Prototype networks for few-shot learning

Fabian Greavu, Machine Learning exam @ unifi

Summary

Intro

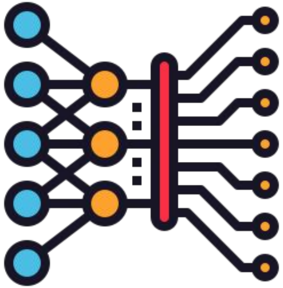
Challenge

'classic' Classification Task

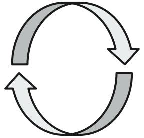
Dataset

(REALLY BIG)

Training / Fine Tuning



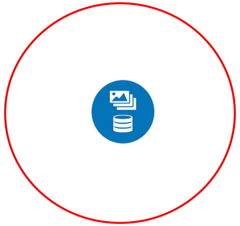
(Specialized into X classes)



Deploy

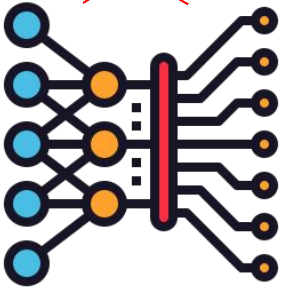
Our Challenge Classification Task

Dataset



(REALLY SMALL)

Training ~~/ Fine~~ Tuning

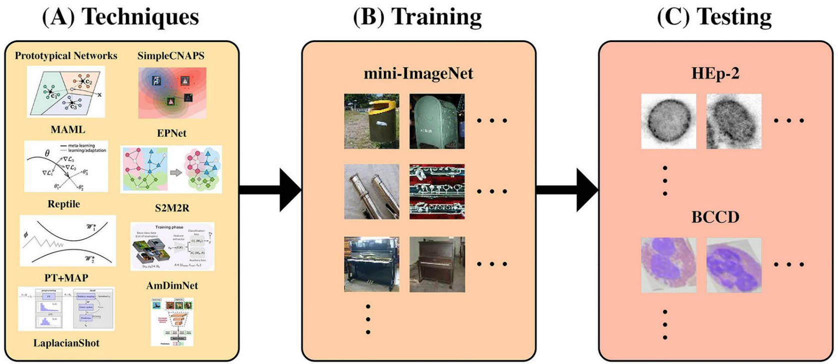


no enough occurrences

• • •

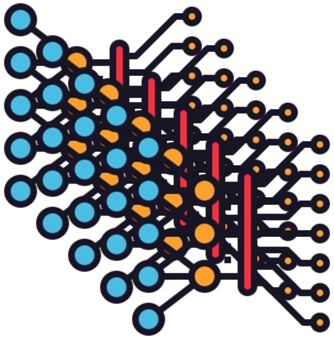
Few-shot learning

- A **meta-learning** technique to ‘learn to learn’
- Uses **small amount** of occurrences
- Can perform with **high accurancies**

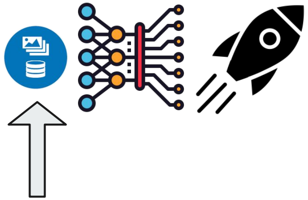


Dataset

Learn to learn



Learn + Deploy



small

Meta-learning definition

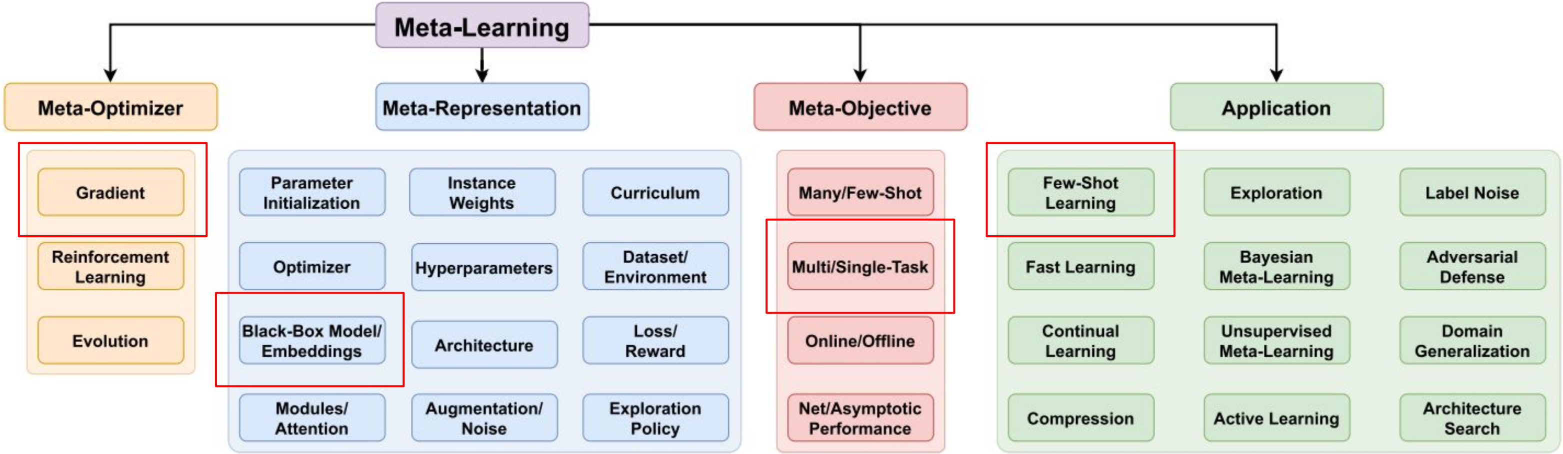


Fig. 1. Overview of the meta-learning landscape including algorithm design (meta-optimizer, meta-representation, meta-objective), and applications.

Meta-Learning in Neural Networks: A Survey, Hospedales, Antoniuou, Micarelli, Snorkey, 2022

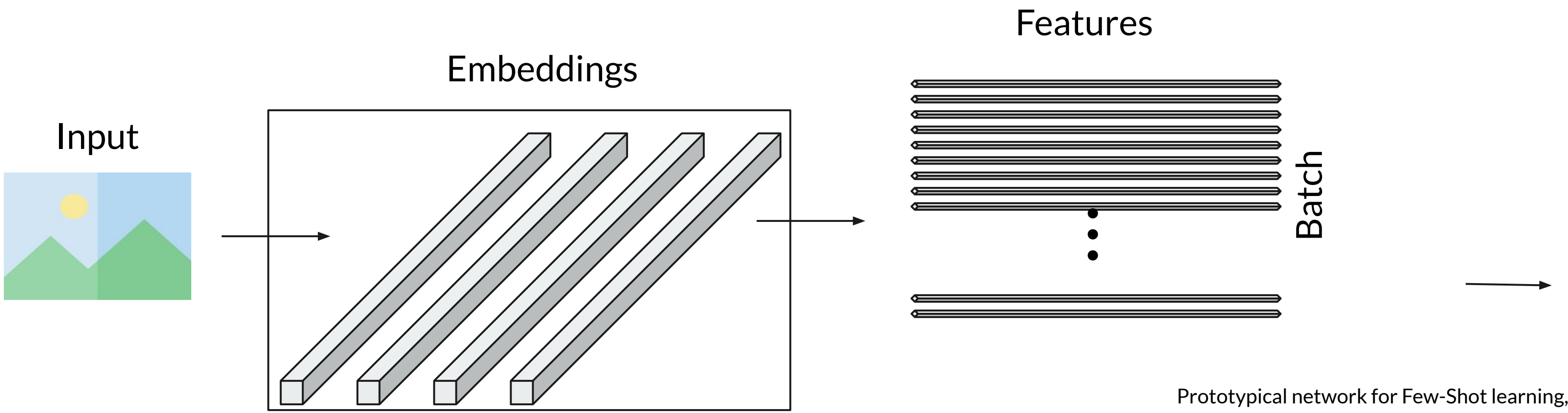
Project workflow

Datasets

- Omniglot: 1623 handwritten characters
  - 80 images per class
  - classes: (1032 train, 172 val, 464 test)
- Mini Imagenet: 100 objects
  - 600 images per class
  - classes: (64 train, 16 val, 20 test)
- Flowers102: 102 flowers
  - 40-120 images per class
  - classes: (64 train, 16 val, 22 test)

Prototypical Networks

- Extracts features with a neural network
  - 4 CNN blocks: Conv out=64, ks=3 / BatchNorm2D / ReLu / MaxPool2D
- Learning: embeddings learning
  - Trained 100 episodes/iterations per epochs (200 ep) to learn embeddings



Prototypical network for Few-Shot learning, Snell, Swesky, Zemel, 2017

Prototypical Networks - centroids

- Use a small part of output as support, other as query
- Centroids are mean(support)
- Calculate distances between query and centroids
- Calculate loss as mean of log_softmax of negative distances

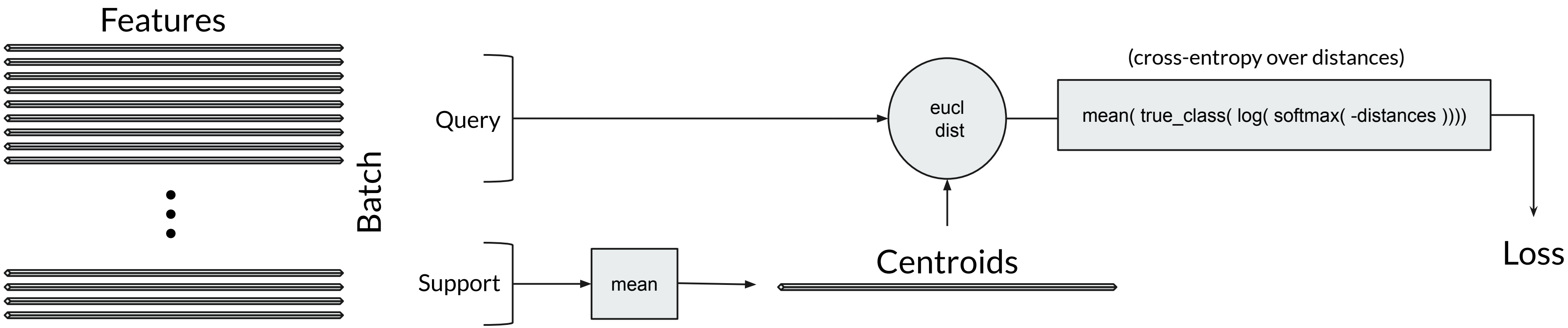
Prototypical networks compute an M-dimensional representation $\mathbf{c}_k \in \mathbb{R}^M$, or *prototype*, of each class through an embedding function $f_\phi : \mathbb{R}^D \rightarrow \mathbb{R}^M$ with learnable parameters ϕ. Each prototype is the mean vector of the embedded support points belonging to its class:

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

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

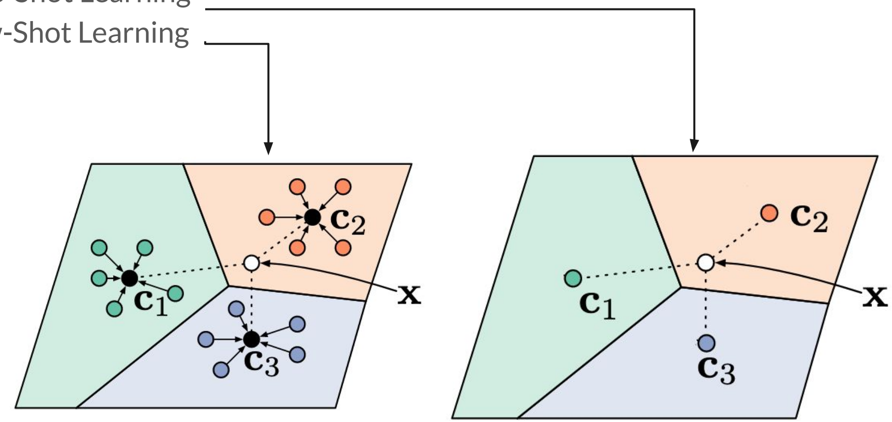
$$p_\phi(y = k \mid \mathbf{x}) = \frac{\exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k))}{\sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_{k'}))} \tag{2}$$

Learning proceeds by minimizing the negative log-probability $J(\phi) = -\log p_\phi(y = k \mid \mathbf{x})$ of the true class k via SGD. Training episodes are formed by randomly selecting a subset of classes from the training set, then choosing a subset of examples within each class to act as the support set and a subset of the remainder to serve as query points. Pseudocode to compute the loss $J(\phi)$ for a training episode is provided in Algorithm 1



Prototypical Networks - NC, NS, NQ

- NC: how many classes to use per each iteration on batch (or ‘ways’)
- NS: how many examples to use as support for centroids calculus (or ‘shots’)
- NQ: how many examples to use as queries for centroids calculus (‘query’)
  - 1 shots -> One-Shot Learning
  - 5 shots -> Few-Shot Learning



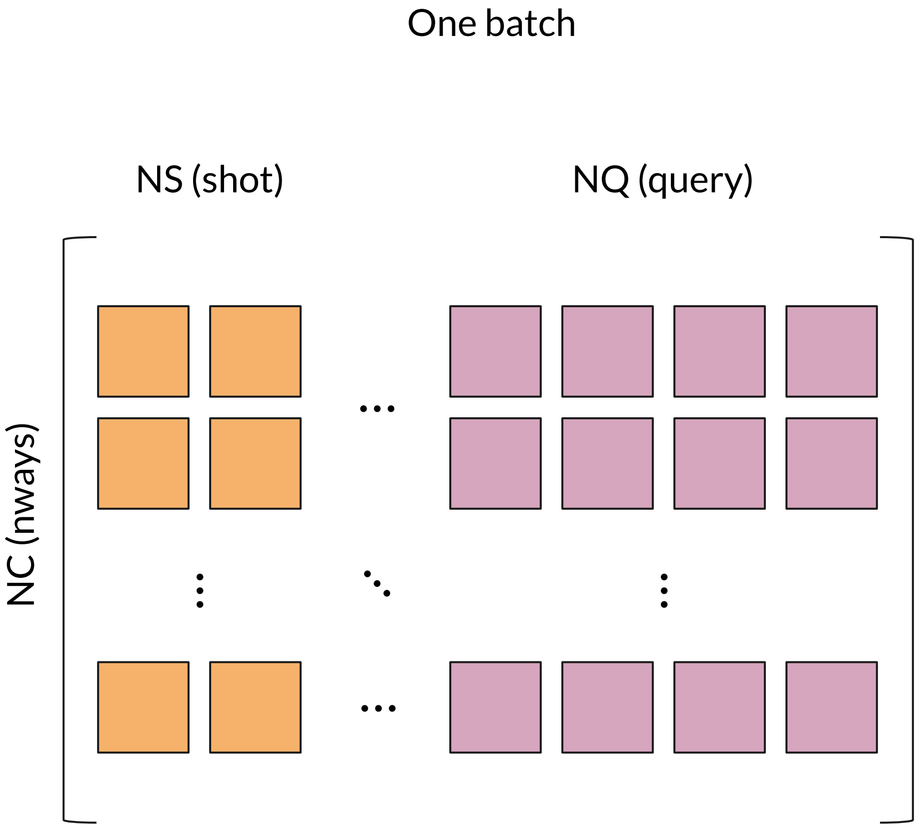
Loss, optimizer, scheduler

- Loss: mean(log_softmax(-distances(query, centroids), targets)
- Optimizer: Adam with lr=0.001 (faster and similar results to SGD)
- Scheduler: StepLr with step_size=20, gamma = 0.5

Training skeleton

```
model = PrototypicalNet()
loss = loss_function(x, y, NC, NS, NQ)
optim = StepLr()

for epoch in epochs:
 for it in iterations:
  x, y = GetSample(NC, NS, NQ)
  out = model(x)
  ... loss
  backward()
 for it in iterations:
  .... eval ....
```



Code

All code is available at [github]

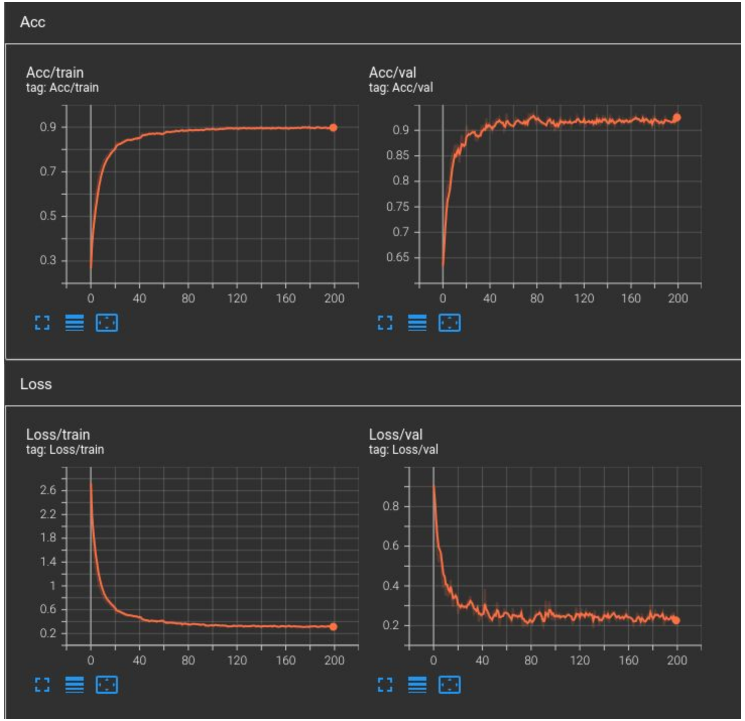
- Full hyperparams control
- 3 available datasets
- results and plots
- simple train.py and test.py scripts

```
python train.py --dataset mini_imagenet \
          --epochs 200 \
          --gpu \
          --train-num-class 30 \
          --test-num-class 5 \
          --number-support 5 \
          --train-num-query 15 \
          --episodes-per-epoch 100 \
          --adam-lr 0.001 \
          --opt-step-size 20 \
          --opt-gamma 0.5 \
          --distance-function "euclidean" \
          --save-each 5
```

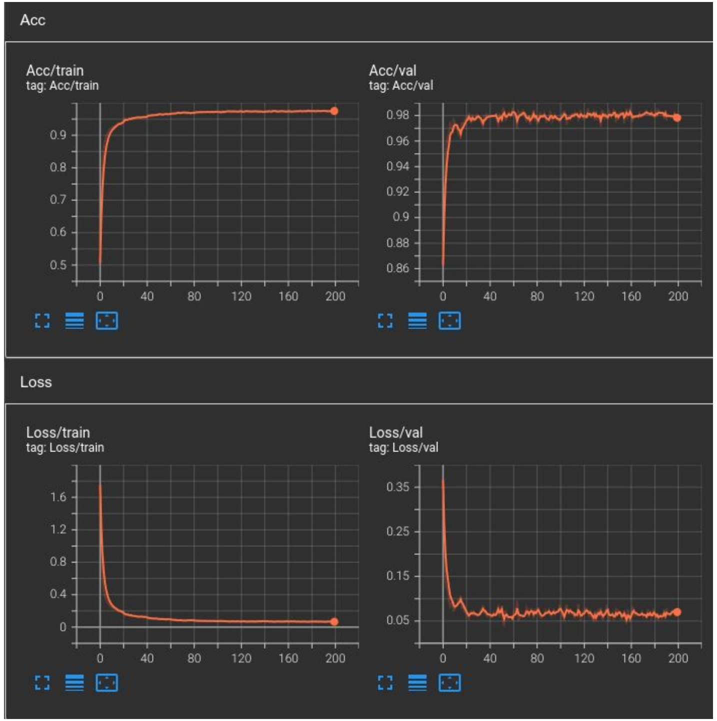
Experiments

Omniglot

1-shot

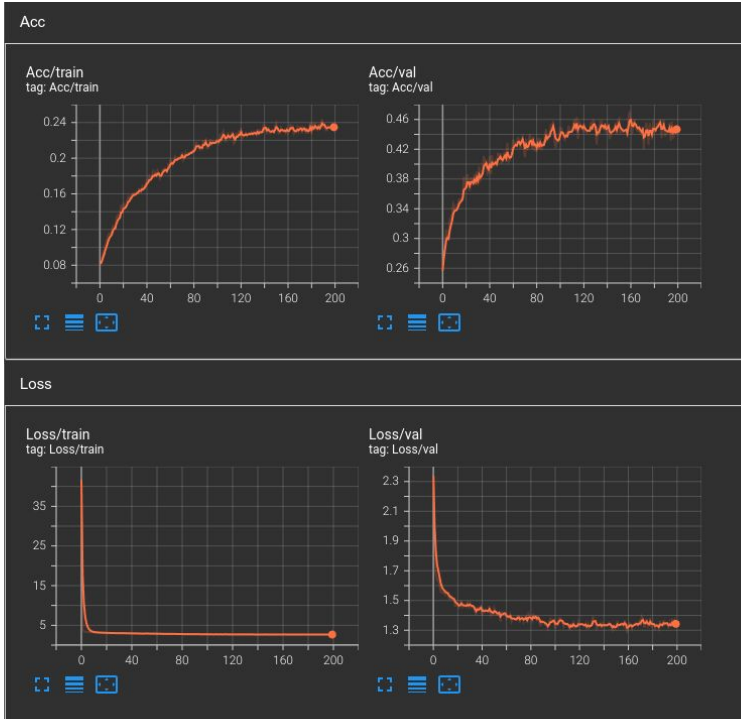


5-shot

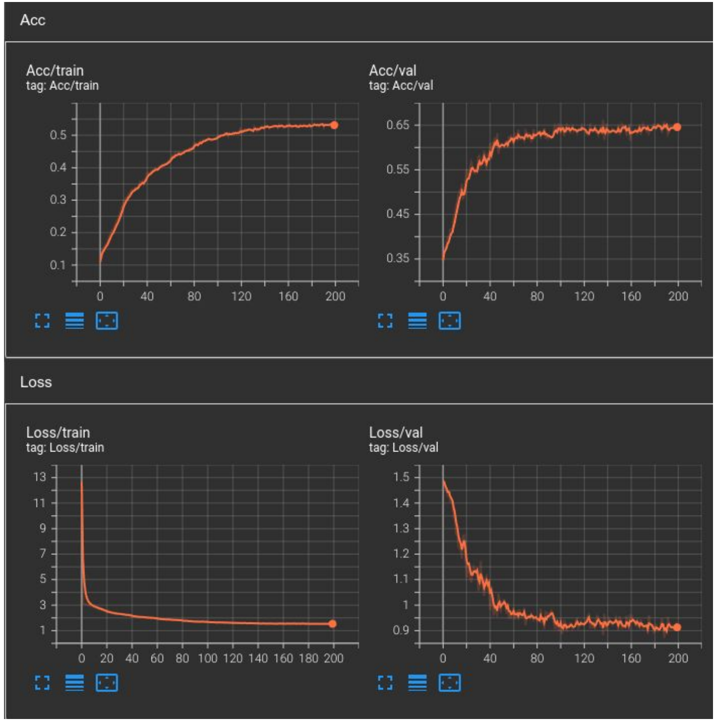


Mini Imagenet

1-shot

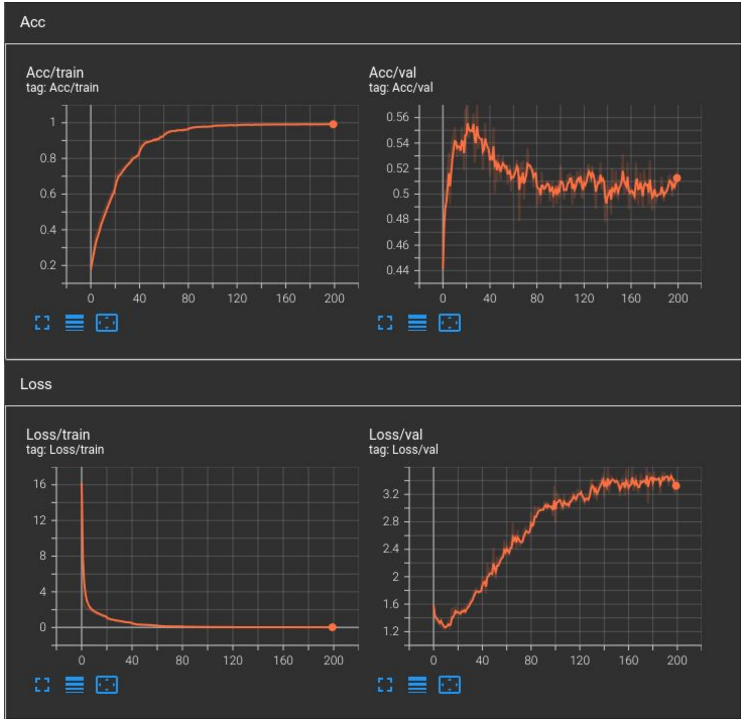


5-shot



Flowers 102

1-shot



5-shot

Comparison - distance metric

Dataset	Cosine (acc)	Euclidean (acc)
mini_imagenet	22.36	**63.62**
omniglot	23.48	**97.77**
flowers102	82.89	**84.48**

Comparison - one vs few shot vs paper

Dataset	Paper res 5-way 5-shot (Acc)	Our res 5-way 5-shot (Acc)	Paper res 5-way 1-shot (Acc)	Our res 5-way 1-shot (Acc)
mini_imagenet	68.20	63.62	49.42	46.13
omniglot	98.80	97.77	98.8	91.93
flowers102	/	84.48	/	56.08

Conclusions

Conclusion

- Euclidean distance performs better than cosine similarity
- Paper results were correctly replicated

Future studies

- Add custom dataset option for training
- Implement proper torch.nn.Dataset and torch.nn.Sampler + torch.nn.DataLoader
- Try different fields than CV