Knowledge Transfer and Adaptation

Meta-Learning, Metric Learning, Few-Shot Learning

Antonio Carta

antonio.carta@unipi.it

Intro

- what is meta-learning
- types of meta-learning algorithms
- metric-based meta-learning for few-shot learning

Motivation

- **MTL**: Train multiple tasks jointly. Sharing parts of the network encourage positive transfer
- **Transfer learning**: Initialize with a pretrained model. Pretraining optimized for transfer, finetuning optimized for adaptation
- **Meta-learning**: Can we optimize explicitly the meta-objectives (transfer, fast adaptation, hyperparameter search)?
    - *a.k.a. Learning to learn*

Meta-Learning

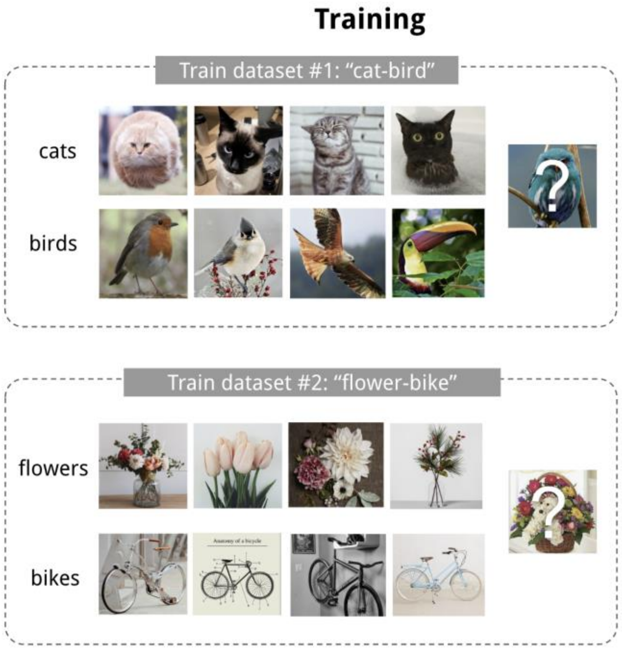
- **classic learning**:
  - given a dataset (sampled from a task), optimize a model
  - input: samples (e.g. a single image)
  - output: trained model (e.g. a CNN that classifies images)
- **meta-learning**:
  - given a set of datasets (tasks), optimize learning algorithms/hyperparameters
  - input: a dataset (e.g. images on different domains)
  - output: a general algorithm that optimizes models on images. Examples:
    - a good model's initialization (optimized for our tasks)
    - a model able to learn from few examples (optimized for our tasks)
    - an optimizer with better convergence (optimized for our tasks)

Example

- We want to train a humanoid robot to walk (this is our family of tasks)
- a robot has been trained to walk in a lab in different scenario (our meta-train set)
- Now we deploy the robot in the real world (meta-test set)
  - different environments, sim2real drift
  - we have to train the robot again
- We know that we have to train the robot again, but can we reuse the previous knowledge?
  - Ideal solution: the robot takes a few steps, stumbles a couple of times, and then it is adapted to the new environment
  - We can always move the robot to a new environment and let it learn to walk again quickly and efficiently

Example

An example of 4-shot 2-class image classification.



Testing

Do you see the difference with "standard" learning? Where is the "meta" part?

In the meta-test phase, we have to train the model on the meta-test tasks

image source: lilianweng.io

Sampling Few-Shot Episodes

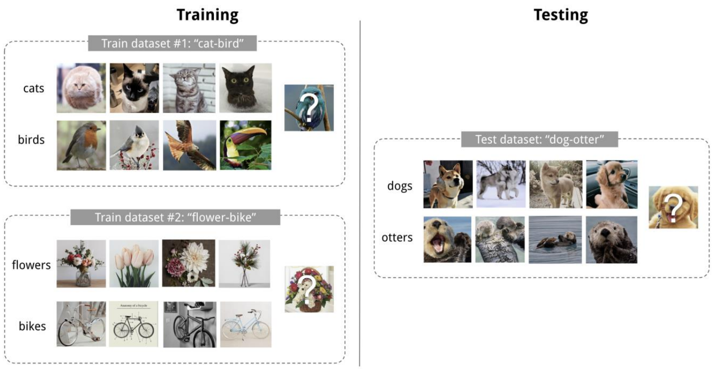
Meta-Train and Meta-Test will sample from a disjoint set of classes

Algorithm 4 Random sampling of episodes for a N_C-way N_S-shot scenario. N samples in the training set, K classes, N_Q query samples per class. $RandomSample(S, N)$ uniformly samples without replacement N elements from S. Training set $\mathcal{D} = \{(\mathrm{x}_1, y_1), \ldots, (\mathrm{x}_N, y_N)\}$. $\mathcal{D}_k$ denotes the subset of $\mathcal{D}$ containing all elements (x_i, y_i) such that $y_i = k$.

procedure SAMPLEFEWSHOTEPISODE($\mathcal{D}$)
    $V \leftarrow RandomSample\left(\{1, \ldots, K\}, N_C\right)$ ▷ Select class indices
    $S, Q \leftarrow \{\}, \{\}$
    for k in $\{1, \ldots, N_C\}$ **do**
        $S_k \leftarrow RandomSample\left(\mathcal{D}_{V_k}, N_S\right)$ ▷ Select support examples
        $Q_k \leftarrow RandomSample\left(\mathcal{D}_{V_k} \setminus S_k, N_Q\right)$ ▷ Select query examples
        $S \leftarrow S \cup S_k$
        $Q \leftarrow Q \cup Q_k$
    return V, S, Q

tasks and meta-tasks

- **task**: $< p_i(x), p_i(y|x), L_i >$
  - During training we have a dataset
  - $D_i = \{< x, y > \sim p_i(x, y)\}$
- **meta-task**: a distribution of tasks + a loss function
  - $\mathcal{T}_1, \dots, \mathcal{T}_n \sim p(\mathcal{T}), \mathcal{T}_j \sim p(\mathcal{T})$
  - During training we have a set of datasets, each one sampled from a different task in $\mathcal{T}$
  - $\{D_i \sim \mathcal{T}_i,\ \ \mathcal{T}_i \sim p(\mathcal{T})\}$



learner and meta-learner

learner: a machine learning model

- Example: a deep neural network

meta-learner: a parameterized learning algorithm

- A learned optimizer
- A learned initialization
- A learned hyperparameter search (e.g. neural architecture search)

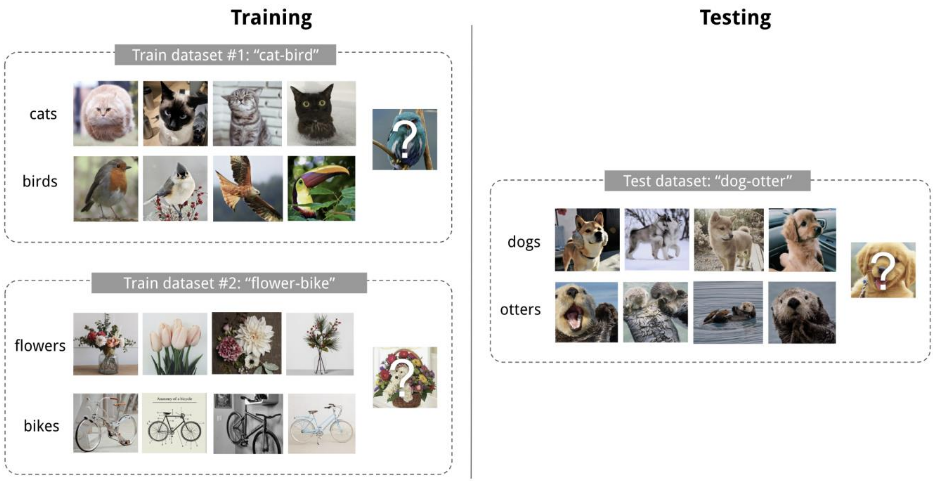
Formal Definition

Objective: $\theta^* = argmin_\theta E_{D\sim\ P(D)}[L_\theta(D)]$

- We sample **datasets** (not instances)
- We sample from $P(D)$ (the datasets distribution defined by our family of tasks)
- We minimize the parameters (θ) over the entire family of tasks
- Assumption: tasks have some shared structure

Common Terminology

- **support set**: task training set $\mathcal{D}_i^{\mathrm{tr}}$
- **query set**: task test dataset $\mathcal{D}_i^{\mathrm{test}}$
- **meta-training**: training process over the meta-train tasks
- **meta-test**: learning a new task given its support set



Example - Omniglot

The meta-learning MNIST

- 50 alphabets split into
  - background set of 30 alphabets
  - evaluation set of 20 alphabets
- Use the background set to learn general knowledge about characters
- Use the evaluation set for one-shot learning

https://github.com/brendenlake/omniglot

Probabilistic vs Optimization Perspective

- **optimization view**:
  - given a set of meta-datasets/tasks
  - find model/optimization algo. able to (quickly) learn new (related) tasks
- **probabilistic view**:
  - given meta-datasets/tasks
  - extract prior knowledge about tasks and use it to infer posterior for new tasks

Meta-Learning Families

- Model-Based
- Optimization-Based
- **Metric-Based**

Model-based Meta-Learning

Design model architecture for fast adaptation on new tasks. Fast adaptation either comes from the network design or from the meta-learner

- ***Memory-Augmented Neural Networks (MANN)*** *such as the Neural Turing Machines have been adapted for meta-learning*
- ***Meta-network*** decompose the network into *fast and slow weights*. Slow weights are updated via SGD while the fast weights are adapted via a meta-learner.

Optimization - based

Model optimization algorithm via a meta-learner that updates the model's parameters

- **LSTM meta-learner** updates the weights with a recurrent network (learned optimizer).
  - Think about the LSTM cell update. It's very similar to the SGD update
- **Model-Agnostic Meta-Learning (MAML)** learns an initialization that generalizes over task (fast adaptation and few-shot)

Metric - based

Metric Learning: learns a metric over the input space

- Intuitively: a KNN where the distance function is learned via a deep neural network
- IDEA: learn the metric during the meta-train, use it during meta-test
- Classification using a distance metric: $P_\theta(y \mid \mathbf{x}, S) = \sum_{(\mathbf{x}_i, y_i) \in S} k_\theta(\mathbf{x}, \mathbf{x}_i) y_i$

Methods:

- Siamese Networks
- Matching Networks
- Relation Network
- Prototypical Networks

Few-Shot Learning

few-shot classification: learning from very small datasets

Meta-Training Loss for Few-Shot Meta-Learning:

$$\theta = \text{argmax}_{\theta} E_{L \subset \mathcal{L}} \left[E_{S^L \subset \mathcal{D}, B^L \subset \mathcal{D}} \left[\sum_{(x,y) \in B^L} P_{\theta}\,(x, y, S^L) \right] \right]$$

- L subset of labels
- S^L support set (data used for training)
- B^L query set (data used for testing during meta-training)
- P_θ classification model (notice the dependency on S^L)
- During meta-test we receive a new support set for training on the new task

def. from https://lilianweng.github.io/posts/2018-11-30-meta-learning/

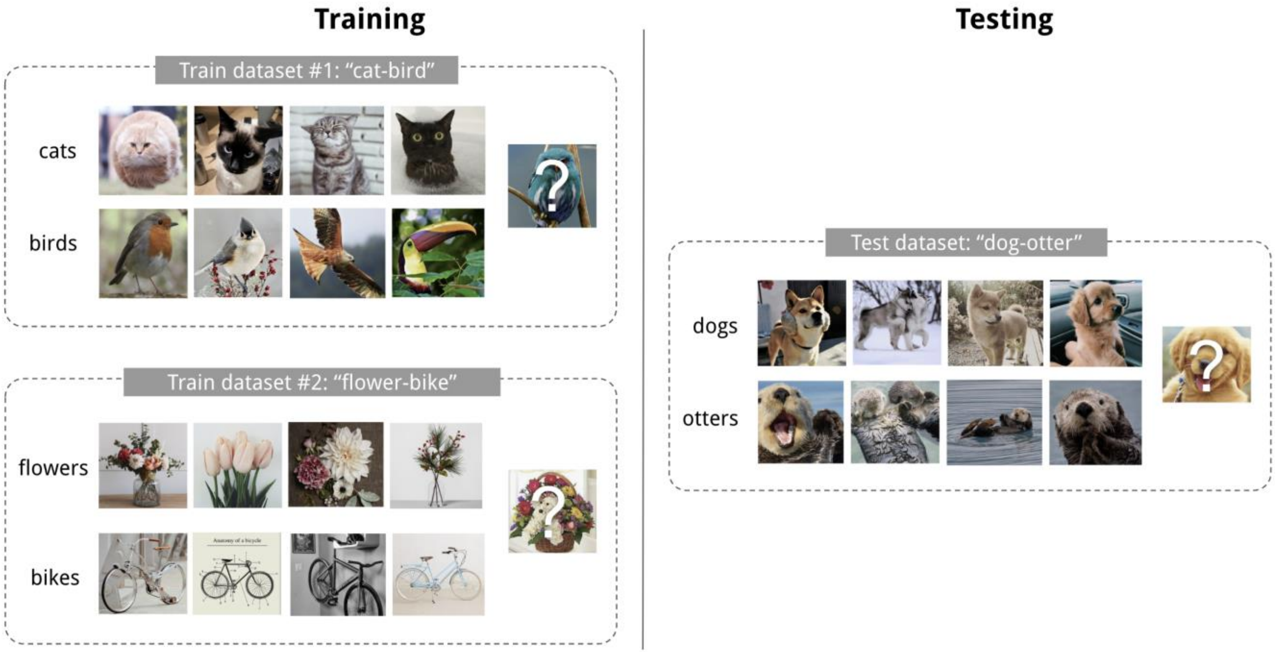
Extreme Settings - One-shot and Zero-shot

- **one-shot**: one example per class
  - "Object Classification from a Single Example Utilizing Class relevance Metrics", M. Fink, NeurIPS 2004
  - "One-shot Learning of Object Categories", Fei-Fei et al, TPAMI 2006
- **zero-shot**: zero examples per class
  - "Learning To Detect Unseen Object Classes by Between-Class Attribute Transfer", Lampert et al, CVPR 2009

Question: how can you even solve zero-shot learning?

Example

An example of 4-shot 2-class image classification.



Non-Parametric Models

Example: kNN is a non-parametric model

- no parameters
- train: store dataset
- inference: output an average of the k closest distances
- A good choice for few-shot learning and low-data regimes in general. The model is simple and works well if we have a good distance metric.

Objective:

- **Meta-Train**: learn a distance metric
- **Meta-Test**: learn a non-parametric model using the distance metric

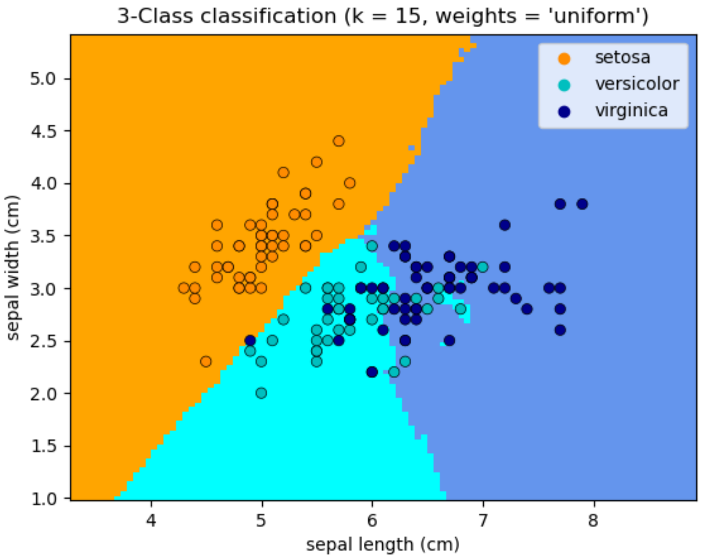


image source: scikit-learn documentation

What Metric?

l2 distance in pixel space is very poor

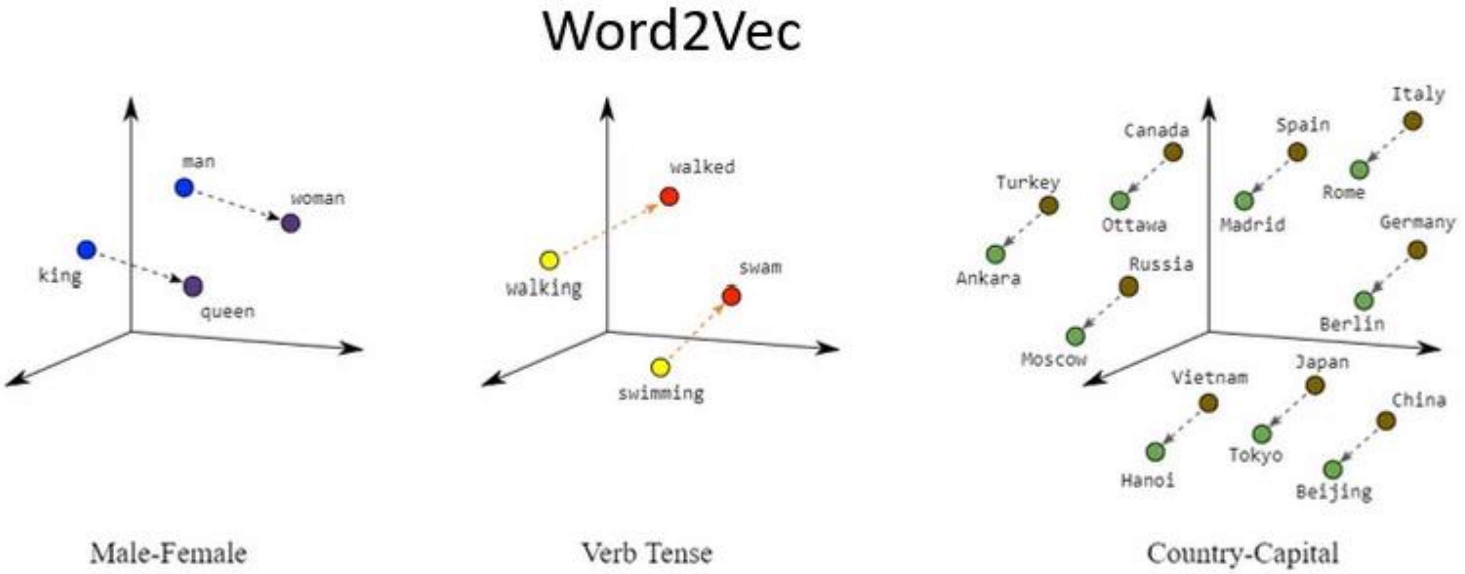
- doesn’t consider invariances (rotations, translations)
- not semantic (background, object recognition)
- Curse of dimensionality (especially bad for few-shot settings)

INTUITION: we have to learn a metric. This is a meta-learning problem!

Metric-based - the importance of embeddings

- we can learn embeddings to map instances in space
- embedding: map instances in a high dimensional space
  - encode relationships between instances
  - semantic relationships encoded as distances in embedding space
- objective: discriminate classes by computing distances in the embedding space
  - Example: “Activation Atlas”, Carter et al, Distill 2019

Example - Embeddings in NLP



General Plan

- given the meta-training data (set of datasets)
- **meta-train**: learn a good metric space
- **meta-test**: use the learned metric space to classify images

Deep Metric Learning

- The distance function is computed via a DNN
- The DNN computes embeddings
- Distances in the embeddings space are semantic
- Distances in the embeddings space are easy to compute (e.g. cosine similarity)

Embed with DNN -> nearest neighbors classification

Siamese Networks

- Two DNN with weight sharing compute embeddings
- **Input**: a pair of images, one for each network
- **Output**: whether the images are from the same class or not (binary classification)

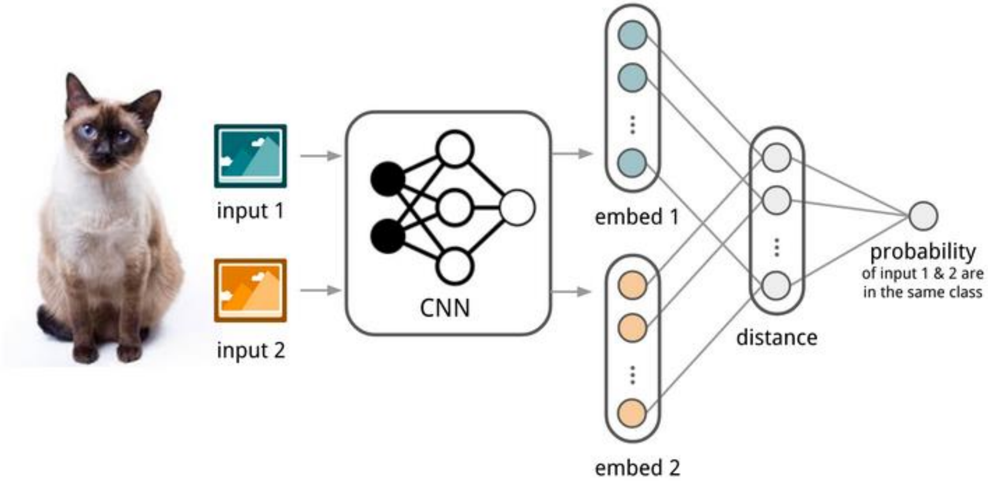


image source: https://lilianweng.github.io/posts/2018-11-30-meta-learning/
Paper: Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-Shot Image Recognition,"

Siamese Networks – Meta-Train and Meta-Test

- **Meta-train**: the siamese network is trained to predict whether two input images are from the same class
- **Meta-test**: the siamese network processes all the image pairs between a test image and every image in the support set
  - final prediction is the class of the support image with the highest probability

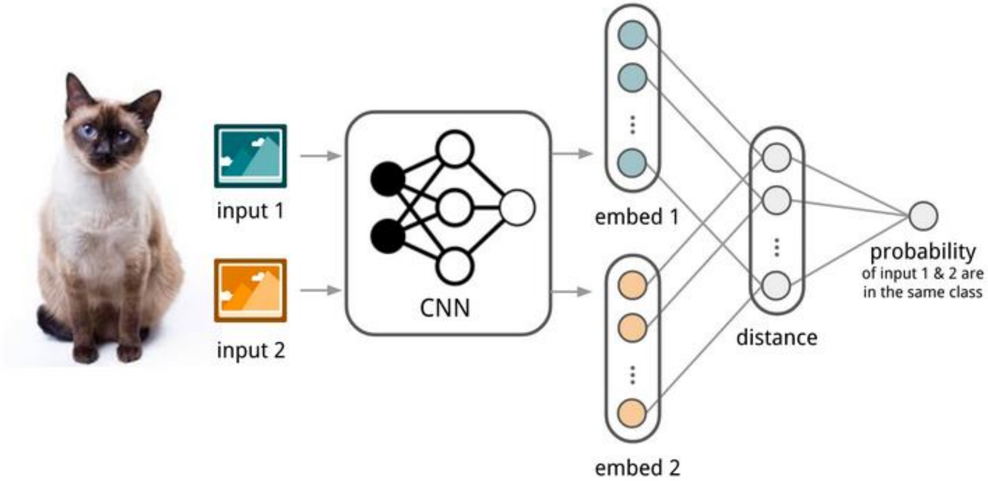
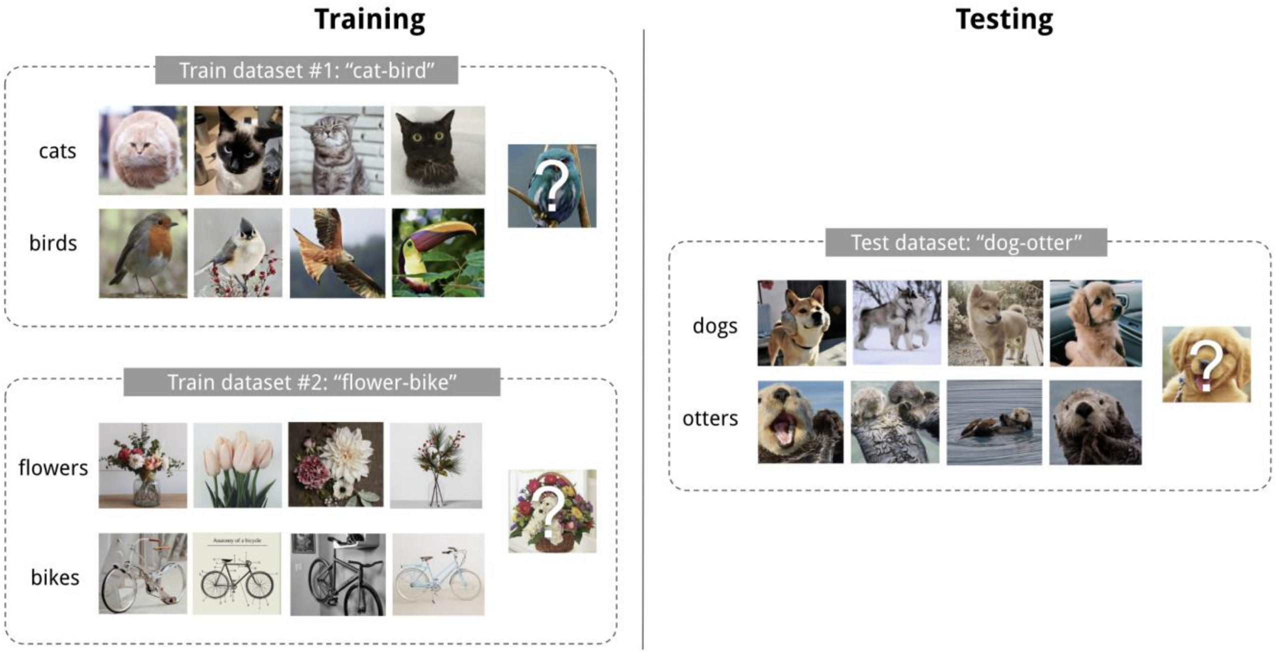


image source: https://lilianweng.github.io/posts/2018-11-30-meta-learning/

Paper: Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-Shot Image Recognition,"

Example

An example of 4-shot 2-class image classification.



Siamese Networks – Meta-Train

- Siamese network f_θ (a CNN) learns to encode two images into embeddings
- L1-distance between two embeddings $|f_\theta(\mathbf{x_i}) - f_\theta(\mathbf{x_j})|$
  - You can use any differentiable distance function
- The distance is converted to a probability p by a linear feedforward layer and sigmoid.
  - probability of whether two images are drawn from the same class.
  - $p(\mathbf{x_i}, \mathbf{x_j}) = \sigma(\mathbf{W}|f_\theta(\mathbf{x_i}) - f_\theta(\mathbf{x_j})|)$
- Cross-entropy loss between pair of images
- $\mathcal{L}(B) = \sum_{(\mathbf{x_i},\mathbf{x_j},y_i,y_j)\in B} \mathbf{1}_{\mathbf{y_i}=\mathbf{y_j}} \log p\left(\mathbf{x_i}, \mathbf{x_j}\right) + \left(1 - \mathbf{1}_{\mathbf{y_i}=\mathbf{y_j}}\right) \log\left(1 - p(\mathbf{x_i}, \mathbf{x_j})\right)$
- Images in the training batch B can be augmented with distortion.

image source: https://lilianweng.github.io/posts/2018-11-30-meta-learning/

Paper: Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-Shot Image Recognition,"

Siamese Networks – Meta-Test

- **Meta-test**:
- Given a support set S and a test image x the final predicted class is
$$\widehat{c_S}(\mathbf{x}) = c\left(\arg\max_{\mathbf{x_i} \in S} P\left(\mathbf{x}, \mathbf{x_i}\right)\right)$$
- $c(\mathbf{x})$ is the class label of an image and $\hat{c}(.)$ is the predicted label.

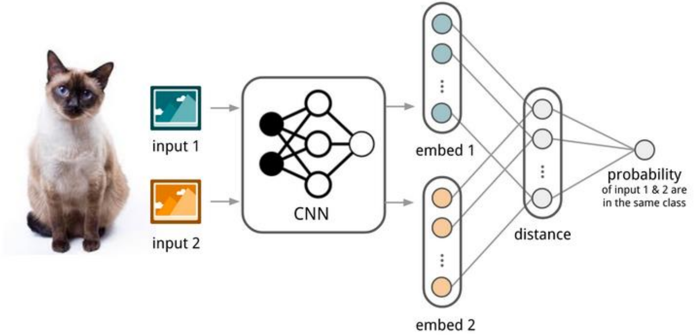


image source: https://lilianweng.github.io/posts/2018-11-30-meta-learning/
Paper: Koch, Gregory, Richard Zemel, and Ruslan Salakhutdinov. "Siamese Neural Networks for One-Shot Image Recognition,"

Siamese Networks – Assumptions and Limitations

- Learned embeddings generalize to unknown classes
  - During meta-test we receive new tasks but we don't update the siamese network (the distance metric)
- Different meta-train and meta-test conditions
  - During meta-train binary classification
  - During meta-test n-way classification (all samples in the support set)

Matching Networks

- **PROBLEM**: We want the same meta-train and meta-test conditions
- **SOLUTION**: do k-way classification during meta-train

COMPONENTS:

- f embeds test sample
- g embeds support set (i.e. the entire dataset)

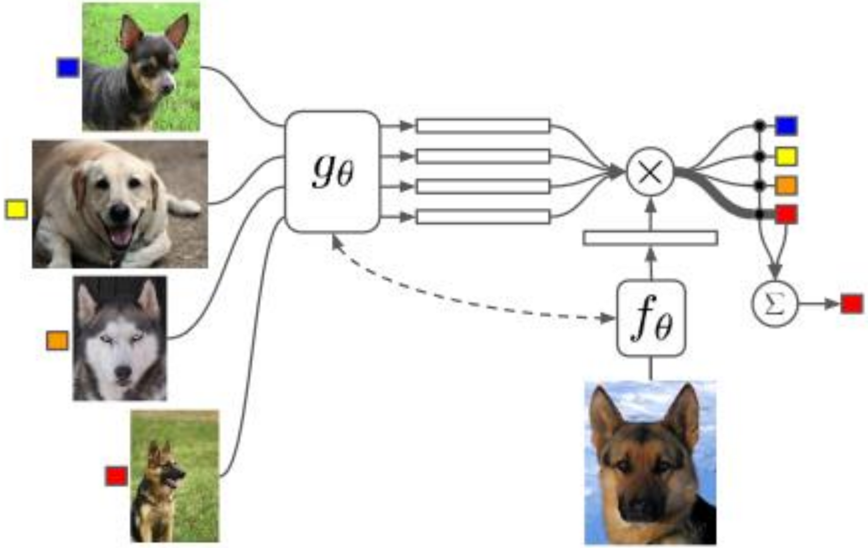


Figure 1: Matching Networks architecture

Matching Networks – Attention

- f embeds test sample
- g embeds support set (i.e. the entire dataset)
- Distance: Cosine similarity
- Compute attention over all the samples in the support set
- $a(\mathbf{x}, \mathbf{x_i}) = \frac{\exp(cossim(f(\mathbf{x}), g(\mathbf{x_i})).}{\sum_{j=1}^{k} \exp\left(cossim\left(f(\mathbf{x}), g(\mathbf{x_j})\right)\right).}$

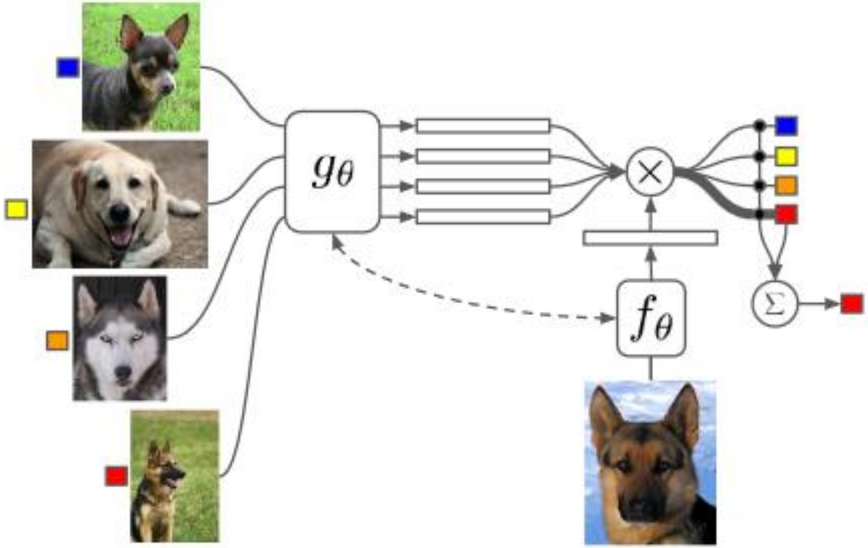


Figure 1: Matching Networks architecture

Matching Networks – Output

- attention $a(\mathbf{x}, \mathbf{x_i})$
- **Output**:
  - Weighted sum of the support set classes
  - Attention weights
  - $c_S(\mathbf{x}) = P(y \mid \mathbf{x}, S) = \sum_{i=1}^{k} a(\mathbf{x}, \mathbf{x_i}) y_i$
  - $S = \{(\mathbf{x_i}, y_i)\}_{i=1}^{k}$

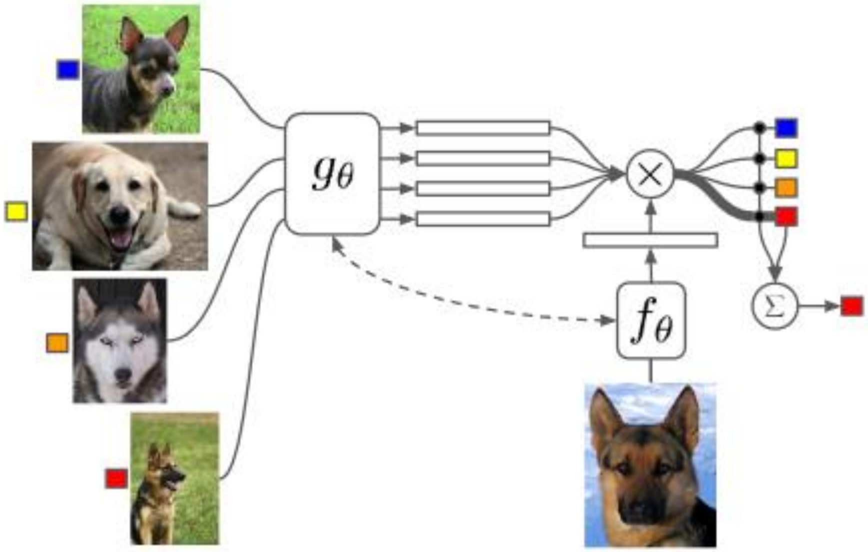


Figure 1: Matching Networks architecture

Matching Networks – Embedding Network

- How do we compute the embeddings?
- **simple embedding**: f takes a single data sample as input.
  - It is possible to set $f = g$
- **full context embeddings**: consider the entire support set together to compute context embeddings:
  - Bidirectional LSTM: $g_\theta(\mathbf{x_i}, S)$

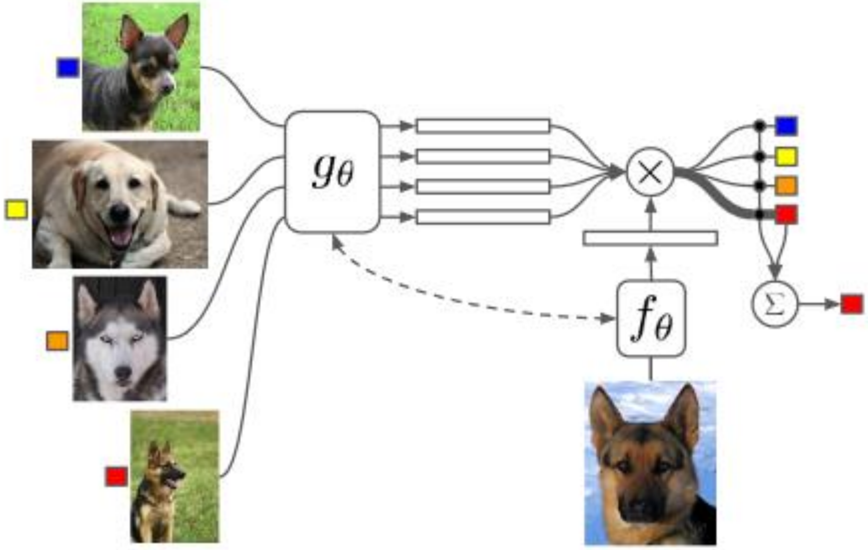


Figure 1: Matching Networks architecture

Matching Networks – Meta-Train and Meta-Test

- **meta-train**: train f and g on k-way classification on the meta-train sets
- **meta-test**: embed support set, k-way classification on unseen data x

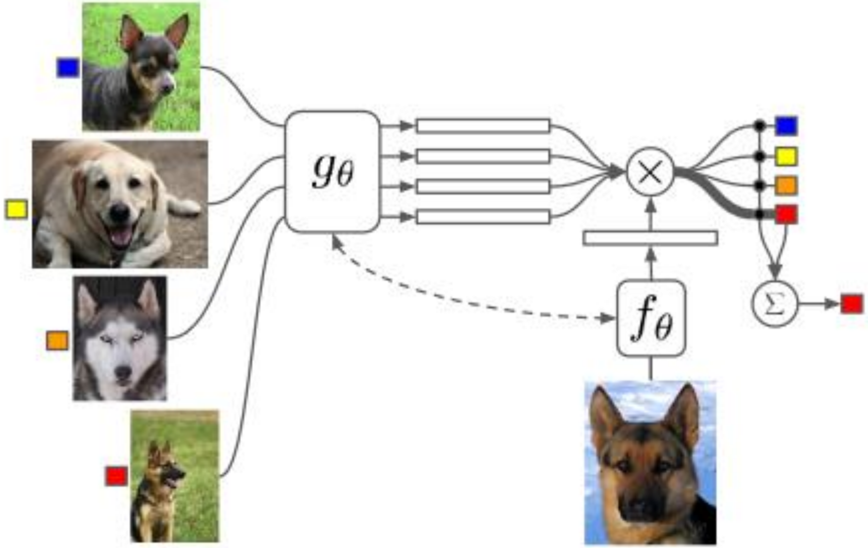


Figure 1: Matching Networks architecture

Results

Model	Matching Fn	Fine Tune	5-way Acc		20-way Acc	
			1-shot	5-shot	1-shot	5-shot
PIXELS	Cosine	N	41.7%	63.2%	26.7%	42.6%
BASELINE CLASSIFIER	Cosine	N	80.0%	95.0%	69.5%	89.1%
BASELINE CLASSIFIER	Cosine	Y	82.3%	98.4%	70.6%	92.0%
BASELINE CLASSIFIER	Softmax	Y	86.0%	97.6%	72.9%	92.3%
MANN (NO CONV) [21]	Cosine	N	82.8%	94.9%	–	–
CONVOLUTIONAL SIAMESE NET [11]	Cosine	N	96.7%	98.4%	88.0%	96.5%
CONVOLUTIONAL SIAMESE NET [11]	Cosine	Y	97.3%	98.4%	88.1%	97.0%
MATCHING NETS (OURS)	Cosine	N	**98.1%**	**98.9%**	**93.8%**	98.5%
MATCHING NETS (OURS)	Cosine	Y	97.9%	98.7%	93.5%	**98.7%**

Table 1: Results on the Omniglot dataset.

Vinyals, Oriol, Charles Blundell, and Timothy Lillicrap. “Matching Networks for One Shot Learning.”

Matching Networks – Advantages

- No difference in the task between meta-train and meta-test
- Can exploit relationship in the entire support set when using full context embeddings

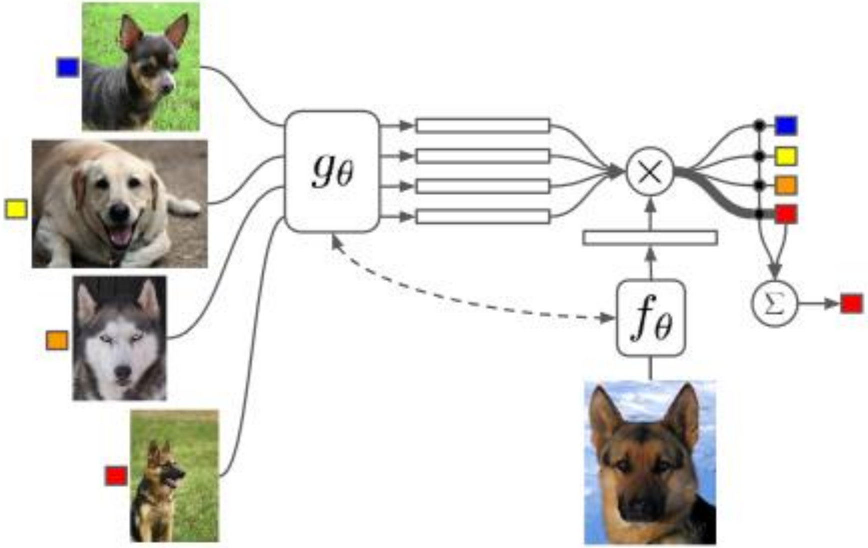
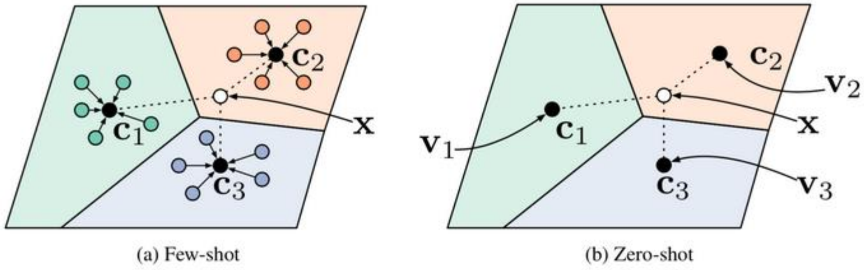


Figure 1: Matching Networks architecture

Prototypical Networks

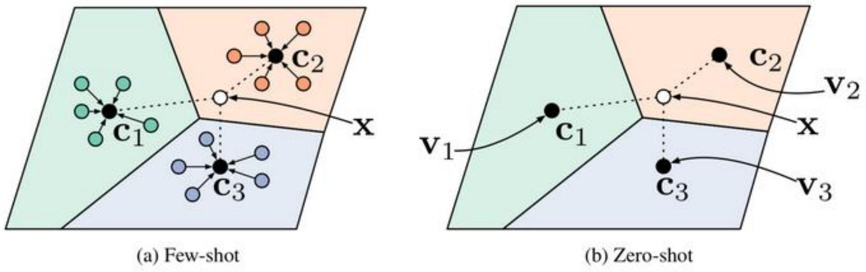
Prototypical networks compute embeddings with an embedding function f_θ and compute prototypes for each class using the support set.



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

Prototypical Networks

- f_θ encodes the input in the embeddings space
- A **prototype** is computed as the average embedding in the support set
  - $\mathbf{v_c} = \frac{1}{|S_c|}\sum_{(\mathbf{x_i},y_i)\in S_c} f_\theta(\mathbf{x_i})$
- Output:
  - d_φ is a differentiable distance (MSE)

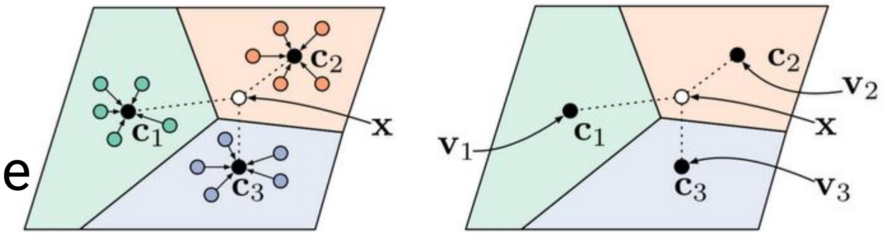


$$P(y = c \mid \mathbf{x}) = \mathrm{softmax}\left(-d_\varphi\left(f_\theta(\mathbf{x}), \mathbf{v}_c\right)\right) = \frac{\exp(-d_\varphi(f_\theta(\mathbf{x}),\mathbf{v}_c))}{\sum_{c' \in \mathcal{C}} \exp(-d_\varphi(f_\theta(\mathbf{x}),\mathbf{v}_{c'}))}$$

J. Snell, et al. 2017. "Prototypical Networks for Few-Shot Learning."

Prototypical Networks

- **Meta-Train**: train f_θ on the meta-train tasks optimizing the cross-entropy
- **Meta-Test**: compute prototypes using the support set and compute $P(y = c|x)$ for the test samples



(a) Few-shot (b) Zero-shot

J. Snell, et al. 2017. "Prototypical Networks for Few-Shot Learning."

Algorithm

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

Input: Training set $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$, where each $y_i \in \{1, \ldots, K\}$. $\mathcal{D}_k$ denotes the subset of $\mathcal{D}$ containing all elements $(\mathbf{x}_i, y_i)$ such that $y_i = k$.

Output: The loss J for a randomly generated training episode.

$V \leftarrow \text{RandomSample}(\{1, \ldots, K\}, N_C)$ ▷ Select class indices for episode

for k in $\{1, \ldots, N_C\}$ **do**

$S_k \leftarrow \text{RandomSample}(\mathcal{D}_{V_k}, N_S)$ ▷ Select support examples

$Q_k \leftarrow \text{RandomSample}(\mathcal{D}_{V_k} \setminus S_k, N_Q)$ ▷ Select query examples

$$\mathbf{c}_k \leftarrow \frac{1}{N_C} \sum_{(\mathbf{x}_i, y_i) \in S_k} f_\phi(\mathbf{x}_i)$$ ▷ Compute prototype from support examples

end for

$J \leftarrow 0$ ▷ Initialize loss

for k in $\{1, \ldots, N_C\}$ **do**

for $(\mathbf{x}, y)$ in Q_k **do**

$$J \leftarrow J + \frac{1}{N_C N_Q} \left[d(f_\phi(\mathbf{x}), \mathbf{c}_k)) + \log \sum_{k'} \exp(-d(f_\phi(\mathbf{x}), \mathbf{c}_k)) \right]$$ ▷ Update loss

end for

end for

Results

Table 1: Few-shot classification accuracies on Omniglot.

Model	Dist.	Fine Tune	5-way Acc. 1-shot	5-way Acc. 5-shot	20-way Acc. 1-shot	20-way Acc. 5-shot
MATCHING NETWORKS [29]	Cosine	N	98.1%	98.9%	93.8%	98.5%
MATCHING NETWORKS [29]	Cosine	Y	97.9%	98.7%	93.5%	98.7%
NEURAL STATISTICIAN [6]	-	N	98.1%	99.5%	93.2%	98.1%
PROTOTYPICAL NETWORKS (OURS)	Euclid.	N	**98.8%**	**99.7%**	**96.0%**	**98.9%**

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

Results

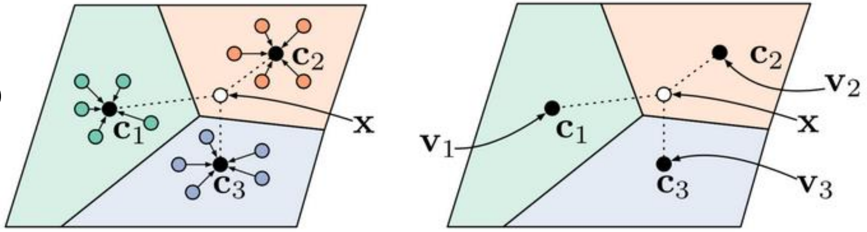
Table 2: Few-shot classification accuracies on *mini*ImageNet. All accuracy results are averaged over 600 test episodes and are reported with 95% confidence intervals. *Results reported by [22].

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	**$49.42 \pm 0.78\%$**	**$68.20 \pm 0.66\%$**

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

Prototypical Networks – Zero-Shot

- **Zero-shot**: in the zero-shot setting, no labeled samples are given. Instead, we have some meta-data for each class
  - i.e. the class prototype is given
  - We still need to train the embeeding function to match the given prototypes
- **Example**: Caltech-UCSD Birds (CUB) 11,788 images of 200 bird species
  - **Meta-data**: 312D attribute vector provided with the CUB dataset encoding various characteristics of the bird species such as their color, shape, and feather patterns.
  - **Model**: pretrained CNN + linear mapping



(a) Few-shot (b) Zero-shot

Table 3: Zero-shot classification accuracies on CUB-200.

Model	Image Features	50-way Acc. 0-shot
ALE [1]	Fisher	26.9%
SJE [2]	AlexNet	40.3%
SAMPLE CLUSTERING [17]	AlexNet	44.3%
SJE [2]	GoogLeNet	50.1%
DS-SJE [23]	GoogLeNet	50.4%
DA-SJE [23]	GoogLeNet	50.9%
PROTO. NETS (OURS)	GoogLeNet	**54.6%**

Recap – Deep Metric Learning

- **Deep Metric Learning**: learn distance metric (meta-train) + distance-based classifier
- **Siamese Networks**: embeddings + pairwise comparisons
  - Difference between meta-train and meta-test hurts performance
- **Matching Networks**: embeddings + k-way attention over support
  - Same meta-train and meta-test conditions
  - It can exploit support set relationships
- **Prototype Networks**: compute class prototypes for classification
  - Simple and effective
  - It can be used for zero-shot learning

Take-Home Messages

- **Meta-learning** is *learning-to-learning,* and it allows to optimize for meta-objectives such as forward transfer and fast adaptation
- **Few-Shot Learning** is a very practical problem that benefits from fast adaptation + transfer
- **Deep Metric Learning** is a simple and effective method to learn in few-shot scenarios

References

- Papers in the footnotes
- Stanford CS330 - Multi-Task and Meta-Learning has some lectures on meta-learning and few-shot learning
- Lilianweng blogpost with many more methods: https://lilianweng.github.io/posts/2018-11-30-meta-learning/

Next Lecture

- Intro to Continual Learning
- The problem of Catastrophic Forgetting
- Notebook