Few Shot Meta Learning

Pinakin Choudhary
Vansh Dhar
Mehul Shrivastava
Aayush Patil

1 Introduction

The paper explores the paradigm of Few-Shot Learning, a crucial aspect of general artificial intelligence enabling machines to recognize new concepts with minimal training data, mirroring human learning. It delves into Meta-Learning, also known as "learning to learn," which trains algorithms on a series of tasks, allowing them to generalize to new classes with only a few examples. Key components such as support and query sets are discussed, alongside the evaluation of various meta-learning algorithms like Model Agnostic Meta-Learning and Metric Learning, which show promising results in addressing the few-shot learning problem. The paper emphasizes the theoretical foundations and practical implications of Few-Shot Meta Learning, inspired by human intelligence's ability to learn quickly from a small number of examples. It introduces the concept of training models on a variety of tasks to enable learning new tasks from only a few examples, aligning more closely with human-like learning algorithms. The framework typically involves tasks structured as $\mathcal{N}$ -way and $\mathcal{K}$-shot, aiming to classify query samples into classes based on support samples. The paper highlights the consistency between training and testing objectives as a key advantage of meta-learning in few-shot classification tasks.

2 Main Idea

Prototypical networks are based on the idea that there exists an embedded space where the data points for each class cluster around a single representative point, called a prototype. To achieve this, a neural network is used to learn a non-linear mapping of the input data into this embedding space. The prototype for each class is then defined as the mean of the data points belonging to that class in the embedding space.

Prototypical networks generate a representative vector, called a *prototype*, for each class using an embedding function $f_\phi : \mathbb{R}^D \to \mathbb{R}^M$ with learnable parameters ϕ. Each prototype is calculated as the average vector of the embedded support points belonging to its respective class:

$$\mathbf{c}_k = \frac{1}{|S_k|} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i)$$

Given a distance function $d : \mathbb{R}^M \times \mathbb{R}^M \to [0, +\infty)$, prototypical networks estimate the probability distribution over classes for a query point $\mathbf{x}$ by comparing distances to the prototypes in the embedding space:

$$p_\phi(y = k \mid \mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'}))}$$

The learning process involves minimizing the negative log-probability $J(\phi) = -\log p_\phi(y = k \mid \mathbf{x})$ of the correct class k using Stochastic Gradient Descent (SGD). During training, episodes are constructed

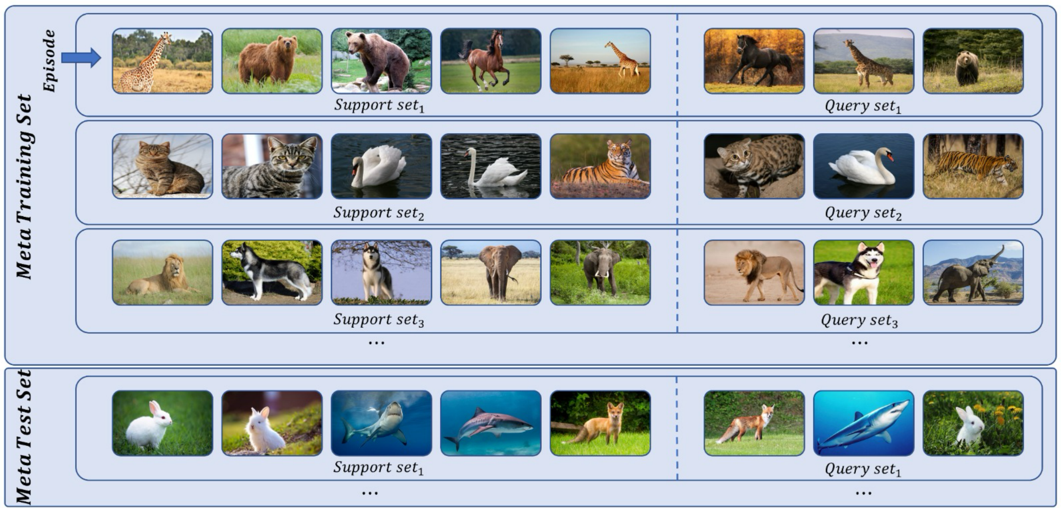
.

by randomly selecting a subset of classes from the training set. Then, within each class, a subset of examples is chosen to form the support set, and the remaining examples are designated as query points. The pseudocode for computing the loss $J(\phi)$ for a training episode is provided in Algorithm below taken from the paper.

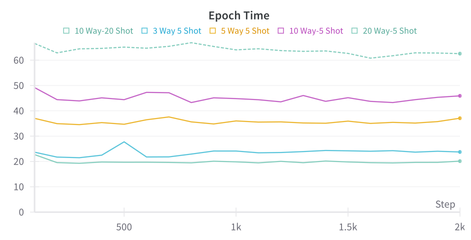
Algorithm 1 Training episode loss computation for prototypical networks. N is the number of examples in the training set, K is the number of classes in the training set, $N_C \leq K$ is the number of classes per episode, N_S is the number of support examples per class, N_Q is the number of query examples per class. RANDOMSAMPLE(S, N) denotes a set of N elements chosen uniformly at random from set S, without replacement.

```
Input: Training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where each $y_i \in \{1, \ldots, K\}$. $\mathcal{D}_k$ denotes the
  subset of $\mathcal{D}$ containing all elements $(\mathbf{x}_i, y_i)$ such that $y_i = k$.
Output: The loss $J$ for a randomly generated training episode.
  $V \leftarrow$ RANDOMSAMPLE$(\{1, \ldots, K\}, N_C)$                    ▷ Select class indices for episode
  for $k$ in $\{1, \ldots, N_C\}$ do
    $S_k \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_{V_k}, N_S)$                          ▷ Select support examples
    $Q_k \leftarrow$ RANDOMSAMPLE$(\mathcal{D}_{V_k} \setminus S_k, N_Q)$                     ▷ Select query examples
    $\mathbf{c}_k \leftarrow \frac{1}{N_C} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_{\boldsymbol{\phi}}(\mathbf{x}_i)$       ▷ Compute prototype from support examples
  end for
  $J \leftarrow 0$                                                     ▷ Initialize loss
  for $k$ in $\{1, \ldots, N_C\}$ do
    for $(\mathbf{x}, y)$ in $Q_k$ do
      $J \leftarrow J + \frac{1}{N_C N_Q} \left[ d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) + \log \sum_{k'} \exp(-d(f_{\boldsymbol{\phi}}(\mathbf{x}), \mathbf{c}_k)) \right]$    ▷ Update loss
    end for
  end for
```

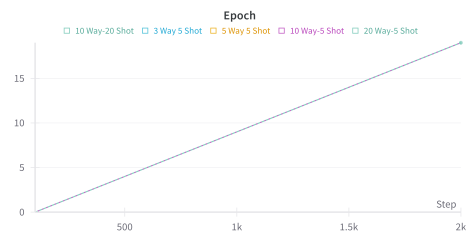
The distance function used by our code implementation is Eucledian Distance.



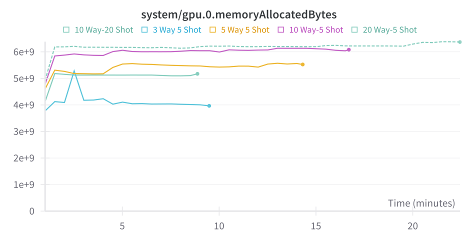
3 Results:



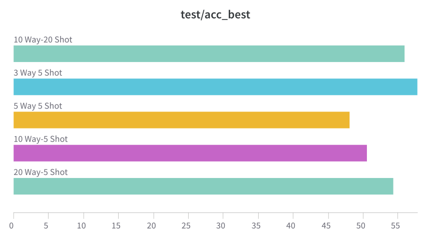
(a) Epoch Time Comparison



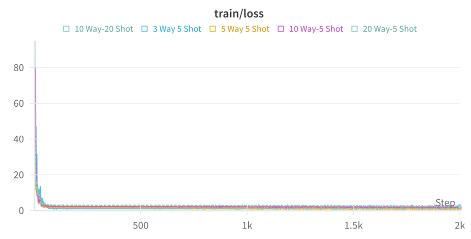
(b) Number of Epochs Comparison



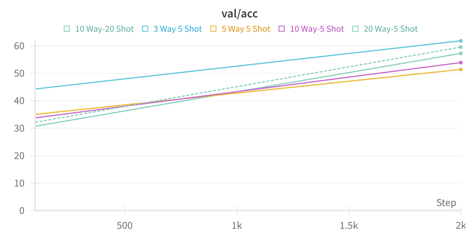
(c) GPU Memory Allocation



(d) Test Accuracy



(e) Training Loss



(f) Validation Loss

Figure 2: Experiment Results

Important Deductions:

- We found that when we use more categories for training, the accuracy is better compared to using less categories. We think this happens because having more categories makes it harder for the model to learn, which actually helps it to understand things better overall. This is because the model has to be more precise in deciding which category something belongs to.
- As we increased number of ways, the GPU memory allocation increased significantly, limiting our capabilities to train model more efficiently.

4 References:

1. https://arxiv.org/pdf/1703.05175.pdf
2. https://arxiv.org/pdf/1703.03400.pdf
3. https://github.com/ashok-arjun/MLRC-2021-Few-Shot-Learning-And-Self-Supervision?tab=readme-ov-file
4. https://www.robots.ox.ac.uk/~vgg/data/flowers/102/