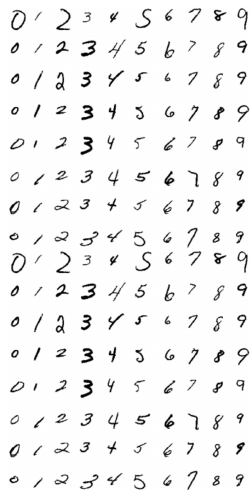


# Meta Learning at a Glance

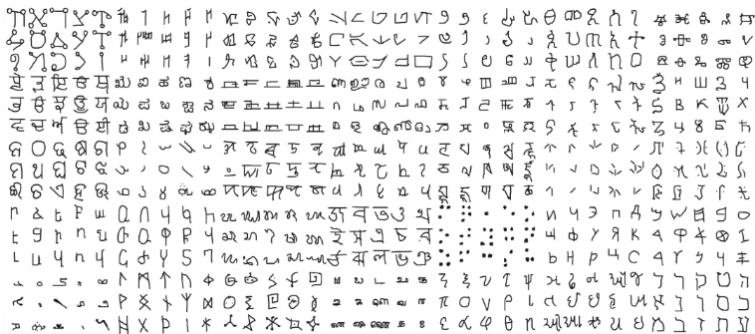
Seminar 2 on *Nôm OCR*

# Problem statement: MNIST vs. Omniglot



MNIST:

few classes, many examples

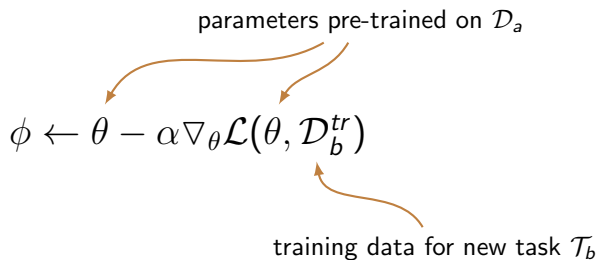


Omniglot:

many classes, few examples

# Transfer learning via fine-tuning approach

**Key idea:** Solve target task  $\mathcal{T}_b$  after solving source task(s)  $\mathcal{T}_a$  by *transferring* knowledge learned from  $\mathcal{T}_a$ .



The diagram illustrates the fine-tuning process. It features the equation  $\phi \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_b^{tr})$  in the center. Above the equation, the text "parameters pre-trained on  $\mathcal{D}_a$ " has two orange arrows pointing to the  $\theta$  and  $\nabla_{\theta}$  terms. Below the equation, the text "training data for new task  $\mathcal{T}_b$ " has an orange arrow pointing to the  $\mathcal{D}_b^{tr}$  term.

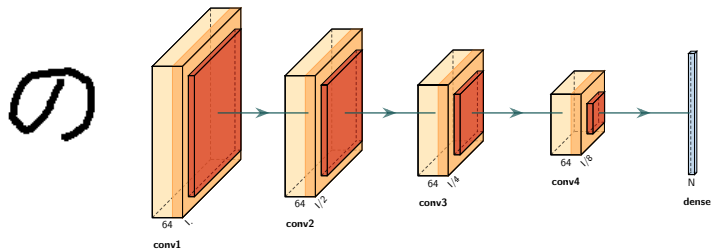
$$\phi \leftarrow \theta - \alpha \nabla_{\theta} \mathcal{L}(\theta, \mathcal{D}_b^{tr})$$

Common assumption: Cannot access data  $\mathcal{D}_a$  during transfer.

Some problems/applications where transfer learning might make sense:

When $\mathcal{D}_a$ is very large and $\mathcal{D}_b$ is somehow smaller.	When you don't care about solving $\mathcal{T}_a$ & $\mathcal{T}_a$ simultaneously.
--	---

# Transfer learning on Omniglot



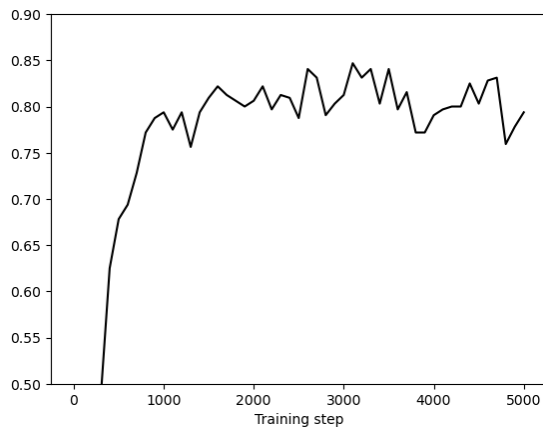
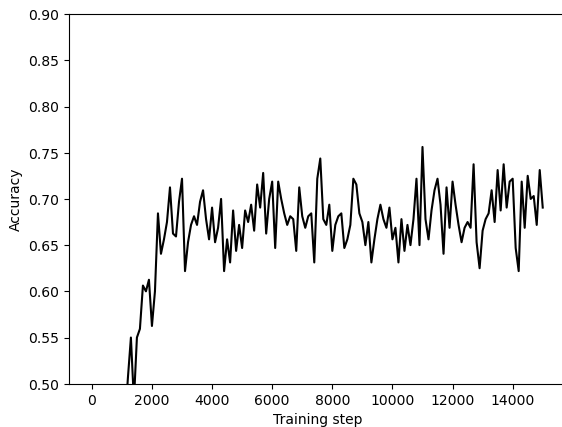
Model's backbone consists of four CNNs with ReLUs and Maxpooling layers

**Pre-training split:** contains  $N = 1200$  characters/classes.

**Fine-tuning split:** contains another  $N = 423$  characters/classes.

**The goal:** is to obtain a model that is initialized with the pre-trained parameters and can distinguish between new 423 classes after being fine-tuned.

# How does transfer learning work on Omniglot?



Accuracy of pre-training vs. fine-tuning phases on validation sets

# Meta Learning

Learning to Learn

# From Transfer Learning to Meta-Learning


**Transfer learning:** Initialize model. Hope that it helps the target task.

**Meta-learning:** Can we explicitly *optimize* for transferability?

Given a set of training tasks, can we optimize for the ability to learn these tasks quickly?  
so that we can learn new tasks quickly too.

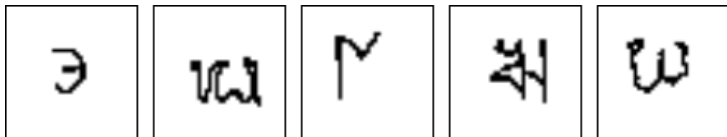
**Scenario:** what if dataset  $\mathcal{D}_i^{tr}$  for task  $\mathcal{T}_i$  is small?  
(number of examples per class is small)

Learning a task:  $\mathcal{D}_i^{tr} \xrightarrow{f_\theta} \theta$

 Can we optimize this function  $f_\theta$ ?

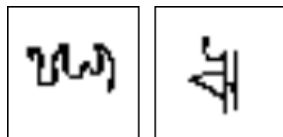
# How does meta-learning work? An example

Given 1 example of 5 classes:



training data  $\mathcal{D}^{tr}$

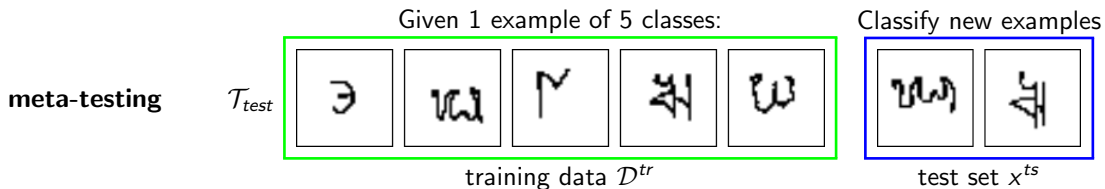
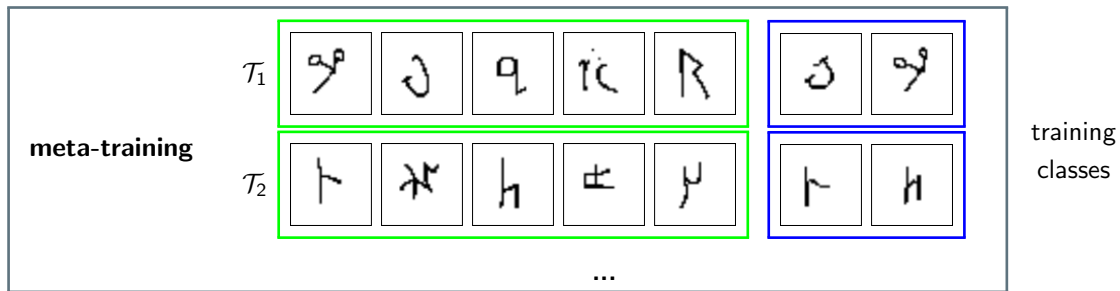
Classify new examples



test set  $x^{ts}$



# How does meta-learning work? An example



Can replace image classification with: regression, language generation, **any ML problem**.

# Meta Learning problem

Transfer Learning with *Many Source Tasks*

Given data from  $\mathcal{J}_1, \mathcal{J}_2, \dots, \mathcal{J}_n$ , solve new task  $\mathcal{J}_{test}$  more quickly/proficiently/stably.

**Key assumption:** meta-training tasks and meta-test task drawn i.i.d. from same task distribution

$$\mathcal{J}_1, \dots, \mathcal{J}_n \sim p(\mathcal{J}), \mathcal{J}_{test} \sim p(\mathcal{J}).$$

Tasks must share underlying structure

What do the tasks correspond to?

- ▶ Recognizing handwritten digits from different languages,
- ▶ Giving feedback to different students on different exams,
- ▶ Classifying flower species in different regions of the world,
- ▶ A robot performing different tasks.

How many tasks do we want for a training stage?

The more the better.

# Some terminologies

task training set  $\mathcal{D}_i^{tr}$  "support set"

task test set  $\mathcal{D}_i^{test}$  "query set"



**k-shot learning:** learning with k examples per class

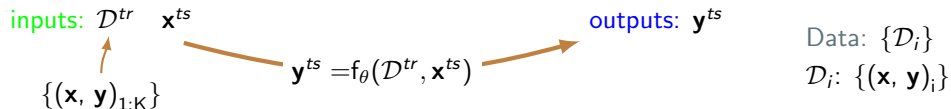
**N-way classification:** choosing between N classes

# Meta-Learning formulation

## Conventional supervised learning:



## Meta supervised learning:



$\theta$  is called meta-parameters

# Meta-Learning on Omniglot

**Model:** Using the *backbone* mentioned in the slide earlier.

Training and evaluating meta-learning algorithms on **5-way 1-shot** tasks with **15 query examples** per task.

**Data:**\*

- ▶ Training set: contains 1100 characters/classes,
  - ▶ Validation set: contains 100 characters/classes,
  - ▶ Testing set: contains 423 characters/classes.
- } exactly from the *pre-training split*

*Only some minor changes in the training setup are introduced to produce a highly fair comparison.*

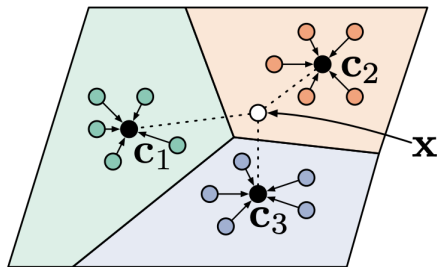
\* *The three sets are disjoint with each other, of course.*

## Some Meta-Learning Algorithms

- ▶ Prototypical Networks
- ▶ Model-Agnostic Meta-Learning
- ▶ Proto-MAML

# Prototypical Networks concept

**Key idea:** *Mapping* images to features such that images of same class are *close to each other* in feature space.



Snell, Jake, Kevin Swersky, and Richard Zemel. "Prototypical networks for few-shot learning." *Advances in neural information processing systems* 30 (2017).

# ProtoNet formulations

Using squared Euclidean distance to **measure distance** between  $f_{\theta}(x)$  of an image  $x$  and each of the prototypes:

$$d(f_{\theta}(x), c_n) = \|f_{\theta}(x) - c_n\|^2$$

Applying the softmax operation to **classify the image** by obtaining the proper probabilities over classes:

$$p(x, c_n) = p(y = n|x, c_n) = \frac{\exp(-d(f_{\theta}(x), c_n))}{\sum_{n'=1}^N \exp(-d(f_{\theta}(x), c_{n'}))}$$



# ProtoNet algorithm

---

**Algorithm** ProtoNet for  $N$ -way  $K$ -shot Meta Supervised Learning

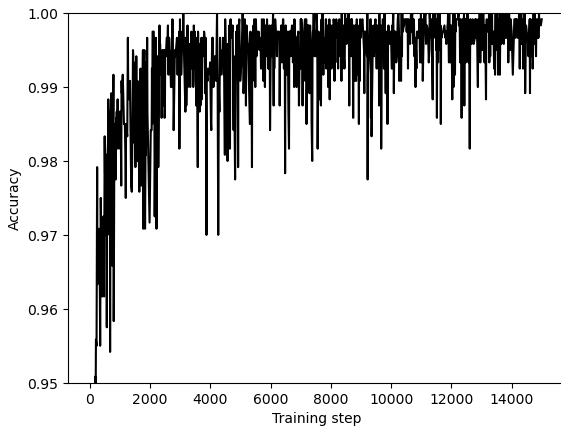
---

**Require:**  $p(\mathcal{T})$ : distribution over tasks

**Require:**  $\alpha$ : learning rate

- 1: Randomly initialize meta-parameters  $\theta$
- 2: **while** not done **do**
- 3:     Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$
- 4:     **for all**  $\mathcal{T}_i$  **do**
- 5:         Sample  $K$  datapoints  $\mathcal{D}_i^{tr}$  from  $\mathcal{T}_i$
- 6:         Sample disjoint datapoints  $\mathcal{D}_i^{query}$  from  $\mathcal{T}_i$
- 7:         Compute prototypes  $c_{n \in 1:N} \leftarrow \frac{1}{K} \sum_{(x,y)_j \in \mathcal{D}_i^{tr}: y_j = n} f_{\theta}(x_j)$
- 8:         Compute distribution over classes  $\mathcal{P}_i \leftarrow p(x_j, c_{n \in 1:N})_{x_j \in \mathcal{D}_i^{query}}$
- 9:         Compute loss  $\ell_{\mathcal{T}_i} \leftarrow \mathcal{L}(\theta, \mathcal{P}_i)$
- 10:     **end for**
- 11:     Update  $\theta \leftarrow \theta - \alpha \nabla_{\theta} \sum_{\mathcal{T}_i} \ell_{\mathcal{T}_i}$
- 12: **end while**

# ProtoNet result on Omniglot



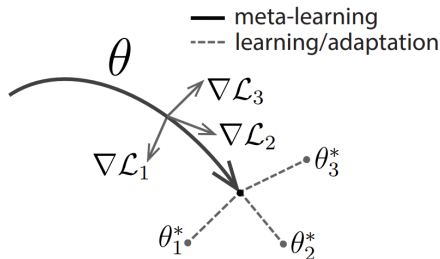
Accuracy on query set during validating

## Some Meta-Learning Algorithms

- ▶ Prototypical Networks
- ▶ Model-Agnostic Meta-Learning
- ▶ Proto-MAML

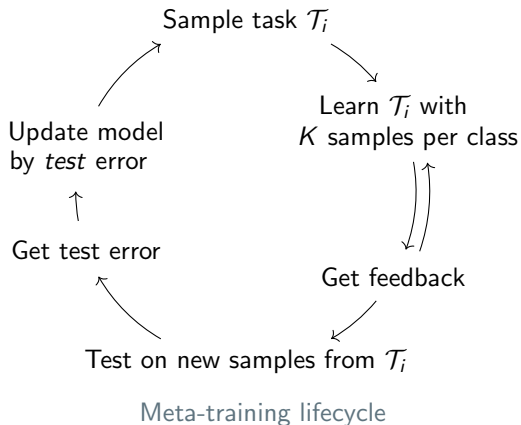
# Model-Agnostic Meta-Learning concept

**Key idea:** To meta-learn **initial**  $\theta$  that can be quickly adapted **via gradient descent** to a new task with a **small amount** of data.



Finn, Chelsea, Pieter Abbeel, and Sergey Levine. "Model-agnostic meta-learning for fast adaptation of deep networks." International conference on machine learning. PMLR, 2017.

# A high-level view of MAML



**Objective:** Learning initial parameters  $\theta$  such that *small changes* in these parameters will produce *high performance* on any unseen task  $\mathcal{J}_{test}$ .

# MAML algorithm

---

**Algorithm** MAML for  $N$ -way  $K$ -shot Meta Supervised Learning

---

**Require:**  $p(\mathcal{T})$ : distribution over tasks

**Require:**  $\alpha, \beta$ : learning rates for inner and outer loop

- 1: Randomly initialize meta-parameters  $\theta$
  - 2: **while** not done **do**
  - 3:   Sample batch of tasks  $\mathcal{T}_i \sim p(\mathcal{T})$
  - 4:   **for all**  $\mathcal{T}_i$  **do** ▷ start of outer loop
  - 5:     Sample  $K$  datapoints  $\mathcal{D}_i^{tr}$  from  $\mathcal{T}_i$
  - 6:     Sample disjoint datapoints  $\mathcal{D}_i^{query}$  from  $\mathcal{T}_i$
  - 7:     Initialize task-specific parameters  $\phi_i \leftarrow \theta$
  - 8:     Optimize  $\phi_i \leftarrow \phi_i - \alpha \nabla_{\phi_i} \mathcal{L}(\phi_i, \mathcal{D}_i^{tr})$  ▷ inner loop
  - 9:     Compute loss  $\ell_{\mathcal{T}_i} \leftarrow \mathcal{L}(\phi_i, \mathcal{D}_i^{query})$
  - 10:   **end for**
  - 11:   Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i} \ell_{\mathcal{T}_i}$  ▷ end of outer loop
  - 12: **end while**
-

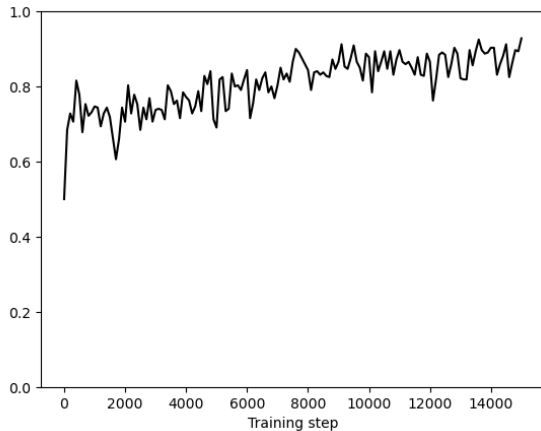
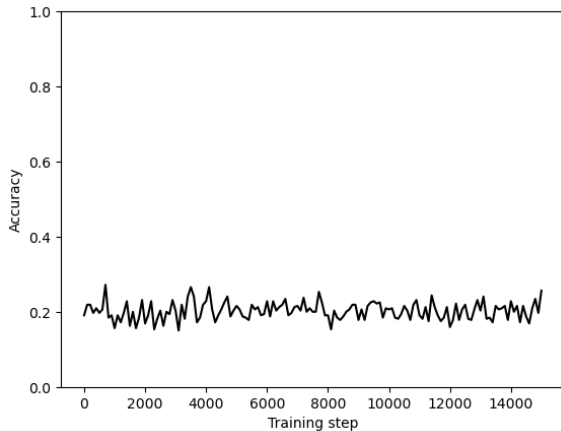
## First-order MAML approximation

Second-order derivatives!!!

11: Update  $\theta \leftarrow \theta - \beta \nabla_{\theta} \sum_{\mathcal{T}_i} \ell_{\mathcal{T}_i}$

But,  PyTorch will give you a hand

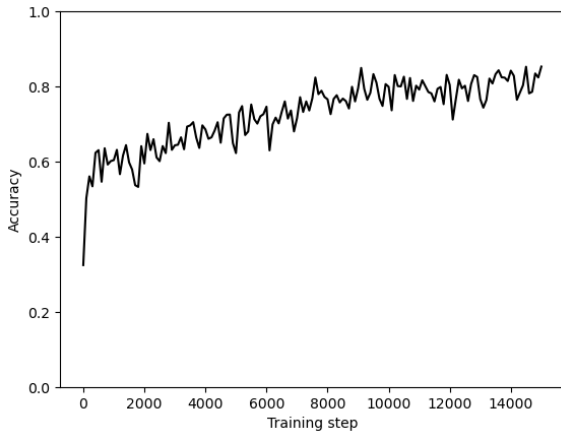
## fo-MAML results on Omniglot



Pre- and Post-adaption accuracies on support set during validating



## fo-MAML results on Omniglot



Accuracy on query set during validating


## Some Meta-Learning Algorithms

- ▶ Prototypical Networks
- ▶ Model-Agnostic Meta-Learning
- ▶ Proto-MAML

# Integrating ProtoNet's output to MAML

**Remember that** ProtoNet use Euclidean distance  $d(f_{\theta}(x), c_n)$  to classify image  $x$ ?

The **output distance** with respect to a prototype  $c_n$  of ProtoNet can be unpacked as:

$$\begin{aligned} -d(f_{\theta}(x), c_n) &= -\|f_{\theta}(x) - c_n\|^2 = -f_{\theta}(x)^{\top} f_{\theta}(x) + 2c_n^{\top} f_{\theta}(x) - c_n^{\top} c_n \\ &= 2c_n^{\top} f_{\theta}(x) - c_n^{\top} c_n + \text{constant} \end{aligned}$$


The diagram illustrates the decomposition of the distance formula into weight and bias terms. An orange arrow points from the term  $2c_n^{\top} f_{\theta}(x)$  in the second line of the equation to the weight definition  $w_n = 2c_n$  on the left. Another orange arrow points from the term  $-c_n^{\top} c_n$  to the bias definition  $b_n = -c_n^{\top} c_n$  below it. A third orange arrow points from the word "constant" to the text "just ignore\*" on the right.

$w_n = 2c_n$

$b_n = -c_n^{\top} c_n$

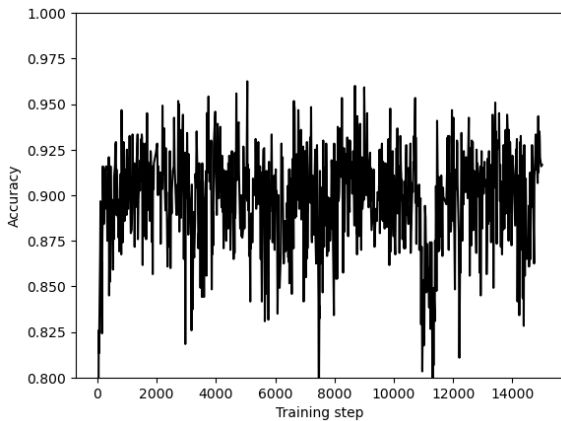
just ignore\*

A Dense output layer!!!

\* *Since it is a class-independent scalar, it leaves the output probabilities unchanged.*

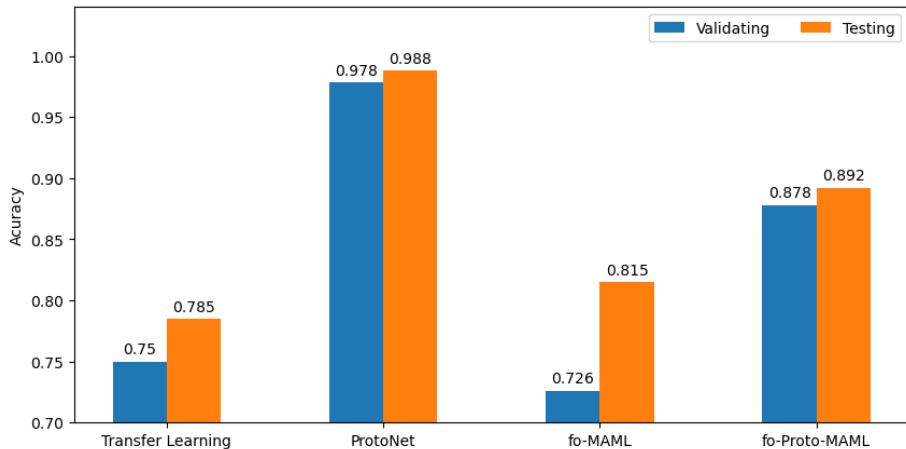
Triantafillou, Eleni, et al. "Meta-dataset: A dataset of datasets for learning to learn from few examples." arXiv preprint arXiv:1903.03096 (2019).

## fo-Proto-MAML result on Omniglot



Accuracy on query set during validating

## Experimental results



Comparison of accuracy on query set over approaches during validating/testing time

# Acknowledgements

This work was mostly derived from (1) the CS330 coursework at Stanford University, and (2) the UvA Deep Learning tutorial 16.

# After credits

## **Codework repositories and their associated experiments:**

- ▶ Three meta-learning algorithms on Omniglot:  
<https://github.com/nthehai01/meta-learning-methods>,
- ▶ Transfer learning on Omniglot: <https://github.com/nthehai01/non-episodic-approach>.