Domain Adaptation - 2

Vinay P. Namboodiri
Department of Computer Science and Engineering
IIT Kanpur

Problem

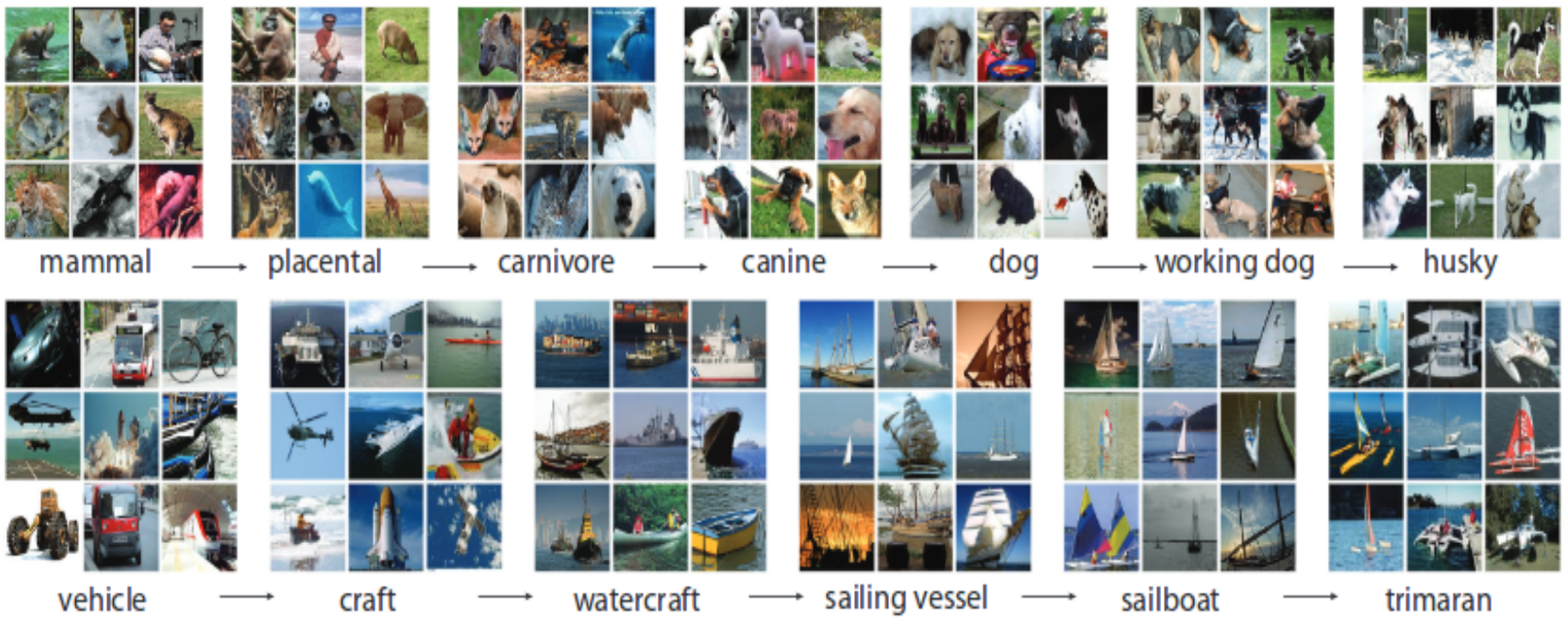


Figure : Typical Computer Vision Dataset

Generally, training and test instances are chosen from the same dataset and hence they are from same probability distribution. Also labels are free of noise, objects are centered and background clutter is less.

Source: Anant Raj

Challenge

Figure : Real World Computer Vision Dataset

Real world data is noisy. Multiple instances of different object can be present in an image and also usually much more clutter is there.

Source: Anant Raj

Domain Adaptation for Detection

- The domain adaptation algorithms have so far been demonstrated for classification

- Detection presents its own set of challenges

Domain Adaptation for Detection

Contrary to image recognition we have to localize and classify.

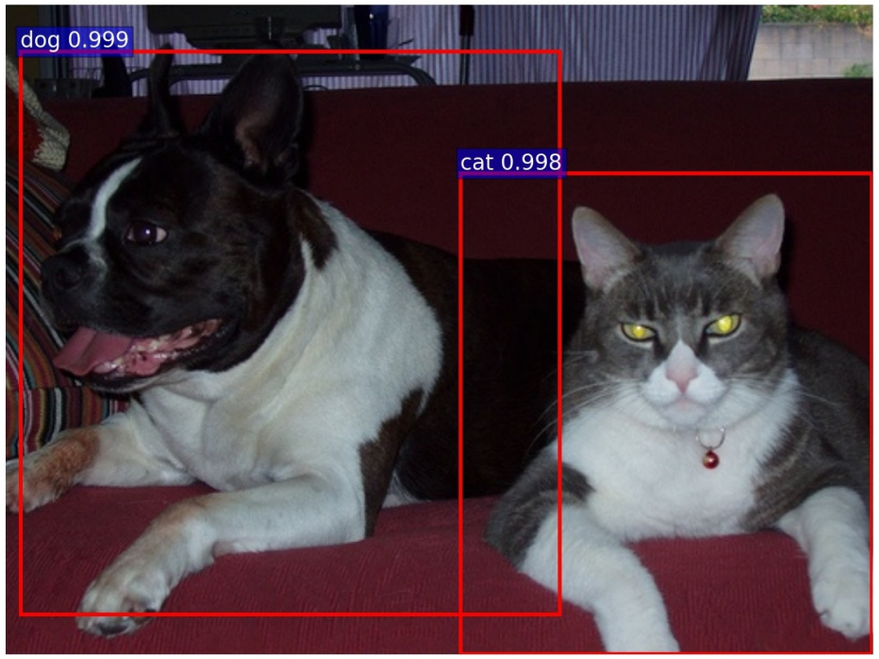


Figure: Example of Object Detection

An initial approach for RCNN

- Unsupervised domain adaptation for detection
- Using subspace alignment over an initial RCNN detector
- We want to align subspaces only for the bounding boxes of the objects for source and target
- However, we assume we do not have target bounding boxes

Subspace Alignment based Domain Adaptation for RCNN Detector
A. Raj, V. Namboodiri, T. Tuytelaars
BMVC 2015

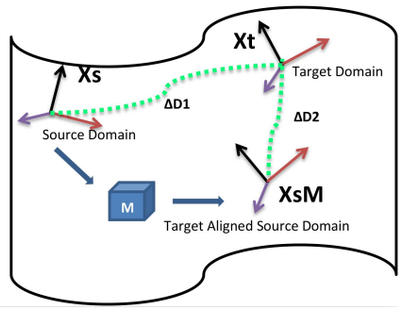
Our approach

- Train RCNN detector on source subspace
- Obtain bounding boxes on source by using detector (avoid non-maxima suppression)
    - This works better than ground truth bounding boxes
- Obtain predicted target bounding boxes using detector trained on source
- Learn the subspace alignment using the obtained source and target bounding boxes

Our approach

- Project Source samples onto target subspace using aligned subspace

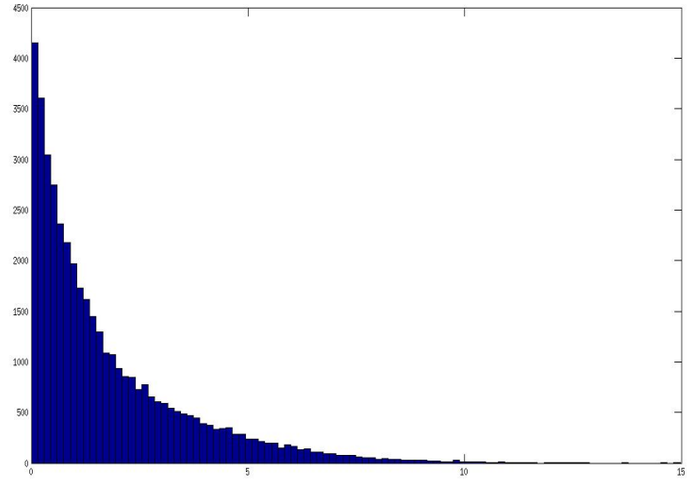
for each class $i \in Object\ Class$ **do**
$ProjectMat(i) \leftarrow SubspaceAlign(X_{source}(i), X_{target}(i))$
end for



- Train RCNN detector again with the source samples projected onto target subspace

Our approach

- PASCAL VOC 2012 and Microsoft COCO dataset
- Statistical dissimilarity between the dataset



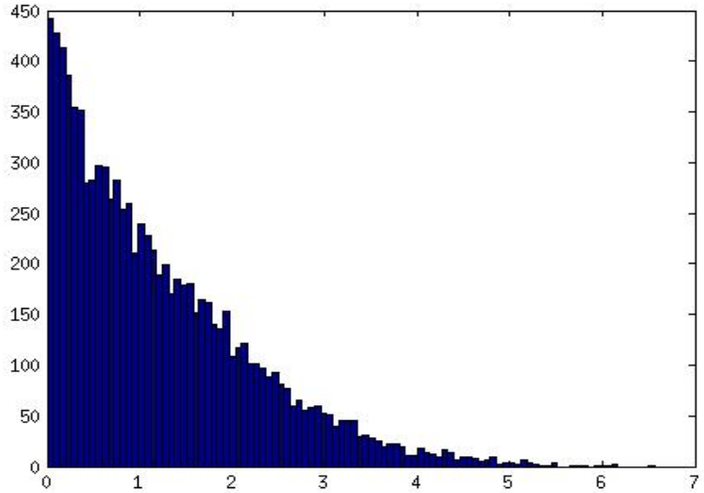


Figure : Histogram of scores. Fig 1 is for CoCo dataset and fig 2 for PASCAL VOC. Scores is taken along x-axis and no. of object region with that score along y-axis

Results

No.	class	RCNN- No Transform	RCNN - Full Transform	Proposed	DPM
1	plane	36.72	35.44	**40.1**	35.1
2	bicycle	21.26	18.95	**23.28**	1.9
3	bird	12.50	12.37	**13.63**	3.7
4	boat	10.45	8.8	**10.61**	2.3
5	bottle	8.75	**11.46**	8.11	7
6	bus	37.47	38.12	40.64	**45.4**
7	car	20.6	20.4	**22.5**	18.3
8	cat	42.4	43.6	**45.6**	8.6
9	chair	**9.6**	6.3	8.8	6.3
10	cow	23.28	20.40	**25.3**	17
11	table	15.9	14.9	**17.3**	4.8
12	dog	28.42	**32.72**	31.3	5.8
13	horse	30.7	31.11	32.9	**35.3**

Results

No.	class	RCNN- No Transform	RCNN - Full Transform	Proposed	DPM
14	motorbike	31.2	29.05	**34.6**	25.4
15	person	27.8	28.8	**30.9**	17.5
16	plant	12.65	7.34	**13.7**	4.1
17	sheep	19.99	21.04	**22.4**	14.5
18	sofa	14.6	8.4	**15.5**	9.6
19	train	39.2	38.4	**41.64**	31.7
20	tv	28.6	26.4	**29.9**	27.9
	Mean AP	23.60	22.7	**25.43**	16.9

Table : Domain adaption detection result on validation set of COCO dataset.

Results

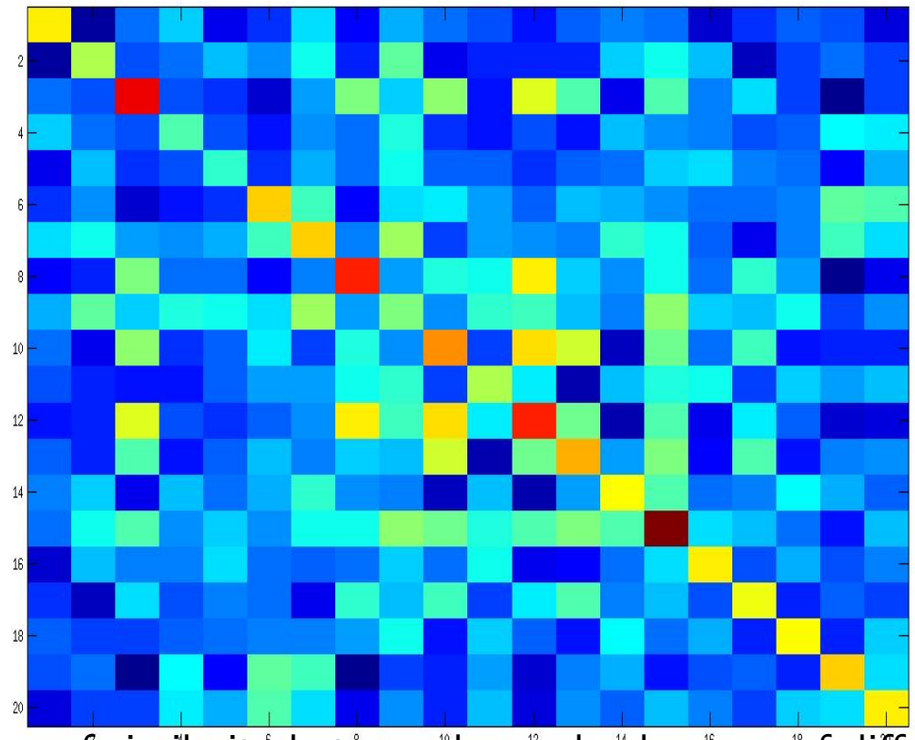
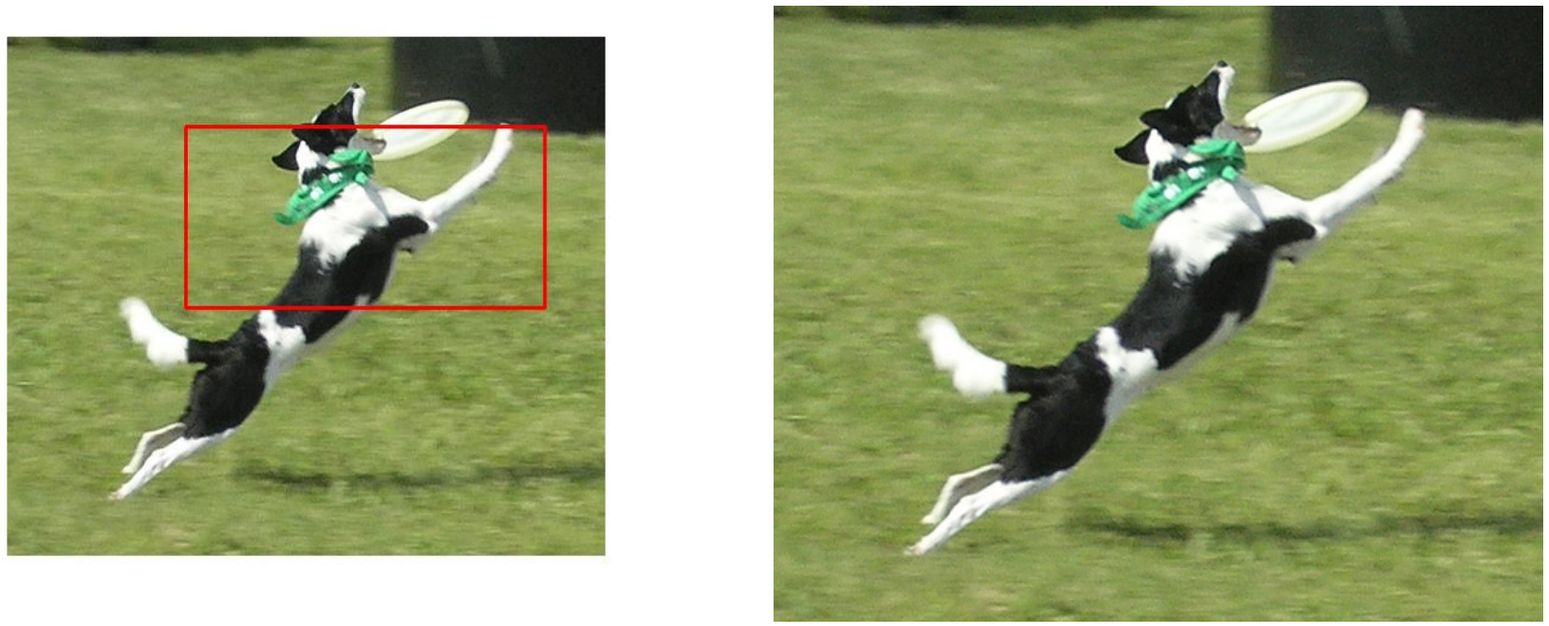


Figure : Color map of similarity between learned subspaces of different categories in source and target dataset

Results

Figure : Few extremely good detections using our proposed method

Results



(a) Our method (b) RCNN

Figure : Examples where RCNN fails to perform but our method performs well

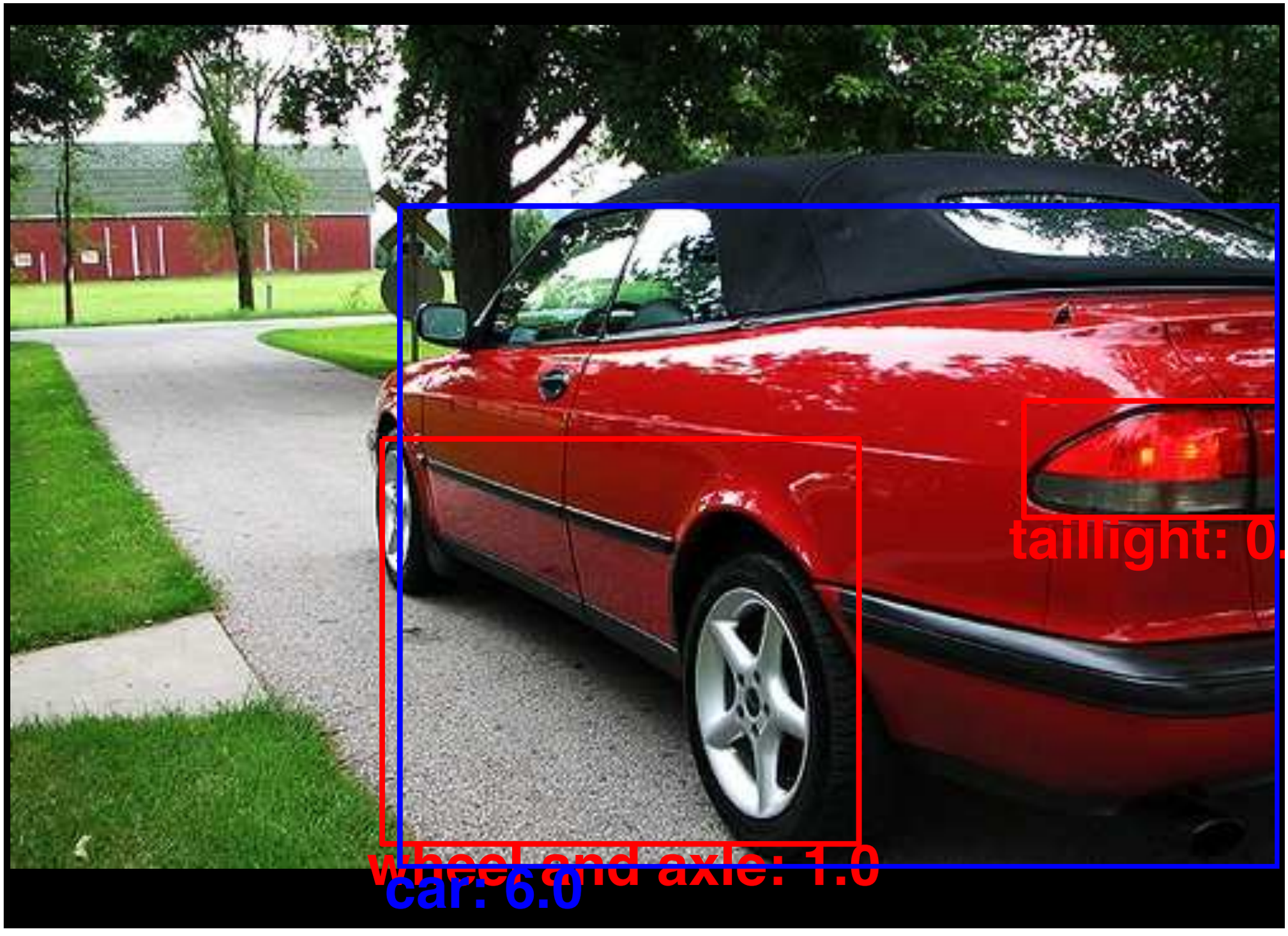
LSDA: Large Scale Detection through Adaptation

Judy Hoffman[◇], **Sergio Guadarrama**[◇], **Eric Tzeng**[◇], **Ronghang Hu**[▽], **Jeff Donahue**[◇],
[◇]EECS, UC Berkeley, [▽]EE, Tsinghua University
{jhoffman, sguada, tzeng, jdonahue}@eecs.berkeley.edu
hrh11@mails.tsinghua.edu.cn

Ross Girshick[◇], **Trevor Darrell**[◇], **Kate Saenko**[△]
[◇]EECS, UC Berkeley, [△]CS, UMass Lowell
{rbg, trevor}@eecs.berkeley.edu, saenko@cs.uml.edu

NIPS 2014


car


taillight: 0.
wheel and axle: 1.0
car: 6.0

Core Idea

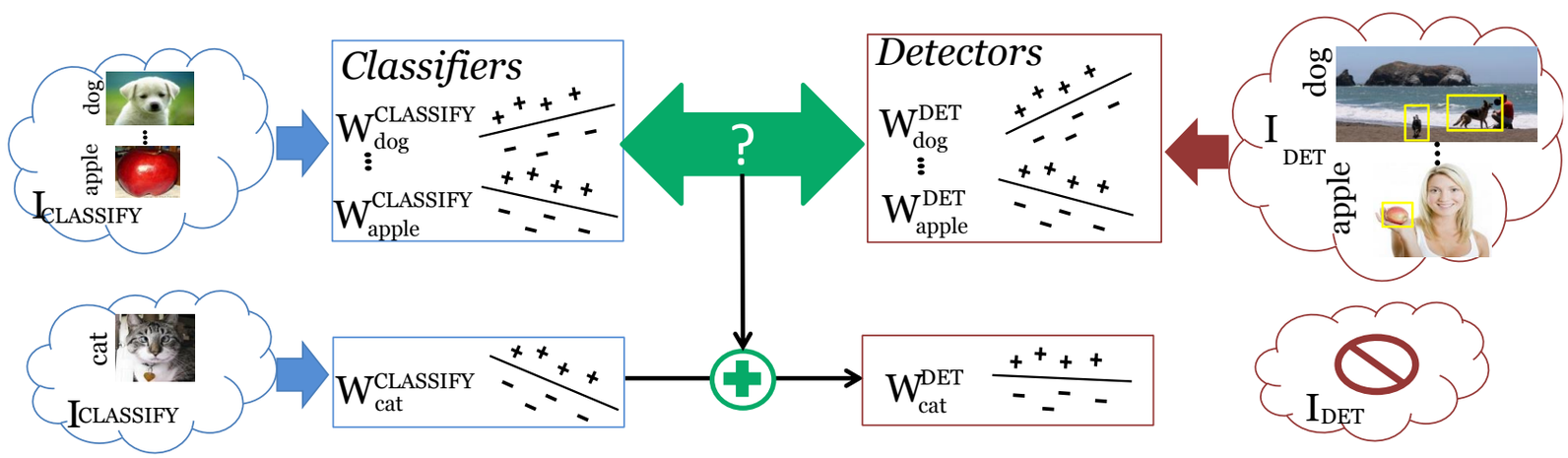
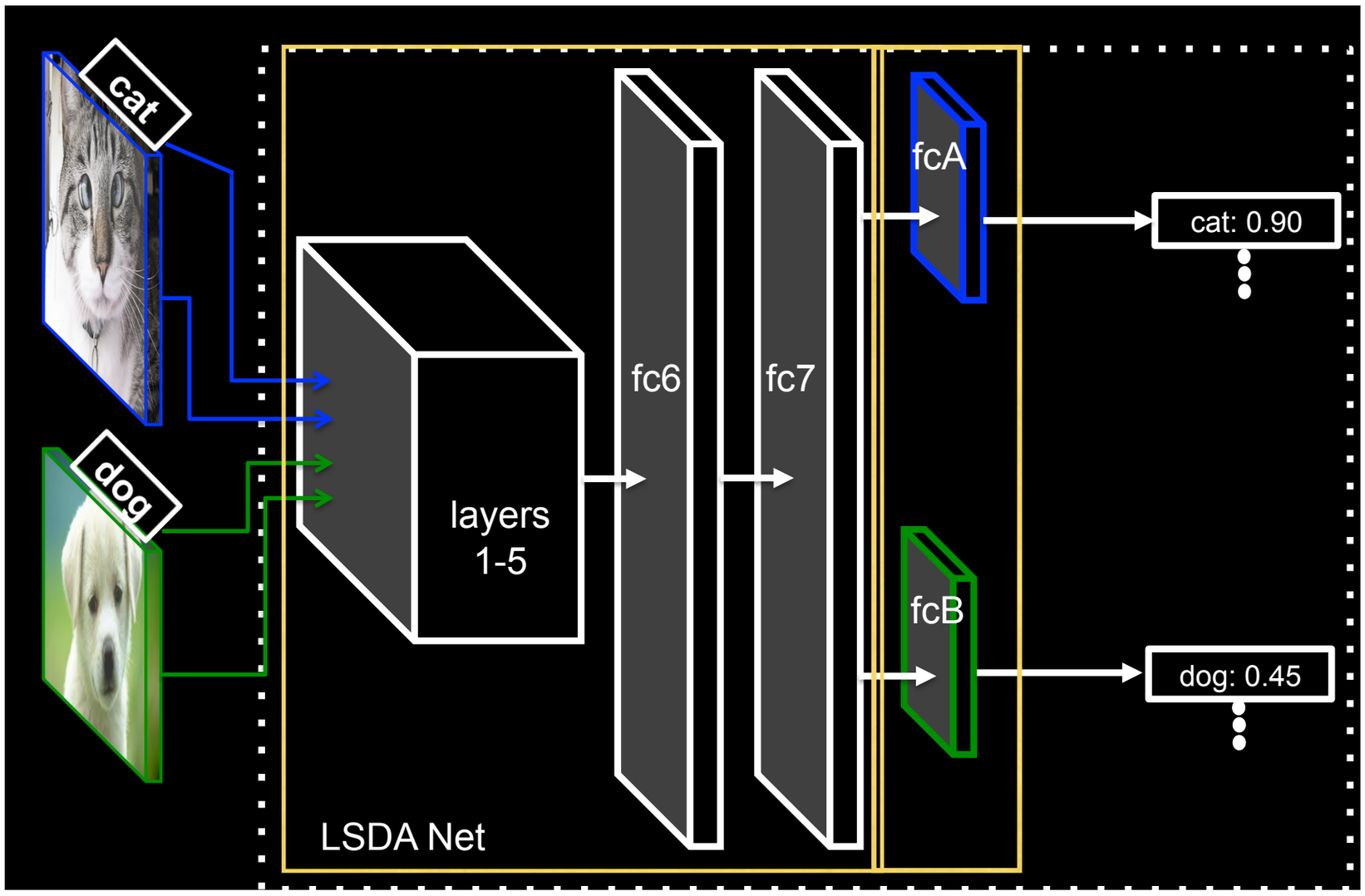


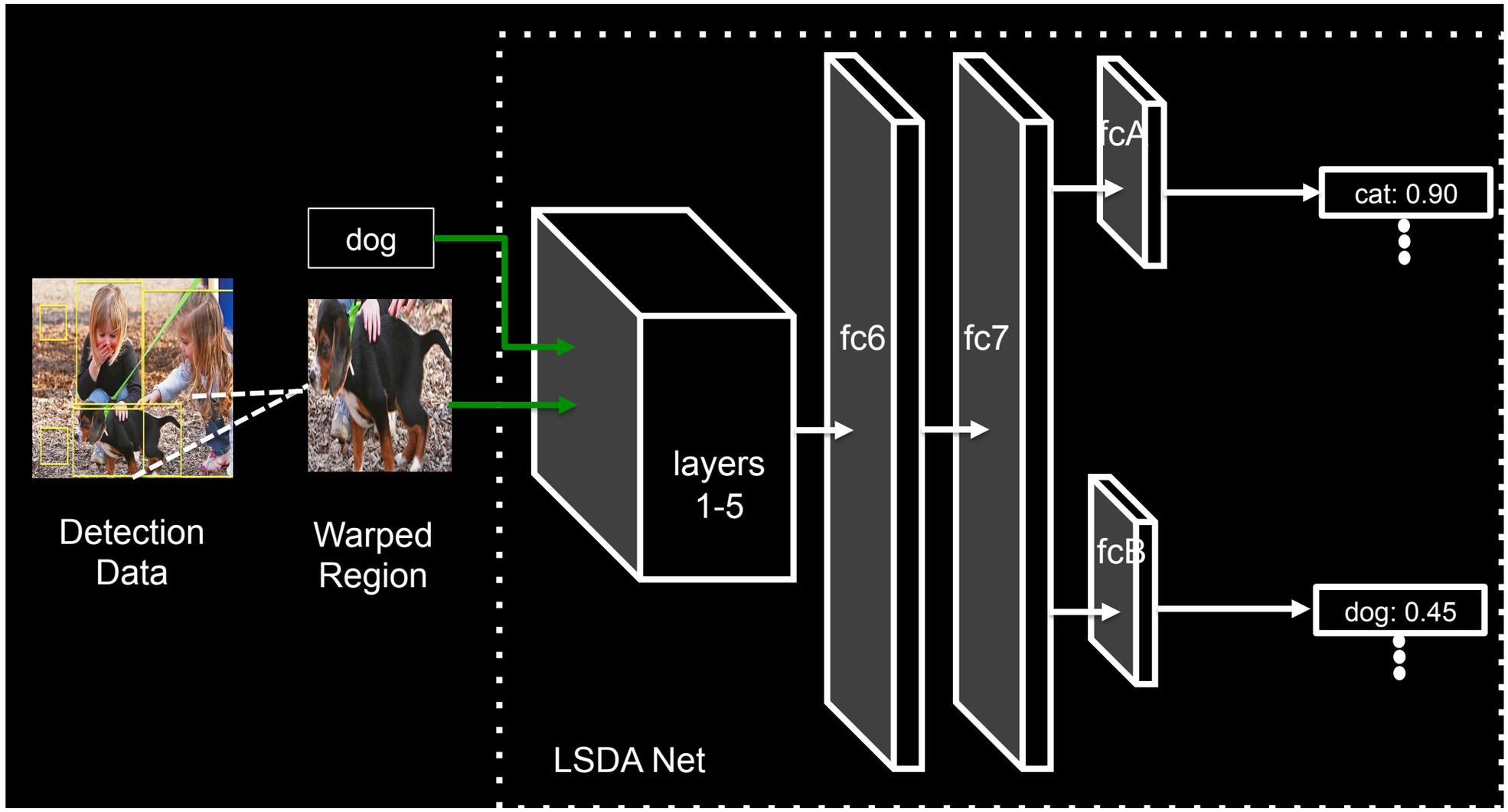
Figure 1: The core idea is that we can learn detectors (weights) from labeled classification data (left), for a wide range of classes. For some of these classes (top) we also have detection labels (right), and can learn detectors. But what can we do about the classes with classification data but no detection data (bottom)? Can we learn something from the paired relationships for the classes for which we have both classifiers and detectors, and transfer that to the classifier at the bottom to make it into a detector?

LSDA



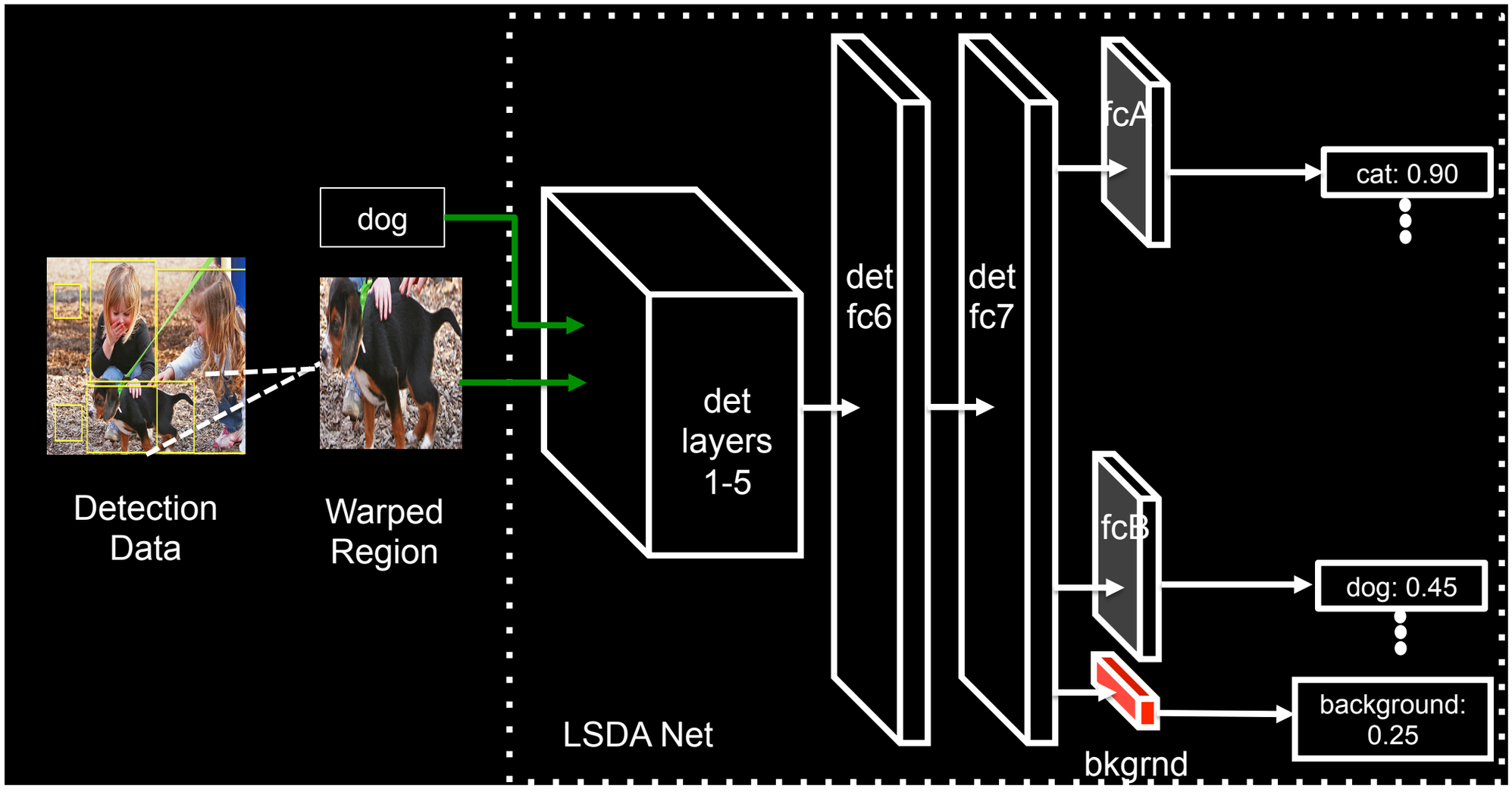
Classification Net

LSDA



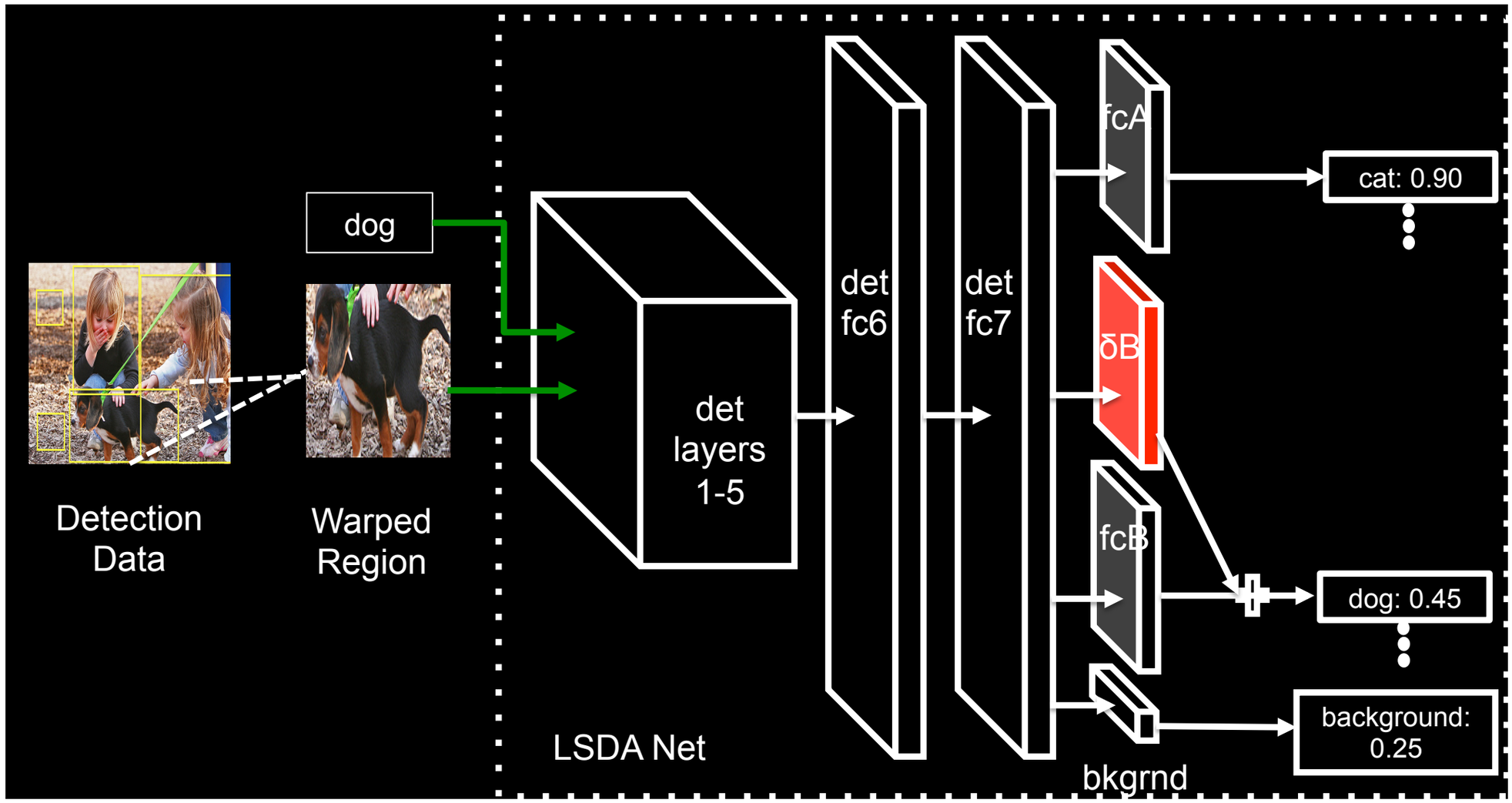
Fine-tune with Detection Data

LSDA



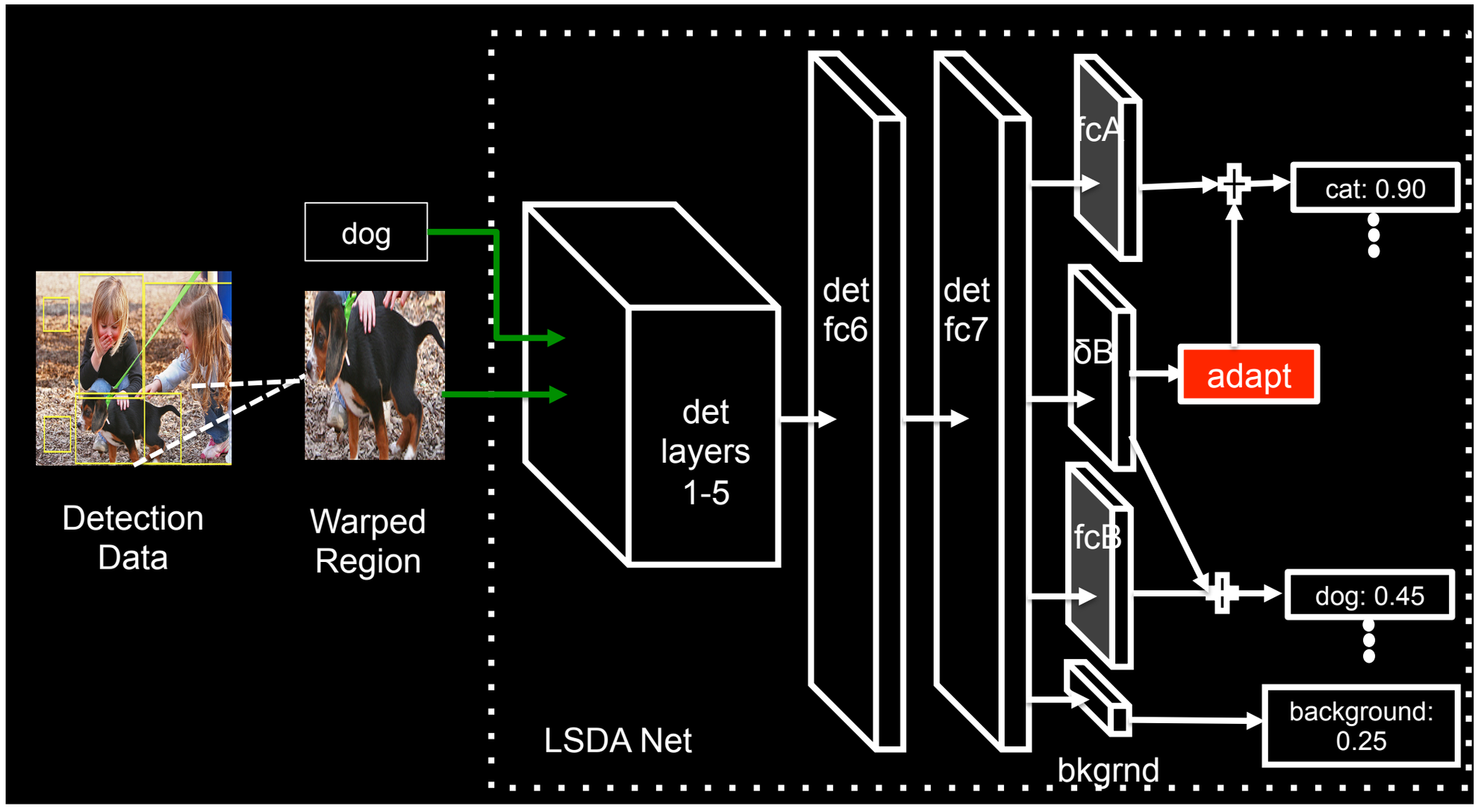
Learn a Background class detector

LSDA



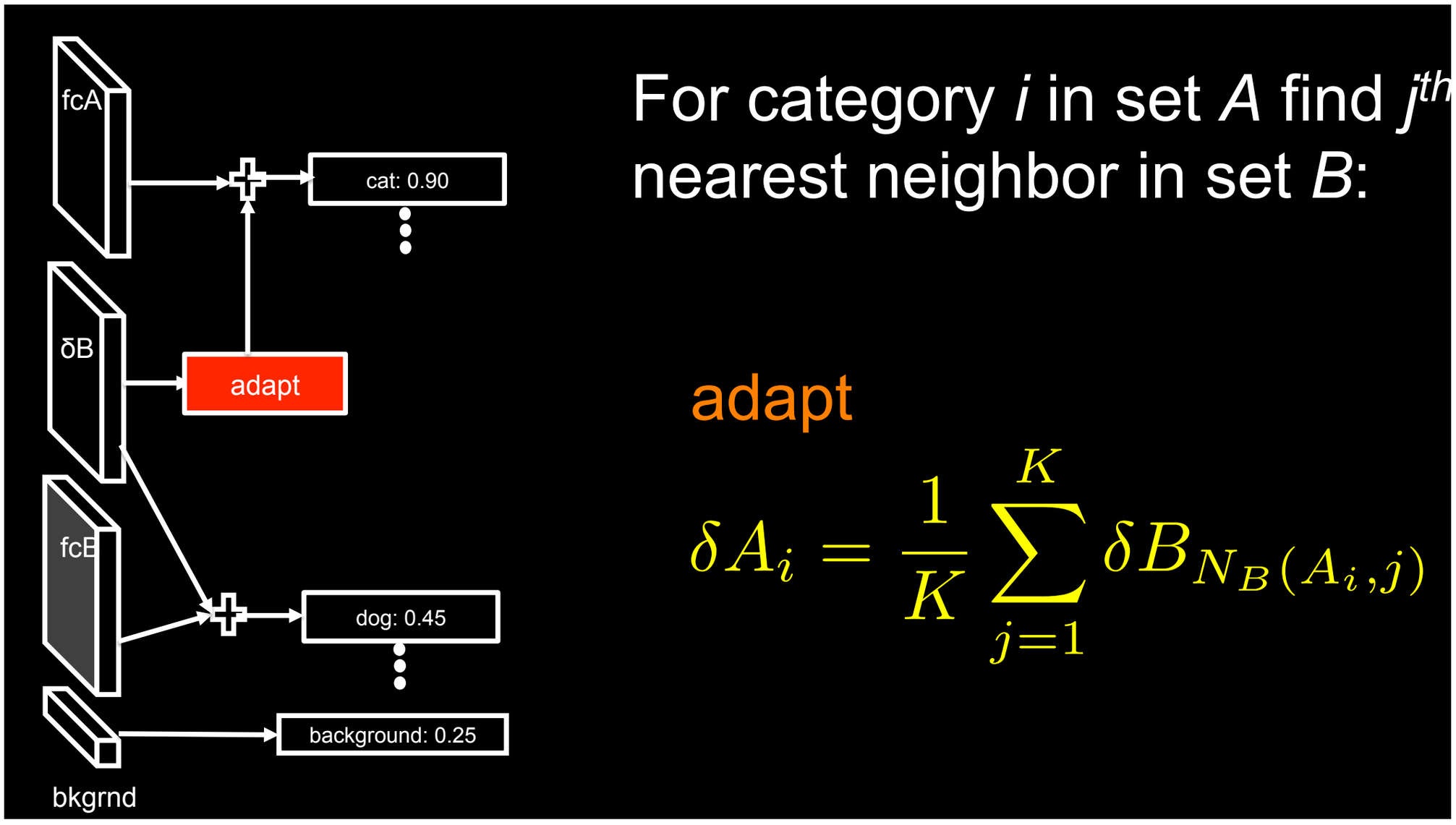
Fine-tune Category Specific Parameters

LSDA



Adapt output layer for held-out classes

LSDA



Adaptation of output layer

LSDA

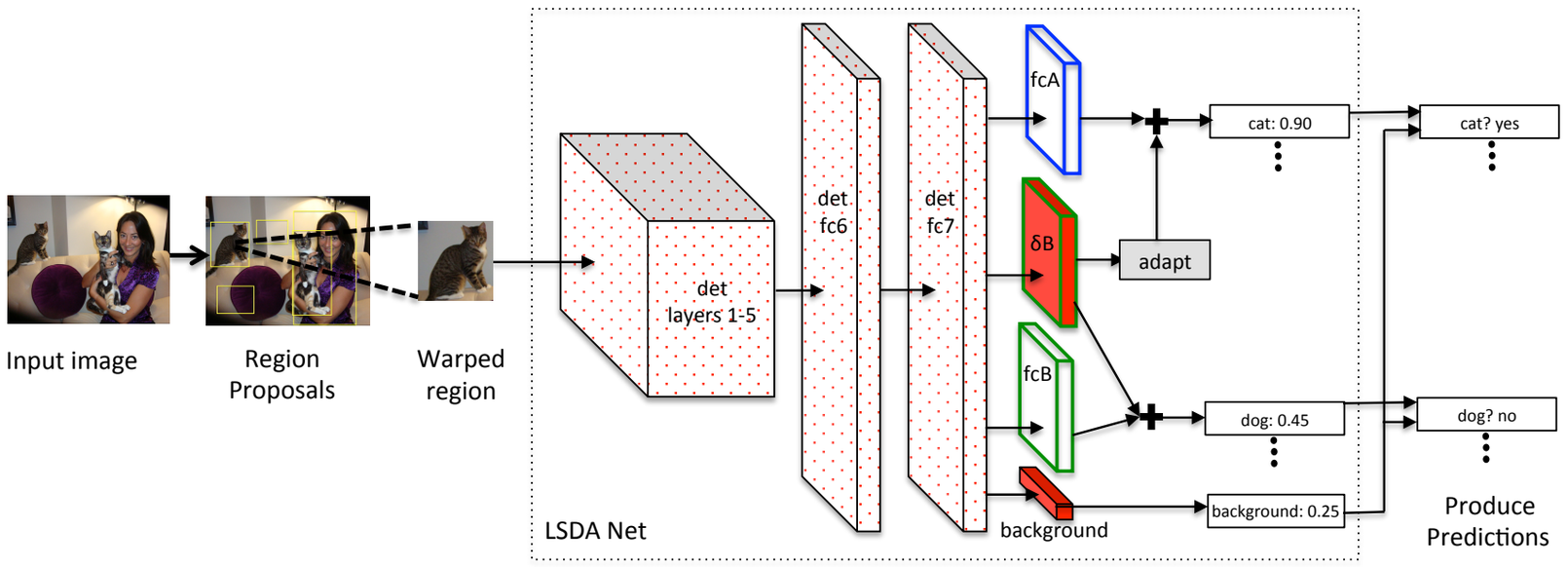
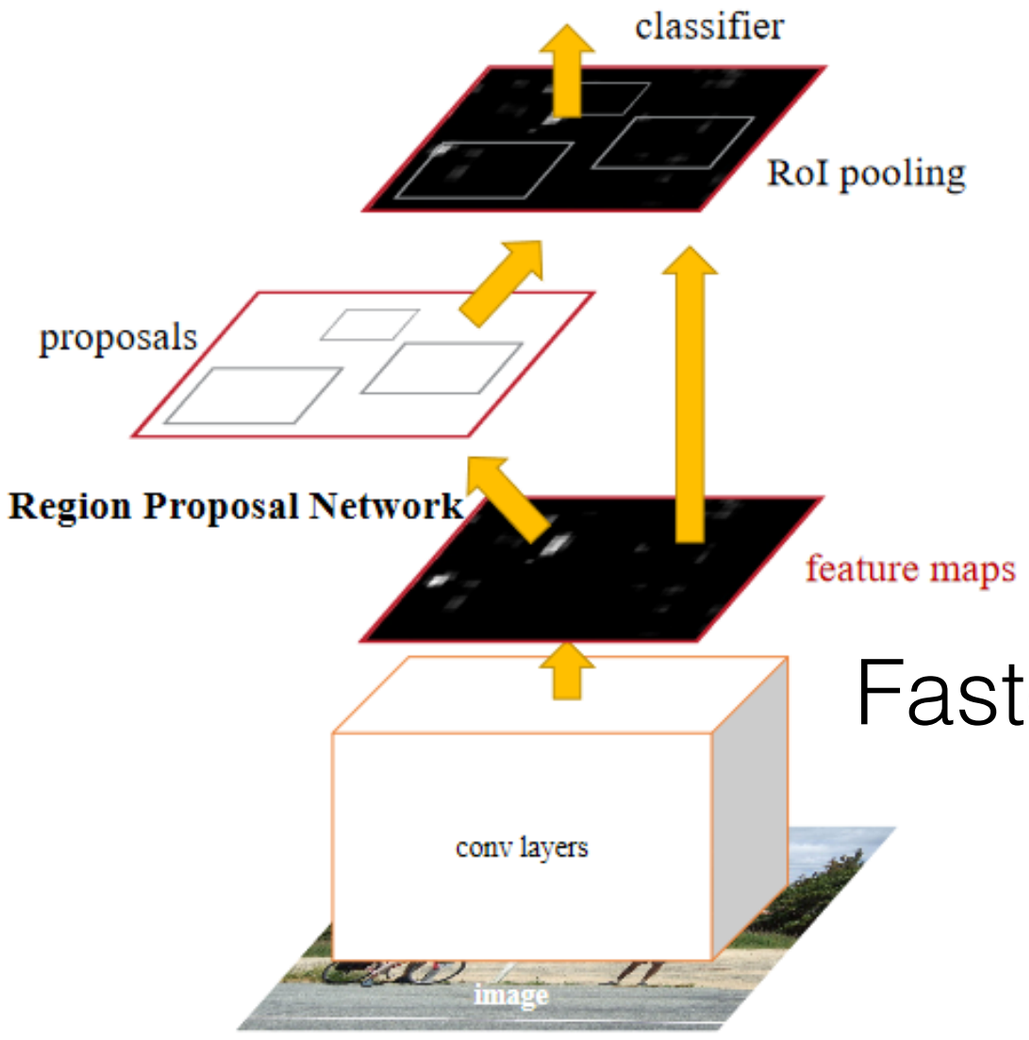


Figure 2: Detection with the LSDA network. Given an image, extract region proposals, reshape the regions to fit into the network size and finally produce detection scores per category for the region. Layers with red dots/fill indicate they have been modified/learned during fine-tuning with available bounding box annotated data.

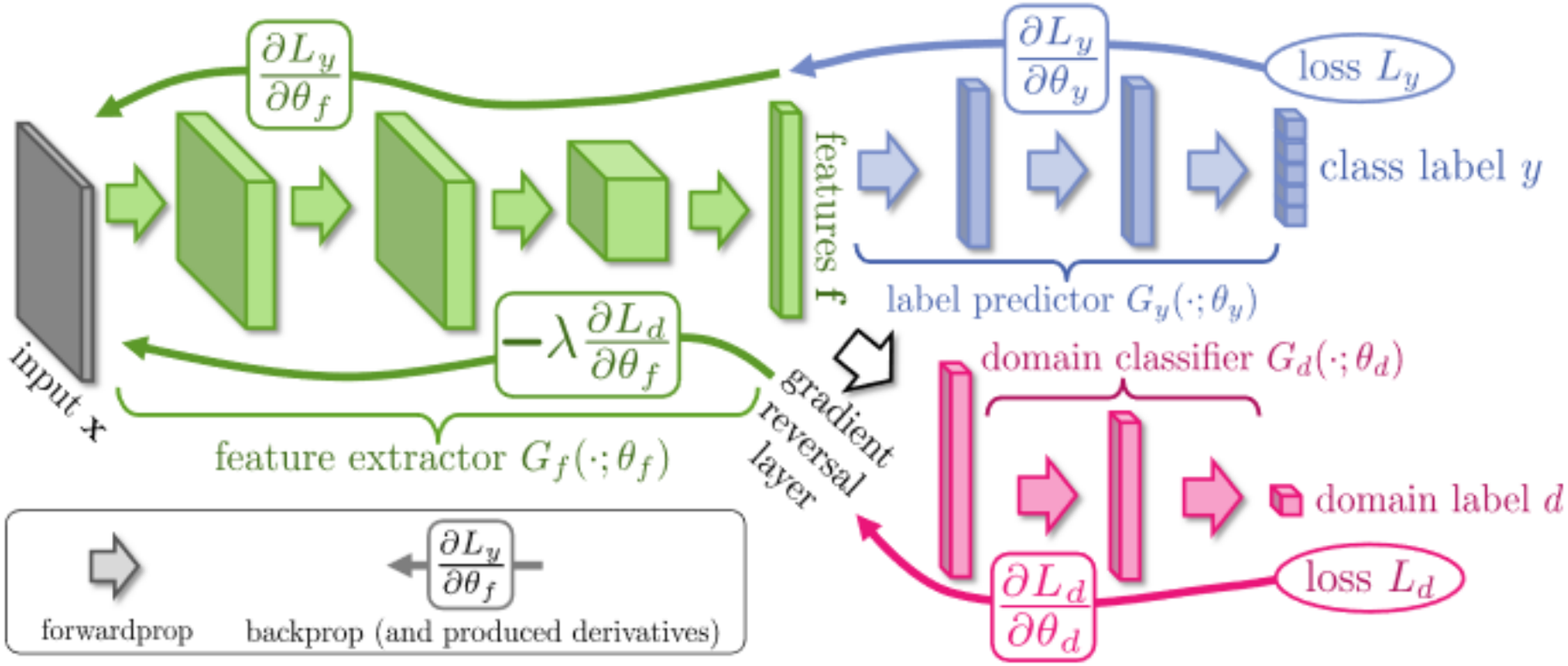
Deep Domain Adaptation for Detection

Deep Domain Adaptation for Detection



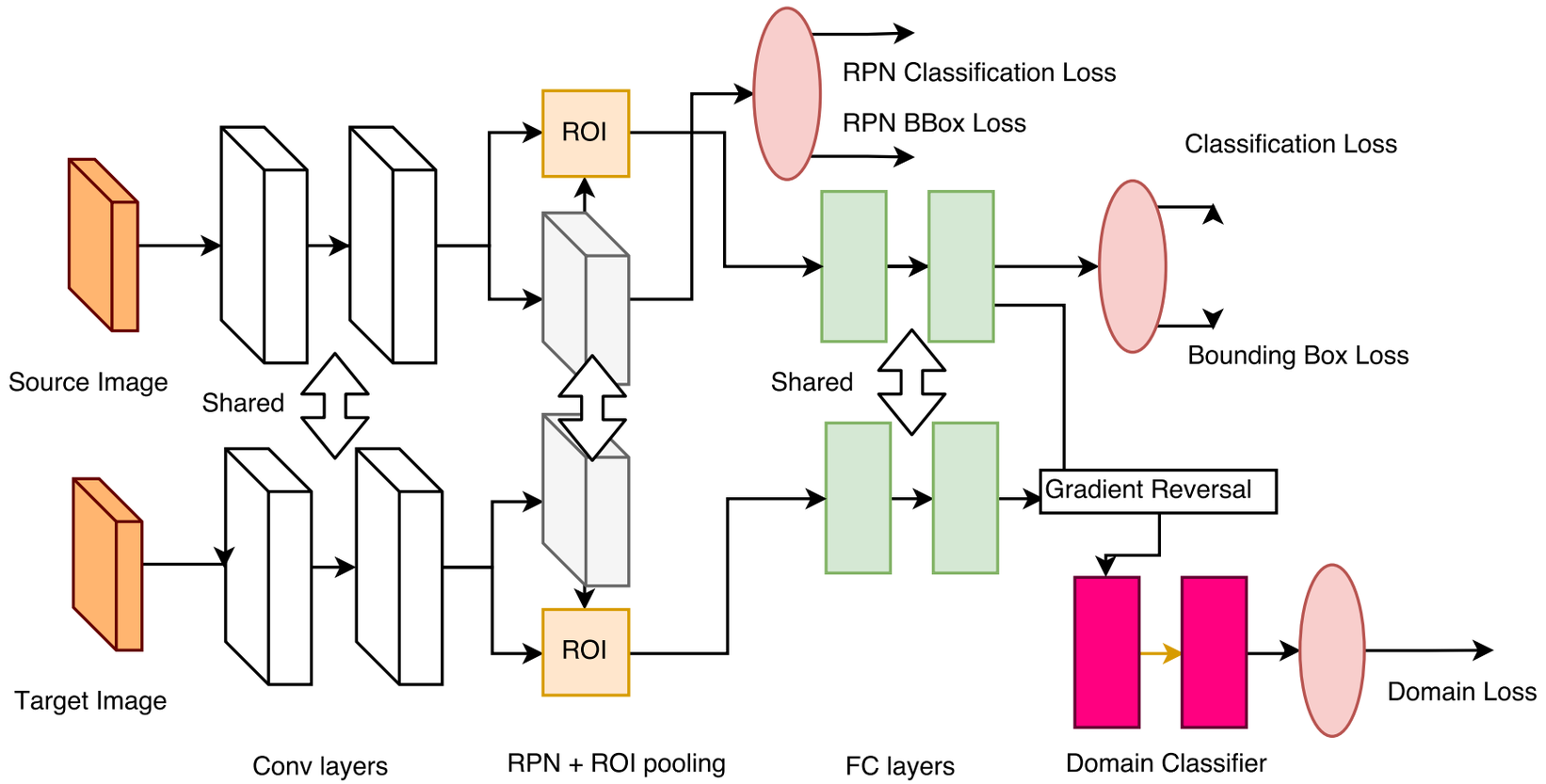
Faster RCNN Network

Deep Domain Adaptation for Detection



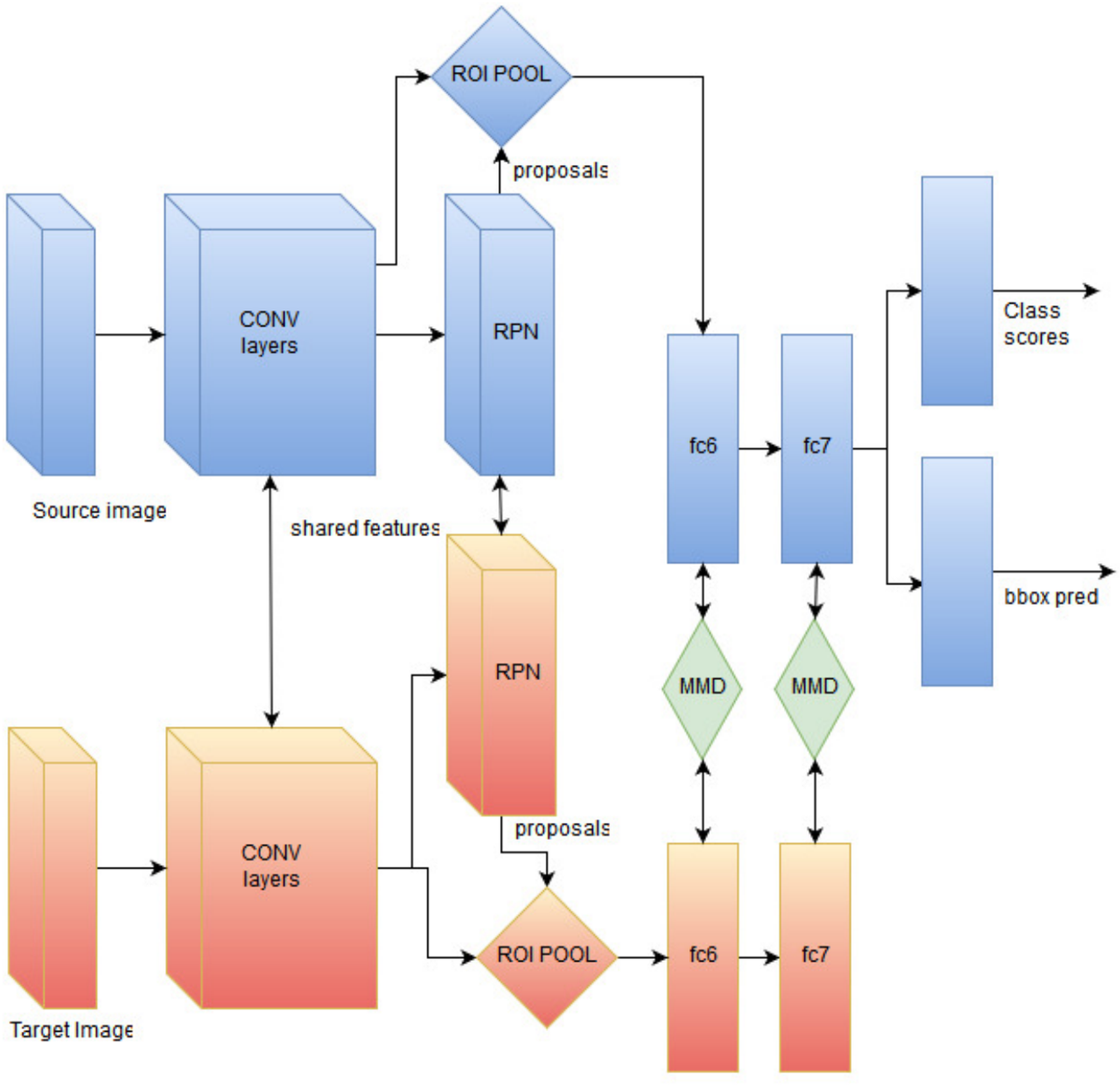
GRL for Classification

Deep Domain Adaptation for Detection



GRL for Detection

Deep Domain Adaptation for Detection



MMD for Detection

Deep Domain Adaptation for Detection - Results

Methods	**Without Data Augmentation**		**With Data Augmentation**	
	IoU:.5	**IoU:[.5:.95]**	**IoU:.5**	**IoU:[.5:.95]**
Source Only	34.95	16.10	35.37	16.44
GRL	35.38	16.52	36.30	17.10
MMD_{fc6}	35.51	17.01	36.36	17.45
MMD_{fc7}	35.44	16.99	36.17	17.36
$MKMMD_{fc6}$	35.28	16.92	36.20	17.38
$MKMMD_{fc7}$	35.24	16.74	36.08	17.15
MMD_{RPN}	36.46	17.15	37.35	17.55
Target Only	46.01	22.40	46.91	22.90

Table: Summary of mAP of all methods

Best result obtained through MMD applied to RPN

Domain Adaptation *in the wild*

Improving assumptions

- Most domain adaptation methods assume that the categories in source and target domain are the same

- Assume that the categories in source that are applicable for target are known

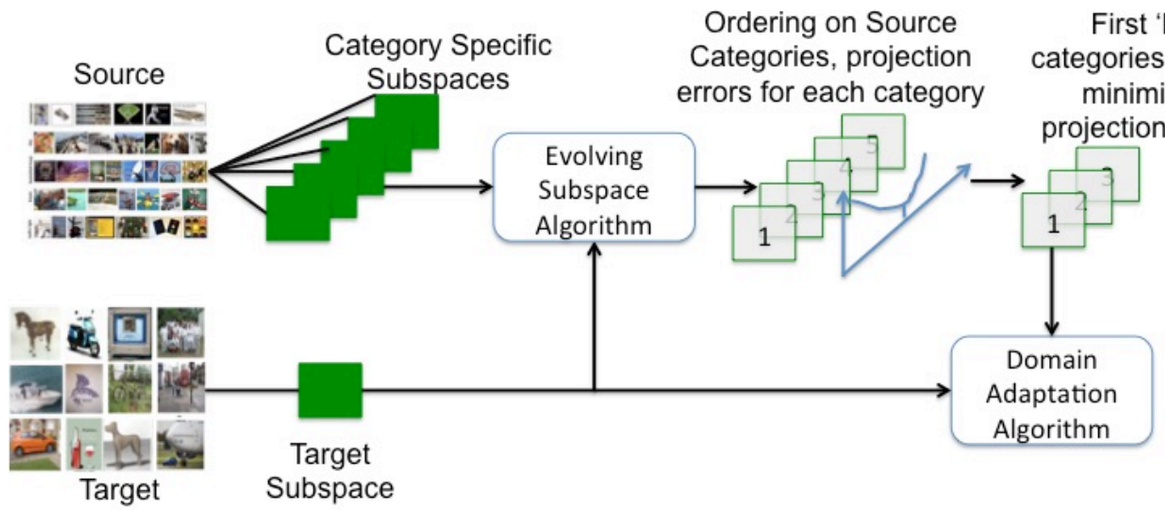
- This can be violated in practice: Suppose we use Imagenet pre-trained classifiers for a robot in a house

Improving assumptions

- Domain adaptation algorithms work best when target label set is same as the source label set

- In practice $Ct < Cs$

An approach



- We divide our approach in three steps:
  - Evolving Subspace Algorithm
  - Category Selection
  - Domain Adaptation

Projection Errors

Let U_S represent target aligned source subspace: $U_S = X_S(X_S'X_T)$

Subspace Alignment Error: $||U_S - X_T||_F$

- Minimizing subspace alignment error aligns principal components.
- If two domains contain same category then the corresponding category specific subspaces will share principal components.

Reprojection Error: $||\mathcal{T} - (\mathcal{T}U_S)U_S'||_F$

- Intuitively, reprojection error computes the discrepancy between the original target dataset and the reconstructed target data after projecting it onto target aligned source subspace.

Step1: Evolving Subspace Algorithm

Algorithm 1 Evolving source subspace to learn target categories

selectedCategories $\leftarrow$ [ ] // Empty List
unselectedSet $\leftarrow C_S$ // Set of categories in source
errors $\leftarrow$ [ ] // Stores projection errors
while *unselectedSet* is not empty **do**
  $c_{min} \leftarrow \underset{c \in unselectedSet}{\text{argmin}}\ ProjectionError(selectedCategories \cup \{c\})$
  Remove c_{min} from *unselectedSet*
  Append c_{min} to *selectedCategories*
  Append projection error for c_{min} on to *errors*
end while
Output: *selectedCategories*, *errors*

Step1: Evolving Subspace Algorithm

- If a category C_i is present in both source and target dataset then adding C_i to *selectedCategories* will bring U_S and X_T closer to each other.
- A natural order on categories: the order in which categories are selected based on minimizing projection error.
- Ideally, we expect categories present in target domain to occur first in ordering.
- We expect projection error to first decrease after each iteration until all categories which are common to source and target are added then it must increase.

Step2: Category Selection

- We expect projection error to first decrease, reach a minima and then increase.
- We select first K categories from *selectedCategories* where minima occurs.
- A lot of local minimas instead of single global minima in practice.

Algorithm 2 Category Selection

Input: // Output of Algorithm 1

$selectedCategories = [C_{\alpha_1}, C_{\alpha_2} \ldots, C_{\alpha_m}]$

$errors = [E_1, E_2, \ldots, E_m]$

$K \leftarrow$ Index in *errors* with Global minima/first local minima.

Output: $C_{train} \leftarrow [C_{\alpha_1}, C_{\alpha_2} \ldots, C_{\alpha_K}]$

Step2: Category Selection

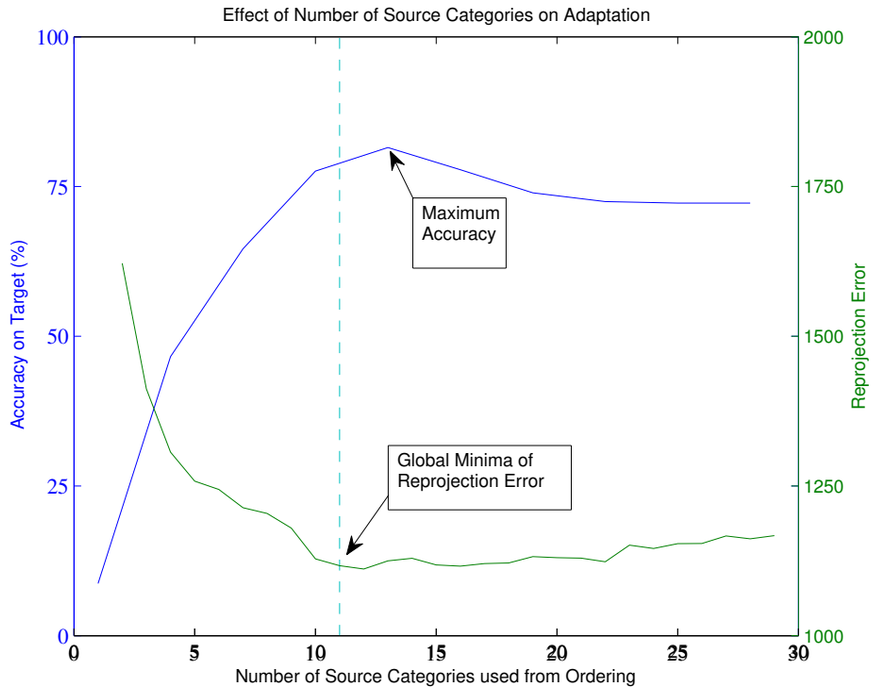
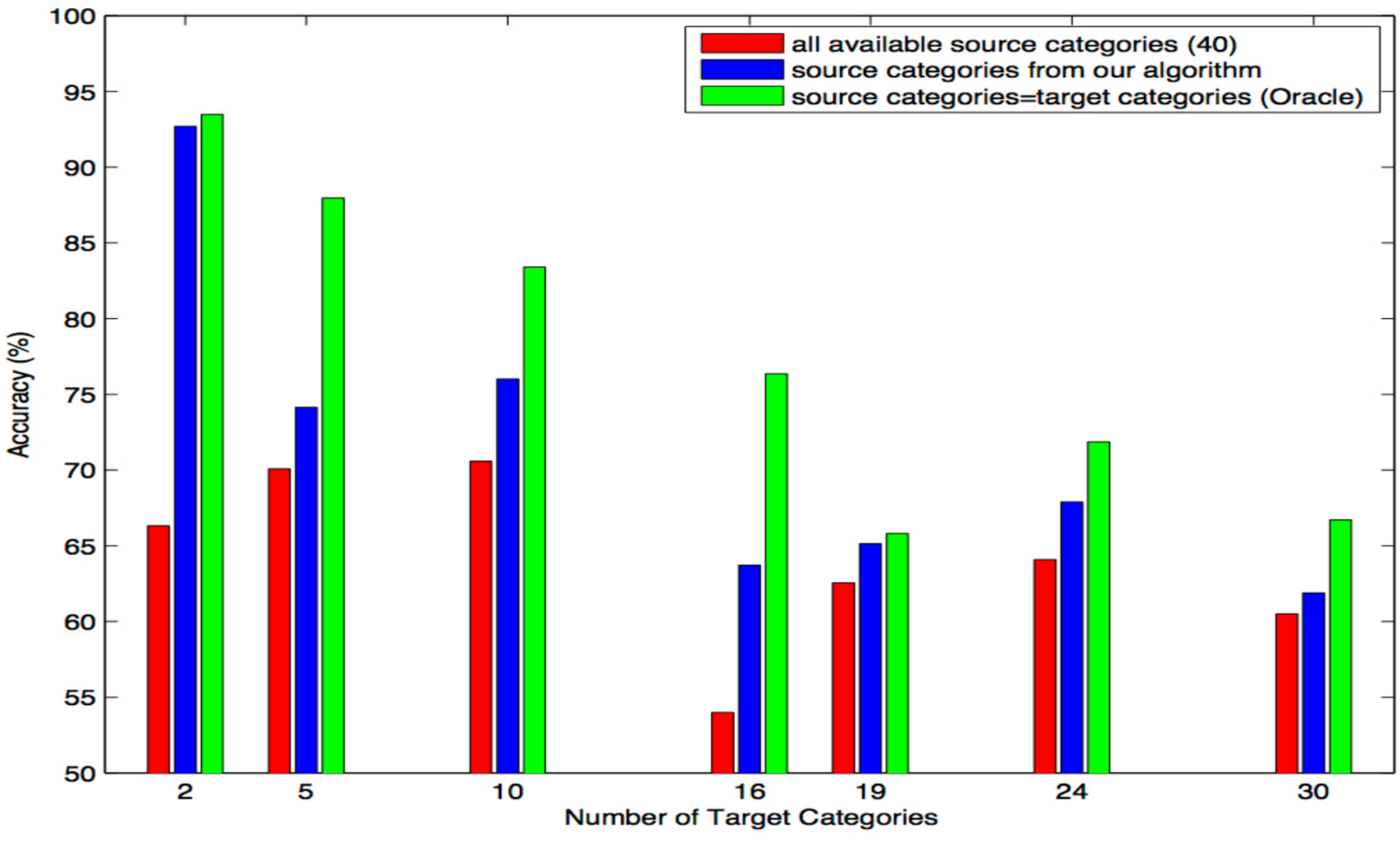


Figure: Performance of our adaptation algorithm (left y-axis) and reprojection error (right y-axis) as we change the number of source categories (x-axis) used for training the source classifier.

Step3: Domain Adaptation

- Adapting source examples with labels in C_{train} to target.
- Can use any state-of-art domain adaptation algorithm for this step.

Results


all available source categories (40)
source categories from our algorithm
source categories=target categories (Oracle)
Accuracy (%)
100
95
90
85
80
75
70
65
60
55
50
2
5
10
16
19
24
30
Number of Target Categories

Extending to Multi-source Domain Adaptation

- More than one source domains are available.
- Adapt from multiple source domains to a single target domain.
- A simple approach is to combine all source domains into a single domain and use single source domain adaptation techniques.
- Another commonly used approach: Learn a single source domain adaptation classifier for each source domain and combine results of these classifiers.
- Our approach automatically picks an optimal combination of categories from different source domains.

Problem description

- Given p labeled source domains: $\mathcal{D}_{S1}, \mathcal{D}_{S2}, \ldots, \mathcal{D}_{Sp}$
- A single unlabeled target domain: $\mathcal{D}_T$
- Generalize p source domains to target domain.
- In literature, multi-source domain adaptation techniques assume label sets to be same for all the domains.
- Our approach doesn't requires this assumption.

Multi-source domain adaptation

Algorithm 3 Multi-source domain adaptation

$L_* \leftarrow \{C_{11}, C_{12}, \ldots, C_{pm}\}$
$//C_{ij}$ represents category C_j of source domain $\mathcal{S}_i$
$\mathcal{S} \leftarrow \mathcal{S}_1 \cup \mathcal{S}_2 \cdots \cup \mathcal{S}_p$ // Combined Source dataset
$ordering \leftarrow EvolvingSubspace(\mathcal{S}, \mathcal{T}, L_*)$
$C_{train} \leftarrow CategorySelection(ordering)$
Output: C_{train}
//If $C_{ij} \in C_{train}$ then we use category C_j of source domain $\mathcal{S}_i$ for training.

Results

Table: Multi-Source domain adaptation results.

Target Domain	Source DSLR	Source Amazon	Source Webcam	Source Caltech	Source Rest three domains	Source Selected by our Algorithm	Number of categories
DSLR	-	86.00	**97.33**	78.00	70.67	**97.33**	10
Amazon	61.27	-	64.62	59.82	62.05	**74.44**	16
Webcam	**82.91**	72.95	-	58.01	66.55	81.49	10
Caltech	62.61	91.62	70.87	-	**91.74**	90.85	17

Conclusion

- We have considered a few techniques for adapting detectors
- The general problem of adaptation for use in the real world is challenging and requires further investigation
- While, domain adaptation is well explored for classification, other vision tasks also need to be adapted

Few Shot Learning

Matching Networks

One-shot learning with attention and memory

- Learn a concept from one or only a few training examples
- Train a fully end-to-end nearest neighbor classifier: incorporating the best characteristics from both parametric and non-parametric models
- Improved one-shot accuracy on Omniglot from 88.0% to 93.2% compared to competing approaches

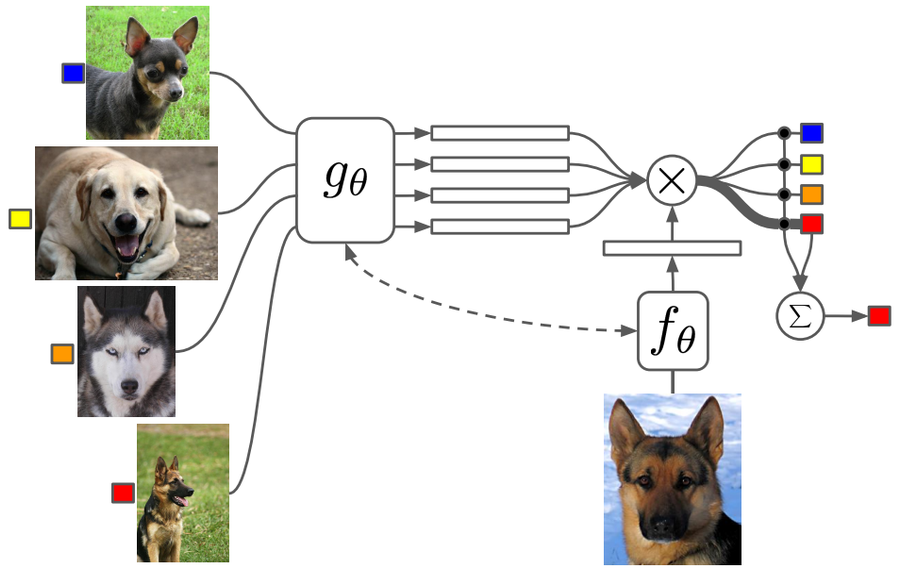
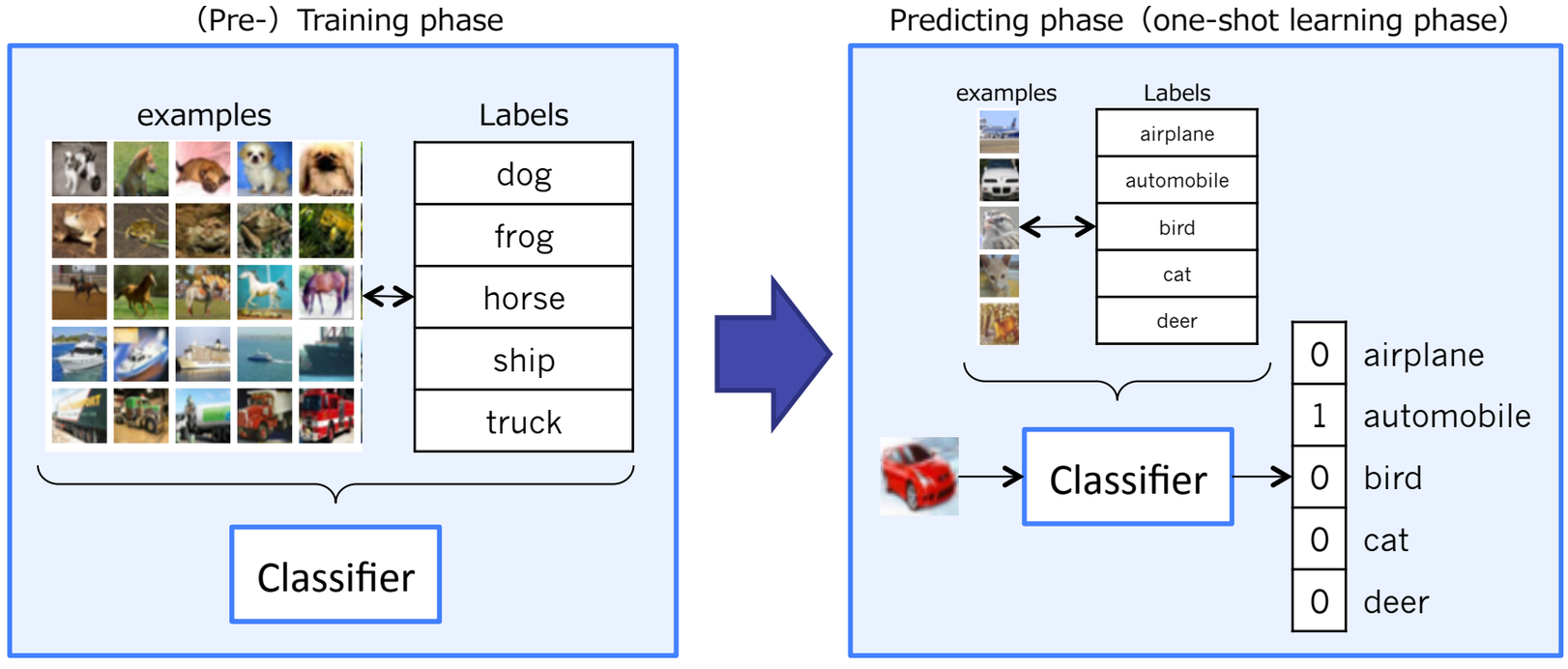


Figure 1: Matching Networks architecture

Matching Networks

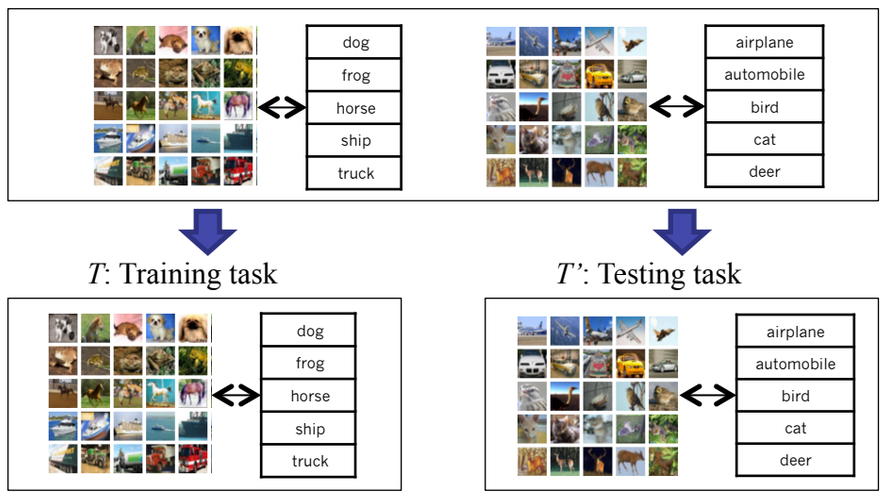
- Learn a concept from one or only a few training examples
  - A classifier can be trained by datasets with labels which don't be used in predicting phase



Matching Networks

One-shot Learning

- Task: N-way k-shot learning

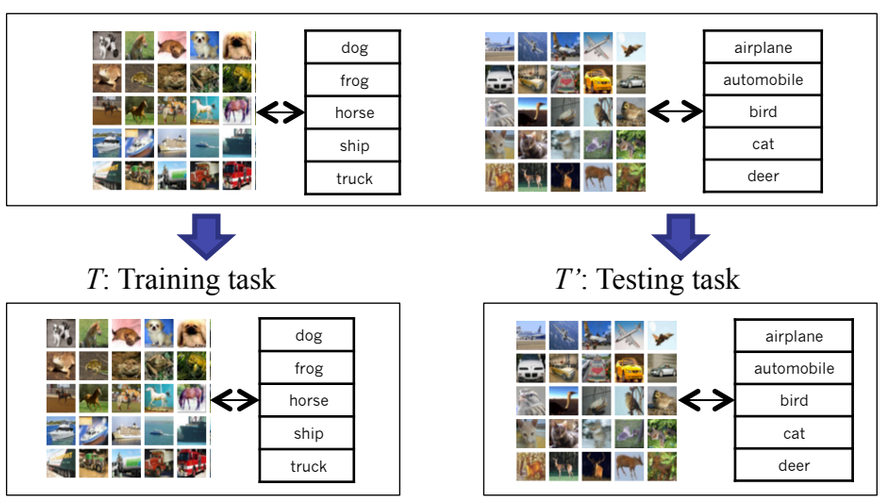


- T can be used freely to train (e.g. Multiclass classification)

- T' is used for one-shot learning

Matching Networks

- Task: N-way k-shot learning



- Task: classify $\hat{x}$ into 3 classes, {automobile, cat, deer}, using support set

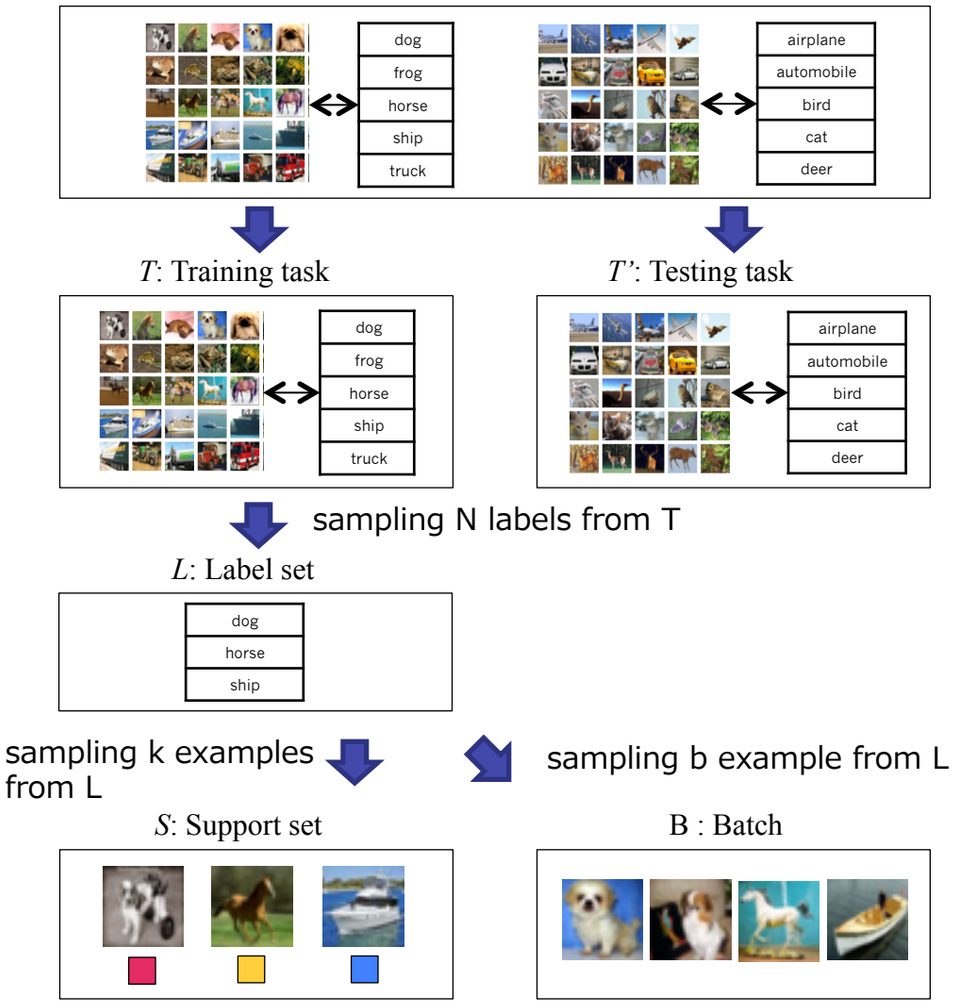
https://www.cs.toronto.edu/~kriz/cifar.html

Matching Networks

- Motivation
  - It is important for one-shot learning to attain rapid learning from new examples while keeping an ability for common examples
    - Simple parametric models such as deep classifiers need to be optimized to treat with new examples
    - Non-parametric models such as k-nearest neighbor don’t require optimization but performance depends on the chosen metric
  - It could be efficient to train a end-to-end nearest neighbor based classifier

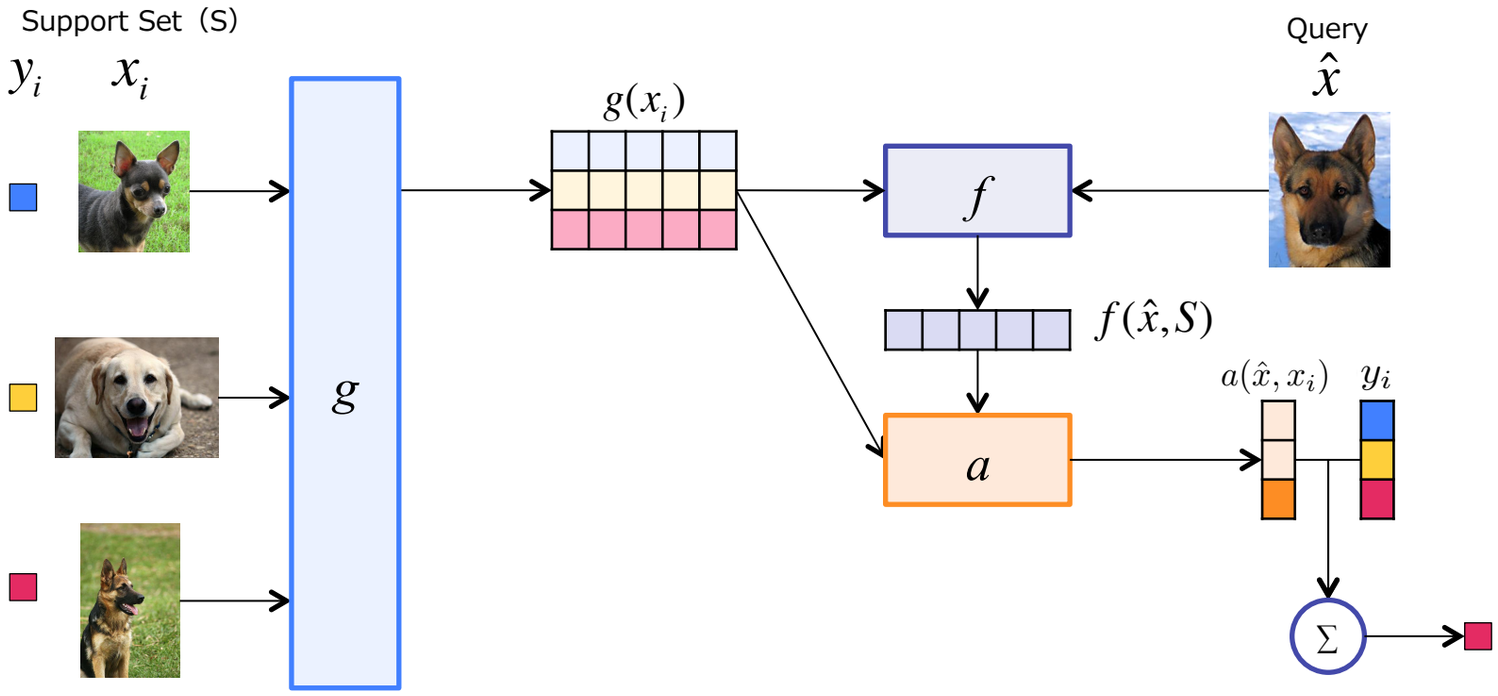
Matching Networks

- Train a classifier through one-shot learning



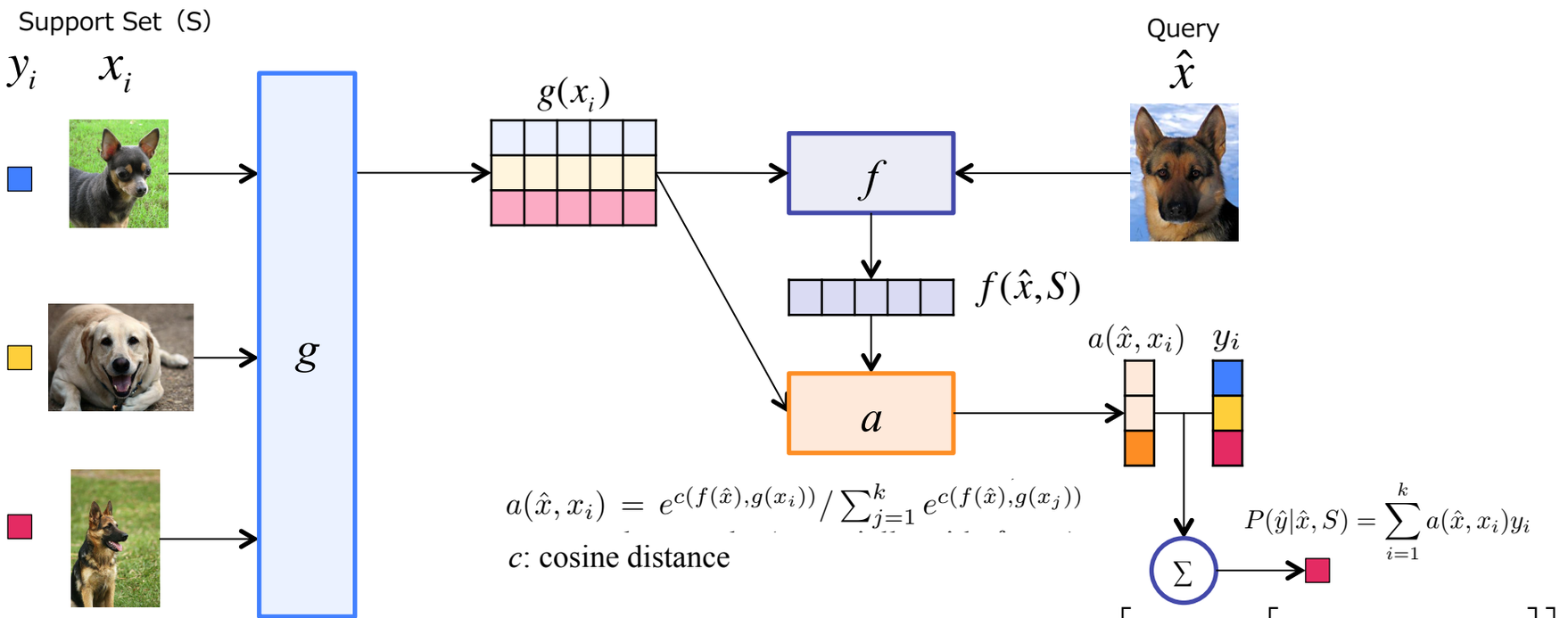
Matching Networks

- System Overview
  - Embedding functions f, g are parameterized as a simple CNN (e.g. VGG or Inception) or a fully conditional embedding function mentioned later



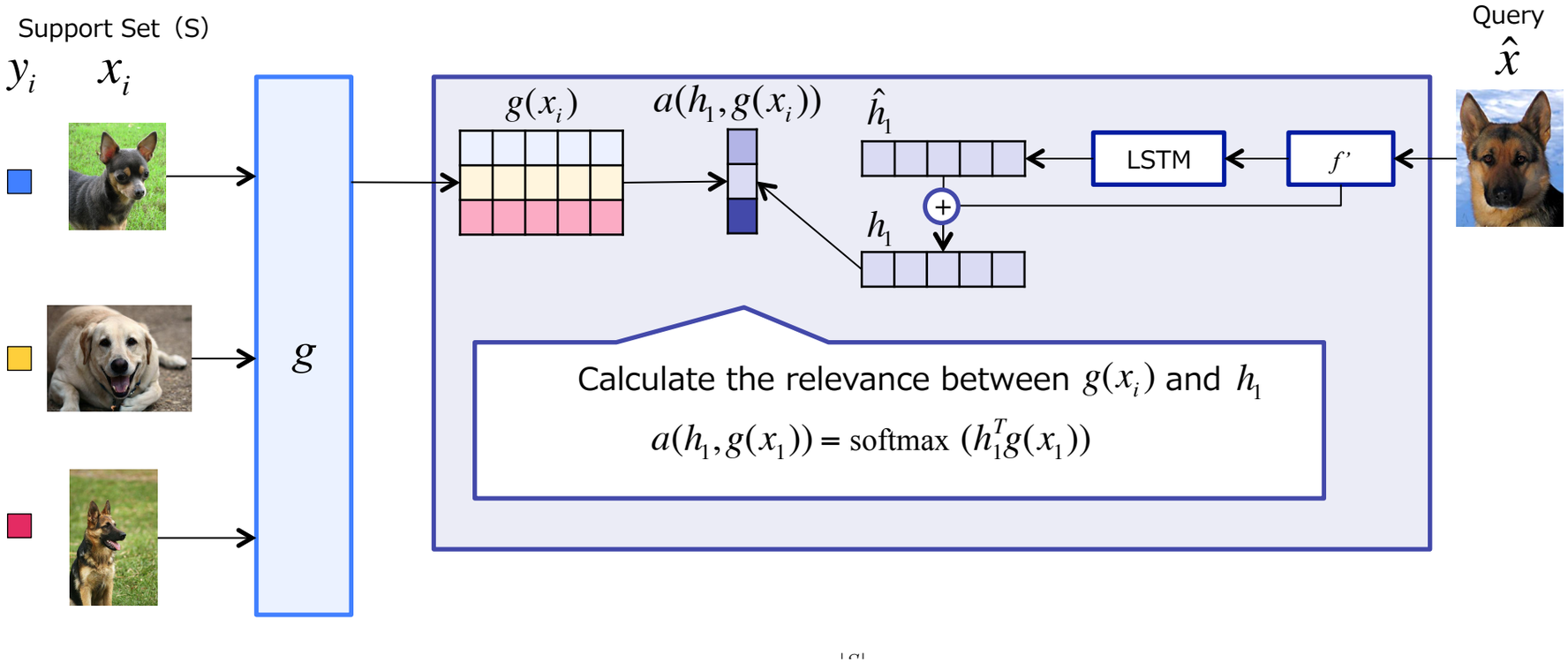
Matching Networks

- 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



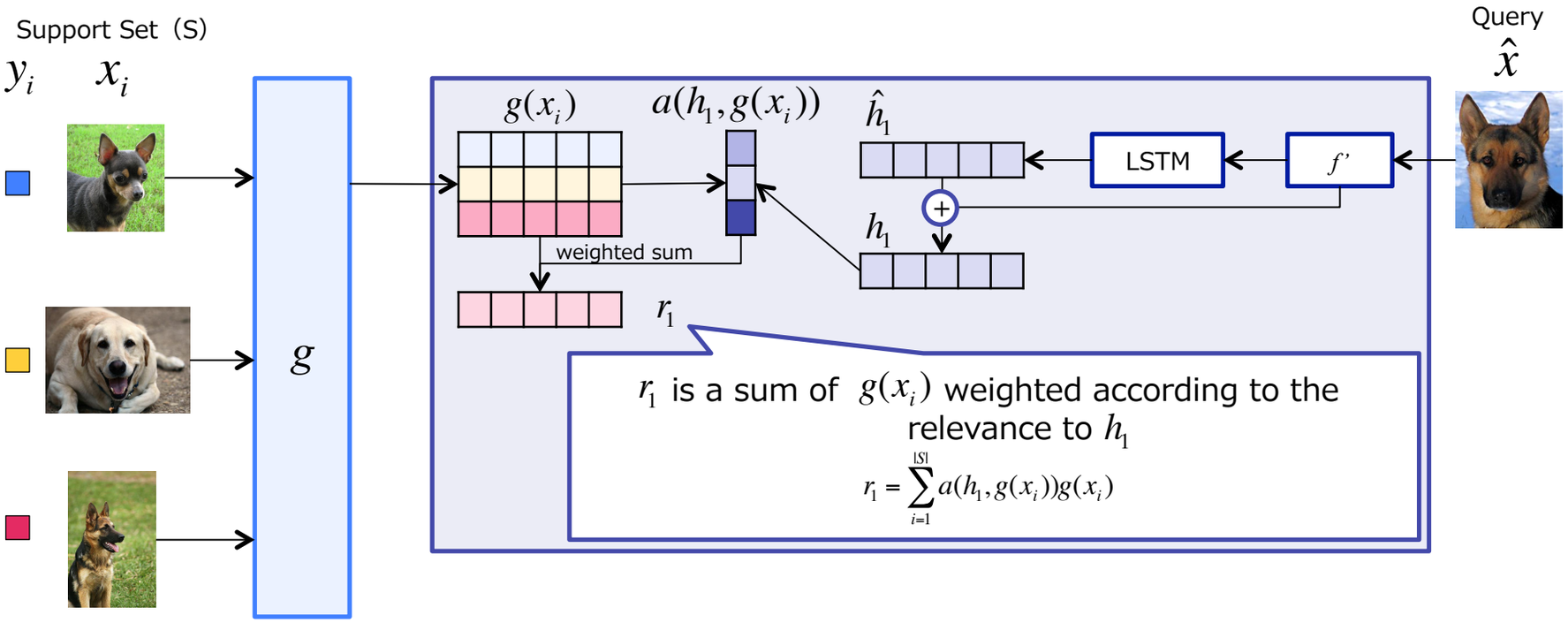
Matching Networks

- The Fully Conditional Embedding f
  - Embed $\hat{x}$ in consideration of S



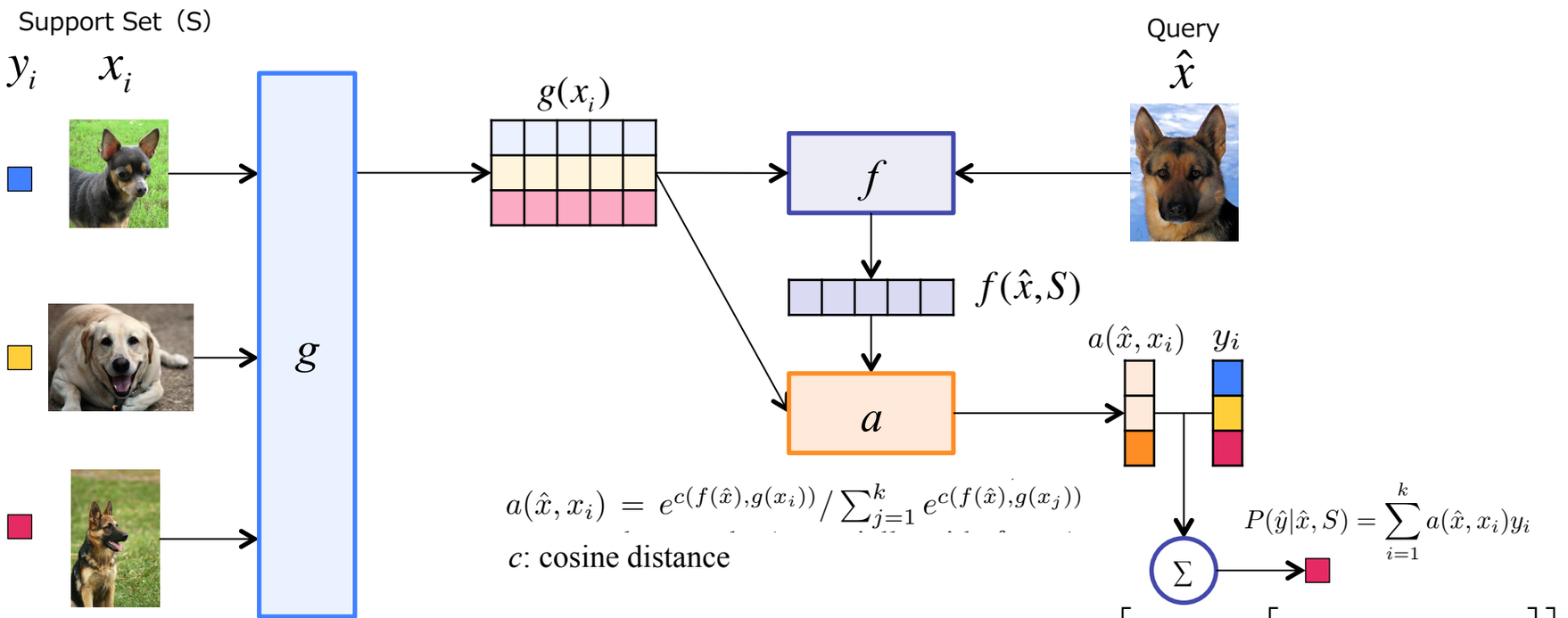
Matching Networks

- The Fully Conditional Embedding f
  - Embed $\hat{x}$ in consideration of S



Matching Networks

- 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



Matching Networks

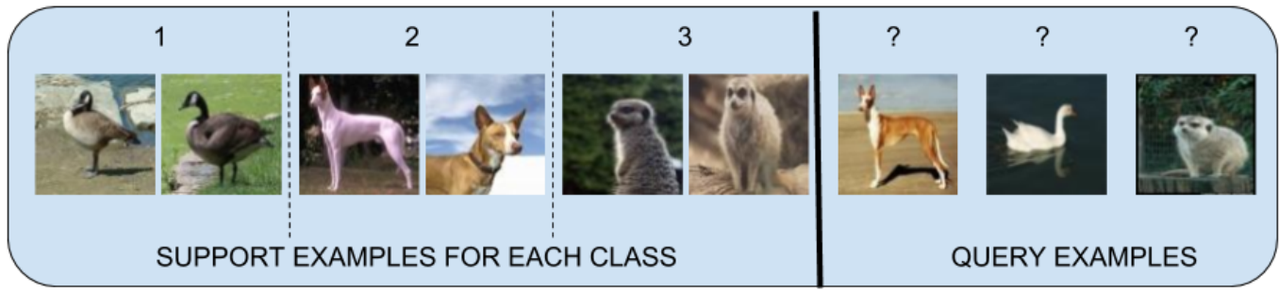
Experimental Results (randImageNet, dogsImageNet)

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)	$\approx 99\%$	$\approx 99\%$	$\approx 99\%$	$\approx 99\%$

Prototypical Networks

- Each episode has a set of support examples per class and a set of query examples for which the output class is to be predicted.
- A deep convolutional neural network (CNN) encodes the examples.
- For each episode, the embeddings of the support examples of each class are averaged to give a "prototype" for each class.
- The prototype that is closest to the query example determines the class of the query example.
- Learns a metric space in which classification can be performed by computing distances to prototype representations of each class.
- Cannot be directly used on a multi attribute dataset.



Prototypical Networks

At prediction time we are given a support set of N labeled examples: $S = \{(x_i, y_i)\}_{i=1}^{N} = S^1 \cup \ldots \cup S^K$ where $S^k = \{(x, y) \in S \,|\, y = k\}$. Our method computes a class representation c_k, or *prototype*, of each class through an embedding function $f_\theta(x)$ parameterized by learnable parameters θ:

$$c_k = \frac{1}{|S^k|} \sum_{(x,y) \in S^k} f_\theta(x) \tag{1}$$

Given a test point $\tilde{x}$, prototypical networks forms a distribution over classes based on a softmax over the Euclidean distances between its embedding and the prototypes:

$$p(y = k \,|\, \tilde{x}) = \frac{\exp(-\|f_\theta(\tilde{x}) - c_k\|^2)}{\sum_{k'} \exp(-\|f_\theta(\tilde{x}) - c'_k\|^2)} \tag{2}$$

Learning proceeds by maximizing the log-probability of the true class $\tilde{y}$:

$$\max_\theta \log p(\tilde{y} \,|\, \tilde{x})$$

Prototypical Networks

	5-way		20-way	
Model	1-shot	5-shot	1-shot	5-shot
Pixels	41.7%	63.2%	26.7%	42.6%
Baseline Classifier	80.0%	95.0%	69.5%	89.1%
Neural Statistician (Edwards & Storkey, 2016)*	-	-	88%	95%
Matching Nets (non-FCE, no fine-tune)	**98.1**%	98.9%	93.8%	98.5%
Prototypical Nets (1-shot)	**98.1**%	**99.5**%	**94.2**%	**98.6**%

Table 1: Omniglot few-shot classification accuracy. *Note that the Neural Statistician used non-standard class splits.

Prototypical Networks

Method	Image Features	Top-1 Acc (50-way)
ALE (Akata et al., 2013)	Fisher Vectors	26.9%
SJE (Akata et al., 2015)	AlexNet	40.3%
Sample-Clustering (Liao et al., 2016)	AlexNet	44.3%
SJE (Akata et al., 2015)	GoogLeNet	50.1%
DS-SJE (Reed et al., 2016)	GoogLeNet	50.4%
DA-SJE (Reed et al., 2016)	GoogLeNet	50.9%
Prototypical Networks	GoogLeNet	**54.6%**

Table 3: CUB-200 zero-shot classification accuracy for methods utilizing attribute vectors as class metadata.

Other Semi-Supervised learning techniques

- Zero Shot learning
- Active Learning
- Weakly Supervised Learning

Thank you