CVPR 2020 Tutorial:

Towards Annotation-Efficient Learning
Few-Shot Learning Methods

Spyros Gidaris

valeo.ai

CVPRVIRTUAL

https://annotation-efficient-learning.github.io/

Agenda

- Introduction
- Main types of few-shot algorithms
- Few-shot learning without forgetting

Agenda

- Introduction
  - **Few-shot learning**
  - Meta-learning paradigm
  - How to evaluate
- Main types of few-shot learning
- Few-shot learning without forgetting

Few-shot learning

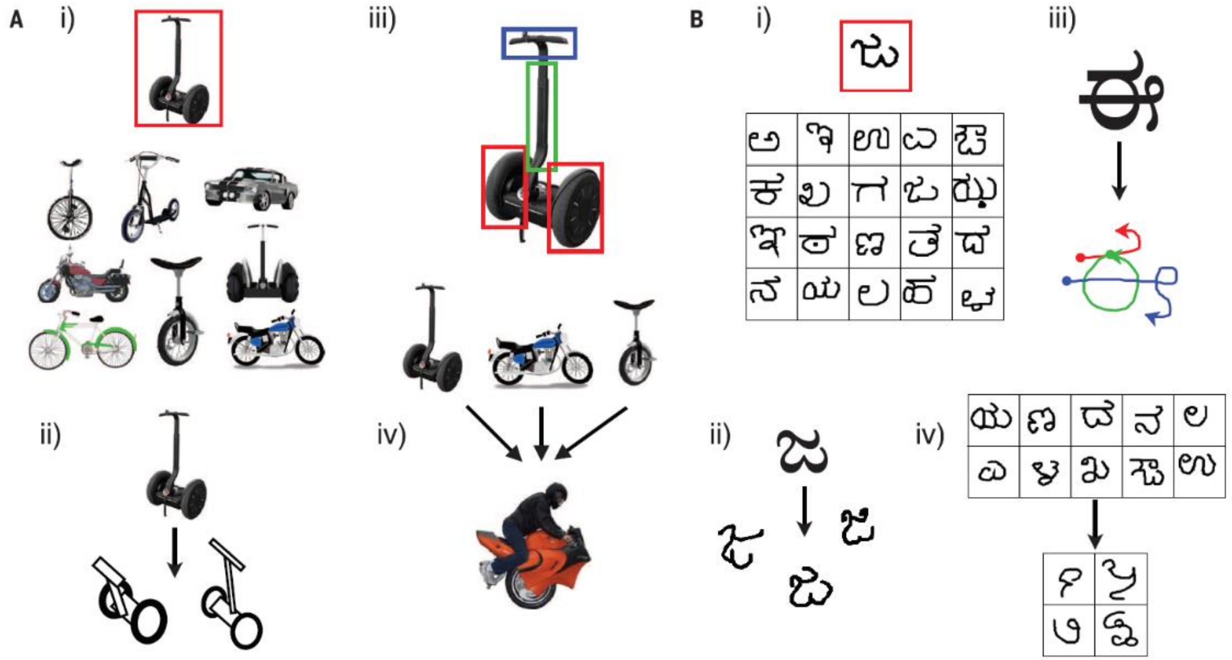
- Have you seen before an **okapi**?
- Can you learn to recognize it from only this image?

Few-shot learning

- **Humans:** able to learn new concepts using few training examples
- **Goal of few-shot learning:** mimic this ability with machine learning methods

Few-shot before the deep learning "revolution"

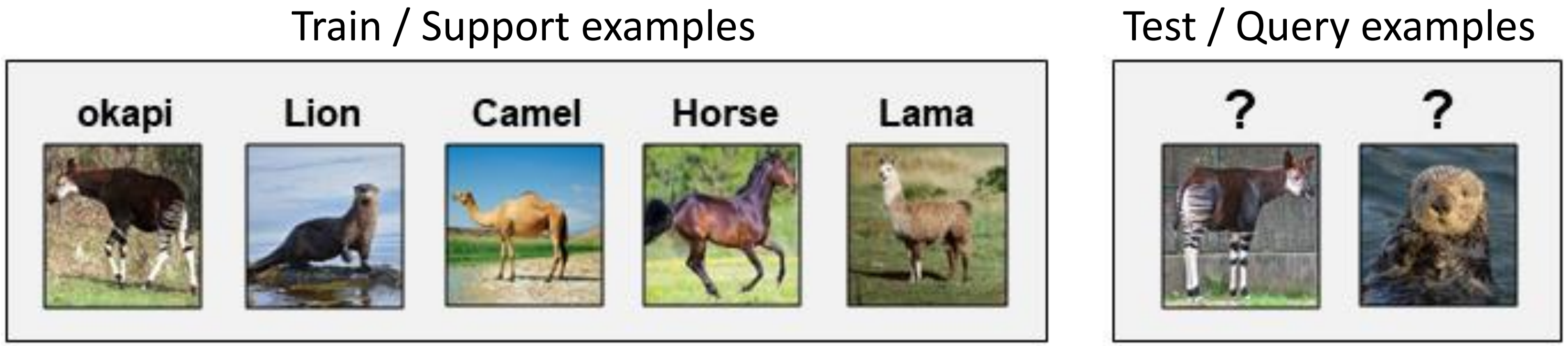
- "One-shot learning of simple visual concepts", Lake et al. 11
- "One-Shot Learning with a Hierarchical Nonparametric Bayesian Model", Salakhutdinov et al. 12
- "A Bayesian Approach to Unsupervised One-Shot Learning of Object Categories", Fei Fei et al. 13
- "Human-level concept learning through probabilistic program induction", Lake et al. 15



Here we will focus on deep learning based methods

Formally: Learn **N**-way **K**-shot classification tasks

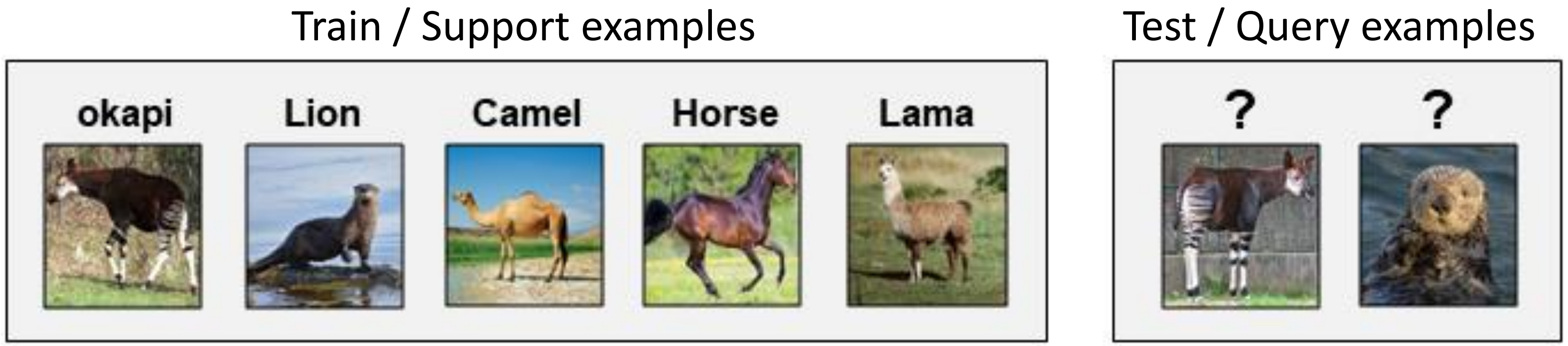
- **N =** number of classes
- **K =** training examples per class, **as small as 1 or 5!**



Example: 5-way **1**-shot classification task

Formally: Learn **N**-way **K**-shot classification tasks

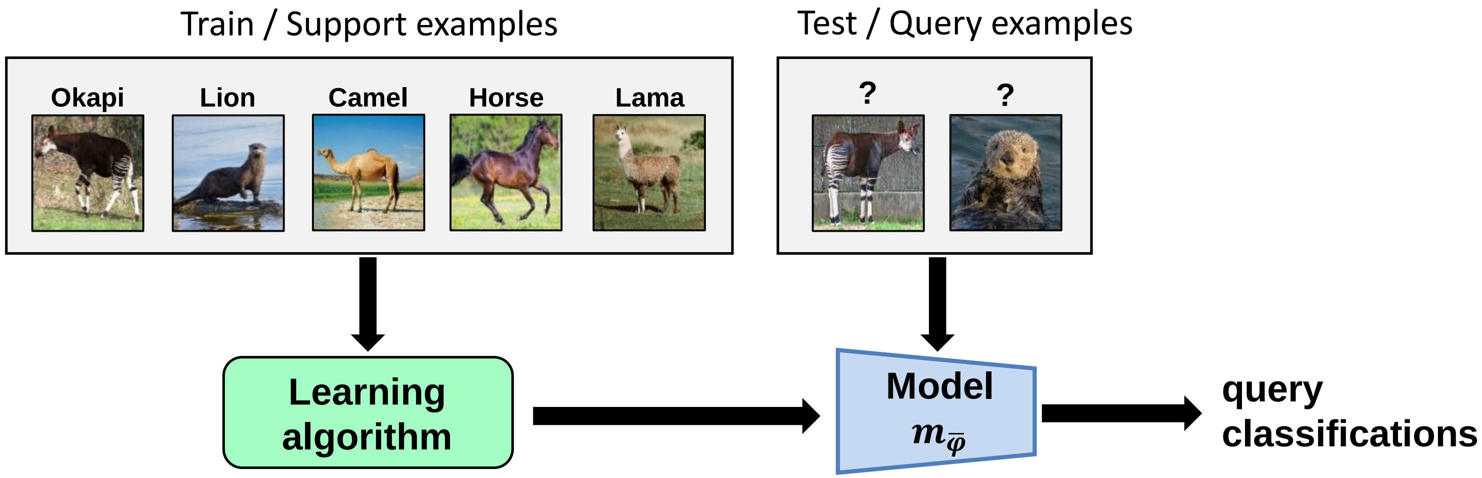
- **N =** number of classes
- **K =** training examples per class, **as small as 1 or 5!**



Example: 5-way **1**-shot classification task

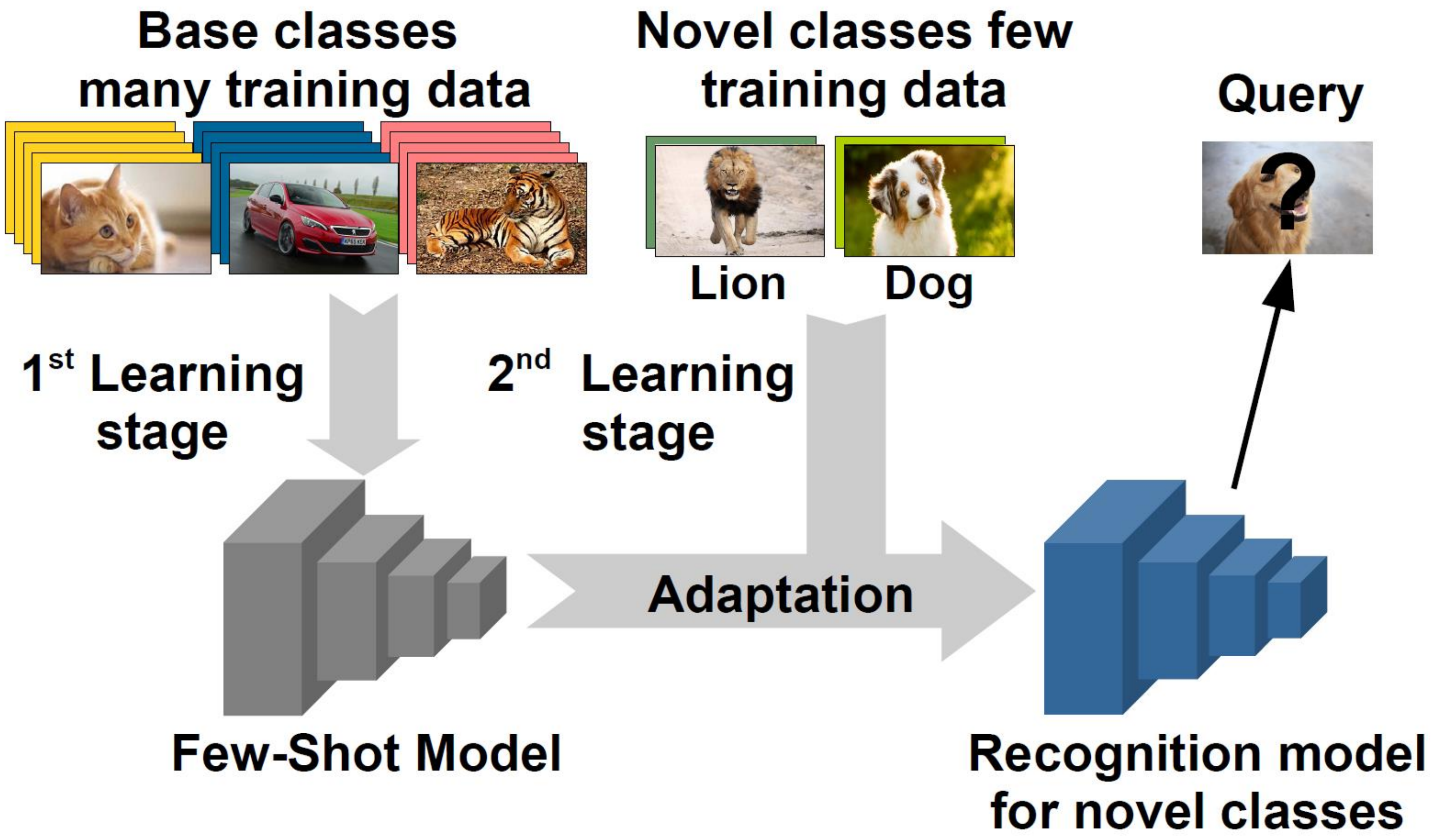
Question: is this possible with deep learning-based models?

Train directly a deep learning model



- Train from scratch a classification network
- Overfit to training data ➔ poor accuracy on test data ☹

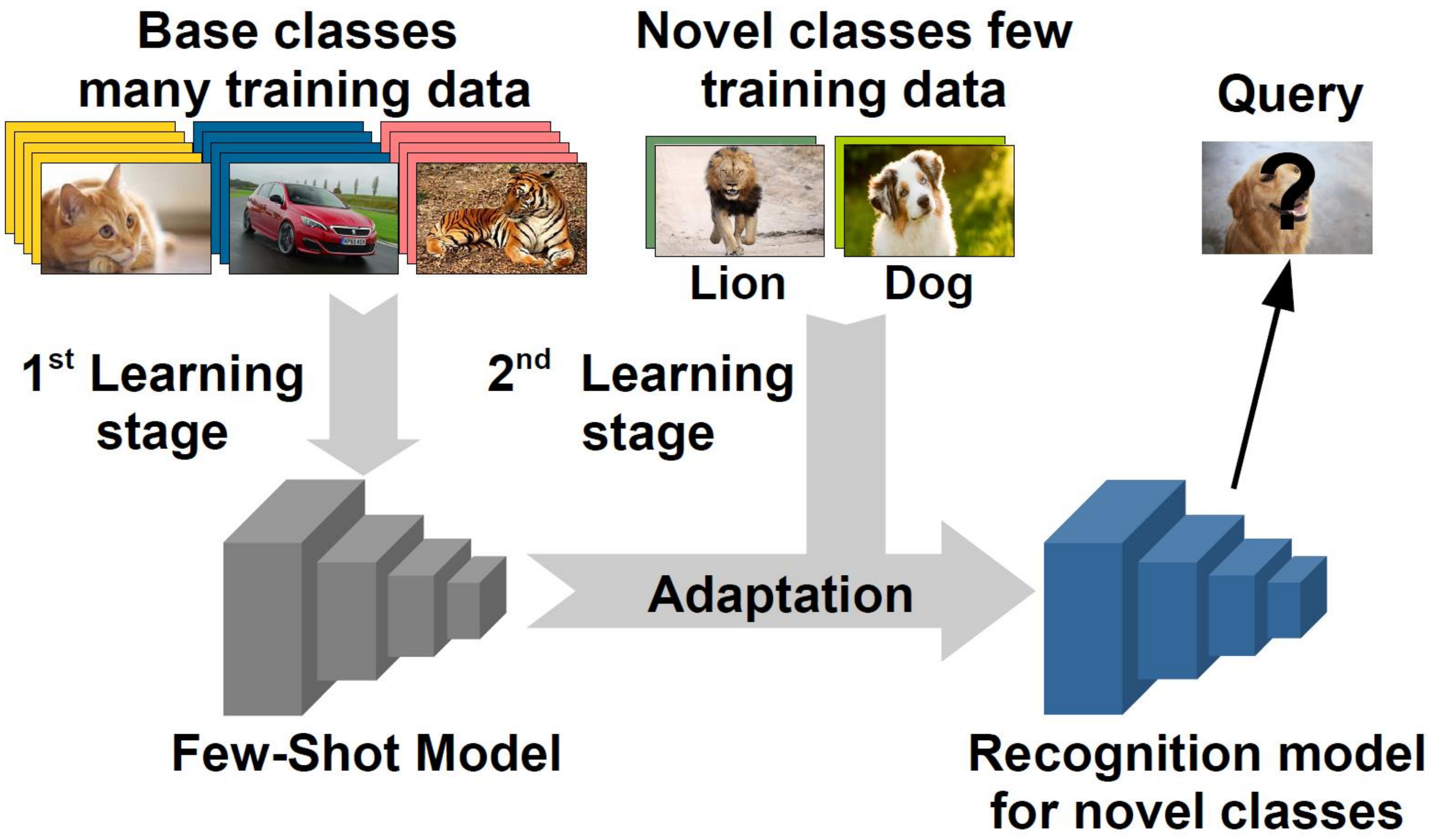
Overcome data scarcity with transfer learning



Overcome data scarcity with transfer learning

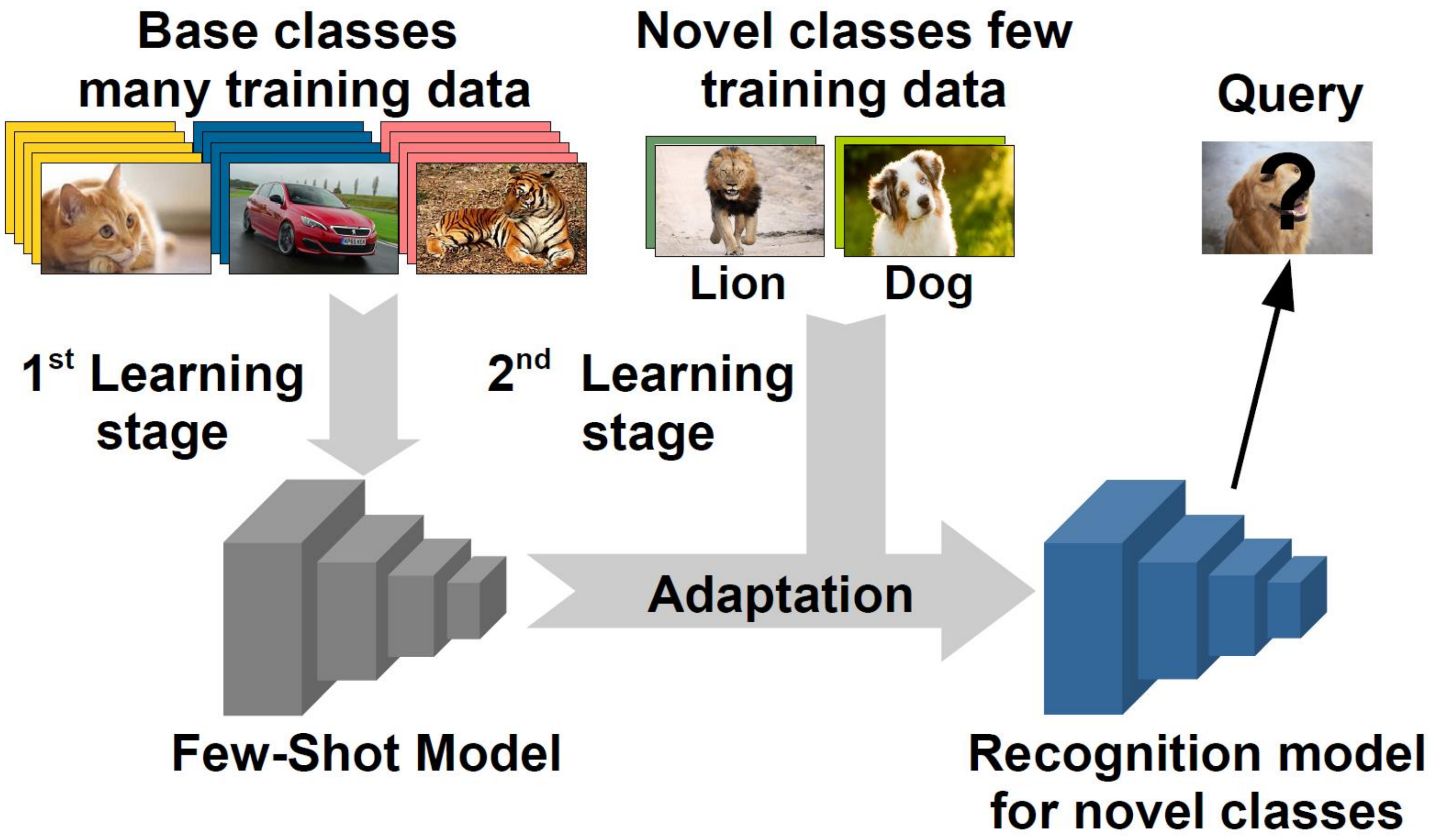
- Recipe followed by all few-shot learning methods

Overcome data scarcity with transfer learning



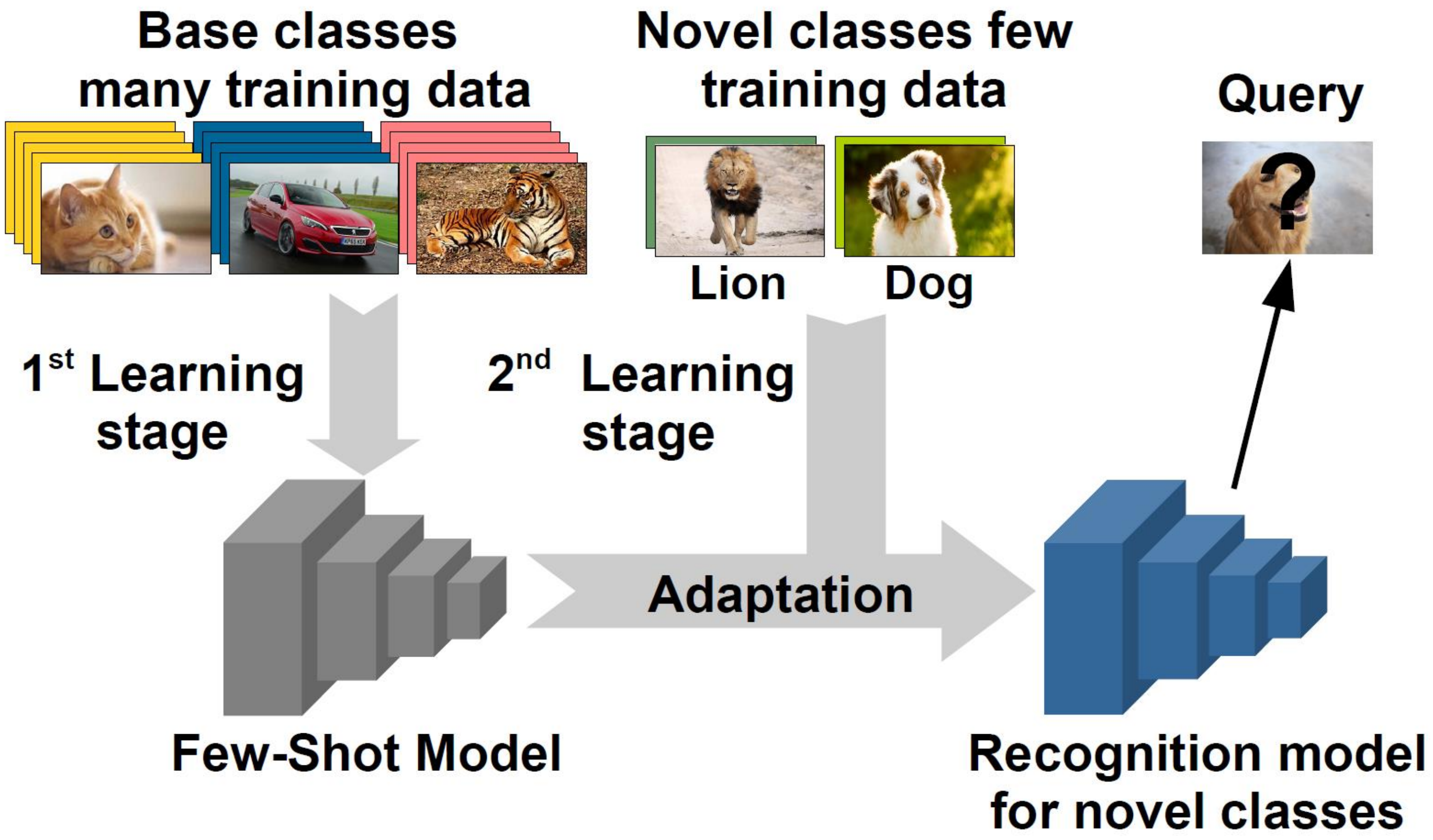
1. **Acquire knowledge:** train on other similar problems
2. **Transfer knowledge:** adapt to the problem of interest

Overcome data scarcity with transfer learning



1. **Acquire knowledge:** use many training data from some **base classes**
2. **Transfer knowledge:** adapt to **novel classes** with few training data

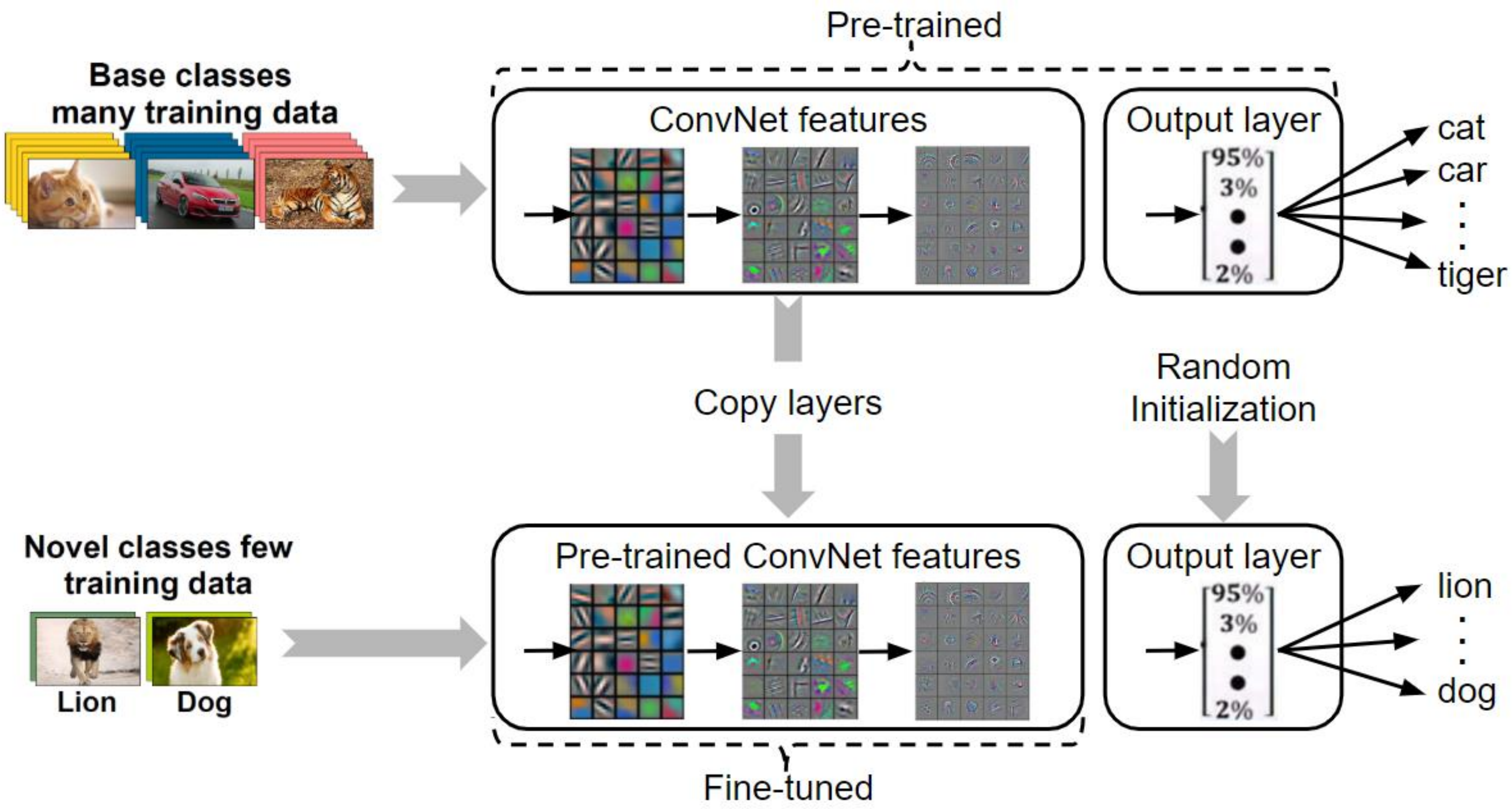
Overcome data scarcity with transfer learning



base classes == train classes
novel classes == test classes

no overlap between them

Common transfer learning example: Fine-tuning



1. **Acquire knowledge: pre-train** a network on the base class data
2. **Transfer knowledge: fine-tune** the network on novel class data

Few-shot learning methods

Fine-tuning: risk of overfitting in case of extremely limited data (few-shot)

Goal of few-shot learning: devise transfer learning algorithms that would work well in the few-shot scenario, e.g., metric learning, meta-learning methods, …

Agenda

- Introduction
  - Few-shot learning problem
  - **Meta-learning paradigm**
  - How to evaluate
- Main types of few-shot learning algorithms
- Few-shot learning without forgetting

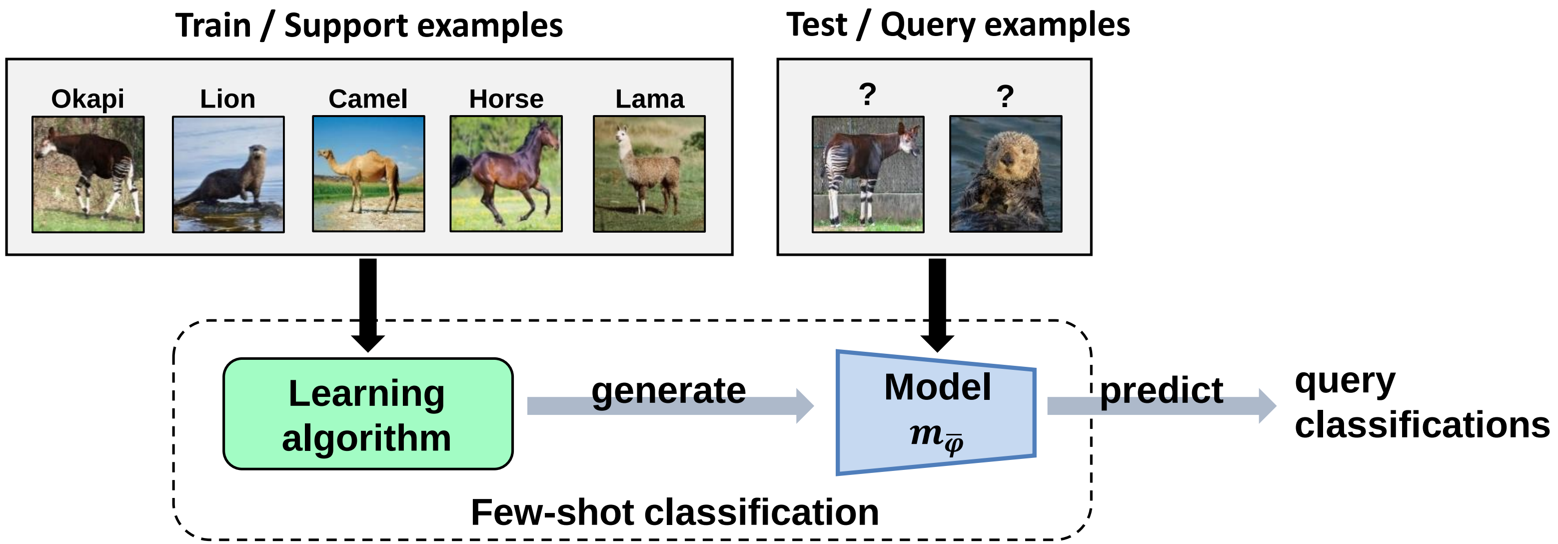
Few-shot meta-learning

Most (but not all) few-shot methods use **meta-learning (learn-to-learn paradigm)**

- “Evolutionary principles in self-referential learning, or on learning how to learn”, Schmidhuber 1987
- “Meta-neural networks that learn by learning”, Naik et al. 1992
- “Lifelong learning algorithms”, Thrun 1998
- “Learning to learn by gradient descent by gradient descent”, Andrychowicz et al. 16
- …

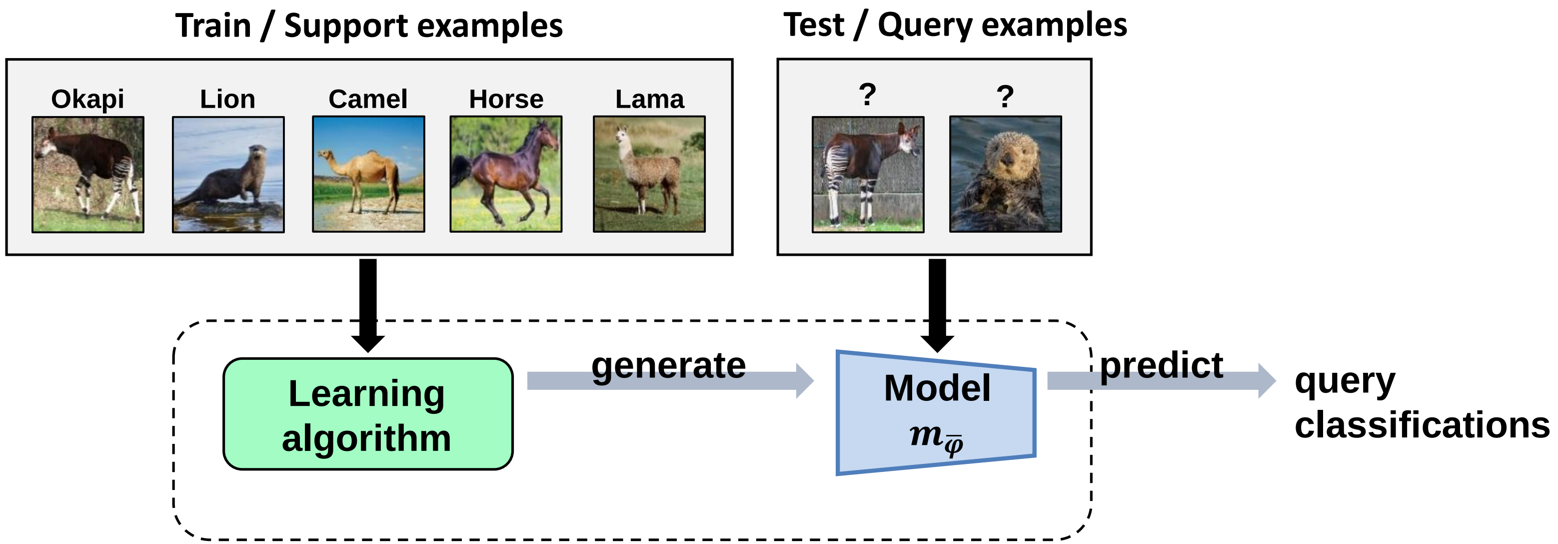
What is few-shot meta-learning?

Few-shot classification



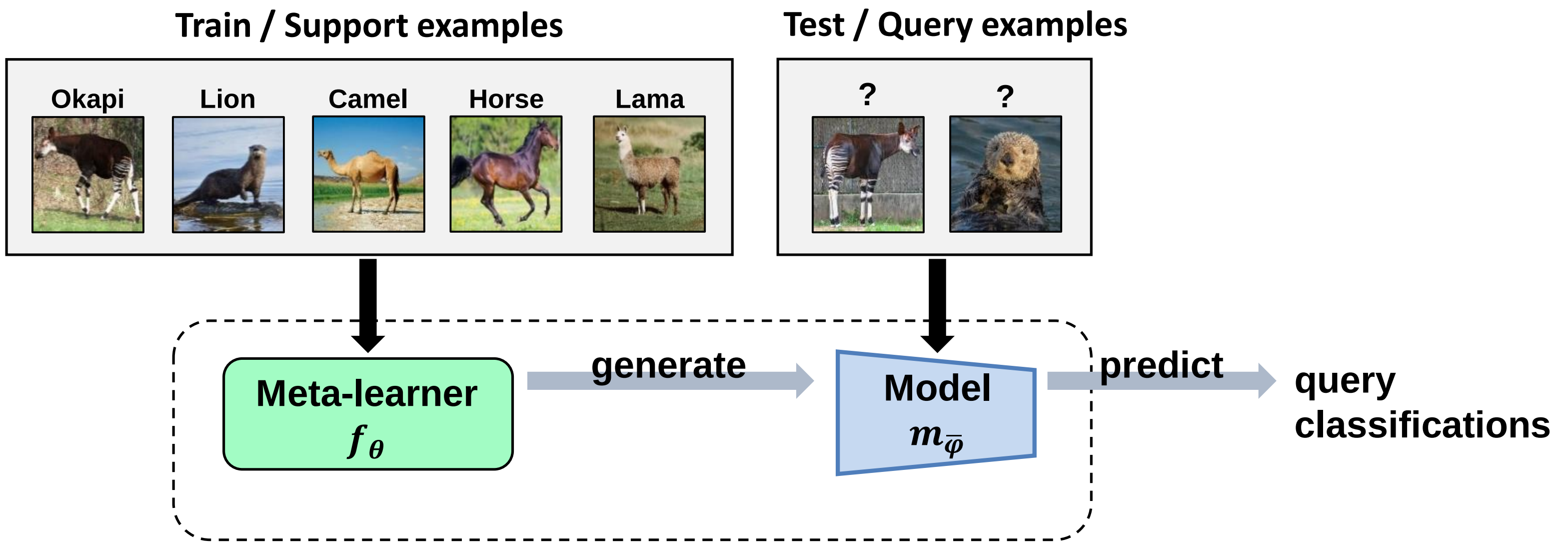
- **input:** labeled support data, unlabeled query data
- **intermediate output:** model for classifying the query images
- **output:** predicted query labels

Few-shot classification with meta-learning



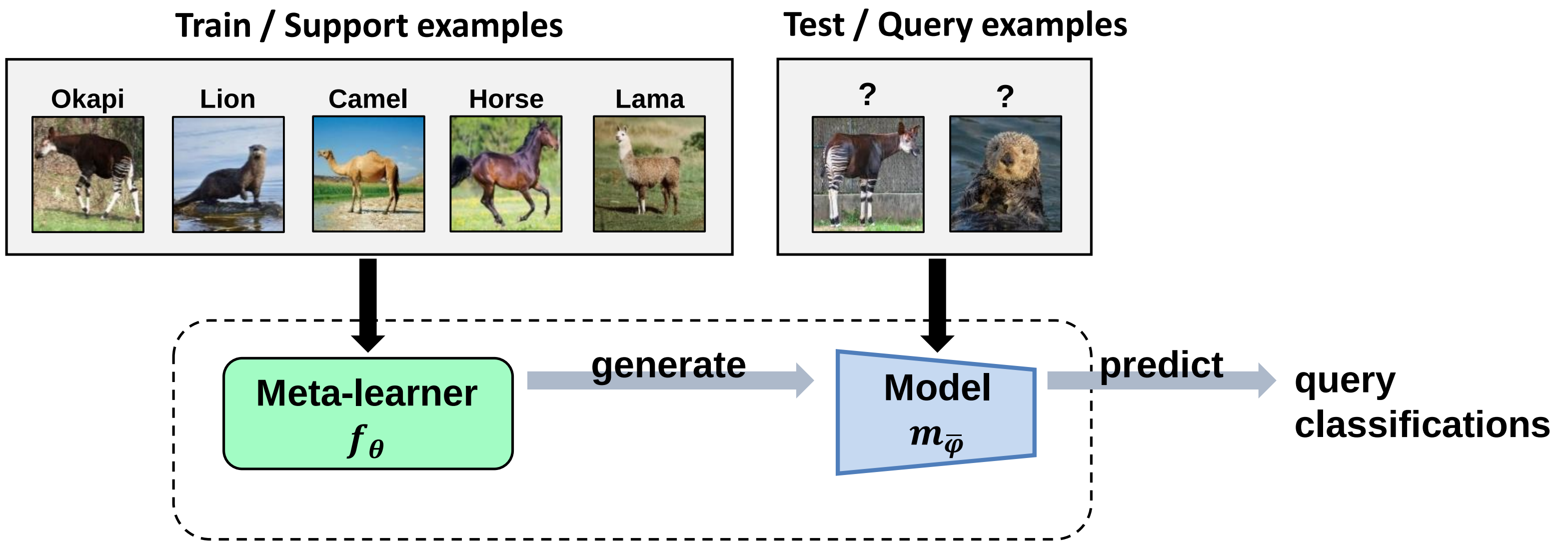
- **Train the learning algorithm** (instead of the classification model)
  - Implement it with a **meta-learner f_θ**
  - Optimize f_θ on learning few-shot classification tasks **(learn-to-learn)**

Few-shot classification with meta-learning



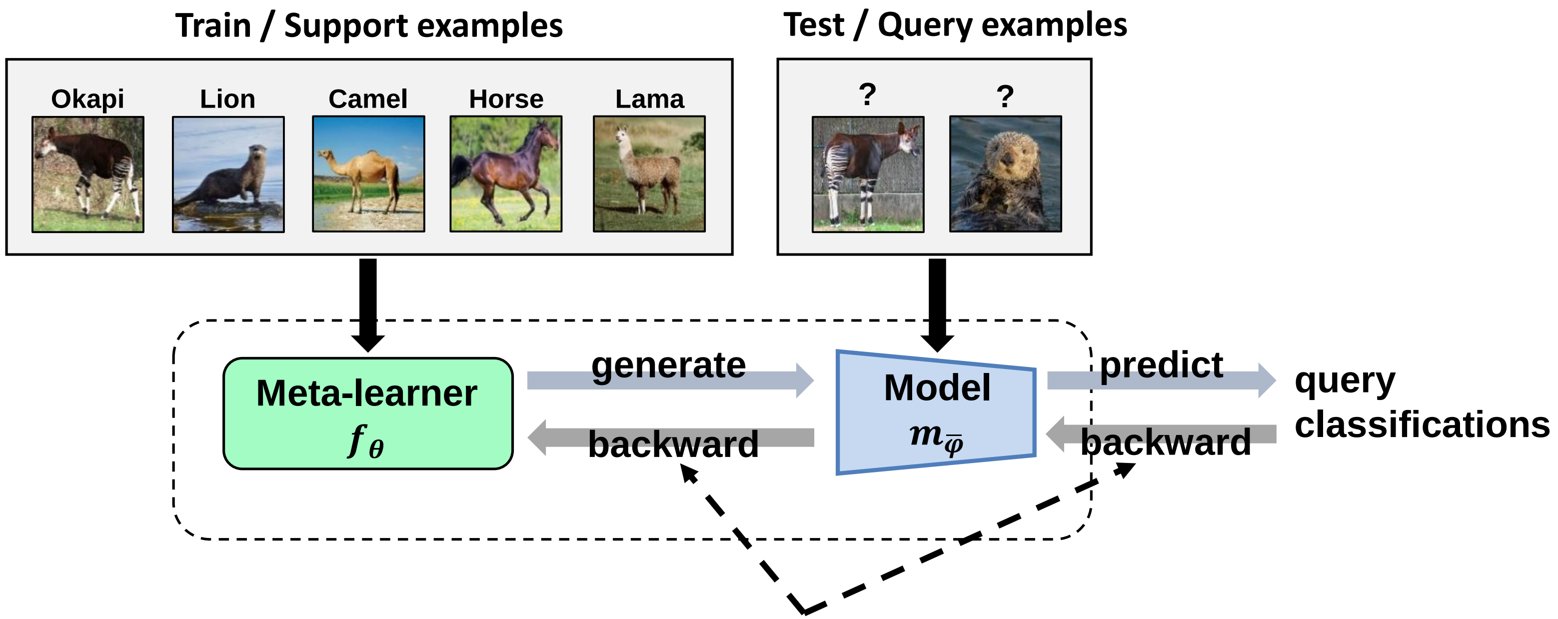
- **Train the learning algorithm** (instead of the classification model)
  - Implement it with a **meta-learner f_θ (somehow)**
  - Optimize f_θ on solving few-shot classification tasks **(learn-to-learn)**

Few-shot classification with meta-learning



- **Train the learning algorithm** (instead of the classification model)
  - Implement it with a **meta-learner f_θ**
  - Optimize f_θ on solving few-shot classification tasks **(learn-to-learn)**

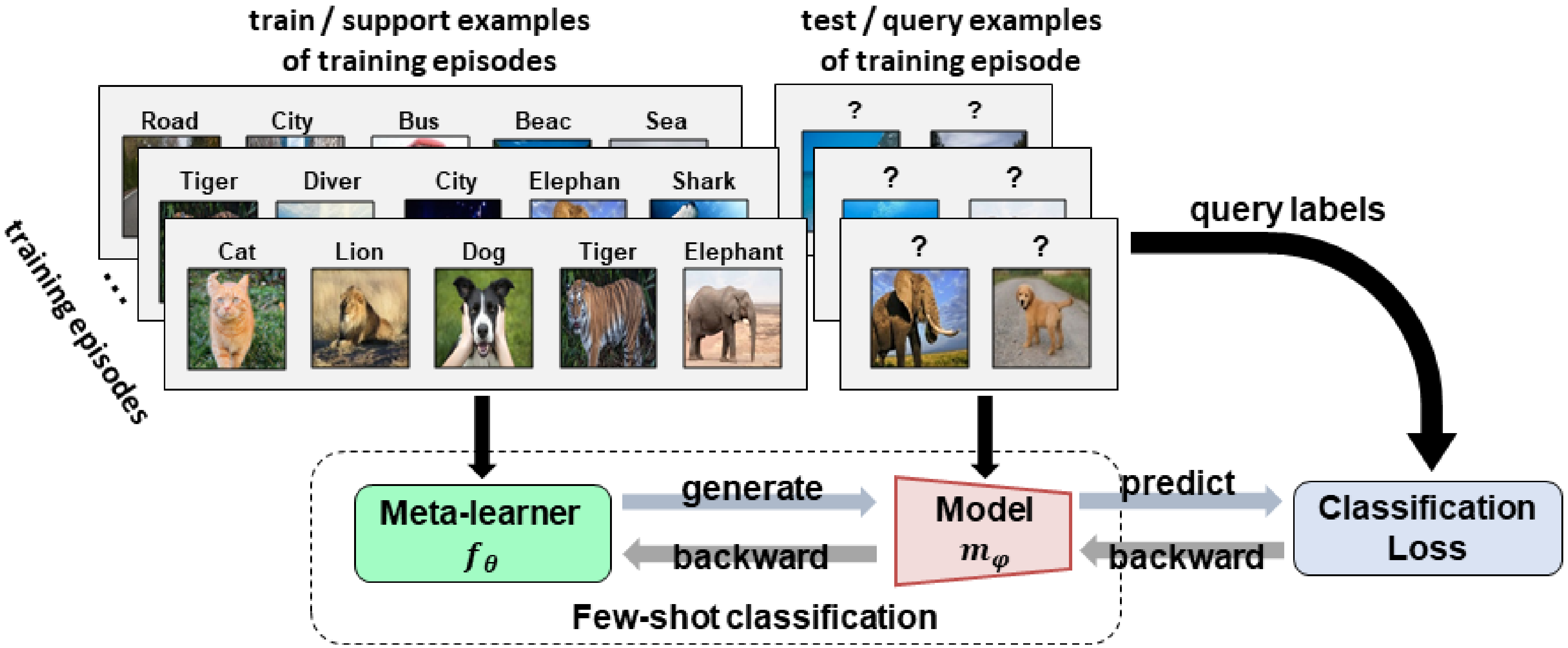
Few-shot classification with meta-learning



must back-propagate the entire few-shot learning process

- **Train the learning algorithm** (instead of the classification model)
  - Implement it with a **meta-learner** $\boldsymbol{f_\theta}$
  - Optimize $\boldsymbol{f_\theta}$ on solving few-shot classification tasks **(learn-to-learn)**

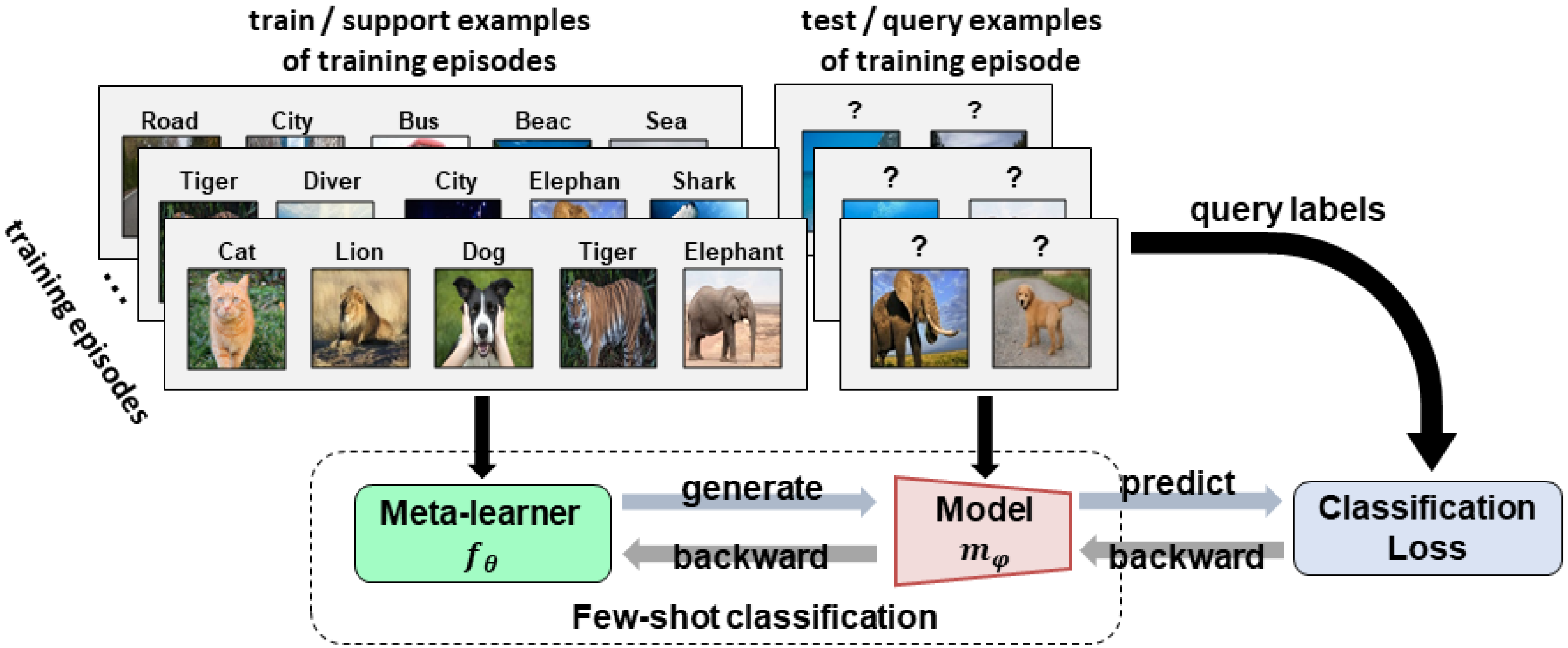
Meta-learning: training time (1st learning stage)



How to train the meta-learner?

- Train it on the same conditions it will be used in 2^{nd} learning stage (meta-test)

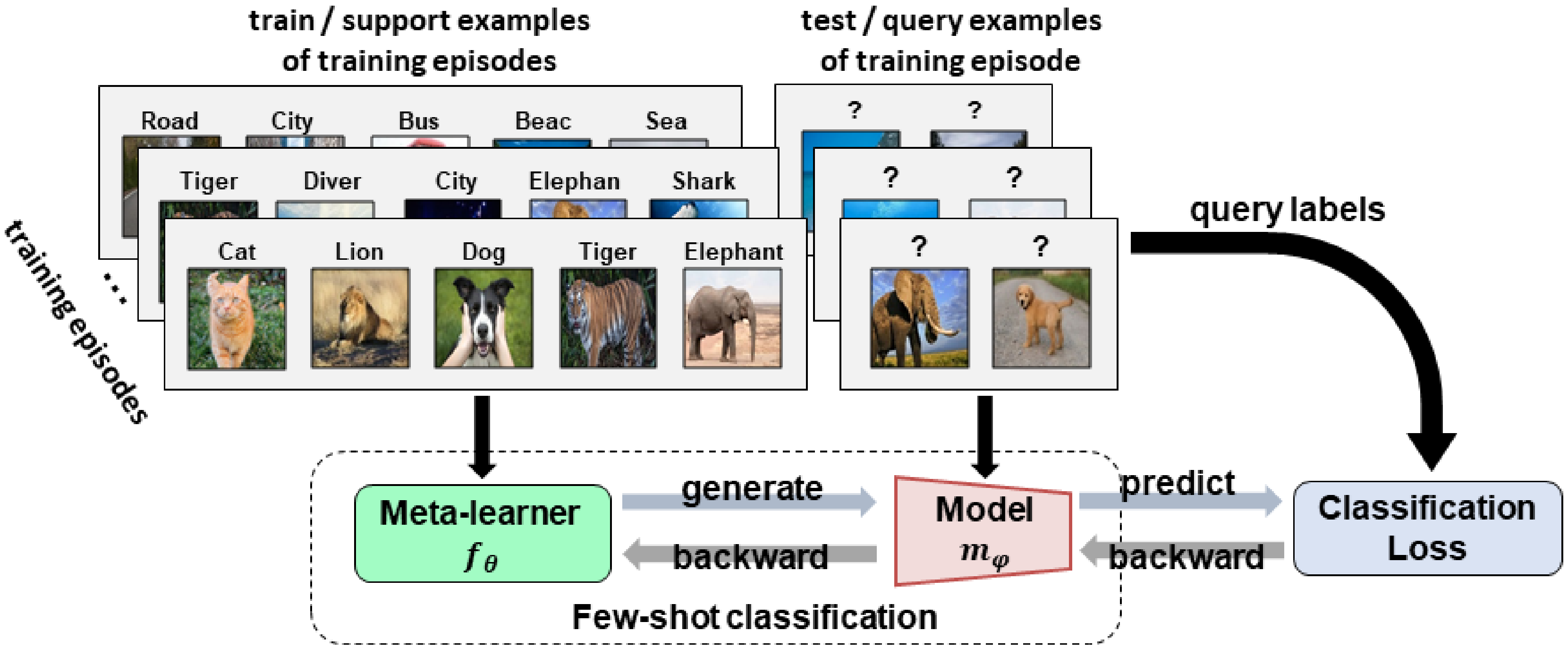
Meta-learning: training time (1st learning stage)



How to train the meta-learner?

- Train **meta-learner** f_θ on solving a distribution of few-shot tasks (aka **episodes**)

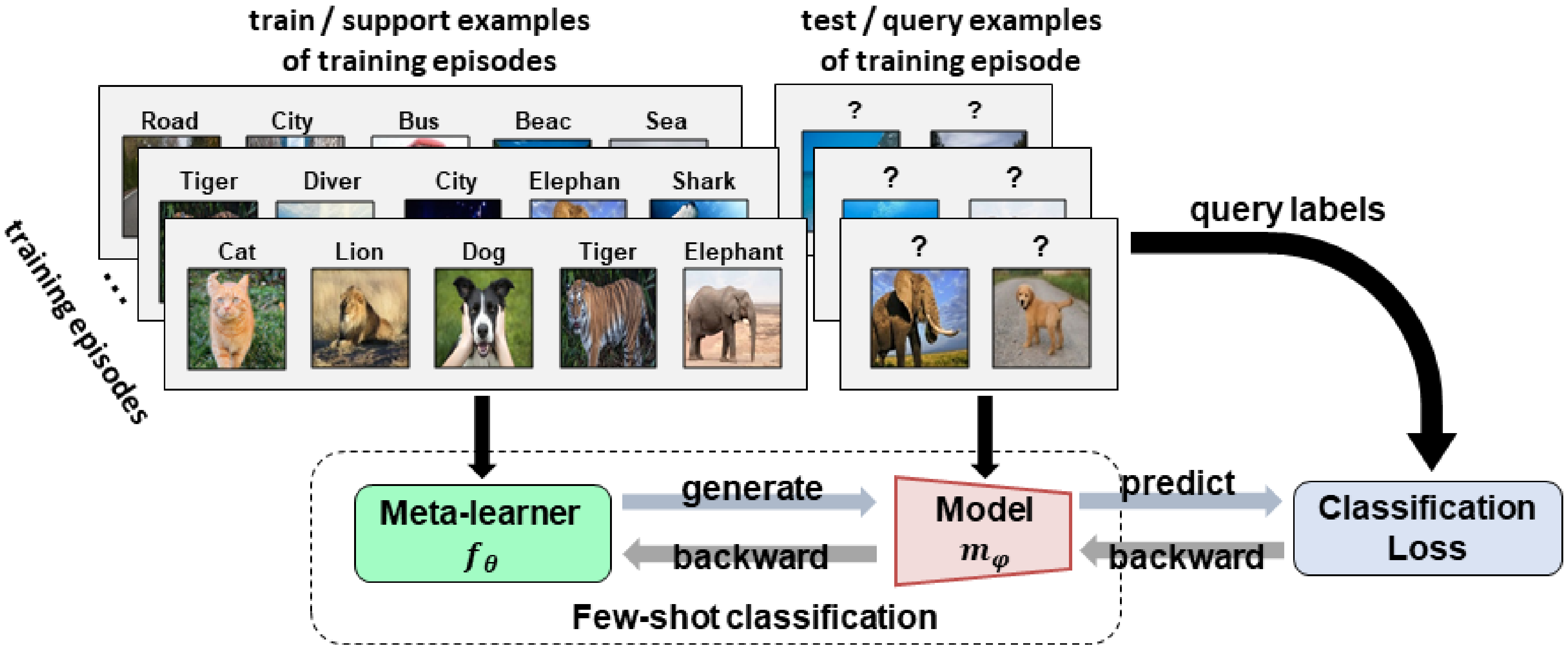
Meta-learning: training time (1st learning stage)



How to train the meta-learner?

- Train **meta-learner f_θ** on solving a distribution of few-shot tasks (aka **episodes**)
- Construct such **training episodes** using the base class data

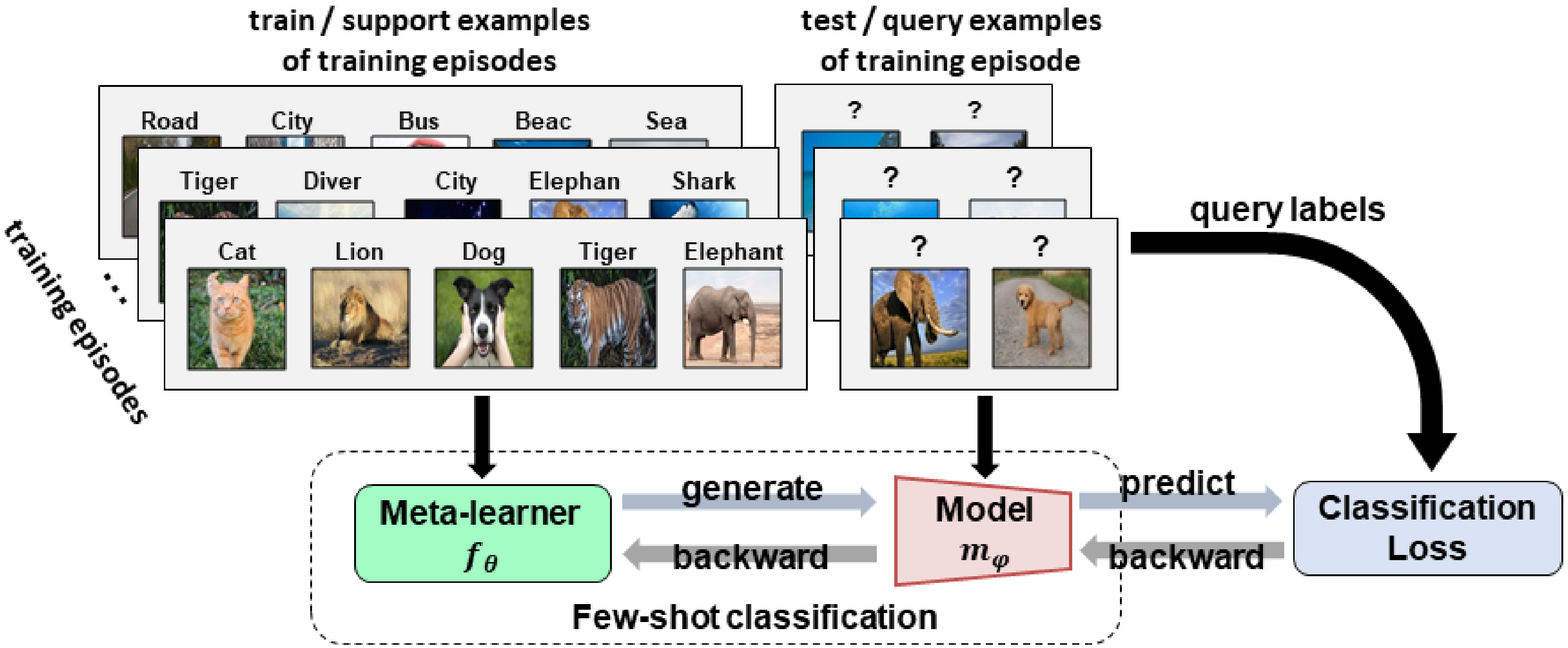
Meta-learning: training time (1st learning stage)



How to train the meta-learner?

- Train **meta-learner** f_θ on solving a distribution of few-shot tasks (aka **episodes**)
- Construct such **training episodes** using the base class data
  - by sampling **N** classes x (**K** support examples + **M** query examples)

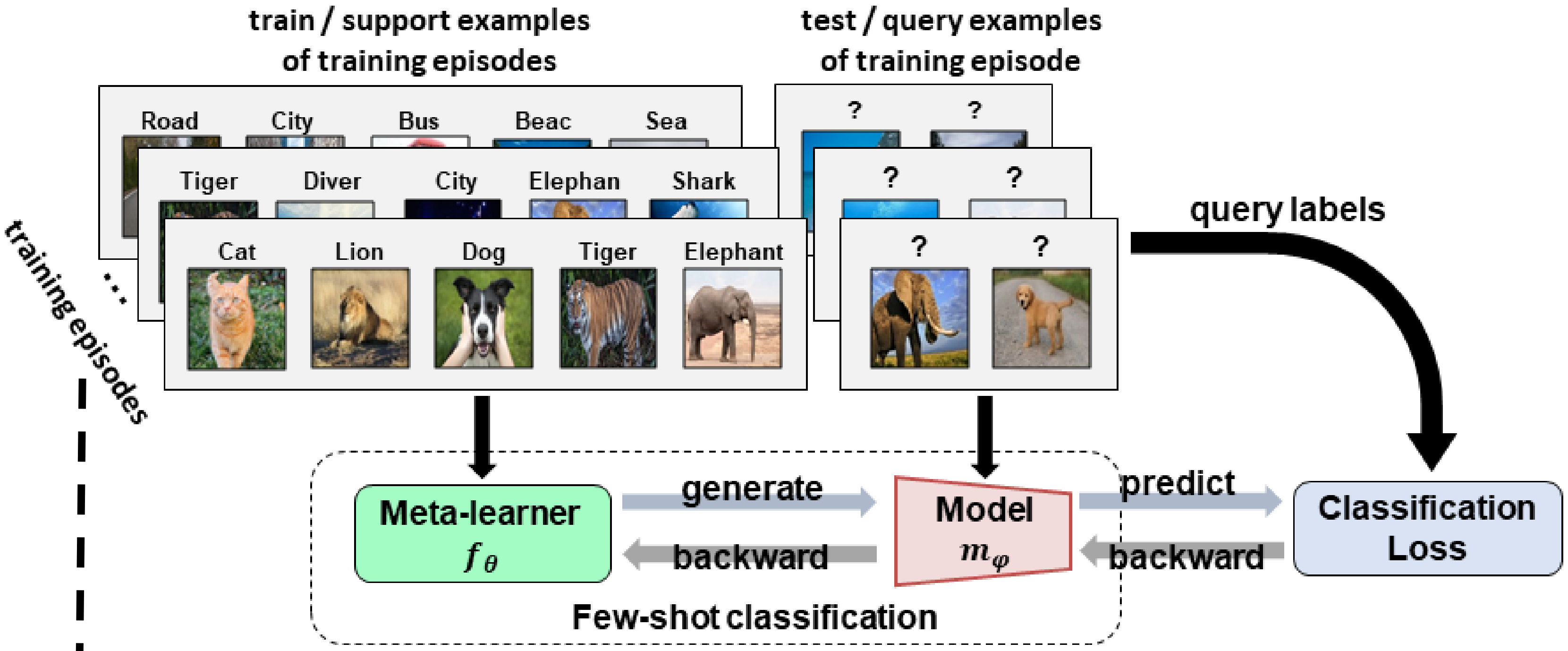
Meta-learning: training time (1st learning stage)



Objective:

$$\min_{\theta} \sum_{(S,Q)} L(f_\theta(S), Q)$$

Meta-learning: training time (1st learning stage)

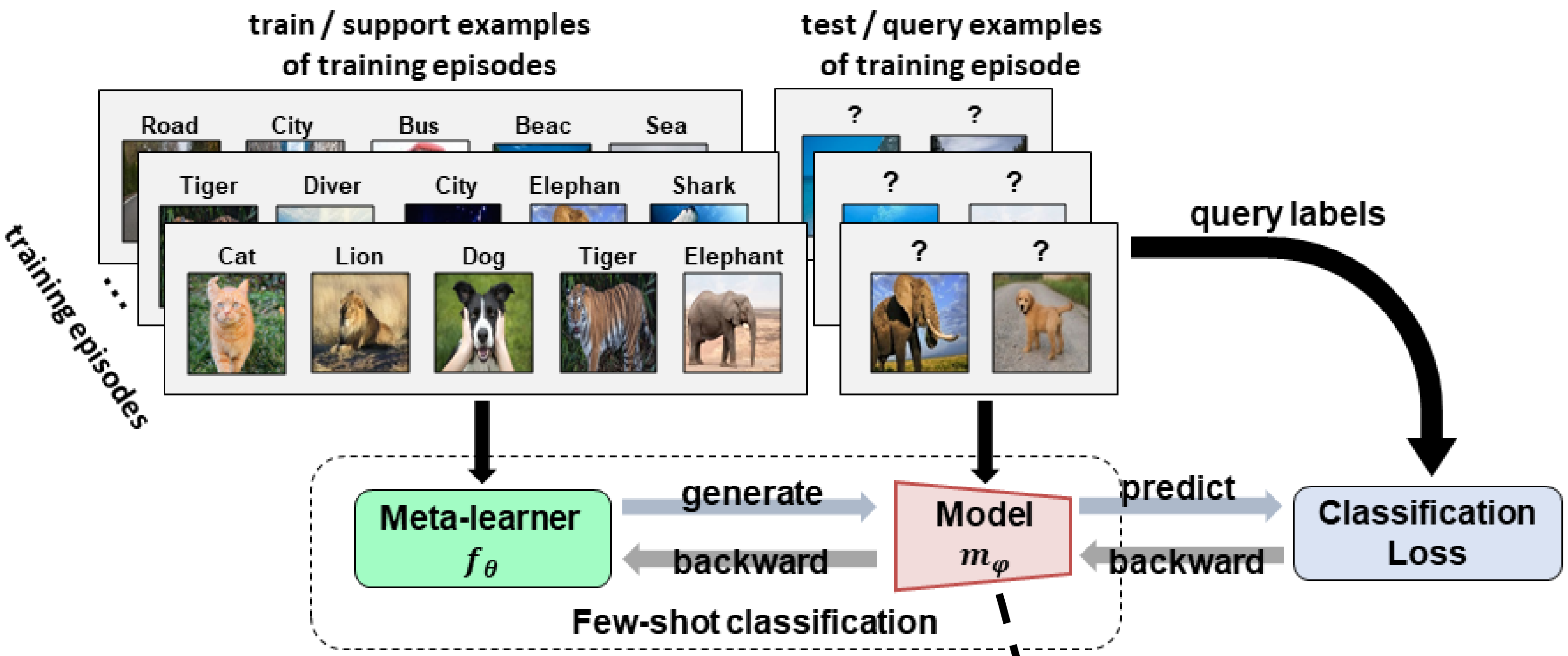


Objective:

$$\min_{\theta} \sum_{(S,Q)} L(f_\theta(S), Q)$$

Episode *(S,Q)*: support set $S = \{x_k^S, y_k^S\}_{k=1}^{N*K}$ and query set $Q = \{x_m^Q, y_m^Q\}_{m=1}^{N*M}$

Meta-learning: training time (1st learning stage)

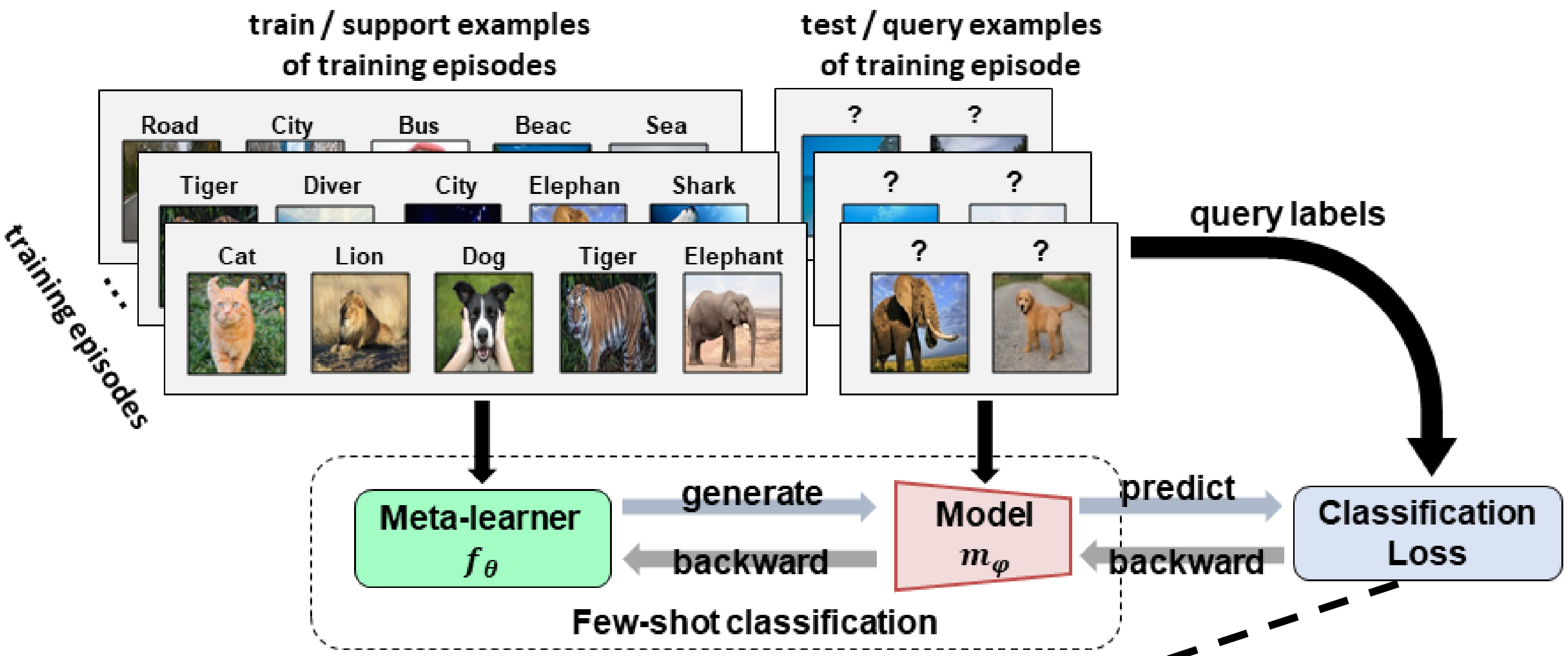


Objective:

$$\min_{\theta} \sum_{(S,Q)} L(f_{\theta}(S), Q)$$

Inner part: generate using the support set *S* the classification model $m_\varphi = f_\theta(S)$

Meta-learning: training time (1st learning stage)

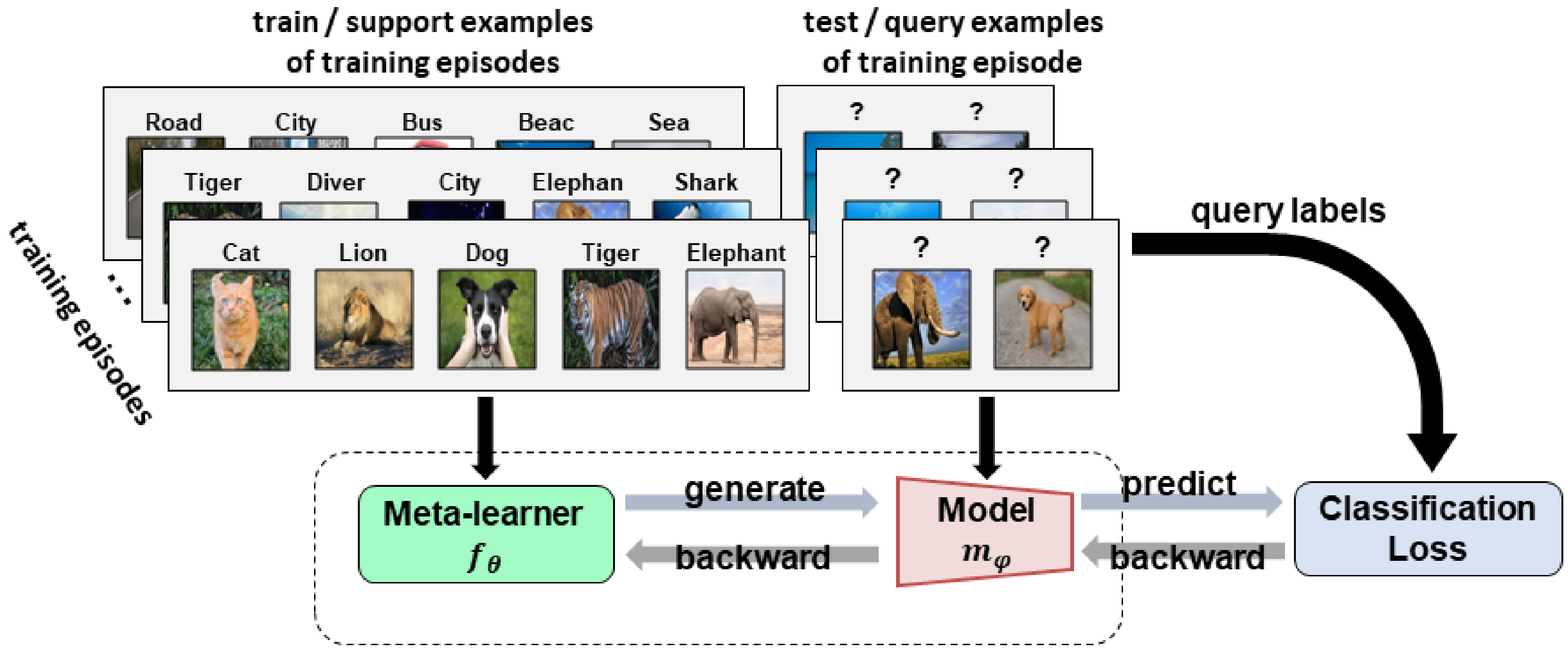


Objective:

$$\min_{\theta} \sum_{(S,Q)} L(f_\theta(S), Q)$$

Outer part: optimize θ w.r.t. the queries classification loss $L(f_\theta(S), Q) = L(m_\varphi, Q)$

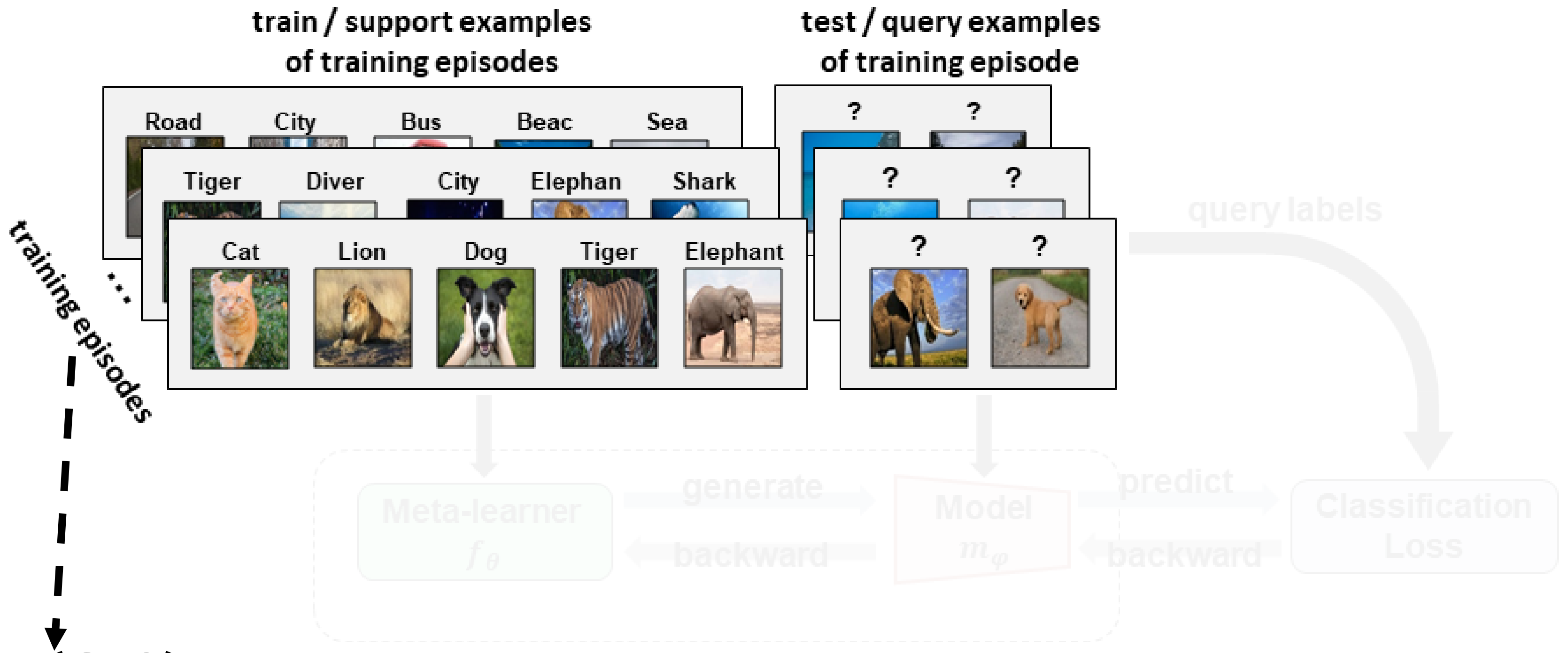
Meta-learning: training time (1st learning stage)



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each x_m^Q in Q
4. Optimize θ w.r.t. the queries classification loss $L(f_\theta(S), Q)$

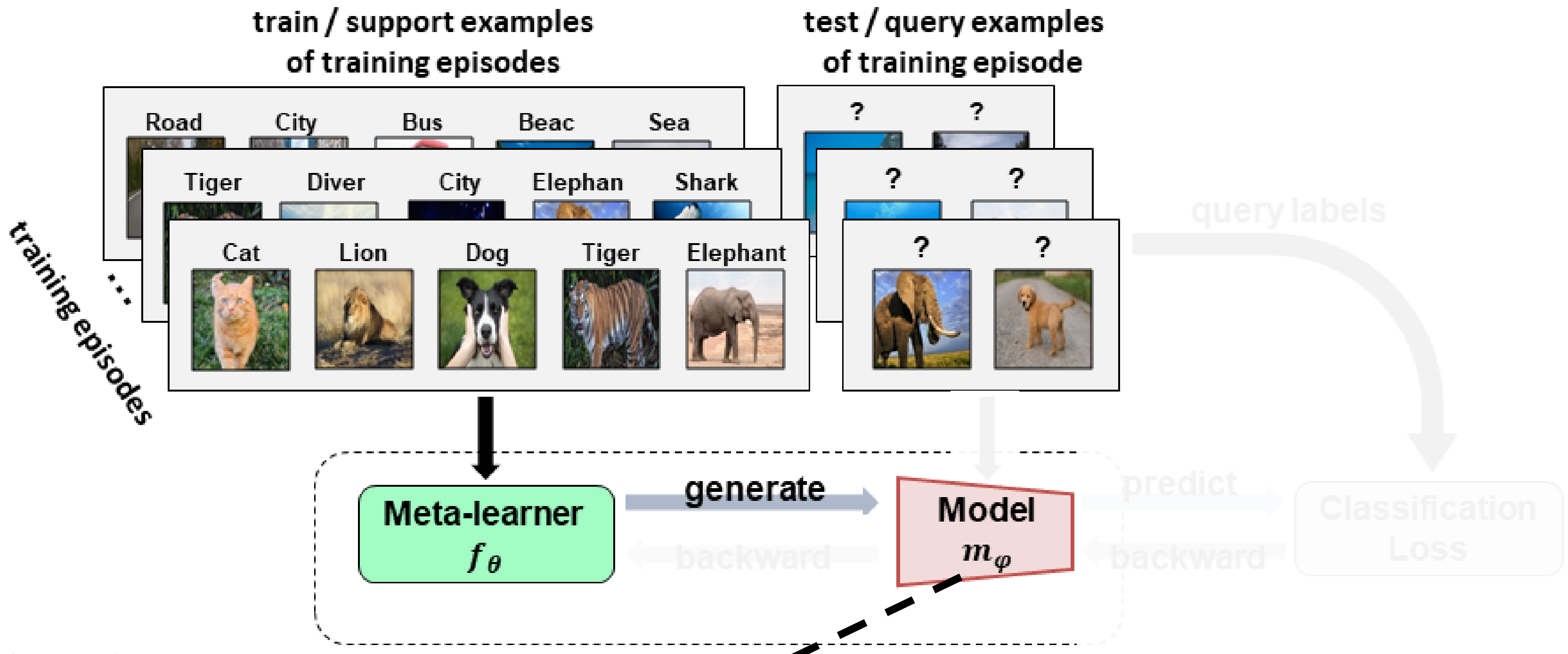
Meta-learning: training time (1st learning stage)



Meta-training routine:

1. **Sample training episode (S, Q)**
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each x_m^Q in Q
4. Optimize θ w.r.t. the queries classification loss $L(f_\theta(S), Q)$

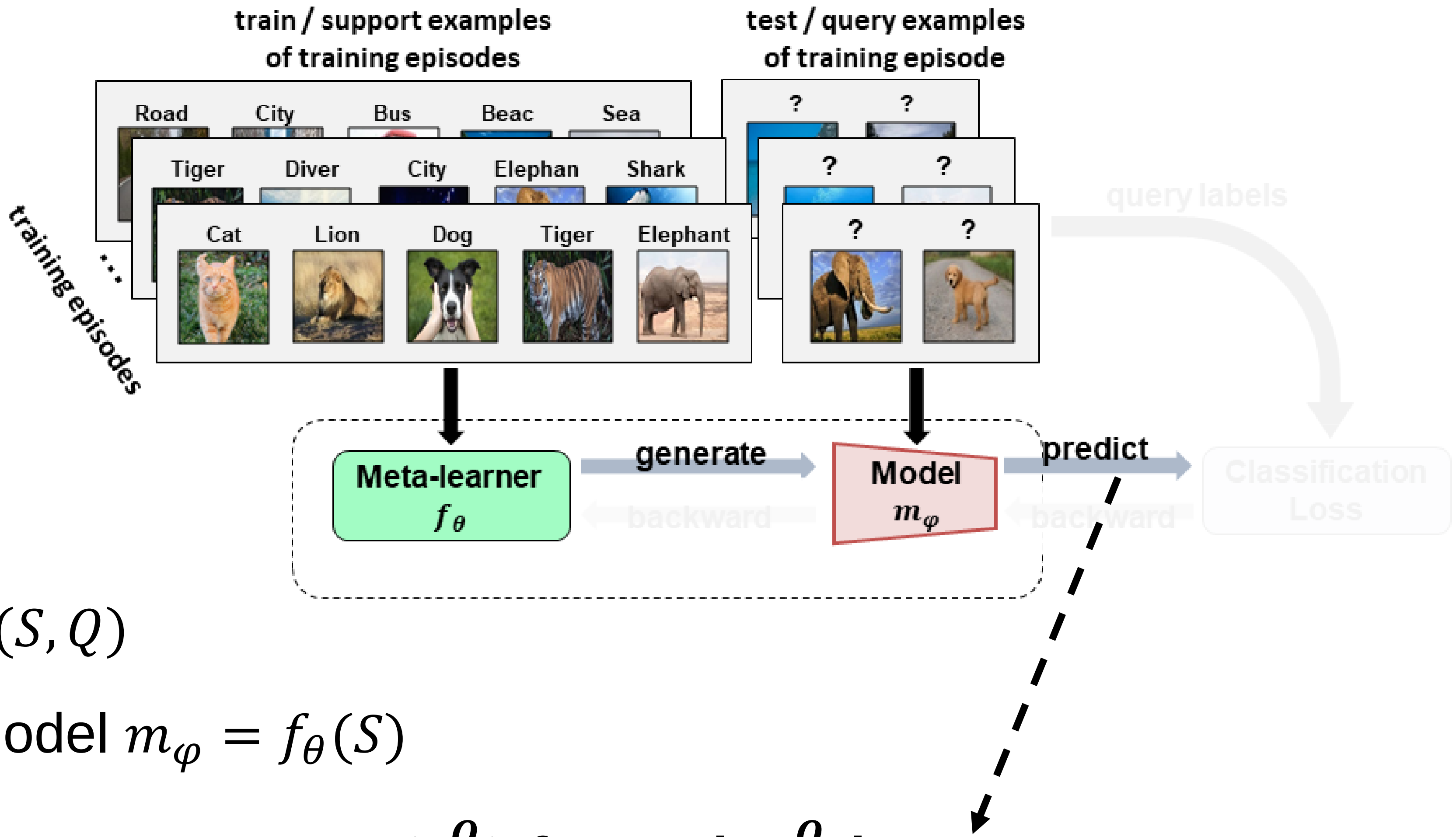
Meta-learning: training time (1st learning stage)



Meta-training routine:

1. Sample training episode (S, Q)
2. **Generate classification model** $\boldsymbol{m_\varphi = f_\theta(S)}$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each x_m^Q in Q
4. Optimize θ w.r.t. the queries classification loss $L(f_\theta(S), Q)$

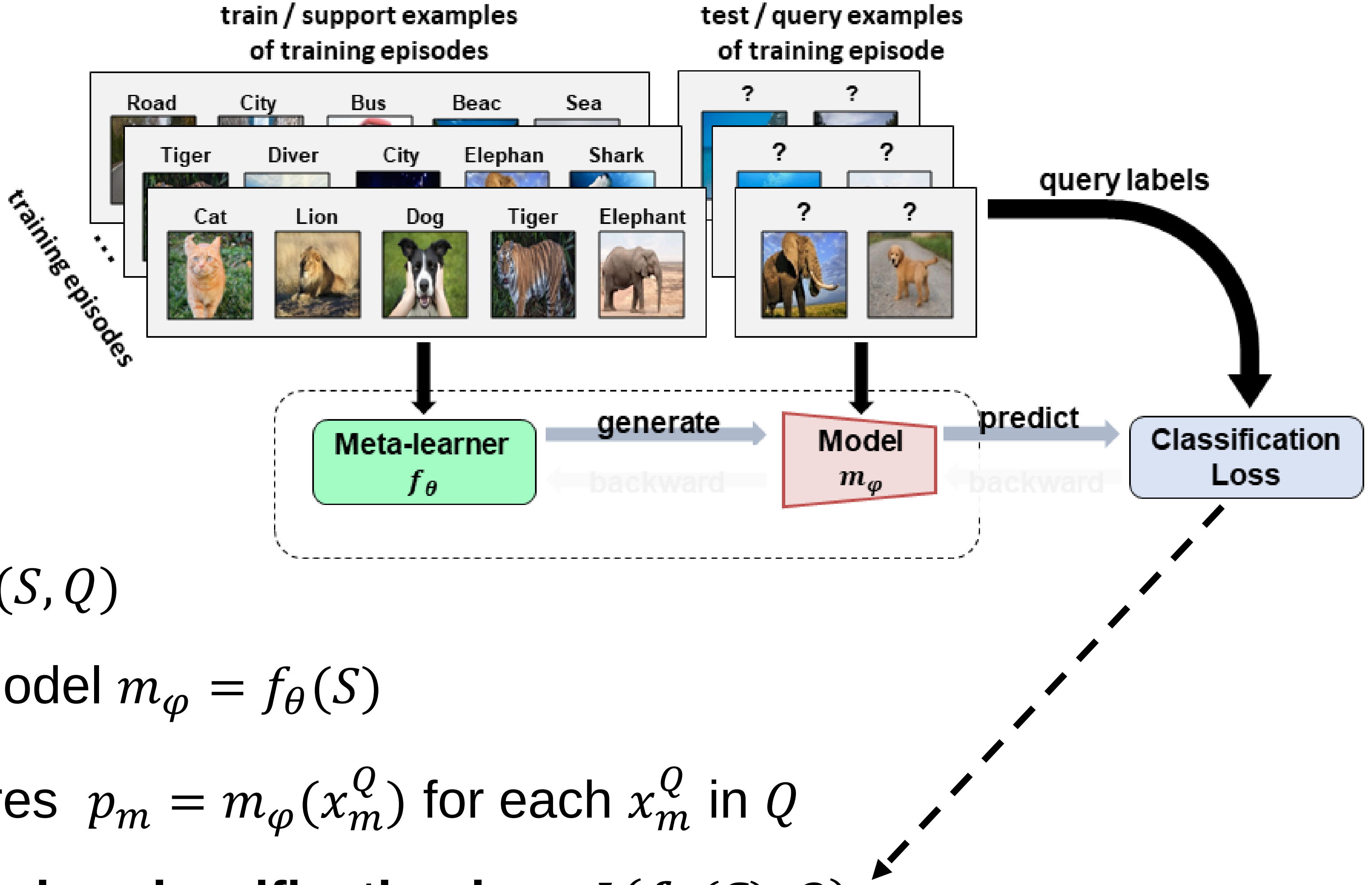
Meta-learning: training time (1^{st} learning stage)



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. **Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each x_m^Q in Q**
4. Optimize θ w.r.t. the queries classification loss $L(f_\theta(S), Q)$

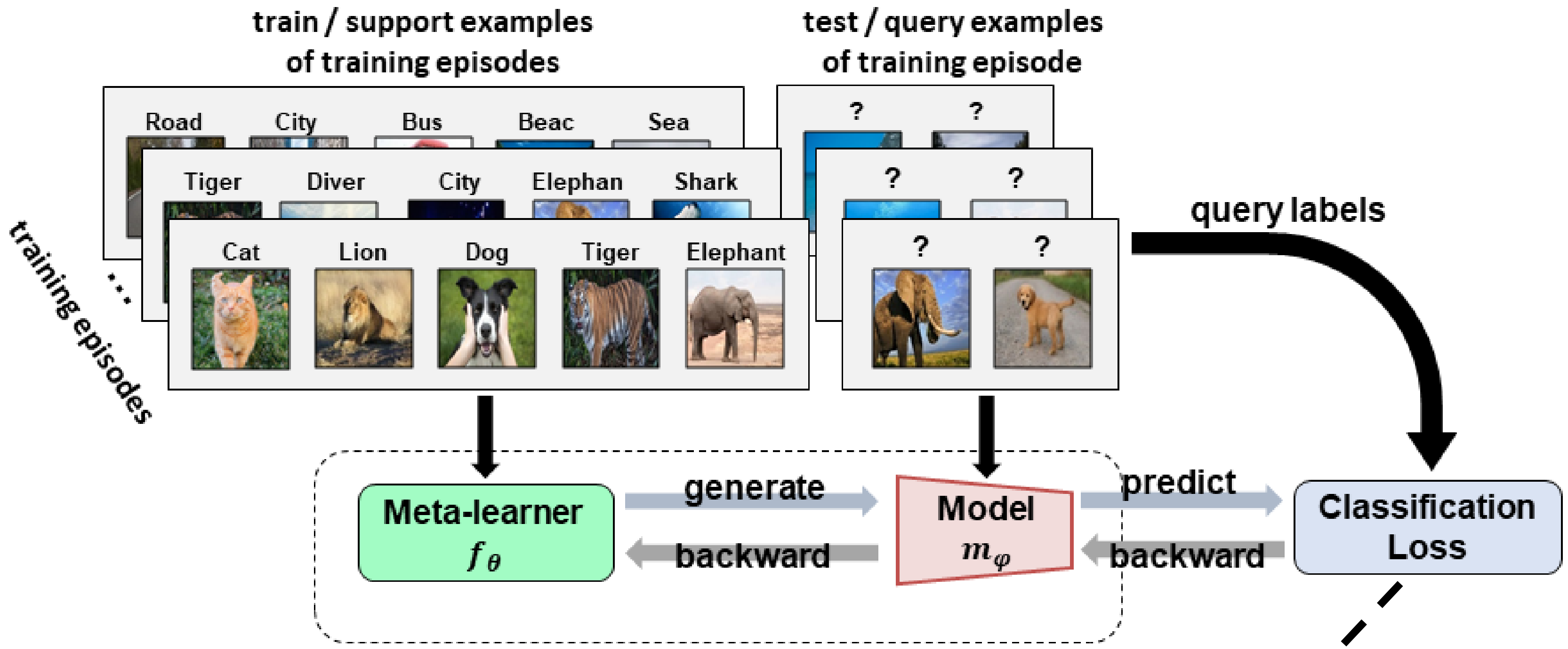
Meta-learning: training time (1st learning stage)



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each x_m^Q in Q
4. **Optimize $\boldsymbol{\theta}$ w.r.t. the queries classification loss $\boldsymbol{L(f_\theta(S), Q)}$**
   - **e.g., cross entropy loss $\sum_m -\boldsymbol{log(p_m[y_m^Q])}$**

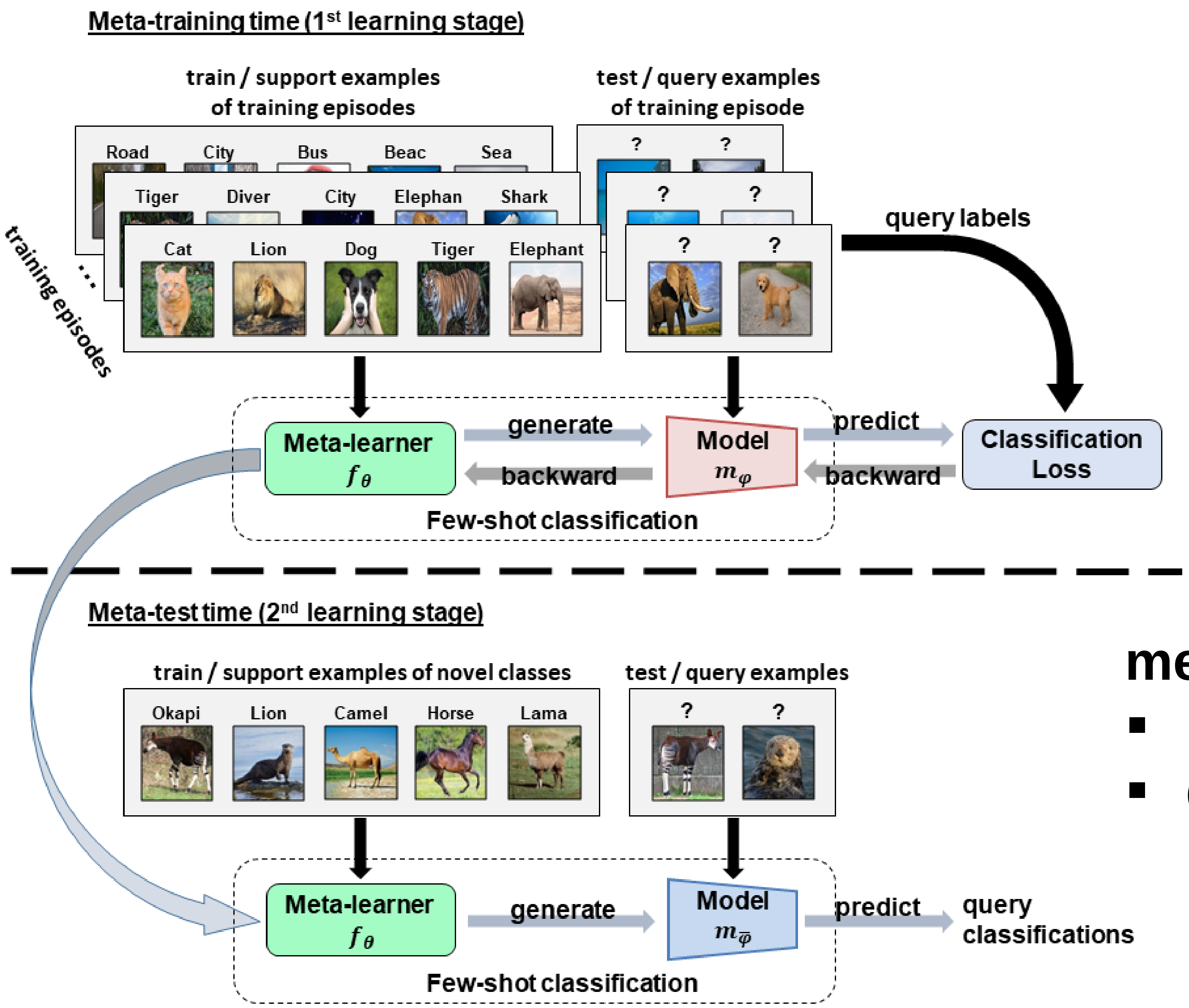
Meta-learning: training time (1st learning stage)



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each x_m^Q in Q
4. **Optimize $\boldsymbol{\theta}$ w.r.t. the queries classification loss $\boldsymbol{L(f_\theta(S), Q)}$**
    - **must back-propagate through the few-shot learning process**

Meta-learning: test time (2nd learning stage)



meta-learner at test time:

- remains fixed (typically)
- generates a model for novel classes

From Supervised Learning to Meta-Learning

- training → meta-training
- test time → meta-test time
- mini-batch of images → mini-batch of few-shot episodes
- training data → meta-training data = all possible training episodes
- test data → meta-test data = test episodes

Few-shot learning vs Meta-learning

Few-shot learning:

- Any transfer learning method that targets on transferring well with limited data
- E.g.: pre-train + fine-tuning, or using metric learning, or using meta-learning

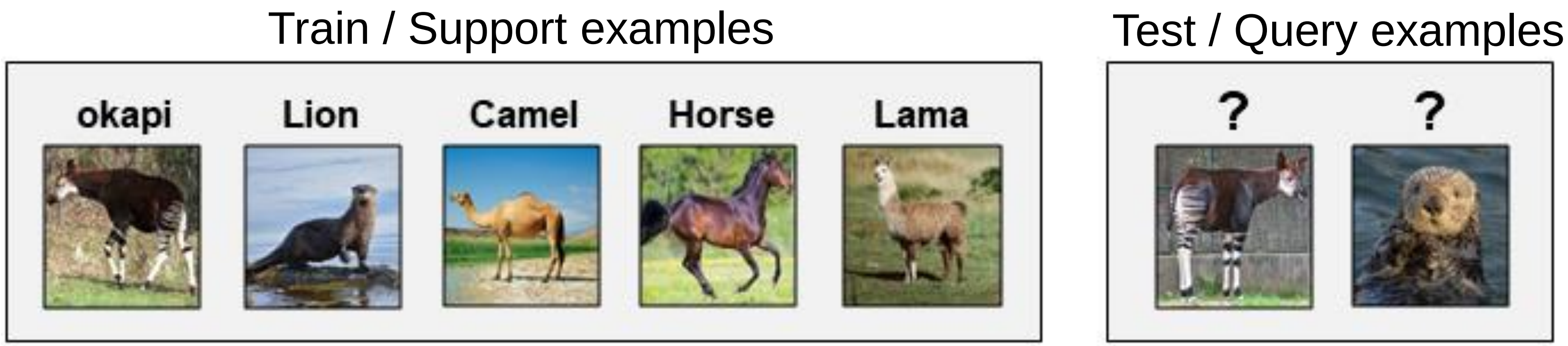
Meta-learning:

- Learn the learning algorithm itself
  - "Learning to learn by gradient descent by gradient descent", Andrychowicz et al. 16
- Ingredient of many few-shot algorithms,
- Also used in multi-task learning, RL, …

Agenda

- Introduction
  - Few-shot learning problem
  - Meta-learning paradigm
  - **How to evaluate**
- Main types of few-shot learning algorithms
- Few-shot learning without forgetting

How to evaluate few-shot algorithms



Example of 5-way 1-shot test task

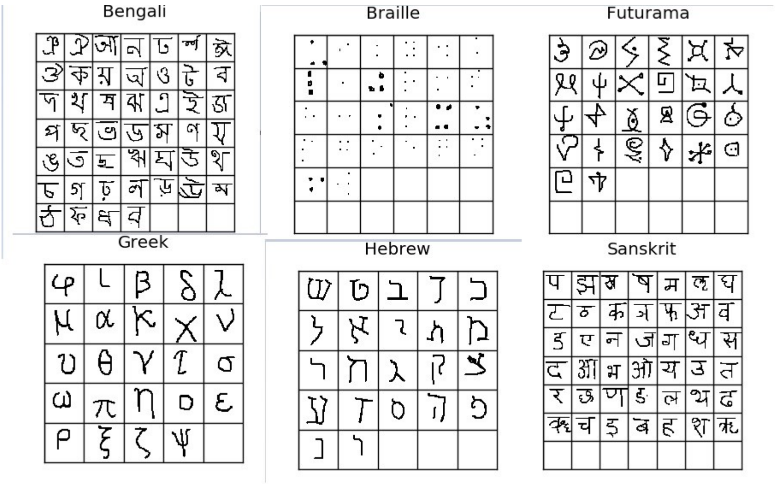
2nd learning stage (meta-test time for meta-learning):

- Use a **held out set of classes**
- Sample a **large number of N-way K-shot few-shot tasks**
- **Report average accuracy** on the **N x M** query examples of all tasks

How to evaluate few-shot algorithms

Datasets / benchmarks

Omniglot: Lake et al. 11
- 1623 characters from 50 alphabets
- 20 instances per character / class
- 5-way and 20-way 1-shot or 5-shot tasks



MiniImageNet: Ravi et al. 17
- 84x84 sized images
- 100 classes: 64 train, 16 val, 20 test
- 1-shot 5-way & 5-shot 5-way tasks

ImageNet-FS: Hariharan et al. 17
- normal ImageNet images
- classes: 389 train, 300 val, 311 test
- 311-way 1, 2, 5, 10, or 20 shot tasks
- more realistic & challenging setting

Also: *tiered*-MiniImageNet (Ren et. al. 18), CIFAR-FS (Bertinetto et al 19), CUB, Tracking in the wild (Valmadre et al. 18), …

Agenda

- Introduction
- Main types of few-shot learning algorithms
  - Metric learning
  - Meta-learning with memory modules
  - Optimization based meta-learning
  - Learn to predict model parameters
- Few-shot learning without forgetting

Agenda

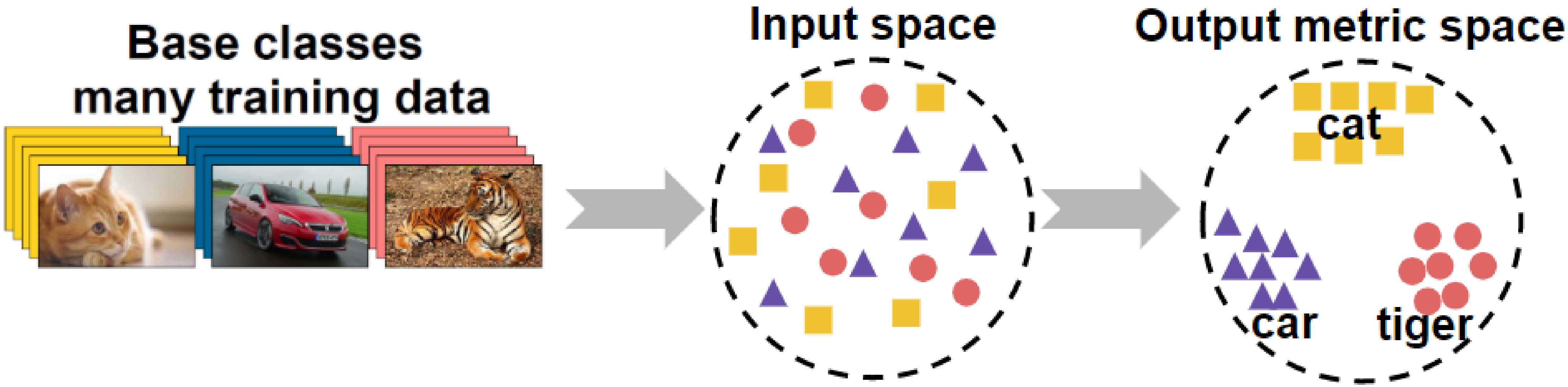
- Introduction
- Main types of few-shot learning algorithms
  - Metric learning
  - Meta-learning with memory modules
  - Optimization based meta-learning
  - Learn to predict model parameters
- Few-shot learning without forgetting

Disclaimer: loose categorization, many combine elements of several types, not exhaustive enumeration

Agenda

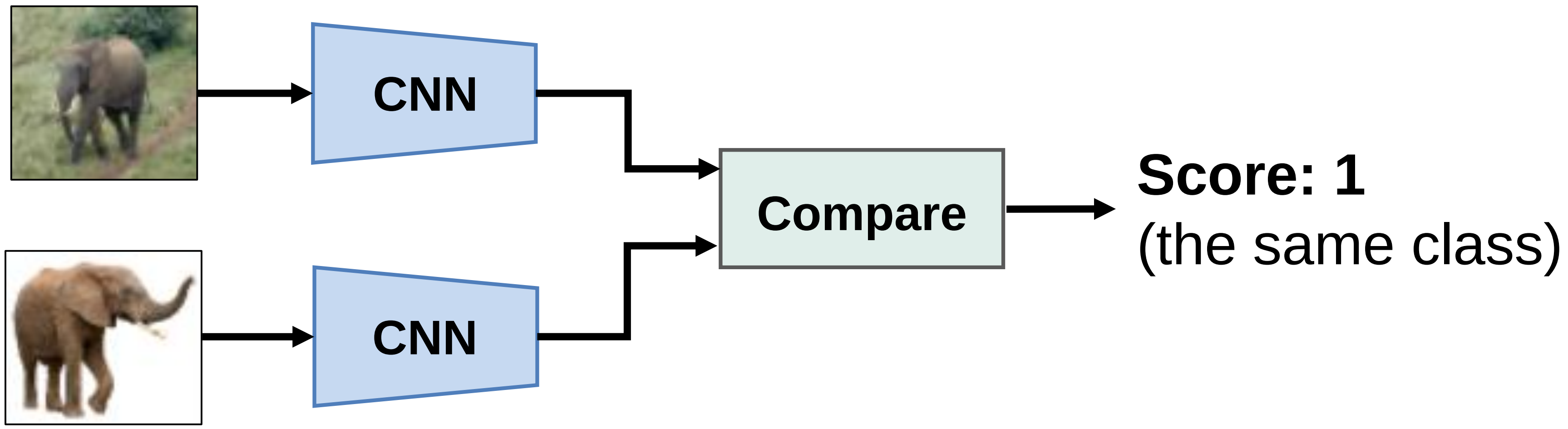
- Introduction
- Main types of few-shot learning algorithms
  - **Metric learning**
  - Meta-learning with memory modules
  - Optimization based meta-learning
  - Learn to predict model parameters
- Few-shot learning without forgetting

Metric learning for few-shot classification



- **1st learning stage:** train a deep metric function on the base class data
- **2nd learning stage:** use it as a nearest neighbor classifier to novel classes
    - **Non-parametric** at this stage
    - **Simple and works well with limited data**

Siamese neural networks



Siamese network:

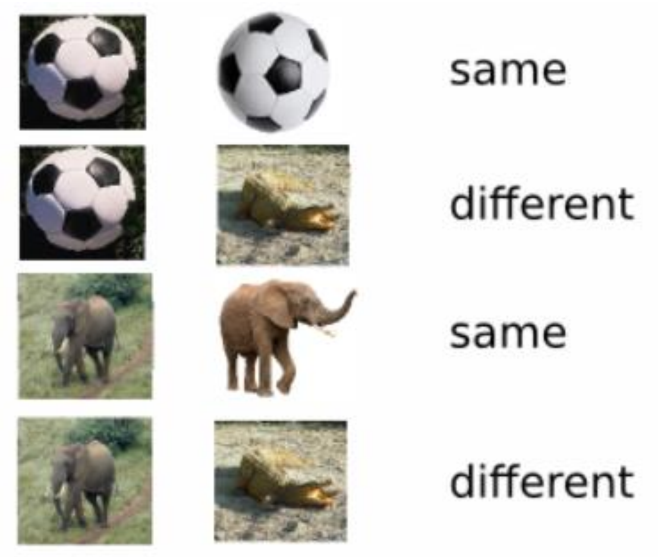
- Given two images: outputs a similarity / distance score.
- Similarity score: 1 if the two image belong to the same class, 0 otherwise

"Siamese neural networks for one-shot image recognition", O. Koch et. al. 2015

Siamese neural networks

1st learning stage – verification task:

Learn with a siamese convnet if 2 images belong to same / different classes.



2nd stage (convnet is fixed):

Classify query to most similar support image

“Siamese neural networks for one-shot image recognition”, O. Koch et. al. 2015

Metric learning

Extensive work on (deep) metric learning:

- “Neighborhood Component Analysis”, Goldberger et. al. 05
- “Dimensionality Reduction by Learning an Invariant Mapping”, Hadsell et. al. 06
- “Distance Metric Learning for Large Margin Nearest Neighbor Classification”, Weinberger et. al. 09
- “Deep Metric Learning Using Triplet Network”, Hoffer et. al. 15
- “Web-Scale Training for Face Identification”, Taigman et. al. 15
- “FaceNet: A Unified Embedding for Face Recognition and Clustering”, Schroff et al 15
- …

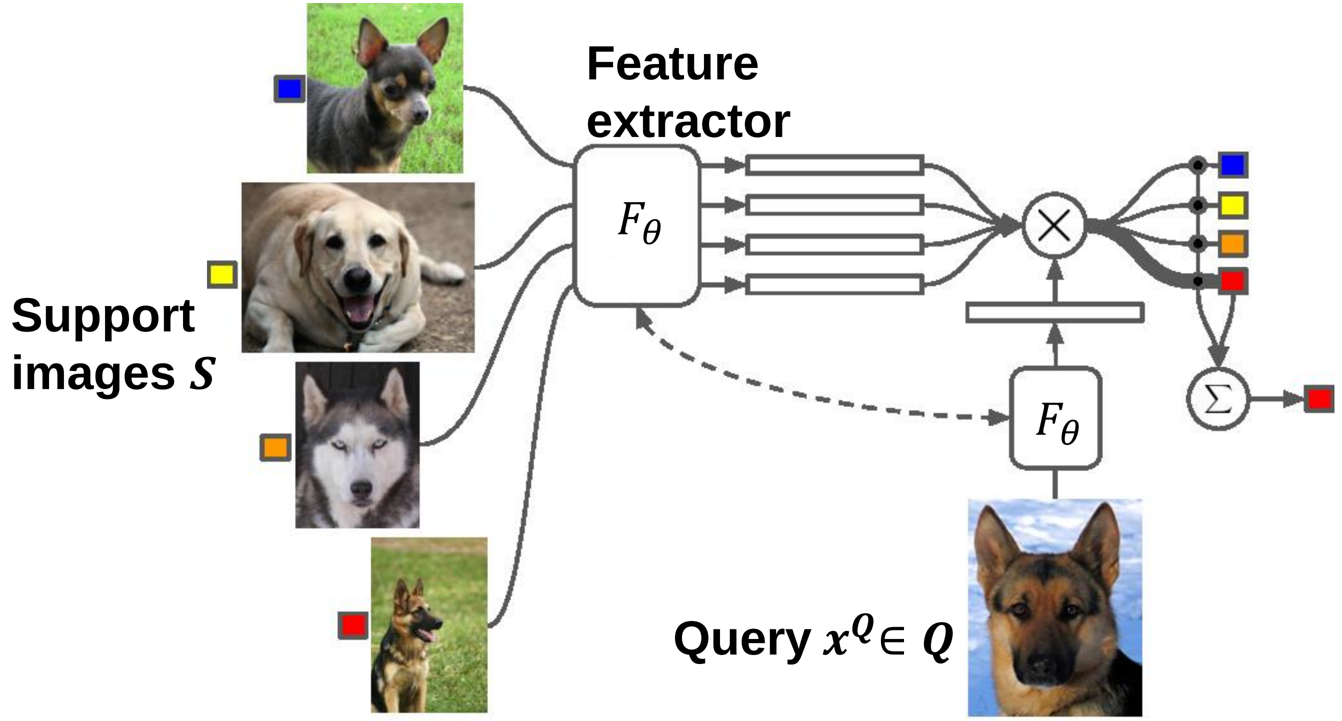
Meta-training based metric learning

Train the metric model on the same way it would be used at 2^{nd} learning stage

- “Matching Networks for one-shot learning”, O. Vinyals et al. 16

Matching Networks

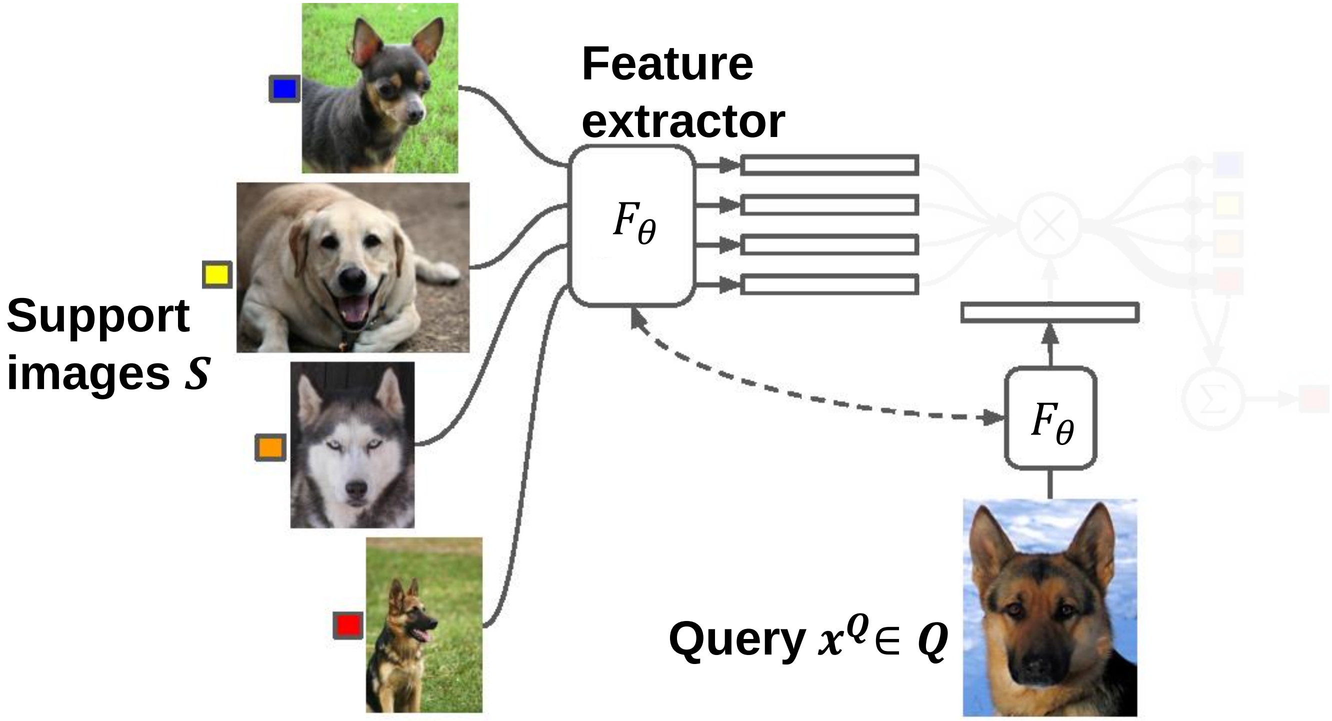
- **Learn to match**



“Matching networks for one shot learning”, Vinyals et al. 2016

Matching Networks

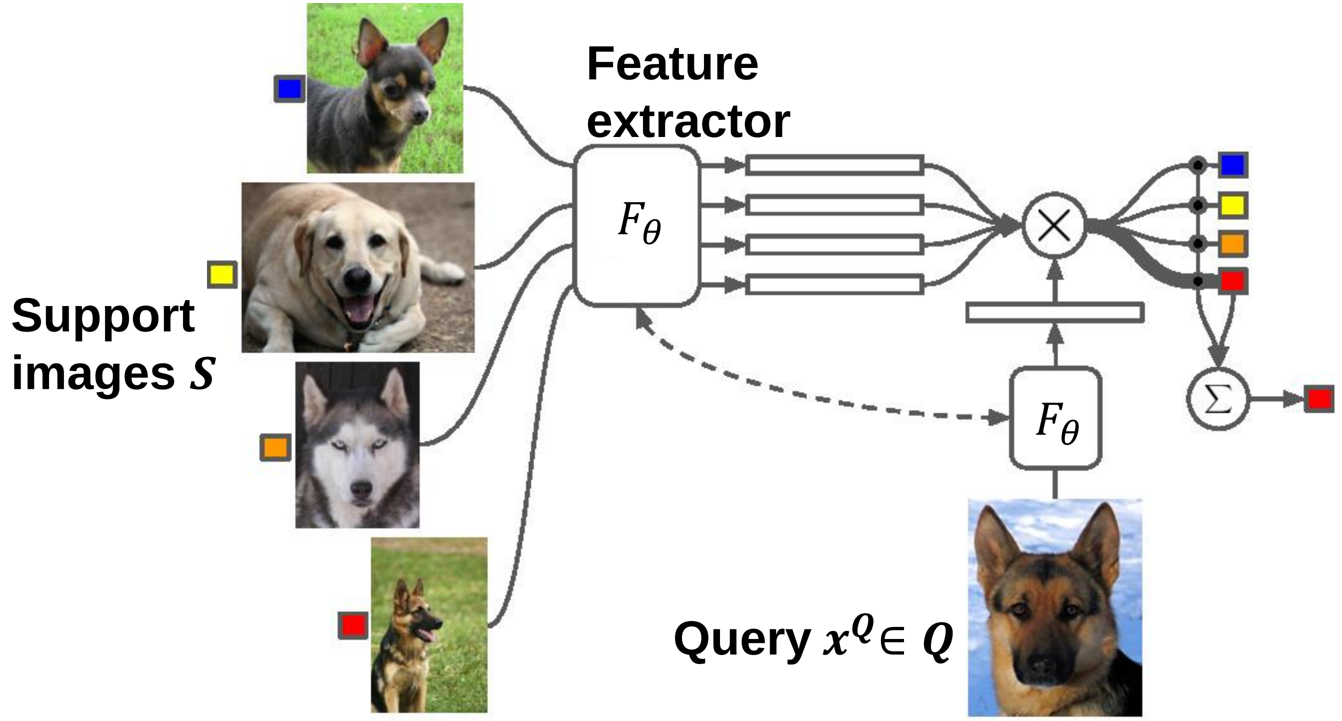
- **Learn to match**
  - Extract features from the query and support images



“Matching networks for one shot learning”, Vinyals et al. 2016

Matching Networks

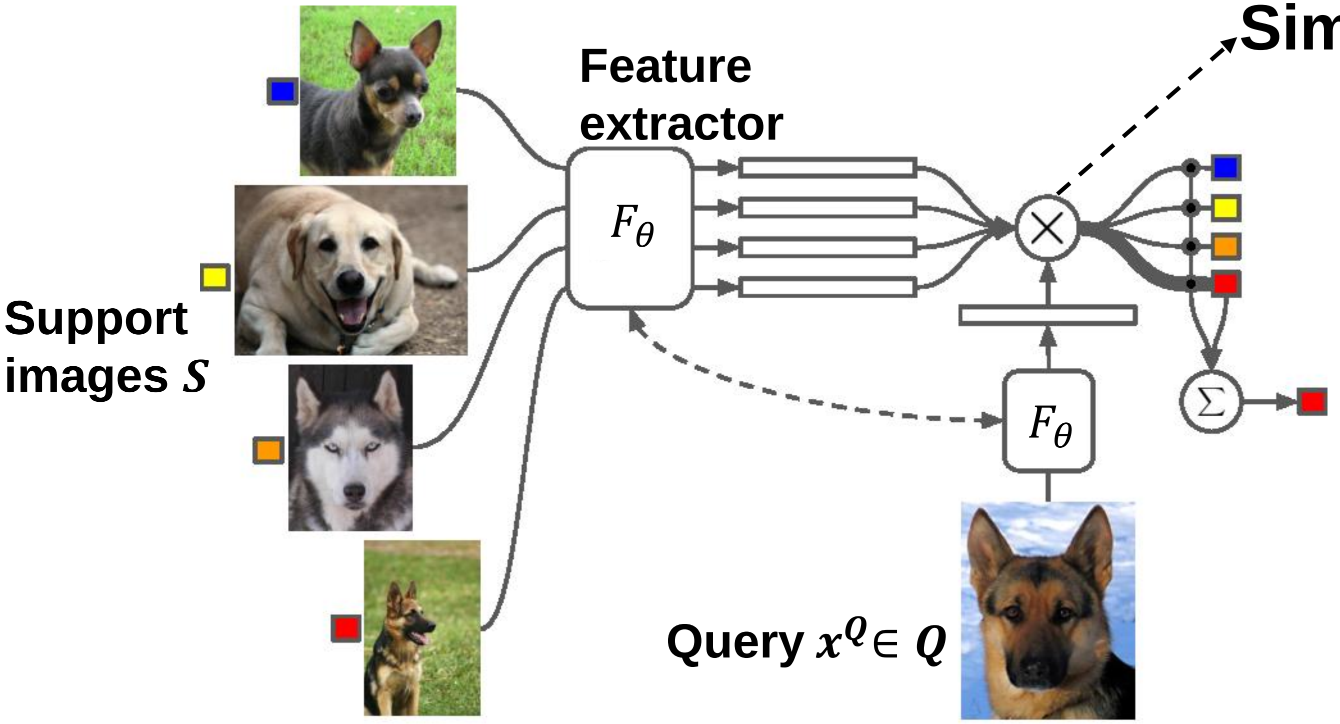
- **Learn to match**
  - Extract features from the query and support images
  - Classify with **differentiable (soft) nearest neighbor classifier**



"Matching networks for one shot learning", Vinyals et al. 2016

Matching Networks

- **Learn to match**
  - Extract features from the query and support images
  - Classify with **differentiable (soft) nearest neighbor classifier**



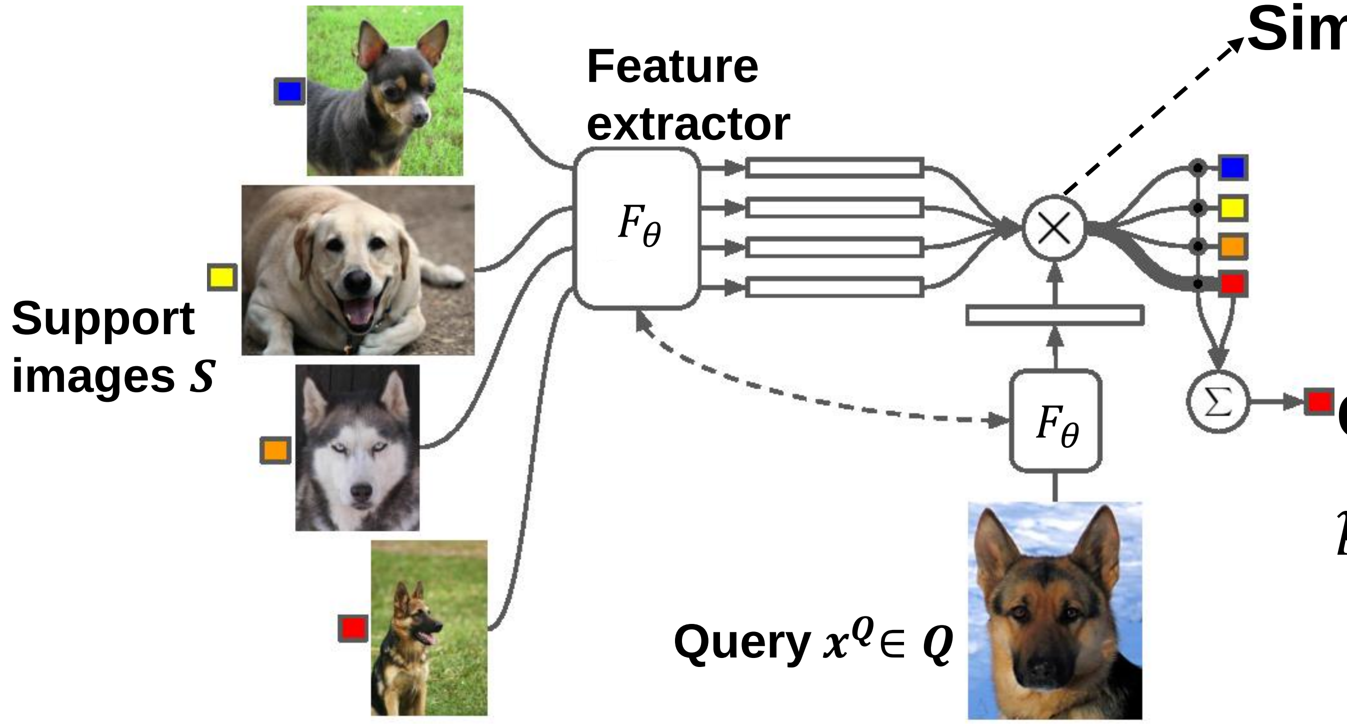
Similarities of x^Q with x_k^S:

$$a(x^Q)[k] = \frac{\exp\left(\cos\left(F_\theta(x^Q), F_\theta(x_k^S)\right)\right)}{\sum_j^{N*K} \exp\left(\cos\left(F_\theta(x^Q), F_\theta\left(x_j^S\right)\right)\right)}$$

“Matching networks for one shot learning”, Vinyals et al. 2016

Matching Networks

- **Learn to match**
  - Extract features from the query and support images
  - Classify with **differentiable (soft) nearest neighbor classifier**



Similarities of x^Q with x_k^S:

$$a(x^Q)[k] = \frac{\exp\left(\cos\left(F_\theta(x^Q), F_\theta(x_k^S)\right)\right)}{\sum_j^{N*K} \exp\left(\cos\left(F_\theta(x^Q), F_\theta\left(x_j^S\right)\right)\right)}$$

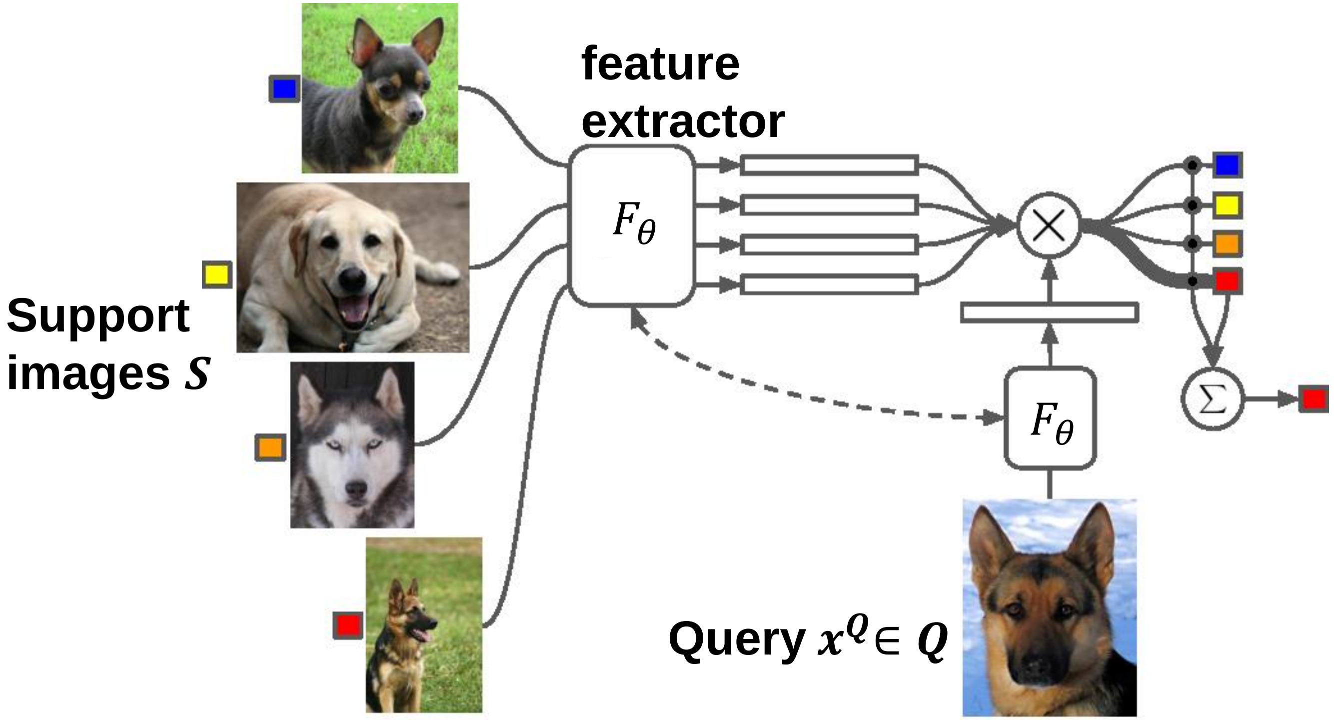
Classification probabilities for x^Q:

$$p = m_\varphi(x^Q) = \sum_k a(x^Q)[k] \cdot one\_hot(y_k^S)$$

“Matching networks for one shot learning”, Vinyals et al. 2016

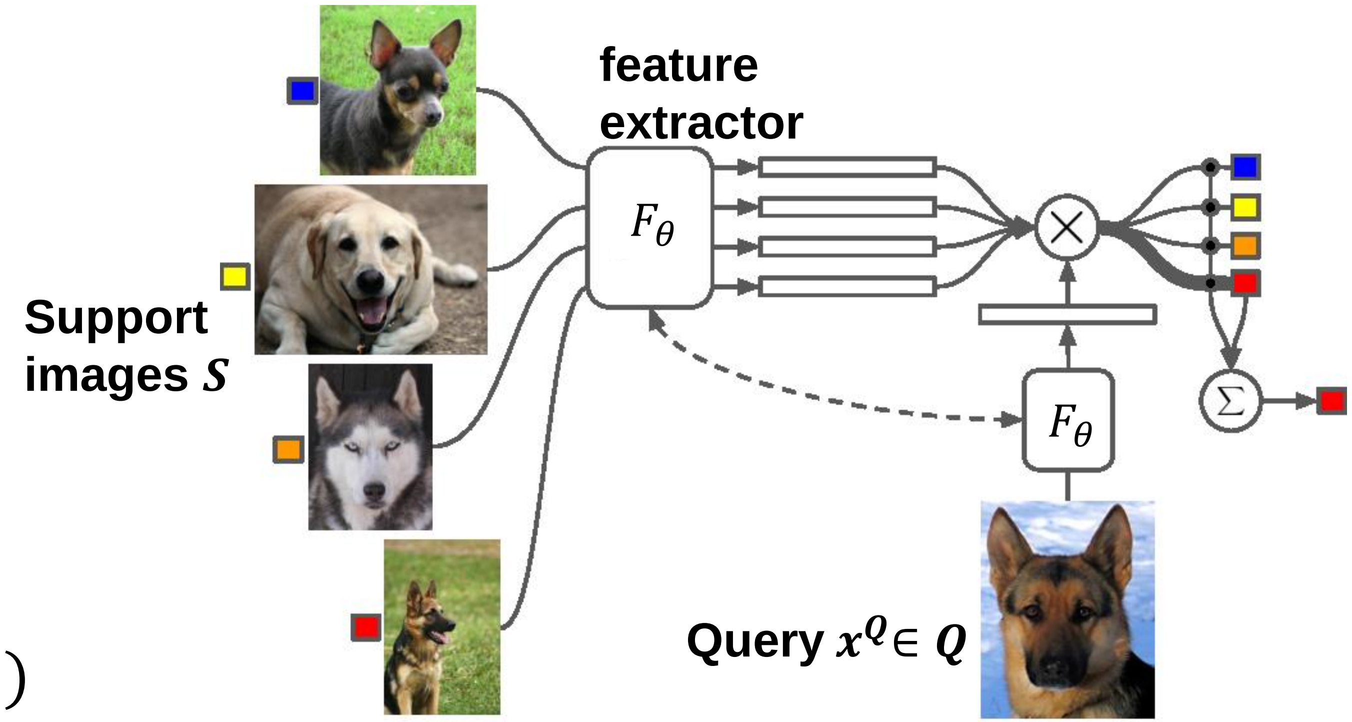
Meta-training in Matching Networks

- **Meta-learner $\boldsymbol{f_\theta}$:** feature extractor $F_\theta(\cdot)$
- **Generated model $\boldsymbol{m_\varphi}$**: extractor $F_\theta(\cdot)$ with support features $\{F_\theta(x_k^S),\ y_k^S\}_{k=1}^{N*K}$



“Matching networks for one shot learning”, Vinyals et al. 2016

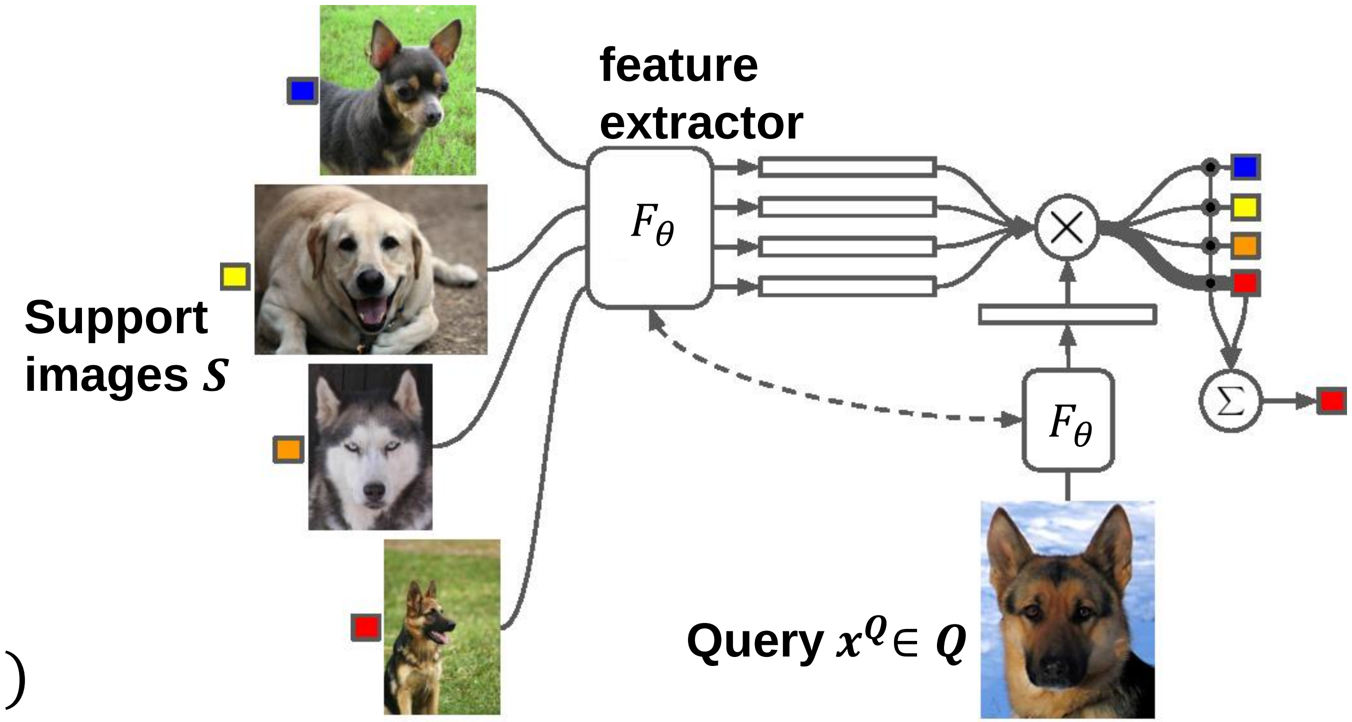
Meta-training in Matching Networks



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S) = \left\{F_\theta(\cdot),\ \{F_\theta\left(x_k^S\right),\ y_k^S\}_{k=1}^{N*K}\right\}$
3. Predict classification scores $p_m = m_\varphi\left(x_m^Q\right) = \sum_k a\left(x_m^Q\right)[k] \cdot one\_hot(y_k^S)$
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(S), Q) = \sum_m -log(p_m[y_m^Q])$

Meta-training in Matching Networks



Meta-training routine:

1. Sample training episode (S, Q)
2. **Generate classification model** $m_\varphi = f_\theta(S) = \left\{F_\theta(\cdot),\ \left\{F_\theta\left(x_k^S\right),\ y_k^S\right\}_{k=1}^{N*K}\right\}$
3. **Predict classification scores** $p_m = m_\varphi\left(x_m^Q\right) = \sum_k a\left(x_m^Q\right)[k] \cdot one\_hot(y_k^S)$
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(S), Q) = \sum_m -log(p_m[y_m^Q])$

Matching Networks

Model	Fine Tune	5-way Acc		20-way Acc	
		1-shot	5-shot	1-shot	5-shot
Baseline Classifier	Y	86.0%	97.6%	72.9%	92.3%
MANN (No Conv) [21]	N	82.8%	94.9%	–	–
Convolutional Siamese Net [11]	N	96.7%	98.4%	88.0%	96.5%
Convolutional Siamese Net [11]	Y	97.3%	98.4%	88.1%	97.0%
Matching Nets (Ours)	N	**98.1%**	**98.9%**	**93.8%**	98.5%
Matching Nets (Ours)	Y	97.9%	98.7%	93.5%	**98.7%**

Table 1: Results on the Omniglot dataset.

Model	Fine Tune	5-way Acc	
		1-shot	5-shot
Baseline Classifier	Y	38.4%	51.2%
Matching Nets (Ours)	N	44.2%	57.0%
Matching Nets (Ours)	Y	**46.6%**	**60.0%**

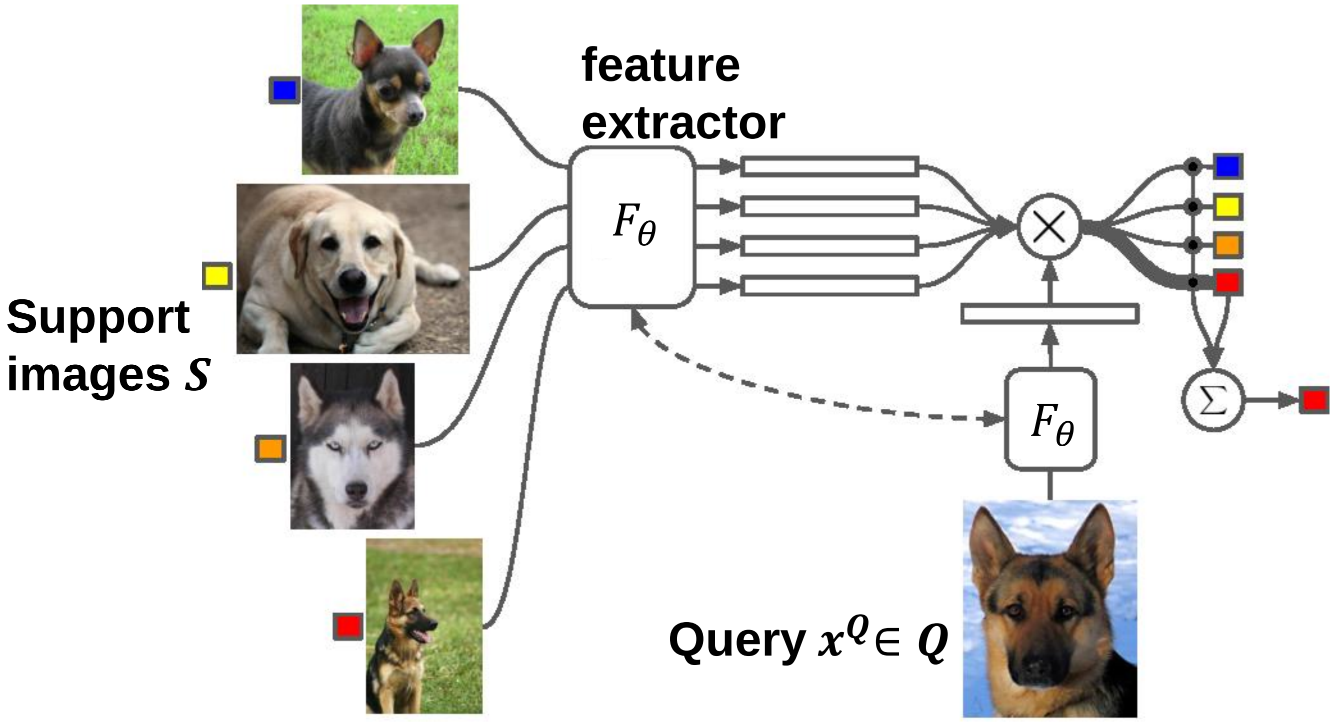
Table 2: Results on *mini*ImageNet.

- **Metric learning:**
  better results than pre-training & fine-tuning
- **Meta-training:**
  improves over siamese networks

Matching Networks

K > 1 support example per class:

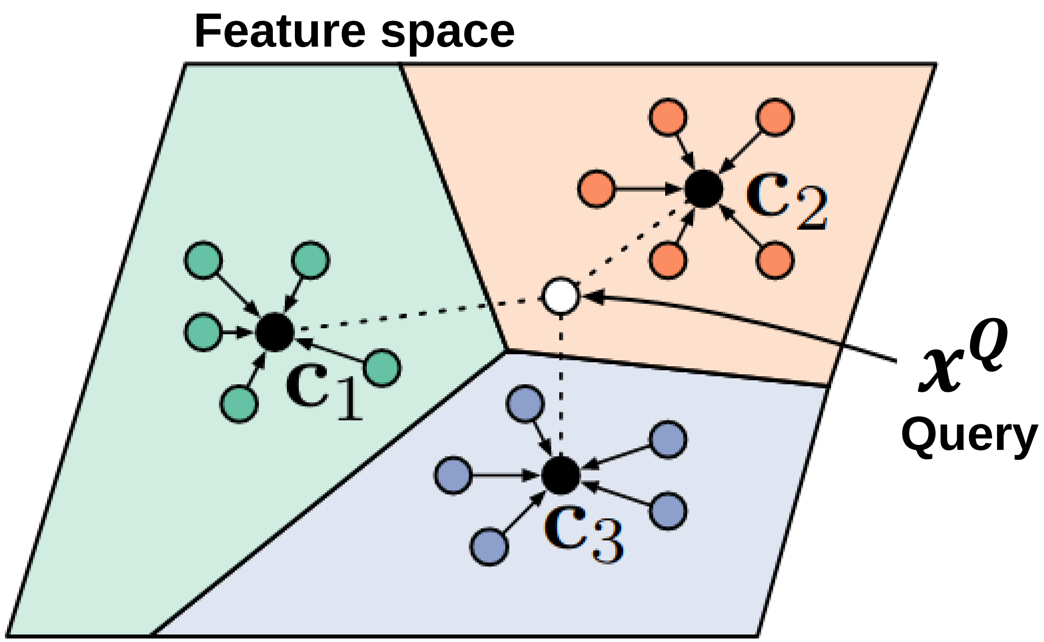
- **Independently matches a query with each support example**
- Can we do something smarter?



"Matching networks for one shot learning", Vinyals et al. 2016

Prototypical Networks

- **Learn to extract class prototypes** for comparisons:
  - **prototype: aggregates information of all support images in a class**

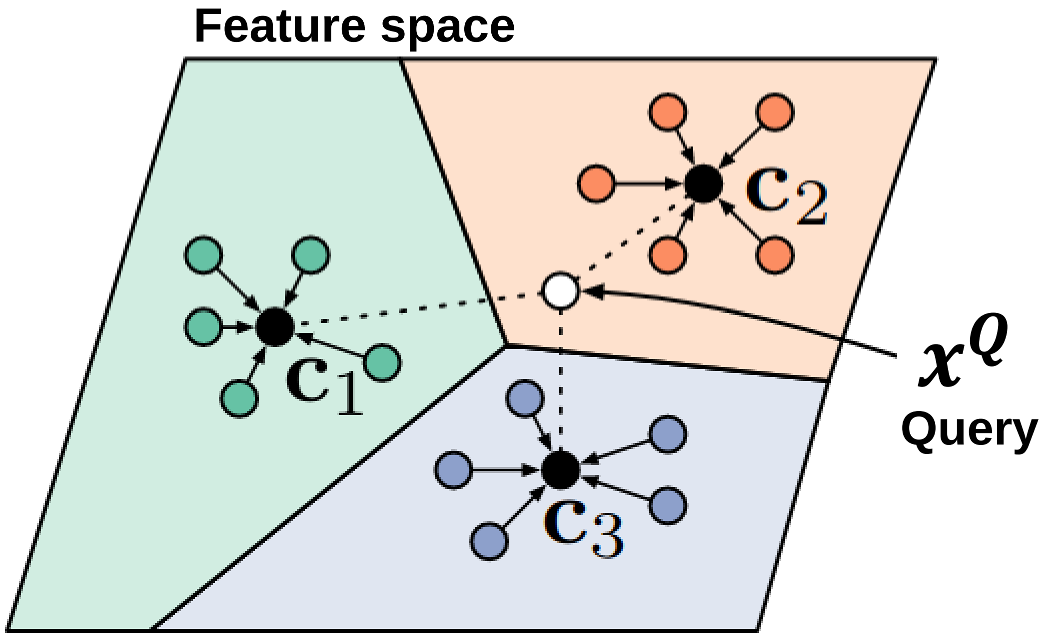


“Prototypical Networks for Few-Shot Learning”, Snell et al. 2017

Prototypical Networks

- **prototype *i*-th class = mean training feature vector** of its support set S_i
  - K=1: the same as matching networks

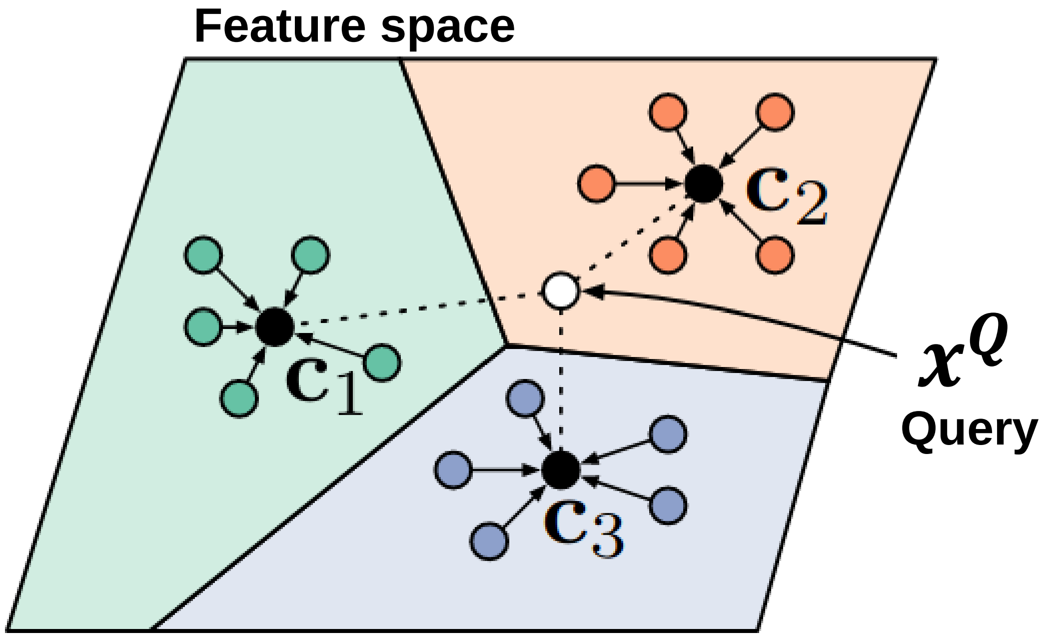
$$c_i = \frac{1}{|S_i|} \sum_{(x_k^S, y_k^S) \in S_i} F_\theta\left(x_k^S\right)$$



Prototypical Networks

- **prototype *i*-th class = mean training feature vector** of its support set S_i

$$c_i = \frac{1}{|S_i|} \sum_{(x_k^S, y_k^S) \in S_i} F_\theta\left(x_k^S\right)$$



- **Classify to closest prototype with prob.**

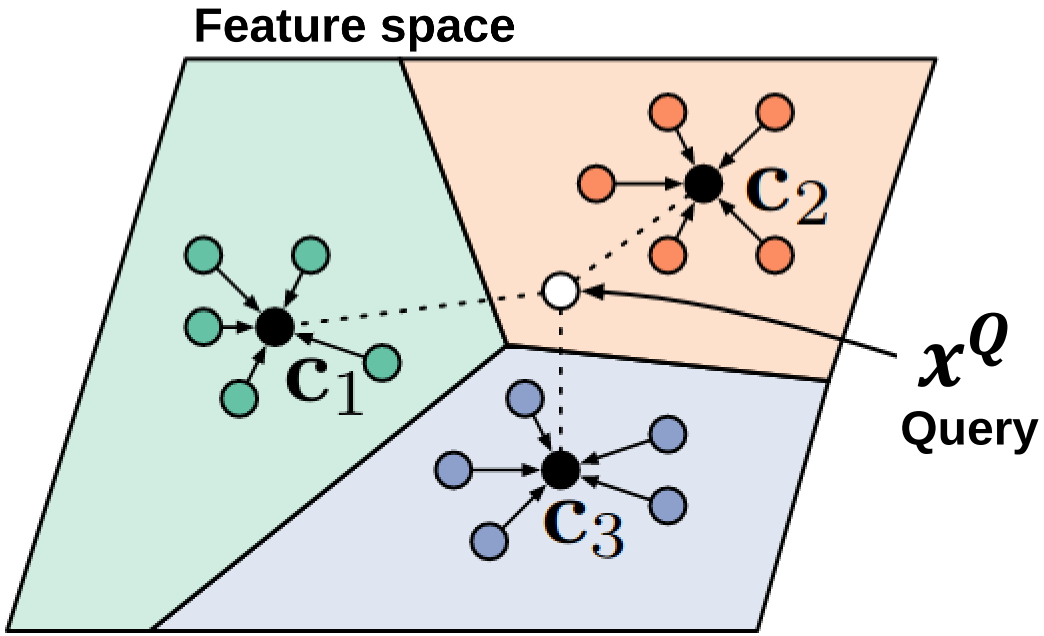
$$p[i] = m_\varphi(x^Q)[i] = \frac{\exp\left(-\mathrm{dist}(F_\theta(x^Q), c_i)\right)}{\sum_j^N \exp\left(-\mathrm{dist}\left(F_\theta(x^Q), c_j\right)\right)}$$

Distance $\mathrm{dist}(\cdot,\cdot)$: Euclidean or cosine

Prototypical Networks

- **prototype *i*-th class = mean training feature vector** of its support set S_i

$$c_i = \frac{1}{|S_i|} \sum_{(x_k^S, y_k^S) \in S_i} F_\theta\left(x_k^S\right)$$



- **Classify to closest prototype with prob.**

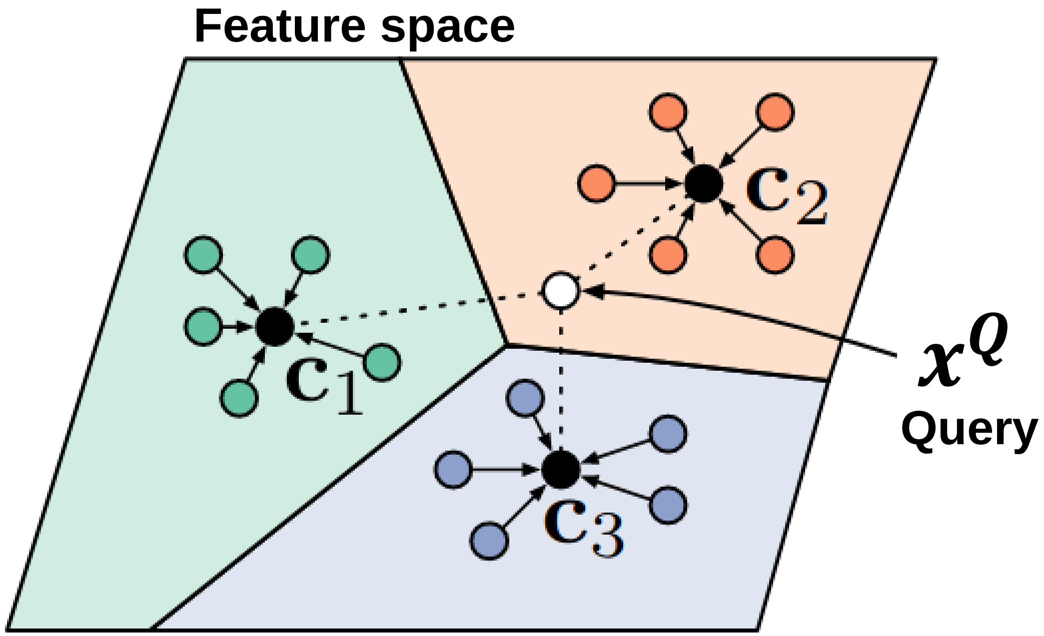
$$p[i] = m_\varphi(x^Q)[i] = \frac{\exp\left(-\mathrm{dist}(F_\theta(x^Q), c_i)\right)}{\sum_j^N \exp\left(-\mathrm{dist}\left(F_\theta(x^Q), c_j\right)\right)}$$

Prototypes: similar to output weights of a classification network with bias = 0

Prototypical Networks

- **prototype $\boldsymbol{i}$-th class = mean training feature vector** of its support set S_i

$$c_i = \frac{1}{|S_i|} \sum_{(x_k^S, y_k^S) \in S_i} F_\theta\left(x_k^S\right)$$

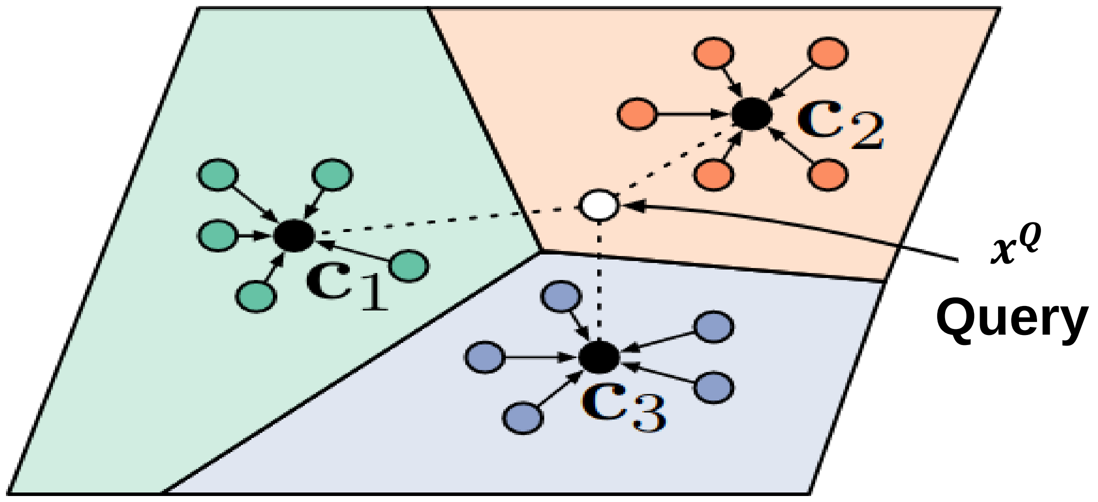


- **Classify to closest prototype with prob.**

$$p[i] = m_\varphi(x^Q)[i] = \frac{\exp\left(-\mathrm{dist}(F_\theta(x^Q), c_i)\right)}{\sum_j^N \exp\left(-\mathrm{dist}\left(F_\theta(x^Q), c_j\right)\right)}$$

During meta-training (optimizing F_θ): back-propagate through the prototypes too

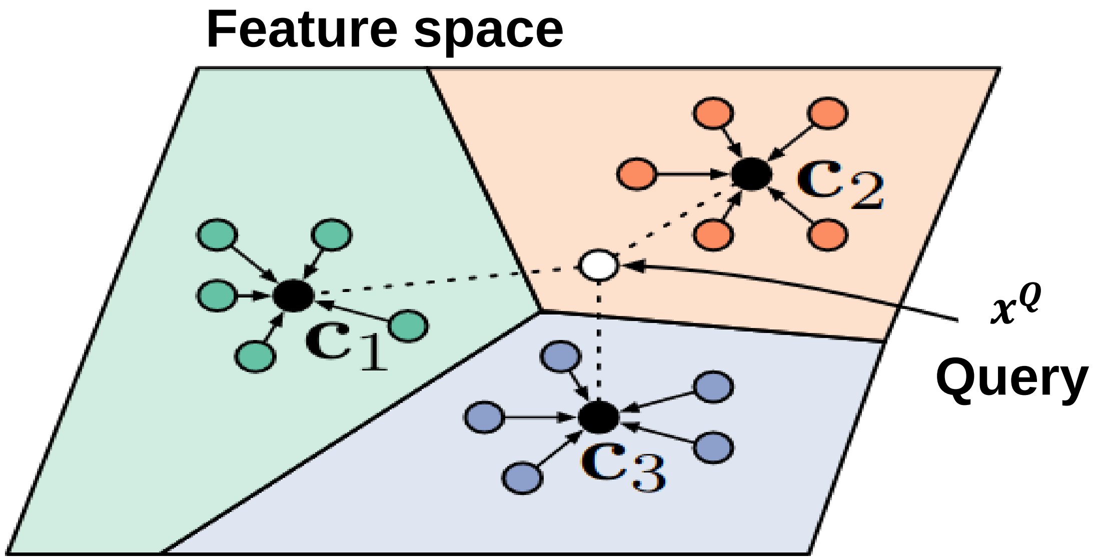
Meta-training in Prototypical Networks



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S) = \left\{F_\theta(\cdot), \{c_i\}_{i=1}^N\right\}$
3. Predict classification scores $p_m = m_\varphi\left(x_m^Q\right) = \frac{\exp\left(-\mathrm{dist}\left(F_\theta\left(x^Q\right), c_i\right)\right)}{\sum_j^N \exp\left(-\mathrm{dist}\left(F_\theta\left(x^Q\right), c_j\right)\right)}$
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(S), Q) = \sum_m -log(p_m[y_m^Q])$

Meta-training in Prototypical Networks



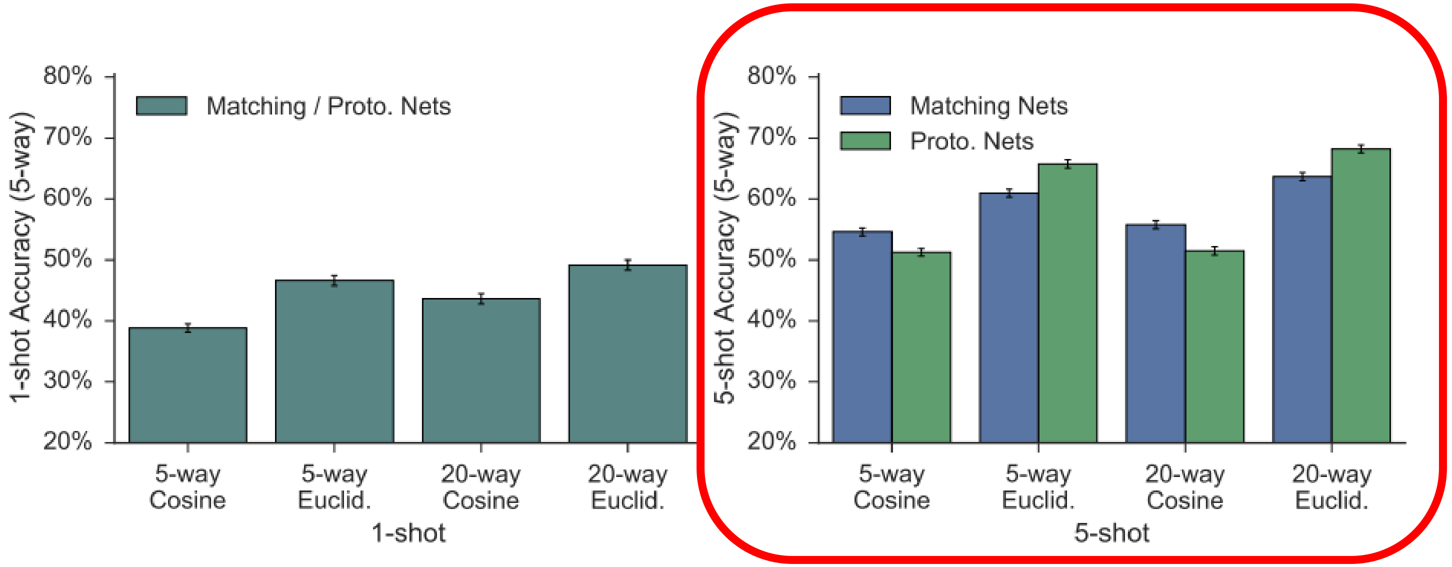
Meta-training routine:

1. Sample training episode (S, Q)
2. **Generate classification model** $m_\varphi = f_\theta(S) = \left\{F_\theta(\cdot), \{c_i\}_{i=1}^N\right\}$
3. **Predict classification scores** $p_m = m_\varphi(x_m^Q) = \frac{exp\left(-dist\left(F_\theta(x^Q), c_i\right)\right)}{\sum_j^N exp\left(-dist\left(F_\theta(x^Q), c_j\right)\right)}$
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(S), Q) = \sum_m -log(p_m[y_m^Q])$

Prototypical Networks

Table 2: Few-shot classification accuracies on *mini*ImageNet. All accuracy results are averaged ove 600 test episodes and are reported with 95% confidence intervals.

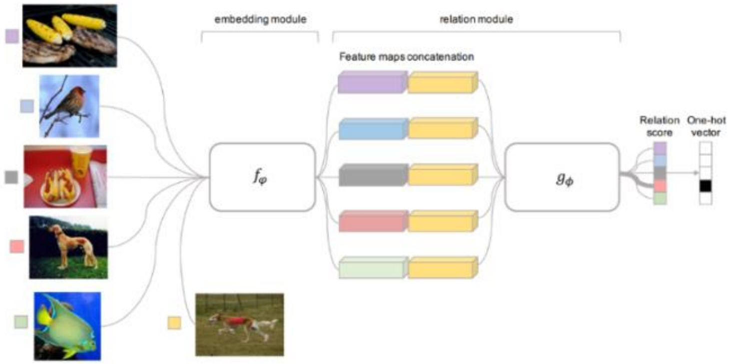
Model	Dist.	Fine Tune	5-way Acc. 1-shot	5-way Acc. 5-shot
BASELINE NEAREST NEIGHBORS*	Cosine	N	$28.86 \pm 0.54\%$	$49.79 \pm 0.79\%$
MATCHING NETWORKS [29]*	Cosine	N	$43.40 \pm 0.78\%$	$51.09 \pm 0.71\%$
MATCHING NETWORKS FCE [29]*	Cosine	N	$43.56 \pm 0.84\%$	$55.31 \pm 0.73\%$
META-LEARNER LSTM [22]*	-	N	$43.44 \pm 0.77\%$	$60.60 \pm 0.71\%$
PROTOTYPICAL NETWORKS (OURS)	Euclid.	N	**$49.42 \pm 0.78\%$**	**$68.20 \pm 0.66\%$**



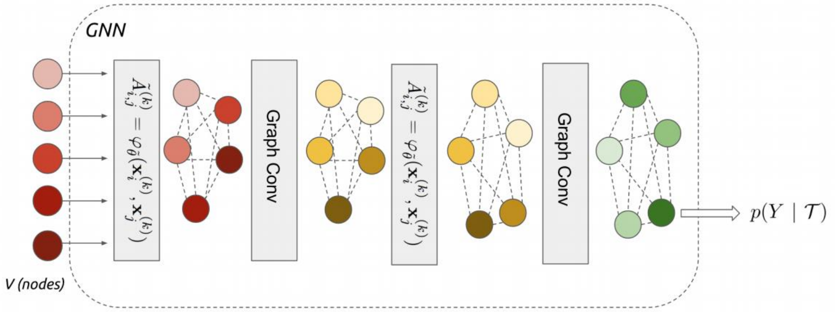
For K>1 shots per class: prototype vectors with Euclidean distance have better accuracy than individual comparison with each support example (Matching Nets)

Meta-training based metric learning

Implement distance function in prototypical nets with a relation network
“Learning to Compare: Relation Network for Few-Shot Learning”, Sung et. al. 18

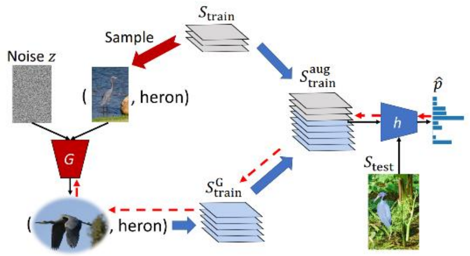


Propagate with a GNN information from the labeled support set to the query
“Few-shot Learning with Graph Neural Networks”, Garcia et al. 18



Learn to synthesize additional support examples for the metric function

“Low-shot learning from imaginary data”, Wang et.al. 18



“Image deformation meta-networks for one-shot learning”, Chen et.al. 19

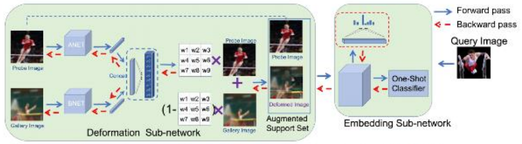
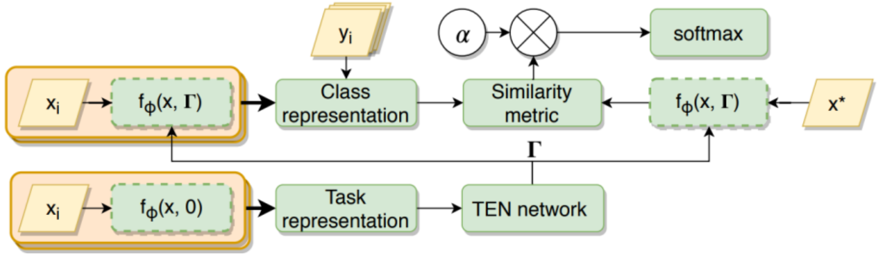


Figure 2. The overall architecture of our image deformation meta-network (IDeMe-Net).

Task-adaptive metric function based on task-context representations
“TADAM: Task dependent adaptive metric for improved few-shot learning”, Oreshkin et. al. 18



Meta-training based metric learning

- **In general: simple and effective methods**

- **But, meta-training can be bothersome:**
  - Train a different metric function for each possible K or N
  - For small N ➔ training with easy examples
  - Not all methods follow this rule, but then, how to tune N, K and M?

Meta-training based metric learning

- In general: simple and effective methods

- But, meta-training can be bothersome

- Is meta-training really necessary for learning good features?

Cosine distance based classification network

- Train typical classification network: **feature extractor + classification head**
- **Classification head:** replace dot-product (i.e., linear layer) with cosine distance

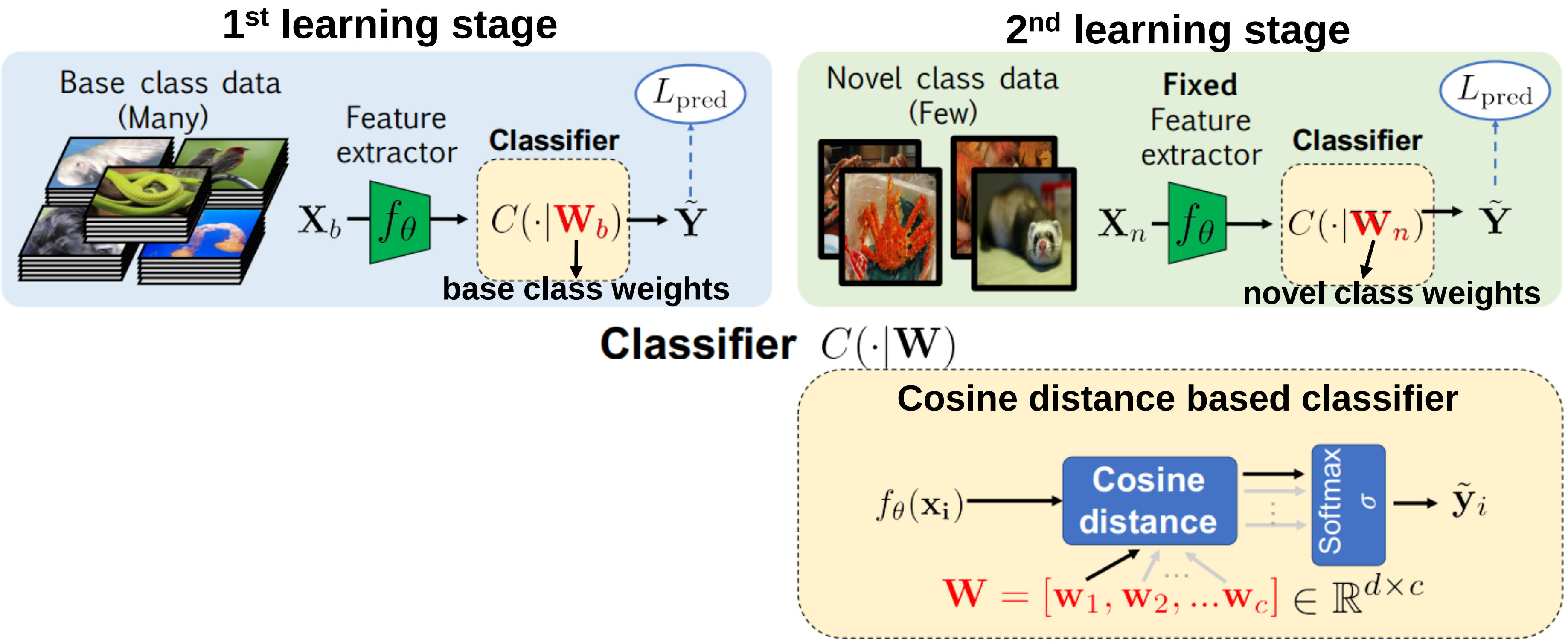


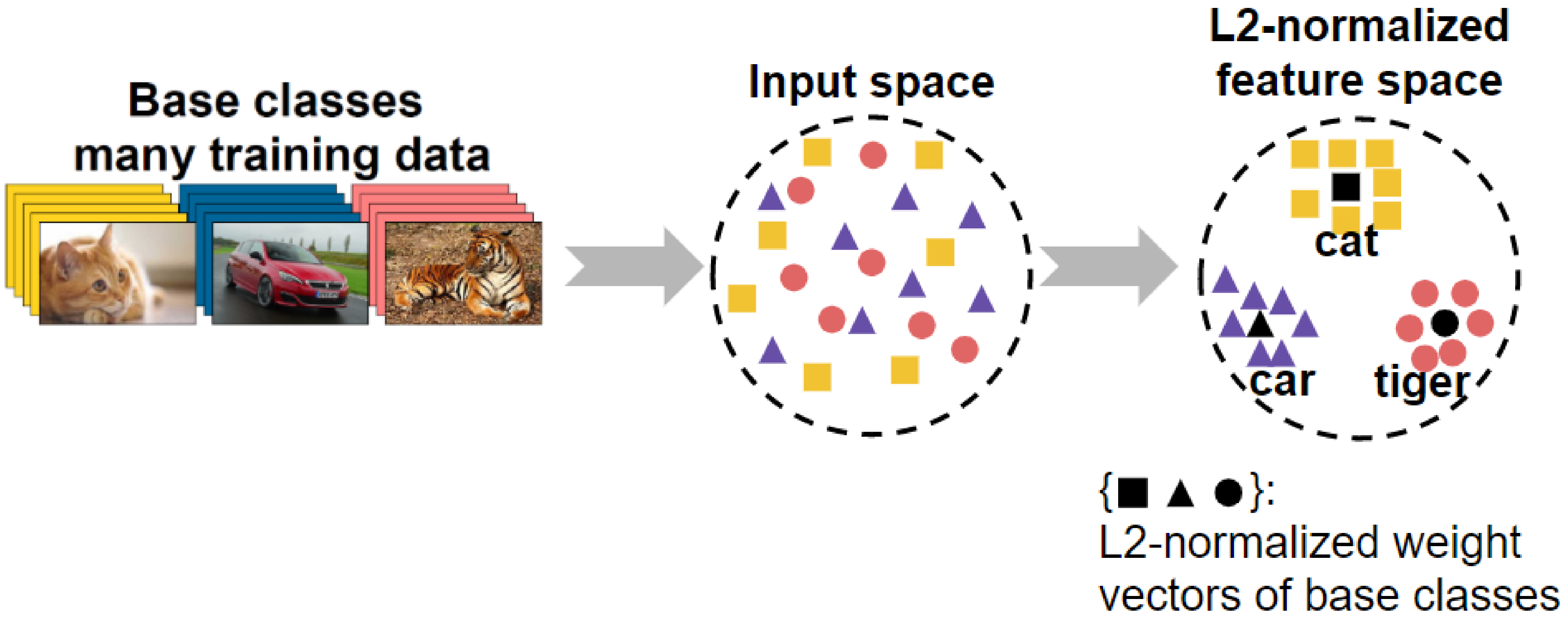
Image source (modified): W. Chen et. al. 2019

"Dynamic Few-Shot Visual Learning without Forgetting", Gidaris et al. 2018
"Low-Shot Learning with Imprinted Weights", Qi et al. 2018

Why distance based classification head?

Enforces **similar behavior as metric learning models:**

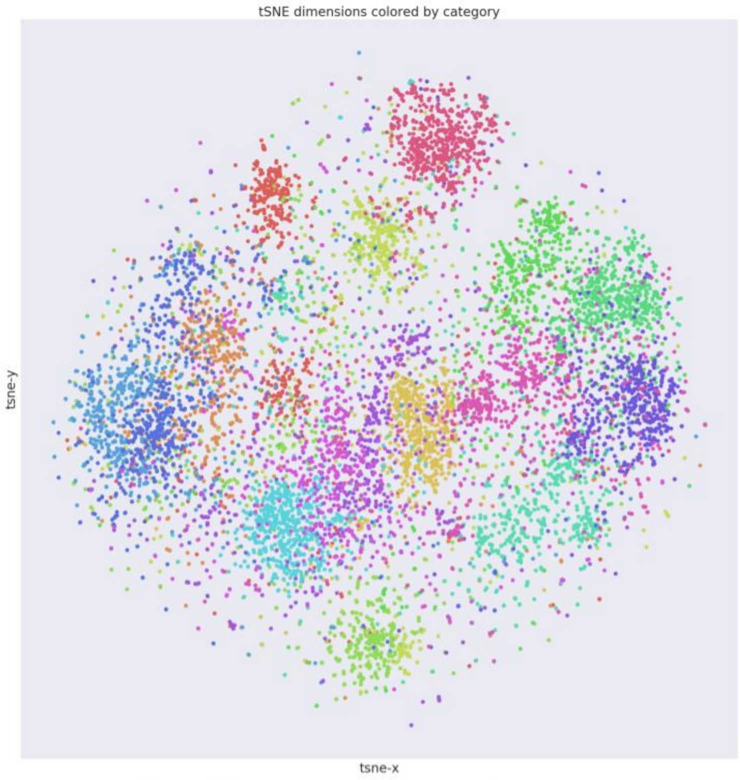
- Given an image, the learned feature must maximize (minimize) cosine similarity with weight vector of the correct class (incorrect classes)



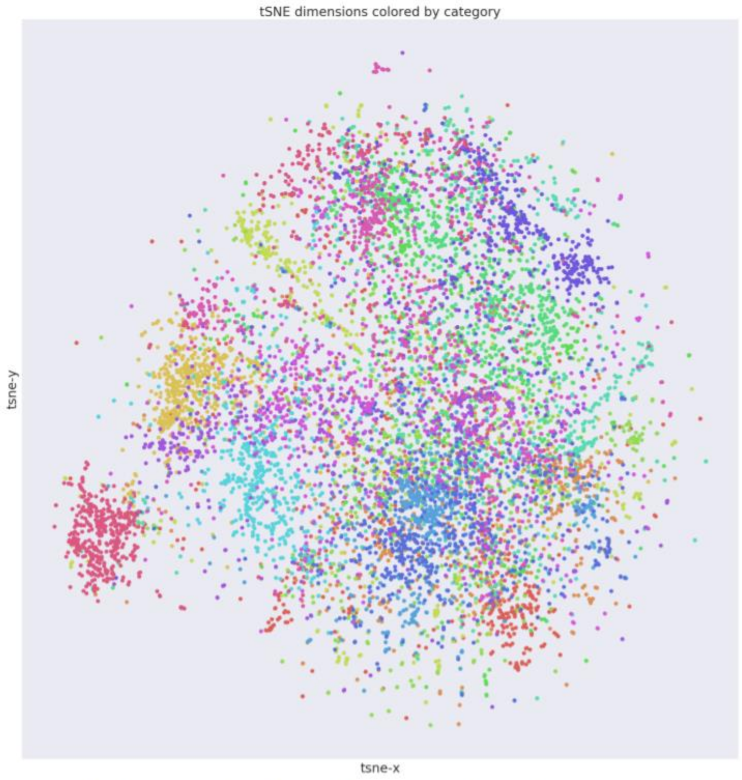
Why distance-based classification head?

Enforces **similar behavior as metric learning models**

- Learn features with **reduced intra-class variance ➔**
- **Better generalization to novel classes**



(a) Cosine-similarity based features of novel categories



(b) Dot-product based features of novel categories

Source:
Gidaris et al. 2018

Cosine distance based classification network

- **1st learning stage:** standard training using the base class data
  - Trains the extractor $\boldsymbol{f_\theta}$ and classification weights $\boldsymbol{W_b}$ of base classes
  - Much simpler than meta-training based metric methods

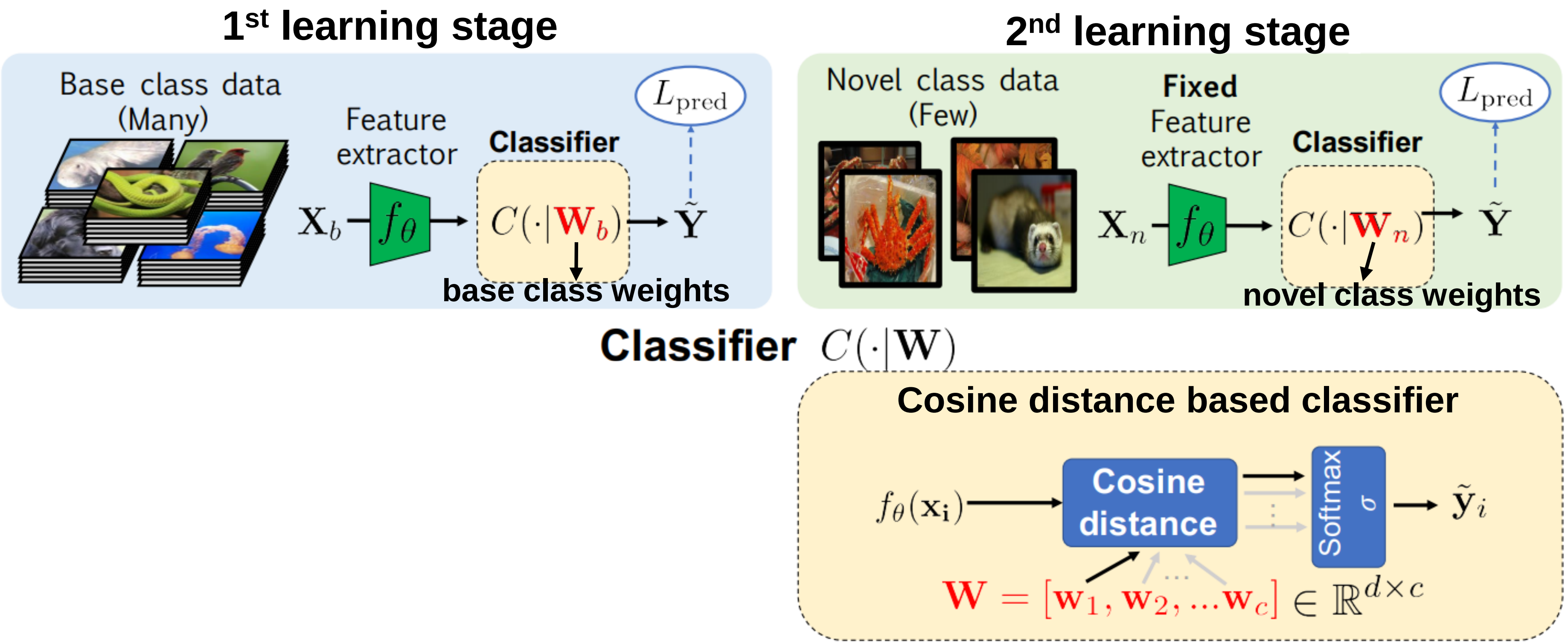
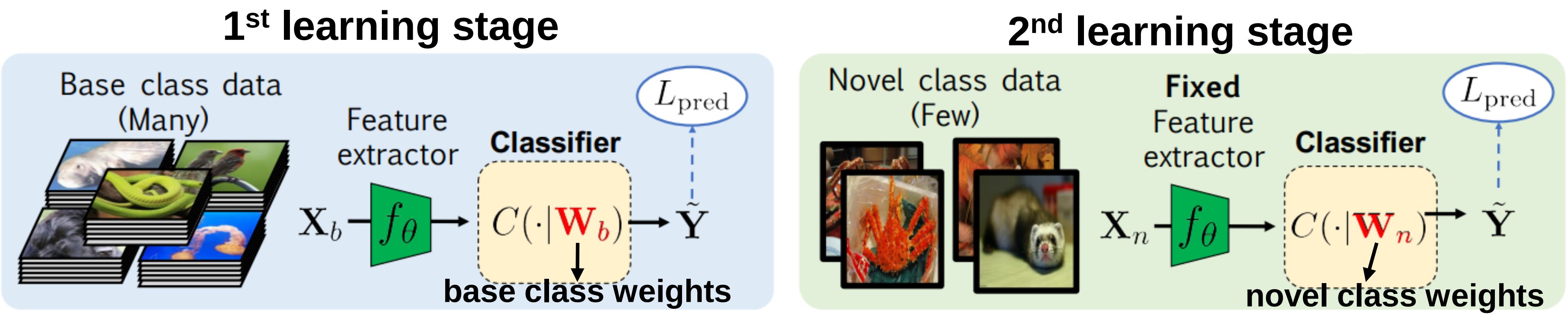


Image source (modified): Chen et al. 2019

Cosine distance based classification network

- **2nd stage:** fix extractor $\boldsymbol{f_\theta}$ + "learn" only the classification weights W_n:
  - **compute W_n with prototypical feature averaging**

$$w_i = \frac{1}{|S_i|} \sum_{(x_k^S, y_k^S) \in S_i} f_\theta\left(x_k^S\right), \forall\, w_i \in W_n$$



Classifier $C(\cdot|\mathbf{W})$

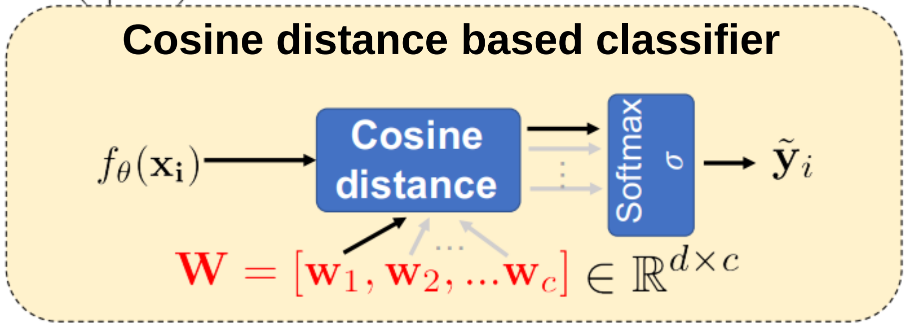


Image source (modified): Chen et al. 2019

Cosine classifier

Models	1-Shot	5-Shot
Matching-Nets [26]	$55.53 \pm 0.48\%$	$68.87 \pm 0.38\%$
Prototypical-Nets [23]	$54.44 \pm 0.48\%$	$72.67 \pm 0.37\%$
Cosine Classifier	$54.55 \pm 0.44\%$	$72.83 \pm 0.35\%$

Table 1: 5-way accuracies on MiniImageNet.

Simpler training with better results than Matching and Prototypical Nets

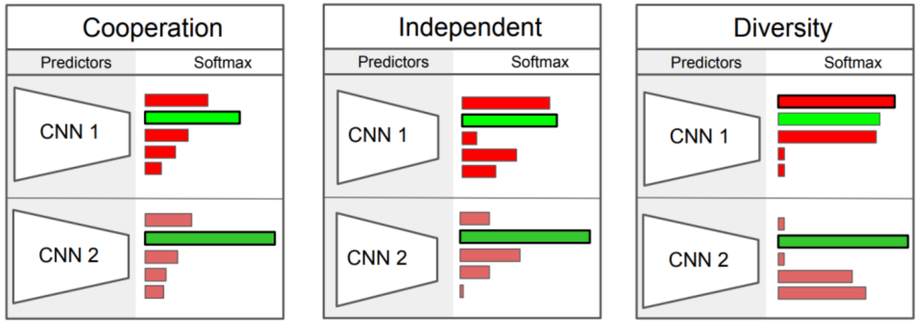
Approach	K =1	2	5	10	20
Prior work					
Prototypical-Nets	39.3	54.4	66.3	71.2	73.9
Matching Networks	43.6	54.0	66.0	72.5	76.9
Cosine Classifier	45.23	56.90	68.68	74.36	77.69

Table 2: 311-way accuracies on ImageNet-FS for *K=1, 2, 5, 10, or 20 examples per novel class.*

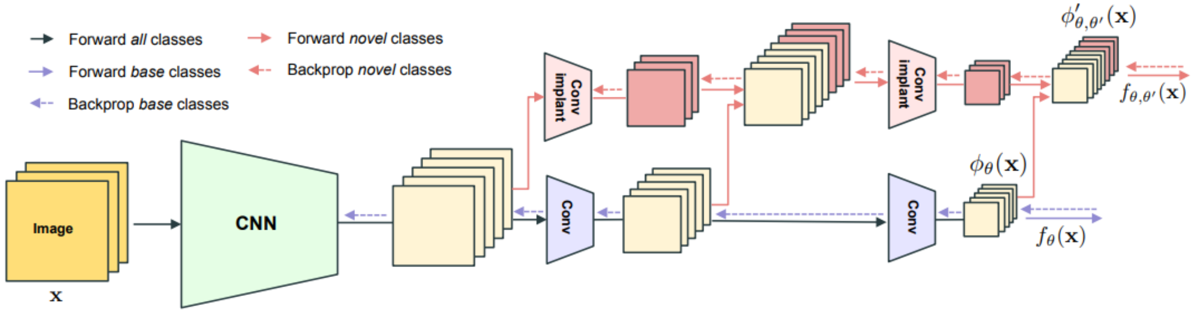
Source: “Dynamic Few-Shot Visual Learning without Forgetting”, Gidaris et al. 18

Cosine classifier

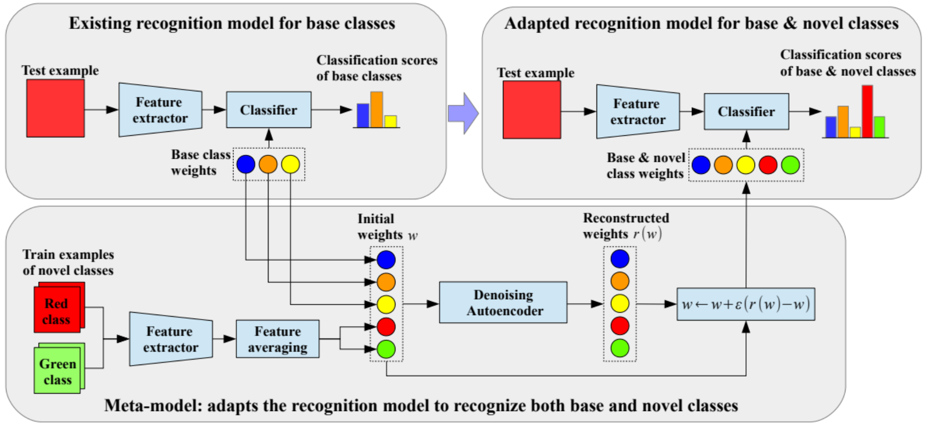
Learn an ensemble of cosine classifiers
"Diversity with cooperation: ensemble methods for few-shot classification", Dvornik et al. 18



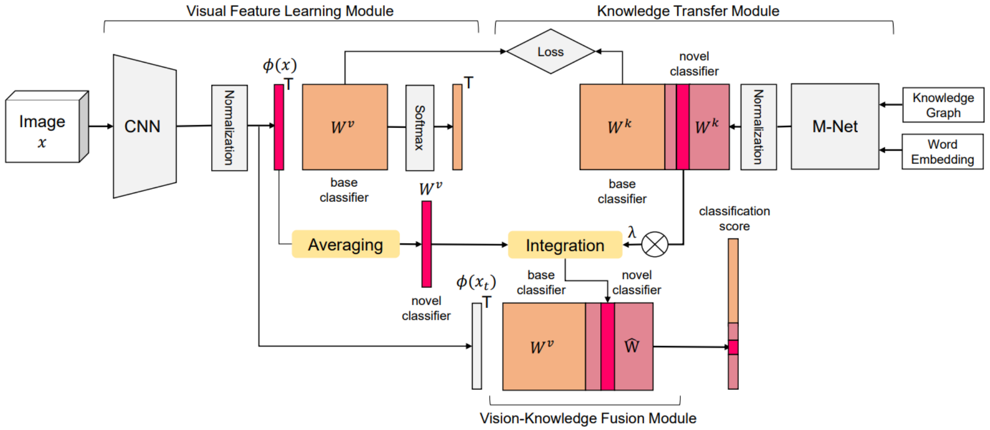
Dense (cosine-based) classification & implanting new task-specific layers
"Dense classification and implanting for few-shot learning", Lifchitz et al. 19



Learn to predict class prototypes for pre-trained cosine classifiers
Gidaris et al. 18, Gidaris et al. 19



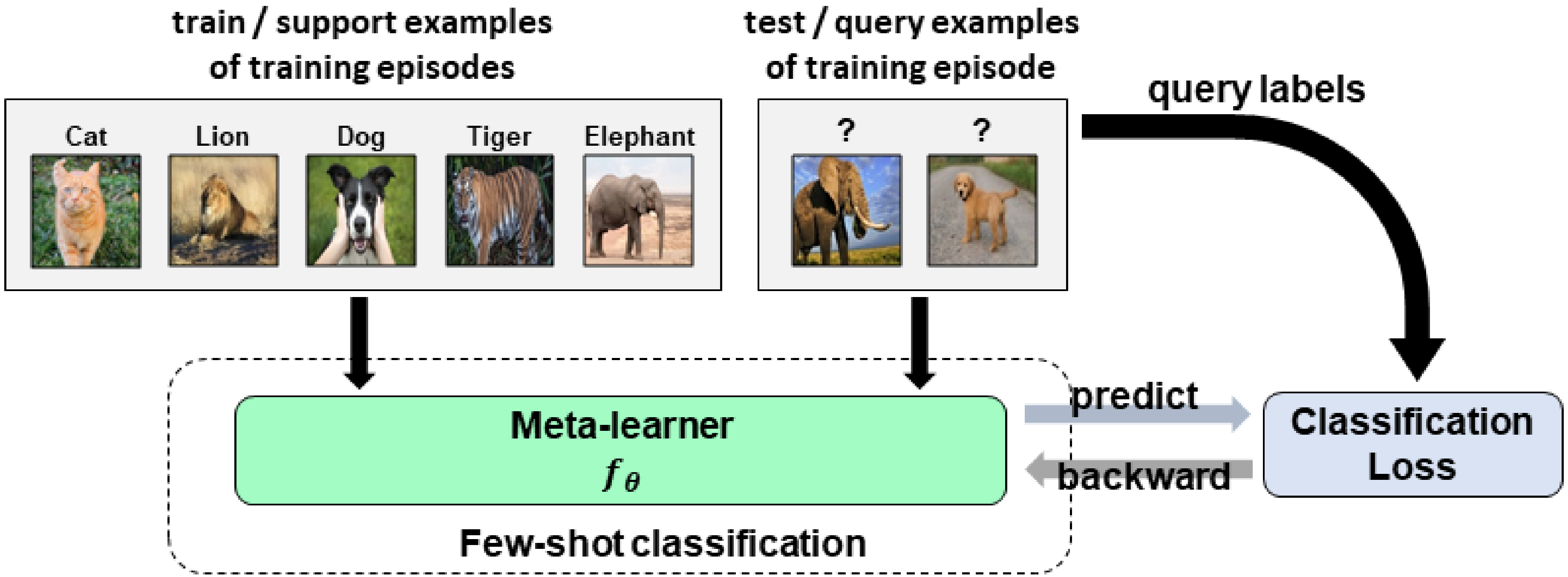
Learn to predict class prototypes for cosine classifiers leveraging word embedding based knowledge graphs Peng et al. 19



Agenda

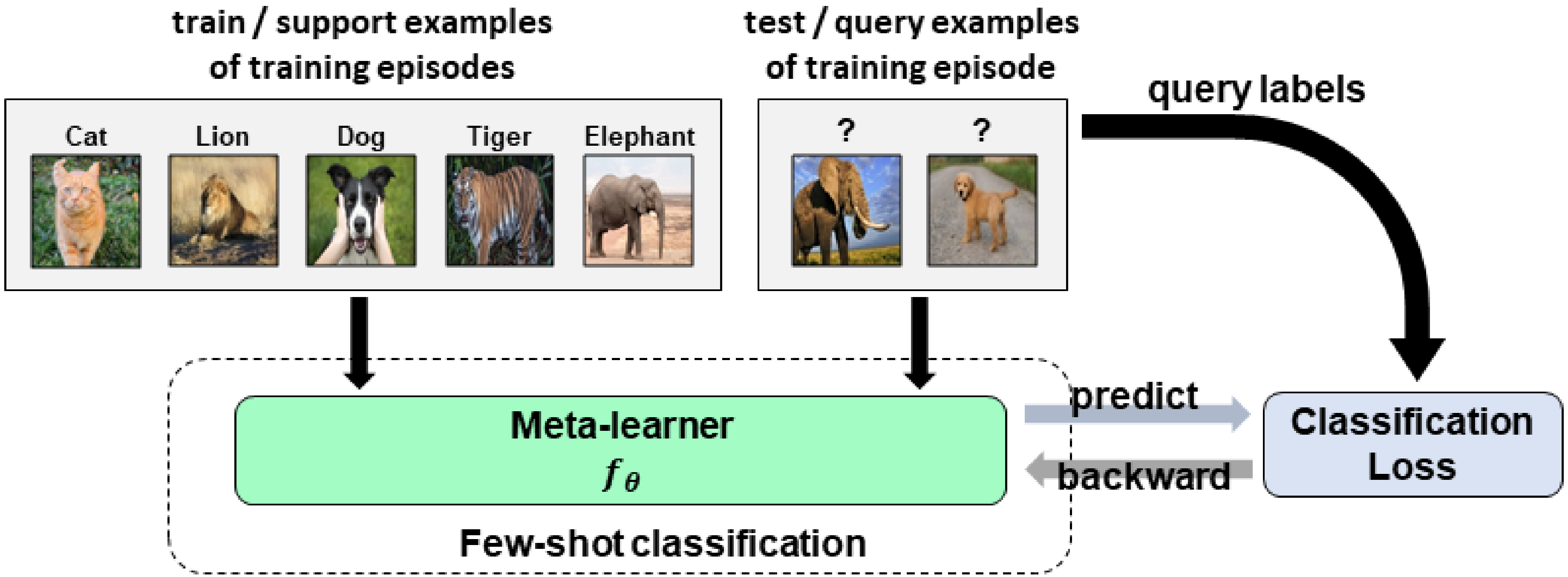
- Introduction
- Main types of few-shot learning algorithms
  - Metric learning
  - **Meta-learning with memory modules**
  - Optimization based meta-learning
  - Learn to predict model parameters
- Few-shot learning without forgetting

Meta-learning with memory modules



- **Few-shot classification:**
  - **input:** labeled support data, unlabeled query data
  - **intermediate step:** generate model
  - **output:** predicted query labels

Meta-learning with memory modules

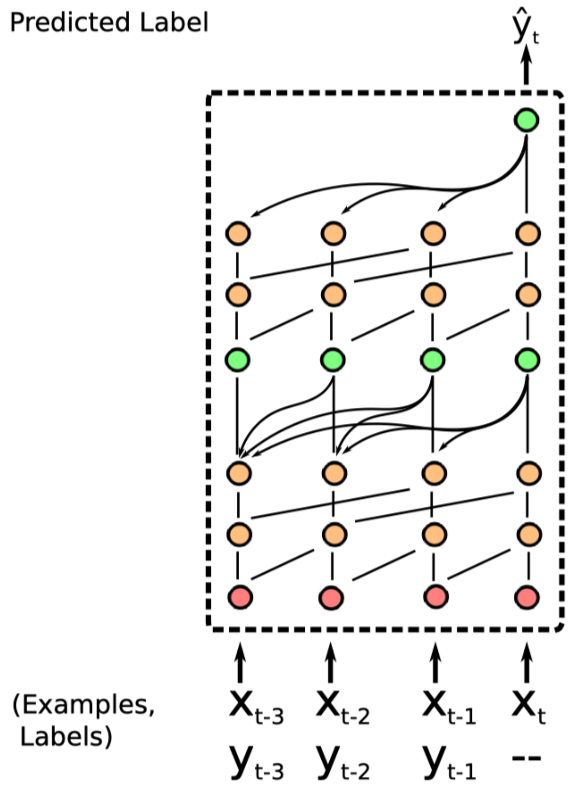


- **Few-shot classification:**
  - **input:** labeled support data, unlabeled query data
  - ~~**intermediate step:** generate model~~ ➔ **store support data to memory**
  - **output:** predicted query labels **by accessing the memory**

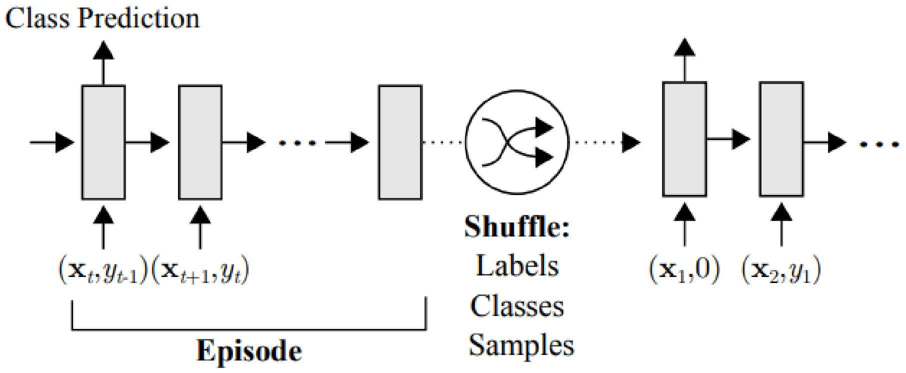
Treats **few-shot classification as a “black box” prediction problem**

Meta-learning with memory modules

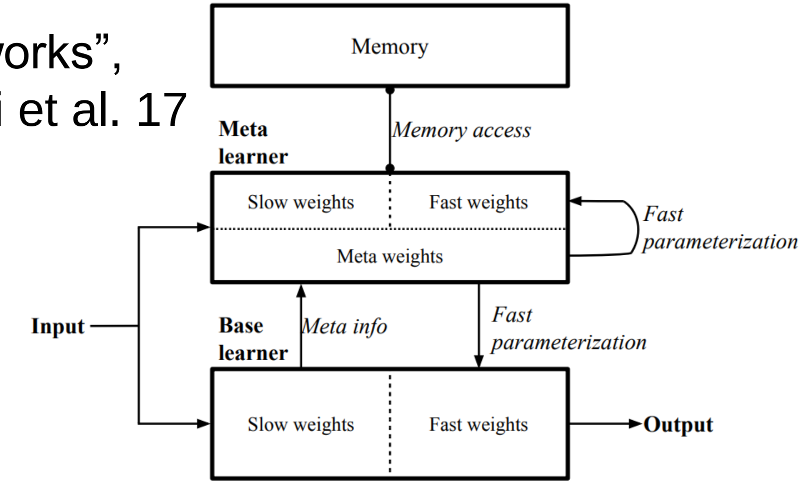
"A Simple Neural Attentive Meta-Learner", Mishra et al. 18



"Meta-Learning with Memory-Augmented Neural Networks", Santoro et al. 16

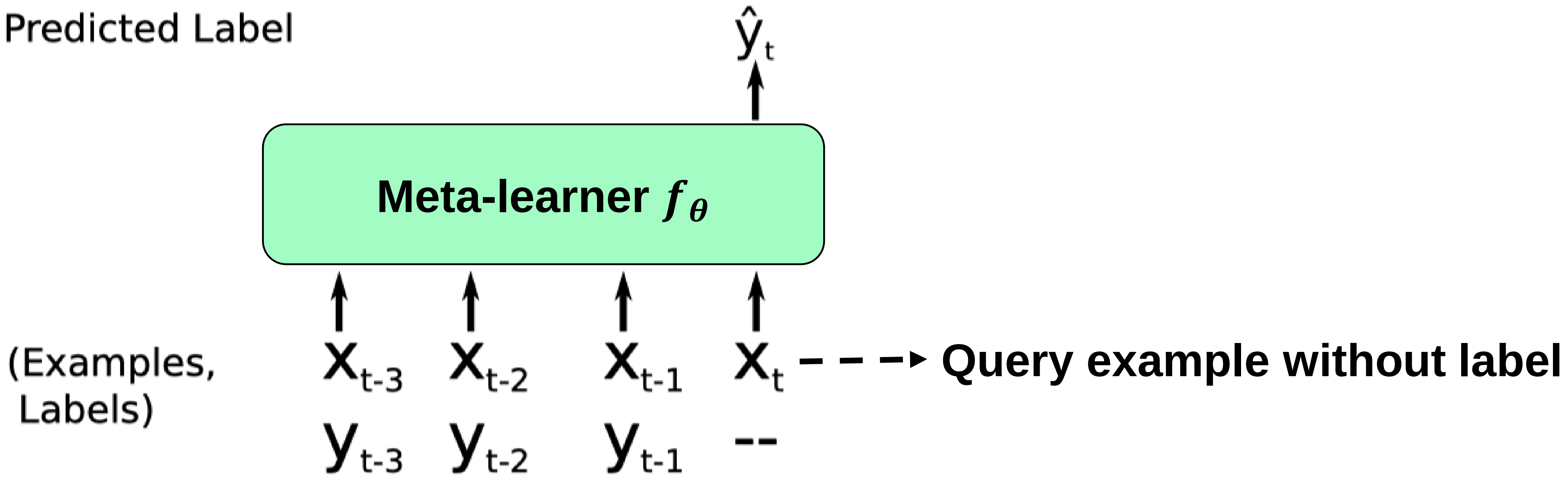


"Meta Networks", Munkhdalai et al. 17



Example: Simple Neural Attentive Meta-Learner

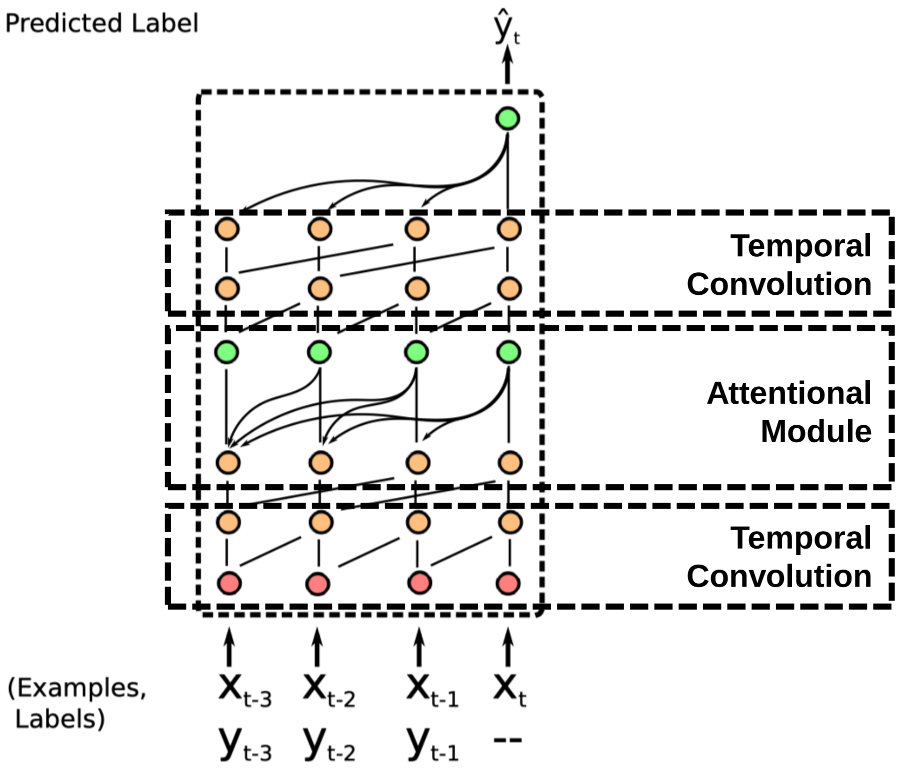
- **Few-shot as a sequence labeling task:**
  - Given past labeled images, what is the label of the current query image



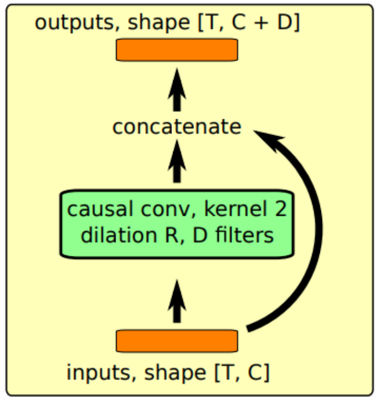
"A Simple Neural Attentive Meta-Learner", Mishra et al. 18

Example: Simple Neural Attentive Meta-Learner

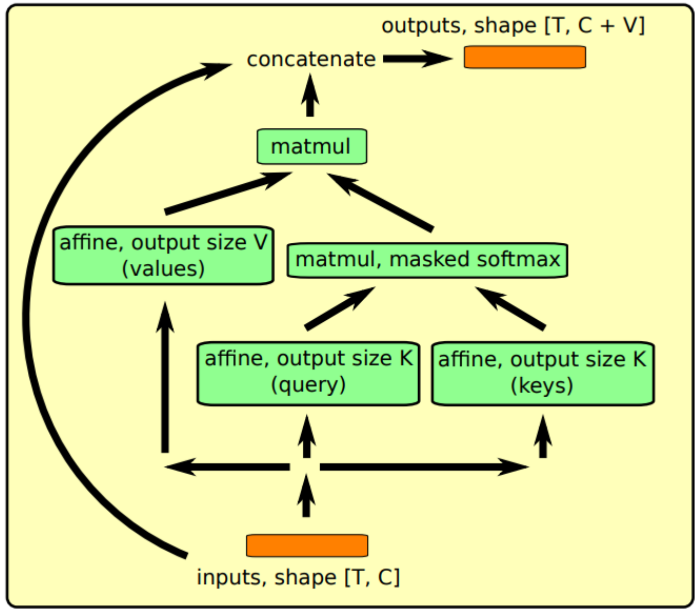
- **Meta-learner implementation:**
  - **Temporal convolutions:** aggregates past information
  - **Attentional Module:** pinpoints to query-specific past information
    - “Attention is all you need”, Vaswani et al. 17



Temporal Conv. Module

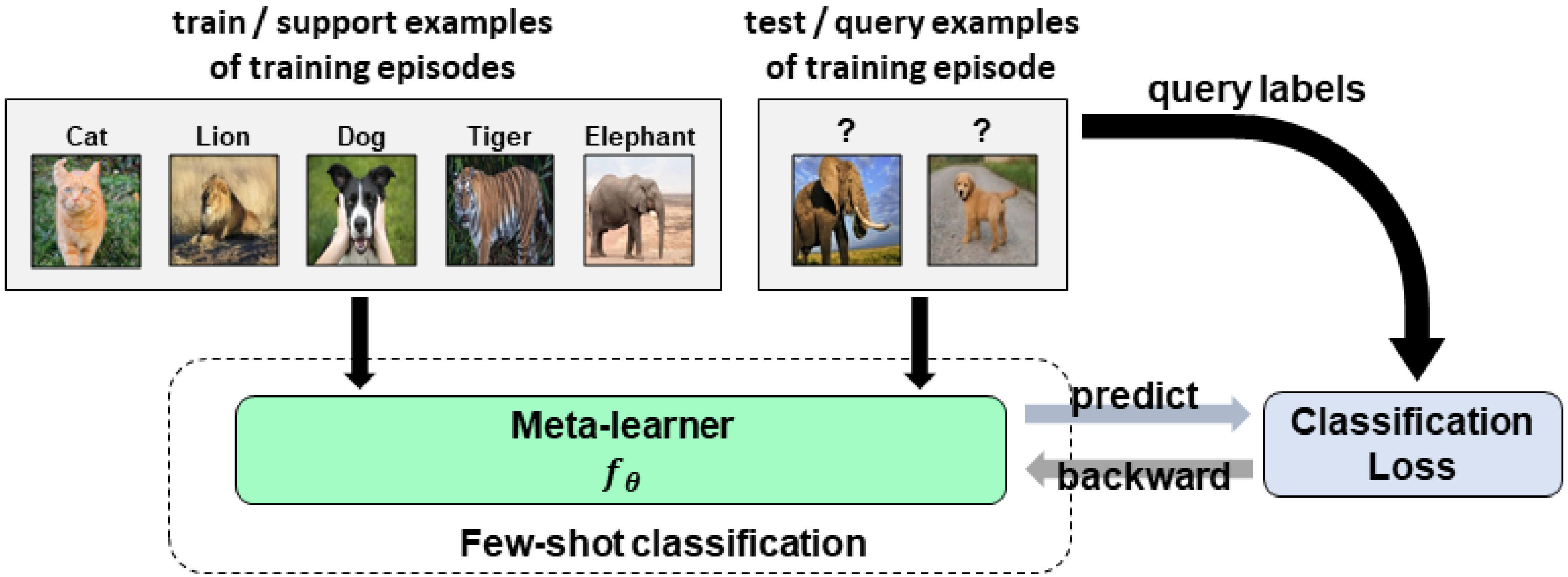


Attention Module



“A Simple Neural Attentive Meta-Learner”, Mishra et al. 18

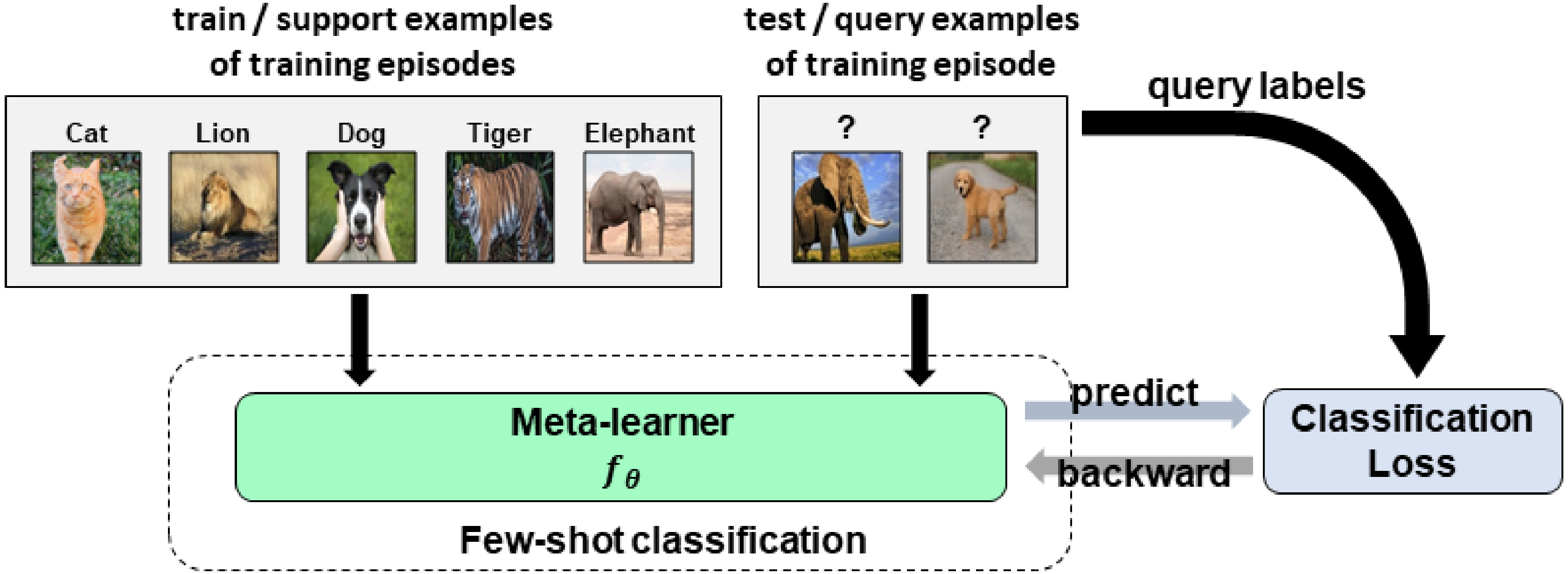
Meta-learning with memory modules



Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(\cdot\,|S), Q) = \sum_m -log(p_m[y_m^Q])$

Meta-learning with memory modules



Meta-training routine:

1. Sample training episode (S, Q)
2. ~~Generate classification model $m_\varphi = f_\theta(S)$~~
3. **Predict classification scores $\{p_m\}_m = f_\theta\left(\{x_m^Q\}_m | S\right)$ for all queries**
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(\cdot\,|S), Q) = \sum_m -log(p_m[y_m^Q])$

store & access support data with a memory module

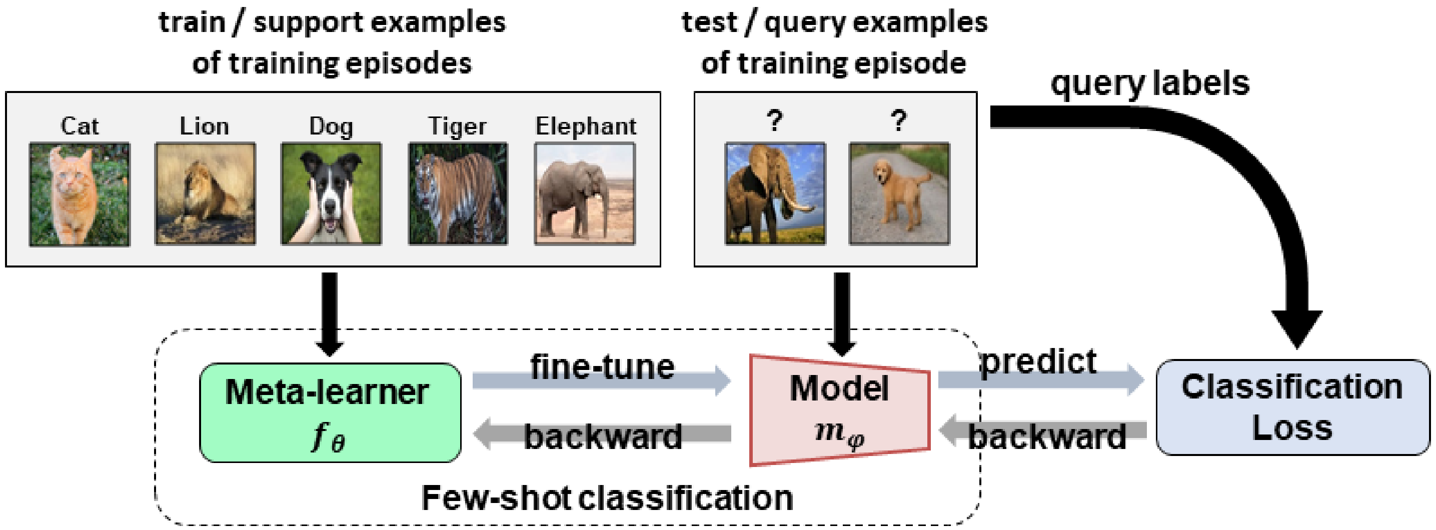
Meta-learning with memory modules

- More generic than metric learning methods
  - applicable to other learning problems: regression, RL, …
- More data hungry (for training the meta-learner)
- More computational expensive

Agenda

- Introduction
- Main types of few-shot learning algorithms
    - Metric learning
    - Meta-learning with memory modules
    - **Optimization based meta-learning**
    - Learn to predict model parameters
- Few-shot learning without forgetting

Optimization-based meta-learning



Key idea: few-shot classification as a parameters optimization problem

- "Optimization as a Model for Few-Shot Learning", Ravi et al. 17

Here we will focus on MAML: "Model-Agnostic Meta-Learning", Finn et al. 17

MAML: Optimization-based meta-learning

Fine-tuning: start from $\boldsymbol{\theta}$ and optimize w.r.t. training loss $L(\theta, S)$ using gradient steps:

$$\boldsymbol{\varphi} \leftarrow \boldsymbol{\theta} - \boldsymbol{\alpha}\, \nabla_{\boldsymbol{\theta}} \boldsymbol{L}(\boldsymbol{\theta}, \boldsymbol{S})$$

(to simplify the description: only the 1st step)

$\boldsymbol{\varphi}$: parameters of novel class model $\boldsymbol{m}_{\boldsymbol{\varphi}}$

Fine-tuning with limited data: **requires "good" pre-trained parameters $\boldsymbol{\theta}$**

MAML: Optimization-based meta-learning

Fine-tuning: start from $\boldsymbol{\theta}$ and optimize w.r.t. training loss $L(\theta, S)$ using gradient steps:

$$\boldsymbol{\varphi} \leftarrow \boldsymbol{\theta} - \boldsymbol{\alpha}\, \boldsymbol{\nabla}_{\boldsymbol{\theta}} \boldsymbol{L}(\boldsymbol{\theta}, \boldsymbol{S})$$

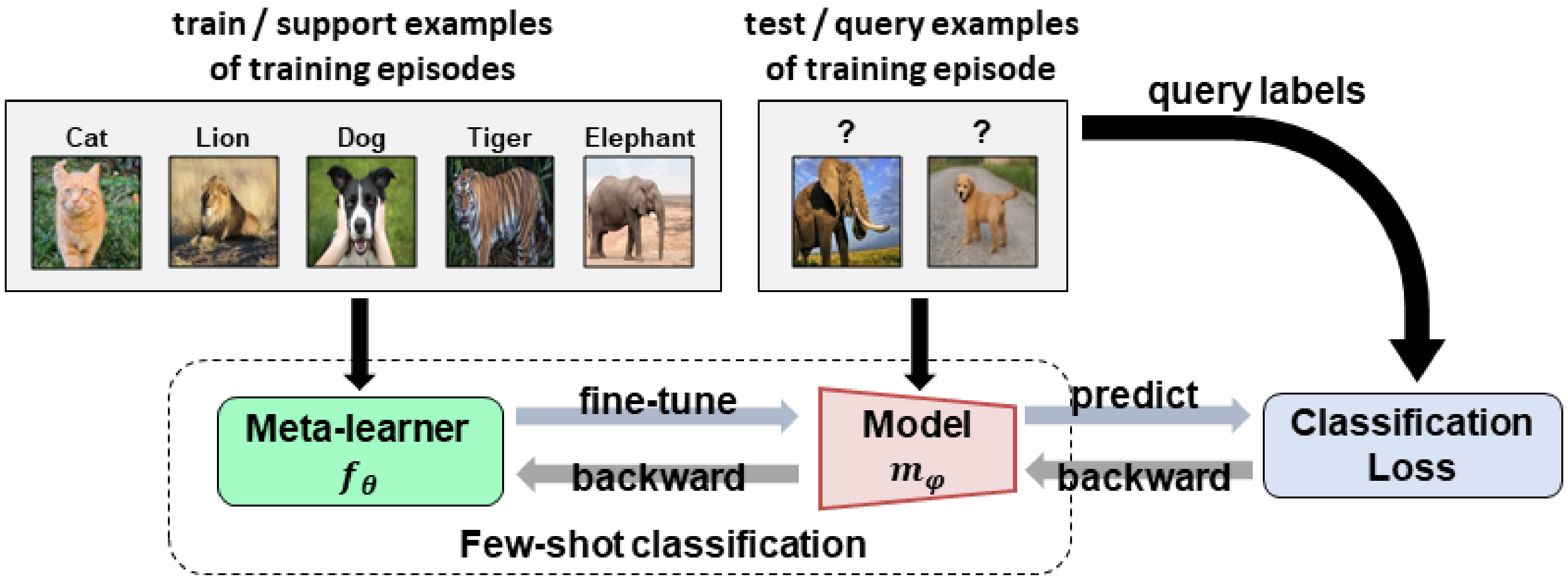
(to simplify the description: only the 1st step)

$\boldsymbol{\varphi}$: parameters of novel class model $\boldsymbol{m}_{\boldsymbol{\varphi}}$

Fine-tuning with limited data: **requires “good” pre-trained parameters $\boldsymbol{\theta}$**

MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

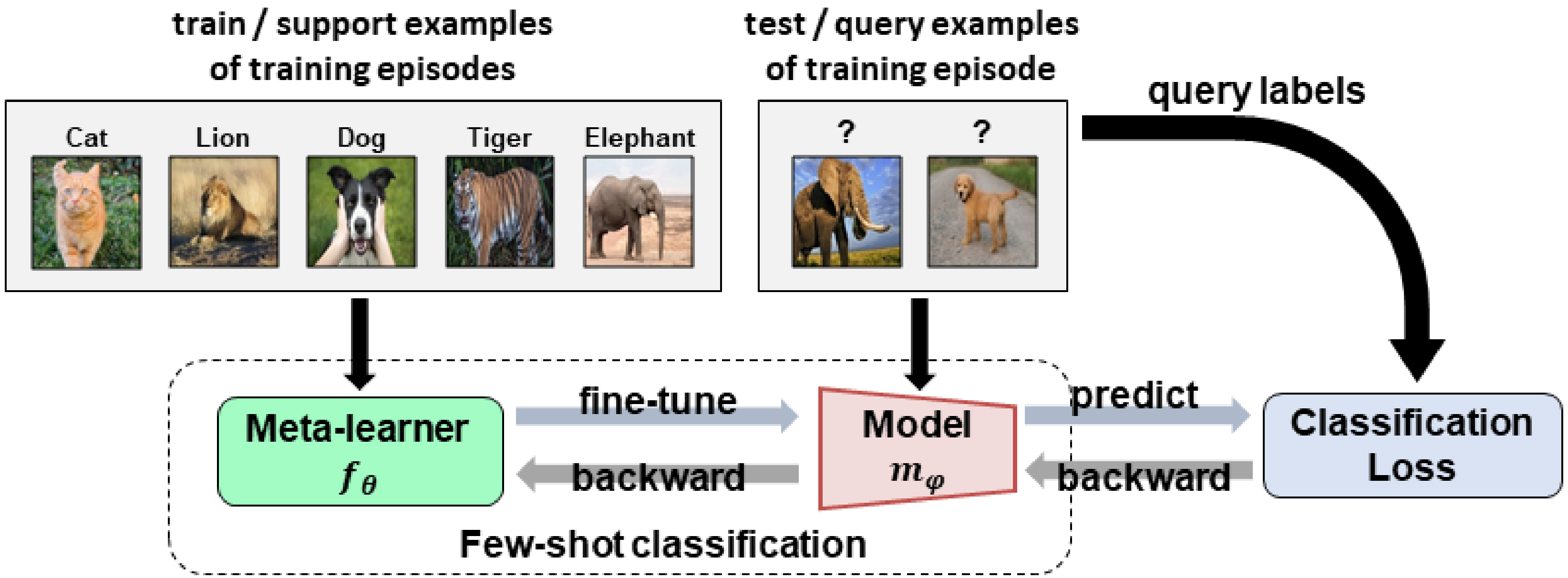
MAML: Optimization-based meta-learning



MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

$$\min_{\theta} \sum_{(S,Q)} L(\theta - \alpha \nabla_{\theta} L(\theta, S), Q)$$

MAML: Optimization-based meta-learning



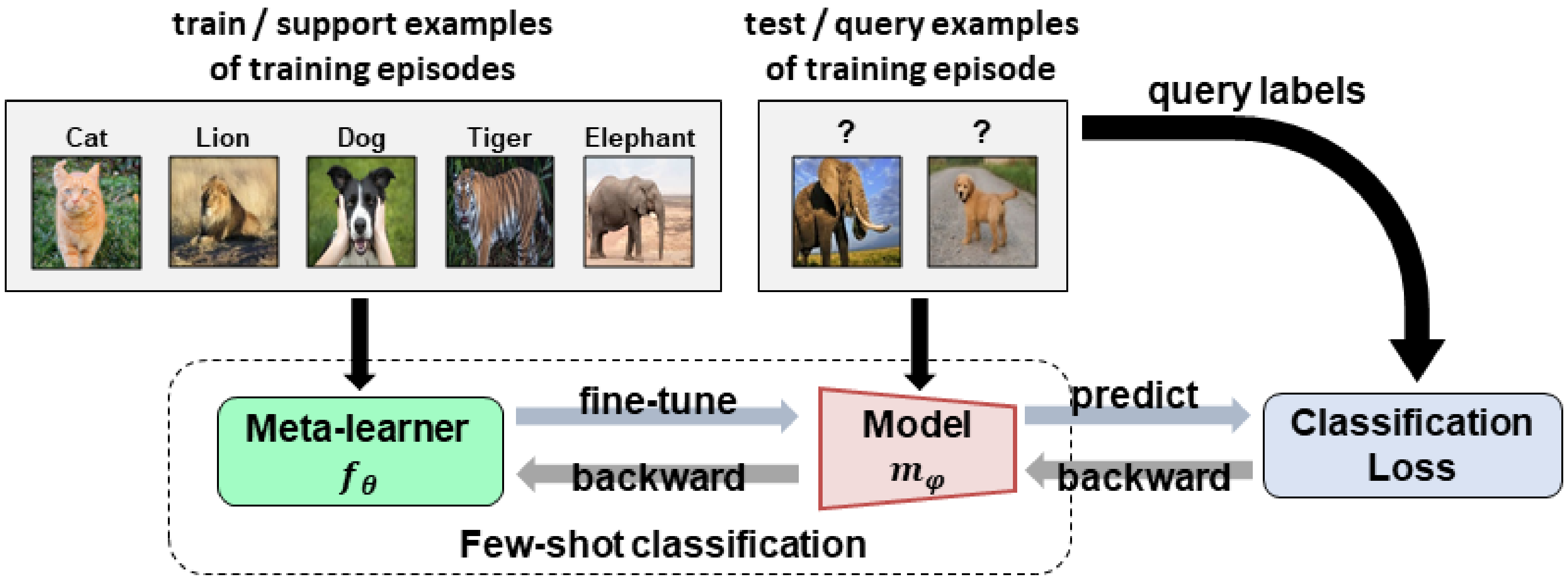
MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

$$\min_{\theta} \sum_{(S,Q)} L(\boldsymbol{\theta} - \boldsymbol{\alpha}\, \boldsymbol{\nabla_\theta L(\theta, S)}, Q)$$

inner optimization:

Fine-tunes θ for the task using the **support data S**

MAML: Optimization-based meta-learning



MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

$$\min_{\theta} \sum_{(S,Q)} L(\boldsymbol{\theta} - \alpha\, \nabla_{\theta} L(\boldsymbol{\theta}, \boldsymbol{S}), \boldsymbol{Q})$$

outer optimization:

minimizes w.r.t. $\boldsymbol{\theta}$ all the task-specific classification losses of **the query data Q**

Meta-training in MAML

Meta-training routine:

1. Sample training episode (S, Q)
2. Inner optimization (fine-tune using train data S): $m_{\varphi=\theta-\alpha\,\nabla_\theta L(\theta,S)}$
3. Predict classification scores $p_m = m_\varphi\left(x_m^Q\right)$ for each query x_m^Q
4. Outer optimization: optimize θ w.r.t. the loss on query data Q:

$$\theta \leftarrow \theta - \beta\,\nabla_\theta L(\theta - \alpha\,\nabla_\theta L(\theta, S), Q)$$

Meta-training in MAML

Meta-training routine:

1. **Sample training episode $(\boldsymbol{S}, \boldsymbol{Q})$**
2. Inner optimization (fine-tune using train data S): $m_{\varphi=\theta-\alpha\,\nabla_\theta L(\theta,S)}$
3. Predict classification scores $p_m = m_\varphi\left(x_m^Q\right)$ for each query x_m^Q
4. Outer optimization: optimize θ w.r.t. the loss on query data Q:

$$\theta \leftarrow \theta - \beta\,\nabla_\theta L(\theta - \alpha\,\nabla_\theta L(\theta, S), Q)$$

Meta-training in MAML

Meta-training routine:

1. Sample training episode (S, Q)
2. **Inner optimization (fine-tune using train data S): $\boldsymbol{m_{\varphi=\theta-\alpha\,\nabla_\theta L(\theta,S)}}$**
3. Predict classification scores $p_m = m_\varphi\left(x_m^Q\right)$ for each query x_m^Q
4. Outer optimization: optimize θ w.r.t. the loss on query data Q:

$$\theta \leftarrow \theta - \beta\,\nabla_\theta L(\theta - \alpha\,\nabla_\theta L(\theta, S), Q)$$

Meta-training in MAML

Meta-training routine:

1. Sample training episode (S, Q)
2. Inner optimization (fine-tune using train data S): $m_{\varphi=\theta-\alpha\,\nabla_\theta L(\theta,S)}$
3. **Predict classification scores $\boldsymbol{p_m = m_\varphi\left(x_m^Q\right)}$ for each query $\boldsymbol{x_m^Q}$**
4. Outer optimization: optimize θ w.r.t. the loss on query data Q:

$$\theta \leftarrow \theta - \beta\,\nabla_\theta L(\theta - \alpha\,\nabla_\theta L(\theta, S), Q)$$

Meta-training in MAML

Meta-training routine:

1. Sample training episode (S, Q)
2. Inner optimization (fine-tune using train data S): $m_{\varphi=\theta-\alpha\,\nabla_\theta L(\theta,S)}$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each query x_m^Q
4. **Outer optimization: optimize $\boldsymbol{\theta}$ w.r.t. the loss on query data $\boldsymbol{Q}$:**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{\beta}\,\boldsymbol{\nabla_\theta L}(\boldsymbol{\theta} - \boldsymbol{\alpha}\,\boldsymbol{\nabla_\theta L}(\boldsymbol{\theta}, \boldsymbol{S}), \boldsymbol{Q})$$

Meta-training in MAML

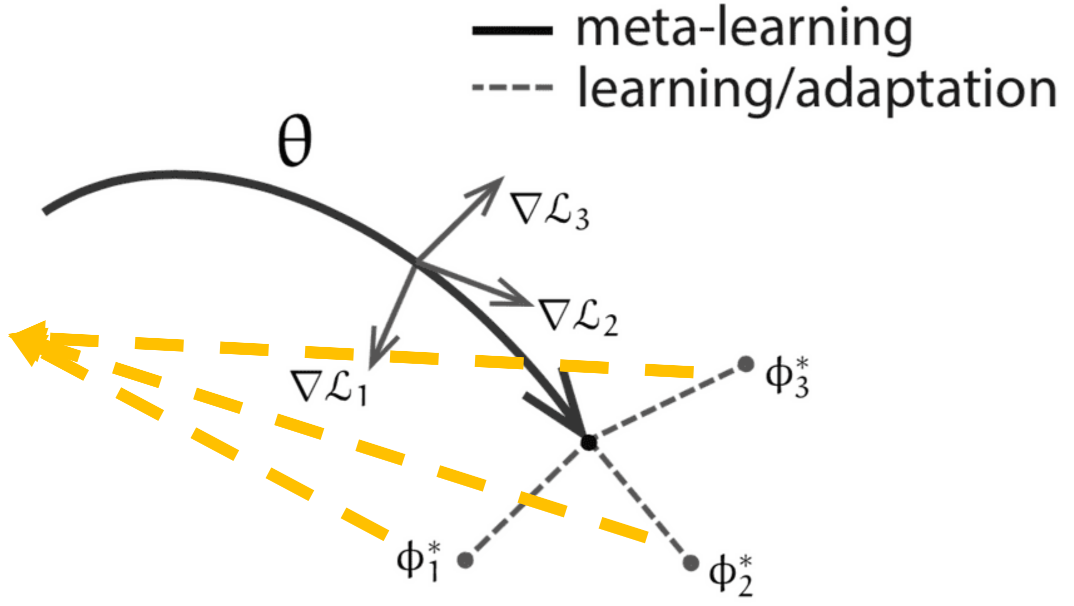
Meta-training routine:

1. Sample training episode (S, Q)
2. Inner optimization (fine-tune using train data S): $m_{\varphi=\theta-\alpha\,\nabla_\theta L(\theta,S)}$
3. Predict classification scores $p_m = m_\varphi(x_m^Q)$ for each query x_m^Q
4. **Outer optimization: optimize $\boldsymbol{\theta}$ w.r.t. the loss on query data $\boldsymbol{Q}$:**

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \boldsymbol{\beta}\,\boldsymbol{\nabla_\theta L}(\boldsymbol{\theta} - \boldsymbol{\alpha}\,\boldsymbol{\nabla_\theta L}(\boldsymbol{\theta}, \boldsymbol{S}), \boldsymbol{Q})$$

back-propagate through gradient descent
➔ 2nd order gradients w.r.t. θ

MAML: Optimization based meta-learning



optimal parameters of training tasks

Figure 1: Illustrative diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

$$\min_{\theta} \sum_{(S,Q)} L(\theta - \alpha \nabla_{\theta} L(\theta, S), Q)$$

MAML: Optimization based meta-learning

before meta-training

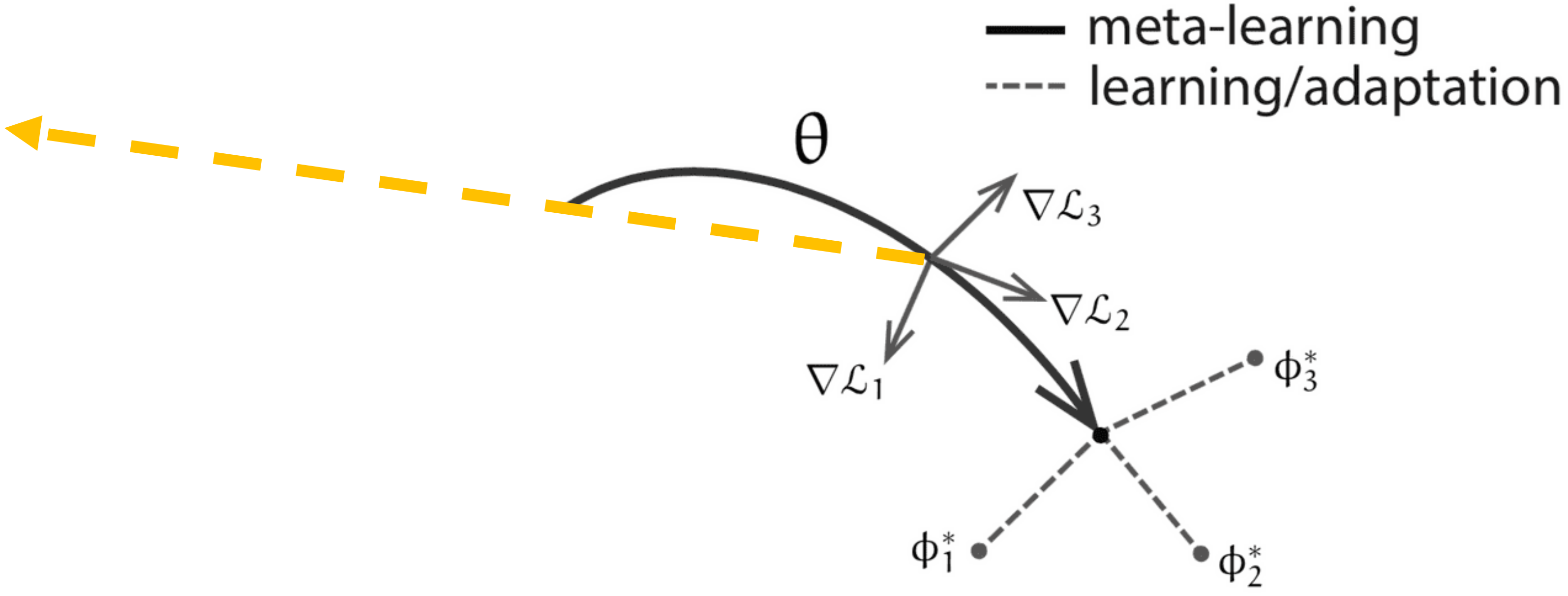


Figure 1: Illustrative diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

$$\min_{\theta} \sum_{(S,Q)} L(\theta - \alpha \nabla_\theta L(\theta, S), Q)$$

MAML: Optimization based meta-learning

before meta-training

after meta-training:
- $\boldsymbol{\theta}$ closer to optimal parameters
- good for fine-tuning to them

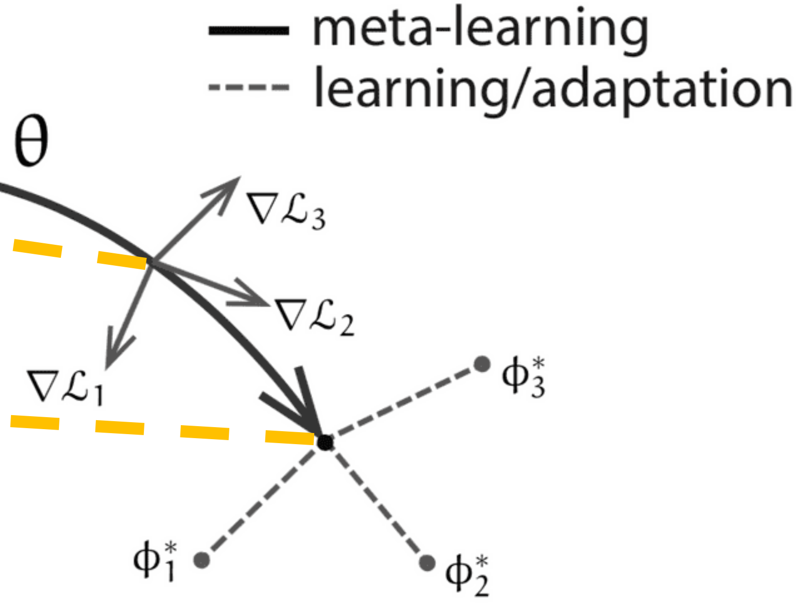


Figure 1: Illustrative diagram of our model-agnostic meta-learning algorithm (MAML), which optimizes for a representation θ that can quickly adapt to new tasks.

MAML: meta-learn $\boldsymbol{\theta}$ so that it transfers well via fine-tuning

$$\min_{\boldsymbol{\theta}} \sum_{(\boldsymbol{S},\boldsymbol{Q})} L(\boldsymbol{\theta} - \alpha\, \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{S}), \boldsymbol{Q})$$

MAML: Optimization based meta-learning

Omniglot (Lake et al., 2011)	5-way Accuracy 1-shot	5-way Accuracy 5-shot	20-way Accuracy 1-shot	20-way Accuracy 5-shot
MANN, no conv (Santoro et al., 2016)	82.8%	94.9%	–	–
MAML, no conv (ours)	**89.7 $\pm$ 1.1%**	**97.5 $\pm$ 0.6%**	–	–
Siamese nets (Koch, 2015)	97.3%	98.4%	88.2%	97.0%
matching nets (Vinyals et al., 2016)	98.1%	98.9%	93.8%	98.5%
neural statistician (Edwards & Storkey, 2017)	98.1%	99.5%	93.2%	98.1%
memory mod. (Kaiser et al., 2017)	98.4%	99.6%	95.0%	98.6%
MAML (ours)	**98.7 $\pm$ 0.4%**	**99.9 $\pm$ 0.1%**	**95.8 $\pm$ 0.3%**	**98.9 $\pm$ 0.2%**

MiniImagenet (Ravi & Larochelle, 2017)	5-way Accuracy 1-shot	5-way Accuracy 5-shot
fine-tuning baseline	28.86 $\pm$ 0.54%	49.79 $\pm$ 0.79%
nearest neighbor baseline	41.08 $\pm$ 0.70%	51.04 $\pm$ 0.65%
matching nets (Vinyals et al., 2016)	43.56 $\pm$ 0.84%	55.31 $\pm$ 0.73%
meta-learner LSTM (Ravi & Larochelle, 2017)	43.44 $\pm$ 0.77%	60.60 $\pm$ 0.71%
MAML, first order approx. (ours)	**48.07 $\pm$ 1.75%**	**63.15 $\pm$ 0.91%**
MAML (ours)	**48.70 $\pm$ 1.84%**	**63.11 $\pm$ 0.92%**

MAML: Optimization based meta-learning

- Consistent with the standard fine-tuning procedure
- Model-agnostic: can accommodate any network architecture
- Applicable to other problems: regression, RL, …

- 2^{nd} order gradients: computationally and memory expensive
- Difficult to train large models
- Need to train a different meta-learner for each N (classes) and K (shots)

Optimization based meta-learning

- "Optimization as a Model for Few-Shot Learning", Ravi et al. 17
  - **Learns the gradient descent step with an LSTM**
  - Actually precedes MAML

- **MAML with only 1st order derivatives for meta-learning $\boldsymbol{\theta}$**
  - "Model-Agnostic Meta-Learning", Finn et al. 17
  - "On first-order meta-learning algorithms", Nichol et al.18

- "Meta-SGD: Learning to quickly learn for few-shot learning", Li et al. 17
- "Meta-learning with implicit gradients", Rajeswaran et al. 19
- "Meta-learning with warped gradient descent", Flennerhag et al. 20

- **Optimize low-dimensional latent task embedding (hybrid method):**
  - "Meta-learning with latent embedding optimization", Rusu et al. 19

- **Meta-learning with closed-form / convex solvers (for output-classification layer):**
  - ridge/logistic regression: "Meta-learning with differentiable closed-form solvers", Bertinetto et al. 19
  - support vector machine: "Meta-learning with differentiable convex optimization", Lee et al. 19

Meta-learning with differentiable convex optimization

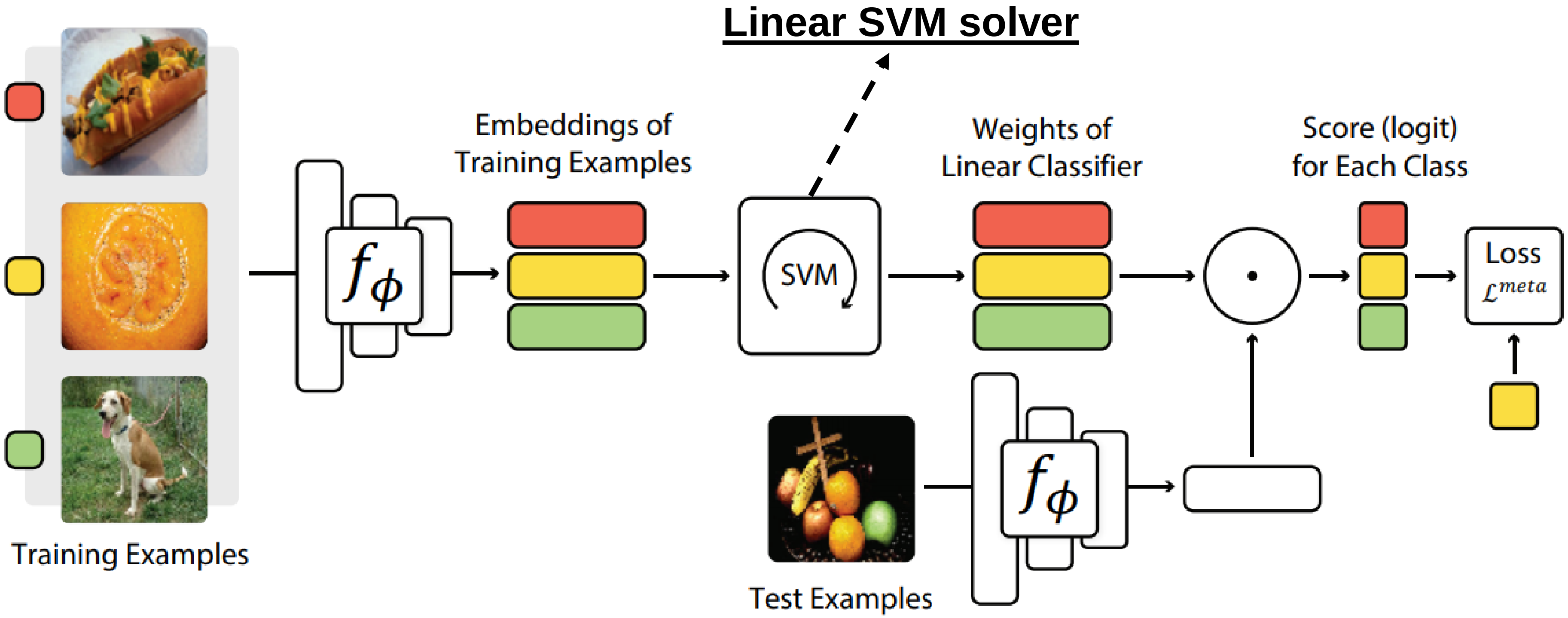
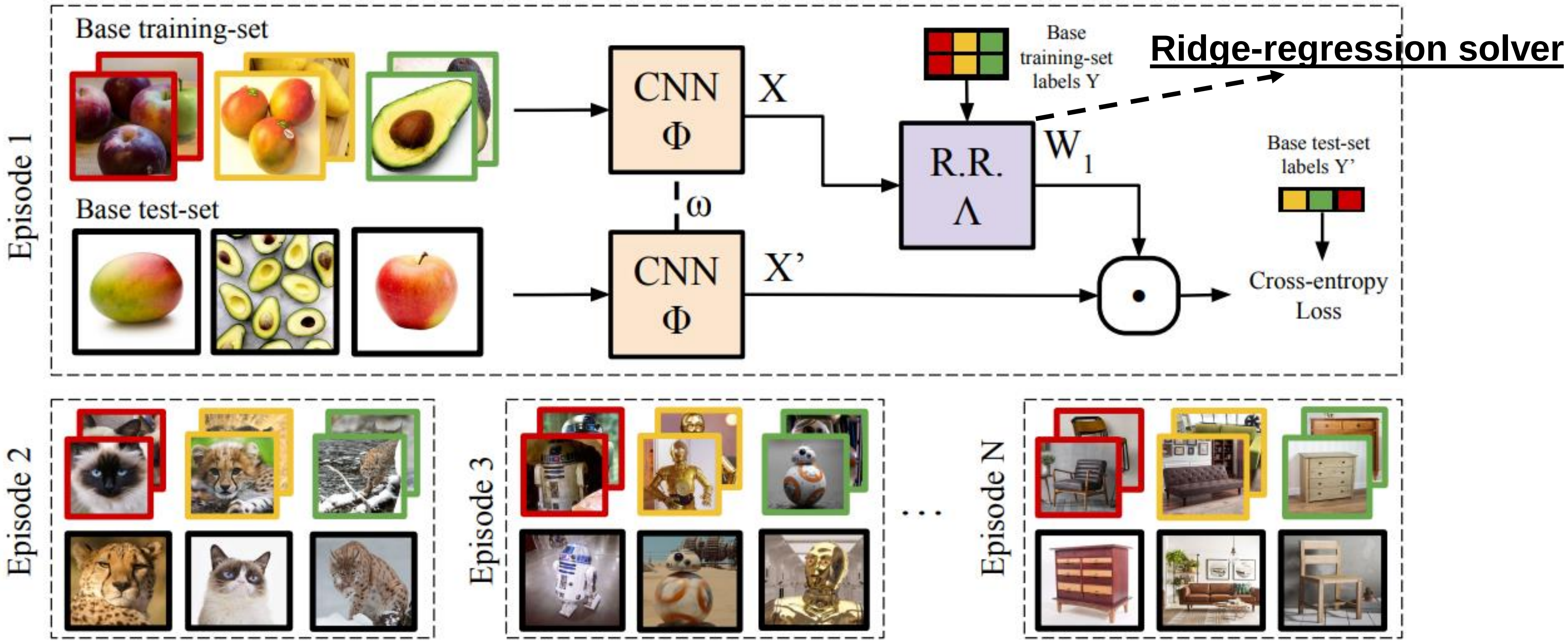


Figure 1. **Overview of our approach.** Schematic illustration of our method MetaOptNet on an 1-shot 3-way classification task. The meta-training objective is to learn the parameters ϕ of a feature embedding model f_ϕ that generalizes well across tasks when used with regularized linear classifiers (*e.g.*, SVMs). A task is a tuple of a few-shot training set and a test set (see Section 3 for details).

Key idea: meta-learn good features for SVM linear classifiers

“Meta-learning with differentiable convex optimization”, Lee et al. 19

Meta-learning with differentiable closed-form solvers



Key idea: meta-learn good features for closed-form solvers for the output layer of the classification network

- Ridge-regression or logistic regression

“Meta-learning with differentiable closed-form solvers”, Bertinetto et al. 19

Agenda

- Introduction
- Main types of few-shot learning algorithms
  - Metric learning
  - Meta-learning with memory modules
  - Optimization based meta-learning
  - **Learn to predict model parameters**
- Few-shot learning without forgetting

Learn to predict model parameters

Key idea: train the meta-learner to predict task-specific model parameters

Usually, a **small subset of model parameters:**

- Predict diagonal of factorized weights:
  - “Learning feed-forward one-shot learners”, Bertinetto et al.16
- Predict weights of classification head
  - “Learning to model the tail”, Wang et al. 17

Learn to predict model parameters

Key idea: train the meta-learner to predict task-specific model parameters

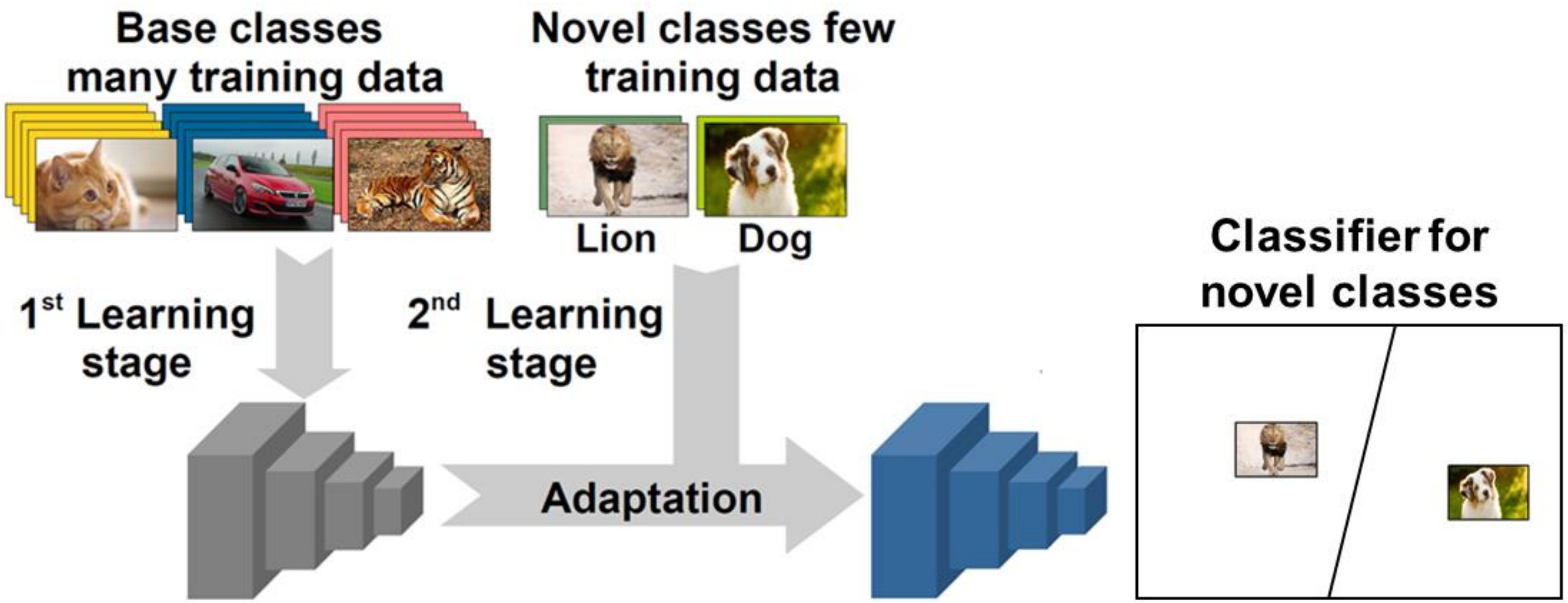
Here focus:

- **predicting the weights of the classification head**
- in the context of the **“few-shot learning without forgetting”** problem

Agenda

- Introduction
- Main types of few-shot learning algorithms
    - Metric learning
    - Meta-learning with memory modules
    - Optimization based meta-learning
    - Learn to predict model parameters
- **Few-shot learning without forgetting**

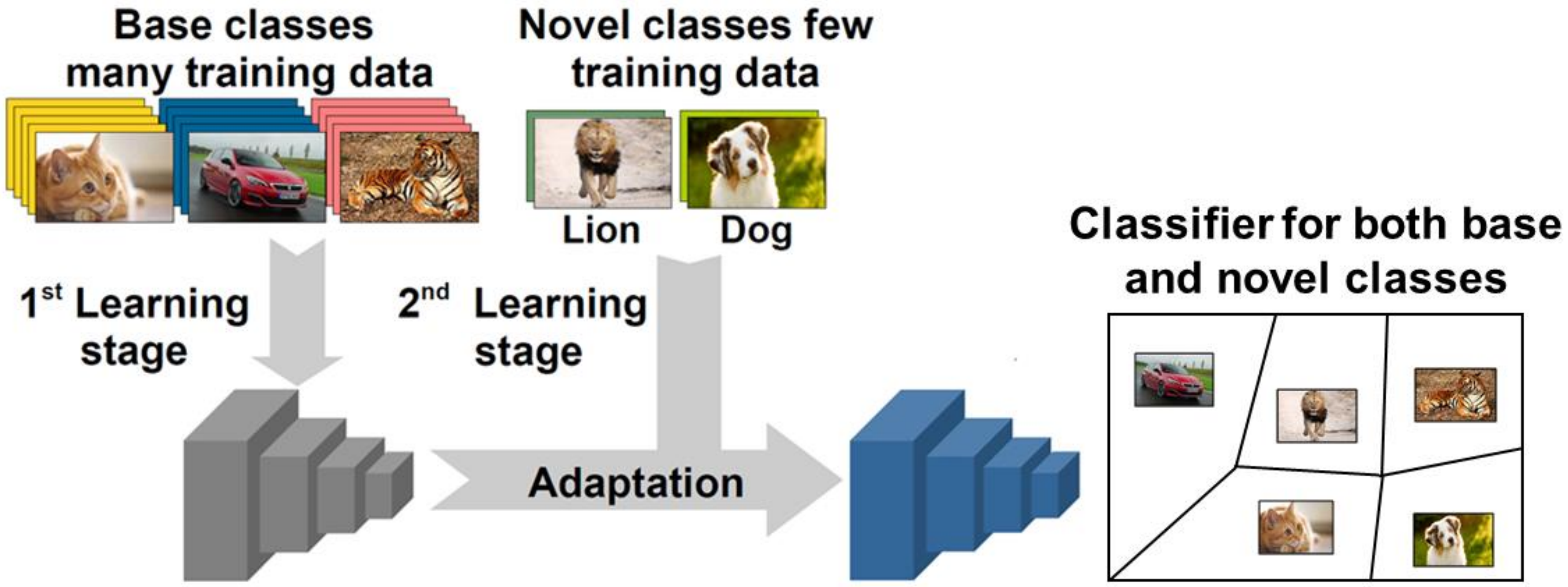
Few-shot learning without forgetting



Typical few-shot models:

- focus on learning novel classes with limited data
- but **“forget” the initial base classes** ☹
  - **“forget”:** worse than base class models or unable to recognize base classes

Few-shot learning without forgetting



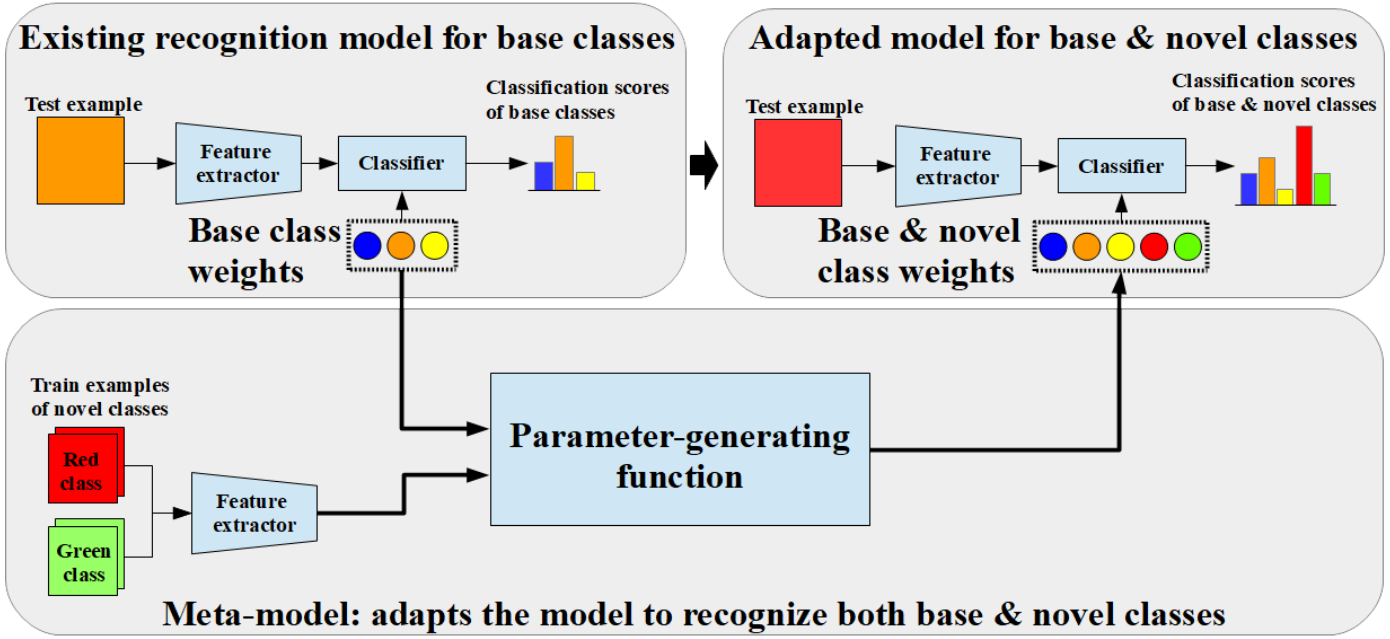
- In contrast, **practical applications** often require**:**
  - to **extend base classes with novels ones** using few training data
  - and **without re-training** on the full dataset (base+novel)
- **“Few-shot learning without forgetting”** targets this problem
  - combines elements from both incremental and few-shot learning

Agenda

- Introduction
- Main types of few-shot learning algorithms
  - Metric learning
  - Meta-learning with memory modules
  - Optimization-based meta-learning
  - **Learn to predict classification weights**
- **Few-shot learning without forgetting**

The description of the "Learn to predict classification weights" methods is in the context of the "few-shot learning without forgetting" setting.

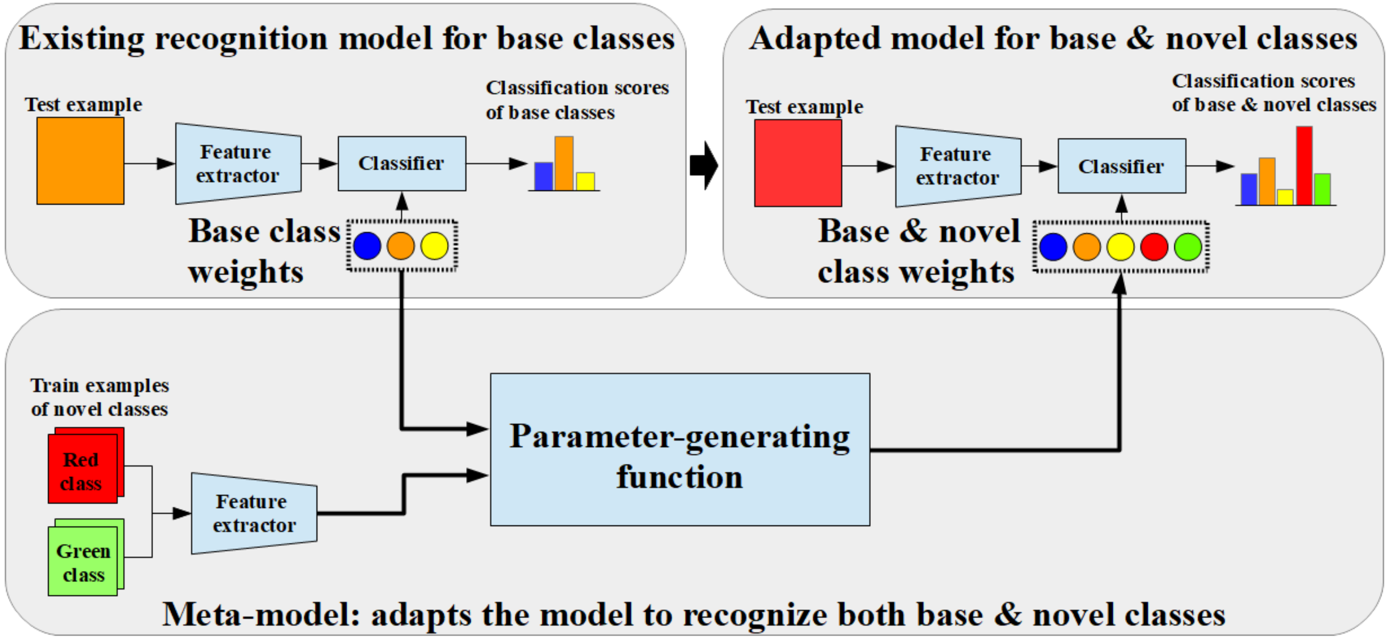
Learn to generate classification weights



- **Pre-trained network:** feature extractor + cosine classification head
- **Extend with parameter-generating function:**
  - **outputs:** new weights for the novel classes

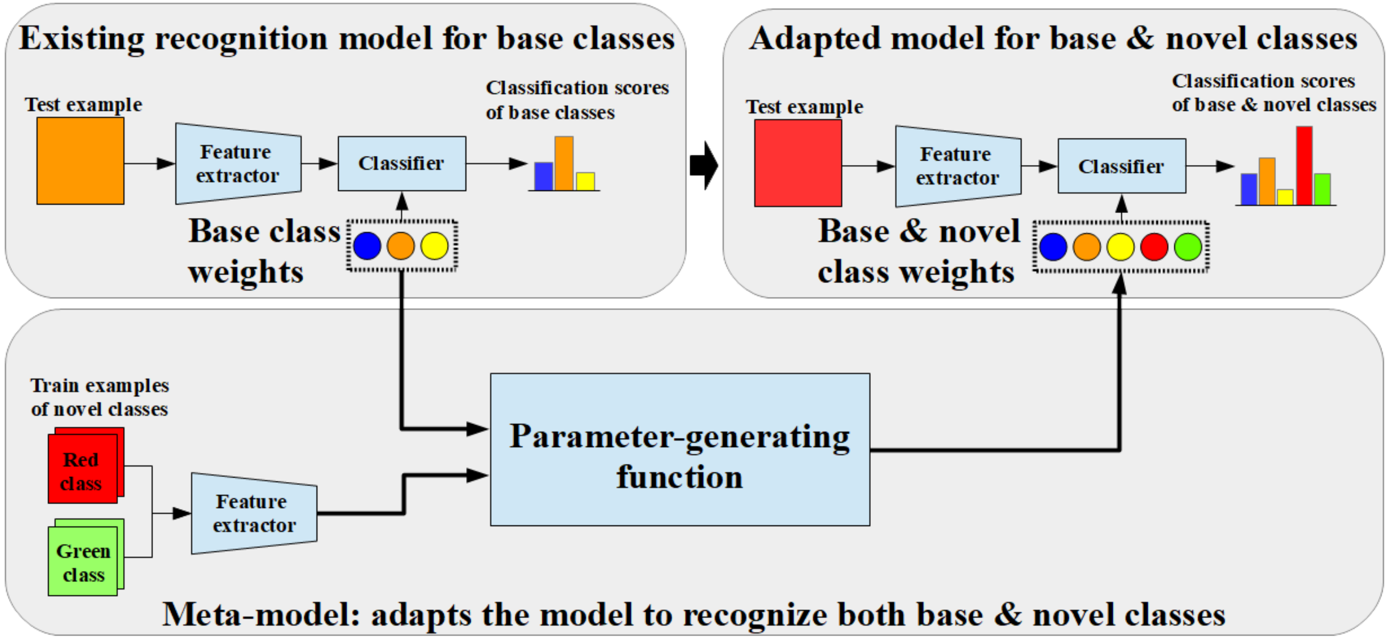
"Dynamic Few-Shot Visual Learning without Forgetting", Gidaris et al. 18
"Low-Shot Learning with Imprinted Weights", Qi et al. 18
"Few-Shot Image Recognition by Predicting Parameters from Activations", Qiao et al. 18
"Learning to model the tail", Wang et al. 17

Learn to generate classification weights



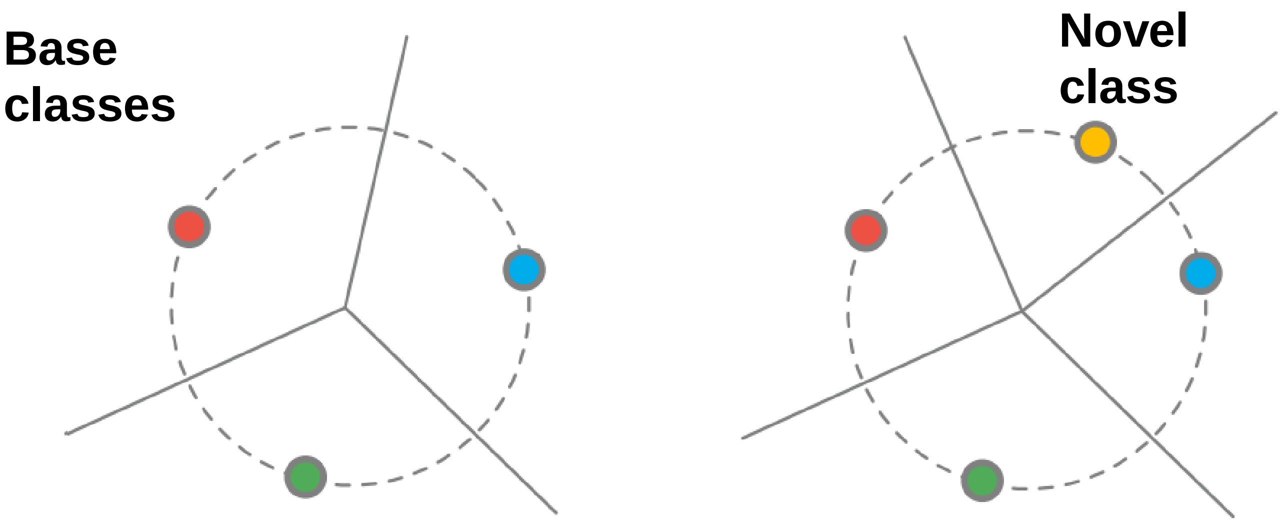
- **Important to use cosine classification head:**
  - L_2**-normalize weights**: all classes have same L_2 norms
    - **avoids class imbalance:** biasing towards classes with bigger L_2norms
  - Easier to add novel weights ➔ unified recognition of both type of classes

Learn to generate classification weights



- **Important to use cosine classification head:**
- Beneficial in the traditional incremental learning setting as well:
  - “Learning a unified classifier incrementally via rebalancing”, Hou et al. 19
  - “Memory efficient incremental through feature adaptation” Iscen et al. 20

Learn to generate classification weights



(a) (b)

Figure 2. Illustration of imprinting in the normalized embedding space. (a) Before imprinting, the decision boundaries are determined by the trained weights. (b) With imprinting, the embedding of an example (the yellow point) from a novel class defines a new region.

Source: “Low-Shot Learning with Imprinted Weights”, Qi et al. 18

Meta-training in few-shot learning without forgetting

Meta-training routine:

1. Sample training episode $(\boldsymbol{S}, \boldsymbol{Q})$
2. Generate classification model $\boldsymbol{m}_{\boldsymbol{\varphi}} = \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{S})$
3. Classification scores $p_m = m_\varphi(x_m^Q)$
4. Optimize $\boldsymbol{\theta}$ w.r.t. the query classification loss $\boldsymbol{L}(\boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{S}), \boldsymbol{Q}) = \sum_{\boldsymbol{m}} -\boldsymbol{log}(\boldsymbol{p}_{\boldsymbol{m}}[\boldsymbol{y}_{\boldsymbol{m}}^{\boldsymbol{Q}}])$

Meta-training in few-shot learning without forgetting

incremental few-shot episode:

- randomly choose some base classes as “fake” novel
- S: examples from the “fake” novel classes
- Q: examples form both “fake” novel and remaining base

Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Classification scores $p_m = m_\varphi(x_m^Q)$
4. Optimize θ w.r.t. the query classification loss, e.g.: $L(f_\theta(S), Q) = \sum_m -log(p_m[y_m^Q])$

Meta-training in few-shot learning without forgetting

incremental few-shot episode

ignore pre-trained base classification weights of classes used as “fake” novel

Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_{\varphi} = f_{\theta}(S)$
3. Classification scores $p_m = m_{\varphi}(x_m^Q)$
4. Optimize θ w.r.t. the query classification loss $L(f_{\theta}(S), Q) = \sum_m -log(p_m[y_m^Q])$

Meta-training in few-shot learning without forgetting

incremental few-shot episode

ignore pre-trained base classification weights of classes used as "fake" novel

Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_\varphi = f_\theta(S)$
3. Classification scores $\boldsymbol{p_m = m_\varphi(x_m^Q)}$ **for both "fake" novel and base classes**
4. Optimize θ w.r.t. the query classification loss $L(f_\theta(S), Q) = \sum_m -log(p_m[y_m^Q])$

Meta-training in few-shot learning without forgetting

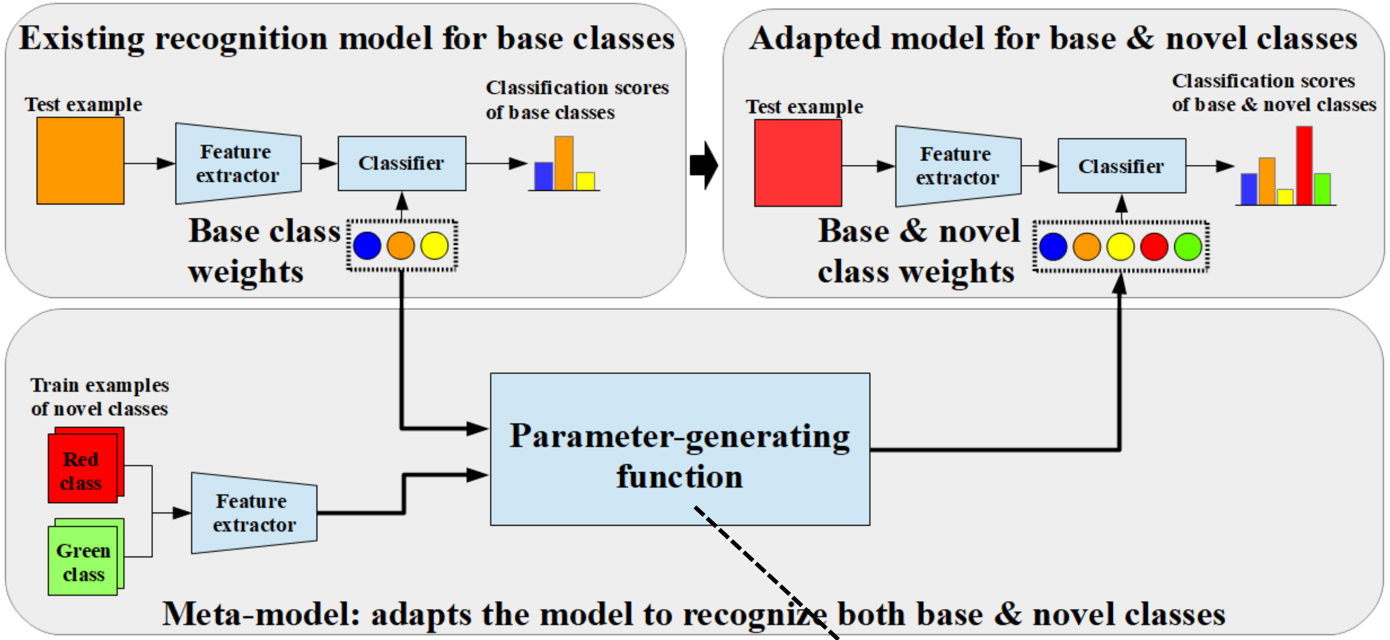
incremental few-shot episode

ignore pre-trained base classification weights of classes used as "fake" novel

Meta-training routine:

1. Sample training episode (S, Q)
2. Generate classification model $m_{\varphi} = f_{\theta}(S)$
3. Classification scores $\boldsymbol{p_m} = \boldsymbol{m_{\varphi}}(\boldsymbol{x_m^Q})$ **for both "fake" novel and base classes**
4. Optimize θ w.r.t. the query classification loss $L(f_{\theta}(S), Q) = \sum_m -log(p_m[y_m^Q])$

Generate weights with prototypical feature averaging



Simplest case:

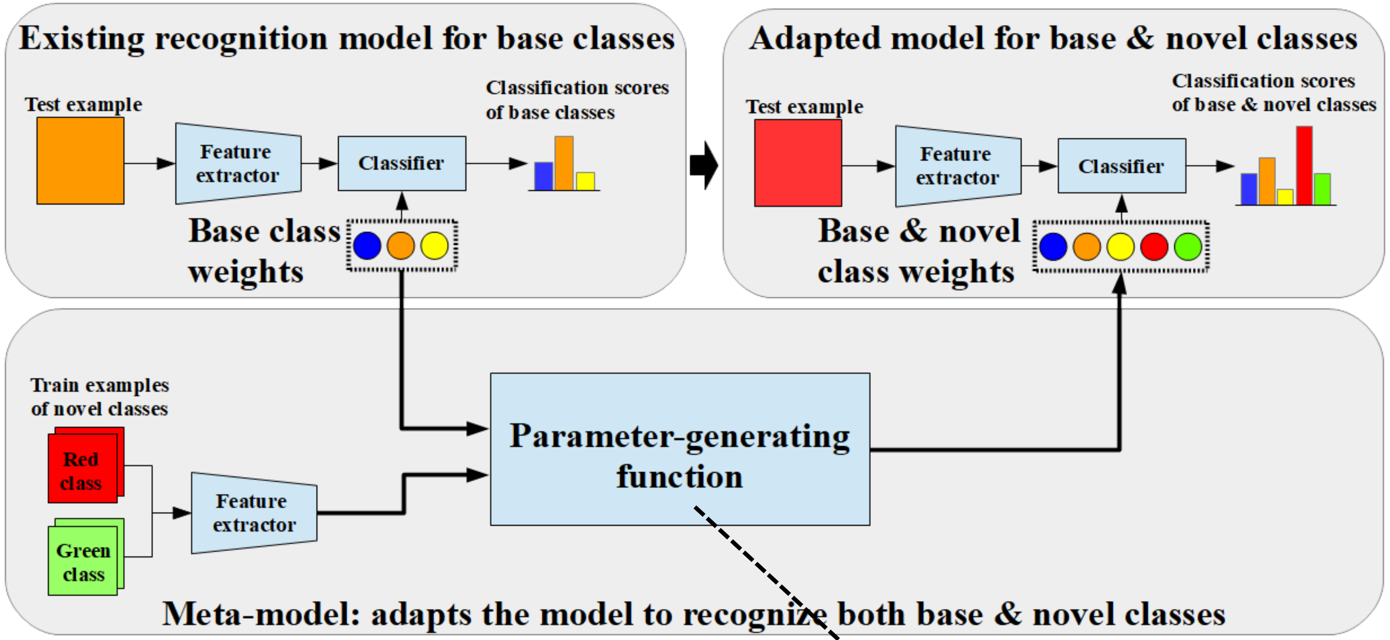
- $\boldsymbol{S_i}$: support set of i-th novel class
- **novel weight** = average feature vector of $\boldsymbol{S_i}$:

$$w_i^{avg} = \frac{1}{|S_i|} \sum_{(x_k^S, y_k^S) \in S_i} F_\theta\left(x_k^S\right)$$

“Low-Shot Learning with Imprinted Weights”, Qi et al. 18

“Dynamic Few-Shot Visual Learning without Forgetting”, Gidaris et al. 18

Generate weights with prototypical feature averaging



Simplest case:

- $\boldsymbol{S_i}$: support set of i-th novel class
- **novel weight** = average feature vector of $\boldsymbol{S_i}$:

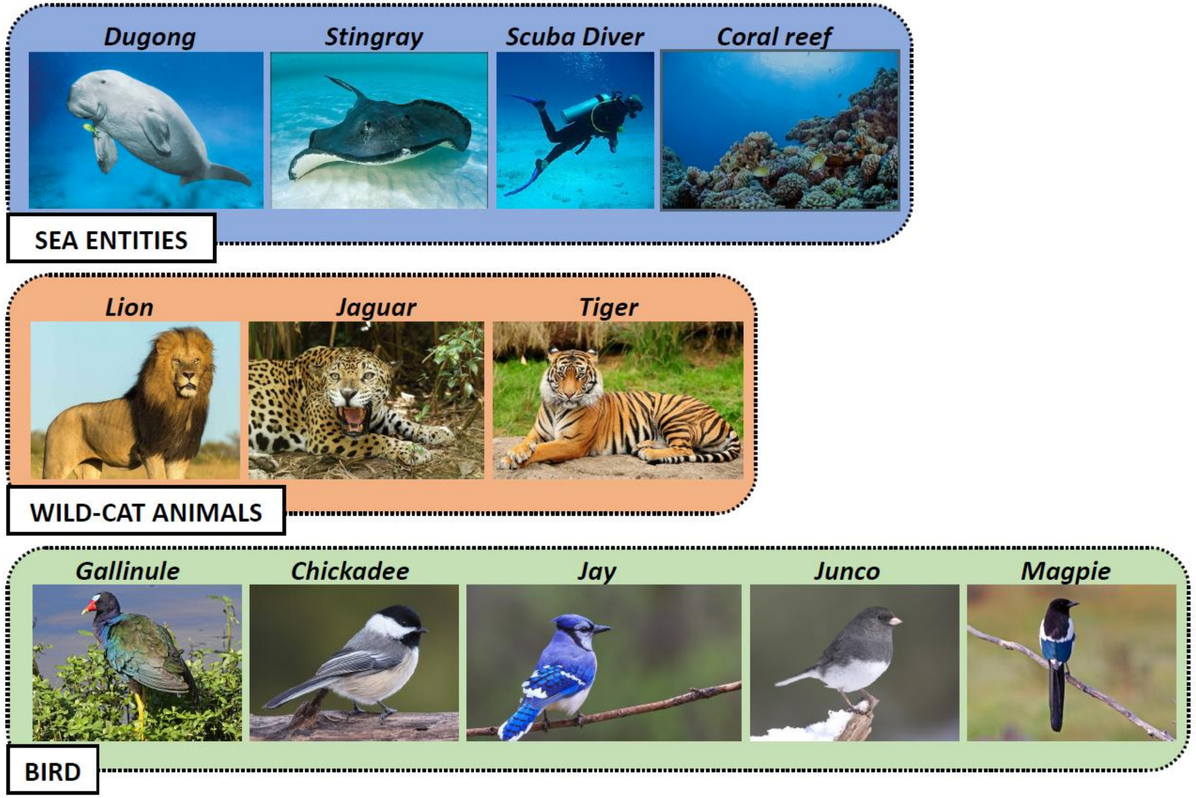
No meta-learning here

$$w_i^{avg} = \frac{1}{|S_i|} \sum_{(x_k^S,\, y_k^S) \in S_i} F_\theta \left(x_k^S\right)$$

"Low-Shot Learning with Imprinted Weights", Qi et al. 18

"Dynamic Few-Shot Visual Learning without Forgetting", Gidaris et al. 18

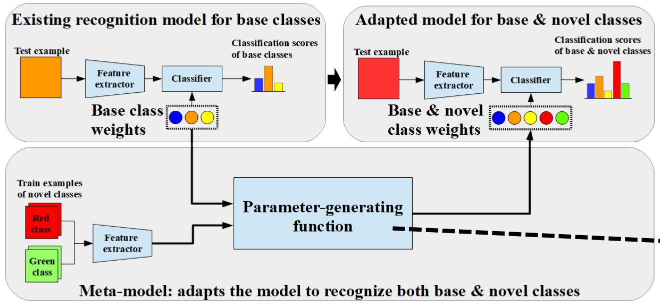
Correlations between classification weights

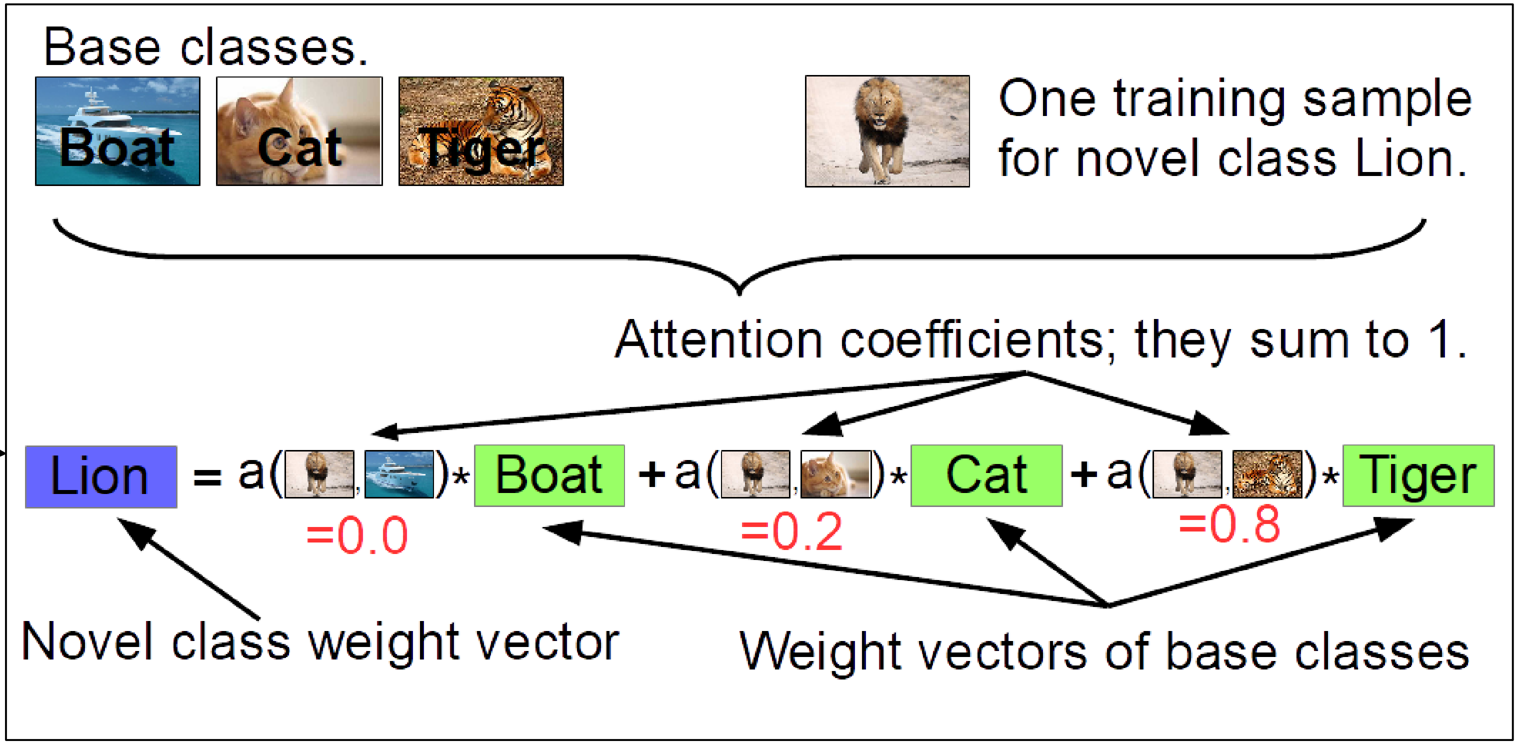


Many classes are semantically / visually related:

- **Correlations between their classification weights**
- Exploit those correlations for generating novel class weights?

Generate classification weights with attention module





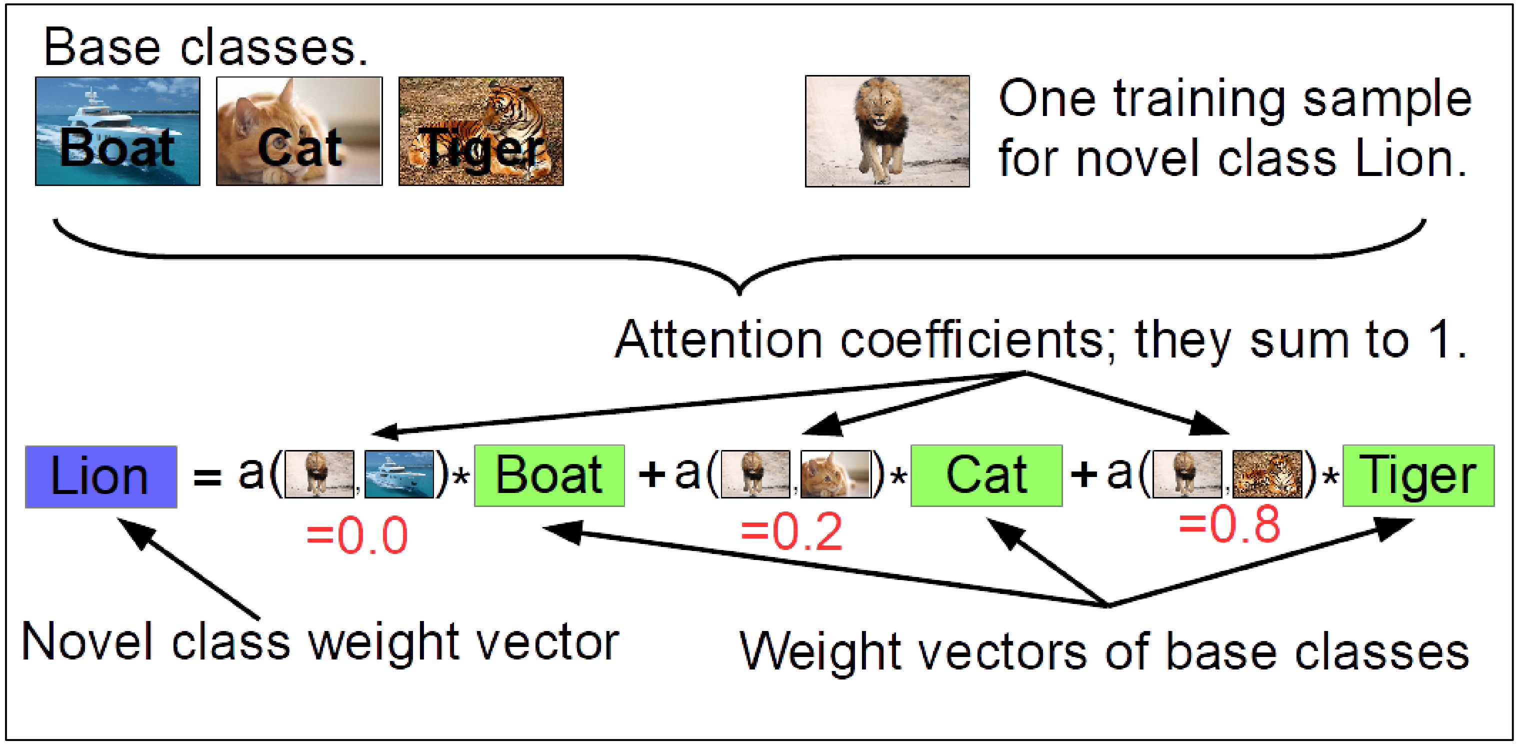
- **Novel weight using attention over base weights w_b:**

$$w_i^{att} = \sum_{b=1}^{N_b} a(S_i)[b] \cdot w_b$$

- N_b: number of base classes

“Dynamic Few-Shot Visual Learning without Forgetting”, Gidaris et al. 18

Generate classification weights with attention module



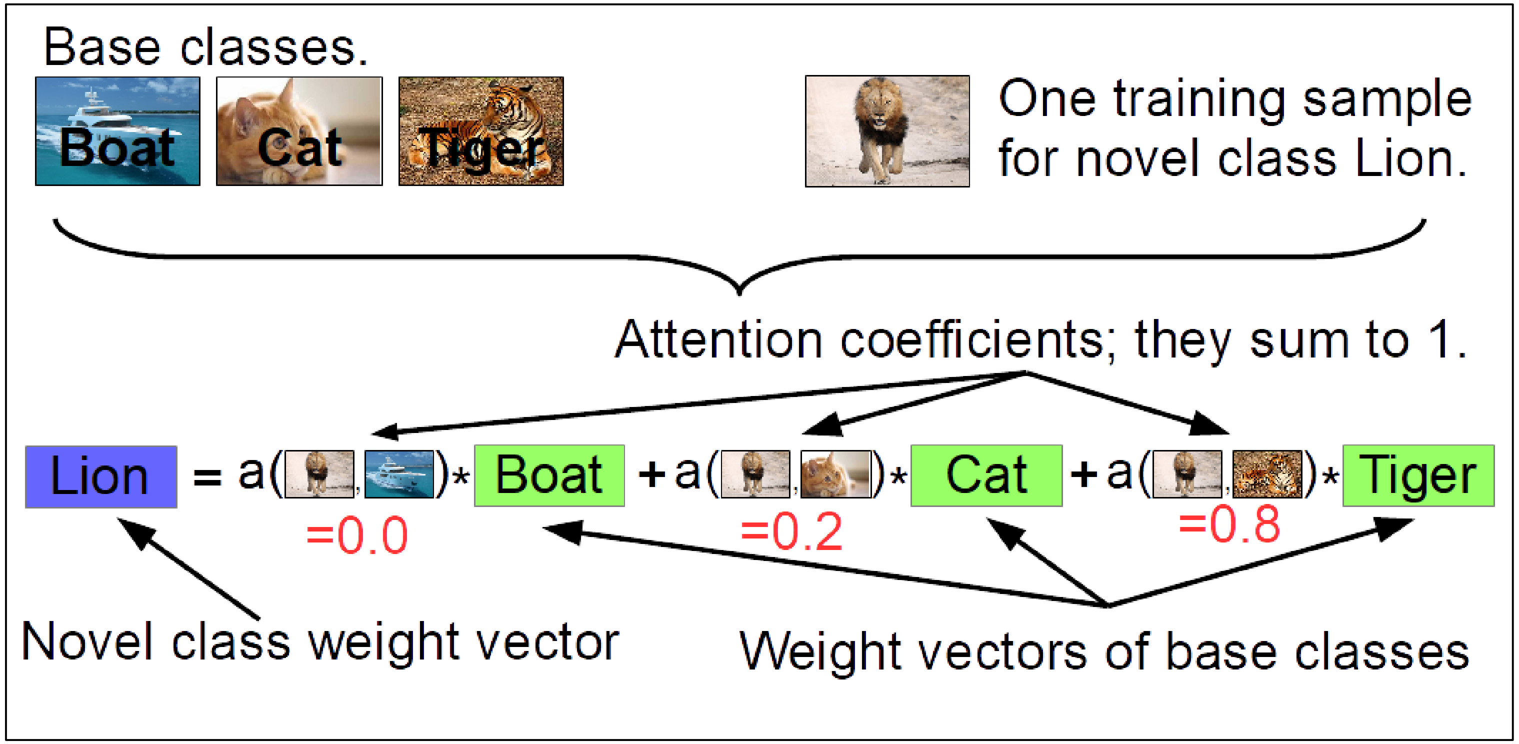
- **Novel weight using attention over base weights $\boldsymbol{w_b}$:**

$$w_i^{att} = \sum_{b=1}^{N_b} a(S_i)[b] \cdot w_b$$

- $\boldsymbol{N_b}$**:** number of base classes

“Dynamic Few-Shot Visual Learning without Forgetting”, S. Gidaris et al. 18

Generate classification weights with attention module



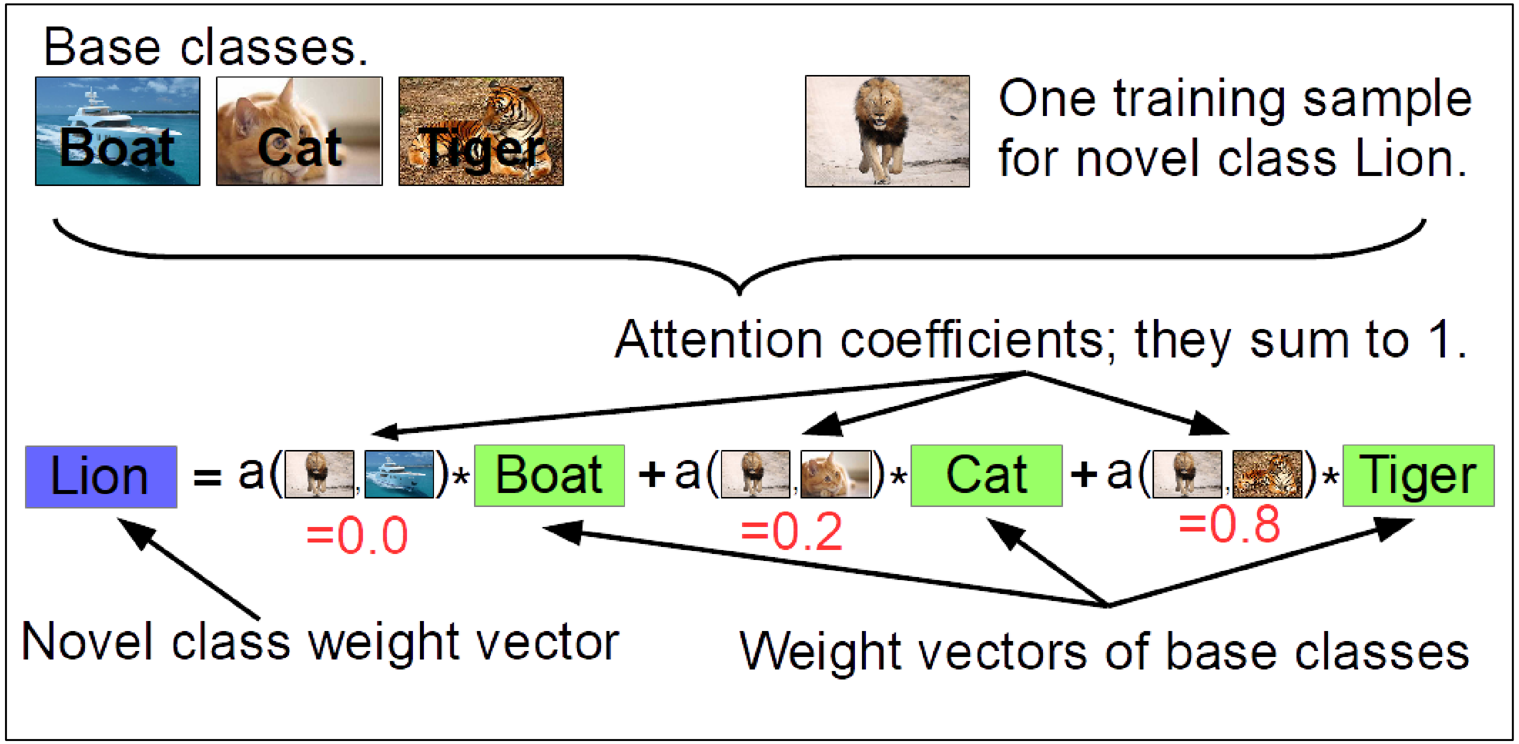
- **Novel weight using attention over base weights $\boldsymbol{w_b}$:**

$$w_i^{att} = \sum_{b=1}^{N_b} a(S_i)[b] \cdot w_b$$

- $\boldsymbol{a(S_i)[b]}$**:** average similarity of support features with base class weight w_b
  - Computed with cosine + softmax

"Dynamic Few-Shot Visual Learning without Forgetting", Gidaris et al. 18

Generate classification weights with attention module



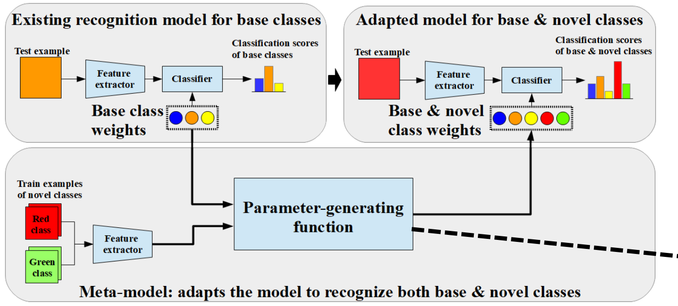
- **Novel weight using attention over base weights $\boldsymbol{w_b}$:**

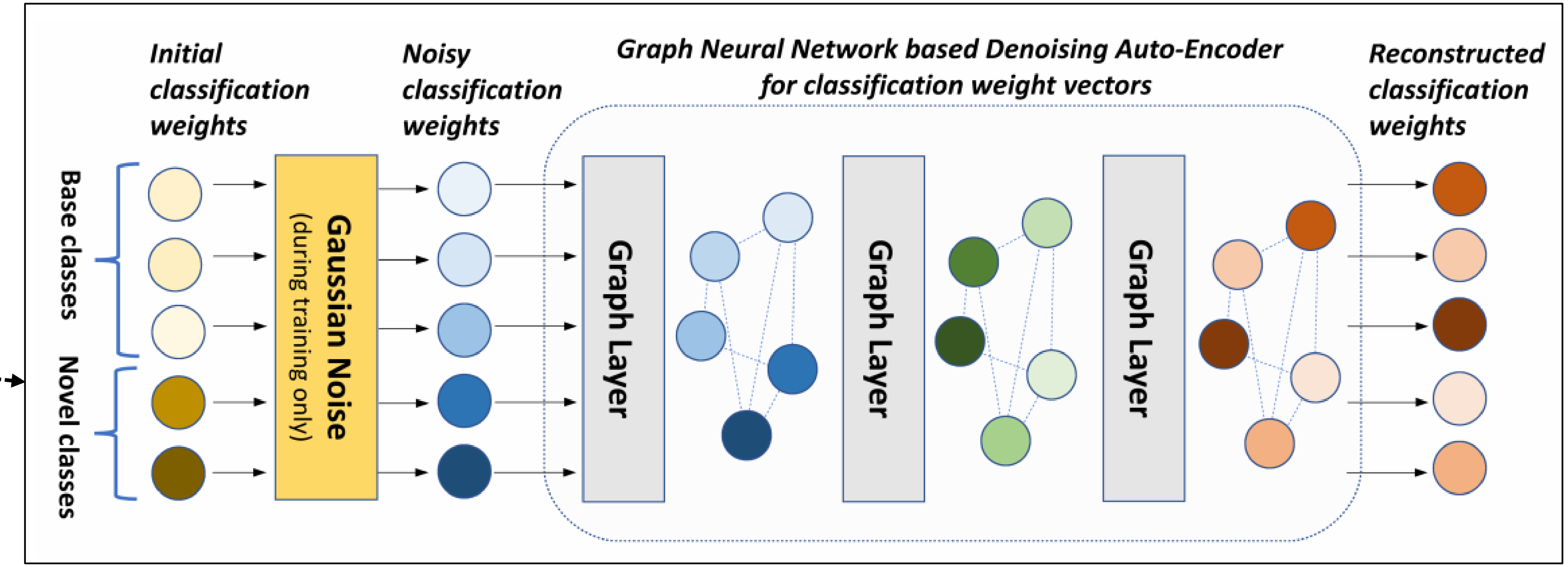
$$w_i^{att} = \sum_{b=1}^{N_b} a(S_i)[b] \cdot w_b$$

- Final novel weight: $\boldsymbol{w_i^{att}}$ **combined with prototypical averaging weight $\boldsymbol{w_i^{avg}}$**

"Dynamic Few-Shot Visual Learning without Forgetting", Gidaris et al. 18

Generate weights with a GNN Denoising AutoEncoder

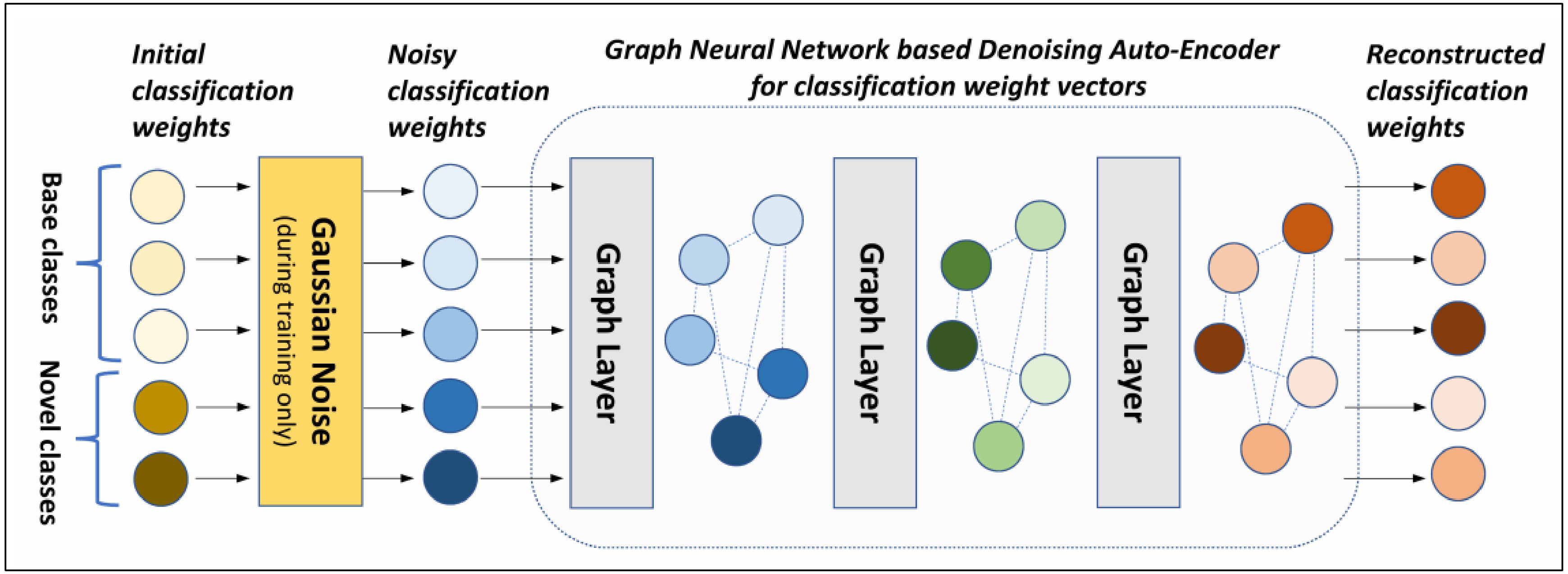




- **Learning inter-class correlations with GNN based Denoising AutoEncoders**
  - Nodes = classes
  - Edges = each class connected to top most similar classes
  - **More expressive than a single layer attention mechanism**

"Generating classification weights with GNN denoising auto encoders for few-shot learning ", Gidaris et al. 19

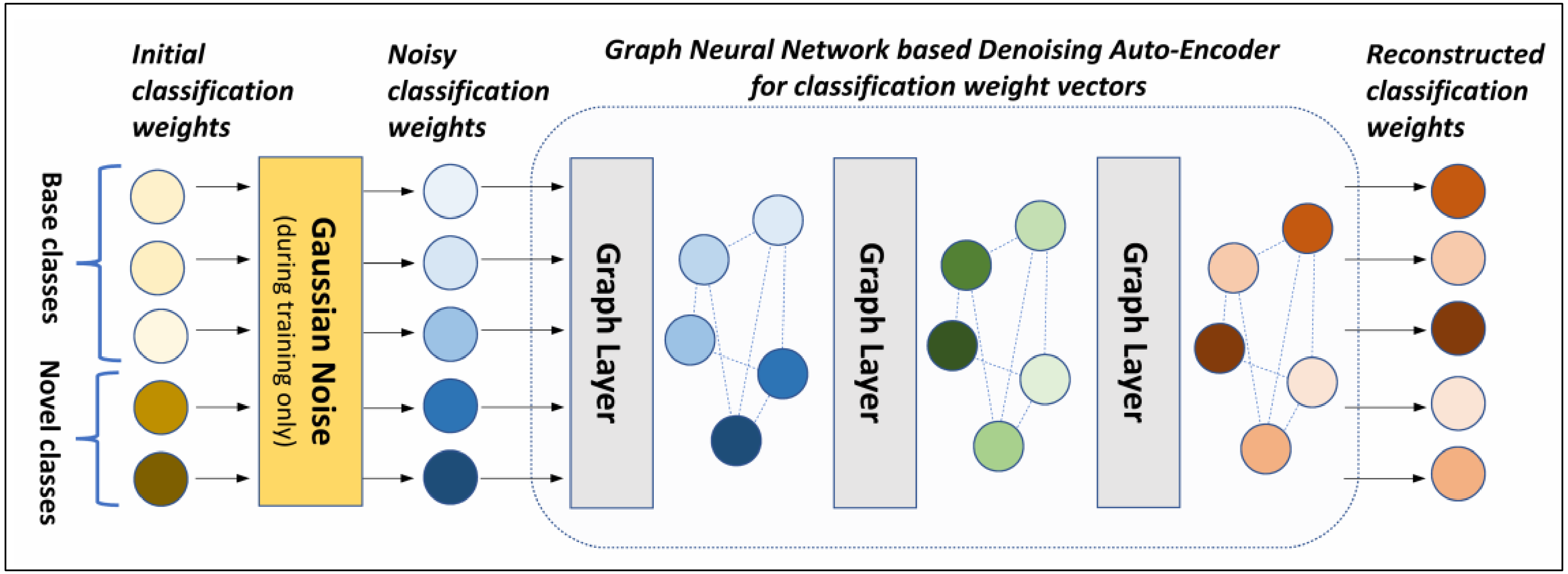
Generate weights with a GNN Denoising AutoEncoder



- **Learning inter-class correlations with GNN based Denoising AutoEncoders**
  - Nodes = classes
  - Edges = each class connected to top most similar classes
  - **More expressive than a single layer attention mechanism**

“Generating classification weights with GNN denoising auto encoders for few-shot learning ”, Gidaris et al. 19

Generate weights with a GNN Denoising AutoEncoder



- **Learning inter-class correlations with GNN based Denoising AutoEncoders**
- **DAE:** reconstructs initial (noisy) prototypical averaging weights
  - Meta-training here can be data hungry
  - injecting noise during meta-training ➔ **regularize meta-training**

"Generating classification weights with GNN denoising auto encoders for few-shot learning ", Gidaris et al. 19

Few-shot learning without forgetting

Approach	Novel classes K=1	2	5	10	20	All classes K=1	2	5	10	20
Prior work										
Prototypical Networks	39.3	54.4	66.3	71.2	73.9	49.5	61.0	69.7	72.9	74.6
Matching Networks	43.6	54.0	66.0	72.5	76.9	54.4	61.0	69.0	73.7	76.5
Logistic regression [Hariharan *et al.* 16]	38.4	51.1	64.8	71.6	76.6	40.8	49.9	64.2	71.9	76.9
Logistic regression w/ H [Hariharan *et al.* 16]	40.7	50.8	62.0	69.3	76.5	52.2	59.4	67.6	72.8	76.9
Prototype Matching Nets w/ H [Wang *et al.* 18]	45.8	57.8	69.0	74.3	77.4	57.6	64.7	71.9	75.2	77.5
Cosine Classifier with few-shot classification weight generation										
Feature Averaging [Gidaris *et al.* 18]	45.4	56.9	68.9	74.5	77.7	57.0	64.3	72.3	75.6	77.3
Attention Mechanism [Gidaris *et al.* 18]	46.2	57.5	69.2	74.8	**78.1**	58.2	65.2	72.7	**76.5**	**78.7**
GNN Denoising AutoEncoder [Gidaris *et al.* 19]	**48.0**	**59.7**	**70.3**	**75.0**	77.8	**59.1**	**66.3**	**73.2**	76.1	77.5

Table 2: Top-5 accuracies on the novel and on all classes for the ImageNet-FS benchmark [13]. To report results we use 100 test episodes.

Exploiting inter-class correlations (attention, GNN) leads to better performance

Source: “Generating classification weights with GNN denoising auto encoders for few-shot learning ”, Gidaris et al. 18

Learn to generate classification weights

- (Almost) simple training:
  - Single classification network, standard supervised pre-training
  - Meta-training: only for the parameter generating module
- More flexible: unified recognition of both base and novel classes
- Same test speed as typical classification networks

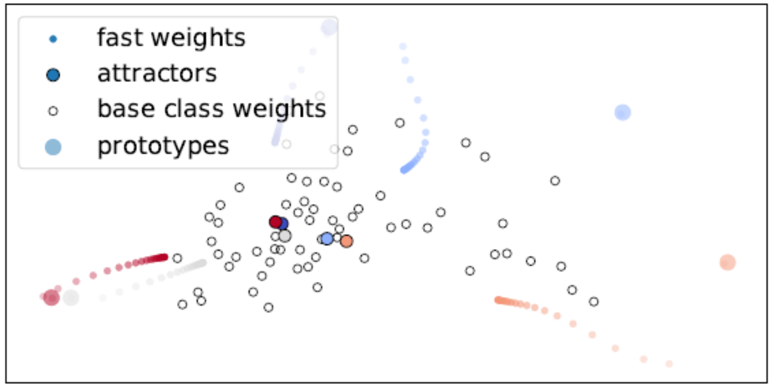
- The parameter generating module might be data hungry
- Constrained by quality of pre-trained representations
  - Similar to metric learning based methods

Learning Weights with Attention Attractor Networks

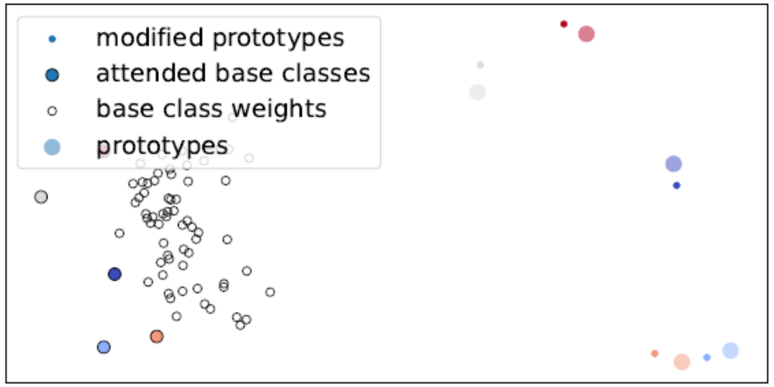
Optimization-based meta-learning with dynamic regularization:

$$W = \min_W \left(\text{CrossEntropyLoss}(S, W) + \sum_i \text{R}(w_i - w_i^{att}) \right)$$

- The meta-learner is trained to predict (using S_i) priors w_i^{att} so that the optimized weights W would minimize the classification loss on the query set Q



(a) Ours



(b) LwoF [9]

Figure 3: Visualization of a 5-shot 64+5-way episode using PCA. **Left:** Our attractor model learns to "pull" prototypes (large colored circles) towards base class weights (white circles). We visualize the trajectories during episodic training; **Right:** Dynamic few-shot learning without forgetting [9].

"Incremental Few-Shot Learning with Attention Attractor Networks", Ren et al. 19

Agenda

- Introduction
- Main types of few-shot learning algorithms
- Few-shot learning without forgetting
- Final notes

Final notes

- **Few-shot visual learning is important**

Final notes

- **Few-shot visual learning is important**

- **But, common few-shot benchmarks are insufficient**
  - Omniglot: saturated
  - MiniImageNet: with proper tuning all methods achieve similar results, not realistic

Final notes

- **Few-shot visual learning is important**

- **But, common few-shot benchmarks are insufficient**
  - Omniglot: saturated
  - MiniImageNet: with enough tuning all methods achieve similar results, not realistic setting

- **More realistic benchmarks:**
  - “Low-shot Visual Recognition by Shrinking and Hallucinating Features”, Hariharan et al. 17
  - “Few-Shot Learning with Localization in Realistic Settings”, Wertheimer et al. 19
  - “Large-Scale Long-Tailed Recognition in an Open World”, Liu et al. 19
  - “Meta-Dataset: A dataset for datasets for learning to learn from few examples”, Triantafillou et al. 19

A Closer Look to Few-Shot Classification

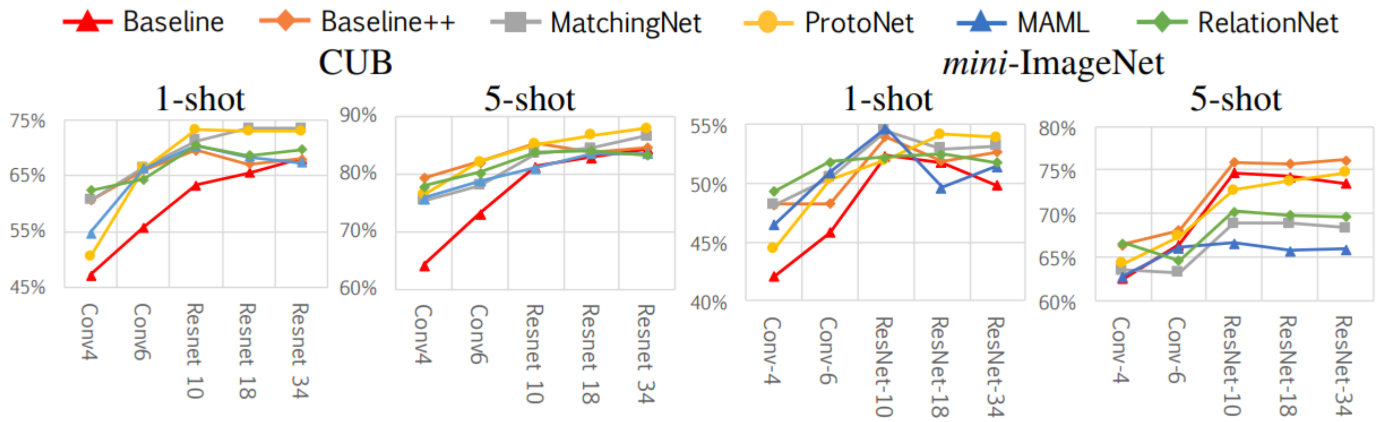


Figure 3: **Few-shot classification accuracy vs. backbone depth**. In the CUB dataset, gaps among different methods diminish as the backbone gets deeper. In *mini*-ImageNet 5-shot, some meta-learning methods are even beaten by Baseline with a deeper backbone. (Please refer to

"A Closer Look to Few-shot classification", Chen et al. 19

Method	CUB 1-shot	CUB 5-shot	*mini*-ImageNet 1-shot	*mini*-ImageNet 5-shot
Baseline	47.12 ± 0.74	64.16 ± 0.71	42.11 ± 0.71	62.53 ± 0.69
Baseline++	60.53 ± 0.83	79.34 ± 0.61	48.24 ± 0.75	66.43 ± 0.63
MatchingNet Vinyals et al. (2016)	60.52 ± 0.88	75.29 ± 0.75	48.14 ± 0.78	63.48 ± 0.66
ProtoNet Snell et al. (2017)	50.46 ± 0.88	76.39 ± 0.64	44.42 ± 0.84	64.24 ± 0.72
MAML Finn et al. (2017)	54.73 ± 0.97	75.75 ± 0.76	46.47 ± 0.82	62.71 ± 0.71
RelationNet Sung et al. (2018)	62.34 ± 0.94	77.84 ± 0.68	49.31 ± 0.85	66.60 ± 0.69

Baseline:
pre-training + fine-tuning last layer

Baseline++:
cosine classifier

- Meta-learning algorithms and network designs of **growing complexity**, but
- **Well-tuned baselines: often on par / better than SoTA meta-learning methods**
- **Baselines: scale better with deeper backbones**

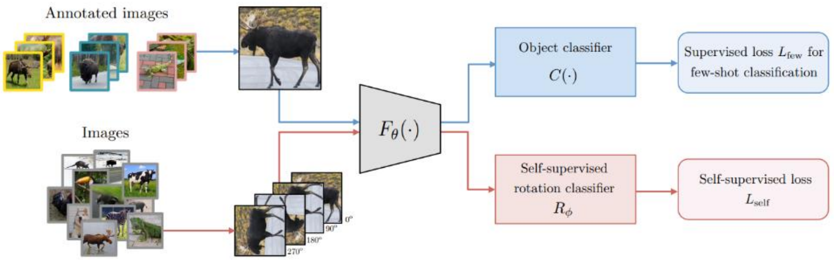
A different direction

Focus on pre-training richer representations

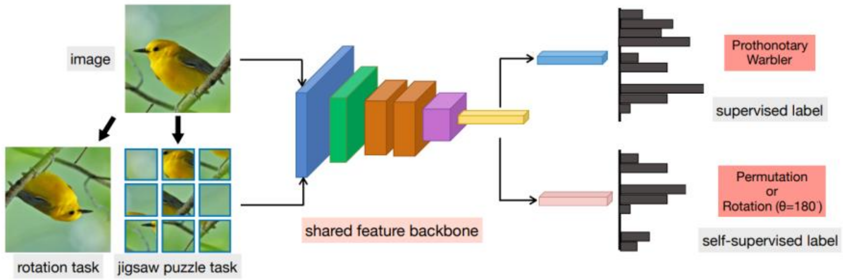
- Representations that know more about the world can adapt better

- **Leveraging self-supervision** (see next talk by Relja and Andrey)

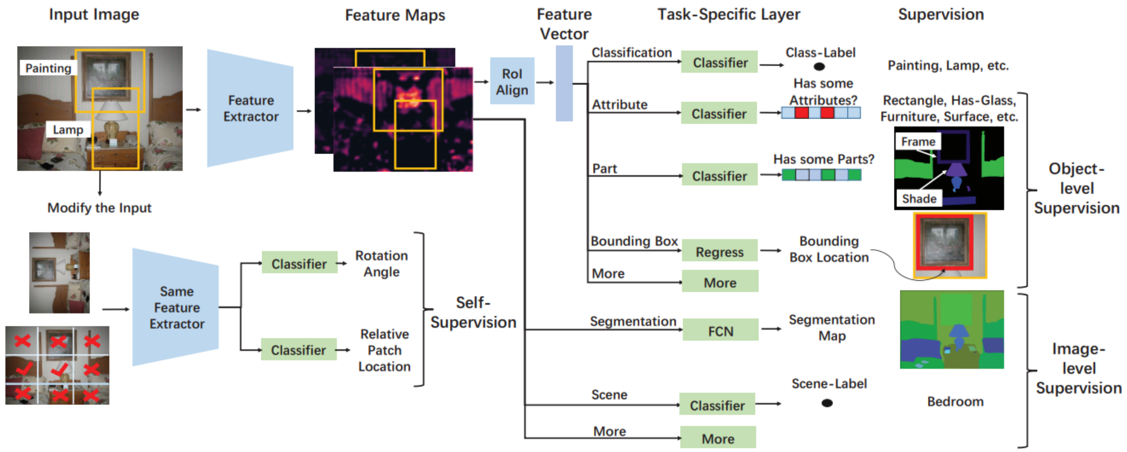
"Boosting few-shot visual learning with self-supervision", Gidaris et al. 19



""When does self-supervision improve few-shot learning?", Su et al. 19



"Learning generalizable representations via diverse supervision", Pang et al. 19



Also:
"Charting the right manifold: manifold mixup for few-shot learning", Mangla et al. 20
"Rethinking few-shot image classification: a good embedding is all you need?", Tian et al. 20

Not covered because of time constraints

- **Semi-supervised few-shot / meta learning:**
  - **“**Low-shot learning with large-scale diffusion**”,** Douze et al. 18
  - **“**Meta-learning for semi-supervised few-shot classification**”,** Triantafillou et al. 18

- **Few-shot / meta learning with noise labels:**
  - “Graph convolutional networks for learning with few clean and many noisy labels”, Iscen et al 20

- **Learning with imbalanced datasets (many-shot and few-shot classes):**
  - “Learning to model the tail”, Wang et al. 17
  - **“**Large-scale long-tailed recognition in an open world”, Liu et al. 19
  - “Decoupling representation and classifier for long-tailed recognition”, Kang et al. 20

- **Few-shot learning beyond image classification:**
  - “Few-shot object detection via feature reweighting”, Kang et al. 19
  - “Meta-learning to detect rare objects”, Wang et al. 19
  - “Few-shot semantic segmentation with prototype learning”, Dong et al. 18
  - “PANet: Few-shot image semantic segmentation with prototype alignment”, Wang et al. 19
  - “Tracking by Instance Detection: A Meta-Learning Approach”, Wang et al. 20
  - …

The end