \frac{1}{n} \sum_{i=1}^{I} \ell(h(x^{(i)}), y^{(i)}),$$

and learning is done by *empirical risk minimization* [124] (perhaps also with some regularizers). For illustrative purpose, let

- $\hat{h} = \arg\min_f R(h)$, where R attains its minima;
- $h^* = \arg\min_{h \in \mathcal{H}} R(h)$, where R is minimized with respect to $h \in \mathcal{H}$;
- $h_I = \arg\min_{h \in \mathcal{H}} R_I(h)$, where R_I is minimized with respect to $h \in \mathcal{H}$.

Assume $\hat{h}$, h^* and h_I are unique for simplicity. The *total error* of learning taken with respect to the random choice of training set can be decomposed into

$$\mathbb{E}[R(h_I) - R(\hat{h})] = \underbrace{\mathbb{E}[R(h^*) - R(\hat{h})]}_{\mathcal{E}_{\text{app}}(\mathcal{H})} + \underbrace{\mathbb{E}[R(h_I) - R(h^*)]}_{\mathcal{E}_{\text{est}}(\mathcal{H}, I)}. \tag{2}$$

where the *approximation error* $\mathcal{E}_{\text{app}}(\mathcal{H})$ measures how closely the functions in $\mathcal{H}$ can approximate the optimal hypothesis $\hat{h}$, the *estimation error* $\mathcal{E}_{\text{est}}(\mathcal{H}, I)$ measures the effect of minimizing the empirical risk $R_I(h)$ instead of the expected risk $R(h)$ within $\mathcal{H}$ [19, 20].

As shown, the total error is affected by $\mathcal{H}$ (the hypothesis space) and I (the number of examples in D^{train}). In other words, learning to reduce the total error can be attempted from the perspectives of data which provides D^{train}, model which determines $\mathcal{H}$ and algorithm which searches through $\mathcal{H}$ for the θ of the best h that fits D^{train}.

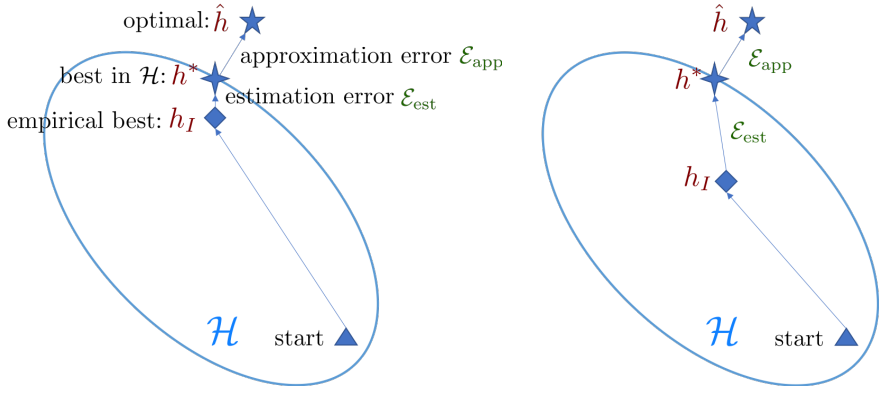
2.4.2 Unreliable Empirical Risk Minimizer. Note that, for $\mathcal{E}_{\text{est}}(\mathcal{H}, I)$ in (2), we have

$$\mathcal{E}_{\text{est}}(\mathcal{H}, \infty) = \lim_{I \to \infty} \mathbb{E}[R(h_I) - R(h^*)] = 0, \tag{3}$$

which means more examples can help reduce $\mathcal{E}_{\text{est}}(\mathcal{H}, I)$. Thus, in common setting of supervised learning task, the training data set is armed with sufficient supervised information, i.e., I is large. Empirical risk minimizer h_I can provide a good, (i.e., according to (3)) approximation $R(h_I)$ to the best possible $R(h^*)$ for h's in $\mathcal{H}$.

However, the number of available examples I is small in FSL. This makes the empirical risk $R_I(h)$ far from being a good approximation for expected risk $R(h)$ and the resultant empirical risk minimizer h_I not good. Indeed, this is the core issue of FSL, i.e., *the empirical risk minimizer* h_I

is no longer reliable. Therefore, FSL is much harder than common machine learning settings. A comparison between common versus few-shot setting is shown in Figure 1.



(a) Large I. (b) Small I.

Fig. 1. Comparison between common setting and few-shot setting in machine learning.

Historically, classical machine learning methods learn with regularizations [41] to generalize the learned methods for new data set. Regularization techniques have been rooted in machine learning, which helps to reduce $\mathcal{E}_{\text{est}}$ and get better learning performance [80]. Classical examples include Tikhonov regularizer [52], and lasso regularizer [116]. Admittedly, these regularizations can restrict the form of models. However, these simple regularization techniques cannot address the problem of FSL. They do not bring in any extra supervised information, therefore they cannot address the unreliability of the empirical risk minimizer caused by small D^{train}. Thus, learning with regularization is not enough to offer good prediction performance for FSL problem.

2.4.3 Sample Complexity. Empirical risk minimization is closely related to sample complexity. Specifically, sample complexity refers to the number of training samples needed to guarantee the loss of minimizing empirical risk $R_I(h)$ instead of expected risk $R(h^*)$ is at most ϵ with probability $1-\delta$ [80]. Mathematically, for $0 < \epsilon, \delta < 0.5$, sample complexity is an integer S such that for $I \geq S$, we have

$$p(R(h_I) - R(h^*) \geq \epsilon) < \delta \Rightarrow p(\mathcal{E}_{\text{est}}(\mathcal{H}, I) \geq \epsilon) < \delta. \tag{4}$$

When S is finite, $\mathcal{H}$ is learnable. For infinite space $\mathcal{H}$, its complexity can be measured by VapnikâĂŞChervonenkis (VC) dimension [125]. VC dimension $\text{VC}(\mathcal{H})$ is defined as the size of the largest set of inputs that can be shattered (split in all possible ways) by $\mathcal{H}$. S is tightly bounded as

$$\Theta\left(\frac{\text{VC}(\mathcal{H})}{\epsilon^2} + \frac{\log(1/\delta)}{\epsilon^2}\right), \tag{5}$$

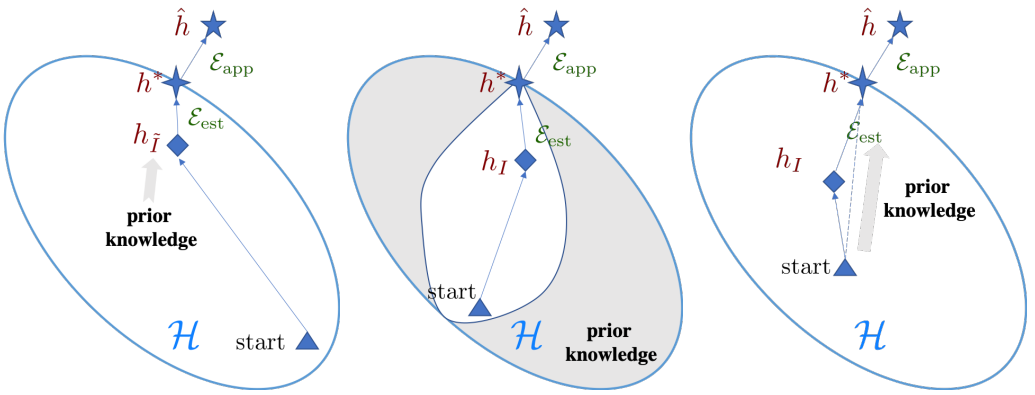
where the lower and upper bound are proved in [125] and [114] respectively.

As shown in (4) and (5), for fixed δ and ϵ, $\mathcal{H}$ needs to be less complicated to make the provided I samples enough for S. FSL methods usually use prior knowledge to compensate for the lacking in samples. One typical kind of FSL methods is Bayesian learning [32, 66]. It combines the provided training data set D^{train} with prior probability which is the probability available before D^{train} is given [17]. In this way, the required S to determine the final probability of h is reduced provably [39, 80]. This inspires us to suspect that FSL methods can satisfy or reduce the S by using prior knowledge. Consequently, the core issue of the unreliable empirical risk minimizer can be solved.

2.5 Taxonomy

In previous sections, we discover that the core issue of FSL is the unreliable empirical risk minimizer h_I. We also show that a reliable empirical risk minimizer can be obtained by satisfying or reducing the required sample complexity S. Existing FSL works use prior knowledge to satisfy or reduce S. Based on how prior knowledge is used, we categorize these works into kinds:

- *Data*: methods that use prior knowledge to augment D^{train} of I samples to $\tilde{I}$ samples. In this way, S can be met [12, 107]. And one can obtain a more accurate empirical risk minimizer $h_{\tilde{I}}$ as shown in Figure 2(a). With more samples, common models and algorithms can be used directly.
- *Model*: methods that design $\mathcal{H}$ based on prior knowledge in experience E to constrain the complexity of $\mathcal{H}$. Learning with this constrained $\mathcal{H}$ leads to a smaller S as proved in [39, 75, 85]. An illustration is shown in Figure 2(b). The gray area is not considered for later optimization as they are known unlikely to contain optimal h^* according to prior knowledge. For this smaller $\mathcal{H}$, D^{train} is enough to learn a more reliable h_I.
- *Algorithm*: methods that take advantage of prior knowledge to search for the θ which parameterizes the best hypothesis h^* in $\mathcal{H}$. The prior knowledge alters the search strategy by providing a good initial point to begin the search, or directly providing the search steps. For methods of this kind, refining some existing parameters optimizes for h_I and targets at h^*, while utilizing meta-learner learned from a set of tasks directly targets at h^*, therefore we show two paths from "startâĂİ to h^* in Figure 2(c). They both provably reduce S [4, 76].



(a) Data. (b) Model. (c) Algorithm.

Fig. 2. How FSL methods solve few-shot problem from the perspectives from data (left), model (middle) and algorithm (right). In Figure 2(a), D^{train} of I samples is augmented to be of $\tilde{I}$ samples by prior knowledge, therefore a more reliable $h_{\tilde{I}}$ is obtained by empirical risk minimization. In Figure 2(b), prior knowledge is used to constrain $\mathcal{H}$, eliminating the region in gray for later optimization. In Figure 2(c), the search strategy for θ of the best hypothesis h^* in $\mathcal{H}$ is altered by prior knowledge. Specially, the dotted line means the optimization for empirical risk can be skipped, which is a strategy used by meta-learning methods.

Accordingly, existing works can be categorized into a unified taxonomy as shown in Figure 3. We will detail each category in next sections.

3 DATA

Methods in this section solve FSL problem by augmenting data D^{train} using prior knowledge, so as to enrich the supervised information in E. With more samples, the data is sufficient to meet the

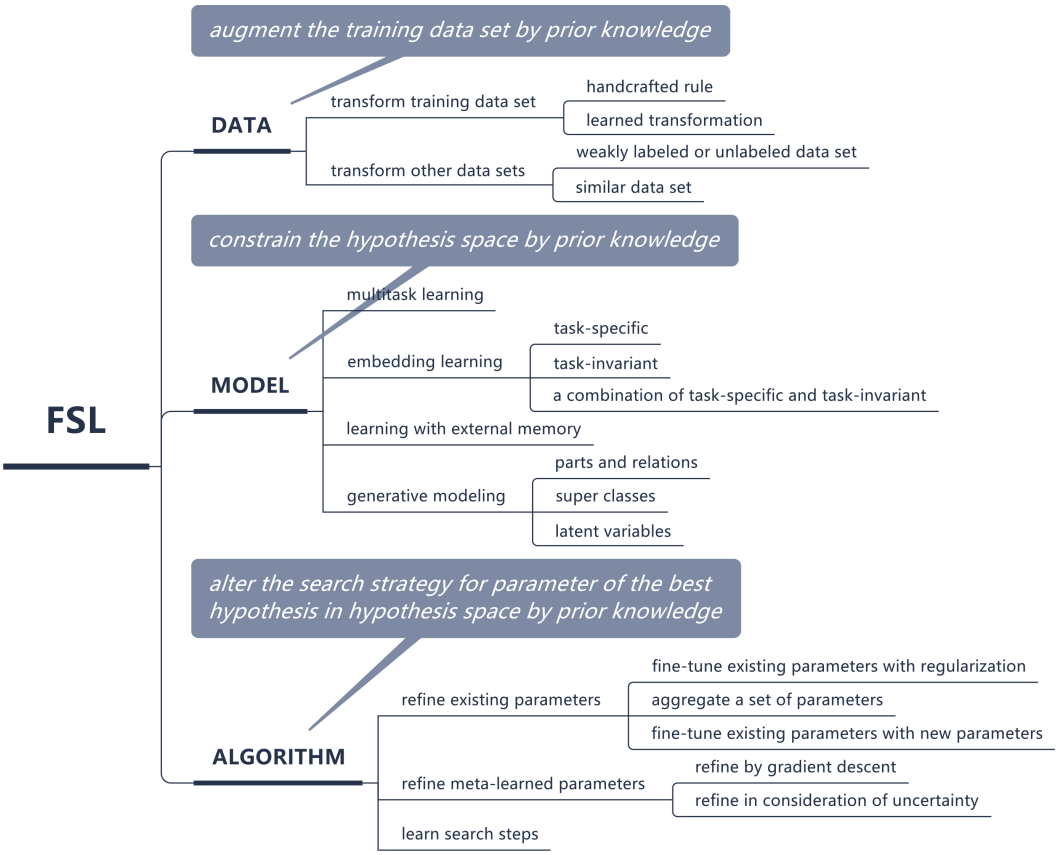


Fig. 3. A taxonomy of FSL based on the focus of each method.

sample complexity needed by subsequent machine learning models and algorithms, and to obtain a more reliable h_I.

Here, we show how data is augmented in FSL using prior knowledge. Depending on the type of prior knowledge, we classify these methods into four kinds as shown in Table 2. Accordingly, an illustration of how transformation works is shown in Figure 4. As the augmentation to each of N classes in D^{train} is done independently, we illustrate using example $(x^{(i)}, y^{(i)})$ of class n in D^{train}.

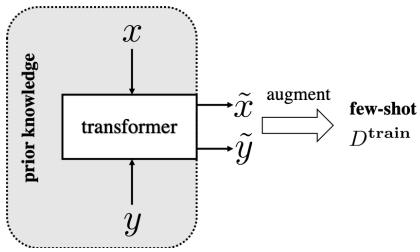


Fig. 4. Illustration of augmenting the data set D^{train} by the output $(\tilde{x}, \tilde{y})$ of transformer which transforms some input (x, y).

Table 2. Characteristics for FSL methods focusing on data perspective.

prior knowledge	transformation		
	input	transformer	output
handcrafted rule	original (x, y)	handcrafted rule on x	(transformed x, y)
learned transformation	original (x, y)	learned transformation on x	(transformed x, y)
weakly labeled or unlabeled data set	weakly labeled or unlabeled x	predictor h trained by D^{train}	(x, output predicted by h)
similar data set	sample from similar data set	aggregate new x and y by weighted average of samples of similar data set	aggregated sample

3.1 Transform D^{train}

This strategy augments D^{train} by transforming each $(x^{(i)}, y^{(i)})$ into several samples with some variation. The transformation procedure, which can be learned from similar data or designed by human expertise, is included in experience E as prior knowledge. It is only applied to images so far, as the synthesized images can be easily evaluated by humans.

3.1.1 Handcrafted Rule. On image recognition tasks, many works augment D^{train} by transforming original examples in D^{train} using handcrafted rules as pre-processing routine, e.g., translating [13, 66, 100, 105], flipping [89, 105], shearing [105], scaling [66, 143], reflecting [31, 63], cropping [89, 143] and rotating [100, 127] the given examples.

3.1.2 Learned Transformation. In contrast, this strategy augments D^{train} by duplicating original examples into several samples which are then modified by learned transformation. The learned transformation itself is the prior knowledge in E, while neither its training samples nor learning procedure is needed for the current FSL task. The earliest paper on FSL [78] learns a set of geometric transformation from a similar class by iteratively aligning each sample with other samples. Then this learned transformation is applied on each $(x^{(i)}, y^{(i)})$ to form a large data set which can be learned normally. Similarly, Schwartz et al. [2018] learn a set of auto-encoders from a similar class, each representing one intra-class variability, to generate new samples by adding the variation to $x^{(i)}$. Assuming all categories share general transformable variability across samples, a single transformation function is learned in [48] to transfer variation between sample pairs learned from other classes to $(x^{(i)}, y^{(i)})$ by analogy. Instead of enumerating the variability within pairs, Kwitt et al. [2016] transform each $x^{(i)}$ to several new samples using a set of independent attribute strength regressors learned from a large set of scene images, and assign these new samples the label of the original $x^{(i)}$. Based on [65], Liu et al. [2018] further propose to learn a continuous attribute subspace which can be used to bring in any attribute variation to x.

3.1.3 Discussion. Transforming D^{train} by handcrafted rules is popularly used in deep models to reduce the risk of overfitting [41]. However, deep models are usually learned from large-scale data sets, where the samples are enough to estimate its rough distributions (either conditional distributions for discriminative models or generating distributions for generative models) [80]. In this case, augmenting D^{train} by more samples can help to draw a clearer shape of the distribution. In contrast, FSL contains only a little supervised information, thus its $p(x, y)$ is not well exposed. These handcrafted rules such as simply scaling and rotation transform all images without considering the task or desired data property available in D^{train}. They do not bring in extra supervised information. Therefore, they are only used as a pre-processing step for image data. As for transforming D^{train} by learned transformation, it is data-driven and exploit prior knowledge akin to D^{train} of task T, therefore it can augment more suitable samples. However, this prior knowledge needs to be extracted from similar tasks, which may not always be available and can be costly to collect.

3.2 Transform Other Data Sets

This strategy transforms samples from other data sets and adapts them to be like $x^{(i)}$ of the target $y^{(i)}$, so as to be augmented to the supervised information D^{train}.

3.2.1 Weakly Labeled or Unlabeled Data Set. This strategy uses a large-scale weakly labeled or unlabeled data set. This data set is known to contain samples of the same label as $y^{(i)}$, but the output is not explicitly given. Therefore, we have to first find these samples with the target labels. As these samples of a large-scale data set contain enormous variations of samples, augmenting them to D^{train} helps depict a clearer $p(x, y)$. Consider video gesture recognition, Pfister et al. [2014] use a large but weakly labeled gesture reservoir, which contains large variations of continuous gestures of different people but no clear break between gestures. A classifier learned from D^{train} is used to pick those samples with the same gestures of D^{train} from the gesture reservoir. Then the final gesture classifier is built using these selected samples. Label propagation is used to label an unlabeled data set directly in [29].

3.2.2 Similar Data Set. This strategy augments D^{train} by aggregating samples pairs from other similar but larger data sets. For example, a data set of tigers is similar to another data set of cats. The assumption is that the underlying optimal hypothesis $\hat{h}$ applies to all classes, and the similarity between x of classes can be transferred to y of classes. Therefore, new samples can be generated by aggregating sample from a similar data set, where the aggregation weight is usually some similarity measure extracted from other information sources, such as text corpus [121]. However, directly augmenting the aggregated samples to D^{train} may not be appropriate, as these samples are not from the target FSL class. Therefore, Gao et al. [2018] design a method based on generative adversarial network (GAN) [42] to generate indiscriminate synthetic $\tilde{x}$ aggregated from data set of many samples.

3.2.3 Discussion. The gathering of weakly labeled or unlabeled data set is usually cheap, as no human effort is needed for labeling. However, along with this cheapness, the quality of these kinds of data sets is usually low, e.g., coarse and lack of strict data set collecting and scrutinizing procedure, resulting in unclear synthesizing quality. Besides, it is also costly to pick useful samples from this large data set. Similar data set shares some property with D^{train}, and contains sufficient supervised information, making it a more informative data source to be exploited. However, determining the key property so as to seek similar data sets can be objective, and collecting this similar data set is laborious.

3.3 Summary

By augmenting D^{train}, methods in this section can meet the desired sample complexity S and obtain a reliable empirical risk minimizer h_I. Methods of the first kind of transform each original sample $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$ by handcrafted or learned transformation rules. They augment D^{train} based on original samples, therefore the constructed new samples will not be too far away from D^{train}. But also due to this reason, generating samples in this way is restricted. Methods of the second kinds transform samples from other data set and adapt them to mimic $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$. These data sets are in large-scale, providing tremendous samples of large variation for transformation. However, how to adapt those samples to be like $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$ can be hard.

In general, solving FSL from the perspective of augmenting D^{train} is straightforward. The data can be augmented in consideration of incorporating the target of the problem which eases learning. And this augmentation procedure is usually reasonable to human. However, as $p(x, y)$ is unknown, perfect prior knowledge is not possible. This means that the augmentation procedure is not precise.

The gap between the estimated one and the ground truth largely interferes the data quality, even leading to concept drift.

4 MODEL

Model determines a hypothesis space $\mathcal{H}$ of hypotheses $h(\cdot;\theta)$ parameterized by θ to approximate the optimal hypothesis $\hat{h}$ from input x to output y.

If common machine learning models are used to deal with the few-shot D^{train}, they have to choose a small hypothesis space $\mathcal{H}$. As shown in (5), a small $\mathcal{H}$ has small sample complexity, thus requiring fewer samples to be trained [80]. When the learning problem is simple, e.g., the feature dimension is low, a small $\mathcal{H}$ can already get desired good learning performance. However, learning problems in real-world are usually very complex, they cannot be well represented by hypothesis h in a small $\mathcal{H}$ due to significant $\mathcal{E}_{\text{app}}(\mathcal{H})$ [41]. Therefore, large $\mathcal{H}$ is preferred for FSL, which makes common machine learning models not feasible. As we will see in the sequel, methods in this section learn a large $\mathcal{H}$ by complementing the lack of samples through prior knowledge in E. Specifically, the prior knowledge is used to affect the design choices of $\mathcal{H}$, such as constraining $\mathcal{H}$. In this way, the sample complexity is reduced, the empirical risk minimization is more reliable, and the risk of overfitting is reduced. In terms of what prior knowledge is used, methods falling in this kind can be further classified into four kinds, as summarized in Table 3.

Table 3. Characteristics for FSL methods focusing on model perspective.

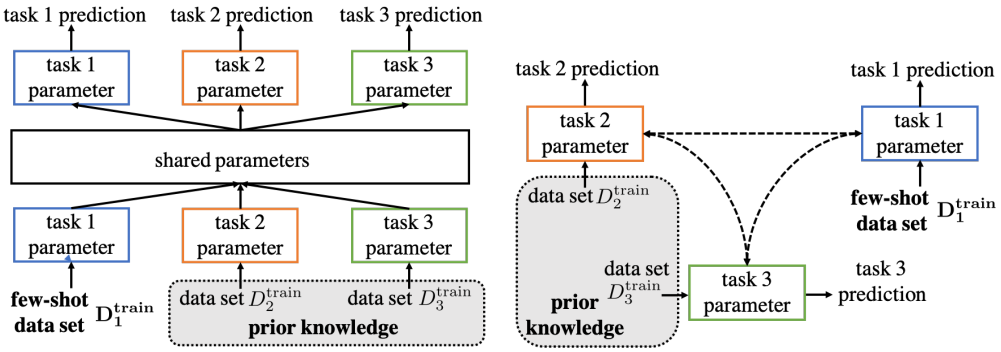
strategy	prior knowledge	how to constrain $\mathcal{H}$
multitask learning	other T's with their data sets D's	share parameter
embedding learning	embedding learned from/together with other T's	project samples to a smaller embedding space where similar and dissimilar samples can be easily discriminated
learning with external memory	embedding learned from other T's to interact with memory	refine samples by D^{train} stored in memory
generative modeling	prior model learned from other T's	restrict the form of distribution

4.1 Multitask Learning

Multitask learning [23] [3] methods learn multiple learning tasks spontaneously, exploiting the generic information shared across tasks and specific information of each task. It is popularly used for applications where multiple related tasks of limited training examples co-exist, therefore they can naturally apply for FSL problems. Note that when multitask learning deals with tasks from different domains, it is also called domain adaptation [11].

Formally, given a set of R related tasks T_t's including both tasks of few samples and many samples, each task T_t operates on data sets D_{T_t}'s where $D_{T_t} = \{D^{\text{train}}_{T_t}, D^{\text{test}}_{T_t}\}$ consists of training set $D^{\text{train}}_{T_t}$, and test set $D^{\text{test}}_{T_t}$. Among these tasks, we call the few-shot tasks as *target tasks*, while the rest as *source tasks*. Multitask learning learns from $D^{\text{train}}_{T_t}$'s to obtain θ_{T_t} for each T_t. As these tasks are related, they are assumed to have similar or overlapping hypothesis space $\mathcal{H}_{T_t}$'s. Explicitly, this is done by sharing parameters among these tasks. And these shared parameters can be viewed as a way to constrain each $\mathcal{H}_{T_t}$ by the other jointly learned tasks. In terms of whether parameter sharing is explicitly enforced, we separate methods in this strategy into hard and soft parameter sharing. Illustrations about hard and soft parameter sharing are in Figure 5.

[3]Here we present some instantiations of using multitask learning for FSL problems. For a comprehensive introduction of multitask learning, please refer to [144] and [97].



(a) Hard parameter sharing. (b) Soft parameter sharing.

Fig. 5. Illustrations of hard and soft parameter sharing strategies used in multitask learning for FSL problem.

4.1.1 Hard Parameter Sharing. This strategy explicitly shares parameter among tasks to promote overlapping $\mathcal{H}_{T_t}$'s, and can additionally learn a task-specific parameter for each task to account for task specialties. In [143], this is done by sharing the first several layers of two networks to learn the generic information, while learning a different last layer to deal with different output for each task. Benaim and Wolf [2018] operate in the opposite way for domain adaptation. They learn separate embedding for source and target tasks in different domains to map them into a task-invariant space, then learn a shared classifier to classify samples from all tasks. Finally, method in [81] first pre-trains a variational auto-encoder from the source tasks in the source domain, clones it for target task. Then it shares some layers to capture generic information, and lets both tasks to have some task-specific layers. The target task can only update their task-specific layers, while the source task can update both shared and their specific layers. It avoids directly updating the shared layers using D^{train} so as to reduce the risk of overfitting.

4.1.2 Soft Parameter Sharing. This strategy does not explicitly share parameters across tasks. Instead, each task T_t has its own hypothesis space $\mathcal{H}_{T_t}$ and parameter θ_{T_t}. It only encourages parameters of different tasks to be similar, resulting in similar $\mathcal{H}_{T_t}$'s. This can be done by regularizing θ_{T_t}'s. Yan et al. [2015] penalize the pairwise difference of θ_{T_t}'s among all combinations, forcing all θ_{T_t} to be learned similarly. Apart from regularizing θ_{T_t}'s directly, another way is to force soft parameter sharing by adjusting θ_{T_t}'s through loss. After optimization, the learned θ_{T_t}'s also utilize information of each other. Luo et al. [2017] initialize the CNN for the target tasks in the target domain by a pre-trained CNN learning from source tasks in source domain. During training, they use an adversarial loss calculated from representations in multiple layers of CNN to force the two CNNs projects samples to a task-invariant space.

4.1.3 Discussion. Multitask learning methods constrain $\mathcal{H}_{T_t}$ learned for each task T_t by a set of tasks jointly learned. By sharing parameters explicitly or implicitly, the jointly learned tasks together eliminate those infeasible regions implicitly. Sharing by hard parameter can be enforced easily. A shared hypothesis space is used to capture the commonality while each task builds its specific model hypothesis space on top it. In contrast, soft parameter sharing only encourages a similar hypothesis, which is a more flexible way to constrain $\mathcal{H}_{T_t}$. But how to enforce the similarity constraint needs careful design.

4.2 Embedding Learning

Embedding learning [55, 109] methods embeds $x^{(i)} \in \mathcal{X} \subseteq \mathbb{R}^d$ to a smaller embedding space $z^{(i)} \in \mathcal{Z} \subseteq \mathbb{R}^m$, where the similar and dissimilar pairs can be easily identified. Therefore, $\mathcal{H}$ is constrained. The embedding function is mainly learned by prior knowledge, and can additionally use D^{train} to bring in task-specific information. Note that embedding learning methods are mainly designed for classification tasks.

Embedding learning methods have the following key components: function $f(\cdot)$ which embeds samples $x^{\text{test}} \in D^{\text{test}}$ to $\mathcal{Z}$, function $g(\cdot)$ which embeds examples $x^{(i)}$ to $\mathcal{Z}$, and similarity measure $s(\cdot,\cdot)$ which is used to calculate the similarity between $f(x^{\text{test}})$ and $g(x^{(i)})$ for each $x^{(i)}$ in $\mathcal{Z}$. Then x^{test} is assigned to the class of the most similar $x^{(i)}$. Although g can be the same as f, sometimes different f and g are used for $x^{\text{test}} \in D^{\text{test}}$ and $x^{(i)}$. This is because $x^{\text{test}} \in D^{\text{test}}$ can be embedded explicitly depending on information from D^{train} so as to adjust comparing interest [15, 127]. Hence we discriminate these two embedding functions. Usually, a set of data sets D_c's with $D_c^{\text{train}} = \{(x_c^{(i)}, y_c^{(i)})\}$ and $D_c^{\text{test}} = \{x_c^{\text{test}}\}$ are used to learn these components. Note that D_c can be data set of many samples or few samples. An illustration of embedding learning strategy is shown in Figure 6. We also present the details of existing methods in embedding learning in terms of f, g and s in Table 4.

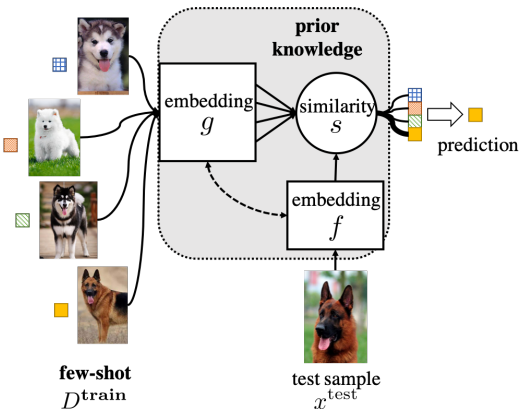


Fig. 6. Illustration of embedding learning strategy for FSL problem. The figure is adapted from [127].

Next, according to what information is embedded in the embedding, we will classify these methods into task-invariant (in other words, general), task-specific and a combination of the two.

4.2.1 Task-specific. Task-specific embedding methods learn an embedding function tailored for D. Triantafillou et al. [2017] learns an embedding to maintain the ranking list for each $(x^{(i)}, y^{(i)})$ in D^{train}, where those of the same class rank higher and otherwise lower. Given the few-shot D^{train}, the sample complexity is largely reduced by enumerating all pairwise comparisons between examples in D^{train} as sample pairs. In this way, each original example can be included in multiple sample pairs, which largely reduces the required sample complexity.

4.2.2 Task-invariant. Task-invariant embedding methods learn embedding function from a large set of data sets D_c's which does not include the D. The assumption is that if the embeddings can successfully separate many data sets on $\mathcal{Z}$, they can be general enough to work well for D without retraining. Fink [2005] proposes the first embedding method for FSL. It learns from auxiliary D_c's a kernel space as $\mathcal{Z}$, embeds both D^{test} and D^{train} to $\mathcal{Z}$, where $x^{\text{test}} \in D^{\text{test}}$ is assigned to the class of nearest neighbor in D^{train}. A recent deep model convolutional siamese net [61] learns twin CNNs

Table 4. Characteristics of Embedding Learning methods. The column "g to f" means f is directly influenced by how $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$ is embedded by g.

method	f for x^{test}	g for D^{train}	g to f	similarity measure s	type
mAP-DLM/SSVM[118]	CNN	the same as f	no	cosine similarity/mAP	specific
class relevance pseudo-metric [33]	kernel	the same as f	no	squared ℓ_2 distance	invariant
convolutional siamese net [61]	CNN	the same as f	no	weighted ℓ_1 distance	invariant
Micro-Set[115]	logistic projection	the same as f	no	ℓ_2 distance	combined
Learnet [15]	adaptive CNN	the same as f	yes	weighted ℓ_1 distance	combined
DyConvNet [145]	adaptive CNN	the same as f	no	-	combined
R2-D2 [14]	adaptive CNN	the same as f	yes	-	combined
Matching Nets [127]	CNN, then LSTM with attention	CNN, biLSTM	yes	cosine similarity	combined
resLSTM [3]	GCN, then LSTM with attention	GCN, then LSTM with attention	yes	cosine similarity	combined
Active MN [8]	CNN	biLSTM	yes	cosine similarity	combined
ProtoNet [108]	CNN	the same as f	no	squared ℓ_2 distance	combined
semi-supervised ProtoNet[95]	CNN	the same as f	no	squared ℓ_2 distance	combined
PMN [130]	CNN, then LSTM with attention	CNN, then biLSTM	yes	cosine similarity	combined
TADAM [86]	CNN	the same as f	yes	squared ℓ_2 distance	combined
ARC [105]	RNN with attention, then biLSTM	the same as f	yes	-	combined
Relation Net [112]	CNN	the same as f	no	-	combined
GNN [101]	CNN, then GNN	the same as f	yes	learned distance	combined
TPN [73]	CNN	the same as f	yes	Gaussian similarity	combined
SNAIL [79]	CNN with attention	the same as f	no	-	combined

to embed sample pairs from a large set of data sets D_c's to a common embedding space $\mathcal{Z}$. It then constructs sample pairs using original samples of D^{train}, and reformulates the classification task as a verification/ matching task which verifies whether the resultant embeddings of the sample pairs belong to the same class or not.

4.2.3 A Combination of Task-invariant and Task-specific. Task-specific embedding methods fully consider the task specialty, while task-invariant embedding methods can rapidly generalize for a new task without re-training. A trend is to combine the best of the above-mentioned methods: learn to adapt the generic task-invariant embedding space learned from prior knowledge by task-specific information contained in D^{train}. Tang et al. [2010] firstly propose to optimize over distribution of FSL tasks, under the name micro-sets. They learn $\mathcal{Z}$ by logistic projection from these FSL tasks. For a new few-shot task T, all samples in D^{train} and D^{test} are mapped to $\mathcal{Z}$. Then x^{test} is classified by nearest neighbor classifier on $\mathcal{Z}$.

Recent works mainly use meta-learning methods[4] to merge the task-invariant and task-specific knowledge. We group them by the core ideas and highlight the representative works.

(1) *Learnet* [15] improves upon convolutional siamese net [61] by incorporating the specialty of D^{train} of each task T to $\mathcal{Z}$. It learns a meta-learner from the meta-training data sets D_{T_s}'s

[4]See Appendix A for a formal definition and a brief introduction of meta-learning. For these meta-learned embedding learning methods, D_c's are the meta-training data sets D_{T_s}'s of meta-training tasks T_s's, and a new task is one of the meta-testing tasks T_t's. As in both meta-training and meta-testing stage, the learner will train by D^{train} and test on D^{test} using the provided data set D of task T, we use D^{train} and D^{test} without marking T_s or T_t for illustration simplicity.

to map $x^{(i)}$ to the parameter of each layer in in convolutional siamese net. To reduce the parameter number of learner, DyConvNet [145] uses a fixed set of filters and only learns to combine them for learner. The recent work [14] replaces the classification layer of Learnet by a ridge regression model whose parameter can be found by cheap closed-form solution.

(2) *Matching Nets* [127] assign $x^{\text{test}} \in D^{\text{test}}$ to the most similar $x^{(i)}$ in $\mathcal{Z}$, where x^{test} and $x^{(i)}$ are embedded differently by f and g. Specially, f is conditioned on D^{train}, and g aggregates information of all examples in D^{train} by a bi-directional LSTM (biLSTM) [45]. However, the bi-directional LSTM implicitly enforces an order among examples in D^{train}. Due to the vanishing gradient problem, nearby examples have a larger influence on each other. To remove the unnatural order, Altae-Tran et al. [2017] replace the biLSTM used in g by LSTM with attention, and further iteratively refine both g and f to encode contextual information. An active learning variant in [8] adds a sample selection stage to matching nets [127], which can label the most beneficial unlabeled sample and use it to augment D^{train}.

(3) *ProtoNet* [108] performs only one comparison between $x^{\text{test}} \in D^{\text{test}}$ and prototype of each class in D^{train}. This class n's prototype is defined as the mean of embeddings of that class, i.e., $c_n = \frac{1}{K}\sum_{k=1}^{K} g(x^{(i)})$ where $x^{(i)}$ is one the K examples of the nth class in D^{train}. The ProtoNet embeds both x^{test} and $x^{(i)}$ using the same CNN, and ignores specialty of different D^{train}'s, Noticed that, a combination of the best of matching nets and ProtoNet is proposed in [130] to account for task-specific information. Further, Oreshkin et al. [2018] average c_n's as the task embedding, which is then mapped to some parameters of CNN used in ProtoNet. A semi-supervised variant of ProtoNet is proposed in [95], which learns to soft-assign related unlabeled samples to augment D^{train} during learning.

(4) *Relative representations* further embed the embedding of x^{test} and each c_n calculated from D^{train} in $\mathcal{Z}$ jointly, which is then directly mapped to similarity score like classification. This idea is independently developed in ARC [105] and relation net [112]. ARC uses a RNN with attention to recurrently compare different regions of x^{test} and each class prototype c_n and produces the relative representation, additionally uses a biLSTM to embed the information of other comparisons as the final embedding. Relation net first uses a CNN to embed x^{test} and $x^{(i)}$ to $\mathcal{Z}$, then concatenates them as the relative representation, and outputs the similarity score by another CNN.

(5) *Relation graph* is a graph maintaining all pairwise relationships among samples. Specifically, the graph is constructed using samples from both D^{train} and D^{test} as nodes, while its edges between nodes are determined by some learned s. Then each $x^{\text{test}} \in D^{\text{test}}$ is predicted using neighborhood information. GCN is used in [101] to learn the relation graph between $(x^{(i)}, y^{(i)})$'s from D^{train} and x^{test} from D^{test}. The resultant embedding for node x^{test} is used to predict y^{test}. In contrast, Liu et al. [2019] meta-learn an embedding which maps each $x^{(i)}$ and x^{test} to $\mathcal{Z}$, build a relation graph there, and label y^{test} by closed-form label propagation rules.

(6) *SNAIL* [79] designs special embedding networks consisting of interleaved temporal convolution layers and attention layers. The temporal convolution is used to aggregate information from past time steps, and the attention selectively attends to specific time step relevant to the current input. The network' parameter is meta-learned across tasks. Within each task, the network takes $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$ sequentially, and predicts x^{test} immediately.

4.2.4 Discussion. Task-specific embedding fully considers the domain knowledge of D. However, the given few-shot D^{train} can be biased, only learning from them may be inappropriate. Modeling ranking list among D^{train} has a high risk of overfitting to D^{train}. Besides, $\mathcal{H}$ learned this way cannot generalize from new tasks or be adapted easily. Using pre-trained task-invariant embeddings has a

low computation cost. However, the learned embedding function does not consider any task-specific knowledge. When special is the reason that D^{train} has only a few examples such as learning for rare cases, simply applying task-invariant embedding function can be not suitable. A combination of task-invariant and task-specific information is usually learned by meta-learning methods. They can provide a good $\mathcal{H}$ and quickly generalize for different tasks by learner. However, how to generalize for a new but unrelated task without bringing in negative transfer is not sure.

4.3 Learning with External Memory

External memory such as neural Turing machine (NTM) [46] and memory networks [111, 133] allows short-term memorization and rule-based manipulation [46]. Note that learning is a process of mapping useful information of training samples to the model parameters. Given new task T with training data set D^{train}, the model has to be re-trained to incorporate its information, which is costly. Instead, learning with external memory directly memorizes the needed knowledge in an external memory to be retrieved or updated, therefore it relieves the burden of learning and allows fast generalization. Formally, denote memory as $M \in \mathbb{R}^{b\times m}$, which has b memory slots $M(i) \in \mathbb{R}^m$. Given a sample $x^{(i)}$, it is first embedded by f as query $q = f(x^{(i)}) \in \mathbb{R}^m$, which then attends to each $M(i) \in \mathbb{R}^k, i = 1, \ldots, b$ through some similarity measure s, e.g, cosine similarity. Then it uses the similarity to determine which knowledge is extracted from the memory, and predicts based on it. Table 5 introduces the detailed characteristics of each method with external memory.

Table 5. Characteristics of learning with external memory strategy. f is the pre-trained embedding function, which is usually CNN or LSTM. f_1 and f_2 are different embedding functions.

method	memory		similarity measure s
	key	value	
MANN [100]	$f([(x^{(i)}, y^{(i)})])$	$f([(x^{(i)}, y^{(i)})])$	cosine similarity
abstraction memory[136]	$f(x^{(i)})$	$f(y^{(i)})$	dot product
life-long memory [56]	$f(x^{(i)})$	$y^{(i)}$ and age	cosine similarity
CMN [146]	$[f_1(x^{(i)}), f_2(x^{(i)})]$	$y^{(i)}$ and age	dot product
APL [91]	$f(x^{(i)})$	$y^{(i)}$	squared ℓ_2 distance
MetaNet [82]	$f(x^{(i)})$	fast weight	cosine similarity
CSNs [83]	$f(x^{(i)})$	fast weight	cosine similarity
MN-Net [22]	$f(x^{(i)})$	$y^{(i)}$	dot product

For FSL, D^{train} has limited samples, re-training the model is infeasible. Learning with external memory can help solve this problem by storing knowledge extracted from D^{train} into an external memory. The embedding function f learned from prior knowledge is not re-trained, therefore the initial hypothesis space $\mathcal{H}$ is not changed. When a new sample comes, relevant contents are extracted from the memory and combine into the local approximation for this sample. Then the approximation is fed to the subsequent model for prediction, which is also pre-trained. As D^{train} is stored in the memory, the task-specific information is effectively used. In sum, methods of this kind refine and re-interprete samples by D^{train} stored in memory, consequently reshaping $\mathcal{H}$. An illustration of embedding learning strategy is shown in Figure 7.

Usually, when the memory is not full, new samples can be written to vacant memory slots. However, when the memory is full, one has to decide which memory slots to be updated or replaced by some designed rules. We group existing works according to different preference revealed in these update rules as follows.

(1) *Update the least recently used memory slot.* The earliest work [100] which uses memory to solve the FSL classification problem, MANN [100] updates the least recently used memory slot for the new sample when the memory is full. As the image-label binding is shuffled across tasks,

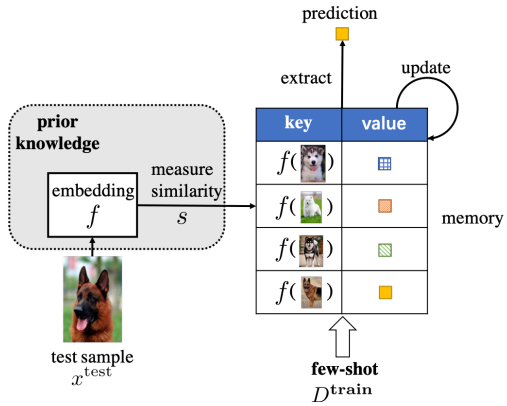


Fig. 7. Illustration of learning with external memory strategy for FSL problem.

MANN cares more about mapping samples of the same class to the same label. In turn, samples of the same class together refine their class representation kept in the memory.

(2) *Update by location-based addressing.* Some works use the location-based addressing proposed in NTM, which updates all memory slots at all time by back-propagation of gradients. The abstract memory [136] uses this update strategy. In each task, the meta-learner first extracts relevant $f([(x^{(i)}, y^{(i)})])$ from a memory containing large-scale auxiliary data, then sends them to the abstraction memory. The output of the abstraction memory is used for prediction.

(3) *Update according to the age of memory slots.* Some memory records age for each memory slot. The memory slot increases its age when it is read, and resets age to 0 when it is updated. The oldest one is more like out-dated information. Both Life-long memory [56] and CMN [146] update the oldest memory slot when the memory is full. However, some times one value the rare events in old memory slots. To deal with it, life-long memory specially prefers to update memory slots of the same classes. As each class occupies comparative number of memory slots, rare classes are protected in a way.

(4) *Update the memory only when the loss is high.* The surprised-based memory module [91] designs a memory update rule which only updates the memory when the prediction loss for some $(x^{(i)}, y^{(i)})$ is above a threshold. Therefore, the computation cost is reduced compared to a differentiable memory, and the memory contains minimal but diverse information for equivalent prediction.

(5) *Use the memory as storage without updating.* MetaNet [82] stores sample-level fast weights for $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$ in a memory, and conditions its embedding and classification by the extracted fast weights, so as to combine the generic and specific information. MetaNet repeatedly applies the fast weight to selected layers of a CNN. In contrast, Munkhdalai et al. [2018] learn fast weight to change the activation value of each neuron, which has a lower computation cost.

(6) *Aggregate the new information into the most similar one.* MN-Net [22] merges the information of new sample to its most similar memory slots. Instead of directly predicting for x^{test} as in matching nets [127], the memory is used to refine the $f(x^{(i)})$, and to parameterize a CNN as in Learnet [14]. Then each x^{test} is embedded by this conditional CNN, and is matched with $f(x^{(i)})$ by nearest neighbor search.

4.3.1 Discussion. Adapting to new tasks can be done by simply putting D^{train} to the memory, where fast generalization can be done easily. Besides, preference such as lifelong learning or reduction of memory updates can be incorporated into the designing of memory updating and accessing rules.

However, it relies on human knowledge to design the desired rule. The existing works do not have a clear winner. How to automatically design or choose update rules according to different settings are important issues.

4.4 Generative Modeling

Generative modeling methods here refer to methods involving $p(x, y)$. They use both prior knowledge and D^{train} to obtain the estimated distribution. Prior knowledge is usually learned prior models represented by parameters of some probability distribution, which are learned from a set of C data sets D_c's consisting of training set $D_c^{\text{train}} = \{(x_c^{(i)}, y_c^{(i)})\}$ and test set $D_c^{\text{test}} = \{x_c^{\text{test}}\}$. Usually, D_c is large and D is not one of D_c's. Generative modeling methods update the probability distribution over D^{train} for prediction. An illustration of this strategy is shown in Figure 8.

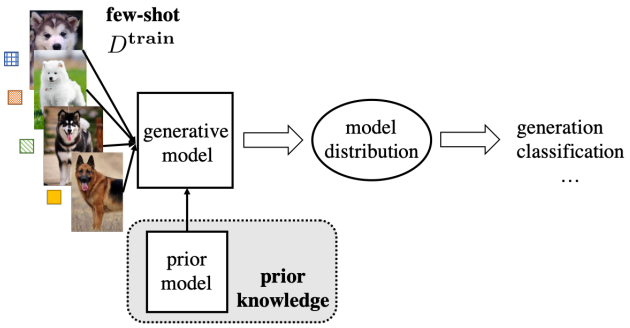


Fig. 8. Illustration of generative modeling strategy for FSL problem.

Specifically, the posterior which is the probability y for x^{test} given D^{train} is computed by Bayes' rule as $p(y|x^{\text{test}}, D^{\text{train}}) = p(x^{\text{test}}|y, D^{\text{train}})p(y)$. Expanding by parametrization, it can be written as $\int_\theta p(x^{\text{test}}|\theta, y)p(\theta|y, D^{\text{train}})d\theta$, where θ is the parameter of $h \in \mathcal{H}$. If D^{train} is large enough, we can use it to learn a well-peaked $p(\theta|y, D^{\text{train}})$, and obtain θ using maximum likelihood estimation (MLE) $\theta^{\text{ML}} = \arg\max_\theta p(\theta|y, D^{\text{train}})$ or maximum a posterior (MAP) $\theta^{\text{MAP}} = \arg\max_\theta p(\theta|y, D^{\text{train}})p(\theta)$. However, D^{train} in FSL tasks has limited samples, which is not enough to learn $p(\theta|y, D^{\text{train}})$. Consequently, it cannot learn a good θ.

Generative models for FSL assume θ is transferable across different y (e.g., classes). Hence θ can be instead learned from a large set of data sets D_c's. In detail, expand $p(\theta|y, D^{\text{train}})$ as $p(\theta|y, D^{\text{train}}) = p(D^{\text{train}}|\theta, y)p(\theta)$, which also has parameter γ. Then we can obtain $p(\theta|y, D^{\text{train}})$ by adapting the distribution or learning γ using D^{train}.

Table 6 summarizes the main bibliographic references falling in this strategy and their characteristics. By learning prior probability out of prior knowledge, the shape of $\mathcal{H}$ is restricted. According to how θ is defined and shared across y, we classify existing methods into parts and relations, super classes and latent variables.

4.4.1 Parts and Relations. This strategy learns parts and relations (a.k.a. θ) from a large set of D_c's as prior knowledge. Although the target few shot classes have few samples, their components such as parts and relations are shared with many classes. With much more samples to use, parts and relations are much easier to learn. For $(x^{(i)}, y^{(i)}) \in D^{\text{train}}$, the model needs to infer the correct combination of related parts and relations, then decides which target class this combination belongs to. Bayesian One-Shot [32] and BPL [66] fall in this category. Bayesian One-Shot leverages shapes and appearances of objects to help recognize objects, while BPL separates a character into types, tokens and further templates, parts, primitives to model characters. As the inference procedure

Table 6. Characteristics of generative modeling methods.

category	method	prior from D_c's	how to use D^{train}	task
parts and relations	Bayesian One-Shot [32]	θ	γ	object recognition
	BPL [66]	θ and γ	fine-tune partial θ	classification, generation
super classes	HB [99]	a hierarchy of θ	as one of D_c^{train}'s	object recognition
	HDP-DBM [117]	a hierarchy of θ	as one of D_c^{train}'s	object recognition
latent variables	SeqGen[96]	θ	as input	generation
	Attention PixelCNN[94]	θ	as input	image flipping, generation
	Neural Statistician [31]	θ	as input	classification, generation
	VERSA [43]	θ	as input	classification, reconstruction
	MetaGAN[142]	θ	as input	classification
	GMN [10]	θ	as input	classification, generation

is costly, a handful of parts is used in Bayesian One-Shot which largely reduce the combinatorial space of parts and relations, while only the five most possible combinations are considered in BPL.

4.4.2 Super Classes. Part and relation turn to model smaller part of samples, while super classes groups similar classes by unsupervised learning. Considering classification task, this strategy finds the best θ which parameterize $h \in \mathcal{H}$ for these super classes as prior knowledge. A new class is first assigned to a super class, then learns its h through adapting the super class's h. In [99], they learn to form a hierarchy of classes using D_c's (which includes D). In this way, similar classes together contribute to learning a precise general prior representing super classes, and in return each super class can provide guidance to its assigned classes, especially for D^{train} of a few examples. The feature learning part of [99] is further improved in [117] by using deep Boltzmann machines to learn more complicated features.

4.4.3 Latent Variables. Separating samples into parts and relations is handcrafted, and relies heavily on knowledge of human expertise. Instead, this strategy models latent variables with no implicit meaning shared across classes. Without decomposition, h learned from D_c's no longer needs to be adjusted, thus the computation cost for the new task is largely reduced. In order to handle more complicated $\mathcal{H}$, the models used in this strategy are usually deep models. According to which classic deep generative model act as basis, existing works can be grouped as follows.

(1) *Variational auto-encoder (VAE)* [59]. Rezende et al. [2016] propose to model $p(x)$ by a set of sequentially inferred latent variables using a sequential VAE. This model repeatedly attends to different regions of each $x^{(i)}$ from D_c^{train}'s and analyzes the capability of the current model to provide feedback, which can model the density estimation well.
(2) *Autoregressive model* [123]. Reed et al. [2018] propose an autoregressive model which decomposes the density estimation of an image into pixel-wise. The model sequentially generates each pixel conditioned on both already generated pixels and related information acquired from a memory storing D^{train}.
(3) *Inference networks* [141]. Edwards and Storkey [2017] learn one inference networks to infer these latent variables, and another inference networks to map D_c^{train} to θ which is the parameter of its generative distribution. Also learning inference networks by amortized variational inference, Gordon et al. [2019] learn to map D_c^{train} to the parameter of a variational distribution which approximates the predictive posterior distribution over output y.
(4) *Generative adversarial networks (GAN)* [42]. Zhang et al. [2018b] jointly learns an imperfect GAN with a discriminative model for classification task. The imperfect GAN generates samples similar to examples in D_c^{train} but slightly different. By learning to discriminate between D_c^{train}'s and the slightly different fake data, it obtains a sharper decision boundary.

(5) *Generative version of Matching Nets* (GMN) [127]. Bartunov and Vetrov [2018] extend the discriminative Matching Nets to generative setting as GMN. GMN replaces the x^{test} used in Matching Nets by a latent variable z. Then, it embeds $x^{(i)}$ by g and z by f to embedding space $\mathcal{Z}$, where $f(z)$ attends to each $g(x^{(i)})$ to obtain the weights to aggregate $g(x^{(i)})$'s. This resultant embedding and z are then fed to decoder networks to generate new sample x.

4.4.4 Discussion. Learning each object by decomposing them into smaller parts and relations leverages the human knowledge to do the decomposition. In contrast to other types of generative modeling methods discussed, parts and relations are more interpretable. However, human knowledge contains high bias, which may not suit the given data set. Besides, it can be hard or expensive to get, putting a strict restriction on application scenarios. In contrast, models learned for super classes can aggregate information from many related classes, and act as a general model for new class. However, they may not be optimal as not specific information is utilized. Methods with latent variables are more efficient and count on less human knowledge. However, as the exact meaning of the latent variable is unknown, methods of this kind are less easy to be understood.

4.5 Summary

In sum, all methods in this section design $\mathcal{H}$ based on prior knowledge in experience E to constrain the complexity of $\mathcal{H}$ and reduce its sample complexity.

Multitask learning methods constrain $\mathcal{H}$ of the few-shot task by a set of jointly learned tasks. They can communicate between different tasks and improve these tasks along the optimization process. They also implicitly augment data as some of the parameters are jointly learned by multiple tasks. However, the target D must be one of D_{T_t}'s to perform joint training. Hence for each new task, one has to learn from scratch, which can be costly and slow. It is not suitable for tasks which only have one shot or prefer fast inference.

Embedding learning methods learn to embed samples a smaller embedding space, where the similar and dissimilar pairs can be easily identified. Therefore, $\mathcal{H}$ is constrained. Most works learn from large-scale data sets for task-invariant information, and can absorb task-specialty of new tasks. Once learned, most methods can easily generalize from new tasks by a forward pass and perform the nearest neighbor among the embedded samples. However, how to mix the invariant and specific information of tasks within θ in a principled way is unclear.

Learning with external memory methods refine and re-interpret each sample by D^{train} stored in memory, consequently reshaping $\mathcal{H}$. By explicitly storing D^{train} in memory, they avoid laborious re-training to adapt for D^{train}. The task-specific information is effectively used and not forgot easily. However, learning with the external memory incurs additional space and computational cost, which increases with an enlarged memory. Therefore, current external memory has a limited size, consequently it cannot memorize much information.

Generative modeling methods learn prior probability from prior knowledge, which shapes the form of $\mathcal{H}$. They have good interpret ability, causality and compositionality [66]. By learning the joint distribution $p(x, y)$, they can deal with broader types of tasks such as generation and reconstruction. The learned generative models can generate many samples to do data augmentation. However, generative modeling methods typically have high computational cost and are difficult to derive compared with other models. For computation feasibility, they require severe simplification on the structure which leads to inaccurate approximations.

5 ALGORITHM

Algorithm is strategy to search in the hypothesis space $\mathcal{H}$ for the parameter θ of the best hypothesis h^*. For example, stochastic gradient descent (SGD) and its variants [19, 20] are one popular strategy

to search in $\mathcal{H}$. In SGD, θ is updated through a sequence of updates $t = 1, \ldots$. At the tth iteration, let $\ell^t(\theta^{t-1}) = \ell(h(x^{(t)}; \theta^{t-1}), y^{(t)})$, then θ^t is updated by

$$\theta^t = \theta^{t-1} - \alpha^t \nabla_{\theta^{t-1}} \ell^t(\theta^{t-1}), \tag{6}$$

where α^t is the step size to be tuned. When supervised information is rich, there are enough training samples to update θ to arbitrary precision, and to find an appropriate α through cross-validation. However, the provided few-shot D^{train} is not enough to reach the required sample complexity. Consequently, the obtained empirical risk minimizer is unreliable.

Methods in this section do not restrict the shape of $\mathcal{H}$, so that common models such as CNN and RNN can still be used. Instead, they take advantage of prior knowledge to alter the search for θ that parameterizes the h^* in $\mathcal{H}$ to solve the FSL problem. In terms of how the search strategy is affected by prior knowledge, we classify methods in this section into three kinds (Table 7):

(1) *Refine existing parameters* θ^0. An initial θ^0 learned from other tasks is used to initialize the search, then is refined by D^{train}.
(2) *Refine meta-learned* θ. A meta-learner is learned form a a set of tasks drawn from the same task distribution as the few-shot task to output a general θ, then each learner refines the θ provided by the meta-learner using D^{train}.
(3) *Learn search steps.* This strategy learns a meta-learner to output search steps or update rules to guide each learner directly. Instead of learning a better initialization, it alters the search steps, such as direction or step size.

Table 7. Characteristics for FSL methods focusing on algorithm perspective

strategy	prior knowledge	how to search θ of the h^* in $\mathcal{H}$
refine existing parameters θ^0	learned θ^0	refine θ^0 by D^{train}
refine meta-learned θ	meta-learner	refine θ by D^{train}
learn search steps	meta-learner	use search steps provided by the meta-learner

5.1 Refine Existing Parameters θ^0

This strategy takes θ^0 from a pre-trained model as a good initialization, and adapts it to θ by D^{train}. The assumption is that θ^0 captures general structures by learning from large-scale data, therefore it can be adapted using a few iterations to work well on D.

5.1.1 Fine-tune θ^0 with Regularization. This strategy fine-tunes the given θ^0 with some regularization. An illustration of this strategy is shown in Figure 9. Fine-tuning is popularly used in practice,

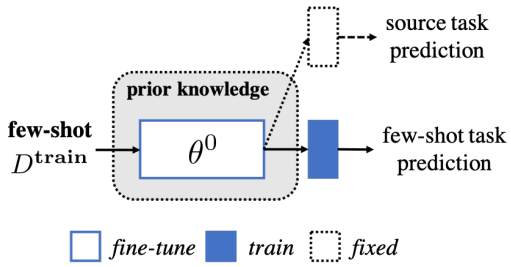


Fig. 9. Illustration of fine-tune θ^0 with regularization strategy.

which adapts the value of a (deep) model trained on large scale data such as ImageNet to smaller data sets through back-propagation [27]. The single θ^0 which contains generic knowledge is usually

parameter of deep models, which parameterizes a huge and complicated $\mathcal{H}$. Given the few-shot D^{train}, simply fine-tuning θ^0 by gradient descent leads to overfitting. How to adapt the value of θ^0 without overfitting to the limited D^{train} is the key design issue.

In this section, methods fine-tune θ^0 with regularization to prevent overfitting. They can be grouped as follows.

(1) *Early-stopping* is used in [6]. However, it requires a separate validation set from D^{train} to monitor the training, which further reduces the number of samples for training. Moreover, using a small set of validation set makes the searching strategy highly biased.
(2) *Selectively update* θ^0 refers to only update a small portion of θ^0 so as to avoid overfitting. Keshari et al. [2018] use a set of fixed filters, and only learn to control the multitude of elements within the filters by fitting D^{train}. Given a pre-trained CNN, both Qiao et al. [2018] and Qi et al. [2018] directly add the weights for each class n in D^{train} as new columns in the weight matrix of the final layer, while leaving pre-trained weights unchanged.
(3) *Cluster* θ^0 and update the resultant parameter groups using the same update information can largely constrain the search strategy. In [139], they use auxiliary data to group filers of a pre-trained CNN, and fine-tune the CNN by group-wise back-propagation using D^{train}.
(4) *Model regression networks* [132] assume there exists a task-agnostic transformation from parameter trained using a few examples to parameter trained using many samples. Wang and Hebert [2016b] then refine θ^0 learned with fixed N-way-K-shot problem. Similarly, Kozerawski and Turk [2018] learn to transform the embedding of $x^{(i)}$ to a classification decision boundary.

5.1.2 Aggregate a Set of θ^0's. Usually, we do not have a suitable θ to fine-tune. Instead, we may have many model parameters θ^0's learned from related tasks, such as the task is face recognition while only recognition models for eye, nose, ear are available. Therefore, one can pick from θ^0's the relevant ones and aggregate them into the suitable initialization to be adapted by D^{train}. An illustration of this strategy is shown in Figure 10.

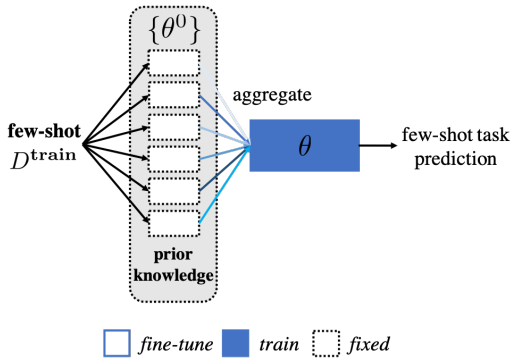


Fig. 10. Illustration of strategy that aggregates a set of θ^0's into θ.

Parameters θ^0's here are usually pre-trained from other data sets. According to what data sets are used, existing methods can be grouped as follows.

(1) *Similar data set.* Bart and Ullman [2005] classify a new class of one image by image fragments. The classifier for the new class is built by replacing the features from already learned classes by similar features taken from the novel class and reusing their classifier parameters. Only the threshold for classification is adjusted to avoid confusion with those similar classes. Similar to [89] and [90], a pre-trained CNN is adapted to deal with new class in [40]. But instead of solely using embedding of $x^{(i)}$ as classifier parameter which is highly biased, it constructs

the classifier for this new class through a linear combination of the embedding of $x^{(i)}$ and a tentative classifier built by attending to other classes' classifier parameter.

(2) *Unlabeled data set.* θ^0's learned from an unlabeled data set can also be discriminative to separate samples. Pseudo-labels are iteratively assigned and adjusted for samples in the unlabeled data set, so as to learn decision boundaries [131]. Further, these learned decision boundaries are incorporated into a pre-trained CNN [131]. Note that in a pre-trained CNN, the higher the embedding layers, the more specific they are. By learning to separate the unlabeled data set, the generality of the embedding of the last layers is improved. Specifically, given a pre-trained CNN, Wang and Hebert [2016a] add a special layer in front of the fully connected layer for classification, fix the rest pre-trained parameters, then learn new γ which can separate the unlabeled data set well. To classify a new task, one only needs to learn the linear layer for final classification and reuses the rest parts of the CNN.

5.1.3 Fine-tune θ^0 with New Parameters. The pre-trained θ^0 may not suit the structure of the new FSL task. Hence we need additional new parameter δ for the specialty of D^{train}. Specifically, this strategy fine-tunes θ^0 while learning δ, making the model parameter to learn becomes $\theta = \{\theta^0, \delta\}$. An illustration of this strategy is shown in Figure 11. Hoffman et al. [2013] use the parameters of

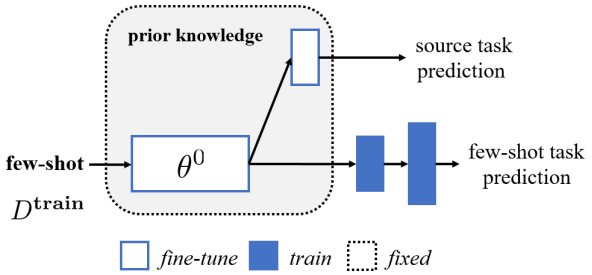


Fig. 11. Illustration of strategy fine-tunes θ^0 with new parameter.

the lower layers of a pre-trained CNN for feature embedding, while learns a linear classifier on top of it using D^{train}. Consider font style transfer task, Azadi et al. [2018] pre-train a network to capture the font in gray images, and fine-tune it together with the training of a network for generating stylish colored fonts.

5.1.4 Discussion. Methods discussed in this section reduce the effort of doing architecture search for $\mathcal{H}$ from the scratch. Since directly fine-tuning can easily overfit, methods that fine-tune a θ^0 with regularization turn to regularize or modify existing parameters. They usually consider a single θ^0 of some deep model. However, suitable existing parameters are not always easy to find. Another way is to aggregate a set of parameters θ^0's from related tasks into a suitable initialization. However, one must make sure that the knowledge embedded in these existing parameters is useful to the current task. Besides, it is costly to search over a large set of existing parameters to find the relevant ones. Fine-tune θ^0 with new parameters leads to more flexibility. However, given the few-shot D^{train}, one can only add limited parameters, otherwise overfitting may occur.

5.2 Refine Meta-learned θ

Methods fall in the following sections all are meta-learning methods[5]. Instead of working towards the unreliable h_I, this strategy directly targets at h^*. In the following, we denote the parameter

[5]See Appendix A for a formal definition of meta-learning.

of meta-learner as θ, and the task-specific parameter for meta-training task T_s as ϕ_{T_s} and meta-testing task T_t as ϕ_{T_t}. During training, the meta-learner (optimizer) parameterized by θ provides information to ϕ_{T_s} of learner for task T_s (optimizee), and the learner returns error signals such as gradients to meta-learner to improve it. Then, given a meta-testing task T_t with D_{T_t}, the meta-learner can be directly used while the learner can learn from D_{T_t}. We illustrate mainly use the meta-testing task T_t. An illustration of this strategy is shown in Figure 12.

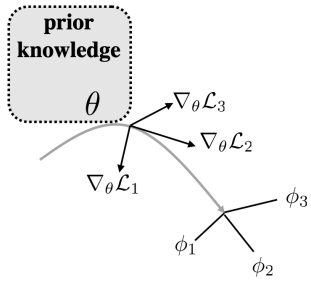


Fig. 12. Illustration of refining meta-learned θ strategy. The figure is adapted from [34].

5.2.1 Refine by Gradient Descent. This strategy refines the meta-learned θ by gradient descent. Model-Agnostic Meta-Learning (MAML) [34] is an representative method of this kind. It meta-learns a θ as a good initialization $\phi^0_{T_s}$ for task $T_s \sim p(T)$. This $\phi^0_{T_s}$ can be adjusted effectively through a few gradient descent using $D^{\text{train}}_{T_s}$ to obtain a good task specific ϕ_{T_s}. Mathematically, this is done by $\phi_{T_s} = \theta - \alpha \nabla_\theta \mathcal{L}^{\text{train}}_{T_s}(\theta)$, where $\mathcal{L}^{\text{train}}_{T_s}(\theta) = \sum_{(x^{(t)}, y^{(t)}) \in D^{\text{train}}_{T_s}} \ell^t(\theta)$ and α is a fixed step size to be chosen. Through summing over all samples, it provides a permutation-invariant ϕ_{T_s}. Then, meta-learner updates θ through the averaged gradient steps across all meta-training tasks, $\theta = \theta - \beta \nabla_\theta \sum_{T_s \sim P(T)} \mathcal{L}^{\text{test}}_{T_s}(\theta)$, where $\mathcal{L}^{\text{test}}_{T_s}(\theta) = \sum_{(x^{(t)}, y^{(t)}) \in D^{\text{train}}_{T_s}} \ell^t(\theta)$ and β is also a fixed step size.

MAML provides the same initialization for all tasks, while neglects the task-specific information. This only suits a set of very similar tasks, while works bad when tasks are distinct. In [69], it learns to choose from a subset of θ's a initialization $\phi^0_{T_t}$ for new T_t. In other words, it meta-learns task-specific subspace and metric for the learner to perform gradient descent. Therefore,different initializations of θ is provided for different T_t's.

As refining by gradient descent may not be reliable, regularization is used to correct the descent direction. ϕ_{T_t} is further adapted by model regression networks [132] in [47]. The adapted ϕ_{T_t} is regularized to be more close to model trained with many samples. The parameter of the model regression networks is learned by gradient descent like θ.

5.2.2 Refine in Consideration of Uncertainty. Learning with a few examples inevitably results in a model with high uncertainty [36]. Can the learned model predict for a new task with high confidence? Will the model be improved with more samples? The ability to measure this uncertainty provides a sign for active learning or further data collection [36].

Therefore are three kinds of uncertainty considered so far.

(1) *Uncertainty over the shared parameter θ.* A single θ may not act as a good initialization for all tasks. Therefore, by modeling θ's posterior distribution, one can sample appropriate initialization $\phi^0_{T_t}$'s for different T_t's. Finn et al. [2018] propose to model the prior distribution of θ. It is solved by point estimate of MAP. Instead, Yoon et al. [2018] learn the prior distribution of θ by Stein Variational Gradient Descent (SVGD). They learn a set of copies of θ with shared

parameter as $\phi^0_{T_t}$. These copies communicate with each other to decide the update direction. With the learned $\phi^0_{T_t}$, ϕ_{T_t} is obtained by taking a few SVGD steps.

(2) *Uncertainty over the task-specific parameter* ϕ_{T_t}. Each ϕ_{T_t} is then the point estimate of posterior over ϕ. However, as $D^{\text{train}}_{T_t}$ contains a few examples, the learned posterior can be skew. Therefore, Grant et al. [2018] improve MAML by replacing this point estimate for posterior by Laplace approximation. Ravi and Beatson [2019] do not use θ as the $\phi^0_{T_t}$. Instead, they learn inference networks to map $D^{\text{train}}_{T_t}$ to $\phi^0_{T_t}$. Then as in MAML, this inference networks are optimized by taking a few gradient descent steps using $D^{\text{train}}_{T_t}$ so as to adapt $\phi^0_{T_t}$ to ϕ_{T_t}. Finally, ϕ_{T_t} is used as the parameter for the variational distribution which approximates the posterior over ϕ_{T_t}.

(3) *Uncertainty over class n's class-specific parameter* $\phi_{T_t,n}$. Finally, class-specific uncertainty is modeled in [98]. It first maps each of the nth class in $D^{\text{train}}_{T_t}$ to the parameter of a class-conditional multivariate Gaussian in a lower dimensional latent space $\mathcal{Z}$ so as to sample the class-dependent initialization $\phi^0_{T_t,n}$. Then it uses an encoder to embed those $(x^{(i)}, y^{(i)})$ of class n to $\mathcal{Z}$, where gradient descent is efficiently taken with respect to $\phi^0_{T_t,n}$. Finally, a decoder is used to map the learned class-dependent parameter to the parameter of another class-conditional multivariate Gaussian which is used to sample $\phi_{T_t,n}$.

5.2.3 Discussion. In this section, θ to be refined is meta-learned from a set of tasks drawn from a task distribution. If the new task belongs to the task distribution which is used to train the meta-learner, using a meta-learned θ as the initialization for a new task can be more appropriate. However, this may not be the case in reality. By refining θ in consideration of uncertainty, it can measure the confidence of predicting for a new task, and provide signals for active learning or further data collection. However, learning to model this uncertainty can be much more costly, and is hard for beginners to design.

5.3 Learn Search Steps

The meta-learner used in previous section is to provide a good initialization θ for each h, while here the meta-learner is learned to output search steps or update rules of θ in $\mathcal{H}$ for each learner directly [5, 70]. It can sequentially take $(x^{(t)}, y^{(t)})$ and interact with the learner. In this way, there is no need to tune the step size α or find the best descent direction, the learning algorithm does that automatically. An illustration of this strategy is shown in Figure 13.

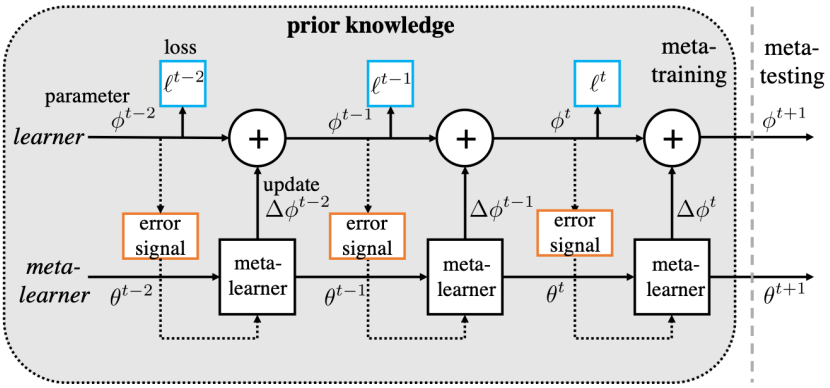


Fig. 13. Illustration of learning search steps strategy. The figure is adapted from [5].

Ravi and Larochelle [2017] firstly propose to solve FSL problem by learning the search steps in $\mathcal{H}$. They improve upon the work of Andrychowicz et al. [2016], which we will introduce as the

basis. During training, θ is not updated within each task T_s. At the tth iteration, the meta-learner calculates $\ell^t(\phi_{T_s}^{t-1}) = \ell(h(x^{(t)}; \phi_{T_s}^{t-1}), y^{(t)})$ using the tth sample $(x^{(t)}, y^{(t)})$ and $\phi_{T_s}^{t-1}$ learned at the $t-1$th iteration, and produces the updates $\Delta\phi_{T_s}^t$ by feeding $\nabla_{\phi_{T_s}^{t-1}}\ell^t(\phi_{T_s}^{t-1})$ to a coordinate LSTM parameterized by θ. Note that $\Delta\phi_{T_s}^t$ corresponds to the combination of α^t and gradient in (6). Finally, it updates $\phi_{T_s}^t$ as $\phi_{T_s}^t = \phi_{T_s}^{t-1} + \Delta\phi_{T_s}^t$. When the learning finishes, for a N-way-K-shot classification task, θ is improved using the loss measured on $D_{T_s}^{\text{test}}$ by gradient descent with back-propagation. Then by learning from a set of T_s's drawn from $P(T)$ to improve the meta-learner, it gets better and better at proposing efficient algorithms for FSL. Based on [5], apart from avoiding tuning α, Ravi and Larochelle [2017] further provide a good initialization for ϕ_{T_s}. They directly obtain $\phi_{T_s}^t$ by instantiating (6) by the cell update within a LSTM.

5.3.1 Discussion. Learning search steps methods alter the search strategy for the θ of h^* in $\mathcal{H}$. By meta-learning from a set of tasks, the meta-learner captures the common search strategy for this kind of tasks. As it sees further through prior knowledge, it can advise better search direction or speeds that may be considered later if the search is done only based on D^{train} of each task.

5.4 Summary

Methods in this section solve FSL problem by designing an appropriate algorithm. Specifically, they take advantage of prior knowledge to search for the θ which parameterizes h^* $\mathcal{H}$. The prior knowledge can alter the search by providing a good initial point to begin the search, or directly providing the search steps.

Refine existing parameters methods reduce the effort to do architecture search for $\mathcal{H}$ or learn from scratch. Using an existing θ^0 as the initialization, learning usually takes less computation cost and iterations to obtain a good hypothesis $h \in \mathcal{H}$. This strategy keeps the parameter of well-trained models rather than the raw data, making the migrating of knowledge scalable. Then, the learning target is refining these existing parameters. As θ^0 is learned from their own tasks rather than the current T, learning how to adapt these parameters to T is indirect for the goal and can easily go astray. In other words, this strategy usually sacrifices precision for convenience.

The other two strategies are both meta-learned, while meta-learned parameter θ acts as a good initialization to learn h^* and learned search steps are directly used as search steps for h^*. By learning from a series of related tasks, the meta-learned θ is closer to the final task-specific parameter ϕ_{T_t} in $\mathcal{H}$ for new task T_t. Then, learning search steps take care of the whole algorithm part in learning by using a meta-learner. However, the initialization for each task is affected by previous tasks, without checking the relatedness of the consecutive tasks. Once the tasks are negatively related or not related at all, this meta-learning method cannot provide a good initialization, and negative transfer incurs easily. Note that methods that refine in consideration of uncertainty can only release the sign of not suitable rather than solve the negative transfer problem.

6 FUTURE WORKS

FSL methods combine prior knowledge with a little supervised information D^{train} provided in E to make the learning of the supervised target T feasible. Here we discuss the four key directions for further development of FSL, i.e., problem setup, techniques, applications and theories.

6.1 Problem Setup

Existing FSL methods use prior knowledge from a single modality, such as images, texts, and videos. Although D^{train} has a few examples in the current modality, there may exist a cheaper modality with abundant supervised samples. For example, one extinct animal class may have limited examples in

the visual domain. Nevertheless, it can be examined in detail in the textual domain such as textbooks or web pages, as people tend to pay special attention to rare class. Therefore, multi-modality prior knowledge can be used to provide prior knowledge in complementary views. The idea of using multi-modality is used in zero-shot learning frequently. They consider prior knowledge including attributes [1, 54], WordNet [1, 54], word embeddings [121, 128] , co-occurrence statistics [77], and knowledge graphs [129].

Recently, there are some works trying to adapt zero-shot learning methods to deal with FSL problems by the supervised data in D^{train}. These methods either use the few-shot D^{train} to fine-tune the parameter learned by zero-shot learning methods [1, 54], or force the embedding or model learned by multiple modalities to match on a shared space [121, 128]. Fine-tuning in this way may lead to overfitting, and may not embed enough information of the new task to the parameter. As modalities may have different structure, for example, texts have to obey syntactic structure while images do not. Forcing them to match may deteriorate the learning. A promising direction is to use the multi-modality as one type of prior knowledge and design methods to deal with them based on those mentioned in data, model and algorithm.

6.2 Techniques

Existing FSL methods are categorized into data (Section 3), model (Section 4), and and algorithm (Section 5). Indeed, to improve them, each component of these methods can be replaced by more recent and advanced ones. For example, using ResNet [50] as embedding function can be better than using VGG [110]. Further, designing a hybrid method which can explicitly quantify the contribution of data, model and algorithm, and adjust the method accordingly can be useful.

Special attention is paid to meta-learning methods, which we discuss in both Section 4 and Section 5. By learning across tasks, it can adapt to new tasks rapidly with small inference cost. However, the tasks considered in meta-learning methods are mainly drawn from a single task distribution $p(T)$. Nevertheless, in practice, we obtain various kinds of tasks where task relatedness is not determined due to high cost or it is just too hard. In this case, directly using these tasks can lead to negative transfer [87]. Finally, they mainly consider static and fixed $P(T)$ [34, 93]. However, $p(T)$ is usually dynamic [35], where new tasks keep coming in streaming order. $p(T)$ should incorporate this change Besides, how to avoid catastrophic forgetting [60] for this dynamic $p(T)$ is another issue, meaning information about past tasks should not be forgotten.

Automated machine learning (AutoML) [138] constructs machine learning programs without human assistance and within limited computational budgets. Existing FSL methods each have pros and cons (as discussed in previous sections), and there is no absolute winner for all settings. Besides, the hypothesis space $\mathcal{H}$ and search strategies in $\mathcal{H}$ still reply on human design. A possible direction is to extend the automated feature engineering [57], model selection [62] and neural architecture search [148] of AutoML to FSL methods. Therefore, one can free human from manually designing methods for different settings by making this process automated.

6.3 Applications

Existing works mainly deal with computer vision applications, such as character recognition [15, 33, 34, 56, 61, 82, 99, 100, 105, 108, 118, 127, 134] and image classification [34, 61, 82, 93, 105, 108, 115, 118, 120, 127, 131, 132, 136]. This is because visual information is easy to acquire and has been extensively examined in machine learning. There are many mature techniques to be transferred to few-shot setting. Moreover, the performance of visual information can be easily understood and evaluated by humans. Currently, the two benchmarks of character recognition and image classification, Ominiglot and miniImageNet, already got very high accuracy leaving

few space for improvement [119]. A large-scale diverse data set constructed from different data sources has been recently presented in [119] as a new benchmark. Further, more computer vision applications can be explored, such as image retrieval [118], object tracking [15], gesture recognition [88], image captioning and visual question answering [28] and video event detection [137].

Apart from computer vision applications, few-shot problem starts to be considered in other fields. In natural language processing, examples are translation [56] and language modeling [127]. In recommendation, cold-start item recommendation has been considered in [126]. In medical applications, few-shot drug discovery is attempted in [3]. One-shot architecture search is considered in [21]. Finally, robotics and game playing by reinforcement-learning [113] from limited experience in new environment start to draw attention [30, 34, 79]. Existing applications are one-shot imitation [30], multi-armed bandits [30], visual navigation [30], continuous control in locomotion [34]. Recently, these applications are further extended to dynamic environment [2, 84].

6.4 Theories

FSL uses prior knowledge to compensate for the lack of supervised information. Basically, the use of prior knowledge reduces the sample complexity, which is proved for methods that utilize unlabeled data set [12, 107], multitask learning [85], generative modeling [39], refining existing parameters [76] and meta-learning [4]. However, the analysis is lacked for recent methods such as conditional embedding learning and learning with external memory methods. Finn and Levine [2018] show that using a sufficiently deep model, MAML [34] can approximate any hypothesis h. Apart from that, recall FSL problems sometimes become domain adaptation problems, such as those mentioned in Section 4.1. Existing bounds on domain adaptation may be inspiring [11, 18], i.e., better risk bounds of feed forward neural networks resulting from transferring representations with fine-tuning is shown in [76]. More recently, the risk of transferring a model trained from other tasks to current task is examined in [25], only one specific meta-learning method. In sum, only some methods under specific assumptions are bounded in a way so far.

Finally, convergence for methods in algorithms is not fully understood. Indeed, using a few examples and prior knowledge to search h in $\mathcal{H}$ does not guarantee to obtain the minimum empirical risk. Especially, for meta-learning methods, they optimize θ over the task distribution instead of empirical risk of a single task. The effect of learning this way is unclear so far. Recently, this is analyzed in [37]. They provide sufficient conditions for one type of meta-learning methods to converge: the meta-learner is the lower layers of a deep models and the learner is the last layer, and they are optimized by gradient descent. However, a general or universal analysis for convergence of meta-learning methods is still missing.

7 CONCLUSION

Few-Shot Learning (FSL) targets at bridging this gap between AI and human-like learning. It can learn new tasks of limited supervised information by incorporating prior knowledge. FSL acts as a test-bed for AI, helps to relieve the burden of collecting large-scale supervised date for industrial usages, or makes the learning of rare cases possible. With both academic dream of AI and industrial needs for cheap learning, FSL draws much attention and becomes a hot topic. In this survey, we provide a comprehensive and systematic review of FSL. We first formally define FSL, and discuss the relatedness and difference of FSL with respect to relevant learning problems such as semi-supervised learning, imbalanced learning, transfer learning and meta-learning. Then, we point out the core issue of FSL based on error decomposition in machine learning. We figure out that it is the unreliable empirical risk minimizer that makes FSL hard to learn. This can be relieved by satisfying or reducing the sample complexity of learning. Understanding the core issue can help

categorize different works into data, model and algorithm according to how they solve the core issue using prior knowledge: data augments the supervised experience of FSL, model constrains the hypothesis space of FSL, and algorithm alters the search strategy for the parameter of the best hypothesis in hypothesis space to solve FSL. Within each category, the pros and cons of different categories are thoroughly discussed and some summary of the insights under each category is presented. For future works, we provide possible directions including problem setup, techniques, applications and theories to explore, in hope of inspiring future research in FSL.

A META-LEARNING

The learning of meta-learner needs large-scale data. Let $p(T)$ be the distribution of task T. In mete-learning, it learns from a set of tasks $T_s \sim p(T)$. Each task T_s operates on data set D_{T_s} of N classes where $D_{T_s} = \{D_{T_s}^{\text{train}}, D_{T_s}^{\text{test}}\}$ consists of training set $D_{T_s}^{\text{train}}$ and testing set $D_{T_s}^{\text{test}}$. Each learner learns from $D_{T_s}^{\text{train}}$ and measures test error on $D_{T_s}^{\text{test}}$. The parameter θ of meta-learner learns to minimize the error across all learners by

$$\theta = \arg\min_{\theta} \mathbb{E}_{T_s \sim p(T)} \sum_{((x^{(i)}, y^{(i)})) \in D_{T_s}} \ell(h(x^{(i)}; \theta), y^{(i)}),$$

Then in meta-testing, another disjoint set of tasks $T_t \sim p(T)$ is used to test the generalization ability of meta-learner. Each T_t works on data set D_{T_t} of N' classes where $D_{T_t} = \{D_{T_t}^{\text{train}}, D_{T_t}^{\text{test}}\}$. The learner learns from training set $D_{T_t}^{\text{train}}$ and tests on test set $D_{T_t}^{\text{test}}$. The averaged loss across T_t's is taken as the meta-learning testing error. To understand meta-learning, an illustration of its setup is in Figure 14.

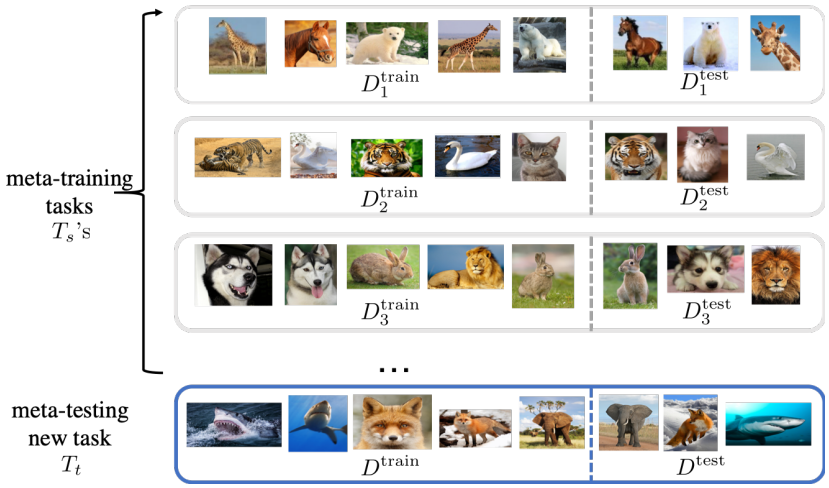


Fig. 14. Meta-learning setup. The figure is adapted from [93].

REFERENCES

[1] Z. Akata, F. Perronnin, Z. Harchaoui, and C. Schmid. 2013. Label-embedding for attribute-based classification. In *International Conference on Computer Vision and Pattern Recognition*. 819–826.

[2] M. Al-Shedivat, T. Bansal, Y. Burda, I. Sutskever, I. Mordatch, and P. Abbeel. 2018. Continuous adaptation via meta-learning in nonstationary and competitive environments. In *International Conference on Learning Representations*.

[3] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. Pande. 2017. Low Data Drug Discovery with One-Shot Learning. *ACS Central Science* 3, 4 (2017), 283–293.

[4] R. Amit and R. Meir. 2018. Meta-Learning by Adjusting Priors Based on Extended PAC-Bayes Theory. In *International Conference on Machine Learning*. 205–214.

[5] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, and N. de Freitas. 2016. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*. 3981–3989.

[6] S. Arik, J. Chen, K. Peng, W. Ping, and Y. Zhou. 2018. Neural voice cloning with a few samples. In *Advances in Neural Information Processing Systems*. 10040–10050.

[7] S. Azadi, M. Fisher, V. G. Kim, Z. Wang, E. Shechtman, and T. Darrell. 2018. Multi-content gan for few-shot font style transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*. 7564–7573.

[8] P. Bachman, A. Sordoni, and A. Trischler. 2017. Learning Algorithms for Active Learning. In *International Conference on Machine Learning*. 301–310.

[9] E. Bart and S. Ullman. 2005. Cross-generalization: Learning novel classes from a single example by feature replacement. In *IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 1. IEEE, 672–679.

[10] S. Bartunov and D. Vetrov. 2018. Few-shot generative modelling with generative matching networks. In *International Conference on Artificial Intelligence and Statistics*. 670–678.

[11] S. Ben-David, J. Blitzer, K. Crammer, and F. Pereira. 2007. Analysis of representations for domain adaptation. In *Advances in Neural Information Processing Systems*. 137–144.

[12] S. Ben-David, T. Lu, and D. Pál. 2008. Does Unlabeled Data Provably Help? Worst-case Analysis of the Sample Complexity of Semi-Supervised Learning.. In *Conference on Learning Theory*. 33–44.

[13] S. Benaim and L. Wolf. 2018. One-Shot Unsupervised Cross Domain Translation. In *Advances in Neural Information Processing Systems*.

[14] L. Bertinetto, J. F. Henriques, P. Torr, and A. Vedaldi. 2019. Meta-learning with differentiable closed-form solvers. In *International Conference on Learning Representations*. https://openreview.net/forum?id=HyxnZh0ct7

[15] L. Bertinetto, J. F. Henriques, J. Valmadre, P. Torr, and A. Vedaldi. 2016. Learning feed-forward one-shot learners. In *Advances in Neural Information Processing Systems*. 523–531.

[16] I. Biederman. 1987. Recognition-by-components: a theory of human image understanding. *Psychological Review* 94, 2 (1987), 115.

[17] C. M. Bishop. 2006. *Pattern recognition and machine learning*. springer.

[18] J. Blitzer, K. Crammer, A. Kulesza, F. Pereira, and J. Wortman. 2008. Learning bounds for domain adaptation. In *Advances in Neural Information Processing Systems*. 129–136.

[19] L. Bottou and O. Bousquet. 2008. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*. 161–168.

[20] L. Bottou, F. E. Curtis, and J. Nocedal. 2018. Optimization methods for large-scale machine learning. *Siam Review* 60, 2 (2018), 223–311.

[21] A. Brock, T. Lim, J.M. Ritchie, and N. Weston. 2018. SMASH: One-Shot Model Architecture Search through HyperNetworks. In *International Conference on Learning Representations*.

[22] Q. Cai, Y. Pan, T. Yao, C. Yan, and T. Mei. 2018. Memory Matching Networks for One-Shot Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4080–4088.

[23] R. Caruana. 1997. Multitask learning. *Machine learning* 28, 1 (1997), 41–75.

[24] J. J. Craig. 2009. *Introduction to robotics: mechanics and control, 3/E*. Pearson Education India.

[25] G. Denevi, C. Ciliberto, D. Stamos, and M. Pontil. 2018. Learning To Learn Around A Common Mean. In *Advances in Neural Information Processing Systems*. 10190–10200.

[26] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition*. Ieee, 248–255.

[27] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. 2014. DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In *International Conference on Machine Learning*. 647–655.

[28] X. Dong, L. Zhu, D. Zhang, Y. Yang, and F. Wu. 2018. Fast Parameter Adaptation for Few-shot Image Captioning and Visual Question Answering. In *ACM Multimedia Conference on Multimedia Conference*. ACM, 54–62.

[29] M. Douze, A. Szlam, B. Hariharan, and H. Jégou. 2018. Low-shot learning with large-scale diffusion. In *IEEE Conference on Computer Vision and Pattern Recognition*. 3349–3358.

[30] Y. Duan, M. Andrychowicz, B. Stadie, J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba. 2017. One-Shot Imitation Learning. In *Advances in Neural Information Processing Systems*. –.

[31] H. Edwards and A. Storkey. 2017. Towards a Neural Statistician. In *International Conference on Learning Representations*.

[32] L. Fei-Fei, R. Fergus, and P. Perona. 2006. One-shot learning of object categories. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28, 4 (2006), 594–611.

[33] M. Fink. 2005. Object classification from a single example utilizing class relevance metrics. In *Advances in Neural Information Processing Systems*. 449–456.

[34] C. Finn, P. Abbeel, and S. Levine. 2017. Model-Agnostic Meta-Learning for Fast Adaptation of Deep Networks. In *International Conference on Machine Learning*. 1126–1135.

[35] C. Finn and S. Levine. 2018. Meta-learning and universality: Deep representations and gradient descent can approximate any learning algorithm. In *International Conference on Learning Representations*.

[36] C. Finn, K. Xu, and S. Levine. 2018. Probabilistic model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*. 9537–9548.

[37] L. Franceschi, P. Frasconi, S. Salzo, R. Grazzi, and M. Pontil. 2018. Bilevel Programming for Hyperparameter Optimization and Meta-Learning. In *International Conference on Machine Learning*. 1563–1572.

[38] H. Gao, Z. Shou, A. Zareian, H. Zhang, and S. Chang. 2018. Low-shot Learning via Covariance-Preserving Adversarial Augmentation Networks. In *Advances in Neural Information Processing Systems*. 983–993.

[39] P. Germain, F. Bach, A. Lacoste, and S. Lacoste-Julien. 2016. PAC-Bayesian theory meets Bayesian inference. In *Advances in Neural Information Processing Systems*. 1884–1892.

[40] S. Gidaris and N. Komodakis. 2018. Dynamic few-shot visual learning without forgetting. In *IEEE Conference on Computer Vision and Pattern Recognition*. 4367–4375.

[41] I. Goodfellow, Y. Bengio, and A. Courville. 2016. *Deep learning*. MIT press.

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. 2014. Generative adversarial nets. In *Advances in Neural Information Processing Systems*. 2672–2680.

[43] J. Gordon, J. Bronskill, M. Bauer, S. Nowozin, and R. Turner. 2019. Meta-Learning Probabilistic Inference for Prediction. In *International Conference on Learning Representations*.

[44] E. Grant, C. Finn, S. Levine, T. Darrell, and T. Griffiths. 2018. Recasting Gradient-Based Meta-Learning as Hierarchical Bayes. In *International Conference on Learning Representations*.

[45] A. Graves and J. Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks* 18, 5-6 (2005), 602–610.

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

[47] L.-Y. Gui, Y.-X. Wang, D. Ramanan, and J. Moura. 2018. Few-shot human motion prediction via meta-learning. In *European Conference on Computer Vision*. 432–450.

[48] B. Hariharan and R. Girshick. 2017. Low-Shot Visual Recognition by Shrinking and Hallucinating Features. In *IEEE International Conference on Computer Vision*.

[49] H. He and E. A. Garcia. 2008. Learning from imbalanced data. *IEEE Transactions on Knowledge & Data Engineering* 9 (2008), 1263–1284.

[50] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *International Conference on Computer Vision and Pattern Recognition*. 770–778.

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

[52] A. E. Hoerl and R. W. Kennard. 1970. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics* 12, 1 (1970), 55–67.

[53] J. Hoffman, E. Tzeng, J. Donahue, Y. Jia, K. Saenko, and T. Darrell. 2013. One-shot adaptation of supervised deep convolutional models, In Proceedings of the 2nd International Conference on Learning Representations. *International Conference on Learning Representations*.

[54] S. J. Hwang and L. Sigal. 2014. A unified semantic embedding: Relating taxonomies and attributes. In *Advances in Neural Information Processing Systems*. 271–279.

[55] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *ACM international conference on Multimedia*. ACM, 675–678.

[56] Ł. Kaiser, O. Nachum, A. Roy, and S. Bengio. 2017. Learning to remember rare events. In *International Conference on Learning Representations*.

[57] J. M. Kanter and K. Veeramachaneni. 2015. Deep feature synthesis: Towards automating data science endeavors. In *IEEE International Conference on Data Science and Advanced Analytics*. IEEE, 1–10.

[58] R. Keshari, M. Vatsa, R. Singh, and A. Noore. 2018. Learning structure and strength of CNN filters for small sample size training. In *IEEE Conference on Computer Vision and Pattern Recognition*. 9349–9358.

[59] D. P. Kingma and M. Welling. 2014. Auto-encoding variational bayes. In *International Conference on Learning Representations*.

[60] J. Kirkpatrick, R. Pascanu, N. Rabinowitz, J. Veness, G. Desjardins, A. A. Rusu, K. Milan, J. Quan, T. Ramalho, A. Grabska-Barwinska, et al. 2017. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences* 114, 13 (2017), 3521–3526.

[61] G. Koch. 2015. *Siamese neural networks for one-shot image recognition.* Ph.D. Dissertation. University of Toronto.

[62] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. 2017. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *The Journal of Machine Learning Research* 18, 1 (2017), 826–830.
[63] J. Kozerawski and M. Turk. 2018. CLEAR: Cumulative LEARning for One-Shot One-Class Image Recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*. 3446–3455.
[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*. 1097–1105.
[65] R. Kwitt, S. Hegenbart, and M. Niethammer. 2016. One-shot learning of scene locations via feature trajectory transfer. In *IEEE Conference on Computer Vision and Pattern Recognition*. 78–86.
[66] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum. 2015. Human-level concept learning through probabilistic program induction. *Science* 350, 6266 (2015), 1332–1338.
[67] B. M. Lake, T. D. Ullman, J. B. Tenenbaum, and S. J. Gershman. 2017. Building machines that learn and think like people. *Behavioral and Brain Sciences* 40 (2017).
[68] C. H. Lampert, H. Nickisch, and S. Harmeling. 2009. Learning to detect unseen object classes by between-class attribute transfer. In *International Conference on Computer Vision and Pattern Recognition*. 951–958.
[69] Y. Lee and S. Choi. 2018. Gradient-Based Meta-Learning with Learned Layerwise Metric and Subspace. In *International Conference on Machine Learning*. 2933–2942.
[70] K. Li and J. Malik. 2016. Learning to optimize. *arXiv preprint arXiv:1606.01885* (2016).
[71] X.-L. Li, P. S. Yu, B. Liu, and S.-K. Ng. 2009. Positive unlabeled learning for data stream classification. In *SIAM International Conference on Data Mining*. SIAM, 259–270.
[72] B. Liu, X. Wang, M. Dixit, R. Kwitt, and N. Vasconcelos. 2018. Feature space transfer for data augmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*. 9090–9098.
[73] Y. Liu, J. Lee, M. Park, S. Kim, E. Yang, S. Hwang, and Y Yang. 2019. LEARNING TO PROPAGATE LABELS: TRANSDUCTIVE PROPAGATION NETWORK FOR FEW-SHOT LEARNING. In *International Conference on Learning Representations*.
[74] Z. Luo, Y. Zou, J. Hoffman, and L. Fei-Fei. 2017. Label efficient learning of transferable representations acrosss domains and tasks. In *Advances in Neural Information Processing Systems*. 165–177.
[75] S. Mahadevan and P. Tadepalli. 1994. Quantifying prior determination knowledge using the pac learning model. *Machine Learning* 17, 1 (1994), 69–105.
[76] D. McNamara and M.-F. Balcan. 2017. Risk bounds for transferring representations with and without fine-tuning. In *International Conference on Machine Learning*. 2373–2381.
[77] T. Mensink, E. Gavves, and C. Snoek. 2014. Costa: Co-occurrence statistics for zero-shot classification. In *International Conference on Computer Vision and Pattern Recognition*. 2441–2448.
[78] E. G. Miller, N. E. Matsakis, and P. A. Viola. 2000. Learning from one example through shared densities on transforms. In *IEEE Conference on Computer Vision and Pattern Recognition*, Vol. 1. 464–471.
[79] N. Mishra, M. Rohaninejad, X. Chen, and P. Abbeel. 2018. A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations*.
[80] M. T. Mitchell. 1997. *Machine learning*. McGraw-Hill.
[81] S. Motiian, Q. Jones, S. Iranmanesh, and G. Doretto. 2017. Few-shot adversarial domain adaptation. In *Advances in Neural Information Processing Systems*. 6670–6680.
[82] T. Munkhdalai and H. Yu. 2017. Meta Networks. In *Advances in Neural Information Processing Systems*. –.
[83] T. Munkhdalai, X. Yuan, S. Mehri, and A. Trischler. 2018. Rapid adaptation with conditionally shifted neurons. In *International Conference on Machine Learning*. 3661–3670.
[84] A. Nagabandi, C. Finn, and S. Levine. 2018. Deep Online Learning via Meta-Learning: Continual Adaptation for Model-Based RL. In *International Conference on Learning Representations*.
[85] H. Nguyen and L. Zakynthinou. 2018. Improved Algorithms for Collaborative PAC Learning. In *Advances in Neural Information Processing Systems*. 7631–7639.
[86] B. Oreshkin, P. R. López, and A. Lacoste. 2018. TADAM: Task dependent adaptive metric for improved few-shot learning. In *Advances in Neural Information Processing Systems*. 719–729.
[87] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 10, 22 (2010), 1345–1359.
[88] T. Pfister, J. Charles, and A. Zisserman. 2014. Domain-adaptive discriminative one-shot learning of gestures. In *European Conference on Computer Vision*. Springer, 814–829.
[89] H. Qi, M. Brown, and D. G. Lowe. 2018. Low-Shot Learning With Imprinted Weights. In *IEEE Conference on Computer Vision and Pattern Recognition*. 5822–5830.
[90] S. Qiao, C. Liu, W. Shen, and A. L. Yuille. 2018. Few-shot image recognition by predicting parameters from activations. In *IEEE Conference on Computer Vision and Pattern Recognition*. 7229–7238.

[91] T. Ramalho and M. Garnelo. 2019. Adaptive Posterior Learning: few-shot learning with a surprise-based memory module. In *International Conference on Learning Representations*.
[92] S. Ravi and A. Beatson. 2019. Amortized Bayesian Meta-Learning. In *International Conference on Learning Representations*.
[93] S. Ravi and H. Larochelle. 2017. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*.
[94] S. Reed, Y. Chen, T. Paine, A. van den Oord, S. M. A. Eslami, D. Rezende, O. Vinyals, and N. de Freitas. 2018. Few-shot Autoregressive Density Estimation: Towards Learning to Learn Distributions. In *International Conference on Learning Representations*.
[95] M. Ren, S. Ravi, E. Triantafillou, J. Snell, K. Swersky, J. B. Tenenbaum, H. Larochelle, and R. S. Zemel. 2018. Meta-Learning for Semi-Supervised Few-Shot Classification. In *International Conference on Learning Representations*.
[96] D. Rezende, I. Danihelka, K. Gregor, and D. Wierstra. 2016. One-Shot Generalization in Deep Generative Models. In *International Conference on Machine Learning*. 1521–1529.
[97] S. Ruder. 2017. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098* (2017).
[98] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell. 2019. Meta-Learning with Latent Embedding Optimization. In *International Conference on Learning Representations*.
[99] R. Salakhutdinov, J. Tenenbaum, and A. Torralba. 2012. One-shot learning with a hierarchical nonparametric Bayesian model. In *ICML Workshop on Unsupervised and Transfer Learning*. 195–206.
[100] A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. 2016. Meta-learning with memory-augmented neural networks. In *International Conference on Machine Learning*. 1842–1850.
[101] V. G. Satorras and J. B. Estrach. 2018. Few-Shot Learning with Graph Neural Networks. In *International Conference on Learning Representations*.
[102] E. Schwartz, L. Karlinsky, J. Shtok, S. Harary, M. Marder, A. Kumar, R. Feris, R. Giryes, and A. Bronstein. 2018. Delta-encoder: an effective sample synthesis method for few-shot object recognition. In *Advances in Neural Information Processing Systems 31*. 2850–2860.
[103] B. Settles. 2009. *Active learning literature survey.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.
[104] J. Shu, Z. Xu, and D Meng. 2018. Small sample learning in big data era. *arXiv preprint arXiv:1808.04572* (2018).
[105] P. Shyam, S. Gupta, and A. Dukkipati. 2017. Attentive Recurrent Comparators. In *International Conference on Machine Learning*. 3173–3181.
[106] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.
[107] A. Singh, R. Nowak, and J. Zhu. 2009. Unlabeled data: Now it helps, now it doesn't. In *Advances in Neural Information Processing Systems*. 1513–1520.
[108] J. Snell, K. Swersky, and R. S. Zemel. 2017. Prototypical Networks for Few-shot Learning. In *Advances in Neural Information Processing Systems*. –.
[109] M. D. Spivak. 1970. *A comprehensive introduction to differential geometry.* Publish or perish.
[110] R. K. Srivastava, K. Greff, and J. Schmidhuber. 2015. Training very deep networks. In *Advances in Neural Information Processing Systems*. 2377–2385.
[111] S. Sukhbaatar, J. Weston, R. Fergus, et al. 2015. End-to-end memory networks. In *Advances in Neural Information Processing Systems*. 2440–2448.
[112] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales. 2018. Learning to compare: Relation network for few-shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition*. 1199–1208.
[113] R. S. Sutton and A. G. Barto. 2018. *Reinforcement learning: An introduction.*
[114] M. Talagrand et al. 1994. Sharper bounds for Gaussian and empirical processes. *The Annals of Probability* 22, 1 (1994), 28–76.
[115] K. D. Tang, M. F. Tappen, R. Sukthankar, and C. H. Lampert. 2010. Optimizing one-shot recognition with micro-set learning. In *International Conference on Computer Vision and Pattern Recognition*. 3027–3034.
[116] R. Tibshirani. 1996. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society* 58, 1 (1996), 267–288.
[117] A. Torralba, J. B. Tenenbaum, and R. R. Salakhutdinov. 2011. Learning to learn with compound HD models. In *Advances in Neural Information Processing Systems*. 2061–2069.
[118] E. Triantafillou, R. Zemel, and R. Urtasun. 2017. Few-shot learning through an information retrieval lens. In *Advances in Neural Information Processing Systems*. 2255–2265.

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

[120] Y.-H. Tsai, L.-K. Huang, and R. Salakhutdinov. 2017. Learning Robust Visual-Semantic Embeddings. In *IEEE Conference on Computer Vision and Pattern Recognition*. 3571–3580.

[121] Y. H. Tsai and R. Salakhutdinov. 2017. Improving One-Shot Learning through Fusing Side Information. *arXiv preprint arXiv:1710.08347* (2017).

[122] M. A. Turing. 1950. Computing machinery and intelligence. *Mind* 59, 236 (1950), 433–433.

[123] A. Van den Oord, N. Kalchbrenner, L. Espeholt, O. Vinyals, A. Graves, et al. 2016. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*. 4790–4798.

[124] V. N. Vapnik. 1992. Principles of risk minimization for learning theory. In *Advances in Neural Information Processing Systems*. 831–838.

[125] V. N. Vapnik and A. J. Chervonenkis. 1974. Theory of pattern recognition. (1974).

[126] M. Vartak, A. Thiagarajan, C. Miranda, J. Bratman, and H. Larochelle. 2017. A meta-learning perspective on cold-start recommendations for items. In *Advances in Neural Information Processing Systems*. 6904–6914.

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

[128] P. Wang, L. Liu, C. Shen, Z. Huang, A. van den Hengel, and H. Tao Shen. 2017. Multi-attention network for one shot learning. In *IEEE Conference on Computer Vision and Pattern Recognition*. 2721–2729.

[129] X. Wang, Y. Ye, and A. Gupta. 2018. Zero-shot recognition via semantic embeddings and knowledge graphs. In *IEEE Conference on Computer Vision and Pattern Recognition*. 6857–6866.

[130] Y.-X. Wang, R. Girshick, M. Hebert, and Bharath Hariharan. 2018. Low-shot learning from imaginary data. In *IEEE Conference on Computer Vision and Pattern Recognition*. 7278–7286.

[131] Y.-X. Wang and M. Hebert. 2016. Learning from Small Sample Sets by Combining Unsupervised Meta-Training with CNNs. In *Advances in Neural Information Processing Systems*. 244–252.

[132] Y.-X. Wang and M. Hebert. 2016. Learning to learn: Model regression networks for easy small sample learning. In *European Conference on Computer Vision*. 616–634.

[133] J. Weston, S. Chopra, and A. Bordes. 2014. Memory networks. *arXiv preprint arXiv:1410.3916* (2014).

[134] M. Woodward and C. Finn. 2017. Active One-shot Learning. *arXiv preprint arXiv:1702.06559* (2017).

[135] Y. Xian, C. H. Lampert, B. Schiele, and Z. Akata. 2018. Zero-shot learning-a comprehensive evaluation of the good, the bad and the ugly. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[136] Z. Xu, L. Zhu, and Y. Yang. 2017. Few-Shot Object Recognition from Machine-Labeled Web Images. In *International Conference on Computer Vision and Pattern Recognition*. –.

[137] W. Yan, J. Yap, and G. Mori. 2015. Multi-Task Transfer Methods to Improve One-Shot Learning for Multimedia Event Detection. In *British Machine Vision Conference*. Citeseer.

[138] Q. Yao, M. Wang, E. H. Jair, I. Guyon, Y.-Q. Hu, Y.-F. Li, W.-W. Tu, Q. Yang, and Y. Yu. 2018. Taking human out of learning applications: A survey on automated machine learning. *arXiv preprint arXiv:1810.13306* (2018).

[139] D. Yoo, H. Fan, V. N. Boddeti, and K. M. Kitani. 2018. Efficient k-shot learning with regularized deep networks. In *AAAI Conference on Artificial Intelligence*.

[140] J. Yoon, T. Kim, O. Dia, S. Kim, Y. Bengio, and S. A. 2018. Bayesian model-agnostic meta-learning. In *Advances in Neural Information Processing Systems*. 7343–7353.

[141] C. Zhang, J. Butepage, H. Kjellstrom, and S. Mandt. 2018. Advances in variational inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2018).

[142] R. Zhang, T. Che, Z. Ghahramani, Y. Bengio, and Y. Song. 2018. MetaGAN: An Adversarial Approach to Few-Shot Learning. In *Advances in Neural Information Processing Systems*. 2371–2380.

[143] Y. Zhang, H. Tang, and K. Jia. 2018. Fine-grained visual categorization using meta-learning optimization with sample selection of auxiliary data. In *European Conference on Computer Vision*. 233–248.

[144] Y. Zhang and Q. Yang. 2017. A survey on multi-task learning. *arXiv preprint arXiv:1707.08114* (2017).

[145] F. Zhao, J. Zhao, S. Yan, and J. Feng. 2018. Dynamic Conditional Networks for Few-Shot Learning. In *European Conference on Computer Vision*.

[146] L. Zhu and Y. Yang. 2018. Compound Memory Networks for Few-shot Video Classification. In *European Conference on Computer Vision*. 751–766.

[147] X. J. Zhu. 2005. *Semi-supervised learning literature survey.* Technical Report. University of Wisconsin-Madison Department of Computer Sciences.

[148] B. Zoph and Q. V. Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578* (2016).