Modelo

Matching Network

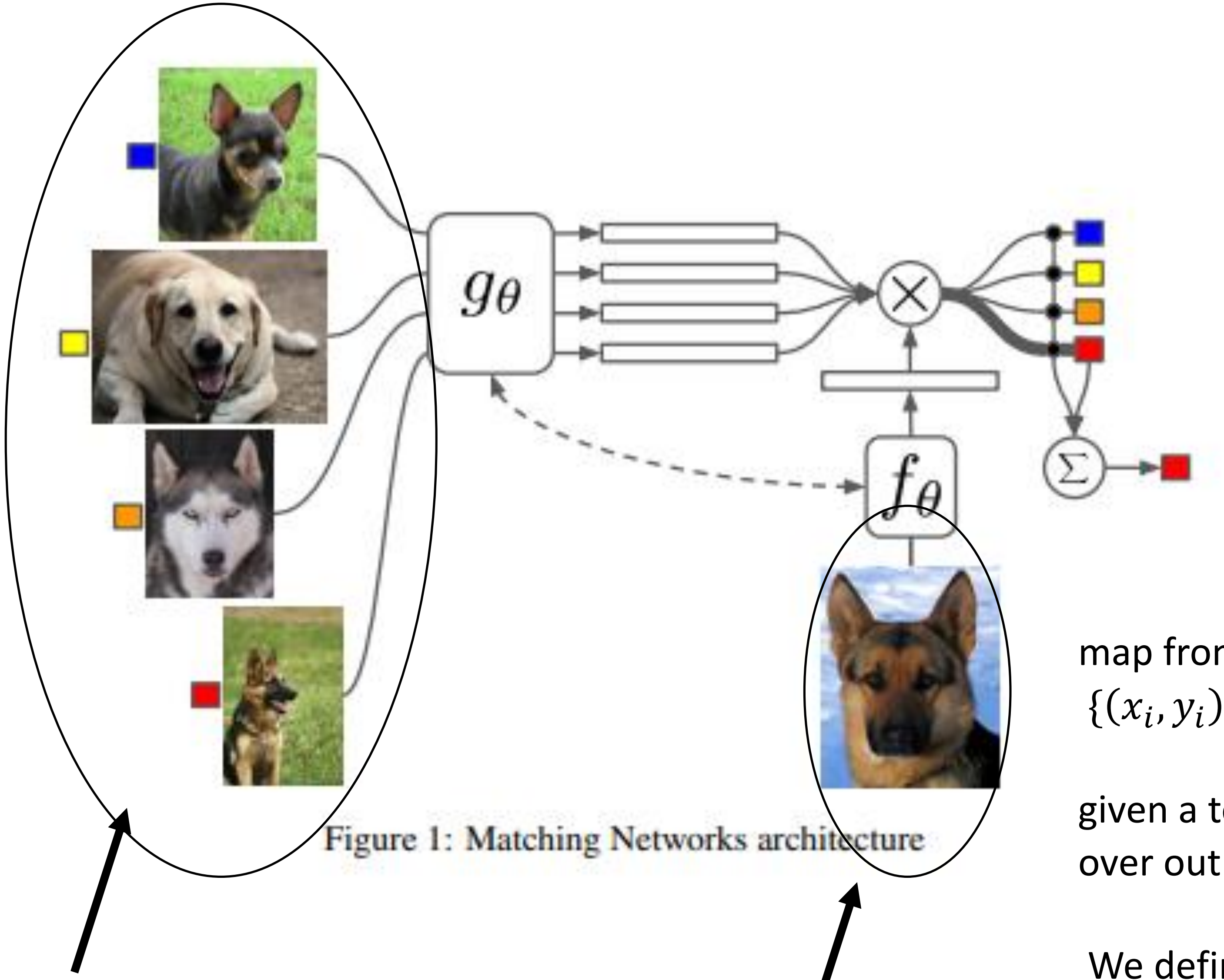


Figure 1: Matching Networks architecture

- Support set (S)

- Query set (S)

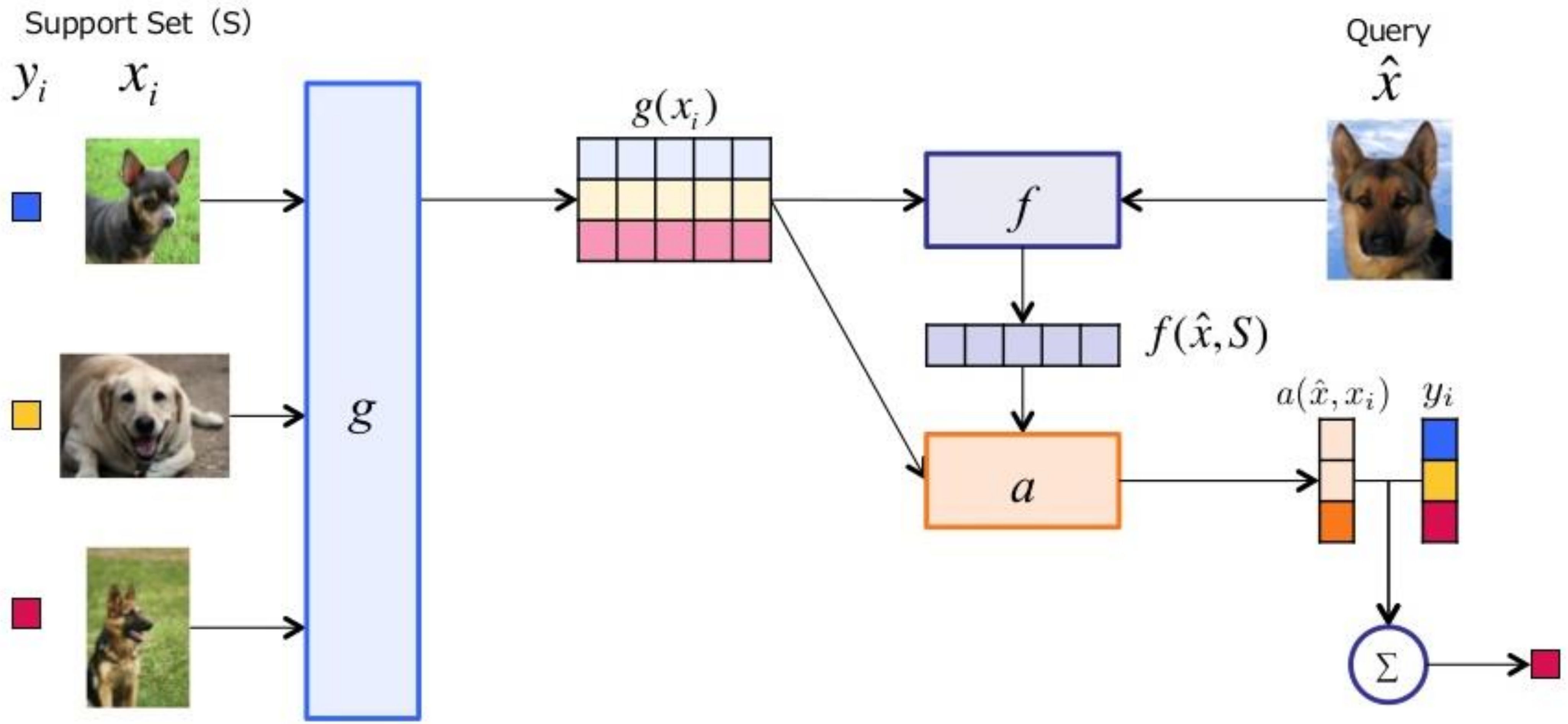
- Uses recent advances in NN with augmented memory.

- Training procedure is based on simple ML principle: test and train conditions must match (Showing only a few examples per class).

map from a **S** of **k** examples of image-label pairs $S = \{(x_i, y_i)\}_{i=1}^{k}$ to a classifier $C_s\ (\widehat{x})$

given a test example$(\widehat{x})$defines a probability distribution over outputs $(\widehat{Y})$

We define the mapping S → $C_s\ (\widehat{x})$ to be $\mathrm{P}(\hat{Y}|\hat{x}, S)$ where **P** is parameterised by a neural network.



Support Set（S）
y_i
x_i
g
$g(x_i)$
f
Query
$\hat{x}$
$f(\hat{x},S)$
a
$a(\hat{x},x_i)$
y_i
Σ

Our model in its simplest form computes $\hat{y}$ as follows:

$$\hat{y} = \sum_{i=1}^{k} a(\hat{x}, x_i) y_i$$

the attention mechanism **a** is a kernel on X × X, then is akin to a **kernel density estimator**

describes the output for a new class as a linear combination of the labels in the support set.

$$a(\hat{x}, x_i) = e^{c(f(\hat{x}), g(x_i))} / \sum_{j=1}^{k} e^{c(f(\hat{x}), g(x_j))}$$

- The Attention Kernel
  - Calculate softmax over the cosine distance between $f(\hat{x}, S)$ and $g(x_i)$
    - Similar to nearest neighbor calculation
  - Train a network using cross entropy loss

embedding functions **f** and **g** being appropriate neural networks (potentially **with f = g**) to embed $\hat{x}$ and x_i .

Loss

- This kind of loss is also related to methods such as **Neighborhood Component Analysis (NCA)**, **triplet loss** or l**arge margin nearest neighbor (Investigar).**

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2} \sqrt{\sum\limits_{i=1}^{n} B_i^2}},$$

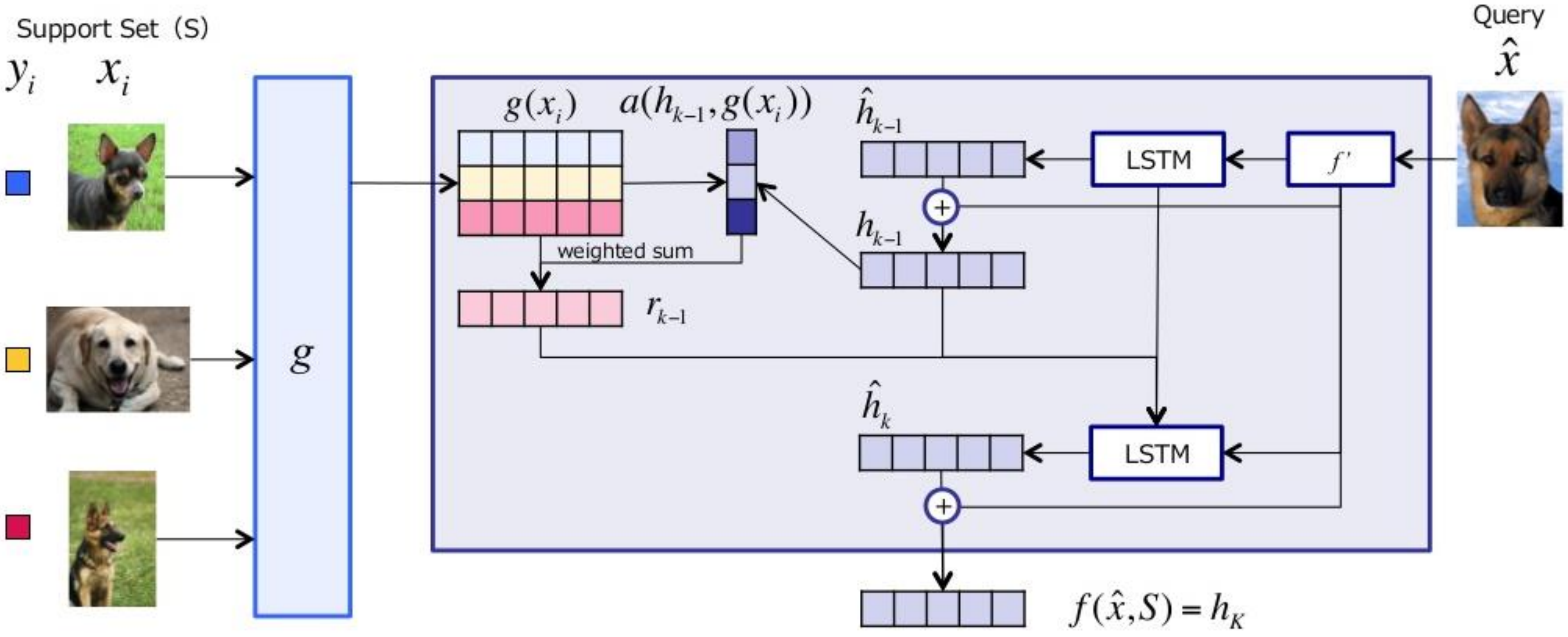
Full Context Embeddings $\boldsymbol{f}$

$$f(\hat{x}, S) = \text{attLSTM}(f'(\hat{x}), g(S), K)$$

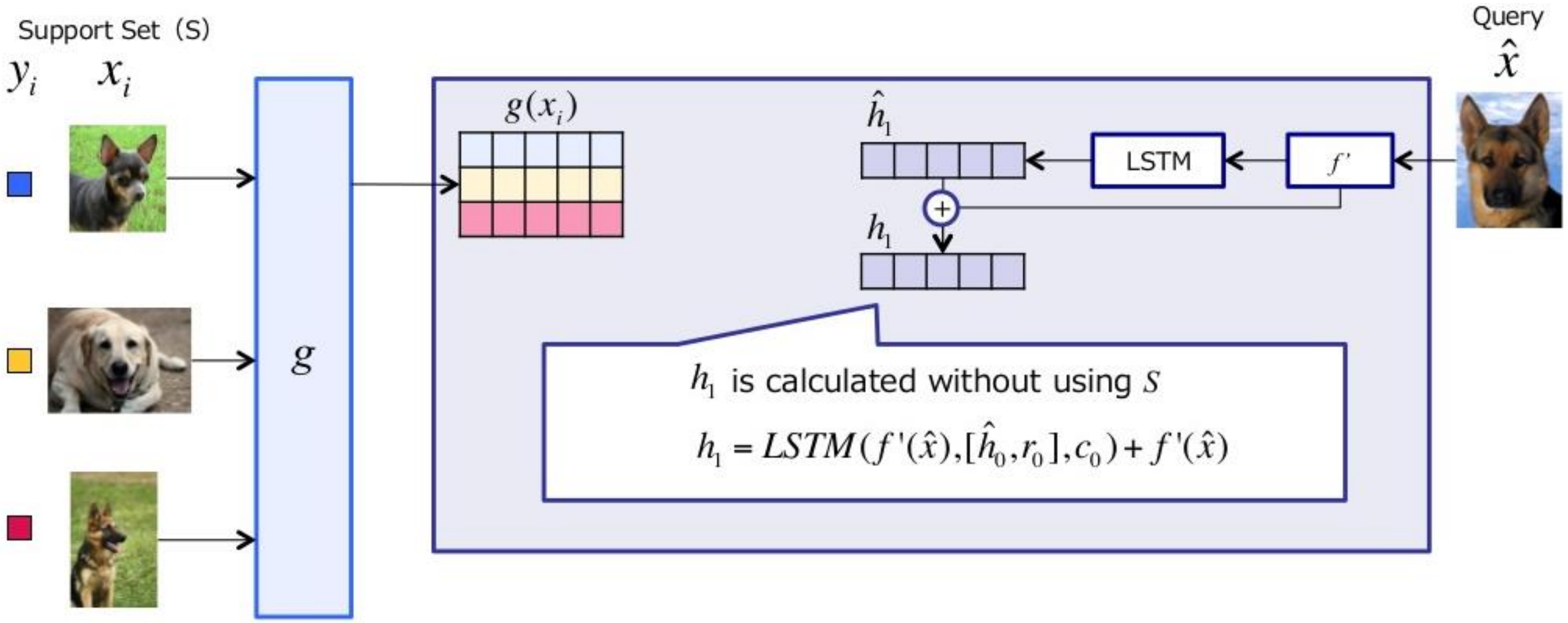
where f' is a neural network (e.g., VGG or Inception, as described in the main text). We define K to be the number of "processing" steps following work from [26] from their "Process" block. $g(S)$ represents the embedding function g applied to each element x_i from the set S.

K steps of "reads", $\text{attLSTM}(f'(\hat{x}), g(S), K) = h_K$

- **a** "content" based attention
- the softmax g(xi).
- The read-out rk−1 from g(S) is concatenated to hk−1.

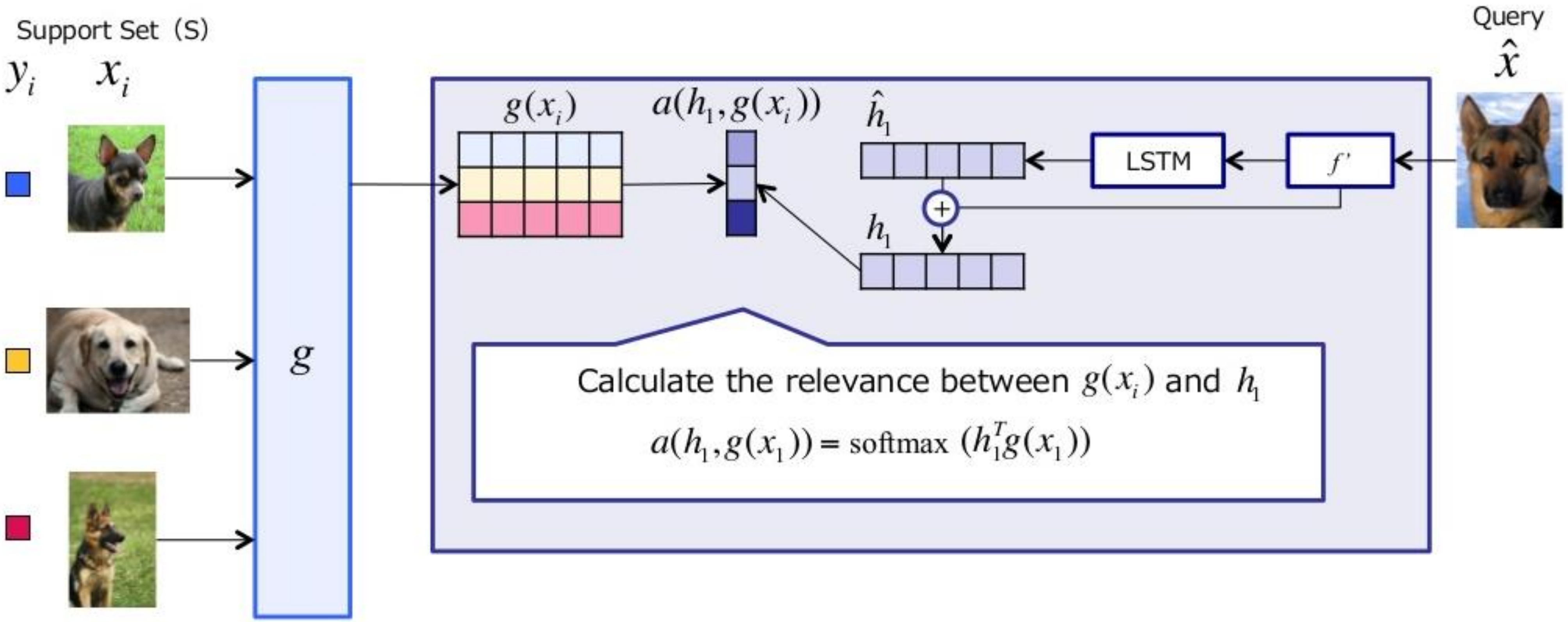


$$
\begin{aligned}
\hat{h}_k, c_k &= \mathrm{LSTM}(f'(\hat{x}), [h_{k-1}, r_{k-1}], c_{k-1}) \\
h_k &= \hat{h}_k + f'(\hat{x}) \\
r_{k-1} &= \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i))g(x_i) \\
a(h_{k-1}, g(x_i)) &= \mathrm{softmax}(h_{k-1}^T g(x_i))
\end{aligned}
$$

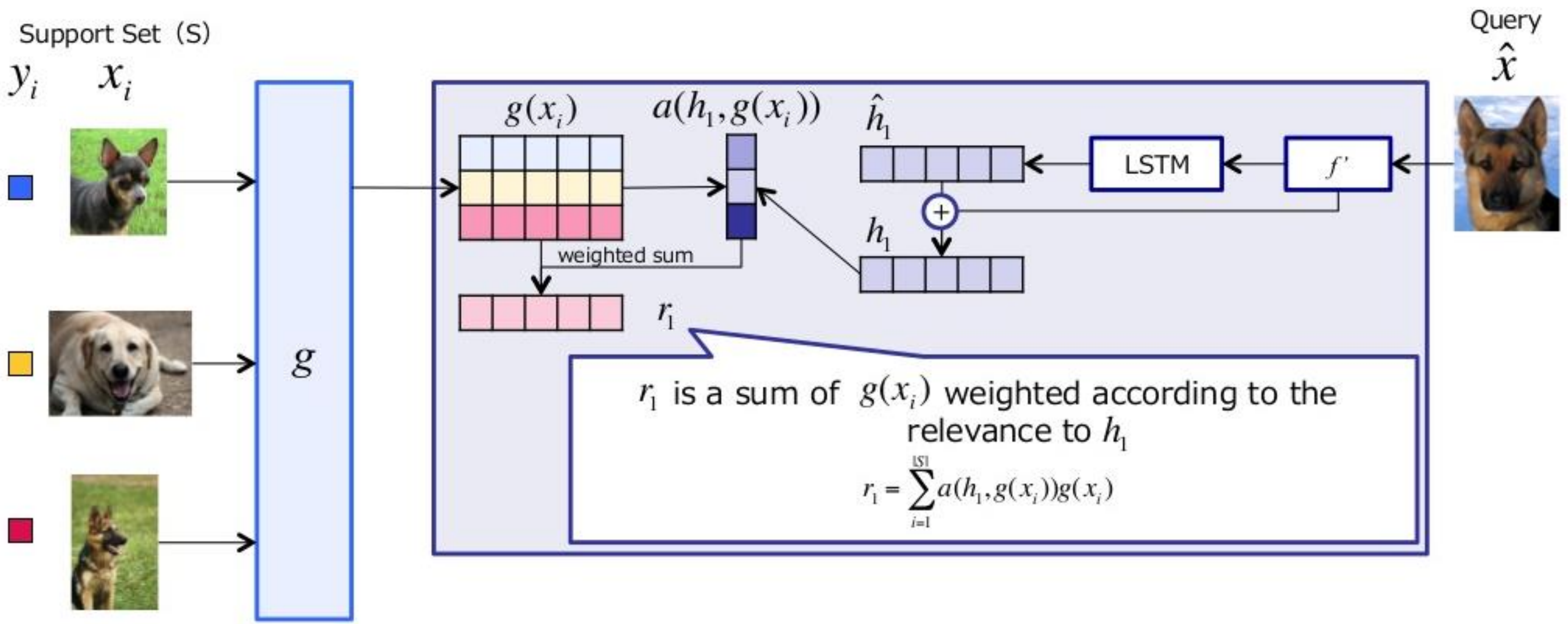


$$\begin{aligned} \hat{h}_k, c_k &= \text{LSTM}(f'(\hat{x}), [h_{k-1}, r_{k-1}], c_{k-1}) \\ h_k &= \hat{h}_k + f'(\hat{x}) \end{aligned}$$

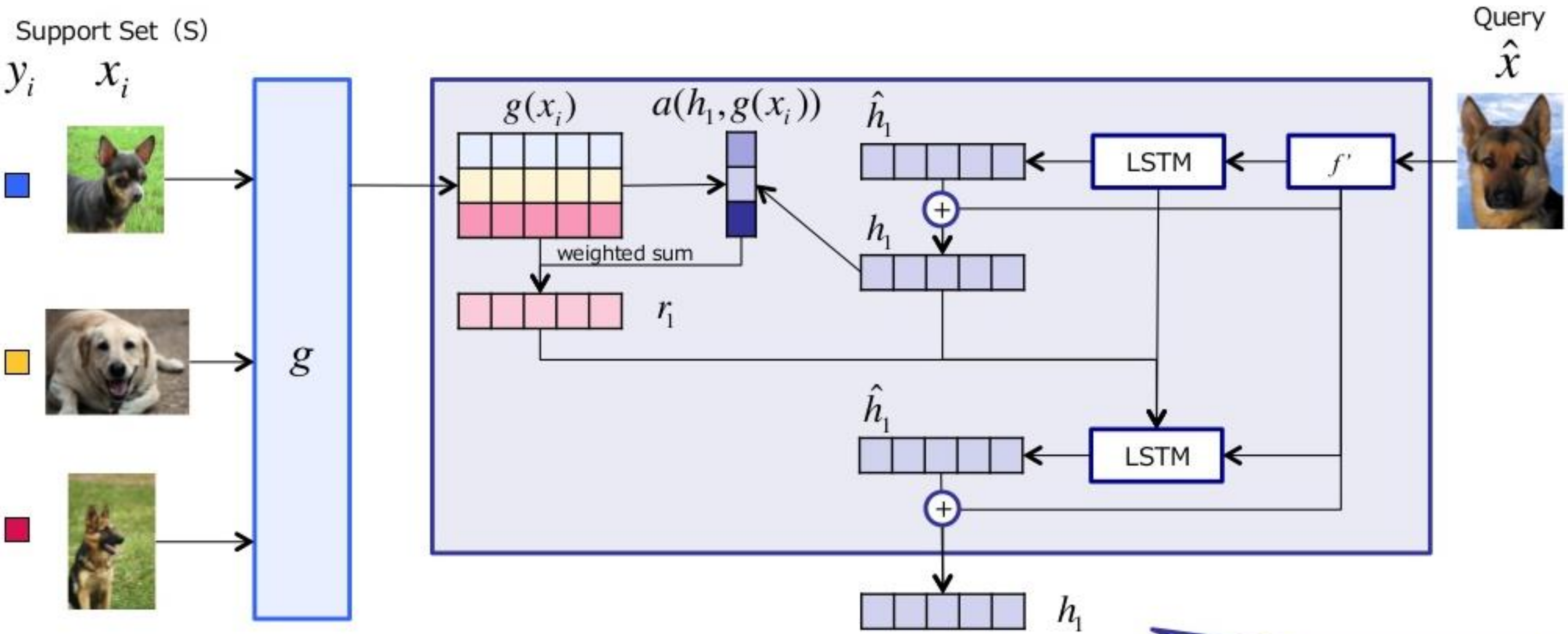
$$r_{k-1} = \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i))g(x_i)$$



$$a(h_{k-1}, g(x_i)) \quad = \quad \text{softmax}(h_{k-1}^T g(x_i))$$



Support Set (S)
y_i
x_i
g
$g(x_i)$
$a(h_1, g(x_i))$
$\hat{h}_1$
LSTM
f'
Query
$\hat{x}$
h_1
weighted sum
r_1
r_1 is a sum of $g(x_i)$ weighted according to the relevance to h_1
$r_1 = \sum_{i=1}^{|S|} a(h_1, g(x_i)) g(x_i)$

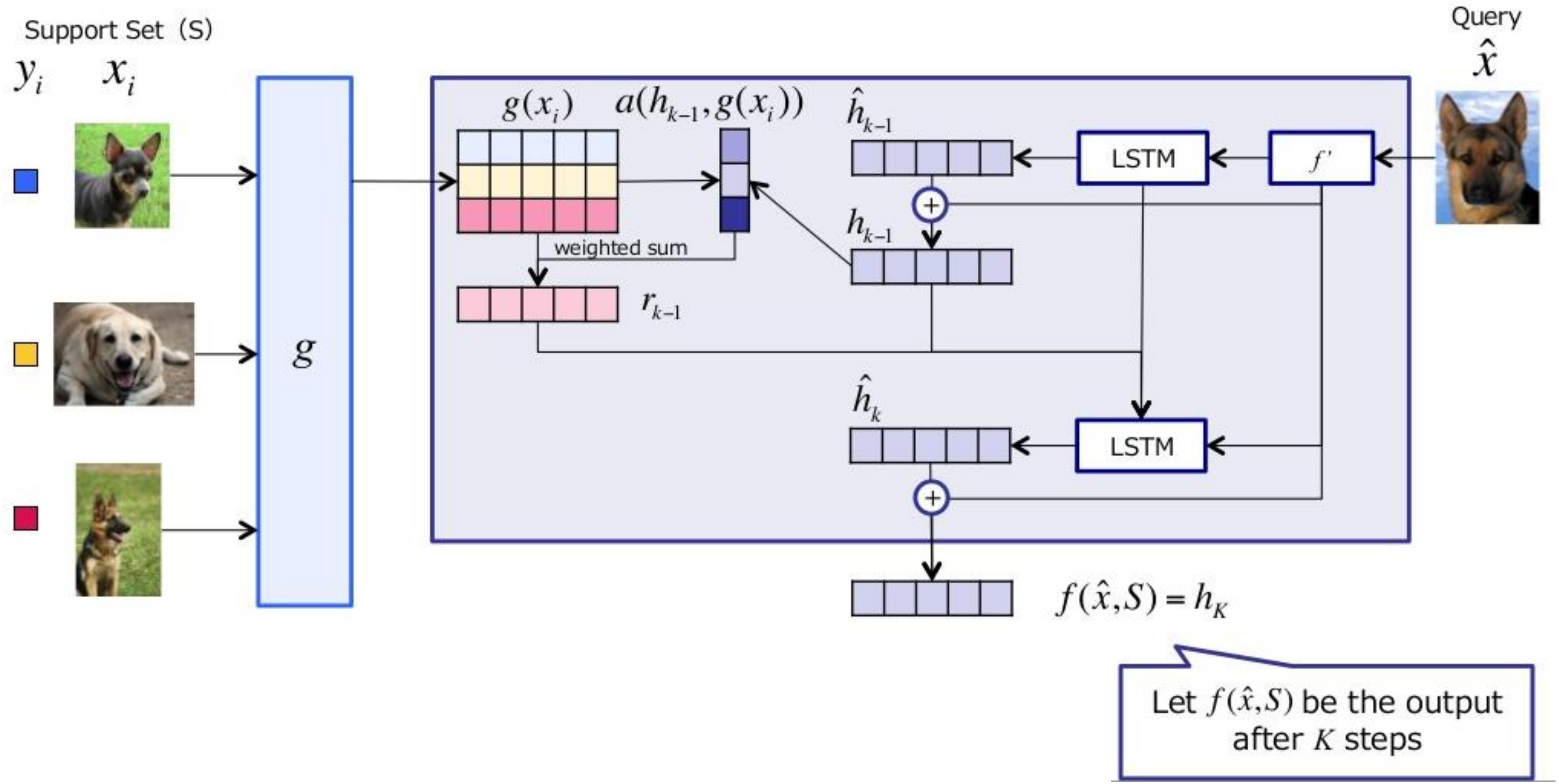


$$
\begin{aligned}
\hat{h}_k, c_k &= \mathrm{LSTM}(f'(\hat{x}), [h_{k-1}, r_{k-1}], c_{k-1}) \\
h_k &= \hat{h}_k + f'(\hat{x})
\end{aligned}
$$

$$
r_{k-1} = \sum_{i=1}^{|S|} a(h_{k-1}, g(x_i))g(x_i)
$$

$|S|$

h_1 is calculated using S



Support Set (S)
y_i
x_i
g
$g(x_i)$
$a(h_{k-1}, g(x_i))$
weighted sum
r_{k-1}
$\hat{h}_{k-1}$
h_{k-1}
LSTM
f'
Query
$\hat{x}$
$\hat{h}_k$
LSTM
$f(\hat{x}, S) = h_K$
Let $f(\hat{x}, S)$ be the output after K steps

Full Context Embeddings g

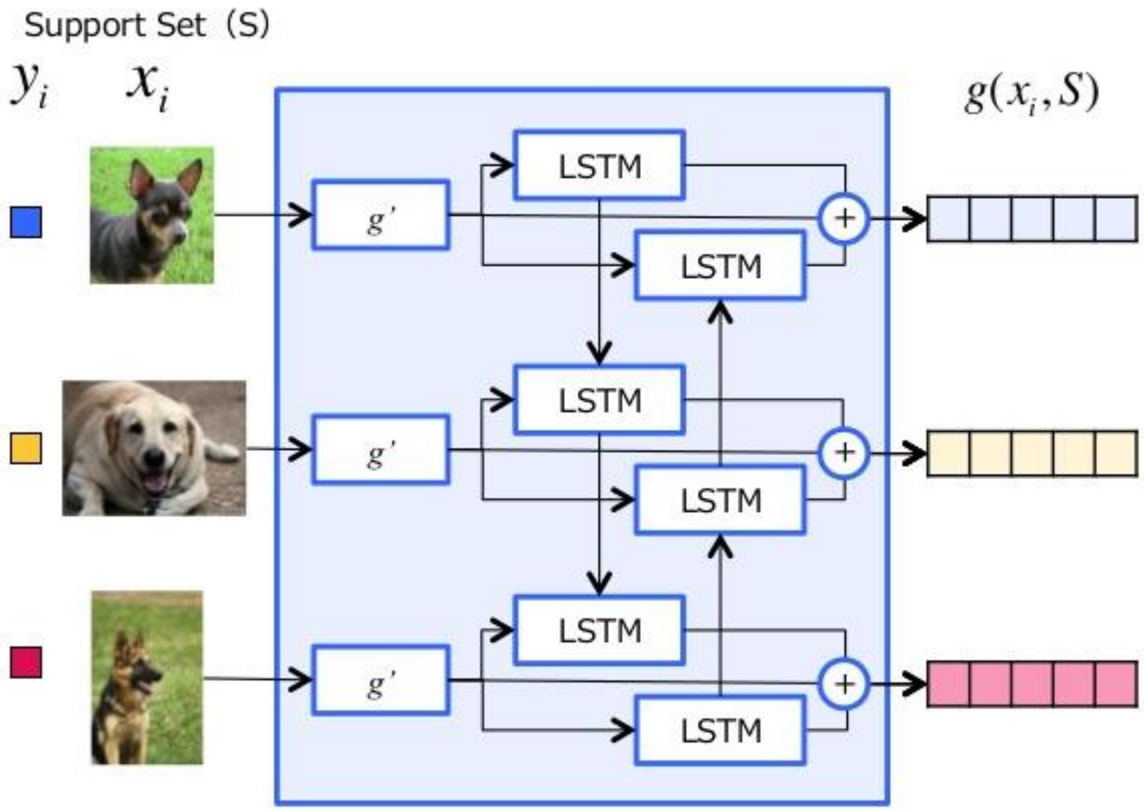
- The Fully Conditional Embedding g
  - Embed x_i in consideration of S

Then we define $g(x_i, S) = \vec{h}_i + \overleftarrow{h}_i + g'(x_i)$ with:

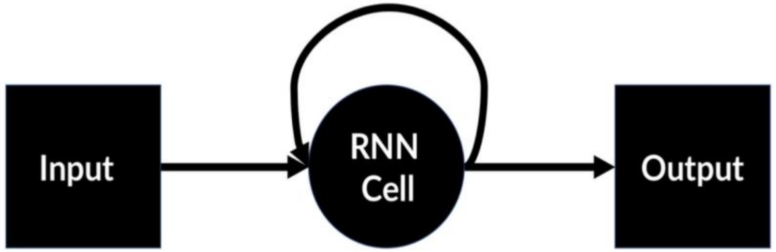
$$\vec{h}_i, \vec{c}_i = \text{LSTM}(g'(x_i), \vec{h}_{i-1}, \vec{c}_{i-1})$$
$$\overleftarrow{h}_i, \overleftarrow{c}_i = \text{LSTM}(g'(x_i), \overleftarrow{h}_{i+1}, \overleftarrow{c}_{i+1})$$

g': neural network (e.g., VGG or Inception)

$\overleftarrow{h}$ starts from $i = |S|$

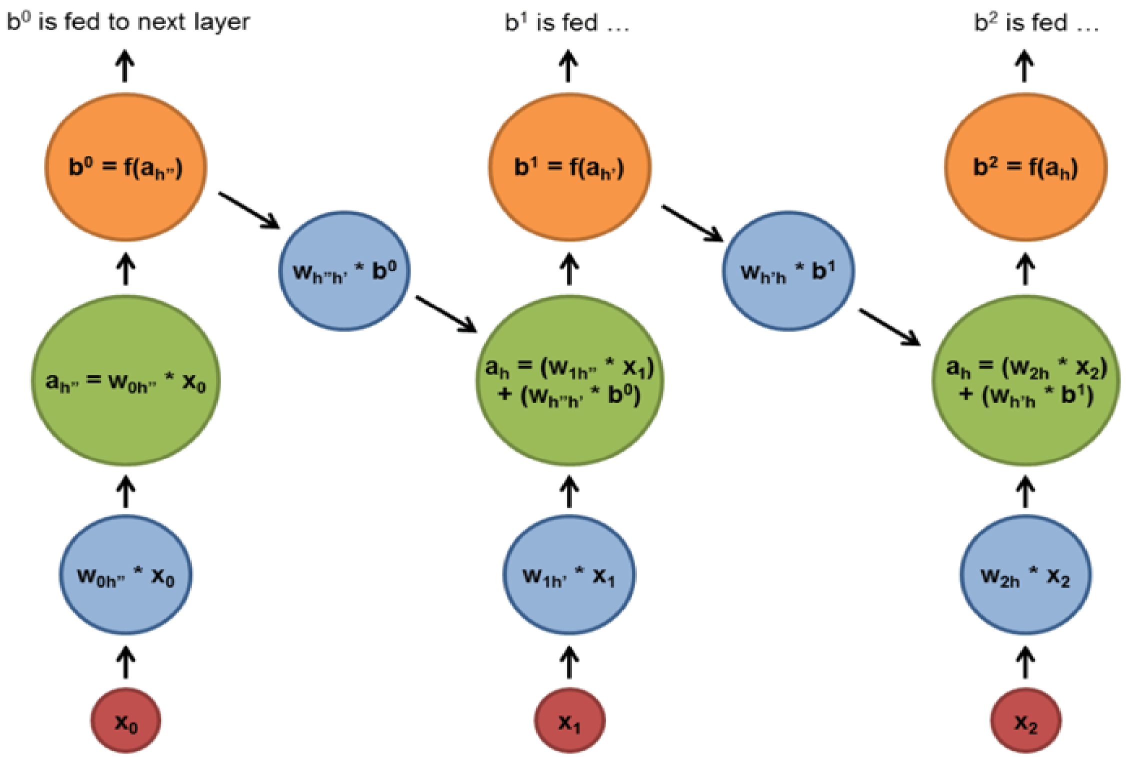


RNN



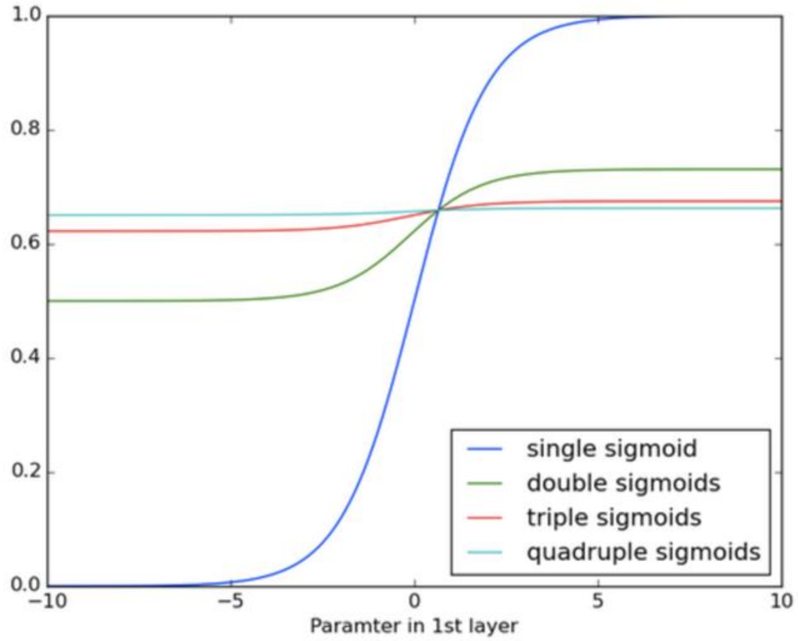
The scheme of an RNN

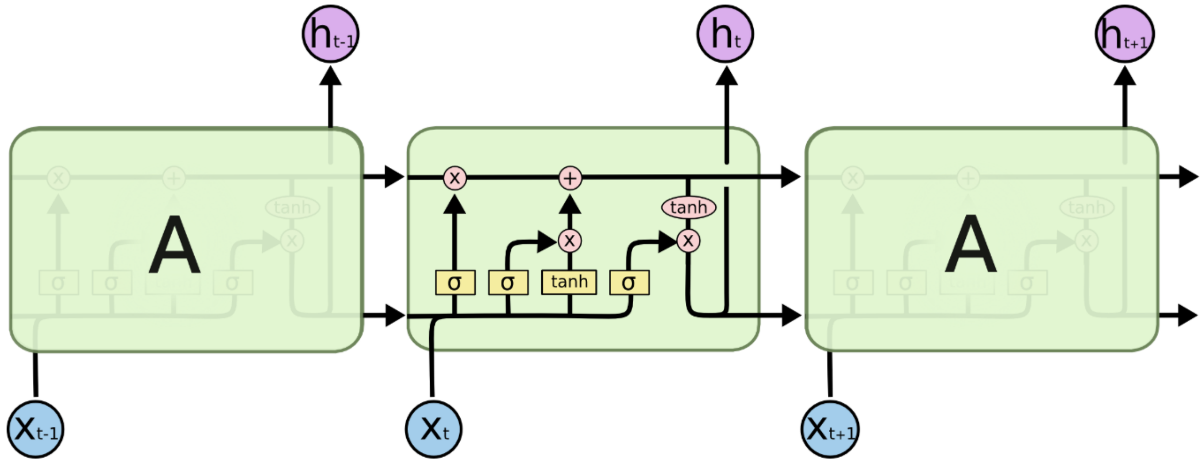
$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right)$$



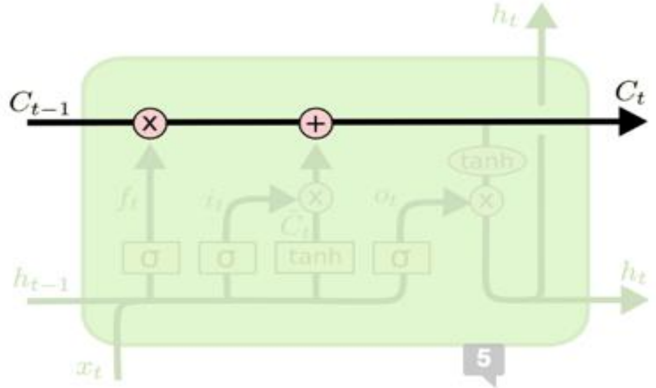
RNN LSTM

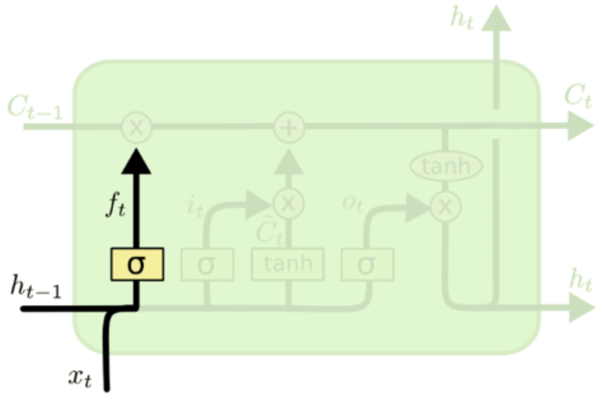
In direct response to the vanishing gradients problem of simple RNNs, the **Long Short-Term Memory (LSTM)** layer was invented. This layer performs much better at longer time series.



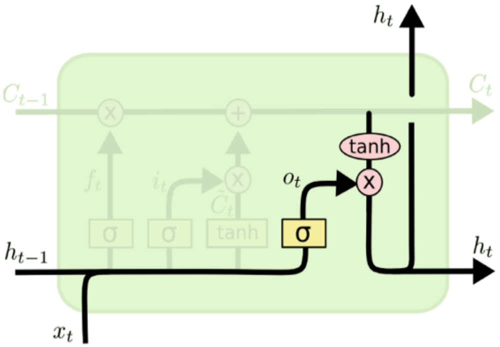


The repeating module in an LSTM contains four interacting layers.

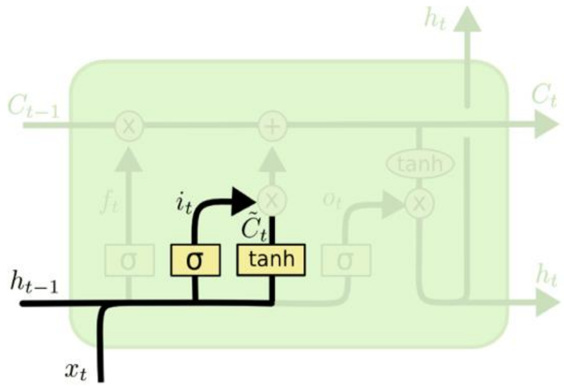




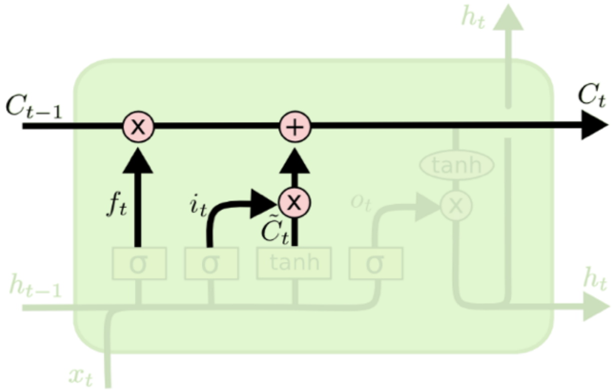
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$



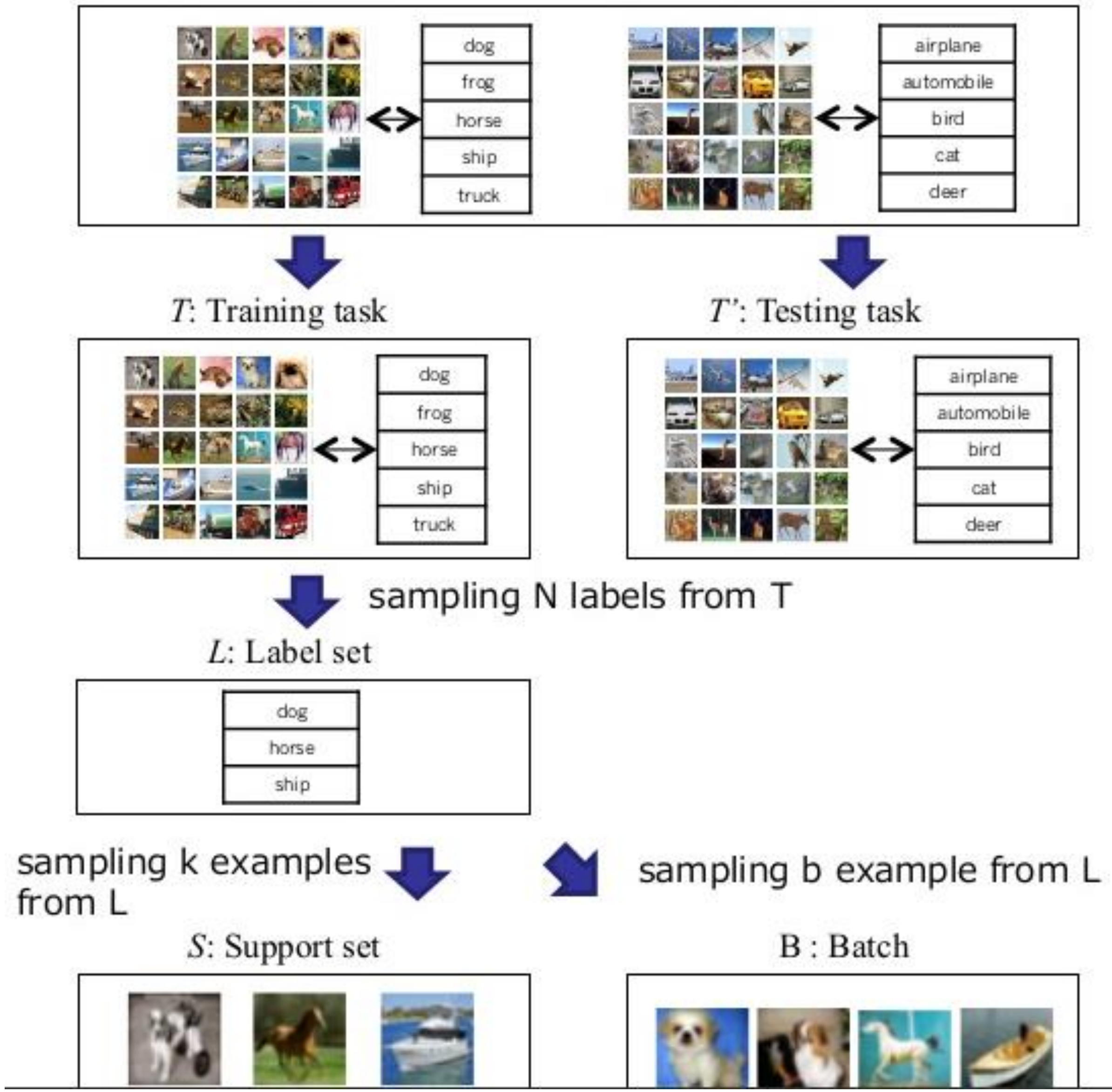
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Training Strategy

- Our model has to perform well with support sets $\grave{S}$ which contain classes never seen during training.
- **T** as distribution over possible label sets **L.**
- T to uniformly weight all data sets of up to a few unique classes (e.g., 5) with a few examples per class (e.g., up to 5
- A label set **L** sampled from a task **T**, **L ~ T**, will typically have 5 to 25 examples.
- “episode” - compute gradients and update our model
- sample **L** from **T** (e.g., **L** could be the label set **{cats, dogs}**). Use **L** to sample the support set **S** and a batch **B** (i.e., both **S** and **B** are labelled examples of cats and dogs).
- The Matching Net is then trained to minimise the **error predicting the labels in the batch B conditioned on the support set S**.

$$\theta = \arg\max_{\theta} E_{L\sim T}\left[E_{S\sim L, B\sim L}\left[\sum_{(x,y)\in B} \log P_{\theta}\left(y|x,S\right)\right]\right]$$

Train a classifier through one-shot learning



Results

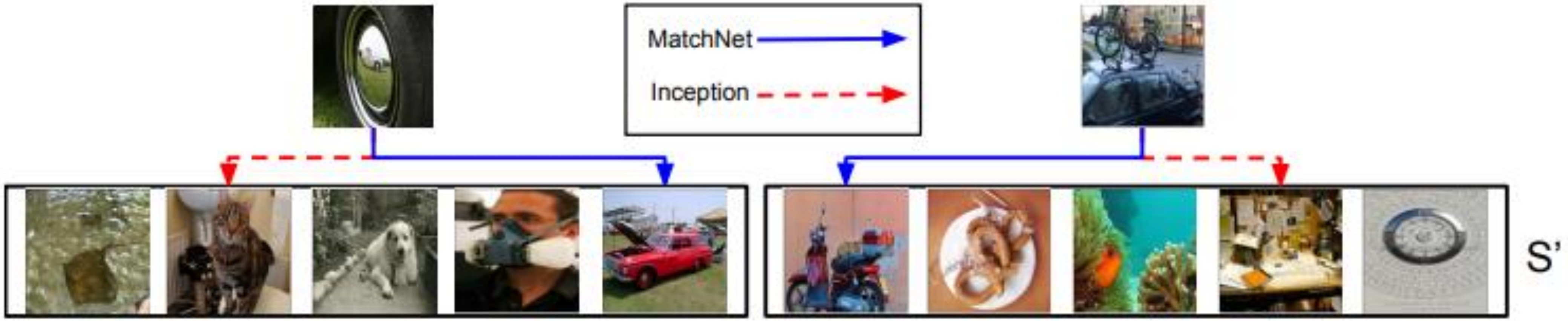


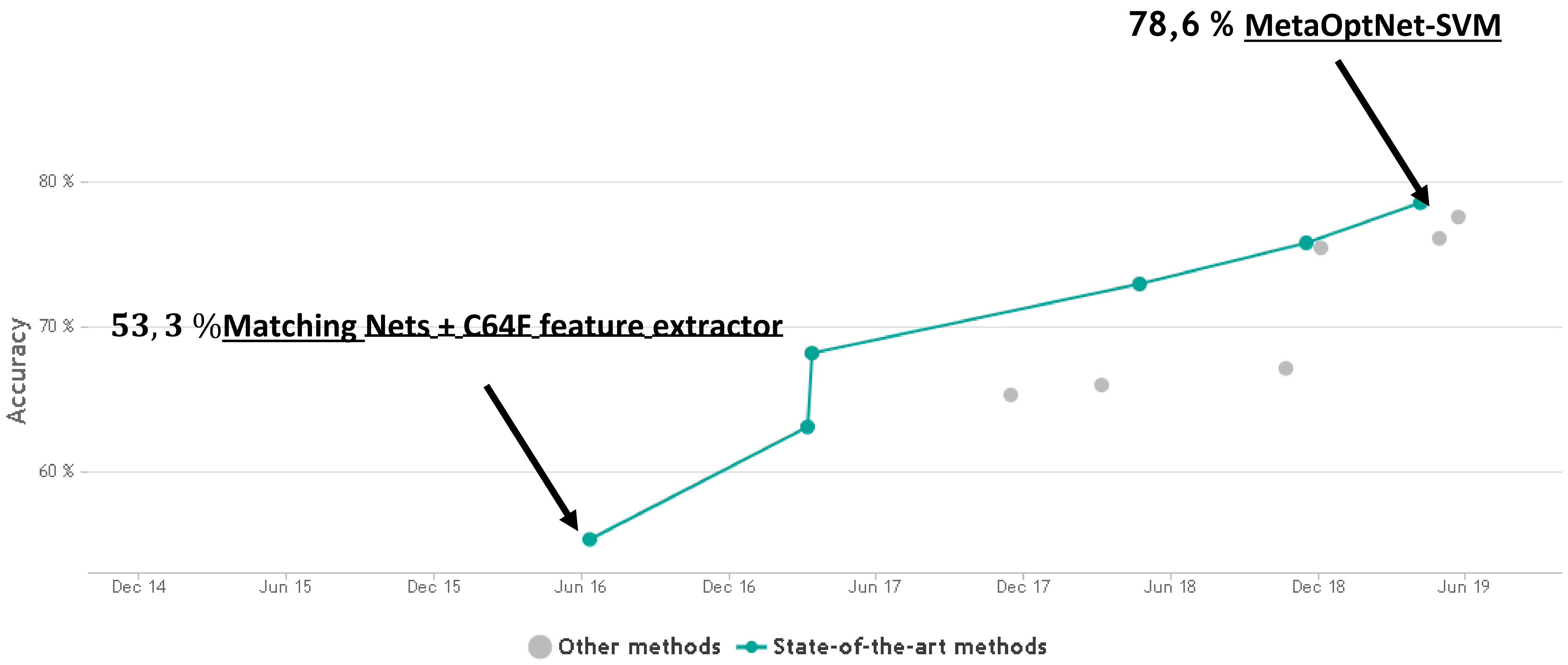
Figure 2: Example of two 5-way problem instance on ImageNet. The images in the set S' contain classes never seen during training. Our model makes far less mistakes than the Inception baseline.

Table 2: Results on *mini*ImageNet.

Model	Matching Fn	Fine Tune	5-way Acc 1-shot	5-way Acc 5-shot
PIXELS	Cosine	N	23.0%	26.6%
BASELINE CLASSIFIER	Cosine	N	36.6%	46.0%
BASELINE CLASSIFIER	Cosine	Y	36.2%	52.2%
BASELINE CLASSIFIER	Softmax	Y	38.4%	51.2%
MATCHING NETS (OURS)	Cosine	N	41.2%	56.2%
MATCHING NETS (OURS)	Cosine	Y	42.4%	58.0%
MATCHING NETS (OURS)	Cosine (FCE)	N	44.2%	57.0%
MATCHING NETS (OURS)	Cosine (FCE)	Y	**46.6%**	**60.0%**

Table 3: Results on full ImageNet on *rand* and *dogs* one-shot tasks. Note that $\neq L_{rand}$ and $\neq L_{dogs}$ are sets of classes which are seen during training, but are provided for completeness.

Model	Matching Fn	Fine Tune	ImageNet 5-way 1-shot Acc L_{rand}	$\neq L_{rand}$	L_{dogs}	$\neq L_{dogs}$
PIXELS	Cosine	N	42.0%	42.8%	41.4%	43.0%
INCEPTION CLASSIFIER	Cosine	N	87.6%	92.6%	**59.8%**	90.0%
MATCHING NETS (OURS)	Cosine (FCE)	N	**93.2%**	**97.0%**	58.8%	**96.4%**
INCEPTION ORACLE	Softmax (Full)	Y (Full)	$\approx 99\%$	$\approx 99\%$	$\approx 99\%$	$\approx 99\%$



78,6 % MetaOptNet-SVM
53, 3 %Matching Nets + C64F feature extractor
80 %
70 %
60 %
Accuracy
Dec 14
Jun 15
Dec 15
Jun 16
Dec 16
Jun 17
Dec 17
Jun 18
Dec 18
Jun 19
Other methods
State-of-the-art methods

Codigo

- https://github.com/AntreasAntoniou/MatchingNetworks