|  |
| --- |
| Lab 3: Few-Shot Classification |

**1. Introduction**

We want to design five few-shot classifiers for character classification in five alphabets: Atlantean, Japanese (Hiragana), Japanese (Katakana), Korea, ULOG. Alphabets contain 26, 52, 47, 40, 26 characters, each of which consist of 20 grayscale images of size 32 x 32. Among 20 images, 5 images are used for training and 5 are used for evaluation. Thus, tasks are 26-way 5-shot, 52-way 5-shot, 47-way 5-shot, 40-way 5-shot, 26-way 5 shot problems, respectively. Can we build good classifiers for these tasks?

**2. Problem 3.1: Implementing the omniglot\_prototype\_collarate\_fn function**

In PyTorch, given a data index, the Dataset object processes data with that index. If so, who is providing such an index? The answer is the sampler object in the DataLoader object. Therefore, in standard way, we should implement mini batch sampler for DataLoader.

But to decrease the level of difficulty in this lab, I implemented it a little different way. The OmniglotPrototype class in "datasets.py" contains a nested list. The first list contains the list of images belonging to the same character. So, given index (character index), the Dataset will return the image of specific character with that index. If the Dataset gives the K images, then we can easily yield a N way K shot task by using the standard DataLoader with batch size N. But the problem is the label. Because we cannot determine the true label in the Dataset. For example, suppose that the Dataset has 1000 classes. Also, we generate a new task by sampling 1, 3, 7, 10, 20 classes (5 way). If so, the query examples of class 3 should have label 1 instead of 3, because it is the second class in the new task. However, the Dataset class doesn't know where the given index will actually be in the batch. So, we should revise it as a post-processing. The collate\_fn in DataLoader is a function that do just that.

The omniglot\_prototype\_collararate\_fn takes batched data as its inputs. The batched data is 3-dimensinal list: (batch, support\_image / query\_image / query\_label, actual data). By implementing it, complete the data processing pipeline for episodic training algorithm. If the implementation is correct, the DataLoader will return support input tensors of size N x K\_s x 1 x 32 x 32, query input tensors of size N x K\_q x 1 x 32 x 32, and query target tensors of size N x K\_q, where N is the number of classes for each episode (task), K\_s is the number of support examples, and K\_q is the number of query example.

**3. Problem 3.2: Implementing the prototypical network**  
In this problem, you should implement three functions:

텍스트이(가) 표시된 사진

자동 생성된 설명

The forward function takes support examples (x\_s) and query examples (x\_q). If you implement correct omniglot\_prototype\_collararate\_fn, both x\_s and x\_q will be 5-dimensional tensors. So, to use the standard 2d convolution layers in EmbeddingNet, we need to change their shape. The hint is that we can consider the first two dimensions (N, K) as batch. Because the operations of convolution layers and linear layers will perform only the last three dimensions. Note that the forward function should give the negative Euclidean distance between query examples and prototypes. So, the y\_q is a tensor of size N x K\_q x N

The compute\_prototy function takes z\_s, which is a tensor of size N x K\_s x 256. Here, 256 is the output channels of the embedding network. The number of prototypes will be N. So, the output tensor of this function should have shape N x 256

The compute\_distance function returns the squared Euclidean distance between z\_q and c\_s. So, the ouput tensor of this function should be (N x K\_q) x N