RETHINKING FEW-SHOT LEARNING FOR SPEECH, CONTINUAL LEARNING AND PRIVACY

by

Archit Parnami

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Computing and Information Systems

Charlotte

2022

Approved by:

Dr. Minwoo Lee

Dr. Min C. Shin

Dr. Liyue Fan

Dr. Razvan Bunescu

Dr. Wenwu Tang

ABSTRACT

ARCHIT PARNAMI. Rethinking Few-Shot Learning For Speech, Continual Learning And Privacy. (Under the direction of DR. MINWOO LEE)

The availability of large amounts of labeled training data is a major contributing factor (and a bottleneck) to the recent progress in the field of Deep Learning. However, collecting and labeling data is a time consuming and expensive process. Oftentimes, the data cannot be collected due to privacy reasons or is just not available. This has led to an emergence of research in *Few-Shot Learning*, a new sub-domain of machine learning that focuses on building models that can learn from a few number of training examples. The recent progress in few-shot learning has shown promising results of achieving up to 90% accuracy on the task of 5-way image classification using just five training examples per class. The success of few-shot learning, however, is too much concentrated on image classification and the emergent field requires strict scrutiny. For example, 1) its behavior on speech data is unknown; 2) the nature of few-shot models to continually update when new category of data is witnessed remains untested; and 3) finally the privacy issues surrounding the data used in the few-shot model have been unaddressed. Therefore, this dissertation study develops few-shot model for audio data (Few-Shot Keyword Spotting), explores how few-shot learning models can continuously learn from the new incoming data (Few-Shot Continual Learning), and discusses how privacy can be an inbuilt part of few-shot learning (Privacy-Enhanced Few-Shot Learning).

DEDICATION

Dedicated to my parents (Reeta & Parveen), my brother (Pulkit) and my dada ji (Sharwan Kumar Parnami) for their everlasting love and support.

ACKNOWLEDGEMENTS

My sincerest thanks to my advisor – Dr. Minwoo Lee, for his continued guidance and supervision in my PhD studies. I also wish to express my gratitude to my committee members: Dr. Min C. Shin, Dr. Liyue Fan, Dr. Razvan Bunescu and Dr. Wenwu Tang for their time and feedback. I thank the University of North Carolina at Charlotte for PhD tuition funding and support. I am grateful for this experience.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

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 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 [3], Machine Translation [4] and Speech Modeling [5]. However, these achievements have relied on the optimization of these deep, high-capacity models that require many iterative updates across many labeled examples. This type of optimization breaks down in the small data regime, where learning from very few labeled examples.

In contrast, humans can quickly learn to solve a new problem from a few examples. For instance, given a few photos of a stranger, a child can easily identify the same person from a large number of photos [6]. This is due not only 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 known as One-Shot Learning.

The practical issues including the cost of correctly labeled data, thus often scarce or poorly labeled, motivate the development of FSL. Instead of collecting thousands of labeled examples to attain a reasonable performance on a new task, FSL helps 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, for e.g. medical imaging, biometric authentication etc. Models that generalize from a few examples would be able to capture this type of data effectively.

Alleviating the large data challenge from deep learning, FSL has shown success in many computer vision problems such as Character Recognition [7], Image Classification [8], Object Detection [9], Image Retrieval [10], Object Tracking [11], Video Classification [12], Motion Prediction [13], Action Localization [14], Person Re-Identification [15], and etc. In Natural Language Processing (NLP), FSL has been used for tasks such as Text Parsing [16], Translation [17], Sentence Completion [18], User Intent Classification [19] etc. FSL has also been useful in other domains, including, but not limited to Drug Discovery [20], Network Architecture Search [21], and Robotics [22, 23].

Despite its success in Computer Vision and recent developments in NLP, the adoption in the speech domain lacks research. With the growing usage of personalized voice assistants, it becomes important to address the large data requirement challenge involved in training voice models for the task of speaker identification or spoken word recognition i.e., a problem widely known as keyword spotting. Secondly, the static nature of few-shot models limits its application where the model needs to update itself to recognize new classes in a continual learning fashion. Finally, few-shot learning models are vulnerable to data privacy attacks when deployed in the cloud. Therefore, in this dissertation, we enhance few-shot learning by proposing techniques for its adaptability in the speech domain, continuity of learning, and data privacy. The following summarizes these three problems and the proposed solution in the context of few-shot learning (Fig. 1.1):

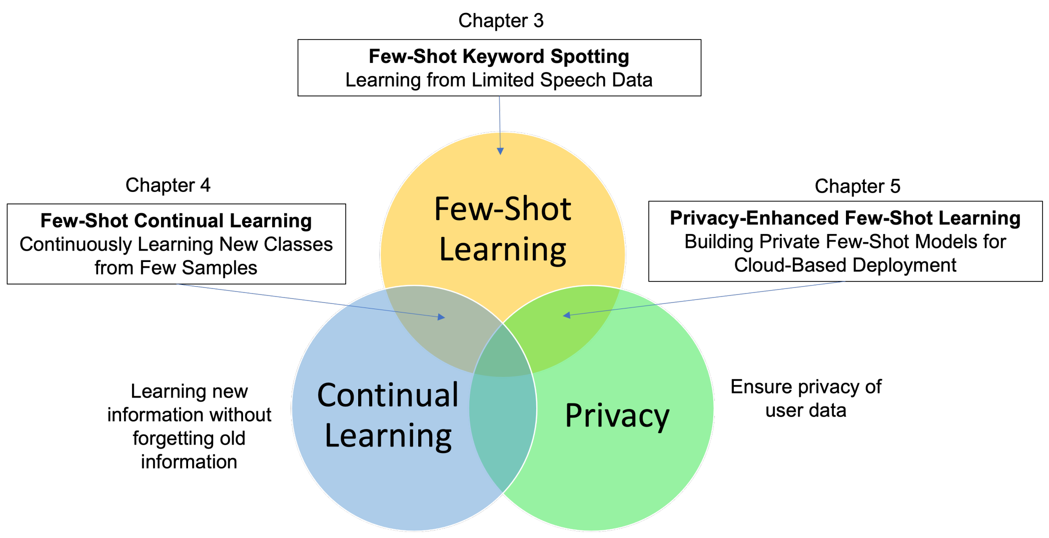

Chapter 3
Few-Shot Keyword Spotting
Learning from Limited Speech Data
Chapter 4
Few-Shot Continual Learning
Continuously Learning New Classes from Few Samples
Chapter 5
Privacy-Enhanced Few-Shot Learning
Building Private Few-Shot Models for Cloud-Based Deployment
Few-Shot Learning
Continual Learning
Privacy
Learning new information without forgetting old information
Ensure privacy of user data

Figure 1.1: Problems addressed in this work 1) Few-Shot Keyword Spotting, 2) Few-Shot Continual Learning and 3) Privacy-Enhanced Few-Shot Learning

1. **Few-Shot Keyword Spotting**: Recognizing a particular command or a keyword, keyword spotting has been widely used in many voice interfaces such as Amazon's Alexa and Google Home. In order to recognize a set of keywords, most of the recent deep learning based approaches use a neural network trained with a large number of samples to identify certain *pre-defined* keywords. This restricts the system from recognizing *new, user-defined* keywords. Therefore, **our objective is to recognize new keywords without the need for large training samples**, hence we first formulate this problem as a *few-shot keyword spotting* and approach it using metric learning. To enable this research, we also synthesize and publish a Few-shot Google Speech Commands dataset. We then propose a solution to the few-shot keyword spotting problem using temporal and dilated convolutions on prototypical networks. Our comparative experimental results demonstrate keyword spotting of new keywords using just a small number of samples.

2. **Few-Shot Continual Learning**: Like most machine learning models, **few-shot models are also static in nature, i.e., they are limited by their ability to be updated with new information without forgetting old information**, a problem otherwise generally known as catastrophic forgetting [24] in neural networks. Moreover, continual learning in a limited data regime is much harder than traditional continual learning as learning new representations from limited data is a challenging task in itself. Recent continual learning approaches for few-shot models then try to rectify this problem of not forgetting old information by trying to fix the problem size and meta-learning a distribution of few-shot continual learning tasks. This is an impractical assumption as in real-time we wouldn't likely ever know the amount of new information that we are going to witness. Therefore, to solve this issue, in this research we develop upon the idea of one-class classification to learn new classes just from a few samples and then use an ensemble of one-class classifiers to solve the problem of few-shot continual learning. Our results show that upon incorporating negative sampling in the learning process of one-class classifiers we can build better few-shot models for continual learning.

3. **Privacy-Enhanced Few-Shot Learning**: Requiring less data for accurate models, few-shot learning has shown robustness and generality in many application domains. However, **deploying few-shot models in untrusted environments may inflict privacy concerns**, e.g., attacks or adversaries that may breach the privacy of user-supplied data. Our work studies the privacy enhancement for the few-shot learning in an untrusted environment, e.g., the cloud, by establishing a novel privacy-preserved embedding space that preserves the privacy of data and maintains the accuracy of the model. We examine the impact of various image privacy methods such as blurring, pixelization, Gaussian noise, and differentially private pixelization (DP-Pix) on few-shot image

classification and propose a method that learns privacy-preserved representation through the joint loss. Our empirical results show how privacy-performance trade-off can be negotiated for privacy-enhanced few-shot learning.

CHAPTER 2: BACKGROUND

In this chapter, we discuss the required background for understanding the idea behind few-shot learning. We categorize the approaches to few-shot learning in Figure 2.1. Broadly, they are classified into meta-learning based methods and non-meta-learning based methods (such as Transfer Learning). The meta-learning based approaches are further classified into metric-based, optimization-based and model-based [25]. In this dissertation, our focus is on meta-learning based approaches, particularly metric-based and optimization based methods. In Section 2.1, we give a formal introduction to meta-learning; in Section 2.3.2, we discuss metric-based meta-learning that is used in Chapter 3 and 4, and in Section 2.3.3, we discuss optimization-based meta-learning used in Chapter 5.

2.1 Meta-Learning

Meta-Learning [26, 27] or Learning to Learn [28] has been the basic technique which most few-shot learning algorithms employ. Motivated by human development theory, meta-learning, a subfield 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.

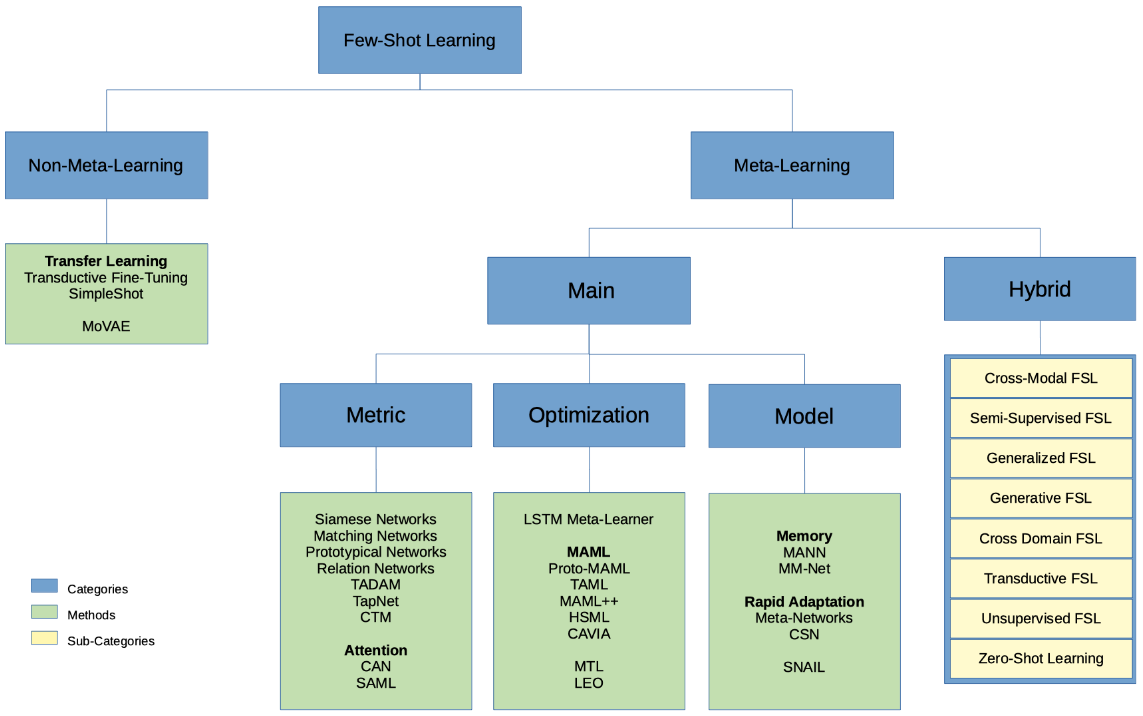


Figure 2.1: Approaches to FSL are categorized into Meta-Learning-based FSL and Non-Meta-Learning-based FSL. The three main meta-learning approaches are: metric-based, optimization-based and model-based meta-learning. Furthermore, variations of the FSL problem which use meta-learning are categorized as hybrid approaches.

Next, this learning is guided by knowledge accrued more gradually *across tasks*, which captures the way task structure varies across target domains [28, 29, 30]

Meta-Learning can be different from similar approaches such as transfer learning, multi-task learning, or ensemble learning. In Transfer Learning [31], 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, the more similar the two domains are the better it performs . Multi-Task Learning [32] involves learning 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 [33] 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 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:

$$\mathcal{D}^{train} = \{(x_k, y_k)\}_{k=1}^{t} \tag{2.1}$$

and

$$\mathcal{D}^{test} = \{(x_k, y_k)\}_{k=t+1}^{n}, \tag{2.2}$$

where t denotes the number of training samples. We optimize parameters θ 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 θ as [1]:

$$y \approx f(x;\theta)\ \forall (x,y) \in \mathcal{D}^{test} \tag{2.3}$$

and

$$\theta = \arg\min_{\theta} \sum_{(x,y)\in\mathcal{D}^{train}} \mathcal{L}(f(x,\theta),y) \tag{2.4}$$

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 θ 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 θ as:

$$y \approx f(\mathcal{D}_i^{train}, x; \theta)\ \forall (x,y) \in \mathcal{D}_i^{test} \tag{2.5}$$

and

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

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

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:

$$\theta = \arg\min_{\theta} \sum_{\mathcal{D}_i \in \mathcal{D}_{meta-train}} \sum_{(x,y) \in \mathcal{D}_i^{test}} \mathcal{L}(f(\mathcal{D}_i^{train}, x; \theta), y). \tag{2.6}$$

That is the meta-learner learns parameters θ 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 2.2 demonstrates a setup for example meta-learning problem.

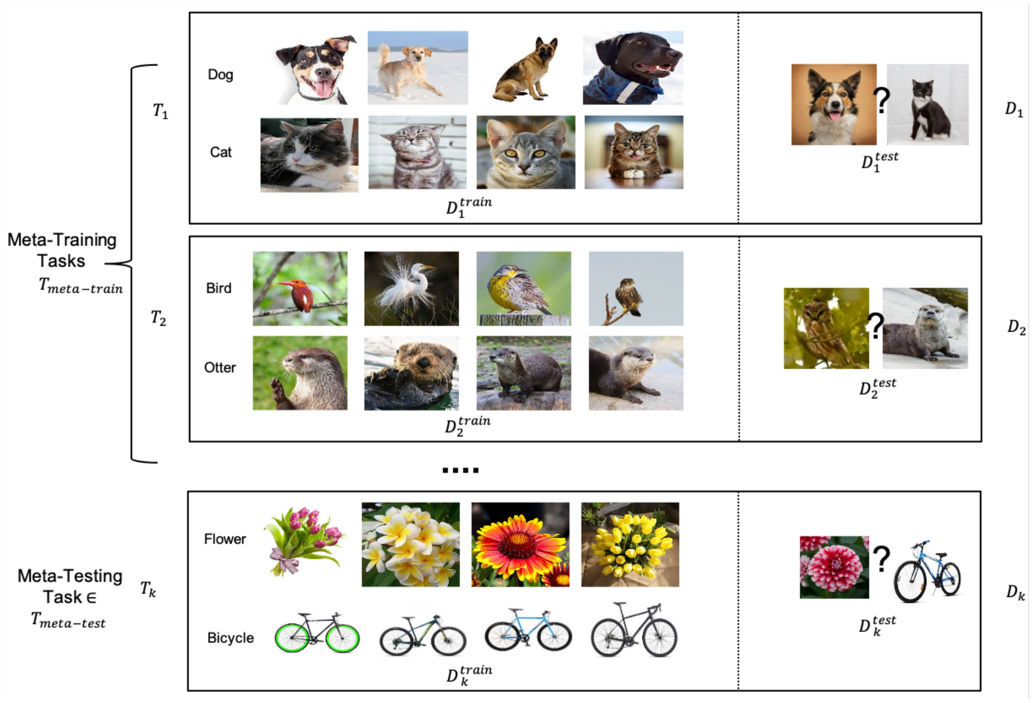


Figure 2.2: Meta-learning example setup. Each task $\mathcal{T}_i$ is a binary classification task with a training set $\mathcal{D}_i^{train}$ and test set $\mathcal{D}_i^{test}$. During meta-training, the labels for samples in $\mathcal{D}_i^{test}$ is known and the goal of meta-learner is to find optimal θ as per equation 2.6. During meta-testing, new task with unseen categories is presented and the labels are predicted as per equation 2.5 .

2.1.2 Nomenclature

In meta-learning and few-shot learning literature, certain notations and terms are used interchangeably. Table 2.1 lists these terms and their equivalent usage. Notation **A** is more commonly used in optimization-based meta-learning literature (Section 2.3.3) while notation **B** is used when discussing metric-based meta-learning methods (Section 2.3.2). Additionally, Table 2.2 lists the commonly used symbols in this dissertation.

Table 2.1: Nomenclature

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

2.2 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.

2.2.1 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. 2.3) with parameters θ (Eqn. 2.4) . This is generally possible when we have sufficient training data $\mathcal{D}^{train}$ (Eqn. 2.1) i.e., t is a large number. However, when t is small,

Table 2.2: Commonly used symbols and their meaning

Symbol	Meaning	Context
$\mathcal{T}_i$	Task i	
$\mathcal{L}$	Loss function	
(x_k, y_k)	Input-Output pair	
f_θ	Model (function) with parameters θ	
g_{θ_1}	Embedding function	Sec. 2.3.2
d_{θ_2} or d	Distance function	Sec. 2.3.2
g_ϕ	Meta-Learning model with parameters ϕ	Sec. 2.3.3
$P_\theta(y\|x)$	Output probability of y for input x using model parameters θ	
$k_\theta(x_1, x_2)$	Kernel function measuring similarity between two vectors x_1 and x_2	Table 2.4
σ	Softmax function	
α, β	Learning rates	
w	Weights	
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 2.5
B	Number of batches (X_b, Y_b) sampled in inner-loop for a randomly sampled task $\mathcal{T}_i$	Table 2.6
I	Number of tasks $\mathcal{T}_i$ sampled in inner-loop	Table 2.6
J	Number of outer-loop iterations	Table 2.6

it becomes difficult to approximate the function f so that it has good generalization performance over $\mathcal{D}^{test}$ (Eqn. 2.2). 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.

Usually people define few-shot classification task as a standard **M-way-K-shot** task [8, 18], 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 2.3 lists such commonly used datasets for conducting experiments for few-shot classification.

Table 2.3: Common datasets used for Few-Shot Learning

Dataset	**Number of classes**	**Samples per class**	**Description**
Omniglot [34]	1623	20	Handwritten characters from different languages.
*mini*ImageNet [18]	100	600	100 classes randomly sampled from ImageNet.
FC100 [35]	100	600	Derived from CIFAR100.
*tiered*ImageNet [36]	608	1280 (avg.)	Like *mini*ImageNet but ensures that there is a wider degree of separation between training, validation and test classes.

2.3 Meta-Learning-based Few-Shot Learning

The objective of meta-learning is to approximate the function f with parameters θ such that the performance on any task $\mathcal{T}_i$ randomly sampled from the task distribution $p(\mathcal{T})$ is optimal (Eqn. 2.5). 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 θ 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 [25]: 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 first two main approaches in the following section and refer the reader to our work [37] for learning more about other approaches.

2.3.1 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 θ. Then for $(x, y) \in Q$, the meta-learning approaches to FSL can be differentiated in the way they model the posterior probability $P_\theta(y|\mathbf{x})$ [25] (Table 2.4).

Table 2.4: Meta-Learning Approaches

	Metric-based	**Optimization-based**	**Model-based**
Key idea	Metric Learning [38]	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.	Flexible optimization in dynamic environments.	Faster inference with memory models.
	Easy to deploy.	Does not require storing S in memory i.e, samples can be discarded post-optimization.	Eliminates the need for defining a metric or optimizing at test.
Disadvantages	Less adaptive to optimization in dynamic environments.	Optimization at inference is undesirable for real-world deployment.	Less efficient to hold data in memory as S grows.
	Computational complexity grows linearly with size of S at test.	Prone to overfitting.	Hard to design.

2.3.2 Metric-based Meta-Learning

Metric Learning [38] 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 θ_1) given a distance function d (such as euclidean distance) or to learn both the embedding function $g(;\theta_1)$ (with parameters θ_1) and the distance function $d(;\theta_2)$ (usually parameterized by another neural network with parameters θ_2). This is illustrated in Figure 2.3 .

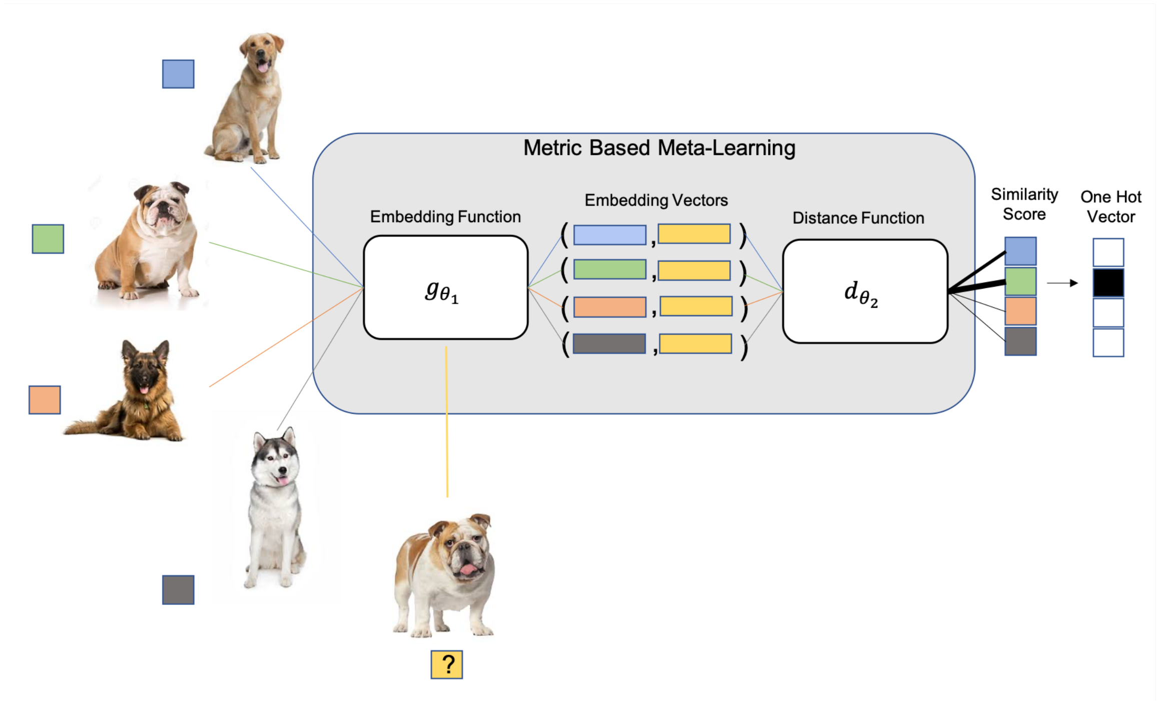


Figure 2.3: Example metric-based meta-learning setup for a 4-way-1-shot classification task. The embedding function g_{θ_1} outputs the embedding vectors for support images (labeled) and the query image (unlabeled, denoted by '?'). Distance function d_{θ_2} measures the distance between support and query 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 2.5 compares the recent metric-based meta-learning methods based on their characteristics like embedding function, distance measure, and if the embedding function is fixed for all tasks i.e., task-independent (**T.I**) or is adaptive (task-dependent).

Table 2.5: Metric-based Meta-Learning Methods

Method	**T.I**	$\mathbf{g_{\theta_1}}$	$\mathbf{d_{\theta_2}}$
Siamese Networks [39]	Yes	CNN	L1
Matching Networks [18]	Yes	CNN + LSTM w/ attention	Cosine Similarity
Prototypical Networks [40]	Yes	CNN	Euclidean
Relation Networks [41]	Yes	CNN	Learned by CNN
TADAM [35]	No	ResNet-12	Cosine/Euclidean
TapNet [42]	No	Resnet-12	Euclidean
CTM [43]	No	Any	Any

2.3.2.1 Metric-based Few-Shot Learning Framework

We base our Few-Shot Keyword Spotting Framework (Fig. 3.2) and Few-shot Private Image Classification Framework (Fig. 5.2) on Prototypical Networks [40] (Fig. 2.4).

The few-shot model is trained on a labeled dataset D_{train} and tested on D_{test}. The set of classes present in D_{train} and D_{test} are disjoint. The test set has only a few labeled samples per class. We follow an episodic training paradigm in which each episode the model is trained to solve an N-way K-Shot few-shot task. Each episode e is created by first sampling N categories from the training set and then sampling two sets of examples from these categories: (1) the support set $S_e = \{(s_i, y_i)\}_{i=1}^{N\times K}$ containing K examples for each of the N categories and (2) the query set $Q_e = \{(q_j, y_j)\}_{j=1}^{N\times Q}$

Algorithm 1: Episodic Training in Metric-based Meta-Learning Methods (Adapted from [40])

```
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.
```

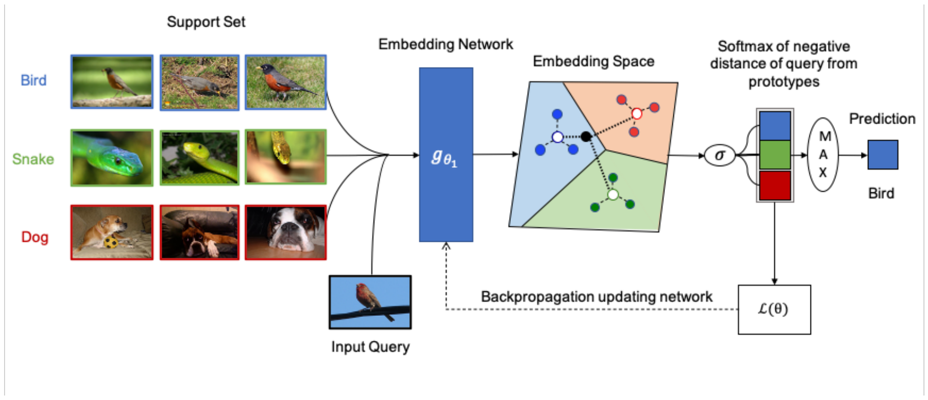


Figure 2.4: Few-shot prototypes $\mathbf{p_c}$ are computed as the mean of embedded support examples for each class. The embedded query points are classified via a softmax over distances to the class prototypes.

containing Q different examples from the same N categories. The episodic training for few-shot task minimizes, for each episode, the loss of the prediction on samples in the query set, given the support set. The model is a parameterized function and the loss is the negative log likelihood of the true class of each query sample:

$$L(\theta) = -\sum_{t=1}^{|Q_e|} \log P_\theta(y_t \mid q_t, S_e), \tag{2.7}$$

where $(q_t, y_t) \in Q_e$ and S_e are, respectively, the sampled query and support set at episode e and θ are the parameters of the model.

Prototypical networks make use of the support set to compute a centroid (prototype) for each category (in the sampled episode) and query samples are classified based on the distance to each prototype. The model is a CNN $f : \Re^{n_v} \rightarrow \Re^{n_p}$, parameterized by θ_f, that learns a n_p-dimensional space where n_v-dimensional input samples of the same category are close and those of different categories are far apart. For every episode e, each embedding prototype p_c (of category c) is computed by averaging the embeddings of all support samples of class c:

$$p_c = \frac{1}{|S_e^c|} \sum_{(s_i, y_i) \in S_e^c} f(s_i), \tag{2.8}$$

where $S_e^c \subset S_e$ is the subset of support examples belonging to class c. Given a distance function d, the distance of the query q_t to each of the class prototypes p_c is calculated. By taking a softmax [44] of the measured (negative) distances, the model produces a distribution over the N categories in each episode:

$$P(y = c \mid q_t, S_e, \theta) = \frac{exp(-d(f(q_t), p_c))}{\sum_n exp(-d(f(q_t), p_n))}, \tag{2.9}$$

where metric d is a Euclidean distance and the parameters θ of the model are updated with stochastic gradient descent by minimizing Equation (2.7). Once the training finishes, the parameters θ of the network are frozen. Then, given any new few-shot task, the category corresponding to the maximum P is the predicted category for the input query q_t.

2.3.3 Optimization-based Meta-Learning

Earlier in Section 2.2.1, we discussed that for a few-shot classification task $\mathcal{T}$ with training data $\mathcal{D}^{train}$ (Eqn. 2.1) where the number of training examples t is small, it is difficult to approximate f (Eqn. 2.3) with parameters θ (Eqn. 2.4) from scratch using gradient-based optimization as it is 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 (Fig. 2.2) and episodic training (Algorithm 1), optimization-based methods enable 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_θ 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. 2.4) will not be able to generalize (Eqn. 2.3).

2. **Meta-Learner:** A meta-learner model g_ϕ is not task specific and is trained on a distribution of tasks $\mathcal{T} \sim p(\mathcal{T})$ (Figure 2.2). Using episodic training, the meta-learner learns (ϕ) to update the learner model's parameters (θ) via training set $\mathcal{D}^{train}$,

$$\theta^* = g_\phi(\theta, \mathcal{D}^{train}). \tag{2.10}$$

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

During meta-training (notation **A** in Table 2.1), the optimization process involves updating ϕ for the meta-learner and θ for individual training tasks. Once the meta-training finishes, the prior knowledge is encompassed into ϕ and only θ is updated for a test task (Eqn. 2.10).

Table 2.6 compares different optimization-based meta-learning methods based on how they update learner's parameters θ and meta-learner parameters ϕ. In all the listed methods, learning happens in two-stages. Initially, in the outer-loop, the meta-learner's parameters ϕ are randomly initialized. Next, in the inner-loop the learner parameters (θ) are updated/proposed by meta-learner (Eqn. 2.10). The learners' training loss $\mathcal{L}^{train}$ is further used to obtain optimal parameters θ^*. Finally, in the outer loop the cumulative test loss of learner obtained using θ^* is used for updating ϕ. In some cases learner's initial parameters θ are also meta-learned along with ϕ.

This chapter is reused from our work **Learning from Few Examples: A Summary of Approaches to Few-Shot Learning** [37] with permission from authors.

Table 2.6: Optimization-based Meta-Learning Methods

Method	**Learner**	**Meta-Learner**
LSTM Meta-Learner [45]	Repeat $\forall b \in [1..B]$ $\mathcal{L}_b \leftarrow \mathcal{L}(f(X_b;\theta_{b-1}), Y_b)$ $\theta_b \leftarrow g((\nabla_{\theta_{b-1}}\mathcal{L}_b, \mathcal{L}_b); \phi_{j-1})$	Repeat $\forall j \in [1..J]$ $\mathcal{L}_j^{test} \leftarrow \mathcal{L}(f(X;\theta_B), Y)$ $\phi_j \leftarrow \phi_{j-1} - \alpha\nabla_{\phi_{j-1}}\mathcal{L}_j^{test}$
MAML [8]	Repeat $\forall i \in [1..I]$ $\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{train};\theta_{j-1}))$ $\theta_i^* \leftarrow \theta_{j-1} - \alpha\nabla_{\theta_{j-1}}\mathcal{L}_t^{train}$ $\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{test};\theta_t^*))$	Repeat $\forall j \in [1..J]$ $\theta_j \leftarrow \theta_{j-1} - \beta\nabla_{\theta_{j-1}}\sum_{i=1}^{I}\mathcal{L}_i^{test}$
MTL [46]	$\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{train};[\theta_{j-1}, \phi_{j-1}, \Theta]))$ $\theta_i^* \leftarrow \theta_{j-1} - \alpha\nabla_{\theta_{j-1}}\mathcal{L}_i^{train}$ $\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{test};\theta_i^*))$	$\theta_j \leftarrow \theta_{j-1} - \beta\nabla_{\theta_{j-1}}\sum_{i=1}^{I}\mathcal{L}_i^{test}$ $\phi_j \leftarrow \phi_{j-1} - \beta\nabla_{\phi_{j-1}}\sum_{i=1}^{I}\mathcal{L}_i^{test}$
LEO [47]	$\phi_{j-1} = \{\phi_e, \phi_r, \phi_d, \alpha\}$ $\mathbf{z_i} \leftarrow g(\mathcal{D}_i^{train};[\phi_e, \phi_r, \Theta])$ $\theta_i \leftarrow g(\mathbf{z_i};\phi_d)$ $\mathcal{L}_i^{train} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{train};\theta_i))$ $\mathbf{z_i^*} \leftarrow \mathbf{z_i} - \alpha\nabla_{\mathbf{z_i}}\mathcal{L}_i^{train}$ $\theta_i^* \leftarrow g(\mathbf{z_i^*};\phi_d)$ $\mathcal{L}_i^{test} \leftarrow \mathcal{L}(f(\mathcal{D}_i^{test};\theta_i^*))$	$\phi_j \leftarrow \phi_{j-1} - \beta\nabla_{\phi_{j-1}}\sum_{i=1}^{I}\mathcal{L}_i^{test}$

CHAPTER 3: FEW-SHOT KEYWORD SPOTTING

3.1 Motivation

The underlying motivation for developing an approach that can learn from a few samples of speech can be found in smart voice recognition devices which have found their way in our everyday lives. Most smart devices these days have an inbuilt voice recognition system which is mainly used for taking voice input from a user. This requires the voice recognition system to detect specific words (keywords/commands), popularly known as the Keyword Spotting (KWS) problem (Fig. 3.1). Most approaches use either Large Vocabulary Continuous Speech Recognition (LVCSR) based models [48, 49] or lightweight deep neural network based models [50]. The former, LVCSR demands a lot of resource and computation power and hence is deployed in the cloud, raising privacy concerns and latency issues. The latter models are trained with a set of pre-defined keywords to recognize using thousands of training examples. However, with smart devices becoming more personalized, there is a growing need for such systems 1) to recognize custom or new keywords on-device without having to retrain the model and 2) to work in resource constrained environments such as embedded systems. Therefore, in this research, we propose to address the problem of learning from a few samples of speech keywords in resource constrained environments, hereon referred to as Few-Shot Keyword Spotting (FS-KWS).

3.2 Related Works

Current approaches to KWS involve extracting audio features from the input keyword and then passing it as input to a Deep Neural Network (DNN) for classification [50, 51, 52, 53, 54]. Especially, the use of convolutional neural networks (CNNs) [55] in

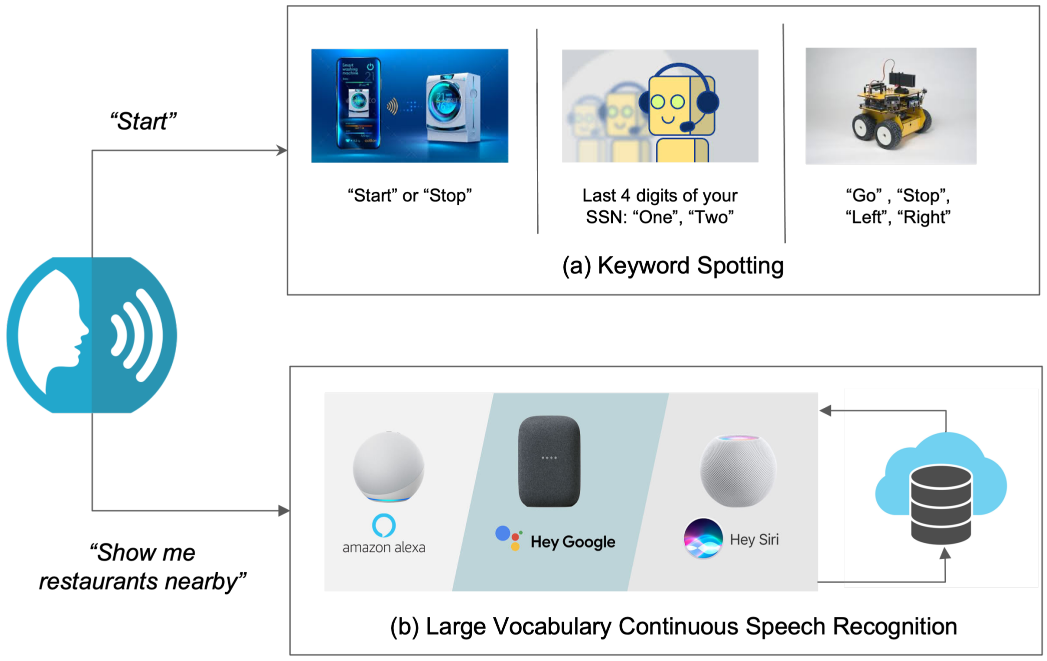

"Start"
"Start" or "Stop"
Last 4 digits of your SSN: "One", "Two"
"Go" , "Stop", "Left", "Right"
(a) Keyword Spotting
"Show me restaurants nearby"
amazon alexa
Hey Google
Hey Siri
(b) Large Vocabulary Continuous Speech Recognition

Figure 3.1: Two types of Speech Recognition: (a) Keyword Spotting and (b) Large Vocabulary Continuous Speech Recognition

adjunction with Mel-frequency Cepstral Coefficients (MFCC) as speech features have shown to produce remarkable results [50, 51, 54, 56, 57]. Some attempts [58] have been made to solve FS-KWS using model-agnostic meta learning (MAML) [8], an optimization based approach to FSC. However, since KWS is deployed on small devices with limited computation capability, an optimization based approach that requires fine-tuning is not feasible. Hence, we approach FS-KWS using metric learning based approach, specifically using Prototypical Networks [40] which can perform inference in an end-to-end manner. The following summarizes our main contributions:

- We propose to enhance few-shot learning for the speech dataset by particularly focusing on the problem of keyword spotting. In this direction, we propose a keyword spotting system that can classify new keywords from limited samples by a few-shot formulation of keyword spotting with metric learning.

- We propose a lightweight temporally dilated CNN architecture as a better embedding function for encoding speech features for FS-KWS, which is also operable in resource constrained environments such as micro-controllers.

- We release a FS-KWS dataset synthesized from Google's Speech command dataset [59]. To make it more challenging, we also incorporate background noise and detection of silence and unknown (negative) keywords.

3.3 Few-Shot Keyword Spotting (FS-KWS) Framework

Consider a set S of *user-defined* keywords such that $S = \{(s_i, y_i)\}_i^{N \times K}$ where s_i is a keyword sample (voice input) and y_i is its label. The set S contains N keywords, each keyword having K samples where K is a small number (for ex., 1,2,5). Then given a user query q, the objective of FS-KWS system is to classify q into one of N keyword classes. The user-defined keywords in S could be new i.e, never seen before during the training of FS-KWS system. Yet, the system should be able to detect q, given S. We base our FS-KWS framework (Figure 3.2) on Prototypical Networks as defined in Section 2.3.2.1.

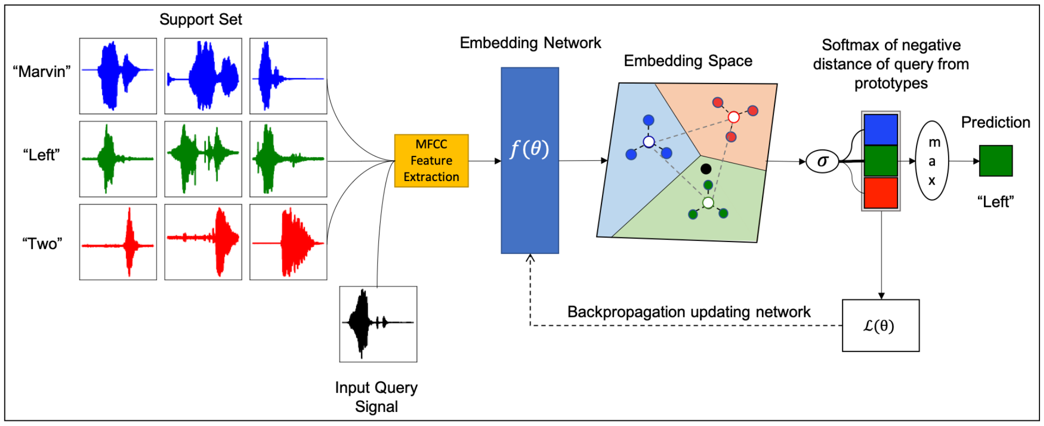
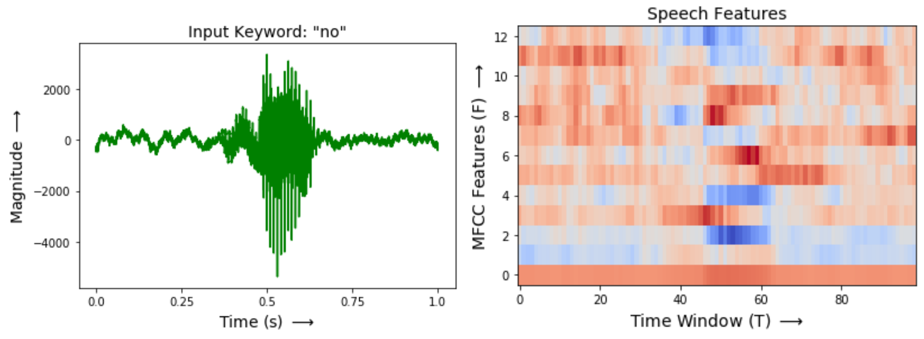


Figure 3.2: Few-Shot Keyword Spotting Pipeline

3.3.1 Audio Feature Extraction

In each episode, we first obtain Mel-frequency Cepstral Coefficients (MFCC) features for all the examples in the support set and the query set which then act as input to the embedding network as shown in Figure 3.2. Following [52], we extract

40 MFCC features from a speech frame of length 40 *ms* and stride 20 *ms* (see Figure 3.3).



(a) Input Speech (b) MFCC Features

Figure 3.3: Example transformation of input speech to MFCC features

3.3.2 Embedding Network

[56] demonstrated improved performance on KWS with temporal convolutions by reshaping the input MFCC features (Figure 3.4). Also, [57] have shown that dilated convolutions are helpful in the processing of keyword signals. Therefore, we combine both techniques by first reshaping the input MFCC features and then performing temporal convolutions along with dilation. We modify the TC-ResNet8 [56] architecture to reduce the size of the kernel to 7×1 and use dilation of 1, 2, and 4 with stride 1 in three ResNet blocks respectively. This proposed architecture TD-ResNet7 (Figure 3.5) is then used to embed the reshaped input MFCC features (Figure 3.4).

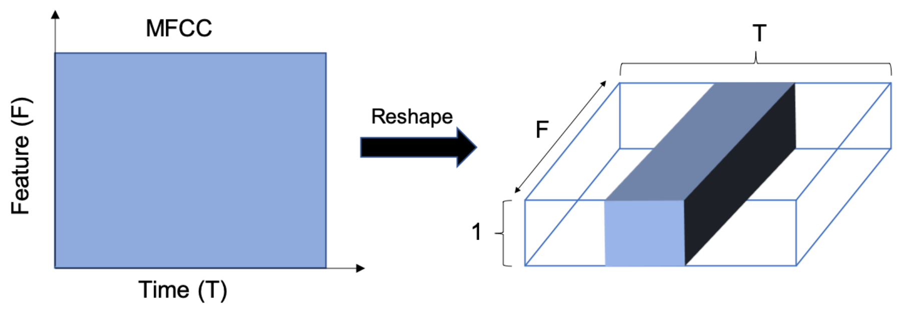
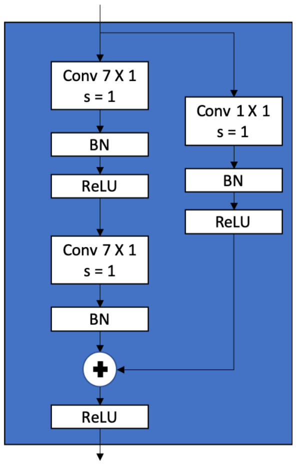
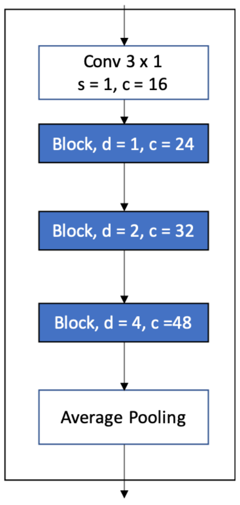


Figure 3.4: Reshaping MFCC features for time convolution.



(a) Block



(b) TD-ResNet7

Figure 3.5: The proposed dilated time convolutional neural network for embedding.

3.4 Few-Shot Google Speech Command Dataset

Google's Speech Commands dataset [59] is used [52, 56] for keyword spotting problems. The dataset has a total of 35 keywords and contains multiple utterances of each keyword by multiple speakers. Each utterance is stored as a one-second (or less) WAVE format file, with the sample data encoded as linear 16-bit single-channel PCM values, at a 16 kHz rate [59]. We curate a FS-KWS dataset from this dataset by performing the following preprocessing steps:

1. **Filtering:** We filter out all the utterances which are less than one second. This ensures the consistency of the output MFCC feature matrix obtained from each audio file.

2. **Grouping:** To train our KWS system to detect if an input query is an unknown keyword (not present in S), we group our keywords into two categories: *Core* and *Unknown*. Keywords having more than 1000 speakers are considered as *core* words and the rest are put in the category of *unknown* words.

3. **Balancing:** Next, we balance the dataset so that all keywords in a group have the same number of samples. As a result, we have 30 *core* keywords each with

Table 3.1: Keyword Statistics

Keywords	Speakers	Utterances		
		Min	Max	Mean
Core				
down	1465	1	14	2.44
zero	1450	1	13	2.59
seven	1450	1	11	2.53
nine	1443	1	12	2.51
five	1442	1	19	2.58
yes	1422	1	20	2.6
four	1421	1	14	2.39
left	1416	1	12	2.47
stop	1413	1	22	2.52
six	1411	1	14	2.55
right	1409	1	15	2.45
on	1403	1	19	2.47
three	1401	1	11	2.43
off	1387	1	16	2.47
dog	1385	1	5	1.31
marvin	1378	1	6	1.33
one	1376	1	12	2.54
go	1372	1	12	2.53
no	1368	1	18	2.59
two	1367	1	15	2.58
eight	1358	1	15	2.53
house	1357	1	5	1.35
wow	1336	1	5	1.35
happy	1332	1	7	1.33
bird	1315	1	7	1.34
cat	1300	1	5	1.32
up	1291	1	17	2.53
sheila	1291	1	6	1.36
bed	1257	1	6	1.34
tree	1062	1	6	1.39
Unknown				
visual	412	1	7	3.57
forward	397	1	10	3.66
backward	396	1	23	3.93
follow	387	1	11	3.76
learn	386	1	24	3.69

1062 samples and 5 *unknown* keywords each with 386 samples and where all samples for a particular keyword come from a different speaker.

4. **Splitting:** *(a) Core Keywords.* They are randomly split into 20, 5, and 5 sets for training, validation, and testing respectively. Note that here the splits do not have any classes (keywords) in common. *(b) Unknown Keywords.* They are used for detecting negative inputs. Since we have only 5 keywords in an unknown category, we utilize them in all three phases of training, validation, and testing. For each keyword in the unknown category, 60% of its samples are used in training, 20% for validation, and 20% for testing. Note that in this case, all the training, validation, and test phases use the same 5 keywords as an unknown class but the samples are still from different speakers.

5. **Mixing Background Noise:** The original speech commands dataset [59] comes with a collection of sounds (6 WAVE files) that can be mixed with one-second utterances of keywords to simulate background noise. Following [60] implementation of mixing background noise, small snippets of these files are chosen at random and mixed at a low volume into audio samples during training. The loudness is also chosen randomly, and controlled by a hyper-parameter as a proportion where 0 is silence, and 1 is full volume. In our experiments, we set the background volume to 0.1 and conduct experiments with both the presence and absence of background noise.

6. **Detecting Silence:** Apart from core classes and unknown classes, we curate another class *silence* to detect the absence of keywords. Again following [60] implementation, we randomly sample 1000 one-second-long sections of data from background sounds. Since there is never complete silence in real environments, we have to supply examples with quiet and irrelevant audio. We conduct experiments in both the presence and absence of samples from silence class.

We provide a script to synthesize this Few-Shot Speech Command dataset at our repository [1].

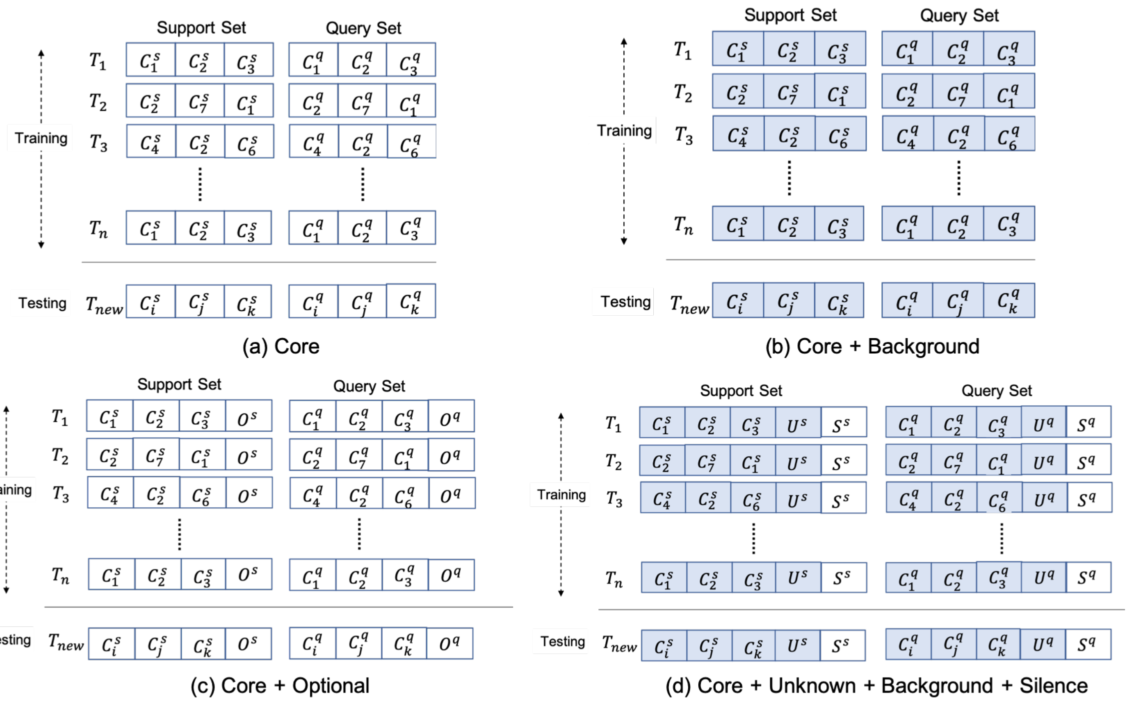


Figure 3.6: Training Cases demonstrated for 3-Way FS-KWS. (a) Core: In each task T_i, 3 *Core* classes are randomly sampled from D_{train}. Then for each *Core* class C_n, s support examples C_n^s and q query examples C_n^q are sampled (different from support examples). For testing, a new task T_{new} is constructed which contains new classes C_i, C_j, C_k sampled from D_{test}. (b) Core + Background: Here each keyword sample is mixed with background noise. (c) Core + Optional: An optional class (O) is present along with *Core* classes both during training and testing. (d) Core + Unknown + Background + Silence: Two optional classes i.e. *Unknown* (U) and *Silence* (S) are present and also the samples are mixed with background noise. (Note: In our experiments, the position of optional classes in (c) and (d) is random and not always at the last position as presented in this figure)

3.5 Experiments

3.5.1 Training

To test the effectiveness of our approach, we divide our experiments in four cases (Figure 3.6):

(a) **Core** - Pure Keyword Detection: Both during training and testing, the keyword

[1]`https://github.com/ArchitParnami/Few-Shot-KWS`

samples in the support (S) and query (Q) sets are from *core keywords* and without any background noise.

(b) **Core + Background**: Same as (a), except the keyword samples are now mixed with random background noise.

(c) **Core + Optional**: To account for scenarios when the input query is not from any of the keywords present in the provided support set or when there is simply no input, we train and test in presence of an optional class. This optional class is *unknown keywords* when we want to detect negatives and is *silence* when we want to detect the absence of any spoken keywords.

(d) **Core + Unknown + Silence + Background**: Samples from both the optional classes i.e, *Unknown* and *Silence* are present and are also mixed with background noise. This case simulates more realistic scenarios when input is often mixed with background noise and could be an unknown word or just silence.

In each of the above cases, we train and test in a N-way K-shot manner where N refers to the number of *core* classes and K refers to the number of training examples per class in each episode as explained in Section 2.3.2.1. In cases where an optional class (*Silence* or *Unknown*) is used, we add K support examples for the optional class in the support sets both during training and testing. We perform episodic training as suggested in [40] and train all our models for 200 epochs where each epoch has 200 training episodes and 100 validation (test) episodes. We use SGD with Adam [61] and an initial learning rate of 10^{-3} and cut the learning rate in half every 20 epochs. We conduct experiments with $N = \{2, 4\}$ and $K = \{1, 5\}$ for all the mentioned cases. The model is trained on the loss computed from 5 queries per class in each episode and evaluated more strictly with 15 queries per class during testing.

3.5.2 Baselines

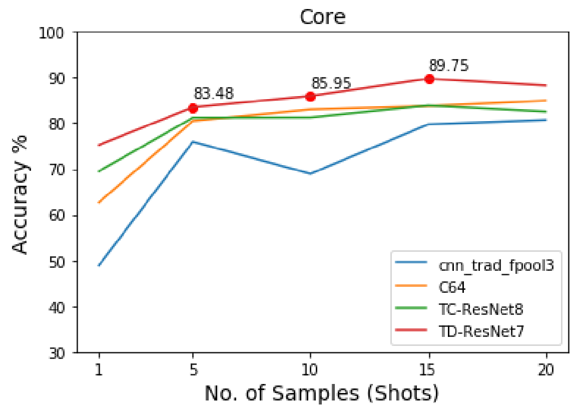
As we formulate and propose a new FS-KWS problem, there is a lack of prior research and a standard FS-KWS dataset. Thus, to show the effectiveness of the proposed framework, we employ three different existing architectures as embedding networks in our FS-KWS framework to examine the performance of the proposed approach. Following are the baseline embedding networks:

- **cnn_trad_fpool3** [50] was originally proposed for KWS problem. It has two convolutional layers followed by a linear, a dense, and a softmax layer. We use the output of the dense layer as network embeddings.
- **C64** [40] is the original 4-layer CNN used in Prototypical Networks for doing few-shot image classification on miniImageNet [18].
- **TC-ResNet8** [56] has demonstrated great results on KWS. We remove the last fully connected and softmax layer and use the remaining architecture as our embedding network in FS-KWS framework.

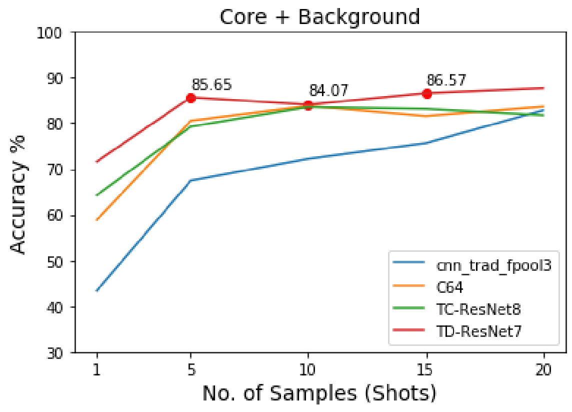
3.5.3 Results

Table 3.2 lists the results for the three baselines and our proposed architecture on experiments mentioned in Section 3.5.1. Given a new 2-way-5-shot KWS task with keywords **not seen** during the training, our TD-ResNet7 model can classify an input query with ∼94% accuracy with the proposed FW-KWS pipeline. This is not even feasible with classical deep learning solutions without FS-KWS formulation.

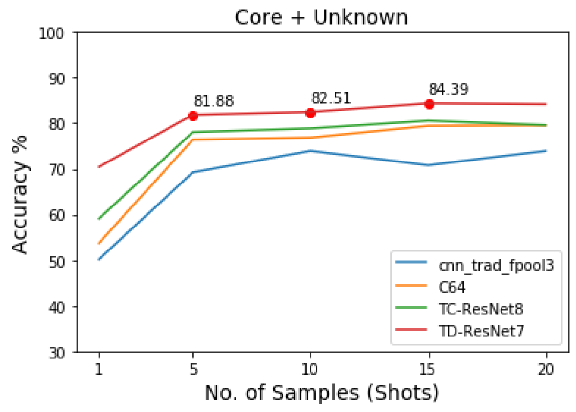
The TD-ResNet7 architecture also outperforms all the existing baselines architectures on all the test cases except in *(b) Core + Background* where the performance of TC-ResNet8 on 2-way 5-shot KWS is slightly better but the difference is not significant ($p = 0.36$ while ANOVA for others presents $p \ll 0.05$). These results are illustrated in Figure 3.7. As we increase the number of shots (samples per class), the



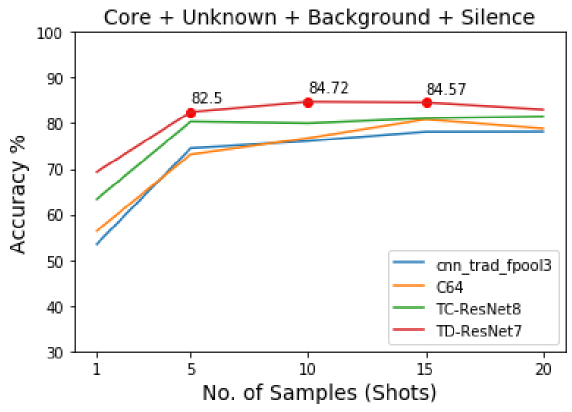
(a) Core



(b) Core + Background



(c) Core + Unknown



(d) Core + Unknown + Background + Silence

Figure 3.7: Comparing test accuracy of embedding network architectures on 4-way FS-KWS as we increase the number of support examples. The results are presented for all the four cases mentioned in section 3.5.1

overall performance improves for all architectures, yet the TD-ResNet7 architecture consistently outperforms other baselines. All the accuracy results are averaged over 100 test episodes and are reported with 95% confidence intervals.

Table 3.2: Performance comparison of different embedding networks when plugged into FS-KWS pipeline for 4 different cases.

Case	**Embedding Network**	**2-way Acc.**		**4-way Acc.**	
		1-shot	5-shot	1-shot	5-shot
core	cnn_trad_fpool3	69.23 ± 0.03	87.07 ± 0.02	48.83 ± 0.02	75.93 ± 0.01
	C64	77.20 ± 0.03	89.97 ± 0.02	62.63 ± 0.02	80.48 ± 0.01
	TC-ResNet8	82.70 ± 0.03	89.00 ± 0.02	69.47 ± 0.02	81.20 ± 0.01
	TD-ResNet7 (ours)	**85.43 ± 0.03**	**94.10 ± 0.01**	**75.22 ± 0.02**	**83.48 ± 0.02**
core + background	cnn_trad_fpool3	69.53 ± 0.04	86.8 ± 0.02	43.3 ± 0.02	67.42 ± 0.01
	C64	78.30 ± 0.03	90.03 ± 0.02	58.83 ± 0.02	80.52 ± 0.01
	TC-ResNet8	77.40 ± 0.03	**91.40 ± 0.02**	64.23 ± 0.02	79.25 ± 0.01
	TD-ResNet7 (ours)	**82.23 ± 0.03**	91.00 ± 0.02	**71.58 ± 0.02**	**85.65 ± 0.01**
core + unknown	cnn_trad_fpool3	58.33 ± 0.03	78.36 ± 0.02	50.15 ± 0.02	69.25 ± 0.01
	C64	63.42 ± 0.03	78.47 ± 0.02	53.69 ± 0.02	76.43 ± 0.01
	TC-ResNet8	68.84 ± 0.03	80.49 ± 0.02	59.08 ± 0.02	78.07 ± 0.01
	TD-ResNet7 (ours)	**77.24 ± 0.02**	**87.22 ± 0.01**	**70.45 ± 0.02**	**81.88 ± 0.01**
core + unknown + background + silence	cnn_trad_fpool3	67.43 ± 0.02	82.32 ± 0.01	53.51 ± 0.02	74.54 ± 0.01
	C64	65.83 ± 0.02	81.15 ± 0.01	56.38 ± 0.01	73.20 ± 0.01
	TC-ResNet8	78.63 ± 0.02	85.98 ± 0.01	63.37 ± 0.02	80.39 ± 0.01
	TD-ResNet7 (ours)	**82.77 ± 0.02**	**89.45 ± 0.01**	**69.34 ± 0.01**	**82.50 ± 0.01**

3.6 Discussion

This work presents a solution for the keyword spotting problem using only limited samples for each keyword. We demonstrate that using prototypical networks with the proposed embedding model which uses temporal and dilated convolutions, can produce significant results with only a few examples. We also synthesize and release a Few-Shot Google Speech command dataset for future research on Few-Shot Keyword Spotting.

This chapter is reused from our work **Few-Shot Keyword Spotting with Prototypical Networks** [62] with permission from authors.

CHAPTER 4: FEW-SHOT CONTINUAL LEARNING

4.1 Motivation

To build truly intelligent systems, we need sustainable models that not only can learn quickly from a small amount of training data but can also continually update themselves when the new data is available. This idea is also referred to as Continual Learning [63] (Fig. 4.1), lifelong or incremental learning. Despite the recent success of neural network models in outperforming human-level performance in tasks such as object recognition [64] and Atari games [65], the current models lack the ability to adapt themselves to new information continuously, i.e., they are inherently static in nature and hence incapable of expanding their function. This is because, when new data is available, a neural network has to be trained again with new and old information altogether. If solely trained with new information, the network will undergo what is known as *catastrophic forgetting* [24] of old information. In recent years, several techniques have been proposed to build continual learning models; however, very few address the real time situation where the available data is limited. With limited data, it becomes extraordinarily difficult to continually develop representations for new data because deep learning models mainly rely on large amounts of training data to remember information. Therefore, we propose to study and address the problem of building continual learning models with limited data, a problem referred to as **Few-Shot Continual Learning**.

4.2 Related Works

Previously, few attempts have been made in addressing continual learning with limited data. [66, 67, 68] propose to address the problem using meta-learning. They

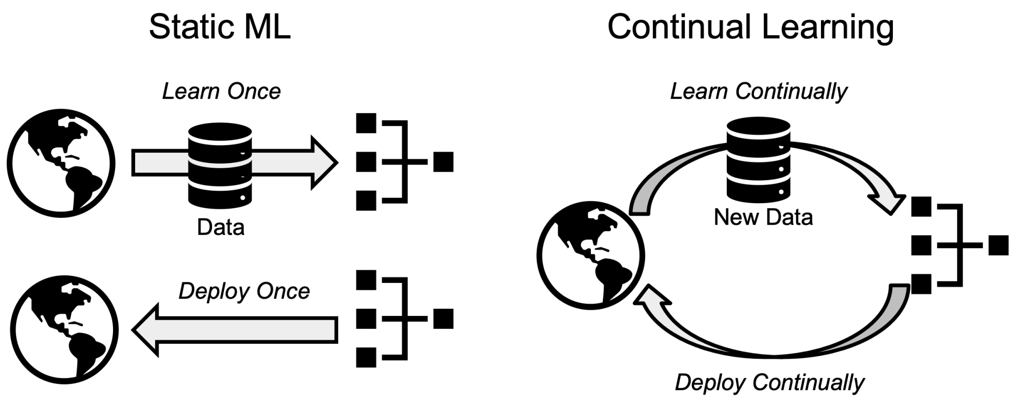


Figure 4.1: Static vs Continual Learning

suggest formulating a distribution of few-shot continual learning tasks and meta-learn representations to solve new few-shot continual learning tasks. **These approaches assume a fixed length task sequence**, i.e., the number of classes to learn is known before and is fixed. This is not a realistic assumption since, at the time of model deployment, it is unlikely to know the number of classes beforehand in the case of online continual learning. Therefore, our study does not limit the number of classes that could be learned by leveraging the idea of one-class classification. To summarize our contribution:

- We develop a method that can learn new classes continuously from a limited number of data samples. The proposed method can have wide practical use in diverse machine learning applications because of growing expense, labor, and restrictions for data collection.

- The proposed method can enable ML practitioners to build sustainable models which can retain new information without forgetting old information and hence will save time and resources that usually go into retraining models.

- Using one-class classification for continual learning ensures that models can theoretically learn a high number of classes and hence are more flexible and adaptable to changing environments. We observe that simply using positive classes for training one-class classifiers is not sufficient for continual learning. Therefore, we incorporate negative sampling for training one-class classifiers and improve the joint class classification performance for continual learning. Additionally, we develop a new method to generate negative samples so that few-shot continual learning can be made possible without explicitly storing samples from other classes to act as negatives when performing the negative sampling.

4.3 Few-Shot Continual Learning Framework

We address the problem of Few-Shot Continual Learning by leveraging meta-learning and one-class classification. The proposed framework continuously learns one-class classifiers where each classifier is a representative of a single positive class. In this framework:

- Meta-learning addresses the issue of learning from limited data and
- Learning an ensemble of one-class classifiers avoids dealing with catastrophic forgetting without restricting the number of classes to be learned.

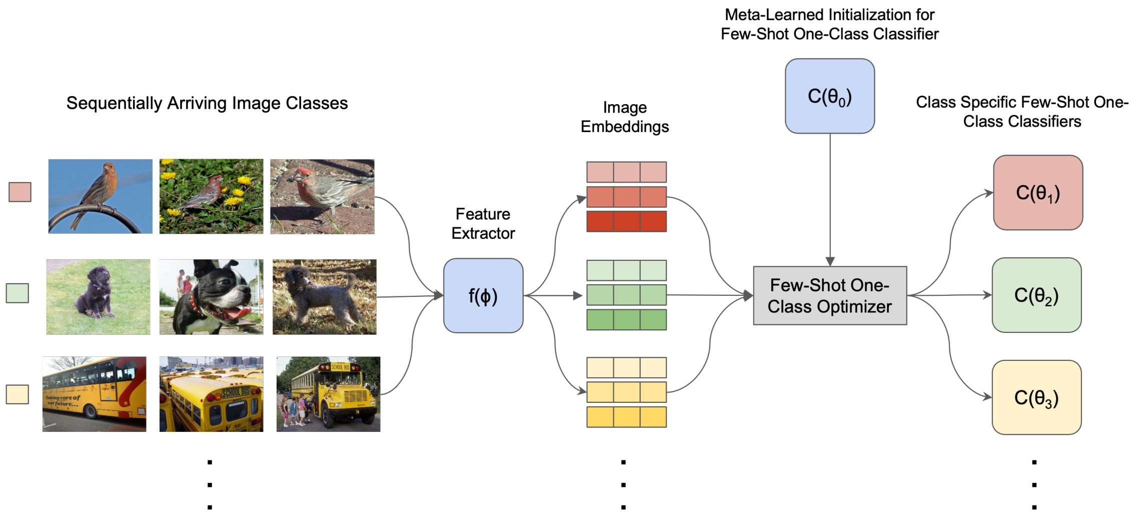

Meta-Learned Initialization for Few-Shot One-Class Classifier
$C(\theta_0)$
Sequentially Arriving Image Classes
Image Embeddings
Class Specific Few-Shot One-Class Classifiers
Feature Extractor
$f(\phi)$
Few-Shot One-Class Optimizer
$C(\theta_1)$
$C(\theta_2)$
$C(\theta_3)$

Figure 4.2: Continually Learning Few-Shot One-Class Classifiers

4.3.1 Training Few-Shot One-Class Classifier for Continual Learning

The pipeline for training is depicted in Figure 4.2. We describe each of the components in the pipeline as follows:

1. **Feature Extraction**: A pretrained feature extractor $f(\phi)$, such as ResNet, is used for generating feature embeddings of the class samples.

2. **Meta-Learning Few-Shot One-Class Classifier**: An initial classifier $C(\theta_0)$ that can be used for few-shot one-class classification can be learned using meta-learning (See Algorithm 2) [69]. However, this initial classifier is not fine-tuned for the detection of a specific class or category.

3. **Few-Shot One-Class Optimizer**: The optimizer finetunes or adapts the parameters of the initial few-shot one-class classifier ($C(\theta_0)$) using the samples of the presented class to obtain a class specific classifier ($C(\theta_i)$). This adaption is quick and can be done using one or two gradient descent steps. This is because the initial meta-learned parameters are close to optimal parameters in the the hypothesis space [8] (See Fig. 4.3):

$$\theta_i = \theta_0 - \alpha \frac{\partial \mathcal{L}_i}{\partial \theta_0}$$

   where $\mathcal{L}_i$ is the binary cross entropy loss of classifier $C(\theta_i)$ on class samples (X_i, Y_i).

4.3.2 Inference by Joint Classification

During inference (Figure 4.4), we first extract the features of the test image and then pass it through each of few-shot one-class classifiers to obtain the positive class score. The classifier with the highest positive class score represents the predicted category.

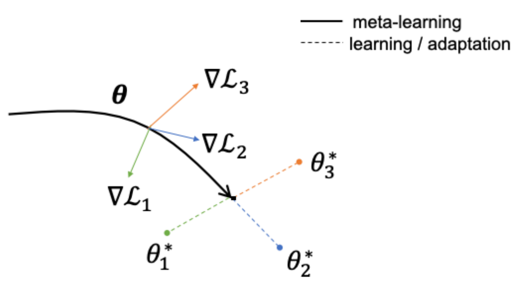


Figure 4.3: Model Agnostic Meta-Learning (MAML) optimizes for a representation θ that can quickly adapt to new downstream tasks.

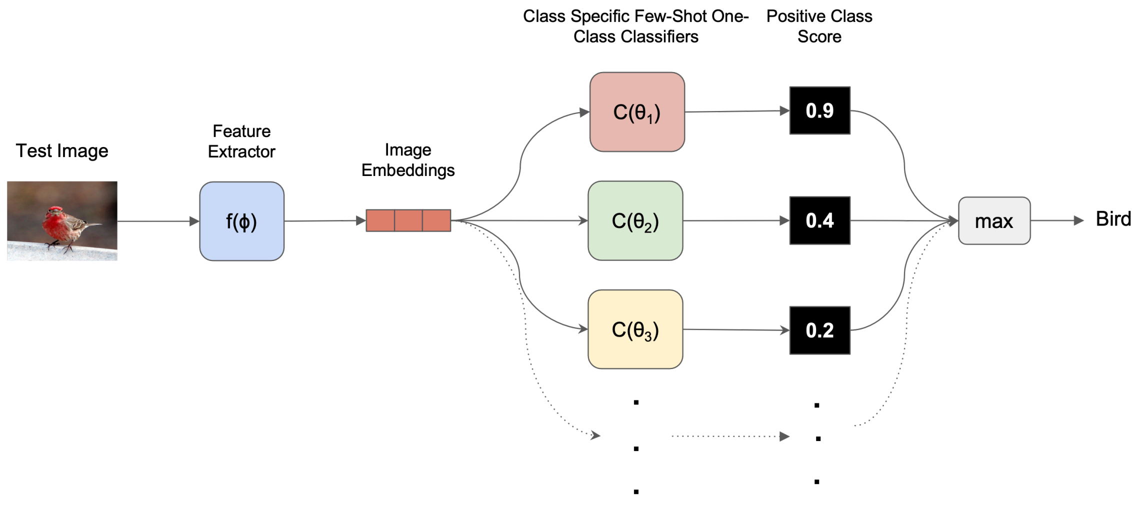


Figure 4.4: Inference by Joint Classification

4.4 Few-Shot Continual Image Classification Dataset

We use MiniImageNet [18] dataset for our experiments following the literature in few-shot continual learning [66, 67, 68]. The dataset contains 100 general object classes where each class has 600 color images. The images are resized to 84×84 and the dataset is split into 64 training, 16 validation, and 20 testing classes. Fig. 4.5 depicts how we split our dataset for training and evaluation of different experiments.

Algorithm 2: Meta-training of OC-MAML (Frikha et al. [69])

```
Require: $S^{tr}$: Set of meta-training tasks
Require: $\alpha, \beta$: Learning rates
Require: $K, Q$: Batch size for the inner and outer updates
Require: $c$: Class Imbalance Ratio (CIR) for the inner-updates
 1: Randomly initialize $\theta_0$
 2: while not done do
 3:     Sample batch of tasks $T_i$ from $S^{tr}$; $T_i = \{D^{tr}, D^{val}\}$
 4:     for each sampled $T_i$ do
 5:         Sample $K$ examples $B$ from $D^{tr}$ such that CIR$= c$
 6:         Initialize $\theta_i' = \theta_0$
 7:         for number of adaptation steps do
 8:             Compute adapted parameters with gradient descent using $B$:
                $\theta_i' = \theta_i' - \alpha \nabla_{\theta_i'} L_{T_i}^{tr}(f_{\theta_i'})$
 9:         end for
10:         Sample $Q$ examples $B'$ from $D^{val}$ with CIR$= 50\%$
11:         Compute outer loop loss $L_{T_i}^{val}(f_{\theta_i'})$ using $B'$
12:     end for
13:     Update $\theta_0$: $\theta_0 \leftarrow \theta_0 - \beta \nabla_{\theta_0} \sum_{T_i} L_{T_i}^{val}(f_{\theta_i'})$
14: end while
15: return meta-learned parameters $\theta_0$
```

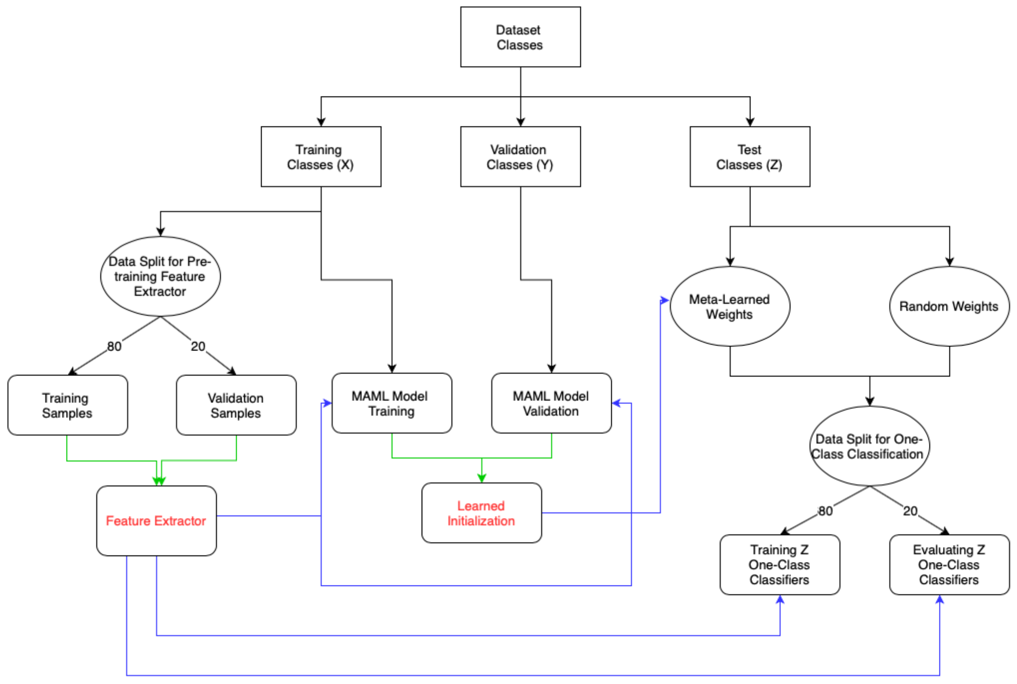
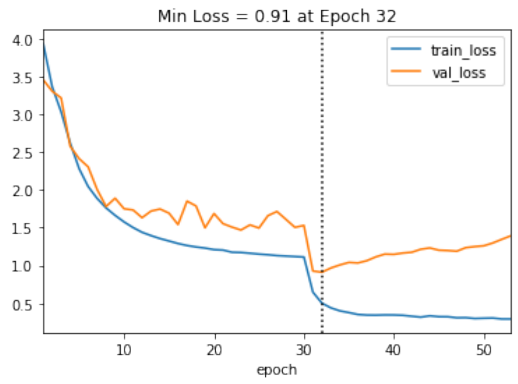


Figure 4.5: Dataset Split

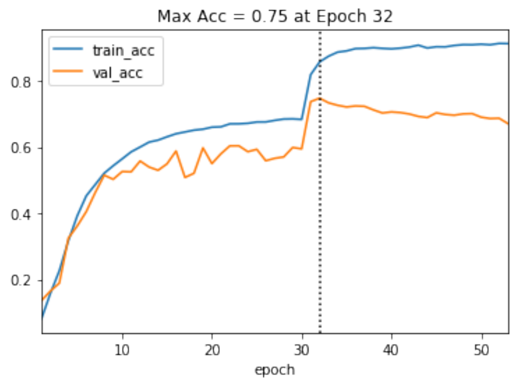
4.5 Experiments

4.5.1 Pre-training ResNet-12 for Feature Extraction

A ResNet-12 [3] is trained on 64 base classes by creating an 80/20 training and validation split. It achieves a maximum accuracy of 75% of the classification task of 64 classes (Fig. 4.6). Then, the last classification layer is removed and the remaining networks are used for feature extraction.



(a) Loss



(b) Accuracy

Figure 4.6: Pre-training ResNet-12

4.5.2 Baseline Experiments

Before we jump into using meta-learning for training one-class classifiers, we simply train one-class classifiers from scratch (random initialization of weights) using the feature embeddings obtained from pretrained ResNet-12. We train 16 one-class classifiers for each of the 16 classes present in the meta-validation set. When training one-class classifiers, there are a number of hyperparameters to experiment with (Table 4.1). Next, we can measure the performance of individual one-class classifier based on these hyperparameters and we can also measure the joint classification performance of all the classifiers on the task of classifying an input image from one of 16 classes (results depicted in Fig. 4.8 as baseline).

Table 4.1: Training Hyperparameters of a One-Class Classifier

Hyperparameter	**Description**
K	The number of positive class samples used for training the one-class classifier
M	The number of negative classes used to sample negative training examples
N	The number of epochs (adaptation steps) the classifier is trained on
K/M	Number of negative examples per negative class

4.5.3 Meta-Learning Experiments

A derivative of OC-MAML [69] is used for learning initial parameters of the one-class classifier $C(\theta_0)$. **Our experiments differ by the fact that we make use of negative samples to improve the performance of the one-class classifier**. Our meta-learning experiments are divided into following scenarios (Fig. 4.7).

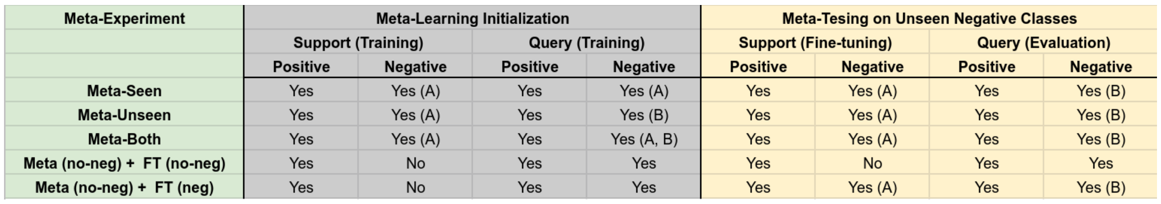
1. **Meta-seen**: When query samples of negative class are drawn from the same classes used in the support samples for the negative class.

2. **Meta-unseen**: When query samples of negative class are drawn from classes not seen in the support set for negative class.

3. **Meta-both**: When query samples of negative class are drawn from both the classes seen and the classes not seen in the support set for negative class.

4. **Meta (no-neg) + FT (no-neg)**: The meta-learned initialization was obtained without using negative samples in the support set (OC-MAML with $c = 0$). The initialization is finetuned with support with only positive samples and no negative samples when training the one-class classifier.

5. **Meta (no-neg) + FT (neg)**: The meta-learned initialization (learned from positive samples only) is finetuned with a support set having both positive and negative samples.



Meta-Experiment	Meta-Learning Initialization				Meta-Tesing on Unseen Negative Classes			
	Support (Training)		Query (Training)		Support (Fine-tuning)		Query (Evaluation)	
	Positive	Negative	Positive	Negative	Positive	Negative	Positive	Negative
Meta-Seen	Yes	Yes (A)	Yes	Yes (A)	Yes	Yes (A)	Yes	Yes (B)
Meta-Unseen	Yes	Yes (A)	Yes	Yes (B)	Yes	Yes (A)	Yes	Yes (B)
Meta-Both	Yes	Yes (A)	Yes	Yes (A, B)	Yes	Yes (A)	Yes	Yes (B)
Meta (no-neg) + FT (no-neg)	Yes	No	Yes	Yes	Yes	No	Yes	Yes
Meta (no-neg) + FT (neg)	Yes	No	Yes	Yes	Yes	Yes (A)	Yes	Yes (B)

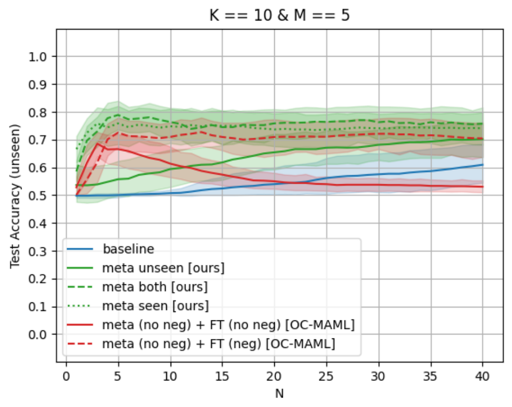
Figure 4.7: Meta-Learning Experiments. A and B represents two disjoint sets from which negative examples are sampled. **Yes** → Sampled. **No** → Not Sampled.

4.5.4 Training Hyperparameters

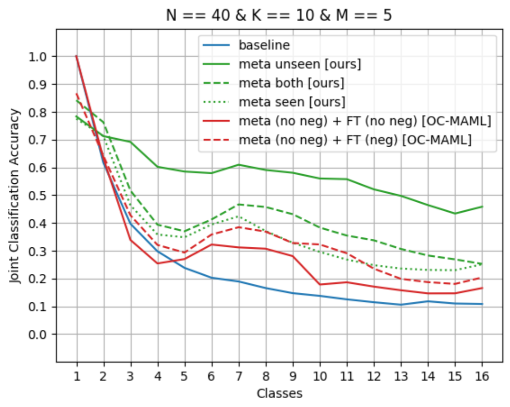
We experiment with a multi-layer perceptron with the following architecture as our one-class classifier: Linear (640, 320) → ReLU → Linear (320, 160) → ReLU → Linear (160, 1) → Sigmoid. During the training of meta-learning experiments, we set the outer loop learning rate to 0.001 and the inner loop learning rate to 0.01. The number of adaptation steps is 5 and the number of tasks in each episode is 8. We train for 200 epochs and perform early stopping with patience 40. The number of positive (K) (or negative training examples) used is 10. The number of negative classes (M) used to sample negative examples is 5. For evaluation, we test 30 query (Q) samples from a positive class and 30 from negative classes (split equally from M classes). For baseline experiments, the learning rate is also 0.01. We present the results with adaptation steps up to 40 (N) at the test.

4.5.5 Results

Fig. 4.8 presents the test accuracy results for the baseline and meta-learning experiments discussed above. In Fig. 4.8 (a), we report the average test accuracy of our one-class classifier (computed using 16 classes) on the y-axis with the number of adaptation (fine-tuning) steps on the x-axis. The baseline approach which trains each OCC from scratch using random initialization naturally does poorly. Whereas, the meta-learning based approaches show better results in just a few adaption steps. Moreover, we notice that our meta experiments (meta both and meta seen) which involve meta-training with negative samples have higher test accuracy on average than OC-MAML [69] based methods (meta (no-neg)) which do not use negative samples during meta-training. Interestingly, OC-MAML when fine-tuned with negative samples (meta (no neg) + FT (neg)), continues to do better than OC-MAML only adapted with positive samples (meta (no neg) + FT (no neg)), as we increase the number of adaptation steps. The meta-unseen method adapts slowly and matches the OC-MAML (FT (neg)) performance at 40 adaptation steps. Overall, these results indicate that on average meta-learning one-class classifiers with negative sampling contributes to higher classification accuracy.



(a) Average OCC Accuracy



(b) Joint Classification Accuracy

Figure 4.8: Test Accuracy Results

Next, we report the performance on few-shot continual learning in Fig. 4.8 (b). As depicted in Fig. 4.2, we continuously train and add new one-class classifiers to our collection when images from a new class are presented. Then, we measure the performance of classification of the classifiers present in our collection by presenting test images from each class (as depicted in Fig. 4.4). Fig. 4.8 (b) presents the results for such scenario. On the x-axis, new classes are added, i.e., a new OCC is trained and on the y-axis the joint classification accuracy is reported on all the classes (OCC's) witnessed so far. Each OCC is trained up to 40 adaptation steps using 10 positive samples and 10 negative samples (where applicable). We observe that our meta-experiments (meta-unseen, meta-both, and meta-seen) which use negative sampling while meta-training, outperforms both the baseline and OC-MAML based experiments on few-shot continual learning. Moreover, the meta-unseen method has exceedingly better joint classification accuracy than the meta-seen and meta-both methods. As before in Fig. 4.8 (a), our results in Fig. 4.8 (b) again demonstrate the importance of using negative samples for meta-training one-class classifiers for the downstream task of few-shot continual learning.

So far, our results have convinced us that using negative samples for training one-class classifier is necessary for better few-shot continual learning. However, where should we obtain these negative samples when training each of OCCs? In our experiments, we assumed that samples from other classes can be stored in memory and hence are available to act negatives for training the classifier at hand. Here we are in violation of our true continual learning objective, i.e., to remember new information without forgetting old information and **without having access to old information**. Therefore, in the next section (Sec 4.6), we focus on eliminating the need for storing samples and develop a method to generate negative samples so that continual learning can be done without storing any positive class samples to act as negative samples to other classes when training one-class classifiers.

4.6 Generating Negatives for Few-Shot One-Class Classification

4.6.1 Motivation

As observed in our results in the previous section (Sec. 4.5.5), using negative samples for training one-class classifiers enables better joint classification performance on few-shot continual learning. However, for continually learning new classes we don't have negative samples unless we store some positive samples for each learned class. Hence resulting in ever expanding memory size. Our goal in this research is to perform continual learning without explicitly storing any samples in the memory. Therefore to this end, we propose a generative model that transforms positive samples into negative samples for training the one-class classifier. After which both sets of samples can be discarded. In the following section, we detail our choice of the generative model.

4.6.2 The Generator: Negative Self-Attention Model

Attention models [70] have been commonly used for natural language processing tasks [71]. There are many variations of attention models such as self-attention, global and local attention. Particularly, self-attention models are responsible for finding out the most important features in a given set of features. Our previous research [72] has shown that self-attention model can be used to transform a given set of features (embeddings) from one dimension to another. Motivated by our findings, we deploy a modified self-attention model to transform embeddings from positive dimensions to the embedding present in the dimension space of negative samples. We achieve this by assigning higher weights to less important features and lower weights to more important features, hence moving away from the features that make a sample positive. Algorithm 3 describes the implementation for our negative self-attention model, which takes positive class embeddings as input and outputs negative embeddings.

Algorithm 3: Negative Self-attention

```
Function NegativeSelfAttention($E_{n\times m}$, d)
    $K_{n\times d}$ = affine(E, d)                                    ▷Keys
    $Q_{n\times d}$ = affine(E, d)                                    ▷Queries
    $logits_{n\times n}$ = matmul(Q, transpose(K))
    $scores_{n\times n}$ = 1 - Softmax(logits, dim=1)                 ▷Reversing Probability
    $values_{n\times d}$ = affine(E, d)
    $output_{n\times d}$ = matmul(scores, values)
    return output
```

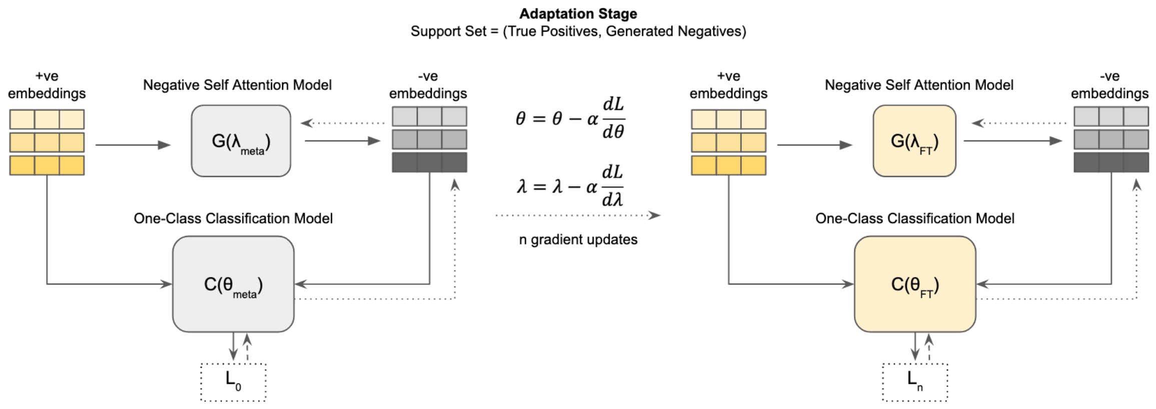


Figure 4.9: Adaptation Stage: Model adapts from meta-learned initialization using positive class samples and correspondingly generated negative samples.

4.6.3 Using Negative Self-Attention Model for Training One-Class Classifier

Our objective is to learn good initial parameters (θ_{meta}) for the one-class classifier such that it can adapt from positives and generated negative samples quickly and can solve the downstream classification task. To make this possible, we also need to find initialization for generator parameters (λ_{meta}) such that it can generate negative embeddings from the positive embeddings of the given class. Once we have the meta parameters (θ_{meta} and λ_{meta}), deploying a one-class classifier for a specific class becomes a two stage process:

1) During the adaptation stage (Fig. 4.9), the positive class samples for a class are fed into the generator model that outputs negative embedding. We then measure the loss of classification using the meta parameters of the classifier. This loss is used to update both the negative self-attention model and the classifier's meta parameters to obtain the finetuned parameters (θ_{FT} and λ_{FT}). The idea is that the attention model should generate negative samples good enough for the classifier to find a good decision boundary in a few adaptation steps. 2) During the inference stage (Fig. 4.10), the adapted model can be used for the classification of true positives and true negative samples.

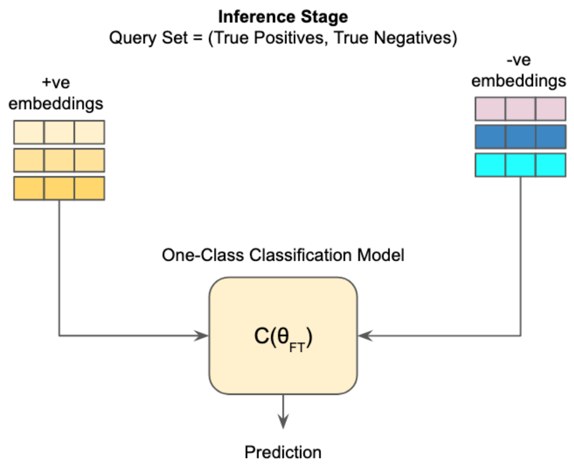

Inference Stage
Query Set = (True Positives, True Negatives)
+ve embeddings
-ve embeddings
One-Class Classification Model
C(θ_FT)
Prediction

Figure 4.10: Inference Stage: The fine-tune model is used for inference of true positives and true negatives.

4.6.4 Learning Meta-Parameters: Experiments

We approach generating negative samples using self-attention model by conducting three strategic meta-learning experiments listed below:

1. **meta-atten**: In this experiment, we only learn the initialization parameters θ_{meta} and not λ_{meta}.

2. **meta-atten with MSE loss**: Here again, we only learn the initialization parameters θ_{meta} and not λ_{meta}. But this time, we penalize the generator model by adding the **inverse** of the Mean Squared Error (MSE) computed between the **generated negative embeddings** and the **positive embeddings** to the classification loss, as a way to make them as far away as possible in the embedding space.

3. **meta-atten (G) with MSE loss**: In this experiment, we meta-learn the initialization parameters for **both** the one-class classifier (θ_{meta}) and the generator model (λ_{meta}) by computing the MSE between **generated negative embeddings** and **true negative embeddings**.

We discuss the above three experiments in detail in the following subsections.

4.6.4.1 Learning θ_{meta}

To learn the initialization parameters θ_{meta} for the one-class classifier, we follow the meta-training [8] procedure on our custom defined tasks as outlined below. We first initialize θ_{meta} randomly. Then, during the **inner loop training** (Fig. 4.11), we sample a batch of tasks (m), where each task is aimed at learning a one-class classifier for the given positive class embeddings and the generated negative embeddings from the generator model (Sec 4.6.2). By calculating the loss of classification on the support set (positive embeddings, generated negative embeddings), we update both classifier parameters and the generator parameters. This process is repeated for n adaptation steps for each task in the batch sampled in the outer loop i as follows:

$$\theta_0^i = \theta_{meta}, \tag{4.1}$$

$$\theta_n^i = \theta_{n-1}^i - \alpha \frac{dL_{n-1}}{d\theta_{n-1}^i}, \tag{4.2}$$

$$\lambda_n^i = \lambda_{n-1}^i - \alpha \frac{dL_{n-1}}{d\lambda_{n-1}^i}, \tag{4.3}$$

where α is the inner loop learning rate.

In the **outer loop** (Fig. 4.12), the adapted parameters of each one-class classifier (θ_n) are used to measure the classification performance on query set which contains true unseen positive class examples and true unseen negative class examples. The loss of classification is measured using binary cross entropy loss L_{test}. The total loss on the query sets is then used to update initial meta parameters θ_{meta}:

$$L_{total} = \sum_{k=1}^{k=m} L_{test}^k, \tag{4.4}$$

$$\theta_0^{i+1} = \theta_0^i - \beta \frac{dL_{total}}{d\theta_0^i}, \tag{4.5}$$

$$\theta_{meta} = \theta_0^{i+1}, \tag{4.6}$$

where β is the outer loop learning rate.

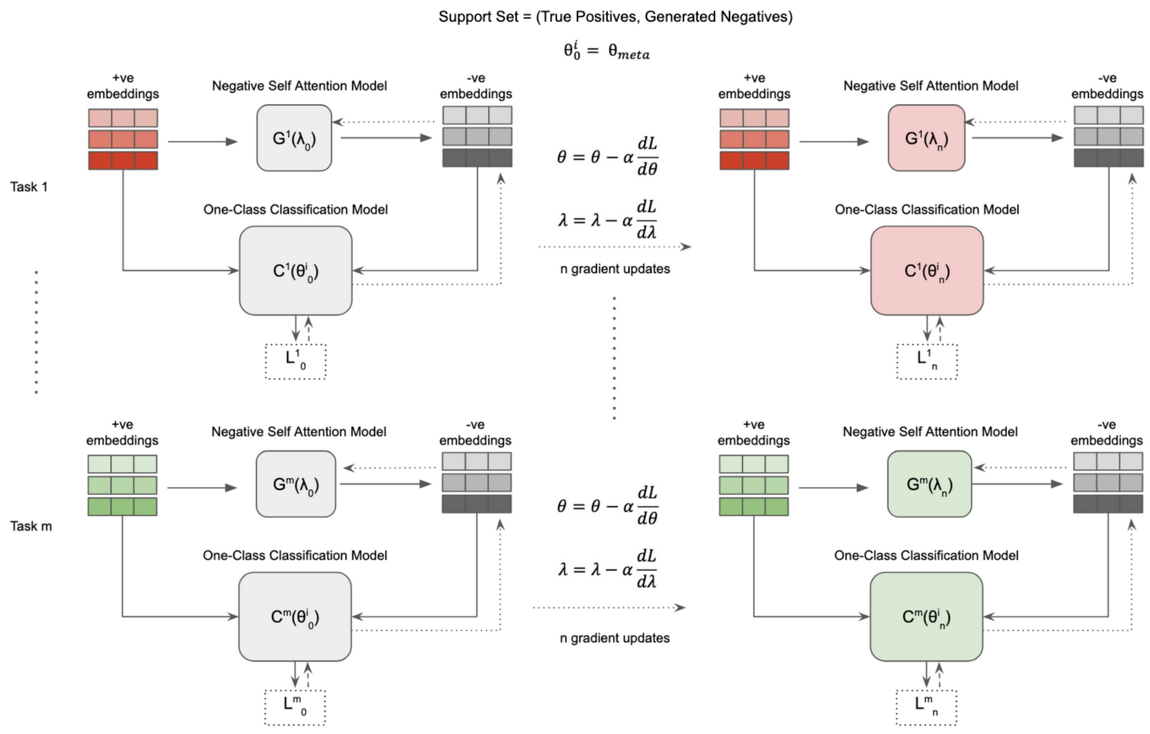

Support Set = (True Positives, Generated Negatives)
$\theta_0^i = \theta_{meta}$
Task 1
Task m
+ve embeddings
-ve embeddings
Negative Self Attention Model
One-Class Classification Model
$G^1(\lambda_0)$
$C^1(\theta^i_0)$
L^1_0
$G^1(\lambda_n)$
$C^1(\theta^i_n)$
L^1_n
$G^m(\lambda_0)$
$C^m(\theta^i_0)$
L^m_0
$G^m(\lambda_n)$
$C^m(\theta^i_n)$
L^m_n
$\theta = \theta - \alpha \frac{dL}{d\theta}$
$\lambda = \lambda - \alpha \frac{dL}{d\lambda}$
n gradient updates

Figure 4.11: Inner Loop Training

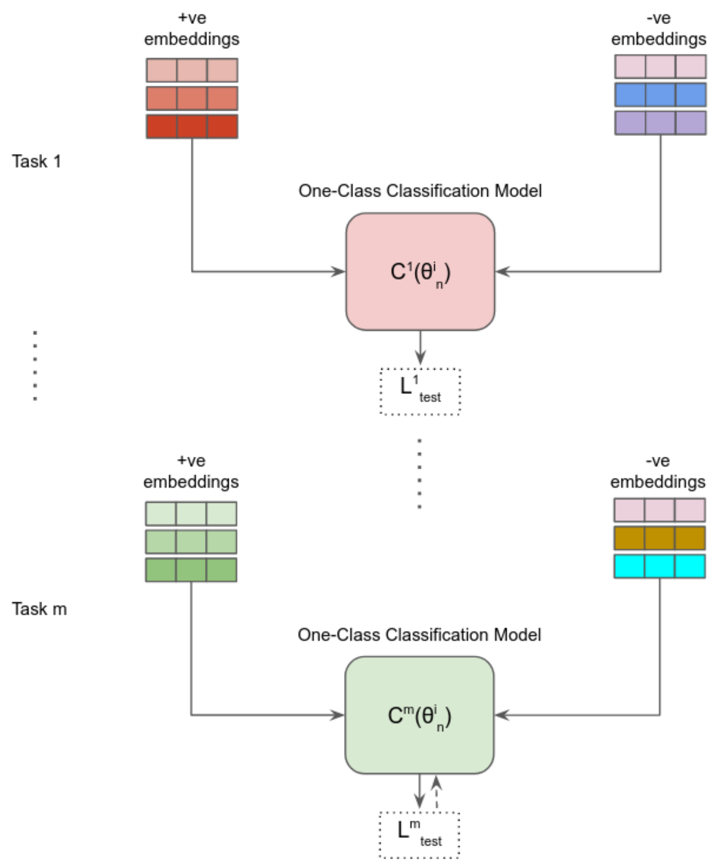


Figure 4.12: Outer Loop Training

4.6.4.2 Learning θ_{meta} with MSE loss

In this experiment, we follow the similar training process as discussed in Sec. 4.6.4.1, except that during the **inner loop** training, we account for the similarity between the generated negative embeddings and the positive embeddings. Since we want these two embeddings to be dissimilar, we measure the Mean Squared Error (denoted by E) between them and add its inverse to classification loss (L) to get the total loss (T) in the inner loop. This is depicted in Fig. 4.13.

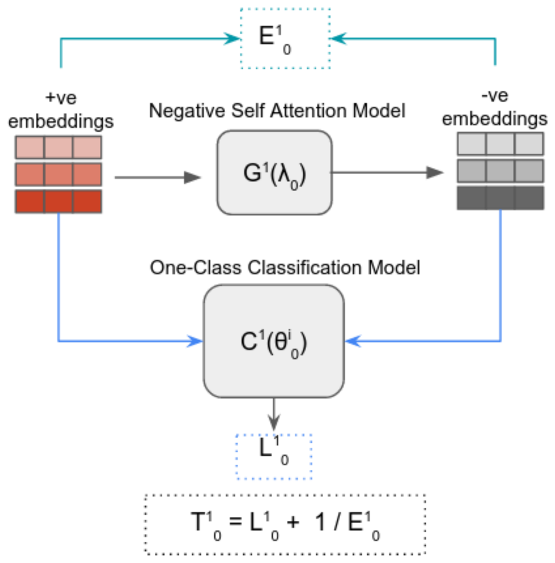


Figure 4.13: Adding MSE in Inner Loop Training

The total loss (T) is used to update the classifier and the generator parameters as follows:

$$\theta_0^i = \theta_{meta}, \tag{4.7}$$

$$\theta_n^i = \theta_{n-1}^i - \alpha \frac{dT_{n-1}}{d\theta_{n-1}^i}, \tag{4.8}$$

$$\lambda_n^i = \lambda_{n-1}^i - \alpha \frac{dT_{n-1}}{d\lambda_{n-1}^i}. \tag{4.9}$$

The **outer loop** training process is exactly the same as illustrated in Fig. 4.12 and, therefore, θ_{meta} is obtained using Eqn. (4.4), (4.5), and (4.6).

4.6.4.3 Learning θ_{meta} and λ_{meta} with MSE loss

In this experiment, we learn the initialization parameters θ_{meta} for the one-class classifier and λ_{meta} for the negative self-attention model. We follow the similar training procedure on our custom defined tasks as discussed in Sec 4.6.4.1. The **inner loop** training process is same, except that this time, we also initialize λ, similar to Eqn (4.1) as

$$\lambda_0^i = \lambda_{meta}. \tag{4.10}$$

Then, the inner loop training proceeds as per Eqn. (4.2) and (4.3).

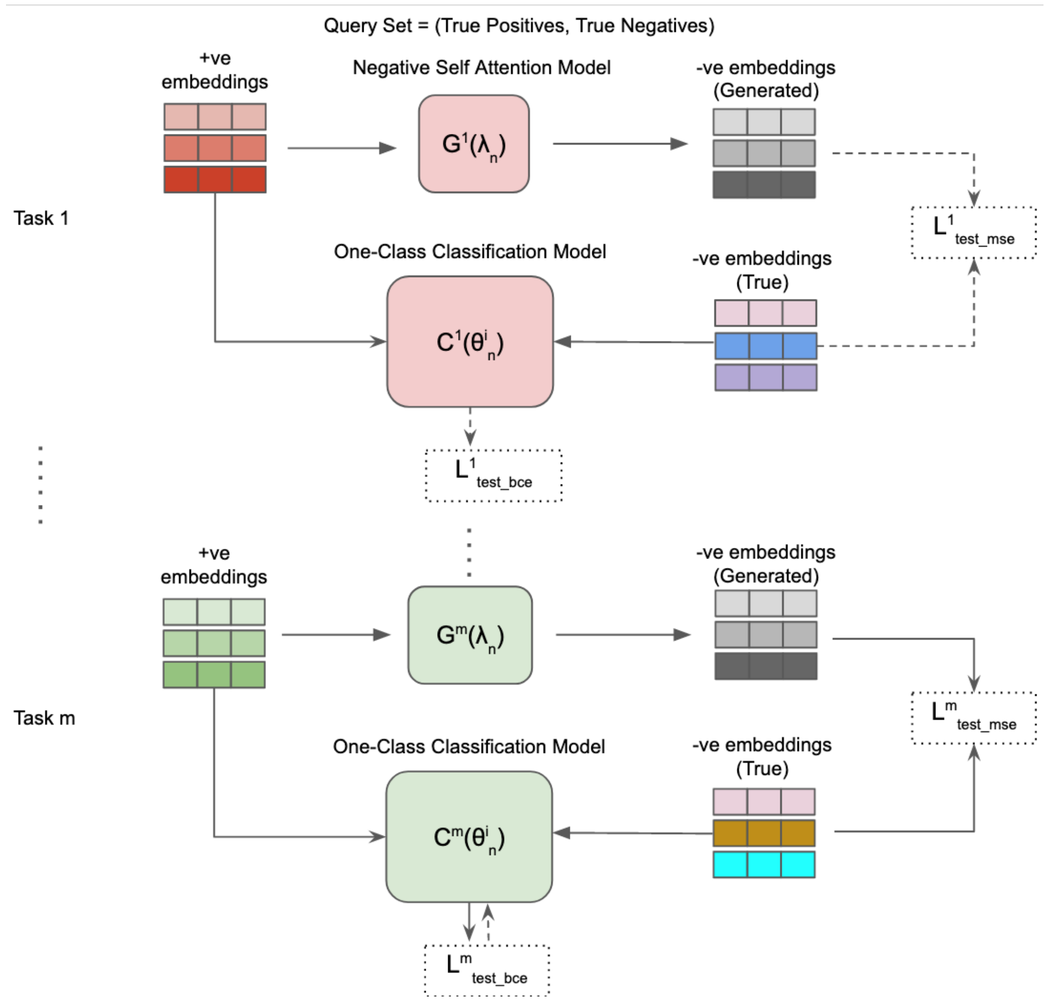


Figure 4.14: Outer Loop Training with MSE

In the **outer loop** (Fig. 4.14), the adapted parameters of each one-class classifier (θ_n) are used to measure the classification performance on query set which contains true unseen positive class examples and true unseen negative class examples. The loss of classification is measured using binary cross entropy loss L_{test_bce}. At the same time, the adapted parameters of each generator model (λ_n) are used to measure the quality of generated embeddings by calculating the mean squared error L_{test_mse} between the **generated negative embeddings** and **true negative embeddings** in the query set. The total loss on the query sets is then used to update meta parameters θ_{meta} and λ_{meta}:

$$L_{total} = \sum_{k=1}^{k=m} L^k_{test\_bce} + L^k_{test\_mse}, \tag{4.11}$$

$$\theta_0^{i+1} = \theta_0^i - \beta \frac{dL_{total}}{d\theta_0^i}, \tag{4.12}$$

$$\lambda_0^{i+1} = \lambda_0^i - \beta \frac{dL_{total}}{d\lambda_0^i}, \tag{4.13}$$

$$\theta_{meta} = \theta_0^{i+1}, \tag{4.14}$$

$$\lambda_{meta} = \lambda_0^{i+1}. \tag{4.15}$$

4.6.5 Results

In this section, we present the results for the above meta-learning experiments (Sec. 4.6.4) conducted with generated negatives using the negative self-attention model in comparison to the meta-learning experiments done with true negatives (Sec. 4.5.3) for few-shot continual learning.

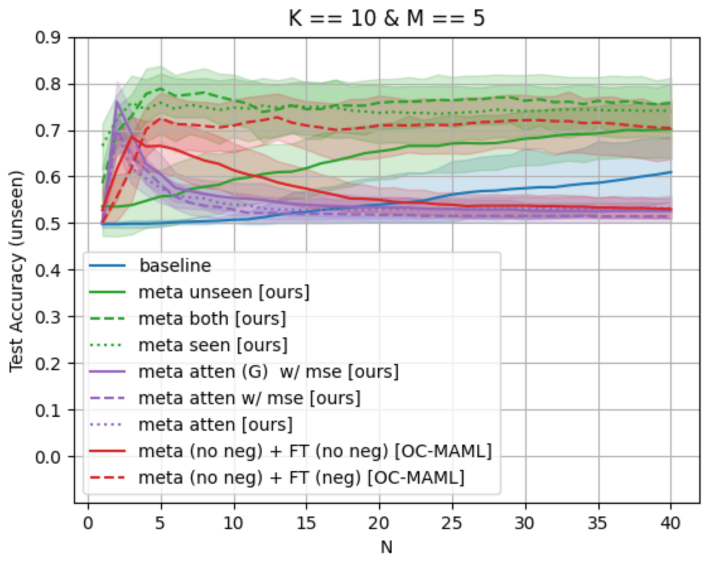


Figure 4.15: Average One-Class Classification Accuracy

Fig. 4.15 reports the average one-class classification accuracy of the 16 one-class classifiers trained with 10 positive samples and 10 negative samples (where applicable). We observe that all the meta-learned models trained with generated negatives (in purple) adapt quickly (3 adaptation steps) and have higher average classification accuracy when compared to baseline (blue) and OC-MAML (meta (no neg)) based models. Particularly, the meta model (meta-atten (G) w/ mse) which also learns initialization for generator parameters (λ_{meta}) performs best and even surpasses the meta-models learned with true negatives (meta-unseen, meta-seen, and meta-both) in 3 adaption steps. However, as expected, on further adaptation with generated negatives, the meta-models (in purple) start to overfit and their performance (measured on true negatives) declines. This behavior is similar to OC-MAML finetuned without negative samples (meta (no neg) + FT (neg)).

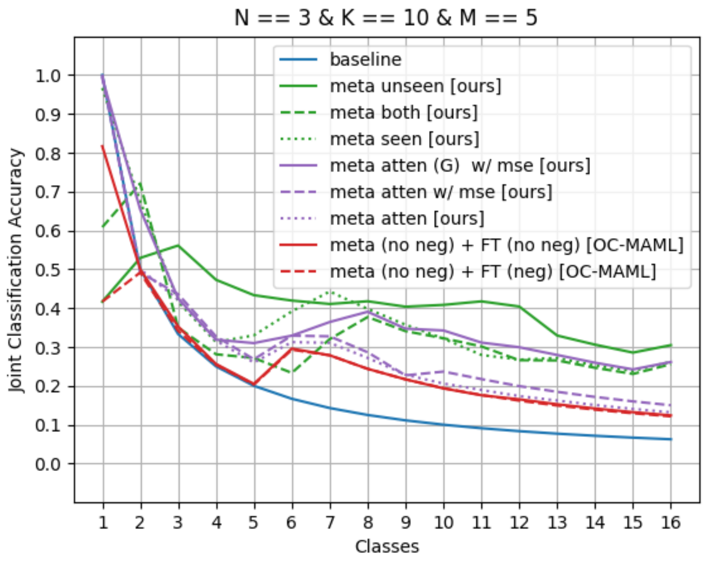


Figure 4.16: Joint Classification Accuracy (MLP)

In Fig. 4.16, we compare the joint classification accuracy of meta-models trained with generated negatives (in purple) vs those with true negatives (green) and no negatives (red) for the task of few-shot continual learning. To best analyze the capability of meta-models that use generated negatives, we perform the comparison at 3 adaption steps. As in Fig. 4.15, we again observe that meta-models using generated negatives (in purple) perform much better than baseline (blue) and OC-MAML based (in red) methods on the task of few-shot continual learning. Interestingly, our meta-model trained with generated negatives (meta-atten (G) w /mse) matches upto the performance of models trained with true negatives (meta-both and meta-seen) but tails the best performing model (meta-unseen).

4.7 Discussion

Our experiments in Sec. 4.5.5 show that the use of negative sampling for training one-class classifiers is beneficial for the downstream task of few-shot continual learning. However, to continually learn such new one-class classifiers, one requires to store samples of the classes witnessed so far to act as negatives to new classes. This defeats the idea of retaining new information without explicitly storing any. Therefore, to achieve our goal of performing continual learning without explicitly storing class samples, we devise a method to generate negative samples from the given class positives to act as a proxy for true negatives. Our results show that this indeed results in performance that is better than baseline and the OC-MAML classifiers which do not make use of negative sampling and also closely match up to methods that use real negatives. However, there still exists some gap between the performance of generated negatives vs true negatives for the purpose of few-continual learning. So far we have managed to close this gap by coming up with incrementally better meta models to generate negatives. Continuing in this direction, future work will focus on finding much better models to bridge the gap. Moreover, by having a fixed memory size and allocating it to store a few positive class samples to act as negatives to other classes, in addition to the generated negatives, will further help in making more realistic and better few-shot continual learning models.

CHAPTER 5: PRIVACY-ENHANCED FEW-SHOT LEARNING

5.1 Motivation

There has been a widespread adoption of cloud-based machine learning platforms recently, such as Amazon Sagemaker [73], Google AutoML [74], and Microsoft Azure [75]. They allow companies and application developers to easily build and deploy their AI applications as a Service (AIaaS). However, the users of AIaaS services may encounter two major challenges. 1) **Large Data Requirement**: Deep Learning models usually require large amounts of training data. This training data needs to be uploaded to the cloud services for the developers to build their models, which may be inconvenient and infeasible at times. 2) **Data Privacy Concerns**: Sharing data with untrusted servers may pose threats to end-user privacy. For instance, a biometric authentication application deployed in the cloud will expose user photos to a third-party cloud service.

To address the *large data requirement* problem, there has been increasing research on the approaches that require less amount of training data, popularly known as Few-Shot Learning [76]. Specifically, metric-based few-shot classification methods [18, 35, 40, 41, 42] learn to map images of unseen classes into distinct embeddings that preserve the distance relation among them and then perform classification of the input query image by the distance to the class embeddings. Recent works have been able to achieve up to ∼90% accuracy on the challenging task of 5-way 5-shot classification on the MiniImageNet dataset [77]. Despite the success and promises of few-shot learning, it is imperative to address the *data privacy concerns* to protect user-supplied sensitive data, e.g., when a metric-based few-shot model is deployed in a cloud server (Fig. 5.1).

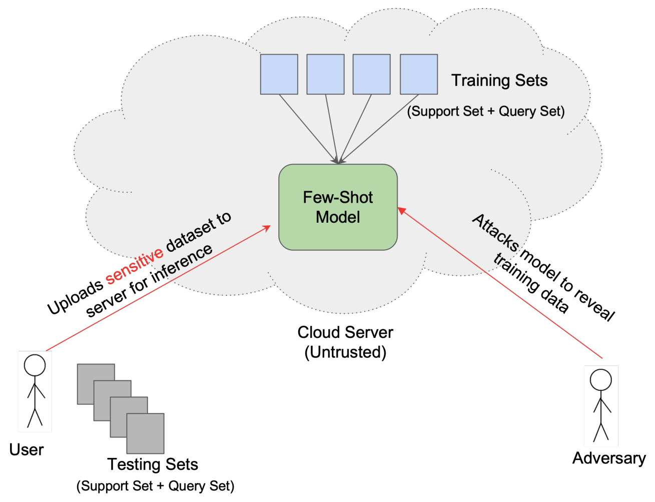

Training Sets
(Support Set + Query Set)
Few-Shot Model
Uploads sensitive dataset to server for inference
Attacks model to reveal training data
Cloud Server (Untrusted)
User
Testing Sets
(Support Set + Query Set)
Adversary

Figure 5.1: Threats in a cloud-based few-shot model. 1) attacks on training data [1, 2] and 2) exposure of sensitive dataset to untrusted cloud server for inference.

Several privacy-preserving approaches may be adopted in machine learning applications, including cryptography, differential privacy, and data obfuscation. Recent works [78] adopted cryptographic techniques to protect the confidentiality of data. For example, remote machine learning services can provide results on encrypted queries [78]; a range of primitives, such as Homomorphic Encryption, may be adopted to manage the encrypted data. Despite promising results, crypto-based methods inflict high computational overheads, creating challenges for practical deployment. Furthermore, such solutions may breach privacy by disclosing the exact computation results, and an adversary may utilize the model's output to launch inference attacks on training data [1, 2]. Differential privacy [79] has been adopted to train machine learning models while providing indistinguishability guarantees for individual records in the training set [80]. However, the strong privacy guarantees tend to reduce the model performance and have shown disparate impacts on the underrepresented classes [81]. In contrast, data obfuscation methods achieve privacy protection without inflicting high computational costs, e.g., image blurring and pixelization. Obfuscation can be applied to protecting both training and testing data, and can provide differential privacy guarantees at individual-level data [82].

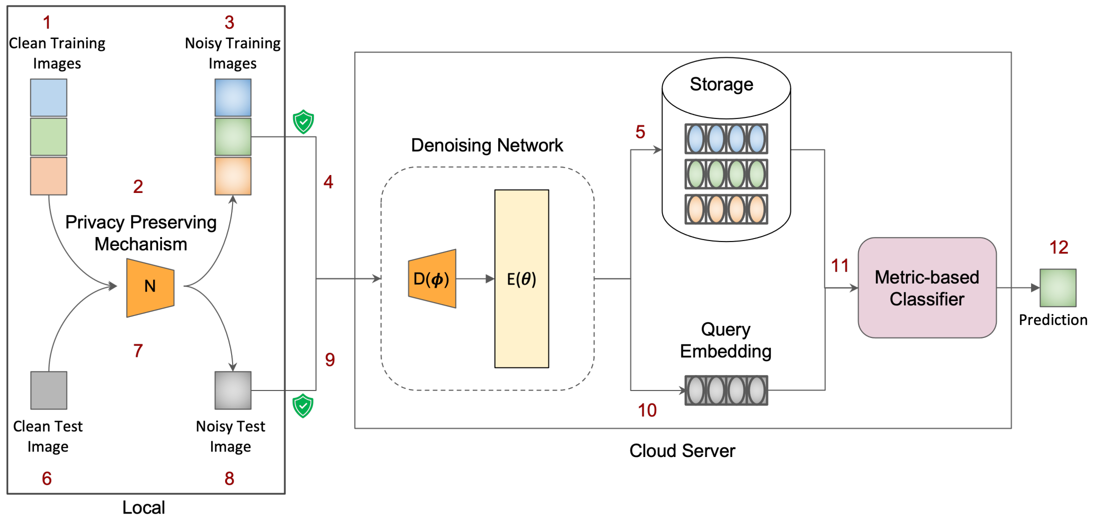


Figure 5.2: Few-Shot Private Image Classification in the Cloud: A denoising network is first trained with non-user data and deployed in the cloud. Using a privacy preserving method (2), a user can obfuscate clean training images (1) to obtain noisy training images (3). These images are then sent to the cloud server where they are first denoised and then encoded (4) to be stored as privacy-preserved embeddings on the server (5). A user can obfuscate the clean test image (6) and query the server using a noisy test image (8) to obtain a prediction (12).

This work focuses on the privacy of testing data (support+query) specifically for few-shot learning. A few-shot model built for clean images exhibits poor performance when tested with noisy/private image data. This is because meta-learning based few-shot models do not work well with out-of-distribution tasks [8, 37, 40]. Therefore, applying the obfuscation methods to the image data and simply using an off-the-shelf pre-trained few-shot model leads to degradation in performance, as observed in our experiments (Fig. 5.6 Baseline Model). Hence, it is imperative to study privacy specifically in context of few-shot learning. To this end, we suggest a private few-shot learning approach trained on noisy data samples as illustrated in Fig. 5.2. Adopting an obfuscation mechanism on the local input data samples, a user transfers privacy-encoded data to the cloud. The proposed jointly-trained, denoised embedding network, the *Denoising Network*, constructs privacy-preserved latent space for robust few-shot classification. To validate the proposed approach, we examine four privacy methods including traditional obfuscation methods such as Pixelization and Blurring, which do not provide quantifiable privacy guarantees [83], and also Differentially Private Pixelization (DP-Pix) [82] which provides differential privacy guarantees.

This study examines practical implications for a holistic private few-shot learning framework on an untrusted service platform, which has not been studied previously. Thus, our main contributions are: 1) first proposing a unified framework for deploying few-shot learning models in the cloud while protecting the privacy of user-supplied sensitive data and 2) thoroughly examining privacy methods on three different datasets of varying difficulty, therefore 3) discovering and observing the existence of the effective privacy-preserved latent space for few-shot learning.

5.2 Related Works

Xie et al. [84] incorporate differential privacy into few-shot learning through adding Gaussian noise into the model training process [80] to protect the privacy of training data. [85, 86] have also provided a strong privacy protection guarantee in pairwise learning for training data. On the other hand, [87] propose to use hashing to store the embedding of the input images. Similar to cryptographic approaches [78], the work [87] incurs high computational complexity to achieve accuracy. Differently, our approach addresses the privacy of user data at source (i.e., the images are already privatized before the server sees them) with strong privacy protection. To the best of our knowledge, ours is the only approach that addresses privacy in the context of few-shot metric learning for user-supplied training and testing data.

5.3 Privacy Methods

We study following methods to introduce privacy in images (depicted in Fig. 5.3).

5.3.1 Independent Gaussian Noise

Introducing some noise in an image is one way to distort information [88]. Kim [89], first publicized the work on additive noise by the general expression $Z = X + \epsilon$,

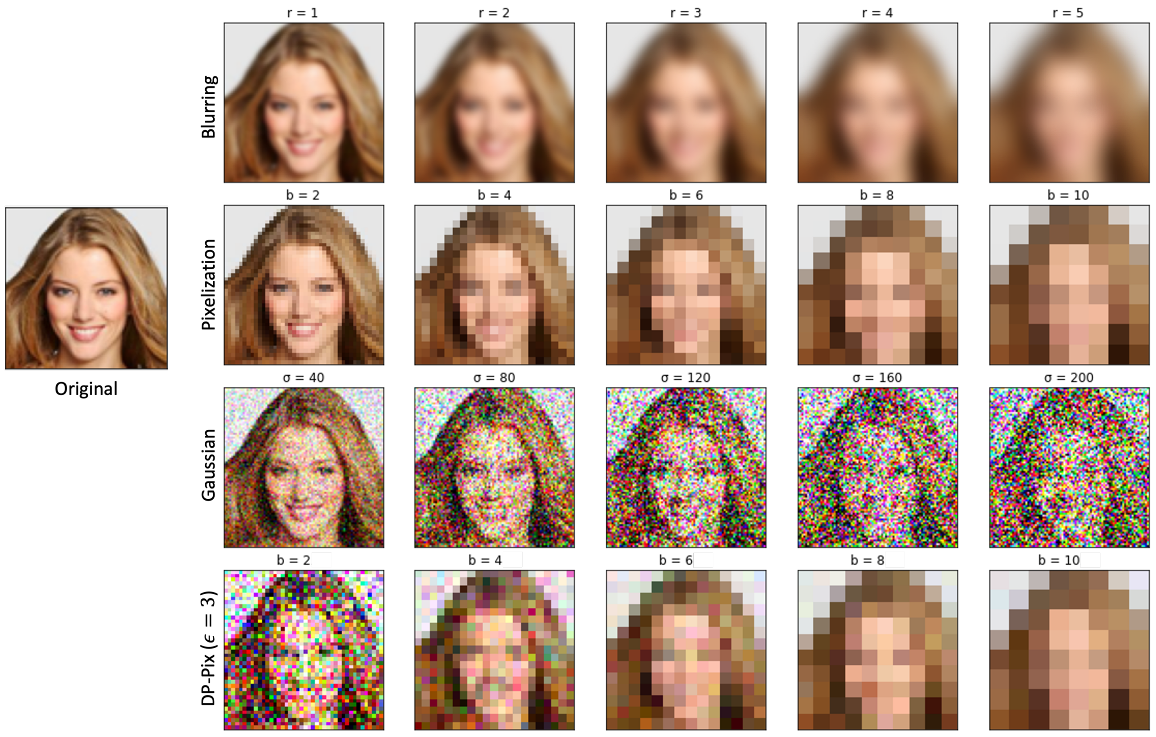


Figure 5.3: Privacy Methods (Original image from CelebA dataset)

where X is the original data point, ϵ is the random variable (noise) with a distribution $\epsilon \sim \mathcal{N}(0, \sigma^2)$ and Z is the transformed data point, obtained by the addition of noise ϵ to the input X.

Therefore, for an image with dimensions (H, W, C), we sample $H \times W \times C$ values from a Gaussian (normal) distribution with mean (μ) zero and standard deviation σ of the probability density function $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x-\mu)^2}{2\sigma^2}$. We use the implementation from [90].

5.3.2 Common Image Obfuscation

Two widely used image obfuscation techniques are *Pixelization* and *Blurring*.

Pixelization [91] (also referred to as mosaicing) can be achieved by superposing a rectangular grid of size $b \times b$ over the original image and averaging the color values of the pixels within each grid cell.

Blurring i.e., Gaussian blur, removes details from an image by convolving a 2D Gaussian kernel with the image. Let the radius of blur be r, then the size of the 2D kernel is given by $(2r+1) \times (2r+1)$. Then, the values in this 2D kernel are sampled from the distribution:

$$G(x, y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp -\frac{(x^2 + y^2)}{2\sigma^2}, \tag{5.1}$$

where (x, y) are the coordinates inside the 2D kernel with origin at the center and the standard deviation σ is approximated from the radius r [92]. We use Pillow Image Library [93] for the implementation.

5.3.3 Differentially Private Image Pixelization

Differential privacy (DP) is the state-of-the-art privacy paradigm for statistical databases [79]. Differentially Private Pixelization (DP-Pix) [82] extends the DP notion to image data publication. It introduces a concept of m-Neighborhood, where two images (I_1 and I_2) are neighboring images if they differ by at most m pixels. By differential privacy, content represented by up to m pixels can be protected. A popular mechanism to achieve DP is the Laplace mechanism. However, the global sensitivity of direct image perturbation would be very high i.e., $\Delta I = 255m$, leading to high perturbation error. The DP-Pix method first performs pixelization P_b (with grid cells of $b \times b$ pixels) on the input image I, and then applies Laplace perturbation to the pixelized image $P_b(I)$, effectively reducing the sensitivity $\frac{255m}{b^2}$. The following equation summarizes the algorithm ($\tilde{P_b}$) to achieve ϵ-differential privacy:

$$\tilde{P_b}(I) = P_b(I) + L_p, \tag{5.2}$$

where each value in L_p is randomly drawn from a Laplace distribution with mean 0 and scale $\frac{255m}{b^2\epsilon}$. The parameter $\epsilon > 0$ specifies the level of DP guarantee, where smaller values indicate stronger privacy. As DP is resistant to post-processing [79], any computation performed on the output of DP-Pix, i.e., the perturbed pixelized images, would not affect the ϵ-DP guarantees. Our approach proposes a denoising module for the obfuscated images by DP-Pix, improving the latent representation without sacrificing DP guarantees.

5.4 Privacy Enhanced Few-shot Image Classification

To build a few-shot model that can preserve the privacy of the input images, we can utilize any of the privacy methods discussed in the previous section. However, doing so may degrade the few-shot classification performance tremendously. To avoid this, we introduce a denoiser and train it jointly for few-shot classification using meta-learning on noisy images (Fig. 5.2). Together, the denoiser and the embedding network forms our *Denoising Network*. Combined with the properly chosen privacy method, the Denoising Network aims to discover a privacy-preserved latent embedding space (not denosing to recover the original image), where **the privacy of input data is be preserved** and **robustness and generality for few-shot classification are maintained**.

5.4.1 Denoiser

Zhang et al. [94] proposed a denoising convolutional neural network (DnCNN) which uses residual learning to output Gaussian noise. Specifically, the input of the network is a noisy observation such that $y = x + v$ where y is the input image, x be the clean image, and v be the actual noise. The network learns the residual mapping function $\mathcal{R}(y) \approx v$ and predicts the clean image using $x = y - \mathcal{R}(y)$. The averaged mean squared error between the predicted residue and actual noise is used as the loss function to train this denoiser with parameters ϕ as

$$\mathcal{L}(\phi) = \frac{1}{2N} \sum_{i=1}^{N} ||\mathcal{R}(y_i; \phi) - (y_i - x_i)||^2. \tag{5.3}$$

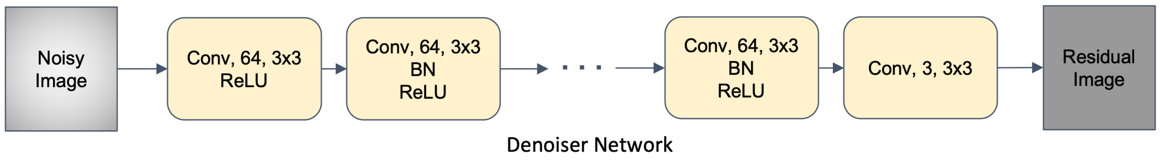


Figure 5.4: DnCNN denoiser with 8 layers

We plug the DnCNN denoiser into our FS-PIC pipeline (Fig. 5.2) to estimate the clean image before pixelization, blurring, Gaussian noise, and DP-Pix. The architecture for the denoiser is depicted in Fig. 5.4.

5.4.2 Embedding Network

Partially denoised images from the denoiser $D(\phi)$ are fed to embedding network $E(\theta)$ to obtain denoised embeddings, which then form the class prototypes. The classification loss is measured using Eq. 2.7.

5.4.3 Denoising Network

The total loss for training the *Denoising Network* (Denoiser + Embedding Network) is formulated as the sum of denoising loss and classification loss:

$$\mathcal{L} = \mathcal{L}(\phi) + \mathcal{L}(\theta). \tag{5.4}$$

The joint loss enforces the reduction of noise in input images while learning the distinctive representations that maximize the few-shot classification accuracy. This simple loss guides the embedding space towards privacy-preserved latent space without losing its generality. For Prototypical Networks, the prototypes are expected to be the centers of the privacy-preserved embeddings for each class. Although the sum of losses can be weighted, our experiments observed that weighting did not significantly impact the final accuracy of the few-shot image classification model as long as the weighting coefficients are non-zero. We outline the episodic training process used for building a FS-PIC model in Alg. 4 and describe the notations used in Table 5.1.

Algorithm 4: FS-PIC model training

```
Input: $D = \{(x_1, y_1), ..., (x_t, y_t)\}$ where $y_i \in \{1, ..., M\}$. $D^c$ denotes the subset
 of D containing all elements $(x_i, y_i)$ such that $y_i = c$.
while True do
    // Select a set of N classes
    $V \leftarrow \text{RandomSample}(\{1, ..., M\}, N)$
    for $c$ in $V$ do
        // Select support examples
        $S_e^c \leftarrow \text{RandomSample}(D^c, K)$
        // Select query examples
        $Q_e^c \leftarrow \text{RandomSample}(D^c \setminus S_e^c, H)$
        // Add noise
        $\hat{S}_e^c \leftarrow \text{AddNoise}(S_e^c, \epsilon)$
        $\hat{Q}_e^c \leftarrow \text{AddNoise}(Q_e^c, \epsilon)$
    end
    // Form a set of all clean images
    $S_e \leftarrow \{S_e^1, S_e^2, ...S_e^N\}$
    $Q_e \leftarrow \{Q_e^1, Q_e^2, ...Q_e^N\}$
    $X_e \leftarrow \{S_e, Q_e\}$
    // Form a set of all noisy images
    $\hat{S}_e \leftarrow \{\hat{S}_e^1, \hat{S}_e^2, ...\hat{S}_e^N\}$
    $\hat{Q}_e \leftarrow \{\hat{Q}_e^1, \hat{Q}_e^2, ...\hat{Q}_e^N\}$
    $\hat{X}_e \leftarrow \{\hat{S}_e, \hat{Q}_e\}$
    // Apply the denoiser
    $\bar{X}_e \leftarrow G(\hat{X}_e; \theta)$
    $\bar{S}_e, \bar{Q}_e \leftarrow \bar{X}_e$
    // Calculate denoising loss
    $\mathcal{L}_d \leftarrow \text{MSE}(\bar{X}_e, X_e)$
    // Compute class prototypes using denoised support examples
    for $c$ in $V$ do
        $\bar{p_c} \leftarrow \frac{1}{K} \sum_{(\bar{x_i}, y_i) \in \bar{S}_e^c} f_\phi(\bar{x_i})$
    end
    $\mathcal{L}_c \leftarrow 0$
    for $c$ in $V$ do
        for $(\bar{x_i}, y_i)$ in $\bar{Q}_e^c$ do
            $\mathcal{L}_c \leftarrow \mathcal{L}_c + \frac{1}{NH}[d(f_\phi(\bar{x_i}), \bar{p_c}) + \log \sum_{c'} \exp(-d(f_\phi(\bar{x_i}), \bar{p_c}))]$
        end
    end
    $\mathcal{L} \leftarrow \mathcal{L}_d + \mathcal{L}_c$
    $\phi \leftarrow \phi - \alpha_\phi \frac{\partial \mathcal{L}}{\partial \phi}$
    $\theta \leftarrow \theta - \alpha_\theta \frac{\partial \mathcal{L}}{\partial \theta}$
end
```

Table 5.1: Notations used in FS-PIC model training

Notation	Description
t	#examples in the training set
M	#classes in the training set
$N <= M$	#classes sampled per episode
K	#support examples sampled per class
H	#query examples sampled per class

5.5 Experiments

5.5.1 Datasets

- **Omniglot** [95] is a dataset of 1623 handwritten characters collected from 50 alphabets. Each character has 20 examples drawn by a different human subject. We follow the same procedure as in [18] by resizing the gray-scale images to 28×28 and augmenting the character classes with rotations in multiples of 90 degrees. Our training, validation, and testing split is of sizes 1028, 172, and 423 characters, respectively (or $4\times$ with augmentation).

- **CelebFaces Attributes Dataset (CelebA)** [96] is a large-scale face attributes dataset with more than 10K celebrity (classes) images. For the purpose of our experiments, we select classes that have at least 30 samples. This gives us 2360 classes in total, out of which 1510 are used for training, 378 for validation, and 427 for testing. We use aligned and cropped version of the dataset in which images are of dimension $218(h) \times 178(w)$. We center crop each image to 176×176 and then resize to 84×84.

- **MiniImageNet** [18] dataset contains 100 general object classes where each class has 600 color images. The images are resized to 84×84, and the dataset is split into 64 training, 16 validation, and 20 testing classes following [40].

5.5.2 Settings for Privacy Methods

We explore the following parameters for each privacy method. Gaussian Blur with radius $r = \{1, 2, 3, 4, 5\}$ is used for blurring images. A filter window of size $b \times b$ where $b = \{2, 4, 6, 8, 10\}$ is used for pixelization. The pixelated image is then resized to match the model input dimensions. We perform experiments with Gaussian noise $\epsilon \sim \mathcal{N}(\mu, \sigma)$ with mean $\mu = 0$ and standard deviation $\sigma = \{40, 80, 120, 160, 200\}$. For DP-Pix, we fix $\epsilon = 3$, $m = 1$ and vary pixelization parameter b with values $\{2, 4, 6, 8, 10\}$.

5.5.3 Denoising Network

We use a lighter version of the DnCNN [94] model i.e., with 8 CNN layers instead of 17, for first denoising the image and subsequently feeding the denoised image into one of the following embedding networks. **Conv-4** is a 4-layered convolutional neural network with 64 filters in each layer originally proposed in [40] for few-shot classification. **ResNet-12** is a 12-layer CNN with 4 residual blocks. It has been shown to have better classification accuracy on few-shot image classification tasks. The architecture of the two embedding networks are detailed in Fig. 5.5.

5.5.4 Training and Evaluation

We train using N-way K-shot PIC tasks (Algorithm. 4) and use Adam optimizer with learning rate $\alpha_\theta = \alpha_\phi = 0.001$ with a decay of 0.5 every 20 epochs. Table 5.2 lists the hyperparameters for the three datasets. The network is trained to minimize total loss of denoiser and classifier (Eq. 5.4). We evaluate the performance by sampling 5-way 5-shot PIC tasks (with same privacy settings) from the test sets and measure the classification accuracy. The final results report the performance averaged over 1000 test episodes for the Omniglot dataset, and 600 test episodes for both MiniImageNet and CelebA datasets. To measure the effectiveness of the proposed denoising embedding space, we both train and evaluate each model's performance in two settings:

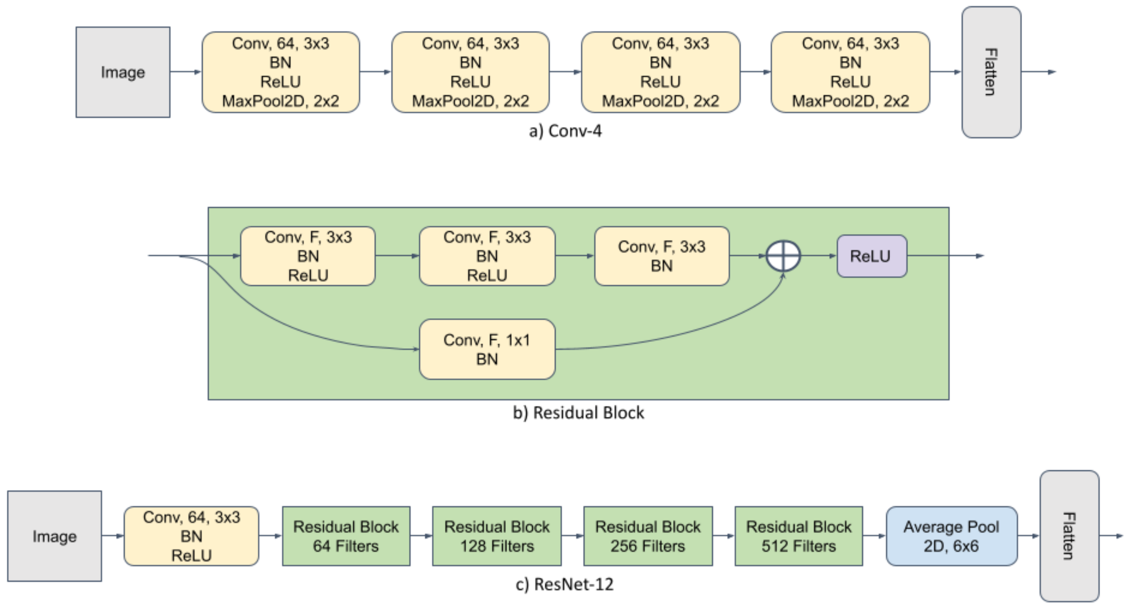


Figure 5.5: Encoders: a) Conv-4, b) Residual Block, c) ResNet-12

Table 5.2: Hyperparameters for Episodic Training

	Omniglot	CelebA	MiniImageNet
Way	60	5	5
Shots	5	5	5
Query	5	5	15
Epochs	500	200	200
Patience	50	20	20
Episodes	100	100	100

1) **without using the denoiser** and 2) **jointly training the denoiser with the classifier** i.e., the proposed *Denoising Network*.

5.5.5 Privacy Risk Evaluation

Privacy attacks on trained models such as model inversion[1] and membership inference[2] are not applicable in our setting because the denoising and embedding models are trained with publicly available classes (data) using meta-learning. The user-supplied test data (support and query set) are obfuscated for privacy protection. A practical privacy attack on obfuscated images is to infer the identities using existing facial recognition systems and public APIs, e.g., Rekognition. In this study, our goal is to investigate (1) the efficacy of the studied image obfuscation methods for privacy protection and (2) whether the proposed denoising approach has effects on privacy. To simulate a powerful adversary, we apply the state-of-the-art face recognition techniques, e.g., FaceNet with the Inception ResNet V1 network[97], on the CelebA dataset; MTCNN [98] is applied to detect and resize the facial region in each input image. Specifically, 1000 entities were randomly selected from the CelebA dataset. For each entity, we randomly sampled 30 images, which were then partitioned between training and testing (20 : 10). Different versions of the test set were generated by applying image obfuscation methods with various parameter values (denoted as `Noisy`) and by applying the proposed Denoising Network (denoted as `Denoised`). We fine-tuned the Inception network and trained an SVC classifier on the clean training data. In Fig. 5.8, we report the accuracy on the noisy and denoised test sets, i.e., success of re-identification, with higher values indicating higher privacy risks.

5.6 Results

Table 5.3: Baseline test accuracy of 5-way 5-shot classification of clean images. Omniglot is not evaluated for ResNet-12 because of its already near 100% performance.

	Omniglot	CelebA	MiniImageNet
Conv-4	0.99	0.90	0.61
ResNet-12	–	0.92	0.65

5.6.1 Task Difficulty

The average 5-way 5-shot classification accuracy of our baseline few-shot model [40] trained on clean images and tested on clean images is 99% on Omniglot dataset, 91% on CelebA dataset, and 61% on MiniImageNet dataset using Conv-4 encoder (Table 5.3). This shows the approximate level of difficulty of few-shot tasks for each dataset i.e., Omniglot tasks are easy, tasks from CelebA have medium difficulty, and MiniImageNet tasks are hard.

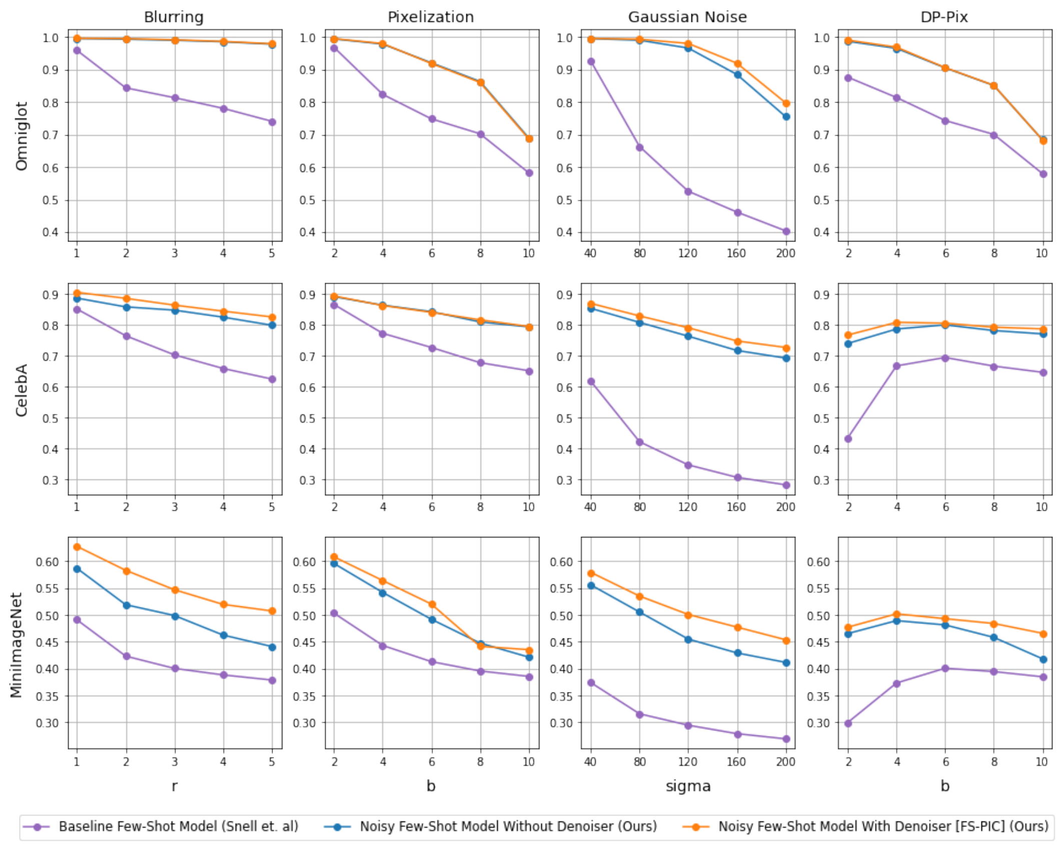


Figure 5.6: Test accuracy (y-axis) of 5-way 5-shot private image classification tasks sampled from Omniglot (top), CelebA (center) and MiniImageNet (bottom) datasets, presented with different privacy settings (x-axis) when using Conv-4 as encoder.

5.6.2 Generalization

We compare results for few-shot private image classification using three models in Fig. 5.6:

1. **Baseline Few-Shot Model:** When the few-shot model is trained on clean images and is tested on noisy images.

2. **Noisy Few-Shot Model Without Denoiser:** When the baseline few-shot model is trained on noisy images and is tested on noisy images with same privacy settings.

3. **Noisy Few-Shot Model With Denoiser:** When the baseline few-shot

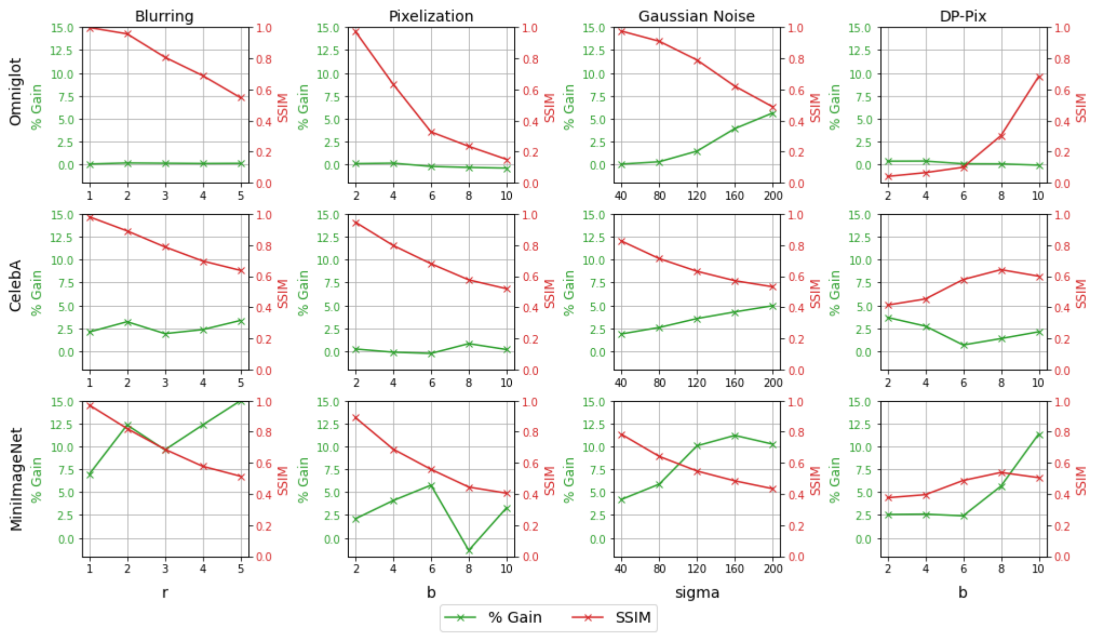


Figure 5.7: % Gain vs SSIM for Conv-4

model is *jointly trained with the denoiser* on noisy images and is tested on noisy images with same privacy settings (Algorithm 4).

In all cases, we observe that noisy few-shot models outperforms the baseline few-shot model with wide gap. Also, in most cases, we note that adding a denoiser improves the accuracy. To better observe the effectiveness of denoiser, in Fig. 5.7, we quantify the improvement by calculating % Gain = $\frac{\text{accuracy with denoiser}-\text{accuracy without denoiser}}{\text{accuracy without denoiser}} \times 100$. We also quantify the change to the original image caused by the privacy method (post denoising) by calculating Structural Similarity Index (SSIM) [99] between denoised image and original clean image, averaged over 100 test images for each dataset and privacy parameter.

Blurring, Pixelization and Gaussian Noise: As we increase the value of privacy parameters, the SSIM decreases, suggesting the higher dissimilarity between the denoised images and the original image (Fig 5.7). Despite the degradation caused by the privacy method to the original image, we observe positive % Gains for all three datasets. Specifically, on hard tasks (MiniImageNet), a gain of upto 15% in accuracy

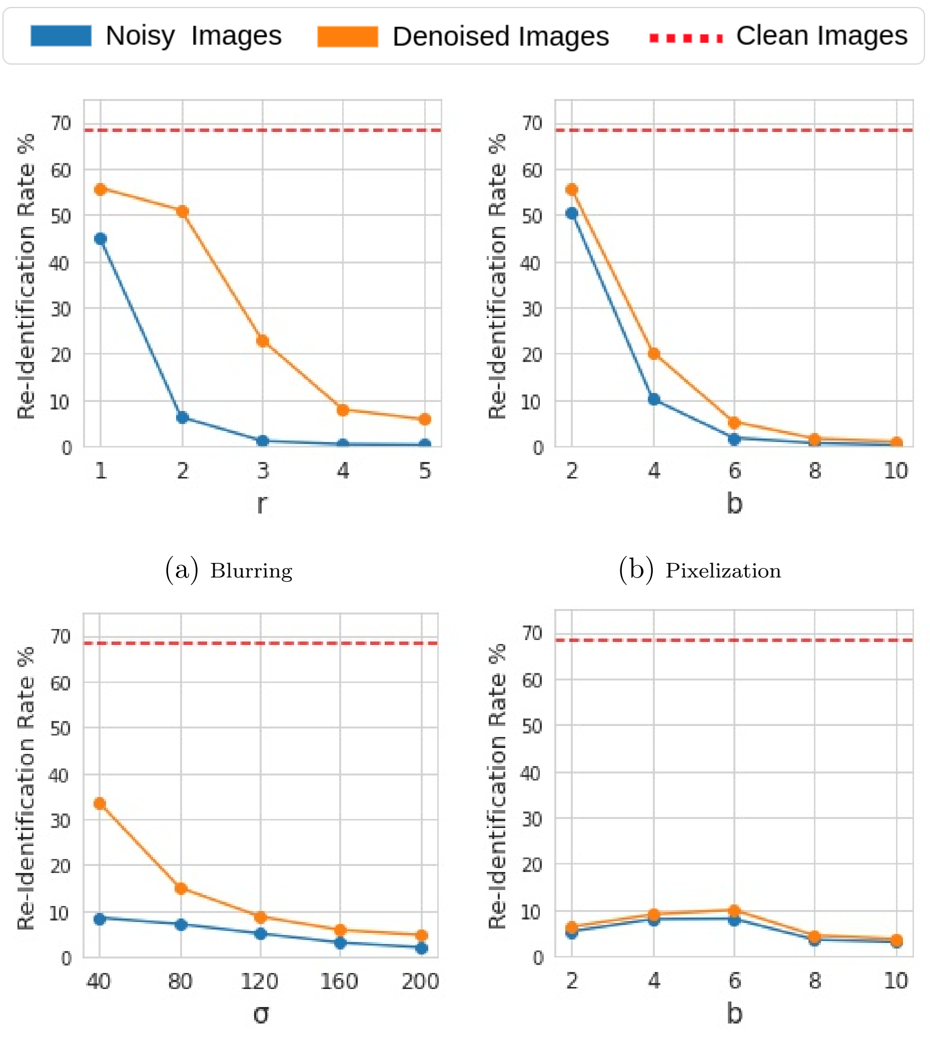
($r = 5$) with the proposed *Denoising Network*, reaffirming the generality of the few-shot learning. For easy (Omniglot) and medium (CelebA) tasks, where the baseline accuracy is already high, a relatively small positive gain of up to 5% ($\sigma = 200$) is reported.

DP-Pix: As we increase the size of pixelization window (b), the amount of Laplace noise that we add to the image decreases (as defined by $\frac{255m}{b^2\epsilon}$); however, the image quality decreases because of increasing pixelization. Therefore, we observe a trade-off point where the accuracy first increases and then decreases as we increase b (Fig 5.6). This trade-off is particularly observed for CelebA and MiniImageNet datasets. For Omniglot dataset, the performance just decreases with increasing b because of the low resolution images in the dataset. From Fig. 5.7, we observe that DP-Pix has the lowest SSIM values when compared with other privacy methods causing the most notable changes on the original image. Interestingly, we note that even with low SSIM values, we find instances that exhibit moderate % gain i.e., at $b = 2, 4, 6$ indicating the presence of privacy preserving denoising embeddings.

5.6.3 Empirical Privacy Risks

As described earlier, this experiment showcases the efficacy of image obfuscation methods against a practical adversary who utilizes state-of-the-art face recognition models trained on clean images. Furthermore, this experiment investigates whether the proposed denoising network has effects on privacy protection empirically. To this end, we vary the algorithmic parameters of the privacy methods and report the face re-identification rates on both obfuscated and denoised images in Fig. 5.8. For the clean test set sampled from CelebA, the re-identification accuracy is 68.12%.

Blurring: In Fig. 5.8a, the privacy risks are quite high with Gaussian kernel size $r = 1$ for both blurred and denoised images. As we increase the radius r, the chance of re-identifying the image decreases rapidly. We also observe that after denoising, the blurred images are more likely to be re-identified. For instance, at $r = 2$, the



(a) Blurring (b) Pixelization (c) Additive Gaussian (d) DP-Pix ($\epsilon = 3$)

Figure 5.8: Privacy Risk Evaluation with CelebA

re-identification rate is 7.34% for blurred images but 54.01% for denoised images.

Pixelization: As shown in Fig. 5.8b, small cell size in pixelization, e.g., $b = 2$, leads to high face re-identification rates for both pixelized and denoised images. Increasing b helps reduce the rate of face re-identification rapidly, e.g., from 53.73% to 4.82% for pixelized images by increasing b from 2 to 6. Denoising slightly increases the privacy risk, but the additional risk diminishes with larger b values and is much lower than observed in blurring.

Additive Gaussian: Over the range of σ values studied in our experiments, Additive Gaussian inflicts lower privacy risks with a small noise ($\sigma = 40$), compared to Blurring and Pixelization. As shown in Fig. 5.8c, increasing σ leads to a moderate reduction in the privacy risk. For example, face re-identification rate is 5.11% at mid noise level ($\sigma = 120$), reduced from 8.54% at low noise level ($\sigma = 40$). Denoising the obfuscated images leads to a significant increase in the privacy risk at low σ values, e.g., 33.55% higher in face re-identification rate when $\sigma = 40$.

DP-Pix Fig. 5.8d presents the re-identification results for images obfuscated with DP-Pix as well as those denoised by our proposed approach. We observe low privacy risks across all b values. Furthermore, performing denoising on DP-Pix obfuscated images does not lead to significant higher privacy risks with any b value, as opposed to other image obfuscation methods. While face re-identification rates are consistently low, higher rates occur when $b = 4$ and 6. Recall that higher utility was observed when $b = 4$ and 6 in Fig. 5.6. It has been reported in [82] that the quality of obfuscated images may be optimized by tuning b value given the privacy requirement ϵ, by balancing the approximation error by pixelization and the Laplace noise.

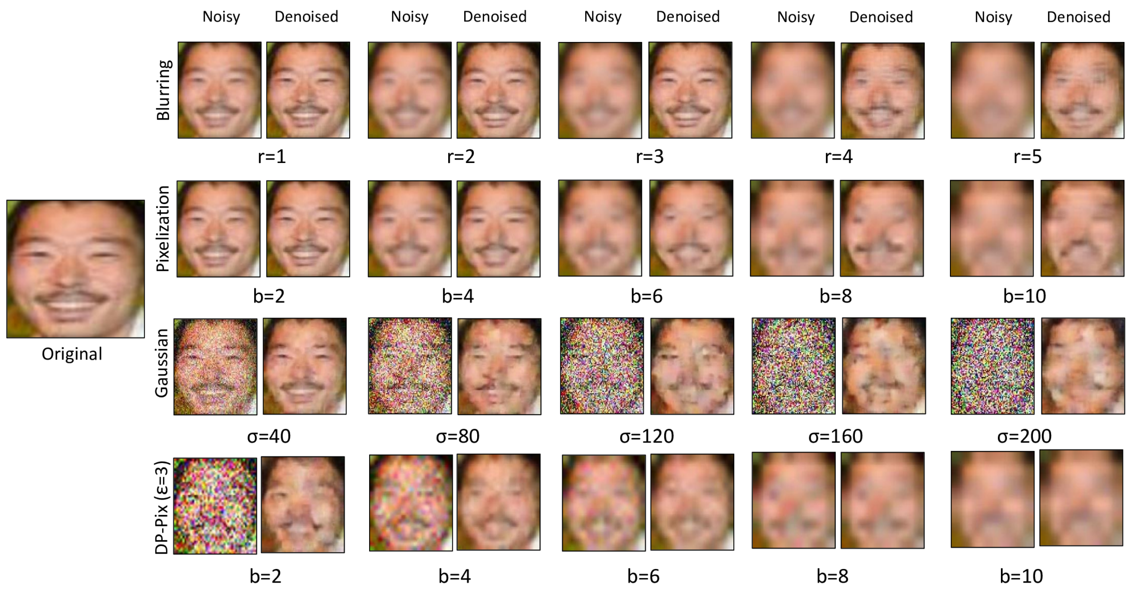


Figure 5.9: Qualitative Evaluation of Privacy Methods: The figure depicts the obfuscated images (`Noisy`) by the studied privacy methods at various parameters, as well as the denoised output, with a sample input from CelebA.

5.6.4 Qualitative Evaluation of Privacy Methods

Fig. 5.9 provides a qualitative evaluation on the obfuscated and denoised images generated for a range of parameter values. MTCNN [98] was applied to a sample input image of CelebA, to detect the facial region. Perceptually, the proposed denoiser may improve the image quality upon the obfuscated images to various extents. However, image quality does not always correlate with empirical privacy risks, i.e., face re-identification with public models. In combination with Fig. 5.8, we observe that the proposed denoising leads to various levels of privacy risk increment, while producing higher quality images. For example, the results show higher privacy risk increment for Blurring with $r = 3$ (23.33%), moderate increment for Pixelization $b = 4$ (10.14%) and Additive Gaussian $\sigma = 80$ (7.96%), and little increment for DP-Pix $b = 2$. Fig. 5.9 confirms that the denoiser performance may vary depending on the image obfuscation method, and that DP-Pix provides consistent privacy protection even with denoising.

5.6.5 Observation of the Privacy-Preserved Embedding Space

Fig. 5.11 shows the evolution of the embeddings in the process of privacy encoding and privacy-preserved representation learning by presenting the t-SNE [100] visualization of the clean, noisy, and denoised embeddings of randomly sampled 100 test images from a total of 5 classes from the CelebA dataset. The embeddings are obtained from the ResNet-12 encoder trained under different noise settings for 5-way 5-shot classification. We say the embeddings are clean when the input images have no noise and the encoder is trained for few-shot classification of clean images. The noisy embeddings are obtained by using the encoder trained for few-shot classification of noisy images and without using the denoiser. The denoised embeddings are obtained by the proposed *Denoising Network* (Fig. 5.2) i.e., the encoder trained in conjunction with denoiser for few-shot classification on noisy images.

We report the results for a case when a few-shot method such as Prototypical

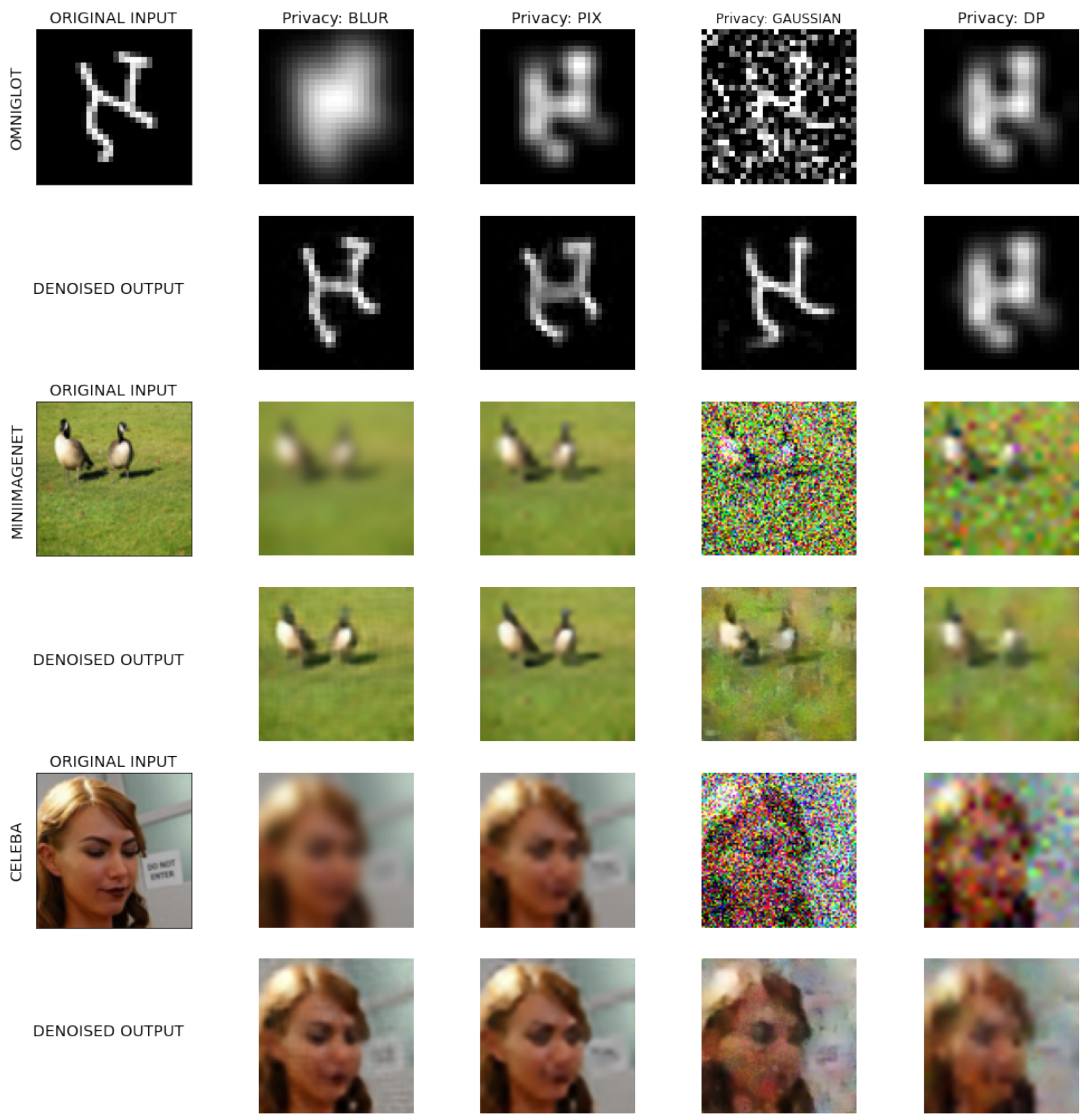


Figure 5.10: The figure shows the results of applying different types of privacy methods i.e., Blurring, Pixelization, Additive Gaussian noise, and Differentially Private noise to a sample from each of Omniglot, MiniImageNet and CelebA datasets. It also shows the denoised output obtained from the DnCNN denoiser. We observe that images protected with DP-Pix are hard to denoise when compared with Blurring, Pixelization, and Gaussian noise.

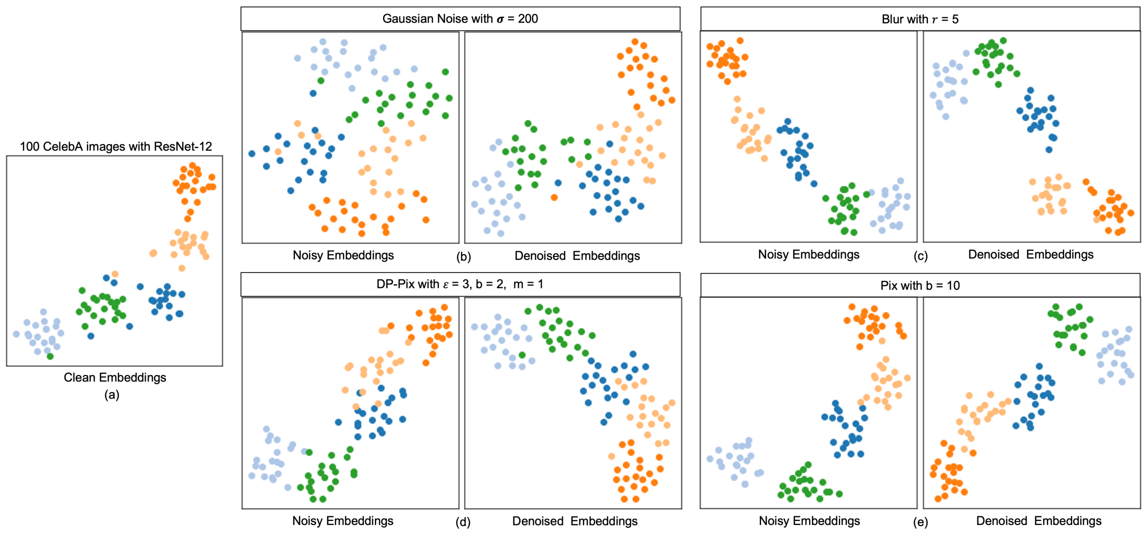


Figure 5.11: t-SNE Visualization

Networks can generate good clusters for the clean images (Fig. 5.11a), and observe the impact on clustering with noisy images and subsequently when those images are denoised. We notice that when the initial clusters are good, pixelization (Fig. 5.11c) and blurring (Fig. 5.11e) will have little impact on the quality of the clusters even with the high amount of noise. Therefore, pixelization and blurring maintain generality (robust to noise) and are also vulnerable to re-identification. Gaussian noise (Fig. 5.11b) distorts the initial clusters more significantly, which can lead to lower few-shot classification performance. Applying denoising to Gaussian noise improves the clustering results, however still poses moderate privacy threat as seen in re-identification experiments (Fig. 5.8c). Similarly, with DP-Pix (Fig. 5.11d), the original clusters are also distorted upon obfuscation. But, when denoised with proposed Denoising Network, we can observe better clustering performance. Because of DP-Pix's privacy guarantee and lowest re-identification rates, we can say that the obtained denoised embeddings are privacy-protected i.e., the network finds the privacy-preserved embedding space which maintains generality (robust to noise) and also preserves privacy.

5.7 Discussion

In this work, we present a novel framework for training a few-shot private image classification model, which aims to preserve the privacy of user-supplied training and testing data. The framework makes it possible to deploy few-shot models in the cloud without compromising users' data privacy. We discuss and confirm that there exists a privacy-preserved embedding space which has both stronger privacy and generalization performance on few-shot classification. The proposed method provides privacy guarantees while preventing severe degradation of the accuracy as confirmed by results on three different datasets of varying difficulty with several privacy methods. Evaluation with re-identification attacks verifies the low empirical privacy risk of our proposed method, especially with DP-Pix. While our study focuses on well-known image obfuscation methods, future research may explore scenarios where users could apply novel image obfuscation methods locally, i.e., different from those applied to training data. Furthermore, our results motivate the future direction of searching for a more effective privacy-preserved space for few-shot learning in other domains such as speech [62]. Examination of other evaluation metrics for privacy-preserved embedding space will promote the relevant future study. We release the code for our experiments at `https://github.com/ArchitParnami/Few-Shot-Privacy`.

This chapter is reused from our work **Privacy Enhancement for Cloud-Based Few-Shot Learning** [101] with permission from authors.

CHAPTER 6: CONCLUSION & FUTURE WORKS

This dissertation explores the practicality of few-shot models to be deployed for domains other than images, such as speech by considering the problem of keyword spotting as a use-case. Next, to be able to use few-shot models in the continual learning settings where the information from new classes needs to be continually accumulated without forgetting old information, a problem referred to as few-shot continual learning is also discussed. Finally, it also raises the question of privacy when deploying few-shot models in the cloud and suggests a solution in that direction. To summarize, our contributions are directed towards the following problems:

- **Few-Shot Keyword Spotting:** In speech domain, keyword spotting is used to detect specific keywords in human speech. Such systems are commonly used in smart embedded devices which do not use Large Vocabulary Continuous Speech Recognition (LVCSR) due to either computation, latency or privacy constraints. Traditionally, training keyword spotting models have required thousands of user samples per keyword. Our work addresses this problem by developing a Few-Shot Keyword Spotting Framework, where few-shot models can be trained using MFCC features to detect new keyword just from 2 samples per keyword. We present our results using Prototypical Networks as an example few-shot model and also release a dataset for future research in this direction.
- **Few-Shot Continual Learning:** Few-shot models are static in nature, i.e., trained to detect a certain number of classes, hence are not capable of growing and learning new classes. On the contrary, the field of continual learning focuses

on building model architectures that supports learning new information (classes) without forgetting old information. However, most such approaches break down in limited data regime. In our work, we propose an approach to perform few-shot continual learning using an ensemble of one-class classifier that are trained to work with few samples and also without theoretically restricting the number of classes that could be learned and realizing the potential of few-shot models to be deployed in more real time scenarios.

- **Privacy Enhanced Few-Shot Learning:** Few-shot learning models are vulnerable to privacy attacks on user supplied training and testing data when deployed in cloud. Incorporating privacy mechanisms on the out-of-box few-shot models results in severe degradation of performance, hence impacting their general adoption. Therefore, to make few-shot models practical for deployment, and yet preserve the privacy of user data, we develop a novel framework for training a few-shot private image classification models. Our proposed method provides privacy guarantees while preventing severe degradation of the accuracy as confirmed by results on three different datasets of varying difficulty with several privacy methods.

Our work, so far has developed approaches to tackle problems like privacy, continual learning and learning from limited speech, individually in the context of Few-shot learning. Therefore, future work could involve testing and upgrading the proposed approaches to work together at the intersection of the problem domains as discussed below:

- **Few-Shot Continual Keyword Spotting**: The few-shot keyword spotting model proposed in this dissertation is static in nature i.e., the model is only able to detect limited number of keywords. However, in dynamic environments such as voice assistants new keywords with few user samples need to be learned.

This requires building models that can perform keyword spotting continuously on new keywords such as [102, 103]. However, learning new keywords without forgetting old keywords is still a challenge in limited data regime. Using the techniques proposed in this dissertation for few-shot continual learning, a few-shot continual keyword spotting model can be developed similarly.

- **Privacy Enhancement for Few-Shot Keyword Spotting:** In Chapter 5, we addressed privacy issues surrounding the deployment of few-shot image classification models in a cloud-based environment. Specifically, we adopt addition of noise to images as means for finding privacy-preserved embedding space for few-shot image classification. However, the question still remains on the applicability of the proposed method on data domains other than images, such as speech. Particularly, we would like to address whether similar noise addition techniques can be used for preserving privacy of speech data when developing few-shot keyword spotting models and how it affects their performance. Future work in this direction will explore building privacy-preserved few-shot keyword spotting models because privacy is always a concern in adoption of smart voice assistants.

- **Privacy Enhanced Few-Shot Continual Learning:** Learning new information without forgetting old information is a challenge for neural networks. Moreover, being able to retain new information just from few samples is even more harder task. Our work in Chapter 4, proposed a technique to overcome this challenge up to a certain extent. Generally, as new knowledge is accumulated, model performance declines. However, to build practical few-shot models that not only are able to learn from few samples continuously but also preserve the privacy of the user information is yet the hardest of the problems discussed in this dissertation. Some preliminary work [104] has been done to address privacy in continual learning models with differential privacy. However, much more research is required to make such models work with few examples. Future work would thus begin with the integration of all three techniques proposed in this dissertation and analyze their effect on the performance of few-shot models for their practical usability.

REFERENCES

[1] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pp. 1322–1333, ACM, 2015.

[2] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *2017 IEEE symposium on security and privacy*, pp. 3–18, IEEE, 2017.

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, IEEE, 2016.

[4] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[5] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, 2016.

[6] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Computing Surveys*, vol. 53, no. 3, 2020.

[7] L. Bertinetto, J. a. F. Henriques, J. Valmadre, P. H. S. Torr, and A. Vedaldi, "Learning feed-forward one-shot learners," in *Proceedings of the 30th Interna-*

tional Conference on Neural Information Processing Systems, pp. 523–531, Curran Associates Inc., 2016.

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

[9] Q. Fan, W. Zhuo, C.-K. Tang, and Y.-W. Tai, "Few-shot object detection with attention-rpn and multi-relation detector," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4013–4022, IEEE, 2020.

[10] E. Triantafillou, R. Zemel, and R. Urtasun, "Few-shot learning through an information retrieval lens," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 2252–2262, Curran Associates Inc., 2017.

[11] H.-U. Kim, Y. J. Koh, and C.-S. Kim, "Online multiple object tracking based on open-set few-shot learning," *IEEE Access*, vol. 8, pp. 190312–190326, 2020.

[12] L. Zhu and Y. Yang, "Compound memory networks for few-shot video classification," in *Proceedings of the European Conference on Computer Vision*, pp. 782–797, Springer International Publishing, 2018.

[13] L.-Y. Gui, Y.-X. Wang, D. Ramanan, and J. M. F. Moura, "Few-shot human motion prediction via meta-learning," in *Proceedings of the European Conference on Computer Vision*, pp. 432–450, 2018.

[14] H. Yang, X. He, and F. Porikli, "One-shot action localization by learning sequence matching network," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1450–1459, 2018.

[15] Y. Wu, Y. Lin, X. Dong, Y. Yan, W. Ouyang, and Y. Yang, "Exploit the unknown gradually: One-shot video-based person re-identification by stepwise learning," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5177–5186, 2018.

[16] V. Joshi, M. E. Peters, and M. Hopkins, "Extending a parser to distant domains using a few dozen partially annotated examples," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1190–1199, Association for Computational Linguistics, 2018.

[17] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," in *Advances in Neural Information Processing Systems*, vol. 33, pp. 1877–1901, Curran Associates, Inc., 2020.

[18] O. Vinyals, C. Blundell, T. Lillicrap, k. kavukcuoglu, and D. Wierstra, "Matching networks for one shot learning," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016.

[19] M. Yu, X. Guo, J. Yi, S. Chang, S. Potdar, Y. Cheng, G. Tesauro, H. Wang, and B. Zhou, "Diverse few-shot text classification with multiple metrics," *arXiv preprint arXiv:1805.07513*, 2018.

[20] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. Pande, "Low data drug discovery with one-shot learning," *ACS Central Science*, vol. 3, no. 4, pp. 283–293, 2017.

[21] A. Brock, T. Lim, J. M. Ritchie, and N. Weston, "Smash: one-shot model

architecture search through hypernetworks," *arXiv preprint arXiv:1708.05344*, 2017.

[22] Y. Wu and Y. Demiris, "Towards one shot learning by imitation for humanoid robots," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2889–2894, 2010.

[23] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss, "Learning manipulation actions from a few demonstrations," in *2013 IEEE International Conference on Robotics and Automation*, pp. 1268–1275, 2013.

[24] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in Cognitive Sciences*, vol. 3, no. 4, pp. 128–135, 1999.

[25] O. Vinyals, "Model vs optimization meta learning," *Meta-Learning Workshop at Advances in Neural Information Processing Systems*, 2017.

[26] J. 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.

[27] T. Schaul and J. Schmidhuber, "Metalearning," *Scholarpedia*, vol. 5, no. 6, p. 4650, 2010. revision #91489.

[28] S. Thrun and L. Pratt, *Learning to Learn: Introduction and Overview*, pp. 3–17. Boston, MA: Springer US, 1998.

[29] C. Giraud-Carrier, R. Vilalta, and P. Brazdil, "Introduction to the special issue on meta-learning," *Machine learning*, vol. 54, no. 3, pp. 187–193, 2004.

[30] L. A. Rendell, R. Sheshu, and D. K. Tcheng, "Layered concept-learning and dynamically variable bias management.," in *Proceedings of the International Joint Conference on Artificial Intelligence*, pp. 308–314, 1987.

[31] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[32] R. Caruana, "Multitask learning," *Machine learning*, vol. 28, no. 1, pp. 41–75, 1997.

[33] R. Polikar, "Ensemble learning," *Scholarpedia*, vol. 4, no. 1, p. 2776, 2009. revision #186077.

[34] B. M. Lake, R. Salakhutdinov, and J. B. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 2015.

[35] B. N. Oreshkin, P. Rodriguez, and A. Lacoste, "Tadam: Task dependent adaptive metric for improved few-shot learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 719–729, Curran Associates Inc., 2018.

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

[37] A. Parnami and M. Lee, "Learning from few examples: A summary of approaches to few-shot learning," *arXiv preprint arXiv:2203.04291*, 2022.

[38] B. Kulis *et al.*, "Metric learning: A survey," *Foundations and Trends® in Machine Learning*, vol. 5, no. 4, pp. 287–364, 2013.

[39] G. Koch, R. Zemel, R. Salakhutdinov, *et al.*, "Siamese neural networks for one-shot image recognition," in *Deep Learning Workshop at International Conference on Machine Learning*, vol. 2, p. 0, 2015.

[40] J. Snell, K. Swersky, and R. Zemel, "Prototypical networks for few-shot learning," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, pp. 4080–4090, Curran Associates Inc., 2017.

[41] F. Sung, Y. Yang, L. Zhang, T. Xiang, P. H. Torr, and T. M. Hospedales, "Learning to compare: Relation network for few-shot learning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1199–1208, 2018.

[42] S. W. Yoon, J. Seo, and J. Moon, "Tapnet: Neural network augmented with task-adaptive projection for few-shot learning," in *International Conference on Machine Learning*, pp. 7115–7123, PMLR, 2019.

[43] H. Li, D. Eigen, S. Dodge, M. Zeiler, and X. Wang, "Finding task-relevant features for few-shot learning by category traversal," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 1–10, 2019.

[44] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*, pp. 227–236, Springer, 1990.

[45] S. Ravi and H. Larochelle, "Optimization as a model for few-shot learning," in *International Conference on Learning Representations*, 2017.

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

[47] A. A. Rusu, D. Rao, J. Sygnowski, O. Vinyals, R. Pascanu, S. Osindero, and R. Hadsell, "Meta-learning with latent embedding optimization," *arXiv preprint arXiv:1807.05960*, 2018.

[48] P. Motlicek, F. Valente, and I. Szoke, “Improving acoustic based keyword spotting using lvcsr lattices,” in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 4413–4416, 2012.

[49] D. Can and M. Saraclar, “Lattice indexing for spoken term detection,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 8, pp. 2338–2347, 2011.

[50] T. N. Sainath and C. Parada, “Convolutional neural networks for small-footprint keyword spotting,” in *Proc. Interspeech 2015*, pp. 1478–1482, 2015.

[51] G. Chen, C. Parada, and G. Heigold, “Small-footprint keyword spotting using deep neural networks,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing)*, pp. 4087–4091, 2014.

[52] Y. Zhang, N. Suda, L. Lai, and V. Chandra, “Hello edge: Keyword spotting on microcontrollers,” *arXiv preprint arXiv:1711.07128*, 2017.

[53] R. Tang and J. Lin, “Deep residual learning for small-footprint keyword spotting,” in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 5484–5488, 2018.

[54] D. C. de Andrade, S. Leo, M. L. D. S. Viana, and C. Bernkopf, “A neural attention model for speech command recognition,” *arXiv preprint arXiv:1808.08929*, 2018.

[55] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[56] S. Choi, S. Seo, B. Shin, H. Byun, M. Kersner, B. Kim, D. Kim, and S. Ha,

"Temporal Convolution for Real-Time Keyword Spotting on Mobile Devices," in *Proc. Interspeech 2019*, pp. 3372–3376, 2019.

[57] A. Coucke, M. Chlieh, T. Gisselbrecht, D. Leroy, M. Poumeyrol, and T. Lavril, "Efficient keyword spotting using dilated convolutions and gating," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 6351–6355, IEEE, 2019.

[58] Y. Chen, T. Ko, L. Shang, X. Chen, X. Jiang, and Q. Li, "Meta learning for few-shot keyword spotting," *arXiv preprint arXiv:1812.10233*, 2018.

[59] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[60] P. Warden, *Launching the speech commands dataset.*, 2017.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[62] A. Parnami and M. Lee, "Few-shot keyword spotting with prototypical networks," *arXiv preprint arXiv:2007.14463*, 2020.

[63] Z. Chen and B. Liu, "Lifelong machine learning, second edition," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 2018.

[64] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015.

[65] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, *et al.*, "A general reinforcement learning

algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[66] A. Antoniou, M. Patacchiola, M. Ochal, and A. Storkey, "Defining benchmarks for continual few-shot learning," *arXiv preprint arXiv:2004.11967*, 2020.

[67] S. Xie, Y. Li, D. Lin, T. L. Nwe, and S. Dong, "Meta module generation for fast few-shot incremental learning," in *IEEE/CVF International Conference on Computer Vision Workshop*, pp. 1381–1390, 2019.

[68] C. Le, X. Wei, B. Wang, L. Zhang, and Z. Chen, "Learning continually from low-shot data stream," *arXiv preprint arXiv:1908.10223*, 2019.

[69] A. Frikha, D. Krompaß, H.-G. Köpken, and V. Tresp, "Few-shot one-class classification via meta-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, pp. 7448–7456, 2021.

[70] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Neural Information Processing Systems*, 2017.

[71] A. Galassi, M. Lippi, and P. Torroni, "Attention in natural language processing," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 10, pp. 4291–4308, 2020.

[72] A. Parnami, M. Deshpande, A. K. Mishra, and M. Lee, "Transformation of node to knowledge graph embeddings for faster link prediction in social networks," *arXiv preprint arXiv:2111.09308*, 2021.

[73] A. V. Joshi, "Amazon's machine learning toolkit: Sagemaker," in *Machine Learning and Artificial Intelligence*, pp. 233–243, Springer, 2020.

[74] E. Bisong, "Google automl: cloud vision," in *Building Machine Learning and Deep Learning Models on Google Cloud Platform*, pp. 581–598, Springer, 2019.

[75] A. Team, "Azureml: Anatomy of a machine learning service," in *Conference on Predictive APIs and Apps*, pp. 1–13, PMLR, 2016.

[76] Y. Wang, Q. Yao, J. T. Kwok, and L. M. Ni, "Generalizing from a few examples: A survey on few-shot learning," *ACM Comput. Surv.*, vol. 53, jun 2020.

[77] "Few-shot image classification on mini-imagenet 5-way (5-shot)," 2021. Available at https://paperswithcode.com/sota/few-shot-image-classification-on-mini-3.

[78] J. Cabrero-Holgueras and S. Pastrana, "Sok: Privacy-preserving computation techniques for deep learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2021, no. 4, pp. 139–162, 2021.

[79] C. Dwork and A. Roth, "The algorithmic foundations of differential privacy," *Foundation and Trends in Theoretical Computer Science*, vol. 9, pp. 211–407, aug 2014.

[80] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pp. 308–318, 2016.

[81] E. Bagdasaryan, O. Poursaeed, and V. Shmatikov, "Differential privacy has disparate impact on model accuracy," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[82] L. Fan, "Image pixelization with differential privacy," in *Data and Applications Security and Privacy XXXII*, pp. 148–162, Springer International Publishing, 2018.

[83] R. McPherson, R. Shokri, and V. Shmatikov, "Defeating image obfuscation with deep learning," *arXiv preprint arXiv:1609.00408*, 2016.

[84] Y. Xie, H. Wang, B. Yu, and C. Zhang, "Secure collaborative few-shot learning," *Knowledge-Based Systems*, vol. 203, p. 106157, 2020.

[85] Z. Xue, S. Yang, M. Huai, and D. Wang, "Differentially private pairwise learning revisited," in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pp. 3242–3248, International Joint Conferences on Artificial Intelligence Organization, 8 2021. Main Track.

[86] M. Huai, D. Wang, C. Miao, J. Xu, and A. Zhang, "Pairwise learning with differential privacy guarantees," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 694–701, Apr. 2020.

[87] R. Gelbhart and B. I. Rubinstein, "Discrete few-shot learning for pan privacy," *arXiv preprint arXiv:2006.13120*, 2020.

[88] K. Mivule, "Utilizing noise addition for data privacy, an overview," *arXiv preprint arXiv:1309.3958*, 2013.

[89] J. J. Kim, "A method for limiting disclosure in microdata based on random noise and transformation," in *Proceedings of the section on survey research methods*, pp. 303–308, American Statistical Association Alexandria, VA, 1986.

[90] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020. Available at https://doi.org/10.1038/s41586-020-2649-2.

[91] S. Hill, Z. Zhou, L. Saul, and H. Shacham, "On the (in)effectiveness of mosaicing and blurring as tools for document redaction," in *Proceedings on Privacy Enhancing Technologies*, vol. 2016, pp. 403–417, 2016.

[92] P. Gwosdek, S. Grewenig, A. Bruhn, and J. Weickert, "Theoretical foundations of gaussian convolution by extended box filtering," in *Scale Space and Variational Methods in Computer Vision*, pp. 447–458, Springer, 2012.

[93] A. Clark, "Pillow (pil fork) documentation," 2015. Available at https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf.

[94] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142–3155, 2017.

[95] B. Lake, R. Salakhutdinov, J. Gross, and J. Tenenbaum, "One shot learning of simple visual concepts," in *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 33, 2011.

[96] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *Proceedings of International Conference on Computer Vision*, December 2015.

[97] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *2015 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 815–823, 2015.

[98] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint face detection and alignment using multitask cascaded convolutional networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[99] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, “Image quality assessment: from error visibility to structural similarity,” *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[100] L. Van der Maaten and G. Hinton, “Visualizing data using t-sne.,” *Journal of machine learning research*, vol. 9, no. 11, 2008.

[101] A. Parnami, M. Usama, L. Fan, and M. Lee, “Privacy enhancement for cloud-based few-shot learning,” *arXiv preprint arXiv:2205.07864*, 2022.

[102] Y. Xiao, N. Hou, and E. S. Chng, “Rainbow keywords: Efficient incremental learning for online spoken keyword spotting,” *arXiv preprint arXiv:2203.16361*, 2022.

[103] Y. Huang, N. Hou, and N. F. Chen, “Progressive continual learning for spoken keyword spotting,” in *IEEE International Conference on Acoustics, Speech and Signal Processing*, pp. 7552–7556, IEEE, 2022.

[104] P. Desai, P. Lai, N. Phan, and M. T. Thai, “Continual learning with differential privacy,” in *International Conference on Neural Information Processing*, pp. 334–343, Springer, 2021.