# LEARNING FROM FEW EXAMPLES: A SUMMARY OF APPROACHES TO FEW-SHOT LEARNING

**Archit Parnami and Minwoo Lee**
Department of Computer Science
The University of North Carolina at Charlotte
Charlotte, NC, USA
{aparnami, minwoo.lee}@uncc.edu

## ABSTRACT

Few-Shot Learning refers to the problem of learning the underlying pattern in the data just from a few training samples. Requiring a large number of data samples, many deep learning solutions suffer from data hunger and extensively high computation time and resources. Furthermore, data is often not available due to not only the nature of the problem or privacy concerns but also the cost of data preparation. Data collection, preprocessing, and labeling are strenuous human tasks. Therefore, few-shot learning that could drastically reduce the turnaround time of building machine learning applications emerges as a low-cost solution. This survey paper comprises a representative list of recently[1] proposed few-shot learning algorithms. Given the learning dynamics and characteristics, the approaches to few-shot learning problems are discussed in the perspectives of meta-learning, transfer learning, and hybrid approaches (i.e., different variations of the few-shot learning problem).

[1] Until January 2020.

## Contents



## List of Tables

# 1 Introduction

The field of Artificial Intelligence (AI) has been through ups and downs since its inception in the 1950s. Yet, the last few years have been marked by exceptional progress in the field of AI. Much of this progress can be attributed to recent advances in "deep learning" characterized by learning large neural network-style models with multiple layers of representation. These models have shown great performance in a variety of tasks with large amounts of labeled data in image classification [1], machine translation [2] and speech modeling [3]. However, these achievements have relied on the fact that the optimization of these deep, high-capacity models requires many iterative updates across many labeled examples. This type of optimization breaks down in the small data regime where we want to learn from very few labeled examples. In contrast, humans can quickly learn to solve a new problem just from a few examples. For instance, given a few photos of a stranger, one can easily identify the same person from a large number of photos. This is not only due to the human mind's computational power but also to its ability to synthesize and learn new information from previously learned information. For example, if a person has a skill for riding a bicycle, that skill can prove helpful when learning to ride a motorcycle.

Recent years have seen a rise of research attempting to bridge this gap between human-like learning and machine learning, which has given birth to this new sub-field of Machine Learning known as **Few-Shot Learning (FSL)** i.e., **the ability of machine learning models to generalize from few training examples**. When there is only one example to learn from, FSL is also referred to as One-Shot Learning. The motivation for FSL lies in the fact that models excelling at this task would have many useful applications. First, we would not be required to collect thousands of labeled examples to attain a reasonable performance on a new task which would help alleviate the data-gathering effort and reduce the computation costs and time spent in training a model. Furthermore, in many fields, data is hard or impossible to acquire due to reasons such as privacy and safety. Models that generalize from a few examples would be able to capture this type of data effectively. Therefore in this survey, we study various approaches that have been recently proposed in an attempt to solve the problem of Few-Shot Learning. We categorize the FSL approaches as seen in Figure 1.

Much of the work in this survey is inspired from prior works that attempted to summarize the field of few-shot learning. Wang et al. [4] defines the core issue of FSL as an unreliable empirical risk minimizer that makes it a hard problem. Chen et al. [5] present a comparative analysis of several representative few-shot classification algorithms. Similar to ours, Weng [6] discusses meta-learning approaches to FSL. In addition, we discuss non-meta-learning and hybrid meta-learning approaches to FSL. We also extend the discussion of main meta-learning approaches by including the recent state-of-the-art methods.

The rest of the survey is organized as follows. Section 2 covers meta-learning which is a prerequisite to understand recent FSL methods. Section 3 defines the few-shot classification problem, introduces the available datasets, and divides the approaches to FSL into two sub-sections: Meta-Learning-based FSL (Section 3.2 and Non-Meta-Learning-based FSL (Section 3.3). Next, we list the performance of various FSL methods and progress made so far in Section 4. Finally, we conclude with a discussion on challenges involved in Section 5.

# 2 Background

In this section, we discuss the necessary background required for understanding the recent few-shot learning algorithms.

## 2.1 Meta-Learning

Meta-Learning [7, 8] or Learning to Learn [9] has been the basic technique which most few-shot learning algorithms employ. Motivated by human development theory, meta-learning, a sub-field of machine learning, focuses on learning priors from previous experiences that can lead to efficient downstream learning of new tasks. For instance, a simple learner learns a single classification task, but a meta-learner gains an understanding of learning to solve a classification task by exposing itself to multiple similar classification tasks. Hence when presented with a similar but new task, the meta-learner could solve it quickly and better than a simple learner which has no prior experience in solving the task. A meta-learning procedure generally involves learning at two levels, within and across tasks. First, rapid learning occurs *within* a task, for example, learning to accurately classify within a particular dataset. Next, this learning is guided by knowledge accrued more gradually *across* tasks, which captures the way task structure varies across target domains.

Meta-Learning can be different from similar approaches such as transfer learning, multi-task learning, or ensemble learning. In Transfer Learning [10], a model is trained on a single task known as the source task in the source domain where the sufficient training data is available. This trained model is then again retrained or finetuned on another single task known as the target task in the target domain. The transfer of knowledge occurs from the source task to the target task. Thus more similar the two domains are the better it performs . Multi-Task Learning [11], involves learning

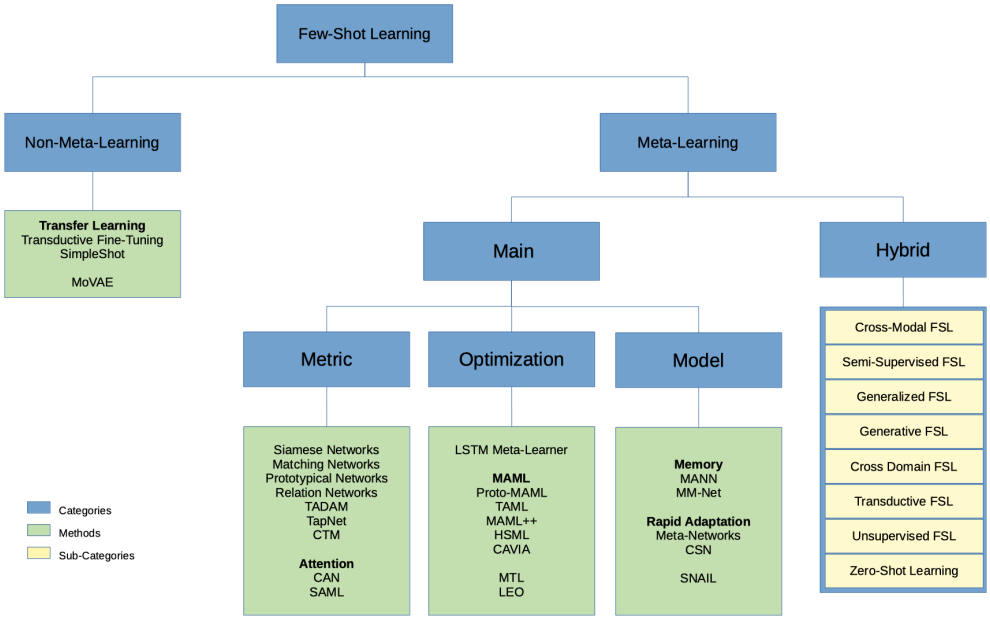


Figure 1: Approaches to FSL are categorized into Meta-Learning-based FSL and Non-Meta-Learning-based FSL. The three main meta-learning approaches are: metric-based, optimization-based and model-based meta-learning. Furthermore, variations of the FSL problem which use meta-learning are categorized as hybrid approaches.

multiple tasks simultaneously. It starts from no prior experience and attempts to optimize over solving multiple tasks at the same time. On the other hand, Ensemble Learning [12] is the process by which multiple models, such as classifiers or experts, are strategically generated and combined to solve a particular task. In contrast, meta-learner first gathers experience across multiple similar tasks and then use that experience to solve new tasks. Nonetheless, these techniques can be and often are meaningfully combined with meta-learning systems. We provide the formal definition for the meta-learning problem and explain it with the help of an example.

### 2.1.1 Problem Definition

In a typical supervised learning setting, we are interested in a task $\mathcal{T}$ with a dataset $\mathcal{D} = \{(x_k, y_k)\}_{k=1}^{n}$ with $n$ data samples. We usually split $\mathcal{D}$ into $\mathcal{D}^{train}$ and $\mathcal{D}^{test}$ such that:

$$\mathcal{D}^{train} = \{(x_k, y_k)\}_{k=1}^{t} \tag{1}$$

and

$$\mathcal{D}^{test} = \{(x_k, y_k)\}_{k=t+1}^{n}, \tag{2}$$

where $t$ denotes the number of training samples. We optimize parameters $\theta$ on the training set $\mathcal{D}^{train}$ and evaluate its generalization performance on the test set $\mathcal{D}^{test}$. Thus the learning problem here is to approximate the function $f$ with parameters $\theta$ as [2]:

$$y \approx f(x; \theta) \text{ where } (x, y) \in \mathcal{D}^{test} \tag{3}$$

and

$$\theta = \arg\min_{\theta} \sum_{(x,y) \in \mathcal{D}^{train}} \mathcal{L}(f(x, \theta), y) \tag{4}$$

[2] We omit sample subscript $k$ for simplicity in the following discussion

where $\mathcal{L}$ is any loss function measuring the error between the prediction $f(x, \theta)$ and the true label $y$.

In meta-learning, we have a distribution $p(\mathcal{T})$ of task $\mathcal{T}$. A meta-learner learns from a set of training tasks $\mathcal{T}_i \overset{train}{\sim} p(\mathcal{T})$ and is evaluated on a set of testing tasks $\mathcal{T}_i \overset{test}{\sim} p(\mathcal{T})$. Each of these task has its own dataset $\mathcal{D}_i$ where $\mathcal{D}_i = \{\mathcal{D}_i^{train}, \mathcal{D}_i^{test}\}$.

Let us denote the set of training tasks as $\mathcal{T}_{meta-train} = \{\mathcal{T}_1, \mathcal{T}_2, ....., \mathcal{T}_n\}$ and the set of testing tasks as $\mathcal{T}_{meta-test} = \{\mathcal{T}_{n+1}, \mathcal{T}_{n+2}, ....., \mathcal{T}_{n+k}\}$. Correspondingly, the training dataset for the meta-learner will be $\mathcal{D}_{meta-train} = \{\mathcal{D}_1, \mathcal{D}_2, ....., \mathcal{D}_n\}$ and the testing dataset will be $\mathcal{D}_{meta-test} = \{\mathcal{D}_{n+1}, \mathcal{D}_{n+2}, ....., \mathcal{D}_{n+k}\}$.

The parameters $\theta$ of the meta-learner are optimized on $D_{meta-train}$ and its generalization performance is tested on $D_{meta-test}$. Then, the meta-learning problem is to approximate the function $f$ with parameters $\theta$ as:

$$y \approx f(\mathcal{D}_i^{train}, x; \theta) \text{ where } (x, y) \in \mathcal{D}_i^{test} \tag{5}$$

and

$$\mathcal{D}_i = \{\mathcal{D}_i^{train}, \mathcal{D}_i^{test}\} \text{ where } \mathcal{D}_i \in \mathcal{D}_{meta-test}$$

i.e., $\mathcal{D}_i$ is the dataset for a test task $\mathcal{T}_i$ sampled from $\mathcal{T}_{meta-test}$. Then the optimal model parameters are obtained as:

$$\theta = \arg\min_{\theta} \sum_{\mathcal{D}_i \in \mathcal{D}_{meta-train}} \sum_{(x,y) \in \mathcal{D}_i^{test}} \mathcal{L}(f(\mathcal{D}_i^{train}, x; \theta), y). \tag{6}$$

That is the meta-learner learns parameters $\theta$ such that given a task $\mathcal{T}_i \sim p(\mathcal{T})$, its performance on its test data $\mathcal{D}_i^{test}$ given its training data $\mathcal{D}_i^{train}$ would be optimal. Figure 2 demonstrates a setup for example meta-learning problem.

#### 2.1.2 Nomenclature

In meta-learning and few-shot learning literature, certain notations and terms are used interchangeably. Table 1 lists these terms and their equivalent usage. Notation **A** is more commonly used in optimization-based meta-learning literature (Section 3.2.1.2) while notation **B** is used when discussing metric-based meta-learning methods (Section 3.2.1.1). Additionally, Table 2 lists the commonly used symbols in this survey.

| Notation A | Term A | Notation B | Term B |
|---|---|---|---|
| $\mathcal{D}_i^{train}$ | Training set for task $\mathcal{T}_i$ | $S_i$ | Support Set for task $\mathcal{T}_i$ |
| $\mathcal{D}_i^{test}$ | Test set for task $\mathcal{T}_i$ | $Q_i$ | Query Set for task $\mathcal{T}_i$ |
| $\mathcal{D}_{meta-train}$ | Meta-training set | $\mathcal{D}_{train}$ | Training Set |
| $\mathcal{D}_{meta-test}$ | Meta-testing set | $\mathcal{D}_{test}$ | Test Set |

Table 1: Nomenclature

## 3 Few-Shot Learning

Much of the recent progress in FSL has come through meta-learning. Therefore we first divide the approaches to FSL into two categories: meta-learning-based FSL and non-meta-learning-based FSL. Also, most of these approaches that we are about to discuss were developed with the perspective of solving the few-shot image classification problem. However, they are still applicable for solving other problems such as regression, object detection, segmentation, online recommendation, reinforcement learning, etc. We discuss the approaches to the few-shot image classification problem in this section and the applications to other domains in Section 4 .

### 3.1 The Few-Shot Classification Problem

Consider the task $\mathcal{T}$ (defined in Section 2.1.1) as a classification task where $x$ is input and $y$ is the output label. The objective is to approximate the function $f$ (Eqn. 3) with parameters $\theta$ (Eqn. 4) . This is generally possible when we have sufficient training data $\mathcal{D}^{train}$ (Eqn. 1) i.e., $t$ is a large number. However, when $t$ is small, it becomes difficult to approximate the function $f$ so that it has good generalization performance over $\mathcal{D}^{test}$ (Eqn. 2). This can be referred to as a *few-shot classification problem* as the number of examples (shots) are too few to learn a good model.

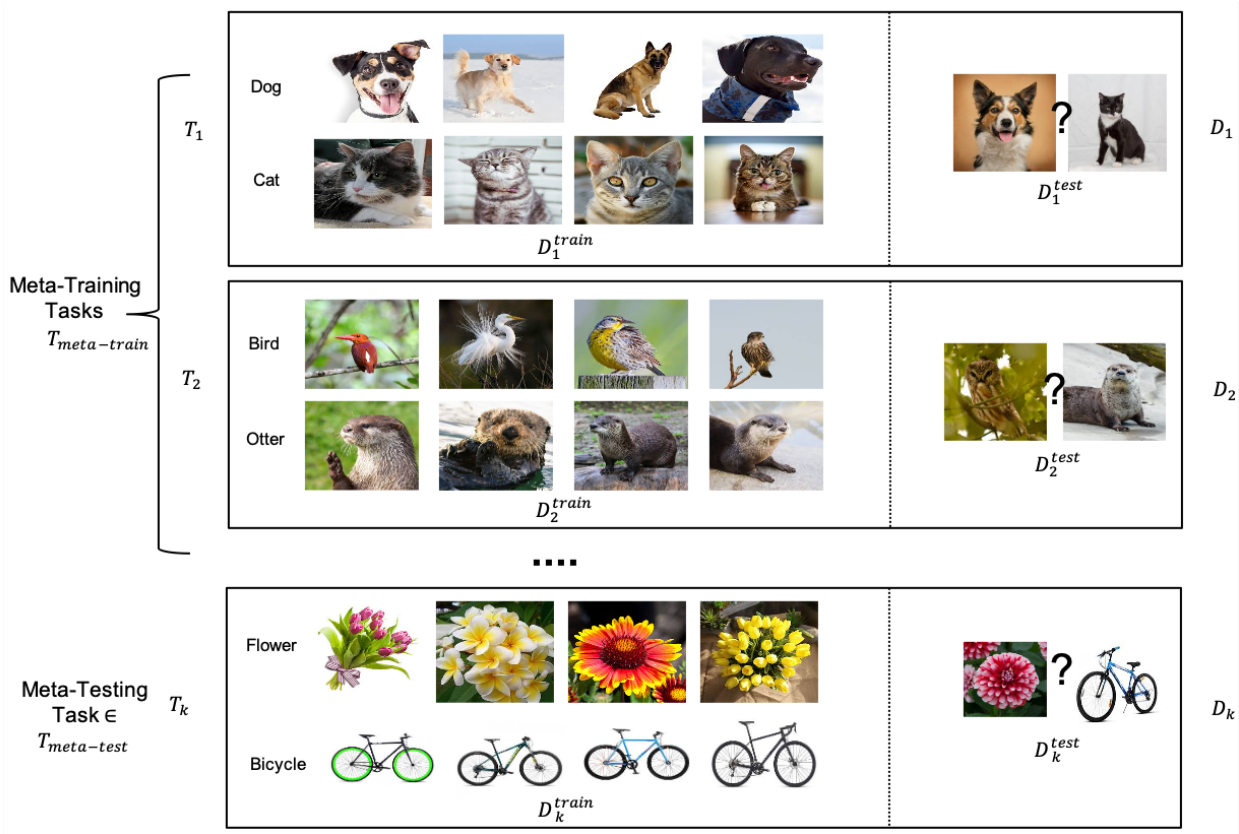


Figure 2: **Meta-Learning Example Setup**. Each task $\mathcal{T}_i$ is a binary classification task with a training set $\mathcal{D}_i^{train}$ and test set $\mathcal{D}_i^{test}$. During meta-training, the labels for samples in $\mathcal{D}_i^{test}$ is known and the goal of meta-learner is to find optimal $\theta$ as per equation 6. During meta-testing, new task with unseen categories is presented and the labels are predicted as per equation 5 .

| Symbol | Meaning | Context |
|---|---|---|
| $\mathcal{T}_i$ | Task $i$ | |
| $\mathcal{L}$ | Loss function | |
| $(x_k, y_k)$ | Input-Output pair | |
| $f_\theta$ | Model (function) with parameters $\theta$ | |
| $g_{\theta_1}$ | Embedding function | Sec. 3.2.1.1 |
| $d_{\theta_2}$ or $d$ | Distance function | Sec. 3.2.1.1 |
| $g_\phi$ | Meta-Learning model with parameters $\phi$ | Sec. 3.2.1.2 |
| $P_\theta(y\|x)$ | Output probability of $y$ for input $x$ using model parameters $\theta$ | |
| $k_\theta(x_1, x_2)$ | Kernel function measuring similarity between two vectors $x_1$ and $x_2$ | Table 4 |
| $\sigma$ | Softmax function | |
| $\alpha, \beta$ | Learning rates | |
| $w$ | Weights | |
| $\mathbf{v}_c$ | Prototype of class $c$ | Eq. 9 |
| $C$ | Set of classes present in S | |
| $S^c$ | Subset of S containing all elements $(x_k, y_k)$ such that $y_k = c$ | |
| $\oplus$ | Concatenation operator | Table 5 |
| $B$ | Number of batches $(X_b, Y_b)$ sampled in inner-loop for a randomly sampled task $\mathcal{T}_i$ | Table 6 |
| $I$ | Number of tasks $\mathcal{T}_i$ sampled in inner-loop | Table 6 |
| $J$ | Number of outer-loop iterations | Table 6 |

Table 2: Symbols

Usually people define few-shot classification task as a standard **M-way-K-shot** task [13, 14], where M is the number of classes and K is the number of examples per class present in $\mathcal{D}^{train}$. Usually K is a small number (ex., 1,5,10) and $|\mathcal{D}^{train}| = M \times K$. The performance is measured by a loss function $\mathcal{L}(\hat{y}, y)$ defined over the prediction $\hat{y} = f(x, \theta)$ and the ground truth $y$.

The **M-way-K-shot** tasks are usually sampled from a larger dataset with classes much higher in number than *M*. Table 3 lists such commonly used datasets for conducting experiments for few-shot classification.

| Dataset | Number of classes | Samples per class | Description |
|---|---|---|---|
| Omniglot [15] | 1623 | 20 | Handwritten characters from different languages. |
| *mini*ImageNet [13] | 100 | 600 | 100 classes randomly sampled from ImageNet. |
| FC100 [16] | 100 | 600 | Derived from CIFAR100 for FSL. |
| *tiered*ImageNet [17] | 608 | 1280 (avg.) | Like *mini*ImageNet but ensures that there is a wider degree of separation between training, validation and test classes. |

Table 3: Common datasets used for Few-Shot Learning

### 3.2 Meta-Learning-based Few-Shot Learning

The objective of meta-learning is to approximate the function $f$ with parameters $\theta$ such that the performance on any task $\mathcal{T}_i$ randomly sampled from the task distribution $p(\mathcal{T})$ is optimal (equation 5). We use this strategy for FSL such that the distribution $p(\mathcal{T})$ is now a distribution of few-shot tasks and each task $\mathcal{T}_i$ is a few-shot task. For example, consider the M-way-K-shot few-shot classification (FSC) task. During training, we meta-learn a prior $\theta$ over a distribution of M-way-K-shot FSC tasks so that at test time we can solve for a new M-way-K-shot FSC task.

Meta-Learning-based FSL can be classified into three approaches [18]: metric-based, optimization-based and model-based. Further, various meta-learning-based hybrid approaches were proposed to handle FSL problems such as cross-domain FSL, generalized FSL, etc. We discuss the three main approaches in Section 3.2.1 and hybrid approaches in Section 3.2.2.

#### 3.2.1 Main Approaches

Consider a task $\mathcal{T}$ with support set $S$ and query set $Q$. Let $f$ be a few-shot classification model with parameters $\theta$. Then for $(x, y) \in Q$, the meta-learning approaches to FSL can be differentiated in the way they model the output probability $P_\theta(y|\mathbf{x})$ [18] (Table 4).

| | Metric-based | Optimization-based | Model-based |
|---|---|---|---|
| **Key idea** | Metric Learning [19] | Gradient Descent | Memory; RNN |
| **How $P_\theta(y\|x)$ is modeled?** | $\sum_{(x_k, y_k) \in S} k_\theta(x, x_k) y_k,$ | $P_{\theta'}(y\|x),$ where $\theta' = g_\phi(\theta, S)$ | $f_\theta(x, S).$ |
| **Advantages** | Faster Inference.<br>Easy to deploy. | Offers flexibility to optimize in dynamic environments.<br>$S$ can be discarded post-optimization. | Faster inference with memory models.<br>Eliminates the need for defining a metric or optimizing at test. |
| **Disadvantages** | Less adaptive to optimization in dynamic environments.<br>Computational complexity grows linearly with size of $S$ at test. | Optimization at inference is undesirable for real-world deployment.<br>Prone to overfitting. | Less efficient to hold data in memory as $S$ grows.<br>Hard to design. |

Table 4: Meta-Learning Approaches

#### 3.2.1.1 Metric-based Meta-Learning

Metric Learning [19] is the task of learning a distance function over data samples. Consider two image-label pair $(x_1, y_1)$ and $(x_2, y_2)$ and a distance function $d$ to measure the distance between two images. If we were to assign a label to a query image $x_3$, we could compute the two distances $d(x_1, x_3)$ and $d(x_2, x_3))$ and assign the label corresponding to the image with a shorter distance, which is also the key idea in nearest neighbors algorithms (k-NN). However, with high dimensional inputs such as images, we typically use an embedding function $g$ to transform the input to a lower dimension before computing the distances:

$$g \colon \mathbb{R}^n \to \mathbb{R}^m \text{ where } n > m.$$

Therefore, the core idea behind metric-based few-shot learning is to leverage meta-learning architecture to either learn an embedding function $g(;\theta_1)$ (parameterized by a neural network with parameters $\theta_1$) given a distance function $d$ (such as euclidean distance) or to learn both the embedding function $g(;\theta_1)$ (with parameters $\theta_1$) and the distance function $d(;\theta_2)$ (usually parameterized by another neural network with parameters $\theta_2$). This is illustrated in Figure 3 .

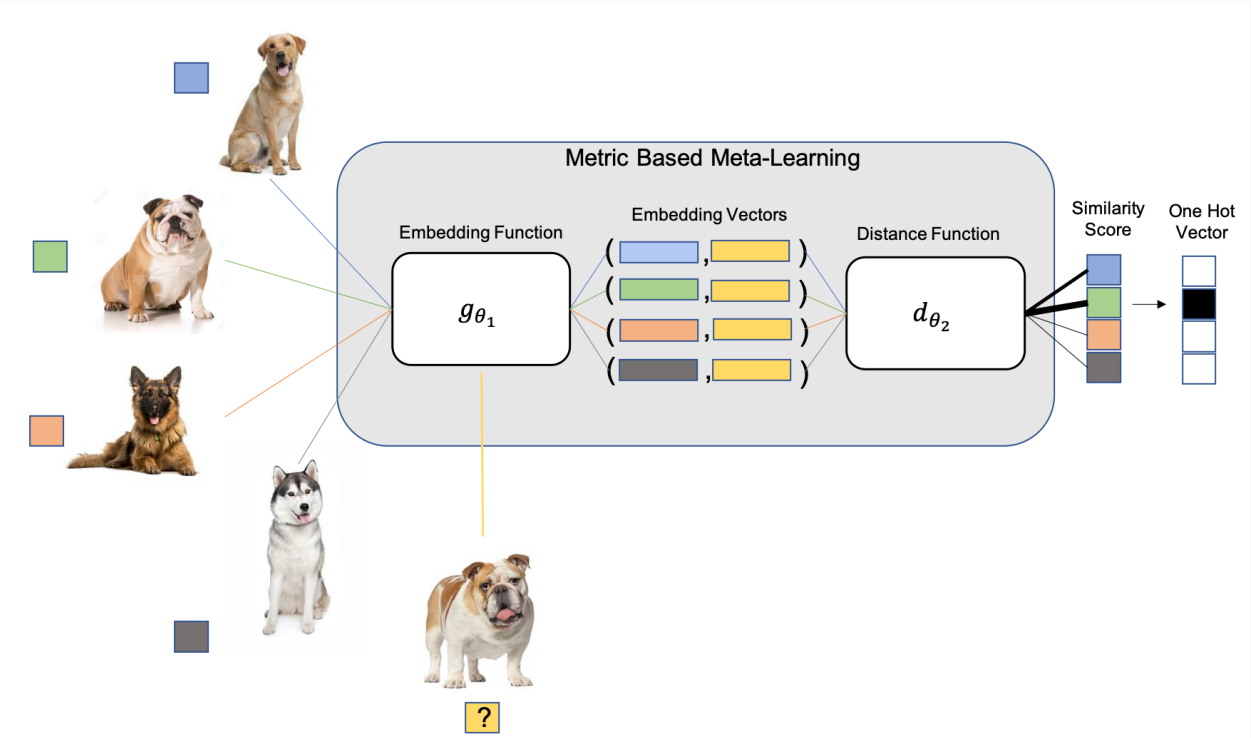


Figure 3: Example metric-based meta-learning setup for a 4-way-1-shot classification task. The embedding function $g_{\theta_1}$ outputs the embedding vectors for support images (labeled) and the query image (unlabeled, denoted by '?'). Distance function $d_{\theta_2}$ measures the distance between support and query vectors to output a similarity score.

Training proceeds by randomly sampling M-way-K-shot episodes from the training set. Each episode has a support set and a query set. The average error computed on query sets across multiple training few-shot episodes is used to update the parameters of the embedding function and the distance function (if any). Finally, new M-way-K-shot episodes are sampled from the testing set to evaluate the performance of the network. This episodic training paradigm is explained in Algorithm 1 .

Table 5 compares the recent metric-based meta-learning methods based on their characteristics like embedding function, distance measure, prediction method, loss function and if the embedding function is fixed for all tasks i.e., task-independent (**T.I**) or is adaptive (task-dependent). In the following paragraphs we discuss each of these methods in further detail.

**Algorithm 1:** Episodic Training in Metric-based Meta-Learning Methods

**Given:** In dataset $D$, $n$ is the number of examples, $N$ is the set of classes, $N_{train}$ is the set of classes used for training, $N_{test}$ is the set of classes used for testing, $M < N_{train}$ is the number of classes per episode, $K$ is the number of support examples per class, $Q$ is the number of query examples per class. $RandomSample(A, B)$ denotes a set of $B$ elements chosen uniformly at random from set $A$, $|N_{train}| + |N_{test}| = |N|$ and $N_{train} \cap N_{test} = \emptyset$.

**Input:** $D = \{(x_1, y_1), ..., (x_n, y_n)\}$ where $y_i \in \{1, ..., N\}$. $D^c$ denotes the subset of D containing all elements $(x_i, y_i)$ such that $y_i = c$.

**Training**: ▷ M-way K-shot training episodes

**while** *True* **do**

▷ 1. Constructing Task

$C \leftarrow RandomSample(N_{train}, M)$ ▷ Sample M classes
$S \leftarrow \{\}$ ▷ Support set
$Q \leftarrow \{\}$ ▷ Query set
**for** $c$ *in* $C$ **do**
$S^c \leftarrow RandomSample(D^c, K)$ ▷ Sample K support
$Q^c \leftarrow RandomSample(D^c \setminus S^c, Q)$ ▷ Sample Q query
$S \leftarrow S \cup S^c$
$Q \leftarrow Q \cup Q^c$
**end**

▷ 2. Learning Metric

**for** $i \leftarrow 1$ **to** $|S|$ **do**
$(x_s, y_s) \leftarrow S[i]$
**for** $j \leftarrow 1$ **to** $|Q|$ **do**
$(x_q, y_q) \leftarrow Q[j]$
$d_{ij} \leftarrow d_{\theta_2}(g_{\theta_1}(x_s), g_{\theta_1}(x_q)))$ ▷ Compute distances
**end**
**end**

Compute total loss $\mathcal{L}$ based on $d_{ij}$'s such that $d_{ij}$ is minimum when $y_i = y_j$ and maximum otherwise.

Update parameters $\theta_1$ and $\theta_2$ on $\mathcal{L}$.

**end**

**Testing**: Sample a random M-way K-shot episode but this time using the classes from $N_{test}$ and evaluate its performance.

| **Method** | **T.I** | $g_{\theta_1}$ | $d_{\theta_2}$ | **Prediction** | **Loss** |
|---|---|---|---|---|---|
| Siamese Networks [20] | Yes | CNN | L1 | $v = w \cdot d(g_{\theta_1}(x_1), g_{\theta_2}(x_2))$ $p = \text{sigmoid}(\sum_j v_j)$ | $-(y \log(p) + (1-y)(\log(1-p))$ |
| Matching Networks [13] | Yes | CNN + LSTM w/ attention | Cosine Similarity | $\hat{y} = \sum_{k=1}^{t} \sigma(d(f_\theta(\hat{x}), g_{\theta_1}(x_k))y_k$ $P(y=c\|\hat{x}) = \hat{y}_c$ | $-\log P$ |
| Prototypical Networks [21] | Yes | CNN | Euclidean | $P(y=c\|x) = \sigma(-d(g_{\theta_1}(x), \mathbf{v}_c))$ | $-\log P$ |
| Relation Networks [22] | Yes | CNN | Learned by CNN | $r_c = d_{\theta_2}(g_{\theta_1}(x) \oplus \mathbf{v_c}))$ | $\sum_{c \in C} (r_c - \mathbf{1}(y == c))^2$ |
| TADAM [16] | No | ResNet-12 | Cosine / Euclidean | $P_\lambda(y=c\|x) = \sigma(-\lambda d(g_{\theta_1}(x, \Gamma), \mathbf{v}_c))$ | $-\log P$ |
| TapNet [23] | No | Resnet-12 | Euclidean | $P(y=c\|x) = \sigma(-d(\mathbf{M}(g_{\theta_1}(x)), \mathbf{M}(\Phi_c)))$ | $-\log P$ |
| CTM [24] | No | Any | Any | - | - |

Table 5: Metric-based Meta-Learning Methods

**Convolutional Siamese Networks**
Siamese Neural Networks [25] are a pair of identical neural networks with shared weights originally proposed for signature verification. The two networks are joined at their output, where the joining neurons measures the distance between the feature vector output of each network. In 2015, Koch et al. [20] used a pair of identical convolutional neural networks (CNNs) [26] with shared weights as done in Siamese Networks for image verification. The network was trained to recognize if the two images belong to the same class or not. The network outputs a probability score (similarity) of the two images belonging to the same class. This idea was further extended to perform one-shot recognition by comparing the similarity score between a query image and support images from different classes. This is illustrated in Figure 4 . Here the embedding function $g_{\theta_1}$ is a CNN and the distance between two embedding vectors is simply the L1 distance i.e., $|g_{\theta_1}(x_1) - g_{\theta_1}(x_2)|$. The distance is converted to the probability of similarity by a linear feed-forward layer and a sigmoid function. The network is then trained on binary cross-entropy loss. Similar to Convolutional Siamese Networks, Mehrotra et al. [27] proposed Skip Residual Pairwise Networks (SRPNs) where they used a Wide Residual Network [28] as the embedding function $g_{\theta_1}$ and a network of residual blocks to act as the distance function $d_{\theta_2}$.

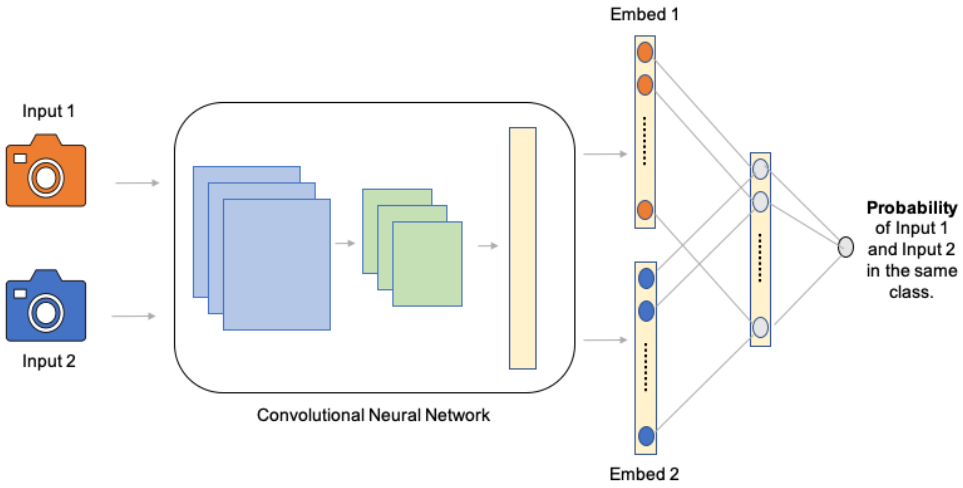


Figure 4: Convolutional Siamese Network

**Matching Networks**
Given a support set $S = \{x_i, y_i\}_{k=1}^{K}$ and a query $\hat{x}$, Matching Networks [13] (Figure 5) define a probability distribution over the output labels $y$ using an attention kernel $a(\hat{x}, x_k)$ . The attention kernel basically computes the cosine similarity between the embeddings of support and query examples and then normalize the similarity score by taking a softmax:

$$a(\hat{x}, x_k) = e^{\cos(f(\hat{x}), g(x_k))} / \sum_{k=1}^{t} e^{\cos(f(\hat{x}), g(x_k))}. \tag{7}$$

The classifier's output is then defined as a sum of the labels (one-hot encoded) of support samples weighted by the attention kernel $a(\hat{x}, x_k)$ :

$$P(y|\hat{x}, S) = \sum_{k=1}^{K} a(\hat{x}, x_k) y_k \tag{8}$$

In a simple case, the embedding function for query ($f_\theta$) and the embedding function for examples in the support set ($g_{\theta_1}$) are same i.e., $f = g$. Alternatively, Matching Networks propose using Full Context Embeddings where embedding functions contextually embeds images i.e for a given support image $x$ and support set $S$, the embedding of $x$ is obtained by $g_{\theta_1}$ in the presence of $S$ as $g_{\theta_1}(x; S)$ . Similarly for a query $\hat{x}$, its embedding would be $f_\theta(\hat{x}; S)$. i.e., $S$ should be able to modify how we embed $\hat{x}$ through $f$. Using contextual embedding showed in improvement in the performance of few-shot classification on miniImageNet but no difference was observed on the simpler omniglot dataset.

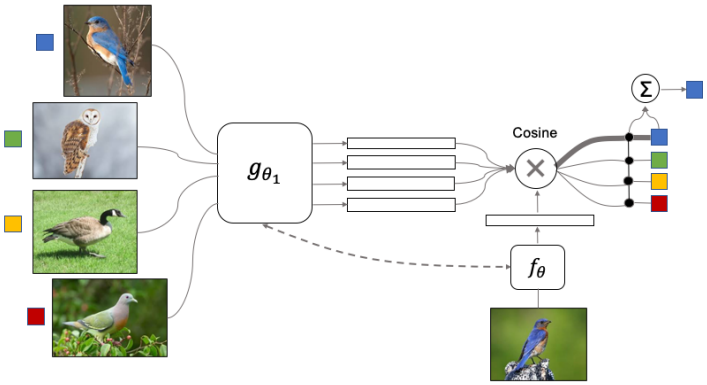


Figure 5: Matching Networks (Figure adapted from [13]).

**Prototypical Networks**
Prototypical Networks [21] use a 4-layer-CNN as an embedding function $g_{\theta_1}$. A prototype for each class is defined by taking an average of embedding vectors obtained from the support images belonging to that class (Eqn. 9):

$$\mathbf{v}_c = \frac{1}{|S^c|} \sum_{(x_k, y_k) \in S^c} g_{\theta_1}(x_k). \tag{9}$$

The similarity is measured by calculating the squared euclidean distance between the query's embedding and each class prototype. The output probability over classes is calculated by taking a softmax over the negative distances (Eqn. 10):

$$P(y = c|\hat{x}) = \text{softmax}(-d(g_{\theta_1}(\hat{x}), \mathbf{v}_c)) = \frac{e^{(-d(g_{\theta_1}(\hat{x}), \mathbf{v}_c))}}{\sum_{\hat{c} \in C} e^{(-d(g_{\theta_1}(\hat{x}), \mathbf{v}_{\hat{c}}))}}. \tag{10}$$

The loss $\mathcal{L}$ is given by the negative log-likelihood of the correct class (Eqn. 11):

$$\mathcal{L}(\theta_1) = -\log P_{\theta_1}(y = c|\hat{x}). \tag{11}$$

Furthermore, to generate more discriminative embeddings, Zhang et al. [29] proposed using a multi-way contrastive loss function with margin to pull the examples belonging to the same class together and push the others away in the embedding space.

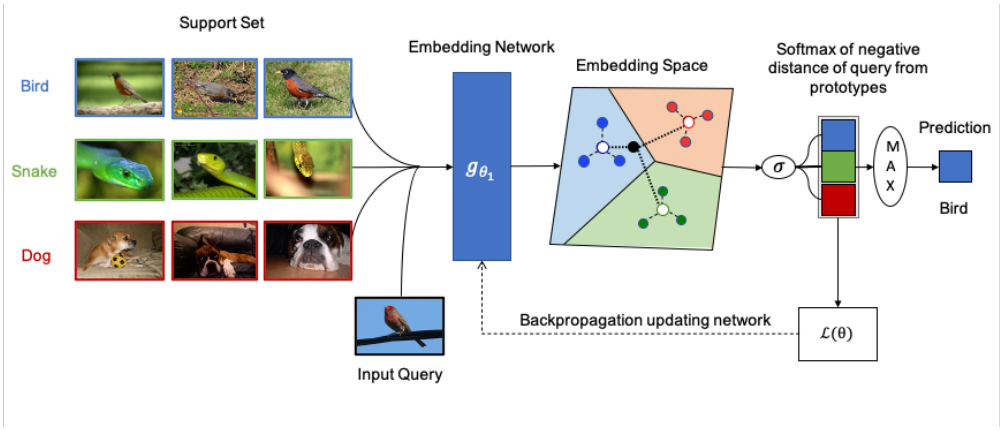


Figure 6: Few-shot prototypes $\mathbf{v_c}$ are computed as the mean of embedded support examples for each class. The embedded query points are classified via a softmax over distances to the class prototypes.

**Relation Networks**
Relation Networks [22] get rid of manually constructing a similarity metric and instead just employs another CNN to output a similarity score. The representations of the support and query examples (obtained from $g$) are concatenated together and fed into $d$ to obtain a relation score between each query and a support class (Figure 3). When the number of support examples per class are more than one, the relation score is calculated between a query and a prototype of a class $\mathbf{v_c}$, which is an element wise sum of embeddings of support examples of that class. The relation score for a match should be 1 and 0 for a no-match. The network is then trained to minimize the mean squared error of relation scores.

*Relation score of query $\hat{x}$ with class $c$ :*

$$r_c = d_{\theta_2}(g_{\theta_1}(\hat{x}) \oplus \mathbf{v_c})$$

*Optimization Objective:*

$$\theta_1, \theta_2 \leftarrow \arg\min_{\theta_1,\theta_2} \sum_{c \in C} (r_c - \mathbf{1}(y == c))^2$$

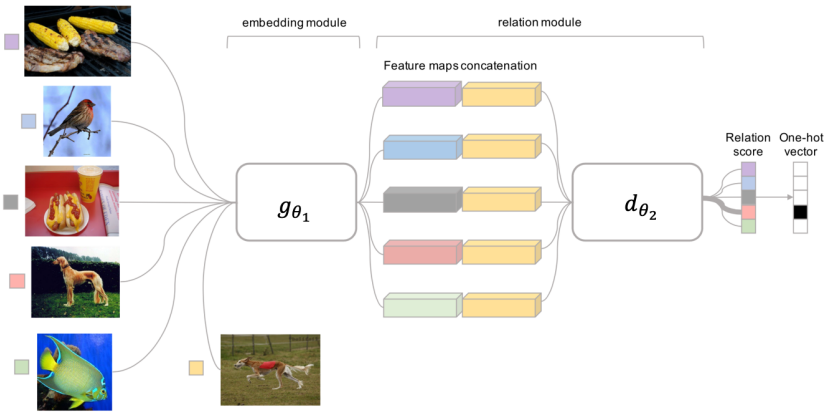


Figure 7: Relation Network (Source: [22])

**Task-Dependent Adaptive Metric (TADAM)**
In the approaches discussed previously, the embedding function $g_{\theta_1}$ was *task-independent*, meaning given any task $\mathcal{T} \sim p(\mathcal{T})$, its samples will be embedded using a fixed embedding function $g_{\theta_1}$. Moreover, the choice of distance metric function (i.e. cosine or euclidean) was also something to be experimented with depending upon the task at hand. Oreshkin et al. [16] proposed using **1)** a learnable softmax temperature [30] with a scaling factor $\alpha$ to bridge the gap between the performance of cosine and euclidean distance metric and **2)** a Task Embedding Network (TEN) factored into the embedding network $g_{\theta_1}$ to output task adaptive representations. These contributions combined with using a deeper embedding network (Resnet-12 [1]) as the feature extractor resulted in 8.5% absolute accuracy improvement over Prototypical Networks [21] on the *mini*ImageNet [13] 5-way 5-shot classification task.

1. **Metric Scaling:** It was observed that Prototypical Networks [21] which used euclidean distance performed better on few-shot image classification when compared to Matching Networks [13] which used cosine distance. Oreshkin et al. [16] suggested that the difference in performance could be directly attributed to the interaction of the different scaling of the metrics with the softmax. Hence they propose to scale the distance metric by a learnable temperature, $\lambda$, and observed that both distances produced equivalent performance when scaled with learned parameter $\lambda$ (Eqn. 12):

$$P_\lambda(y = c|x) = \text{softmax}(-\lambda d(g_{\theta_1}(x), \mathbf{v}_c)). \tag{12}$$

2. **Task Conditioning:** Previously, Matching Networks used contextual embeddings i.e, embedding for an input image $x$ was obtained in presence of its support set $S$, $g_{\theta_1}(x; S)$. This was achieved with a bidirectional LSTM as a post-processing of a fixed feature extractor. Differently, TADAM explicitly define a dynamic feature extractor $g_{\theta_1}(x, \Gamma)$ where $\Gamma$ is the set of parameters predicted from a task representation such that the performance of $g_{\theta_1}(x, \Gamma)$ is optimized given the task sample set $S$.

**Task-Adaptive Projection (TapNet)**
In addition to the embedding function $g_{\theta_1}$ and the distance function $d$, TapNet [23] proposed a concept of per-class reference vectors $\mathbf{\Phi}$ and a task dependent projection space or mapping $\mathbf{M}$. Unlike class prototypes in Prototypical Networks, reference vectors $\Phi$ for each class are learned. The projection space $\mathbf{M}$ is non-parametric and is built specific to each task. The input query $x$ is then classified by measuring its distance to different reference vectors $\Phi$ in the projection space $\mathbf{M}$ (Eqn. 13):

$$P(y = c|x) = \text{softmax}(-d(\mathbf{M}(g_{\theta_1}(x)), \mathbf{M}(\Phi_c))). \tag{13}$$

The motivation is to find a projection space $\mathbf{M}$ which could remove the misalignment between the task-embedded features and the references and thus resulting in better classification performance.

**Task-Relevant Features (CTM)**
Similar to Matching Networks, Li et al. [24] propose using a plugable component called Category Traversal Module (CTM) parameterized by $\phi$ to find contextual embeddings of the images in the support set $S$ and the query set $Q$. The parameters $\phi$ are learned during the training along with parameters $\theta_1$ of the embedding function $g$.

$$\text{CTM}(g_{\theta_1}(S); \phi) \rightarrow I(S)$$

$$\text{CTM}(g_{\theta_1}(Q); \phi) \rightarrow I(Q)$$

The CTM takes support set features $g_{\theta_1}(S)$ as an input and produces a mask $p$ via a concentrator and projector that make use of intra and inter-class views respectively. The mask $p$ is applied to reduced-dimension features of both the support and query, producing improved features $I$ with dimensions relevant for the current task. These improved feature embeddings are finally fed into a metric learner. This is illustrated in Figure 8.

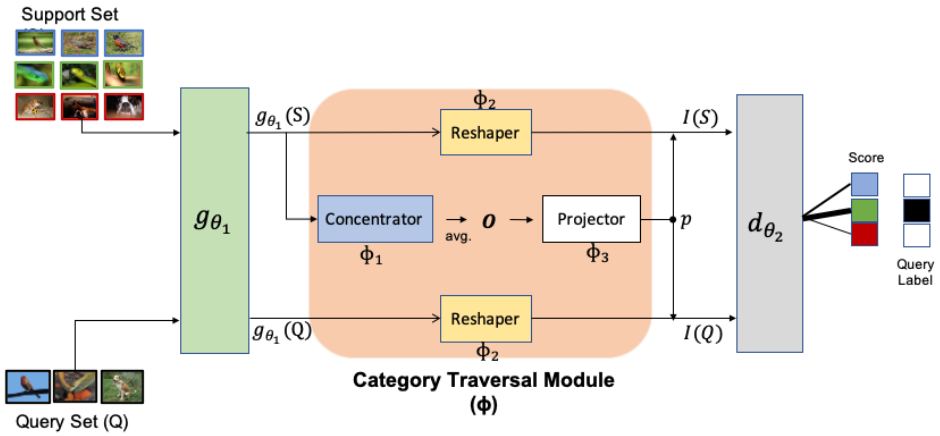


Figure 8: Category Traversal Module (CTM) (Figure adapted from [24])

#### 3.2.1.1.1 Attention-based methods

Much like Matching Networks and CTM, few others have proposed integrating attention [31] modules in existing methods for learning more discriminative feature embeddings. Hou et al. [32] proposed Cross Attention Networks (CAN) with a Cross Attention Module (CAM) to transform the class prototypes $P$ and query $Q$ to more discriminative prototypes $\bar{P}$ and query $\bar{Q}$. Furthermore, Hao et al. [33] propose Semantic Alignment Metric Learning (SAML) method to align semantically relevant local regions on support and query images using attention maps.

#### 3.2.1.2 Optimization-based Meta-Learning

Earlier in Section 3.1, we discussed that for a few-shot classification task $\mathcal{T}$ with training data $\mathcal{D}^{train}$ (Eqn. 1) where the number of training examples $t$ is small, it is difficult to approximate $f$ (Eqn. 3) with parameters $\theta$ (Eqn. 4) from scratch using gradient-based optimization as its not designed to cope with small number of training samples and thus will lead to overfitting. This ponders the question, "is there any way to optimize on limited training data and still achieve good generalization performance?" Optimization-based meta-learning for FSL answers to this question. Basically, leveraging the meta-learning architecture (Figure 2) and episodic training (Algorithm 1), optimization-based methods enables an optimization procedure to work on limited training examples.

**Learner and Meta-Learner**

Optimization-based methods generally involves learning in two stages:

1. **Learner:** A learner model $f_\theta$ is task-specific and trained for a given task. For a given few-shot task, a stand-alone learner model trained from scratch using gradient descent (Eqn. 4) will not be able to generalize (Eqn. 3).

2. **Meta-Learner:** A meta-learner model $g_\phi$ is not task specific and is trained on a distribution of tasks $\mathcal{T} \sim p(\mathcal{T})$ (Figure 2). Using episodic training, the meta-learner learns ($\phi$) to update the learner model's parameters ($\theta$) via training set $\mathcal{D}^{train}$,

$$\theta^* = g_\phi(\theta, \mathcal{D}^{train}). \tag{14}$$

   The objective of the meta-learner model is to produce updated learner model parameters $\theta^*$ such that they are better than stand-alone learner model parameters $\theta$.

During meta-training (notation **A** in Table 1), the optimization process involves updating $\phi$ for the meta-learner and $\theta$ for individual training tasks. Once the meta-training finishes, the prior knowledge is encompassed into $\phi$ and only $\theta$ is updated for a test task (Eqn. 14).

Table 6 compares different optimization-based meta-learning methods based on how they update learner's parameters $\theta$ and meta-learner parameters $\phi$. In all the listed methods, learning happens in two-stages. Initially, in the outer-loop, the meta-learner's parameters $\phi$ are randomly initialized. Next, in the inner-loop the learner parameters ($\theta$) are updated/proposed by meta-learner (Eqn. 14). The learners' training loss $\mathcal{L}^{train}$ is further used to obtain optimal parameters $\theta^*$. Finally, in the outer loop the cumulative test loss of learner obtained using $\theta^*$ is used for updating $\phi$. In some cases learner's initial parameters $\theta$ are also meta-learned along with $\phi$.

In the following paragraphs, we discuss the the recent optimization based meta-learning approaches in more detail.

**LSTM Meta-Learner**

Normally when given a task $\mathcal{T}$, we try to learn function $f(\theta)$ on its training data $\mathcal{D}^{train}$. The parameters $\theta$ of the neural network $f$ are updated using some form of gradient descent as such:

$$\theta_{i+1} = \theta_i - \alpha \nabla f(\theta_i). \tag{15}$$

Instead of using an hand designed optimizer such as SGD and a fixed learning rate $\alpha$, Andrychowicz et al. [34] learn an optimizer function $g_\phi$ such that:

$$\theta_{i+1} = \theta_i + g_i(\nabla f(\theta_i); \phi). \tag{16}$$

Similarly, Ravi & Larochelle [35] modeled an LSTM [36] as a meta-learner $g_\phi$ to propose parameters for the learner $f$:

$$\theta_{i+1} = g_i(\nabla f(\theta_i), \theta_i; \phi). \tag{17}$$

Optimizer $g$ is trained to produce parameters $\theta$ in just few steps for the few-shot learning task $\mathcal{T}$. It follows the episodic training paradigm and mimics the testing scenario. This training procedure is described in Algorithm 2 and depicted in Figure 9.

| Method | Learner | Meta-Learner |
|---|---|---|
| LSTM Meta-Learner [35] | Repeat $\forall b \in [1..B]$<br>$\mathcal{L}_b \leftarrow \mathcal{L}(f(X_b; \theta_{b-1}), Y_b)$<br>$\theta_b \leftarrow g((\nabla_{\theta_{b-1}} \mathcal{L}_b, \mathcal{L}_b); \phi_{j-1})$ | Repeat $\forall j \in [1..J]$<br>$\mathcal{L}_j^{test} \leftarrow \mathcal{L}(f(X; \theta_B), Y)$<br>$\phi_j \leftarrow \phi_{j-1} - \alpha \nabla_{\phi_{j-1}} \mathcal{L}_j^{test}$ |
| MAML [14] | Repeat $\forall i \in [1..I]$<br>$\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{train}; \theta_{j-1}))$<br>$\theta_i^* \leftarrow \theta_{j-1} - \alpha \nabla_{\theta_{j-1}} \mathcal{L}_t^{train}$<br>$\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{test}; \theta_t^*))$ | Repeat $\forall j \in [1..J]$<br>$\theta_j \leftarrow \theta_{j-1} - \beta \nabla_{\theta_{j-1}} \sum_{i=1}^{I} \mathcal{L}_i^{test}$ |
| MTL [37] | $\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{train}; [\theta_{j-1}, \phi_{j-1}, \Theta]))$<br>$\theta_i^* \leftarrow \theta_{j-1} - \alpha \nabla_{\theta_{j-1}} \mathcal{L}_i^{train}$<br>$\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{test}; \theta_i^*))$ | $\theta_j \leftarrow \theta_{j-1} - \beta \nabla_{\theta_{j-1}} \sum_{i=1}^{I} \mathcal{L}_i^{test}$<br>$\phi_j \leftarrow \phi_{j-1} - \beta \nabla_{\phi_{j-1}} \sum_{i=1}^{I} \mathcal{L}_i^{test}$ |
| LEO [38] | $\phi_{j-1} = \{\phi_e, \phi_r, \phi_d, \alpha\}$<br>$\mathbf{z_i} \leftarrow g(\mathcal{D}_i^{train}; [\phi_e, \phi_r, \Theta])$<br>$\theta_i \leftarrow g(\mathbf{z_i}; \phi_d)$<br>$\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{train}; \theta_i))$<br>$\mathbf{z_i^*} \leftarrow \mathbf{z_i} - \alpha \nabla_{\mathbf{z_i}} \mathcal{L}_i^{train}$<br>$\theta_i^* \leftarrow g(\mathbf{z_i^*}; \phi_d)$<br>$\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{test}; \theta_i^*))$ | $\phi_j \leftarrow \phi_{j-1} - \beta \nabla_{\phi_{j-1}} \sum_{i=1}^{I} \mathcal{L}_i^{test}$ |

Table 6: Optimization-based Meta-Learning Methods

**Algorithm 2:** Train Meta-Learner

**Input:** Meta-training set $D_{meta-train}$, Learner $f$ with parameters $\theta$, Meta-Learner $g$ with parameters $\phi$

$\phi_0 \leftarrow$ random initialization
**for** $j \leftarrow 1$ **to** $J$ **do**
  $D^{train}, D^{test} \leftarrow$ random dataset from $D_{meta-train}$
  $\theta_0 \leftarrow c_0$
  **for** $b \leftarrow 1$ **to** $B$ **do**
    $\mathbf{X_b}, \mathbf{Y_b} \leftarrow$ random batch from $D^{train}$
    $\mathcal{L}_b \leftarrow \mathcal{L}(f(\mathbf{X_b}; \theta_{b-1}), \mathbf{Y_b})$ ▷ Get loss of learner on train batch
    $c_b \leftarrow g((\nabla_{\theta_{b-1}} \mathcal{L}_b, \mathcal{L}_b); \phi_{j-1})$ ▷ Get output of meta-learner
    $\theta_b \leftarrow c_b$ ▷ Update learner parameters
  **end**
  $\mathbf{X}, \mathbf{Y} \leftarrow D^{test}$
  $\mathcal{L}_{test} \leftarrow \mathcal{L}(f(\mathbf{X}; \theta_b), \mathbf{Y})$ ▷ Get loss of learner on test batch
  Update $\phi_j$ using $\nabla_{\phi_{j-1}} \mathcal{L}_{test}$ ▷ Update meta-learner parameters
**end**

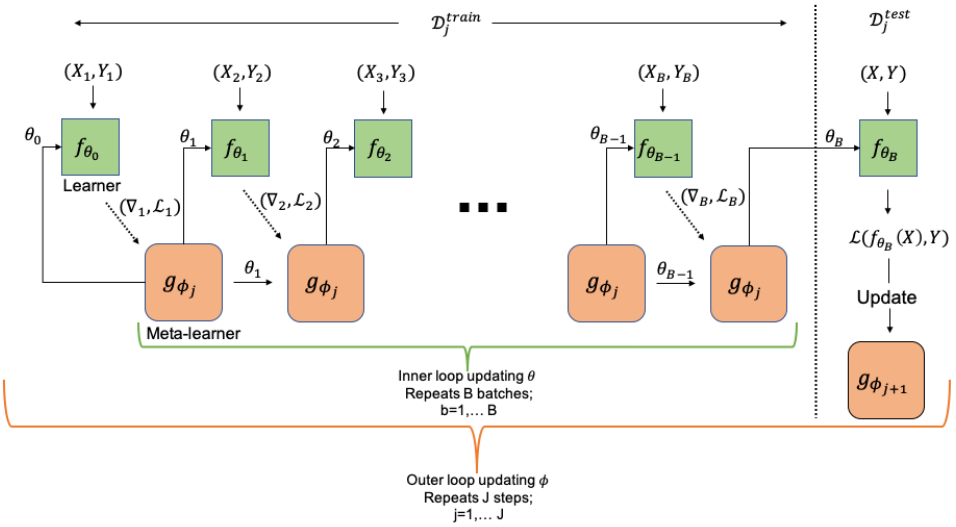


Figure 9: Computational graph for the forward pass of the meta-learner (Figure adapted from [35]).

**Model-Agnostic Meta-Learning (MAML)**
They key idea in MAML or Model-Agnostic Meta-Learning [14] is to achieve good initialization parameters $\theta$ such that new tasks can optimize quickly from $\theta$ through one or more gradient descent steps computed with a small amount of data (Figure 10). These initial parameters $\theta$ are meta-learned over a distribution of tasks $p(\mathcal{T})$. Unlike LSTM meta-learner which has two separate models, a meta-learner model $g_\phi$ and a task learner model $f_\theta$, MAML has a single model with parameters $\theta$. Individual tasks use the model parameters $\theta$ as initialization to arrive at optimal parameters $\theta^*$ for the task at hand (Algorithm 3).

*Proto-MAML* [39] extends MAML by combining ideas from Prototypical Networks [21] and MAML [14]. During training, the former focuses on learning a good embedding function $g_\theta$ and does not perform any task adaptation whereas later approach focuses on learning a good initialization parameters $\theta$ which are then fine-tuned to adapt to each task. In Proto-MAML, the initial weights of the classifier are obtained from prototypical networks which are then subsequently fine-tuned to the individual task.

*Task Agnostic Meta-Learning (TAML)* [40] argues that the initial model of the meta-learner could be too biased toward existing tasks to adapt to new tasks, especially in the case of few-shot learning with discrepancy between new tasks from those in the training tasks. In such case, we wish to avoid an initial model over-performing on some tasks. Therefore,

**Algorithm 3:** MAML

**Require:** $p(\mathcal{T})$ : distribution over tasks
**Require:** $\alpha, \beta$ : step size hyperparameters
$\theta_0 \leftarrow$ random initialization
**for** $j \leftarrow 1$ **to** $J$ **do**
  Sample a batch of $I$ tasks randomly from $p(\mathcal{T})$
  **for** $i \leftarrow 1$ **to** $I$ **do**
    $D_i^{train}, D_i^{test} \leftarrow$ dataset for task $\mathcal{T}_i$ from $D_{meta-train}$
    $\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(D_i^{train}; \theta_{j-1}))$ ▷ Get loss of learner on training data
    $\theta_i^* \leftarrow \theta_{j-1} - \alpha \nabla_{\theta_{j-1}} \mathcal{L}_i^{train}$ ▷ Adapt to learner's training loss
    $\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(D_i^{test}; \theta_i^*))$ ▷ Get test loss of learner on adapted parameters
  **end**
  $\theta_j \leftarrow \theta_{j-1} - \beta \nabla_{\theta_{j-1}} \sum_{i=1}^{I} \mathcal{L}_i^{test}$ ▷ Update parameters on total test loss of learners
**end**

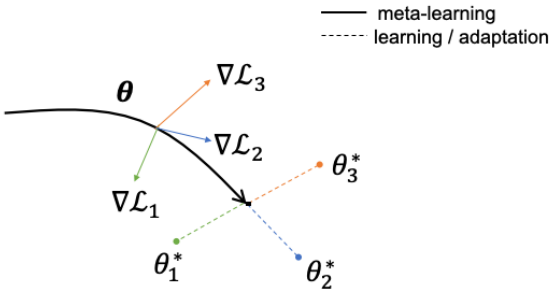


Figure 10: Diagram of MAML, which optimizes for a representation $\theta$ that can quickly adapt to new tasks (Figure adapted from [14]).

TAML aims to meta-train an unbiased initial model by preventing it from over-performing on some tasks or directly minimizing the inequality of performance across different tasks, in hope to make it more generalizable to unseen tasks.

Antoniou et al. [41] argue that even though MAML is a simple yet elegant meta-learning framework, it suffers from a variety of problems which can cause 1) instability during training, 2) restricted generalization performance, 3) reduction in the framework's flexibility, 4) increase in the system's computational overhead and 5) a costly hyperparameter tuning before it can work robustly on a new task. Hence they proposed *MAML++* [41], an improved meta-learning framework that offers the flexibility of MAML along with stability, computational efficiency and improved generalization performance.

*Hierarchically Structured Meta-Learning (HSML)* [42] attempts to address the challenge of task uncertainty and heterogeneity in meta-learning, which can not be handled by globally sharing knowledge among tasks or to learn a single initialization for all kinds of tasks as done in MAML. Therefore, HSML explicitly tailor transferable knowledge to different clusters of tasks by learning task representations where each cluster of tasks has its own initial parameters (Figure 11).

*Fast **C**ontext **A**daptation **Via** Meta-Learning* (CAVIA) [43] is a simple extension to MAML that is more interpretable and less prone to overfitting. CAVIA partitions the model parameters $\theta$ into two parts: context parameters $\theta_{context}$ that serve as additional input to the model and are adapted on individual tasks, and shared parameters $\theta_{shared}$ that are meta-trained and shared across the tasks. At test time, only context parameters are updated, leading to a low dimensional task representation.

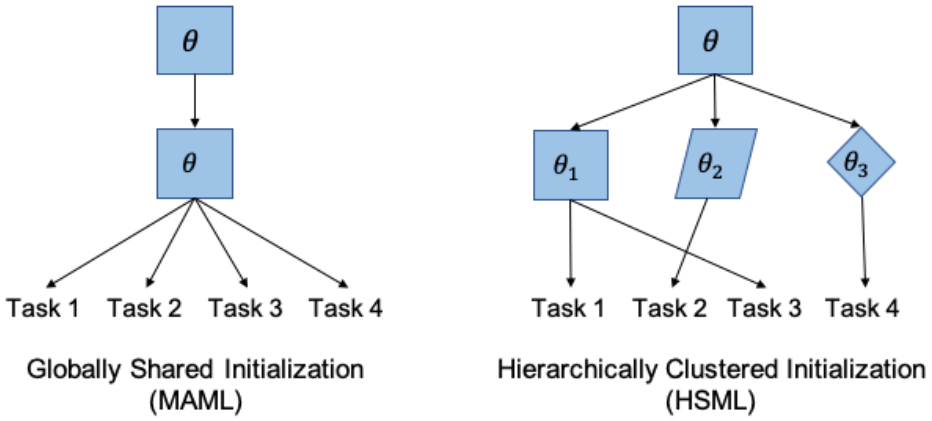


Figure 11: MAML vs HSML (Figure adapted from [42])

**Meta-Transfer Learning (MTL)**
In MAML [14], the meta-learned model parameters $\theta$ are adapted as $\theta^*$ to an individual task. The effectiveness of this strategy is limited to shallow networks, as the adaptation in deep networks can lead to overfitting. Therefore, Sun et al. [37] propose a technique called Meta-Transfer Learning (MTL) (Figure 12). The key idea is to use a pretrained DNN as a feature extractor ($\Theta$) and meta-learn only the last layer classifier parameters $\theta$. In addition, MTL also meta-learns scale ($\phi_{S_1}$) and shift ($\phi_{S_2}$) parameters to adapt the $\Theta$ to individual tasks. The scale and shift parameters are few in number when compared to $\Theta$.

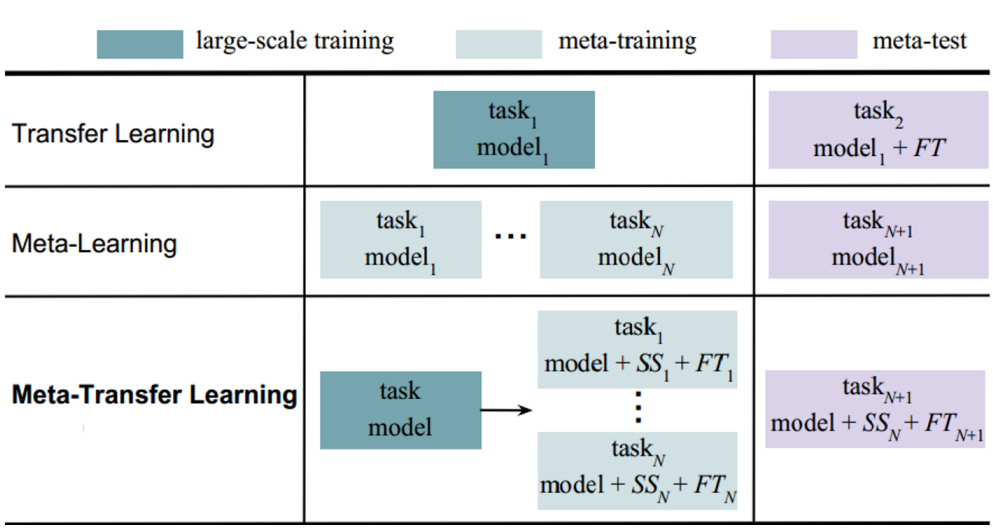


Figure 12: Difference between learning strategies. Meta-Task Learning (MTL) adopts transfer learning strategy for Meta-Learning. A pre-trained model's weights are shift and scaled on task basis. The shift and scale (SS) parameters are learned through meta-learning across tasks (Source: [22]).

**Latent Embedding Optimization**
Latent Embedding Optimization(LEO) [38] learns a low-dimensional latent embedding of model parameters and performs optimization-based meta-learning in this space. Learning low-dimensional latent representation is motivated by the fact that it is difficult to optimize in high-dimensional spaces in extreme low-data regimes. The following steps explain the optimization process of LEO:

1. **Pretraining**: Like MTL, input feature embeddings are not learned, but are pretrained on a deep network (WRN [28]). Using the meta-training dataset, a ResNet classifier is trained to distinguish between training classes. Features from an intermediate layer of this trained classifier is used for getting input embeddings.
2. **Inner Loop Training**: Each task's dataset $\mathcal{D}^{train}$ is used to obtain the initial parameters $\theta$ of the classifier $f$.
   - The pre-trained embeddings for examples $\mathcal{D}^{train}$ are fed into a Encoder-Relation network which outputs latent representation $\mathbf{z}$ for each class (like a prototype).
   - The low dimensional latent representation generates task classifier parameters $\theta$ through a decoder.
   - The training loss $\mathcal{L}^{train}$ of $f_\theta$ is used to update the latent representations $\mathbf{z}$.
3. **Outer Loop Training**: The loss $\mathcal{L}^{test}$ calculated on each task's test examples $\mathcal{D}^{test}$ with $f_\theta$ is used to update the parameters of encoder, relation network and decoder.

Intuitively, this provides two advantages. First, the initial parameters for a new task are conditioned on the training data, which enables a task-specific starting point for adaptation. Second, by optimizing in the lower-dimensional latent space, the approach can adapt the behavior of the model more effectively (Figure 13).

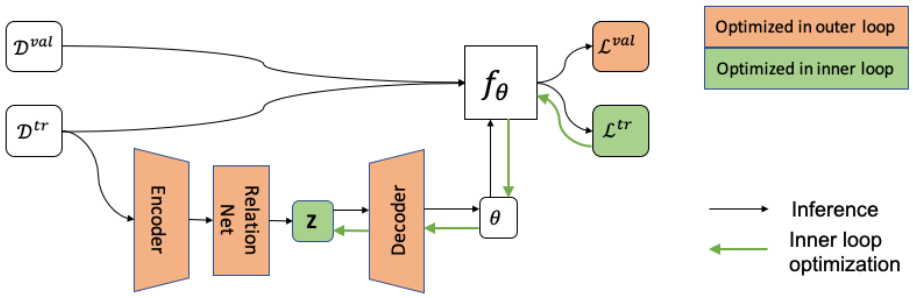


Figure 13: Overview of the architecture of LEO (Source: [38])

#### 3.2.1.3 Model-based Meta-Learning

Metric-based methods for FSL learn a function $g_\theta$ to generate discriminative embeddings based on a metric where as optimization-based methods learn a priors $\phi, \theta$ to optimize quickly from. Different from these approaches, model-based meta-learning methods makes no assumption on the form of $P_\theta(y|x)$. Rather it involves model architectures specifically tailored for fast learning. Table 7 summarizes the methods in this category. Based on the kind of model architecture ($f_\theta$), these methods are further categorized into memory-based, rapid-adaptation-based and miscellaneous models.

| Method | Type | Key Idea |
|---|---|---|
| MANN [44] | Memory | Using NTM [45] for sequence prediction |
| MM-Net [46] | Memory | Key-Value Memory Networks [47] combined with Matching Networks [13] |
| MetaNets [48] | Rapid Adaptation | Fast-Weights stored in memory |
| CSN [49] | Rapid Adaptation | Task specific conditioning of activation values stored in memory |
| SNAIL [50] | Miscellaneous | Temporal Convolutions and Causal Attention layers for sequence prediction |

Table 7: Model-based Meta-Learning Methods

##### 3.2.1.3.1 Memory as a component

A family of model architectures integrates an external memory component to facilitate their learning process. This external memory component is usually a 2D matrix called the memory bank, memory matrix or just plain *memory*. The memory acts as a storage buffer to which neural networks can write new information and retrieve previously stored information. Note that this memory component is different than *internal memory* found in vanilla RNNs or LSTMs. Neural Turing Machines (NTM) [45] and Memory Networks [51], [52, 47] are the examples of two such model architectures which incorporates external memory in their learning process. In the context of FSL, memory as an external component can relieve the burden of training in low data regime and can allow for faster generalization. Next, we discuss such model architectures which integrates memory into their design and use meta-learning for learning from few-examples.

**Memory Augmented Neural Networks**

Memory Augmented Neural Networks (MANN) [44] use a modified NTM to rapidly assimilate new data into memory and leverage this data to make accurate predictions after only a few samples.

- **Neural Turing Machine**
  A Neural Turing Machine (NTM) couples a controller neural network with external memory storage (Figure 14). Like most neural networks, the controller interacts with the external world via input and output vectors. Unlike standard networks, the controller also learns to read and write memory rows by soft attention, while the memory serves as a knowledge repository. The attention weights are generated by its addressing mechanism.

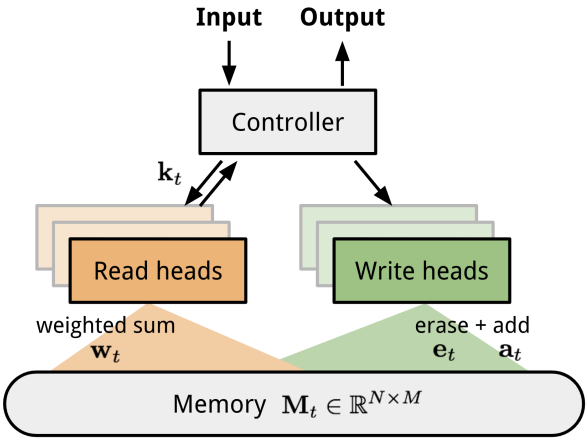


Figure 14: The architecture of NTM. The memory at time t, $M_t$ is a matrix of size $N \times M$, containing N vector rows and each has M dimensions (Source: [6]).

- **Addressing Mechanism in MANN**
  The controllers in MANN are either LSTMS or feed-forward networks. Given some input $x_t$ at time $t$, the

controller produces a key feature vector $k_t$, which is then either stored in a row of a memory matrix $M_t$, or used to retrieve a particular memory, i, from a row; i.e. $M_t(i)$. A memory, $r_t$, is retrieved using the weighting vector $w_t^r(i)$ as:

$$r_t \leftarrow \sum_i w_t^r(i) M_t(i) \text{ where } w_t^r(i) = \text{softmax}(\frac{k_t \cdot M_t(i)}{||k_t|| \cdot ||M_t(i)||}).$$

Writing to memory in MANN model involves the use of Least Recently Used Access (LRUA) module. The LRUA module is a pure content-based memory writer that writes memories to either the least used memory location or the most recently used memory location.

- **Meta-Learning Setup for MANN**
  Memory encoding and retrieval in a NTM external memory is rapid, with vector representations being replaced into or taken out of memory potentially every time-step. This ability makes the NTM a perfect candidate for meta-learning and low-shot prediction, as it is capable of both long-term storage via slow update of weights, and short term storage via its external memory module. Training in MANN follows the same episodic paradigm discussed earlier, except the truth label $y_t$ is presented with one step offset i.e $\{(x_t, y_{t-1}), (x_{t+1}, y_t), ...\}$. The network is tasked to output the appropriate label for $x_t (i.e., y_t)$ at the given timestep. This prevents the network from slowly learning sample-class bindings in its weights. Instead, it must learn to hold data samples in memory until the appropriate labels are presented at the next time-step, after which sample-class information can be bound and stored for later use (Figure 15).

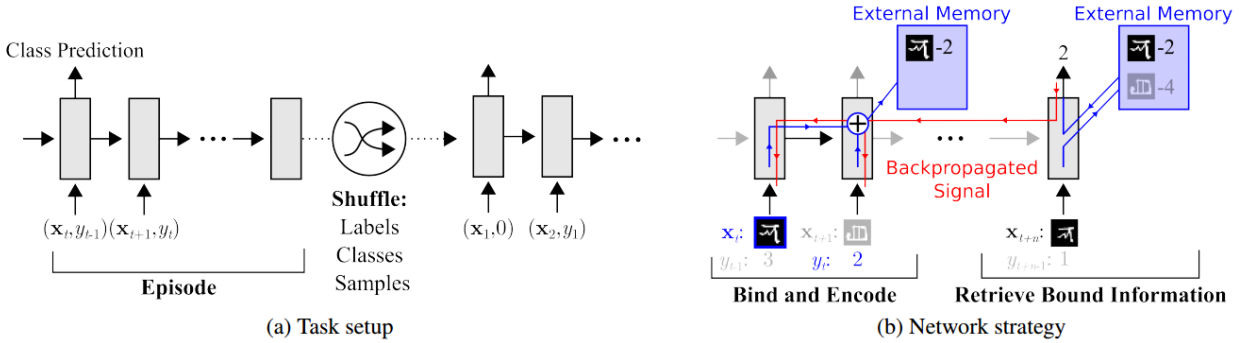


Figure 15: Task Structure. (a) Images $x_t$ are presented with time-offset labels, (b) External memory stored the bounded sample representation-class label information (Source: [44]).

**Memory Matching Networks**
Memory Matching Networks (MM-Net) [46] integrates the memory module from Key-Value Memory Networks [47] into Matching Networks [13]. It extends the idea of metric-based meta-learning with a memory module to encode and generalize the whole support set into memory slots. Given the support set $S$, the memory module encodes the sequence of $N$ support images into $M$ memory slots with the *write controller*. For each support image $x$ and its embedded representation $z$ in memory key space, the *read controller* measures the dot product similarity between the input support image and the memory slots to retrieve a conditioned representation $g(x|M)$. Meanwhile, a contextual learner is devised to predict the parameters of CNNs for embedding unlabeled image in the query set.

#### 3.2.1.3.2 Rapid Adaptation

The following model-based approaches uses techniques like "fast-weights" to rapidly adapt the parameters of a model for a given task. Normally weights in the neural networks are updated by stochastic gradient descent in an objective function and this process is known to be slow. One faster way to learn is to utilize one neural network to predict the parameters of another neural network and the generated weights are called fast weights. In comparison, the ordinary SGD-based weights are named slow weights.

**Meta Networks**
Meta Networks (MetaNet) [48] is a meta-learning model with architecture and training process designed for *rapid* generalization across tasks. It consists of two main learning components, a base learner and a meta-learner, and is equipped with an external memory. The rapid generalization of MetaNet relies on fast-weights. The external memory is used to store these fast weights and the input representations.

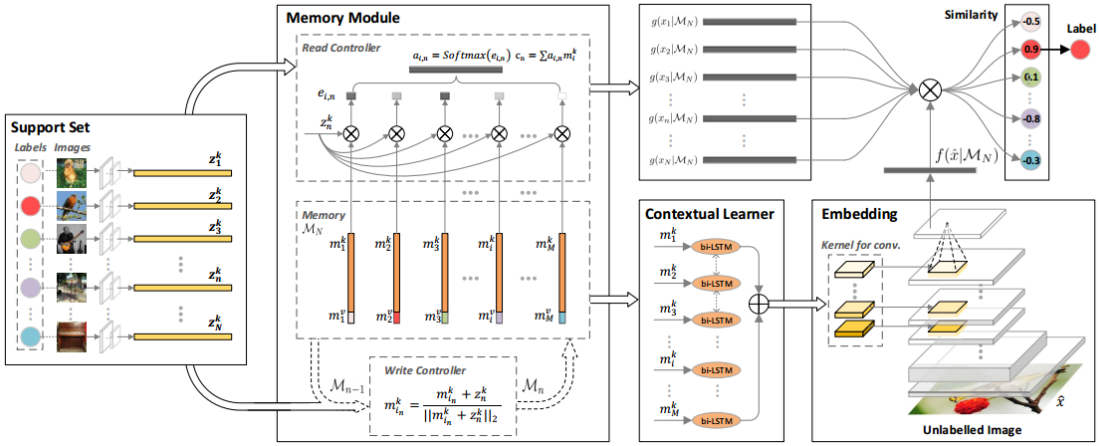


Figure 16: Architecture of Memory Matching Networks (Source: [46])

In MetaNet, loss gradients are used as meta information to populate models that learn fast weights. Slow and fast weights are combined to make predictions in neural networks.

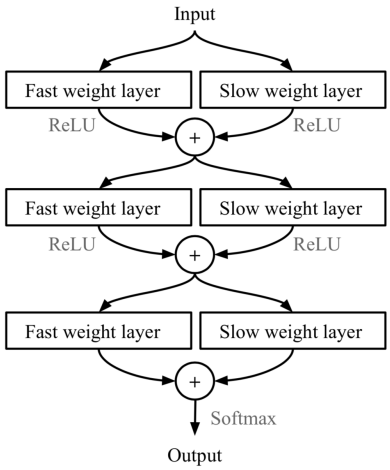


Figure 17: Combining slow and fast weights in a MLP. $\bigoplus$ is element-wise sum (Source: [48]).

**Conditionally Shifted Neurons**

Conditionally Shifted Neurons (CSNs) [49] modify their activation values with task-specific shifts retrieved from a memory module, which is populated rapidly based on a limited task experience. Building upon Meta-Networks, CSNs also have a base learner, a meta-learner and a memory module. The learning happens in the following manner:

- A base learner works on individual tasks. Using its current weights it makes predictions on the examples in the support set (description phase).
- The loss incurred from each prediction from support set is stored in the form of conditioning information $I$ in the key-value memory where keys are the embeddings of inputs and the values are the conditioning information.
- To classify a query (prediction phase), its embeddings are compared with the embeddings of the keys in the memory using cosine similarity and the similarity score is then weighted using a softmax. The conditioning information for each key is weighted by its similarity score and summed together to obtain joint conditioning information.

- The base network is updated with this joint conditioning information and the prediction for the query is made with these updated weights.
- The loss obtained on the query is then used to update the key embeddings network $f$, the value network $g$ and the prediction network (the initial base network).

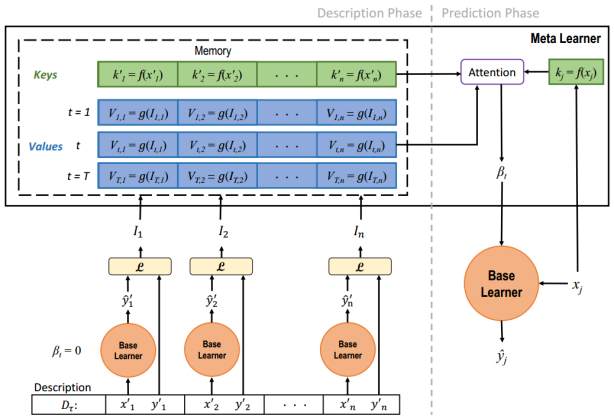


Figure 18: Rapid Adaptation with Conditionally Shifted Neurons. In the description phase, the meta learner populates working memory with keys and values, based on the base learner's performance on the task description; in the prediction phase, the meta learner retrieves task-specific shifts from memory through key-based attention and feeds them to the base learner to adapt it to the task (Source: [49]).

#### 3.2.1.3.3 Miscellaneous Models: SNAIL

Each episode in SNAIL [50] receives as input a sequence of example-label pairs $(x_1, y_1), ..., (x_{t-1}, y_{t-1})$ for timesteps 1, ..., $t-1$, followed by an unlabeled example $(x_t, \_)$ (Figure 19). It is then tasked to output prediction for $x_t$ based on the previous labeled examples it has seen. Formalizing meta-learning as a sequence-to-sequence problem, it argues that meta-learner should be able to internalize and refer to past experience. It proposes use of embedding networks with interleaved temporal convolutions and causal attention layers; the former to aggregate information from past experience and the latter to pinpoint specific pieces of information. The temporal convolution layers in SNAIL provides high-bandwidth access over its past experience without constraints on the amount of experience it can effectively use. SNAIL architectures are easier to train than traditional RNNs such as LSTM or GRUs and can be efficiently implemented so that an entire sequence can be processed in a single forward pass.

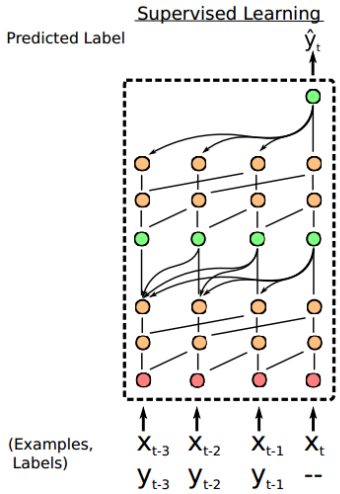


Figure 19: Overview of SNAIL; in this example, two blocks of temporal convolution layers (orange) are interleaved with two causal attention layers (green) (Source: [50]).

### 3.2.2 Hybrid Approaches

This section discusses the variations of the few-shot learning problem and the hybrid meta-learning based approach towards them. Table 8 list the hybrid approaches and summarizes the key idea behind them.

| Approach | Key Idea |
|---|---|
| Cross-Modal FSL | Leverage semantic data from a **different modality**. |
| Semi-Supervised FSL | Using few training examples, label the given **unlabeled** examples and improve the few-shot classifier. |
| Generalized FSL | Classify query into **either** one of the meta-training classes or a class from the support set. |
| Generative FSL | Learning to **generate** more samples from the given few. |
| Cross Domain FSL | Training in **one domain** and testing **in another**. |
| Transductive FSL | **Jointly predict** all the query examples. |
| Unsupervised FSL | Support examples are **unlabeled**. |
| Zero-Shot Learning | **No support examples** present. |

Table 8: Hybrid Approaches

**Cross-Modal Few-Shot Learning**
The recent progress in few-shot image classification has primarily been made in the context of unimodal learning. In order to alleviate the problem of limited data in image domain, some approaches [53, 54] employ data from different modality (eg., text). This is referred to as Cross-Modal Few-Shot Learning. For example, Xing et al. [54] propose an Adaptive Modality Mixture Mechanism (AM3) that combines information from an image and its label's word embedding to develop a better prototype of its class.

**Semi-Supervised Few-Shot Learning**
Semi-Supervised FSL considers the scenario when there is limited *labeled* data but sufficient *unlabeled* data is available during the training. For example in approaches which use meta-learning for few-shot classification, a weakly supervised classifier trained on support set (labeled) learns to label unlabeled data and then use it to improve its performance on the classification task. Using metric-based meta-learning approach, Ren et al. [17] propose three semi-supervised variants of Prototypical Networks [21], basically using Soft k-Means method to tune clustering centers with unlabeled data. Alternatively, Sun et al. [55] use an optimization-based meta-learning approach for learning to initialize a classification model for semi-supervised FSL.

**Generalized Few-Shot Learning**
In general, FSL approaches involve meta-training on base (seen) classes and meta-testing on novel (unseen) classes. Given a novel task sampled from the meta-testing set, the few-shot classifier will classify a query into one of the classes present in the novel task's support set (task's training set) and will not be able to recognize if the query example is from a base class. Generalized Few-Shot Learning (GFSL) focuses on the *joint* classification of both the base classes and novel classes. In particular, the goal is, for the model trained on the seen categories to be capable of incorporating the limited unseen class instances and make predictions for test/query instances in the both set of classes. Recent work of Gidaris & Komodakis [56], Ye et al. [57] and Ren et al. [58] attempts to address this problem.

**Generative Few-Shot Learning**
One usual way one may think to alleviate the problem of learning from low data samples is to augment them with synthesized samples. To this end, Wang et al. [59] propose a meta-learning approach for generating samples from limited data. This generative model is referred to as *hallucinator*, a model that maps real examples to hallucinated examples. The few-shot training set is first fed to the hallucinator, and it produces an expanded training set, which is then used by the learner. We refer to approaches of this kind as Generative Few-Shot Learning.

**Cross Domain Few-Shot Learning**
The FSL approaches discussed so far in the context of few-shot classification aimed to recognize novel categories with only few labeled examples in each class. One assumption that was followed was that all few-shot tasks belong to the same distribution or domain. For example, most approaches sampled tasks from *mini*ImageNet during training and as well as testing. While promising results were observed following this assumption, the existing methods often fail to generalize to unseen domains due to large discrepancy of the feature distribution across domains. This problem of few-shot learning under domain shifts is referred to as Cross Domain FSL. To this end, the early works of Tseng et al. [60] attempt to address this problem by simulating various feature distributions under different domains in the training stage.

**Transductive Few-Shot Learning**
Although, meta-learning is an effective strategy for few-shot learning as it aims at generalizing to unseen classification tasks, the fundamental difficulty with learning with scarce data remains for a novel classification task. One way to achieve larger improvements with limited amount of training data is to consider relationships between instances in the test set and thus predicting them as whole, which is referred to as transduction or transductive inference [61]. Therefore, transductive few-shot learning techniques [32, 62, 63] utilize the information present in unlabeled examples in the query set as whole to make prediction about individual queries. For example Liu et al. [62] proposed Transductive Propagation Networks, where the examples in the support and query sets are modeled as nodes of a graph. The labels of the support set nodes are known and the task is to predict the labels of the query set nodes which is achieved using their label propagation algorithm.

**Unsupervised Few-Shot Learning**
In supervised few-shot learning, the labels of the examples in the support set are available during the training and the label of the examples in the query set have to be estimated. In contrast, in unsupervised few-shot learning, the examples in the support set are also unlabeled. Huang et al. [64] proposed a strategy for unsupervised few-shot classification task which involves first performing clustering on the examples in the support set and then assigning the query to one of the clusters.

**Zero-Shot Learning**
Zero-Shot Learning (ZSL) [65] attempts to solve a task without the presence of any training examples for that task. For an image classification task, ZSL methods rely mostly on visual-auxiliary modality alignment. Often times, the auxiliary data is image's label i.e., samples for the same class from two modalities are mapped together so that two modalities obtain the same semantic structure. Because ZSL does not have access to any visual information when learning new concepts, ZSL models have no choice but to align the two modalities. This way, during test the image query can be directly compared to auxiliary information for performing classification [66].

### 3.3 Non-Meta-Learning based Few-Shot Learning

In this section, we discuss strategies other than meta-learning that can aid learning in limited data regime.

#### 3.3.1 Transfer Learning

Transfer Learning [67] is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. In few-shot learning scenario where the data is too limited to train a deep network from scratch, transferring knowledge from another network can be a viable option. For a classification task, this knowledge transfer is achieved by pretraining a deep network on large amounts of training data on base classes (seen) and then fine-tuning it on a new few-shot classes (unseen). However, naive fine-tuning using just few example can lead to overfitting and thus poor generalization performance on the few-shot task. Therefore, in this section, we discuss approaches which attempts to address this problem.

**Distance Metric Classification Using Embeddings from a Pretrained Network**
In section 3.2.1.1, we discussed approaches that used meta-learning to extract feature embeddings and perform classification using a nearest neighbour classifier with a distance metric. SimpleShot [68] instead uses a pretrained deep network to get feature embeddings for the input and query images, perform centering & L2 normalization on the obtained features and use euclidean distance as the distance measure for nearest neighbour classification. This simple approach has shown considerable improvement in accuracy in comparison to meta-learning approaches. Similarly chen et al. [69], shows comparable results from using cosine metric for nearest neighbour classification on embeddings obtained from a network trained on base classes.

**Training a New Classifier Using Embeddings from a Pretrained Network**
Training a classifier from scratch isn't possible when the number of training samples are limited, due to the poor resulting representations. However one could still obtain the representations from a pre-trained network and then train a new classifier using them. Tian et al. [70], demonstrate this exactly. Along with using representations from a pre-trained network, they also L2 normalize them before training a new classifier for each few-shot task. They also show that this approach outperforms the simple nearest neighbour classification using pretrained embeddings.

**Transductive Inference Using Embeddings from a Pretrained Network**
Certain approaches attempt to exploit the structure of information present in the query set and collectively classify the examples in the query set. This is know as transductive inference. For example, in section 3.2.2, we mentioned Transductive Propagation Networks [62] which utilized meta-learning to transductively assign labels to the examples in the query set. Alternatively, instead of using meta-learning, Dhillon et al. [71] choose to transductively fine-tune a

pretrained network on a given few-shot task. That means along with the support examples (labeled), query samples are also utilized in the fine-tuning process. The proposed transductive fine-tuning phase solves for:

$$\theta^* = \arg\min_{\theta} \frac{1}{|S|} \sum_{(x,y)\in S} -\log p_\theta(y|x) + \frac{1}{|Q|} \sum_{(x,y)\in Q} \mathbb{H}(p_\theta(.|x)). \tag{18}$$

The first term in the equation is the data fitting term using labeled support samples whereas the second term, the regularizer, uses the unlabeled query samples to minimize the entropy of predictions.

Similarly, Ziko et al. [72], propose a transductive laplacian-regularized inference for few-shot tasks. Using the feature embeddings learned from the base classes (pretrained), they minimize a quadratic binary-assignment function containing two terms:

$$\mathcal{E}(\mathbf{Y}) = \mathcal{N}(\mathbf{Y}) + \frac{\lambda}{2}\mathcal{L}(\mathbf{Y}), \tag{19}$$

where

$$\mathcal{N}(\mathbf{Y}) = \sum_{q=1}^{N}\sum_{c=1}^{C} y_{q,c} d(x_q - \mathbf{m}_c)$$

and

$$\mathcal{L}(\mathbf{Y}) = \frac{1}{2}\sum_{q,p} w(x_q, x_p)||\mathbf{y}_q - \mathbf{y}_p||^2$$

- $\mathcal{N}(\mathbf{Y})$, a unary term, is minimized globally when each query point is assigned to the class of the nearest prototype $\mathbf{m}_c$ (obtained from the support set) using a distance metric $d(x_q, m_c)$.
- $\mathcal{L}(\mathbf{Y})$, a pairwise Laplacian term, encourages nearby points $(x_p, x_q)$ in the label space to the same latent label assignment ($w$ is any similarity metric).

These transfer learning based methods have often come to exhibit better or equivalent performance on FSL tasks when compared to the complex meta-learning methods discussed earlier.

#### 3.3.2 Miscellaneous: Autoencoders

Mocanu [73] proposes a one-shot learning method, dubbed MoVAE (Mixture of Variational Autoencoders), to perform classification. A Variational Autoencoder (VAE) [74] provides a probabilistic manner for describing an observation in latent space. Thus, rather than building an encoder which outputs a single value to describe each latent state attribute, a VAE encoder to describe a probability distribution for each latent attribute. Given C classes, C VAEs are trained, one for each class. Then the reconstruction loss from the VAEs are measured to perform classification on the unlabeled data sample. A downside of this approach is that requires constructing and training new Autoencoders for each new task even at test. In contrast, most meta-learning methods work out-of-the-box at test time.

## 4 Progress in Few-Shot Learning

Early few-shot research focused on computer vision applications and mainly image classification [13, 21, 16, 14]. This is because visual information is easy to acquire and has been extensively examined in machine learning. Other computer vision problems such as Object Detection [75, 76, 77, 78] and Segmentation [79] have also received recent attention from the few-shot learning community. Apart from computer vision applications, FSL has been used for fault diagnosis [80], text classification [81, 82], image colorization [83] and cold-start item recommendation [84, 85]. In graph modeling, FSL has been used for node classification [86], edge labelling [87] and relation classification [88]. In audio, its used for few-shot speaker recognition [89, 90] and sound recognition [91]. Finally, imitation learning [92] in robotics and control [14] in reinforcement learning [93].

Since its emergence in 2016, the field of few-shot learning has shown promising improvement in learning from limited data. Figure 20 shows the trend of improvement in accuracy for 5-way 1-shot classification task on *mini*ImageNet [13]. Beginning with Matching Networks [13] at 43% accuracy, various optimization, metric, model-based and hybrid approaches were proposed in past four years, pushing the accuracy to 80% (as of Jan 2020). Although model-based approaches have seen less progress, there is no clear consensus if any one particular approach is the best way forward. Table 9 lists the accuracy of methods discussed in this survey (sorted by 1-shot accuracy and color coded by type). Metric-based, Optimization-based, Hybrid Meta-Learning and even Non-Meta-Learning approaches, all seem to be leading the race.

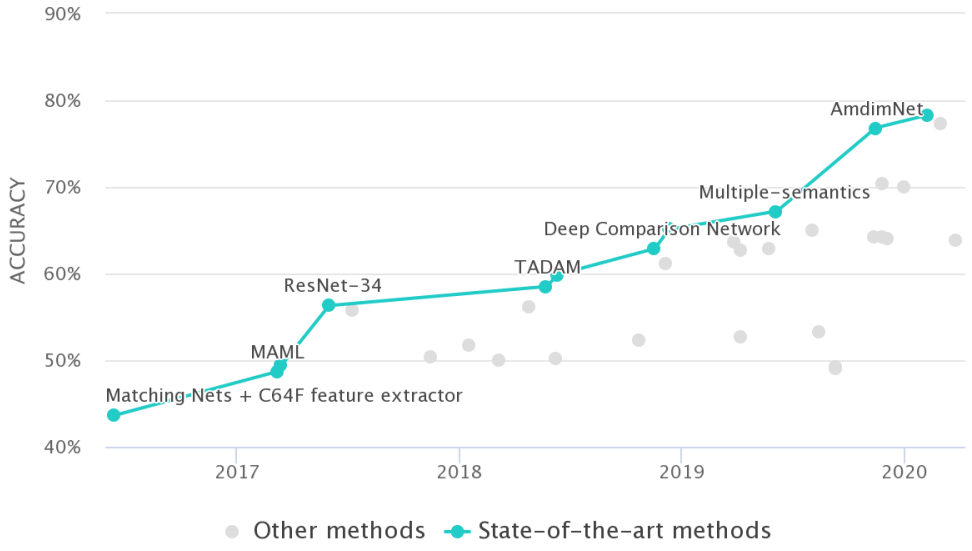


Figure 20: Progress in FSL[3]

## 5 Challenges and Open Problems

Recent years witnessed a great deal of interest from the machine learning community in solving the problem of learning from few examples. Consequently, many approaches were proposed, and each of them involved some kind of knowledge transfer, either though meta-learning or transfer learning. The success of these approaches relied on certain assumptions, which might become challenging to uphold in real-world settings. Therefore, in this section we discuss the challenges involved in the nitty-gritty of FSL approaches.

**Training the Same Way as Testing** Most meta-learning methods employ M-way K-shot episodic training paradigm (Algorithm 1), i.e, we train the few-shot learning model to distinguish between M classes each with exactly K training examples. This makes the trained few-shot classifier very rigid to be deployed in real world scenarios as one cannot expect to know beforehand the number of classes (M) and the number of training examples (K) available for an unseen task. Moreover, the model will suffer performance degradation if the number of examples present are less than K [94].

[3] https://paperswithcode.com/sota/few-shot-image-classification-on-mini-2

| Model | 1-shot | 5-shot | Type |
|---|---|---|---|
| Matching Networks [13] | 43.56 | 55.31 | Metric |
| MAML [14] | 48.7 | 63.15 | Optimization |
| ProtoNet [21] | 49.42 | 68.2 | Metric |
| Relation Net [22] | 50.44 | 65.32 | Metric |
| ProtoNet + Margin [29] | 51.62 | 70.24 | Metric |
| CAVIA [43] | 51.82 | 65.85 | Optimization |
| SNAIL [50] | 55.71 | 68.88 | Model |
| TPN [62] | 55.51 | 69.86 | Hybrid |
| Dynamic FSL [56] | 56.2 | 73 | Hybrid |
| CSN [49] | 56.88 | 71.94 | Model |
| SAML [33] | 57.69 | 73.03 | Metric |
| TADAM [16] | 58.5 | 76.7 | Metric |
| MTL [37] | 61.2 | 75.5 | Optimization |
| TapNet [23] | 61.65 | 76.36 | Metric |
| LEO [38] | 61.76 | 77.59 | Optimization |
| CTM [24] | 62.05 | 78.63 | Metric |
| SCA [63] | 62.86 | 77.64 | Optimization |
| CAN [32] | 63.85 | 79.44 | Metric |
| SimpleShot [68] | 64.29 | **81.5** | Non-Meta-Learning |
| AM3 [54] | 65.3 | 78.1 | Hybrid |
| LST [55] | 70.1 | 78.7 | Hybrid |

Table 9: Accuracy of 5-way classification task on *mini*ImageNet

**Learning Constrained to a Single Distribution of Tasks**
In few-shot learning experiments, training and testing tasks are sampled from the same distribution $p(\mathcal{T})$, which constraints the learning to a single domain. For example, a FSL model trained for character classification tasks on Omniglot [15] may not work well with digit classification on MNIST [95]. Similarly a general few-shot image classifier built on miniImagenNet [13] may not work well for fine-grained classification of Birds [96] or Cars [97]. This problem is also referred to as Cross Domain FSL.

**Performing Joint Classification from Seen and Unseen Classes**
The classes used in the training phase are not retained by the few-shot classifier, i.e. once the training finishes, the model can only classify the incoming query into one of the classes present in the support set of the given task. In real-world usage, it would be desirable to jointly classify the incoming query from training classes (seen) and the new classes (unseen) present in the support set of the task. This challenge is also referred to as Generalized Few-Shot Learning.

**Few-Shot Learning in Data Domains Other than Images**
Availability of large amount of image datasets have resulted in considerable progress in few-shot image classification. Besides the availability, image datasets can be constructed to have uniformly sized images easily categorized into different classes. Contrary to images, data domains like audio and wireless signals have limited large-scale datasets that are uniformly curated. Since uniformity of a dataset is essential to meta-learning, it is a challenge to deploy FSL methods from images to domains like signals.

## References

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[2] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[3] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[4] Yaqing Wang, Quanming Yao, James T. Kwok, and Lionel M. Ni. Generalizing from a few examples: A survey on few-shot learning. 2019.

[5] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Frank Wang, and Jia-Bin Huang. A closer look at few-shot classification. *ArXiv*, abs/1904.04232, 2019.

[6] Lilian Weng. Meta-learning: Learning to learn fast. 2018.

[7] Jurgen Schmidhuber. Evolutionary principles in self-referential learning. on learning now to learn: The meta-meta-meta...-hook. Diploma thesis, Technische Universitat Munchen, Germany, 14 May 1987.

[8] T. Schaul and J. Schmidhuber. Metalearning. *Scholarpedia*, 5(6):4650, 2010. revision #91489.

[9] Sebastian Thrun and Lorien Pratt, editors. *Learning to Learn*. Kluwer Academic Publishers, USA, 1998.

[10] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2009.

[11] Rich Caruana. *Multitask Learning*, pages 95–133. Springer US, Boston, MA, 1998.

[12] R. Polikar. Ensemble learning. *Scholarpedia*, 4(1):2776, 2009. revision #186077.

[13] Oriol Vinyals, Charles Blundell, Timothy P. Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. Matching networks for one shot learning. *CoRR*, abs/1606.04080, 2016.

[14] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[15] Brenden M. Lake, Ruslan Salakhutdinov, and Joshua B. Tenenbaum. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

[16] Boris N. Oreshkin, Pau Rodríguez López, and Alexandre Lacoste. Tadam: Task dependent adaptive metric for improved few-shot learning. In *NeurIPS*, 2018.

[17] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B. Tenenbaum, Hugo Larochelle, and Richard S. Zemel. Meta-learning for semi-supervised few-shot classification. *ArXiv*, abs/1803.00676, 2018.

[18] Oriol Vinyals. Model vs optimization meta learning. *Meta-Learning Symposium at NIPS*, 2017.

[19] Brian Kulis. Metric learning: A survey. *Foundations and Trends® in Machine Learning*, 5(4):287–364, 2013.

[20] Gregory R. Koch. Siamese neural networks for one-shot image recognition. 2015.

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

[22] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1199–1208, 2018.

[23] Sung Whan Yoon, Jun Seo, and Jaekyun Moon. Tapnet: Neural network augmented with task-adaptive projection for few-shot learning. In *ICML*, 2019.

[24] Hongyang Li, David Eigen, Samuel F. Dodge, Matthew D. Zeiler, and Xiaogang Wang. Finding task-relevant features for few-shot learning by category traversal. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–10, 2019.

[25] Jane Bromley, James W. Bentz, Léon Bottou, Isabelle Guyon, Yann LeCun, Cliff Moore, Eduard Säckinger, and Roopak Shah. Signature verification using a "siamese" time delay neural network. In *IJPRAI*, 1993.

[26] Yann Lecun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[27] Akshay Mehrotra and Ambedkar Dukkipati. Skip residual pairwise networks with learnable comparative functions for few-shot learning. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2019, Waikoloa Village, HI, USA, January 7-11, 2019*, pages 886–894. IEEE, 2019.

[28] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 87.1–87.12. BMVA Press, September 2016.

[29] Xianchao Zhang, Jinlong Nie, Linlin Zong, Hong Yu, and Wenxin Liang. One shot learning with margin. In *PAKDD*, 2019.

[30] Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. *ArXiv*, abs/1503.02531, 2015.

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

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

[33] Fusheng Hao, Fengxiang He, Jun Cheng, Lei Wang, Jian zhong Cao, and Dacheng Tao. Collect and select: Semantic alignment metric learning for few-shot learning. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8459–8468, 2019.

[34] Marcin Andrychowicz, Misha Denil, Sergio Gomez Colmenarejo, Matthew W. Hoffman, David Pfau, Tom Schaul, and Nando de Freitas. Learning to learn by gradient descent by gradient descent. In *NIPS*, 2016.

[35] Sachin Ravi and Hugo Larochelle. Optimization as a model for few-shot learning. In *ICLR*, 2017.

[36] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[37] Qianru Sun, Yaoyao Liu, Tat-Seng Chua, and Bernt Schiele. Meta-transfer learning for few-shot learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 403–412, 2018.

[38] Andrei A. Rusu, Dushyant Rao, Jakub Sygnowski, Oriol Vinyals, Razvan Pascanu, Simon Osindero, and Raia Hadsell. Meta-learning with latent embedding optimization. *ArXiv*, abs/1807.05960, 2018.

[39] Eleni Triantafillou, Tyler Zhu, Vincent Dumoulin, Pascal Lamblin, Kelvin Xu, Ross Goroshin, Carles Gelada, Kevin Swersky, Pierre-Antoine Manzagol, and Hugo Larochelle. Meta-dataset: A dataset of datasets for learning to learn from few examples. *ArXiv*, abs/1903.03096, 2020.

[40] Muhammad Abdullah Jamal, Guo-Jun Qi, and Mubarak Shah. Task agnostic meta-learning for few-shot learning. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11711–11719, 2018.

[41] Antreas Antoniou, Harrison A Edwards, and Amos J. Storkey. How to train your maml. *ArXiv*, abs/1810.09502, 2018.

[42] Huaxiu Yao, Ying Wei, Junzhou Huang, and Zhenhui Li. Hierarchically structured meta-learning. In *ICML*, 2019.

[43] Luisa M. Zintgraf, Kyriacos Shiarlis, Vitaly Kurin, Katja Hofmann, and Shimon Whiteson. Fast context adaptation via meta-learning. In *ICML*, 2019.

[44] Adam Santoro, Sergey Bartunov, Matthew M Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *ArXiv*, abs/1605.06065, 2016.

[45] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *ArXiv*, abs/1410.5401, 2014.

[46] Qi Cai, Yingwei Pan, Ting Yao, Chenggang Clarence Yan, and Tao Mei. Memory matching networks for one-shot image recognition. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4080–4088, 2018.

[47] Alexander H. Miller, Adam Fisch, Jesse Dodge, Amir-Hossein Karimi, Antoine Bordes, and Jason Weston. Key-value memory networks for directly reading documents. *ArXiv*, abs/1606.03126, 2016.

[48] Tsendsuren Munkhdalai and Hong Yu. Meta networks. *Proceedings of machine learning research*, 70:2554–2563, 2017.

[49] Tsendsuren Munkhdalai, Xingdi Yuan, Soroush Mehri, and Adam Trischler. Rapid adaptation with conditionally shifted neurons. In *ICML*, 2017.

[50] Nikhil Mishra, Mostafa Rohaninejad, Xi Chen, and Pieter Abbeel. A simple neural attentive meta-learner. In *ICLR*, 2017.

[51] Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks. *CoRR*, abs/1410.3916, 2015.

[52] Sainbayar Sukhbaatar, Arthur Szlam, Jason Weston, and Rob Fergus. End-to-end memory networks. In *NIPS*, 2015.

[53] Peng Wang, Lingqiao Liu, Chunhua Shen, Zi Huang, Anton van den Hengel, and Heng Tao Shen. Multi-attention network for one shot learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 6212–6220, 2017.

[54] Chen Xing, Negar Rostamzadeh, Boris N. Oreshkin, and Pedro H. O. Pinheiro. Adaptive cross-modal few-shot learning. In *NeurIPS*, 2019.

[55] Qianru Sun, Xinzhe Li, Yaoyao Liu, Shibao Zheng, Tat-Seng Chua, and Bernt Schiele. Learning to self-train for semi-supervised few-shot classification. In *NeurIPS*, 2019.

[56] Spyros Gidaris and Nikos Komodakis. Dynamic few-shot visual learning without forgetting. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

[57] Han-Jia Ye, Hexiang Hu, De-Chuan Zhan, and Fei Sha. Learning classifier synthesis for generalized few-shot learning. *ArXiv*, abs/1906.02944, 2019.

[58] Mengye Ren, Renjie Liao, Ethan Fetaya, and Richard S. Zemel. Incremental few-shot learning with attention attractor networks. In *NeurIPS*, 2019.

[59] Yu-Xiong Wang, Ross B. Girshick, Martial Hebert, and Bharath Hariharan. Low-shot learning from imaginary data. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7278–7286, 2018.

[60] Hung-Yu Tseng, Hsin-Ying Lee, Jia-Bin Huang, and Ming-Hsuan Yang. Cross-domain few-shot classification via learned feature-wise transformation. *ArXiv*, abs/2001.08735, 2020.

[61] Vladimir N. Vapnik. Transductive inference and semi-supervised learning. In *Semi-Supervised Learning*, 2006.

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

[63] Antreas Antoniou and Amos J. Storkey. Learning to learn via self-critique. *ArXiv*, abs/1905.10295, 2019.

[64] Gabriel Huang, Hugo Larochelle, and Simon Lacoste-Julien. Centroid networks for few-shot clustering and unsupervised few-shot classification. *ArXiv*, abs/1902.08605, 2019.

[65] Yongqin Xian, Bernt Schiele, and Zeynep Akata. Zero-shot learning - the good, the bad and the ugly. *CoRR*, abs/1703.04394, 2017.

[66] Li Zhang, Tao Xiang, and Shaogang Gong. Learning a deep embedding model for zero-shot learning. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3010–3019, 2017.

[67] Lisa Torrey and Jude Shavlik. Transfer learning. In *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*, pages 242–264. IGI Global, 2010.

[68] Yan Wang, Wei-Lun Chao, Kilian Q. Weinberger, and Laurens van der Maaten. Simpleshot: Revisiting nearest-neighbor classification for few-shot learning. *ArXiv*, abs/1911.04623, 2019.

[69] Yinbo Chen, Xiaolong Wang, Zhuang Liu, Huijuan Xu, and Trevor Darrell. A new meta-baseline for few-shot learning. *ArXiv*, abs/2003.04390, 2020.

[70] Yonglong Tian, Yue Wang, Dilip Krishnan, Joshua B. Tenenbaum, and Phillip Isola. Rethinking few-shot image classification: a good embedding is all you need?, 2020.

[71] Guneet S. Dhillon, P. Chaudhari, Avinash Ravichandran, and Stefano Soatto. A baseline for few-shot image classification. *ArXiv*, abs/1909.02729, 2019.

[72] Imtiaz Masud Ziko, Jose Dolz, Éric Granger, and Ismail Ben Ayed. Laplacian regularized few-shot learning. *ArXiv*, abs/2006.15486, 2020.

[73] Decebal Constantin Mocanu and Elena Mocanu. One-shot learning using mixture of variational autoencoders: a generalization learning approach. *ArXiv*, abs/1804.07645, 2018.

[74] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

[75] Hao Chen, Yali Wang, Guoyou Wang, and Yu Qiao. Lstd: A low-shot transfer detector for object detection. *ArXiv*, abs/1803.01529, 2018.

[76] Bingyi Kang, Zhuang Liu, Xin Wang, Fisher Yu, Jiashi Feng, and Trevor Darrell. Few-shot object detection via feature reweighting. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 8419–8428, 2018.

[77] Qi Fan, Wei Zhuo, and Yu-Wing Tai. Few-shot object detection with attention-rpn and multi-relation detector. *ArXiv*, abs/1908.01998, 2019.

[78] Eli Schwartz, Leonid Karlinsky, Joseph Shtok, Sivan Harary, Mattias Marder, Sharath Pankanti, Rogério Schmidt Feris, Abhishek Kumar, Raja Giryes, and Alexander M. Bronstein. Repmet: Representative-based metric learning for classification and few-shot object detection. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5192–5201, 2018.

[79] Claudio Michaelis, Ivan Ustyuzhaninov, Matthias Bethge, and Alexander S. Ecker. One-shot instance segmentation. *ArXiv*, abs/1811.11507, 2018.

[80] Ansi Zhang, Shaobo Li, Yuxin Cui, Wanli Yang, Rongzhi Dong, and Jianjun Hu. Limited data rolling bearing fault diagnosis with few-shot learning. *IEEE Access*, 7:110895–110904, 2019.

[81] Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. Diverse few-shot text classification with multiple metrics. *arXiv preprint arXiv:1805.07513*, 2018.

[82] Shumin Deng, Ningyu Zhang, Zhanlin Sun, Jiaoyan Chen, and Huajun Chen. When low resource nlp meets unsupervised language model: Meta-pretraining then meta-learning for few-shot text classification. *arXiv*, pages arXiv–1908, 2019.

[83] Seungjoo Yoo, Hyojin Bahng, Sunghyo Chung, Junsoo Lee, Jaehyuk Chang, and Jaegul Choo. Coloring with limited data: Few-shot colorization via memory augmented networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11283–11292, 2019.

[84] Manasi Vartak, Arvind Thiagarajan, Conrado Miranda, Jeshua Bratman, and Hugo Larochelle. A meta-learning perspective on cold-start recommendations for items. In *NIPS*, 2017.

[85] Zhengxiao Du, Xiaowei Wang, Hongxia Yang, Jingren Zhou, and Jie Tang. Sequential scenario-specific meta learner for online recommendation. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2019.

[86] Fan Zhou, Chengtai Cao, Kunpeng Zhang, Goce Trajcevski, Ting Zhong, and Ji Geng. Meta-gnn: On few-shot node classification in graph meta-learning. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, pages 2357–2360, 2019.

[87] Jongmin Kim, Taesup Kim, Sungwoong Kim, and Chang D Yoo. Edge-labeling graph neural network for few-shot learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11–20, 2019.

[88] Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. Fewrel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *EMNLP*, 2018.

[89] Prashant Anand, Ajeet Kumar Singh, Siddharth Srivastava, and Brejesh Lall. Few shot speaker recognition using deep neural networks. *arXiv preprint arXiv:1904.08775*, 2019.

[90] Archit Parnami and Minwoo Lee. Few-shot keyword spotting with prototypical networks. *arXiv preprint arXiv:2007.14463*, 2020.

[91] Szu-Yu Chou, Kai-Hsiang Cheng, Jyh-Shing Roger Jang, and Yi-Hsuan Yang. Learning to match transient sound events using attentional similarity for few-shot sound recognition. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 26–30. IEEE, 2019.

[92] Yan Duan, Marcin Andrychowicz, Bradly Stadie, OpenAI Jonathan Ho, Jonas Schneider, Ilya Sutskever, Pieter Abbeel, and Wojciech Zaremba. One-shot imitation learning. In *Advances in neural information processing systems*, pages 1087–1098, 2017.

[93] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[94] Tianshi Cao, Marc Teva Law, and Sanja Fidler. A theoretical analysis of the number of shots in few-shot learning. *ArXiv*, abs/1909.11722, 2020.

[95] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[96] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[97] J. Krause, M. Stark, J. Deng, and L. Fei-Fei. 3d object representations for fine-grained categorization. In *2013 IEEE International Conference on Computer Vision Workshops*, pages 554–561, 2013.