# Towards Set Learning and Prediction
## using deep neural networks

**Prof. Laura Leal-Taixé**

Slides by

S. Hamid Rezatofighi

# Introduction

Deep neural networks

✓ Benefits: Remarkable performance on many real-world problems

✗ Limitation: built to deal with *structured* input and output data, *i.e.* vectors, matrices or tensors

- Ordered
- Fixed in size

Many real-world problems, however, are naturally described as ***sets***, rather than vectors

# Set VS Vector

As opposed to a vector, a set

- is not fixed in size
- is invariant to the ordering of entities within it

*For example*

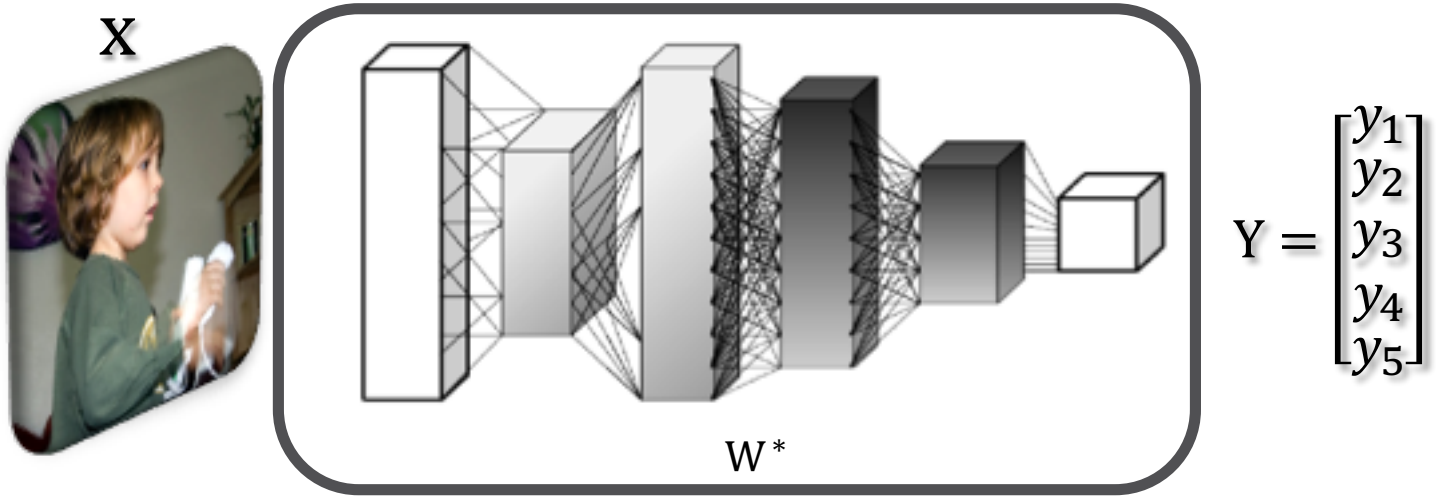
$$\mathcal{Y}_1 = \{1,2,3,4\}$$
$$\mathcal{Y}_2 = \{3,1,2,4\}$$
$$\Rightarrow \quad \mathcal{Y}_1 \triangleq \mathcal{Y}_2$$

*But*

$$Y_1 = (1,2,3,4)$$
$$Y_2 = (3,1,2,4)$$
$$\Rightarrow \quad Y_1 \neq Y_2$$

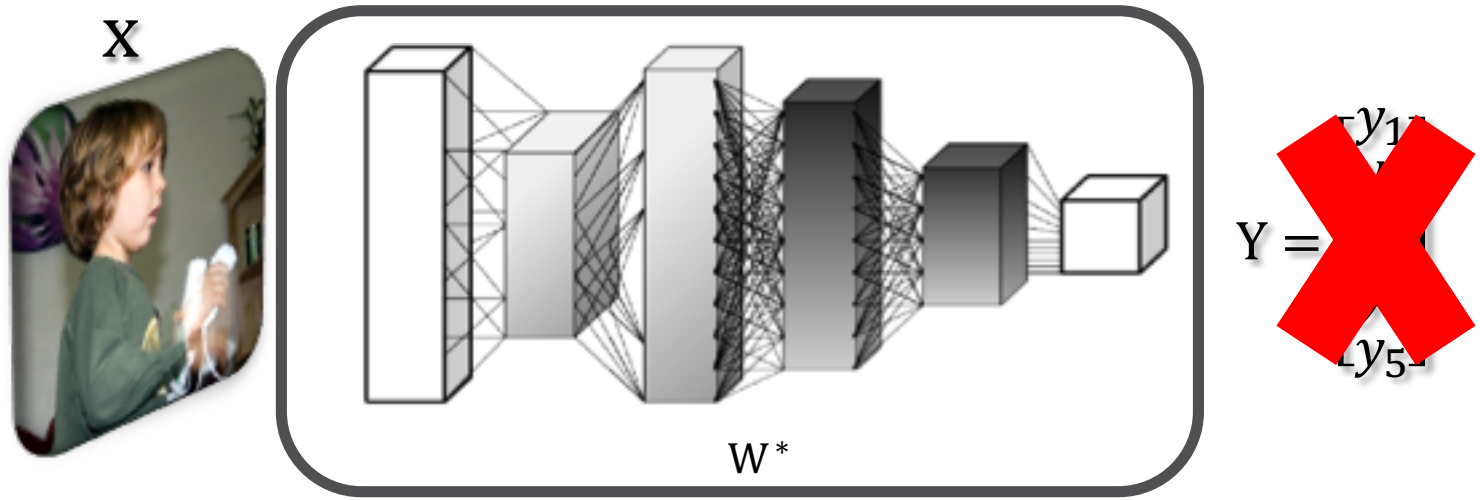
# Conventional Approach

Conventional machine learning approaches learn the optimal parameters $w^*$ of the distribution $p(Y|x, w^*)$, that maps a **structured input**, *e.g.* a tensor, to **a structured output**, *e.g.* a fixed length vector



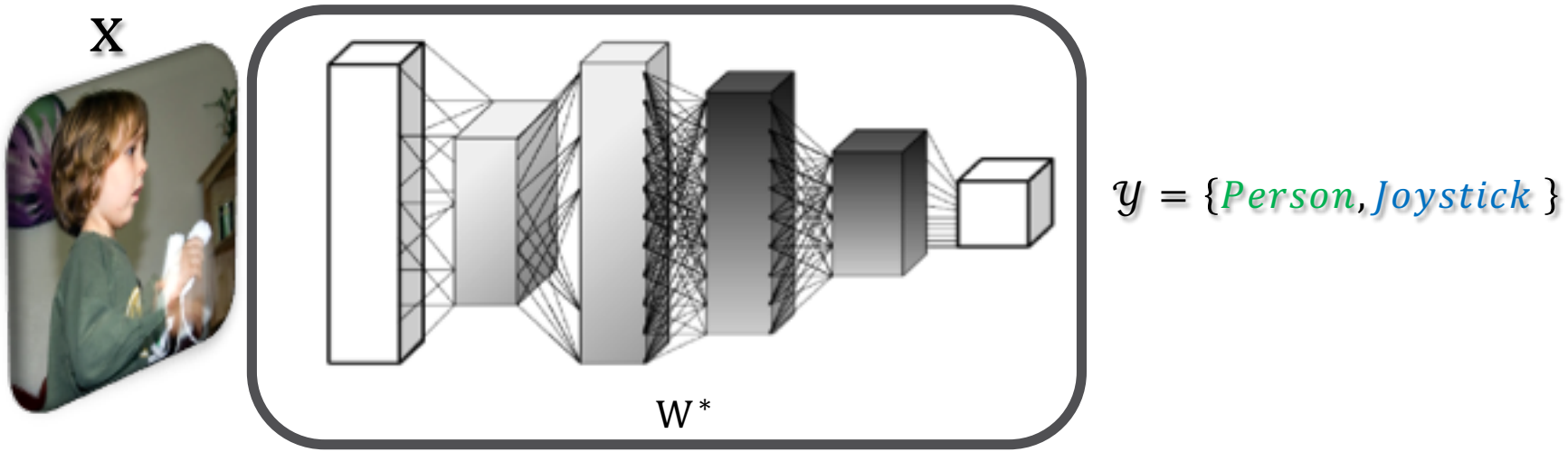
# Conventional Approach

However, the output of some problems, *e.g.* image tagging, are indeed a set of elements (*e.g.* tags) rather than vectors



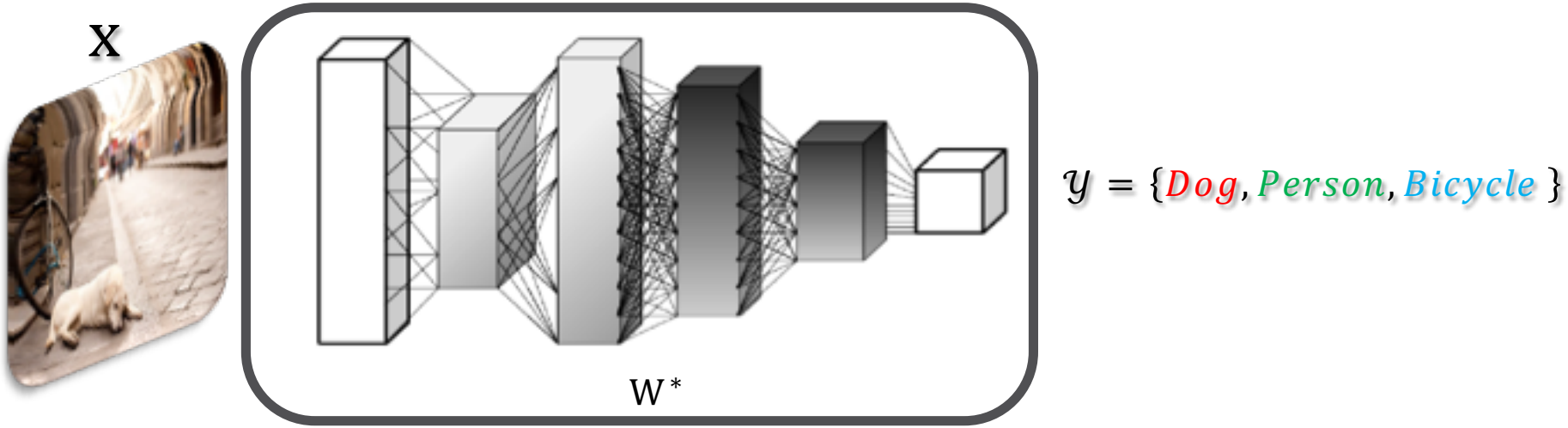
# Conventional Approach

However, the output of some problems, *e.g.* image tagging, are indeed a set of elements (*e.g.* tags) rather than vectors



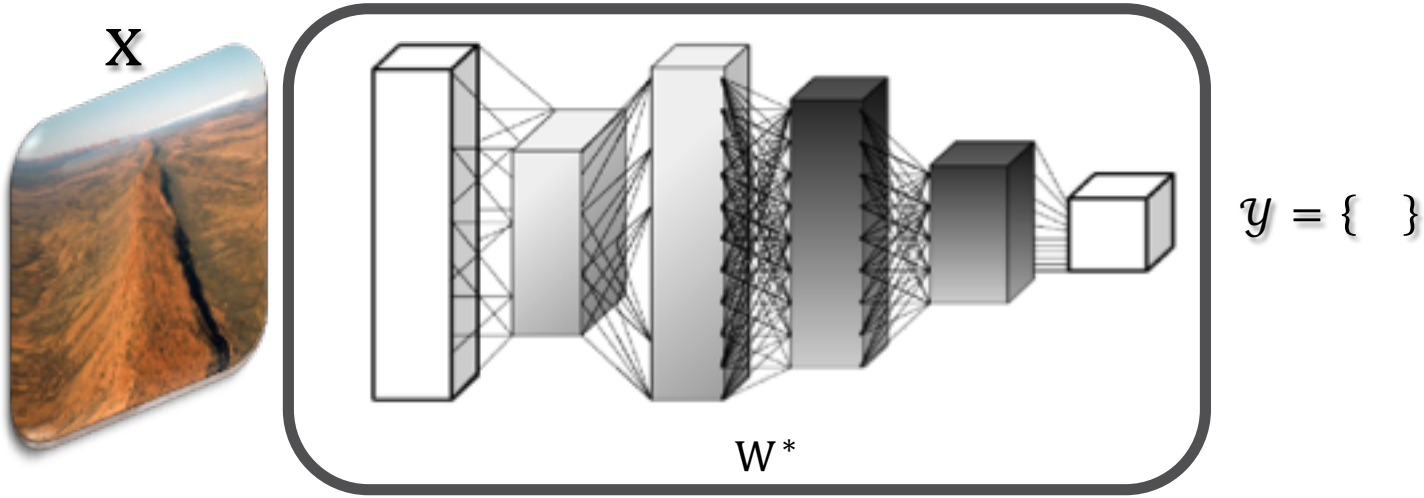
# Conventional Approach

However, the output of some problems, *e.g.* image tagging, are indeed a set of elements (*e.g.* tags) rather than vectors



# Conventional Approach

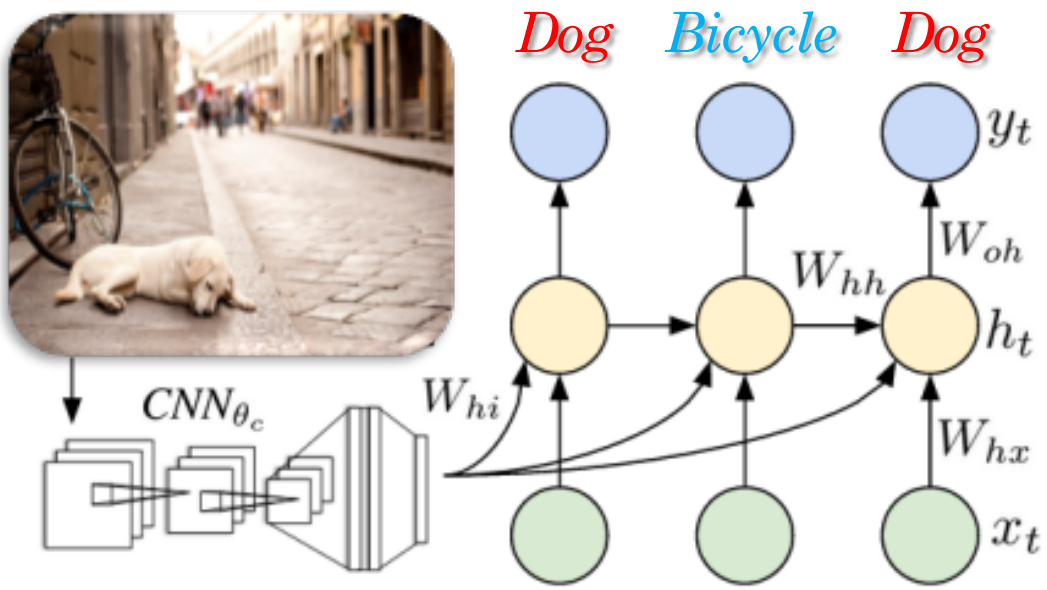
However, the output of some problems, *e.g.* image tagging, are indeed a set of elements (*e.g.* tags) rather than vectors



# Existing (Approx.) Solutions

Existing techniques that allow for sequential data, *e.g.* RNN or LSTM

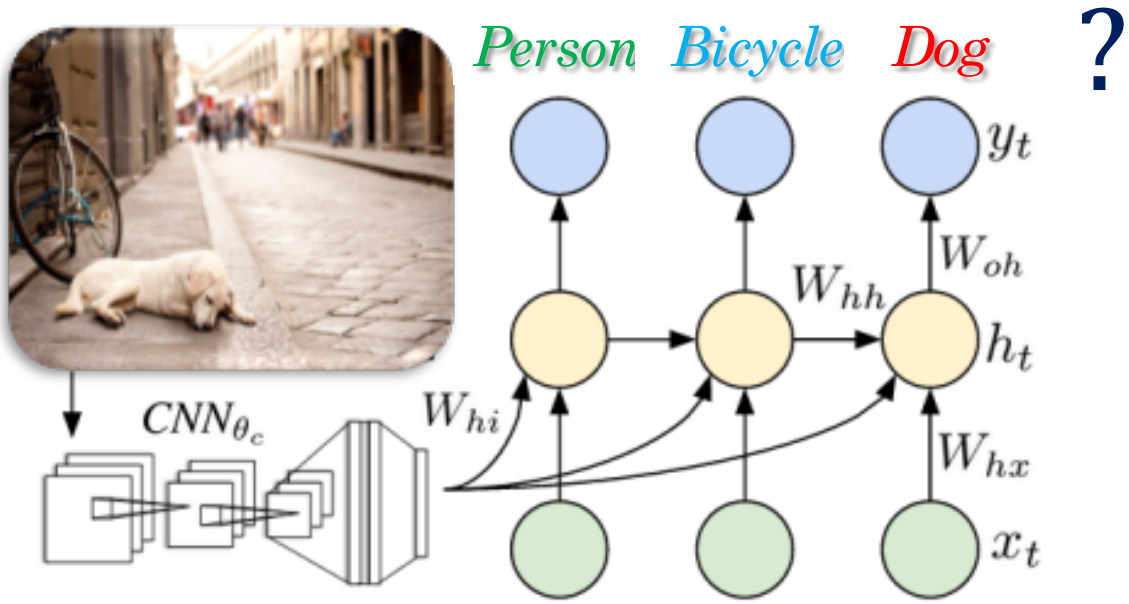
✘ do not guarantee a valid solution



# Existing (Approx.) Solutions

Existing techniques that allow for sequential data, *e.g.* RNN or LSTM

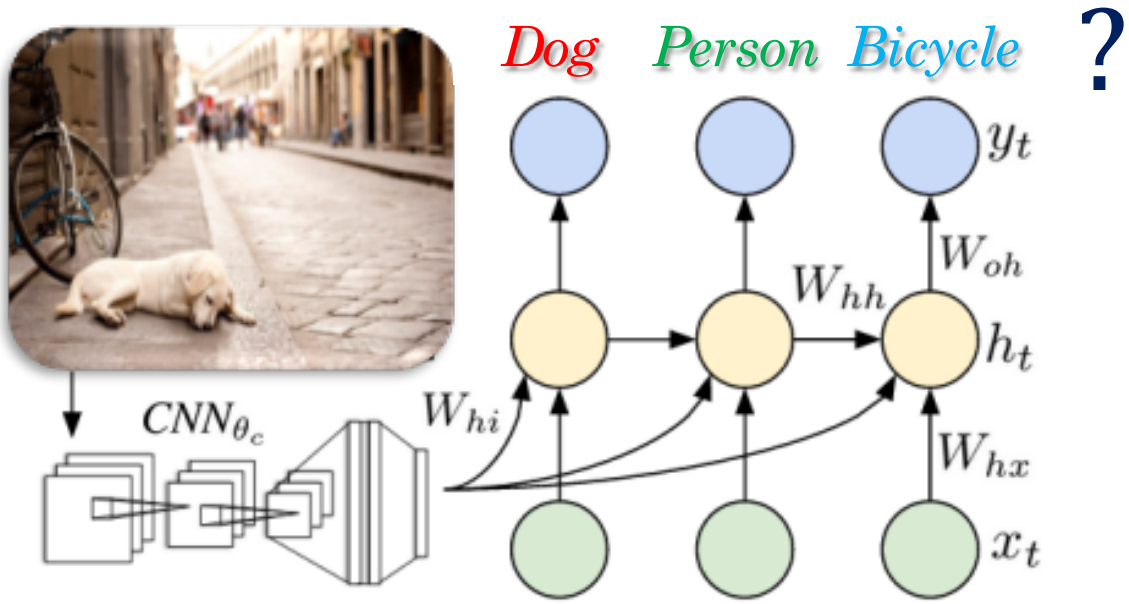
- ✘ do not guarantee a valid solution
- ✘ heavily depend on the input and output order



# Existing (Approx.) Solutions

Existing techniques that allow for sequential data, *e.g.* RNN or LSTM

- ✘ do not guarantee a valid solution
- ✘ heavily depend on the input and output order



# Application Examples

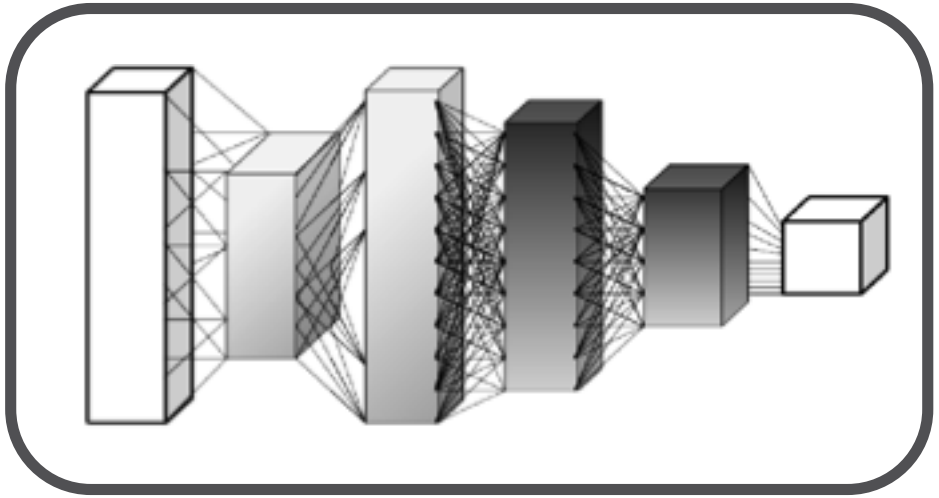
- ✓ Object Detection and Localization

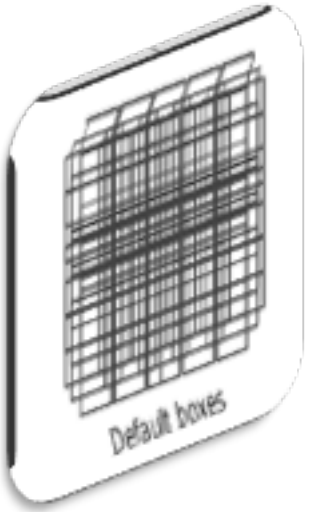
*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO

# Application Examples

- ✓ Object Detection and Localization

*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO

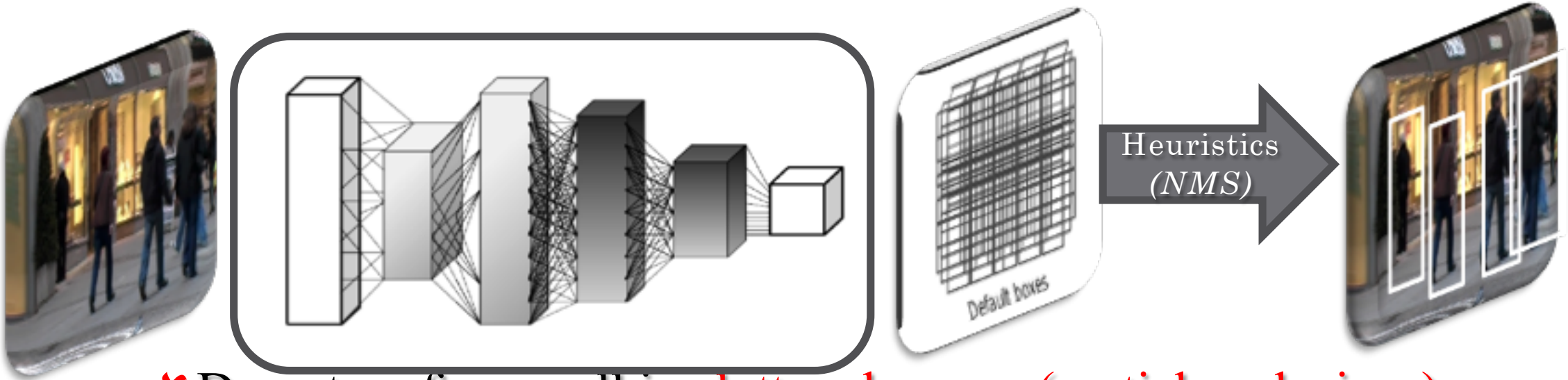




# Application Examples

✓ Object Detection and Localization

*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO




✘ Do not perform well in cluttered scenes (partial occlusions)

# Application Examples

✓ Object Detection and Localization

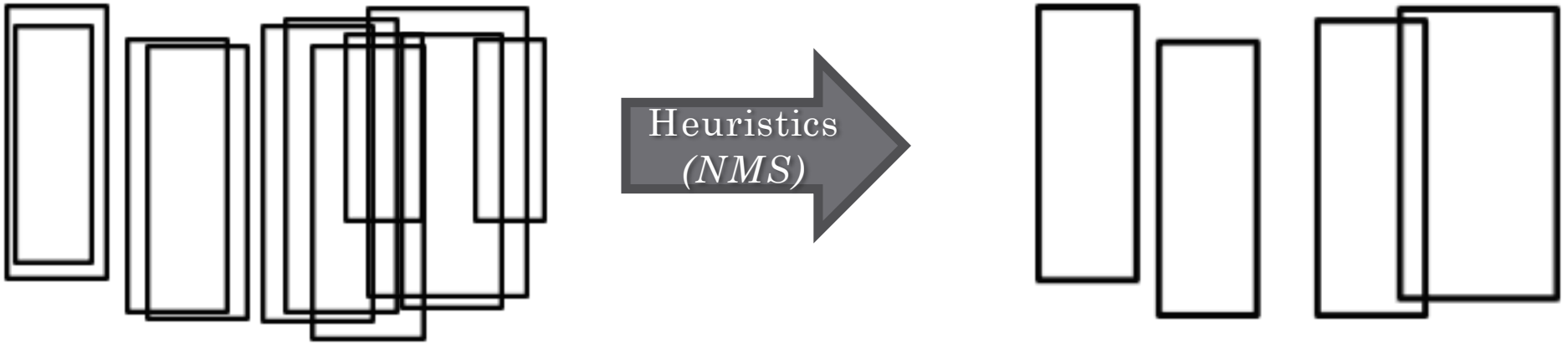
*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO


✘ Do not perform well in cluttered scenes (partial occlusions)

# Application Examples

✓ Object Detection and Localization

*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO

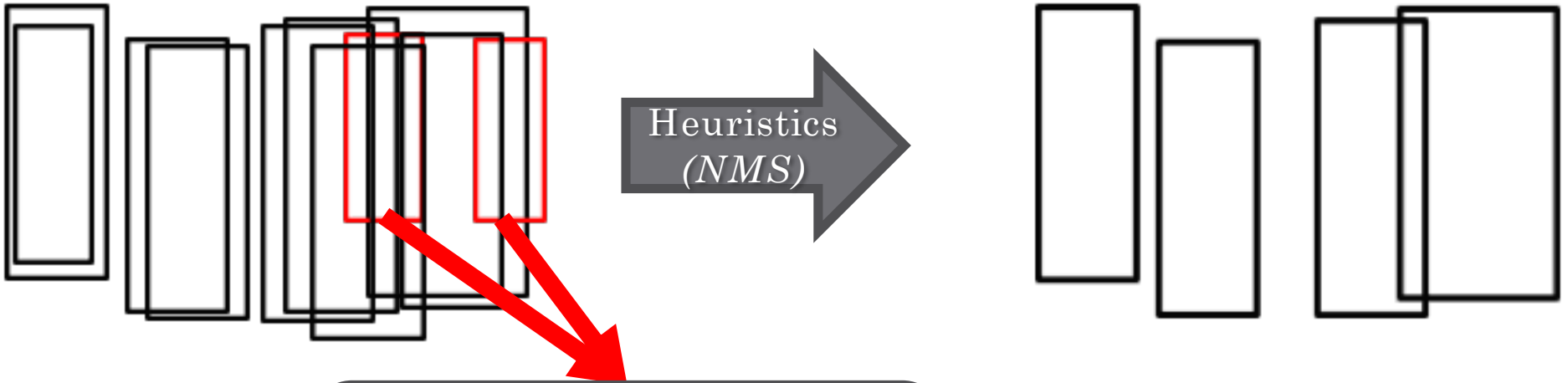



✘ Do not perform well in cluttered scenes (strong occlusions)

# Application Examples

- ✓ Object Detection and Localization

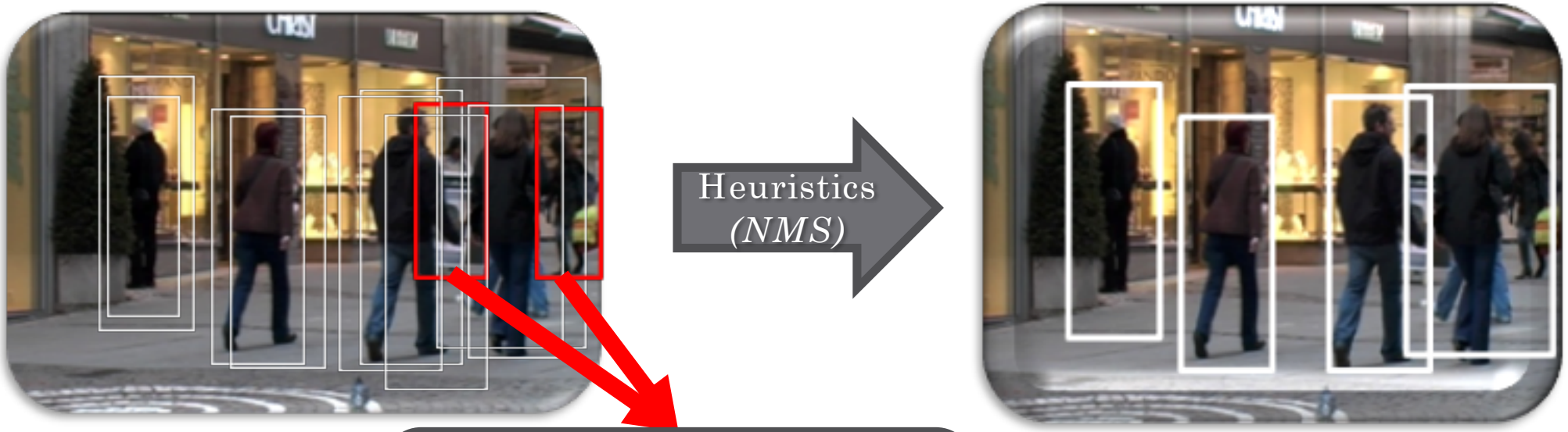
*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO



# Application Examples

- ✓ Object Detection and Localization

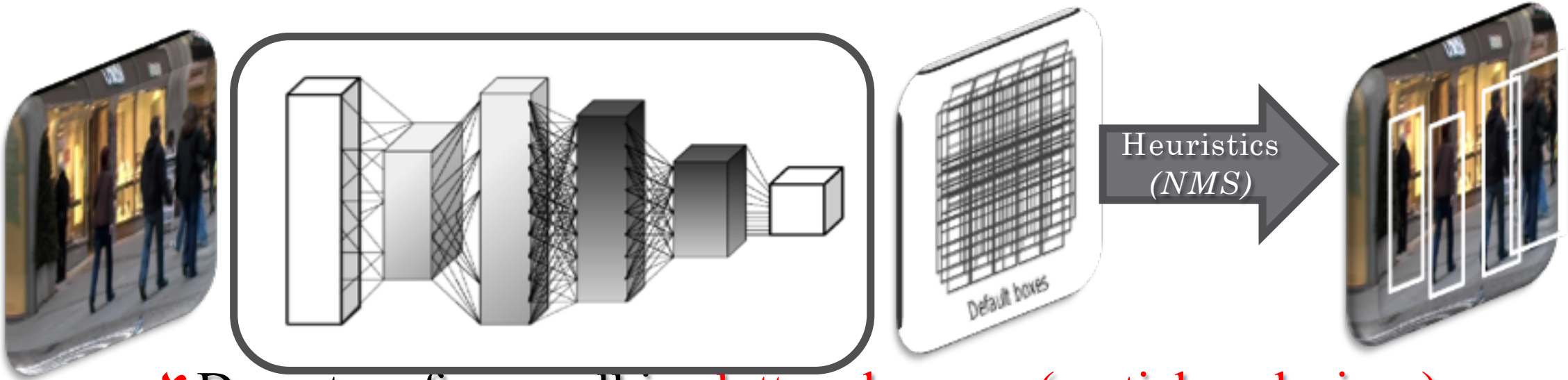
*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO



# Application Examples

✓ Object Detection and Localization

*State-of-the-art detectors*: Faster-RCNN, SSD &YOLO




✘ Do not perform well in cluttered scenes (partial occlusions)

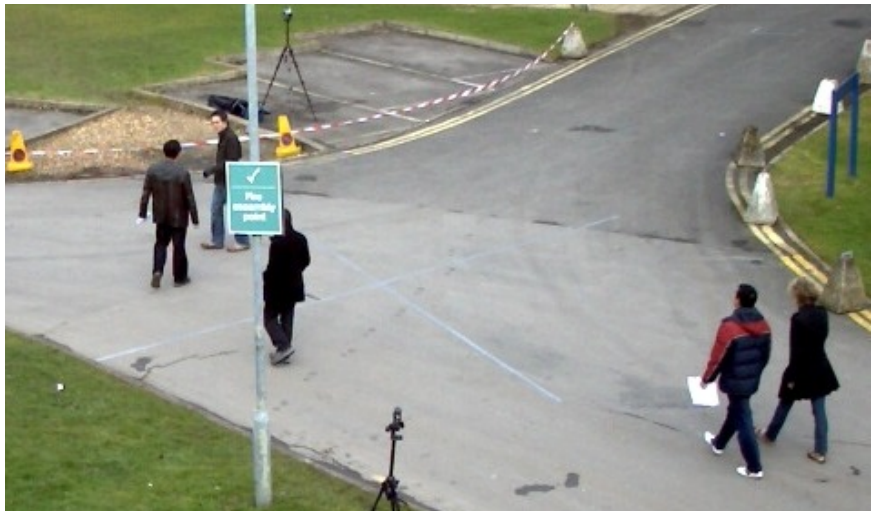
# Application Examples

- ✓ Object Detection and Localization

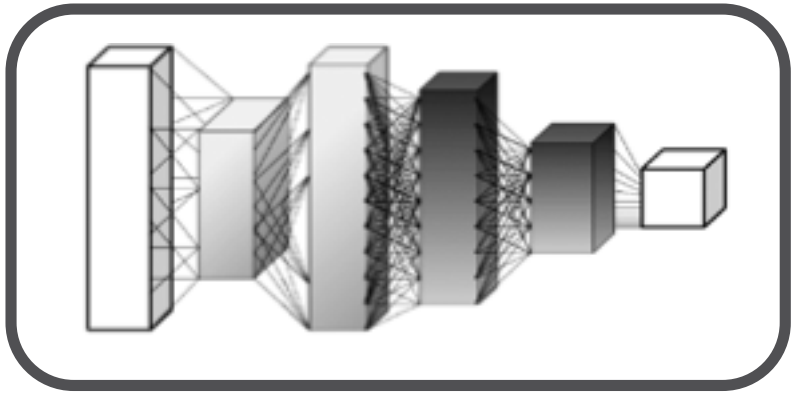
*Objective*: Train a network to detect 5 bounding boxes *e.g.* using regression

# Application Examples

- ✓ Object Detection and Localization

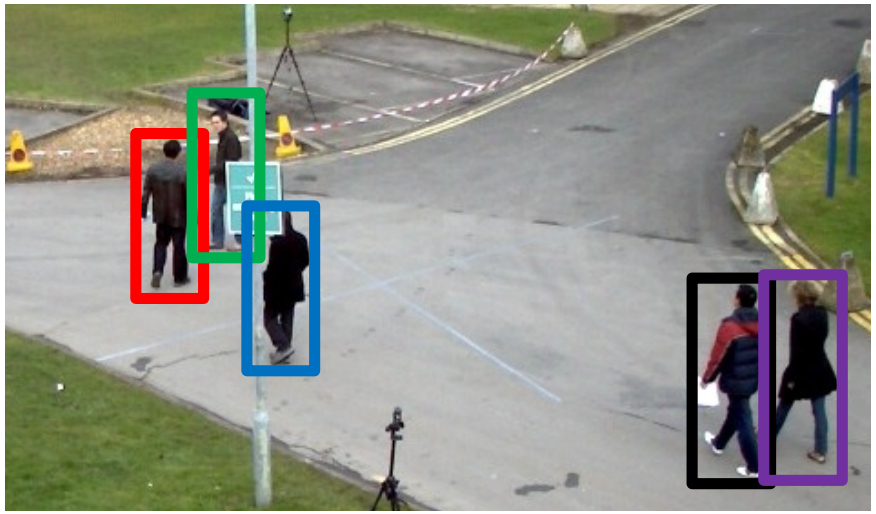


1st Training instance

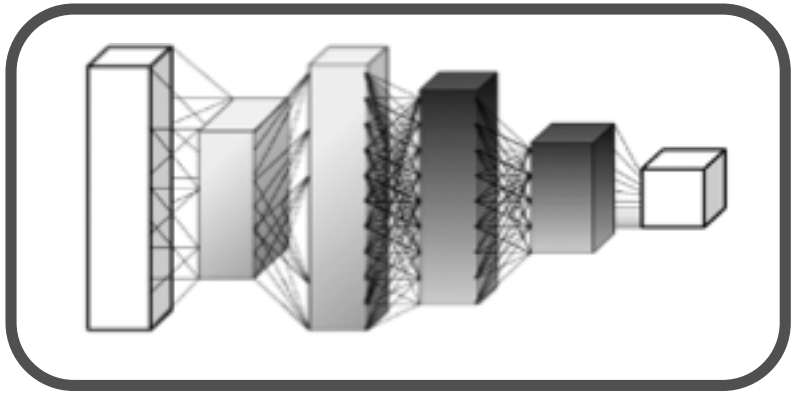


# Application Examples

- ✓ Object Detection and Localization

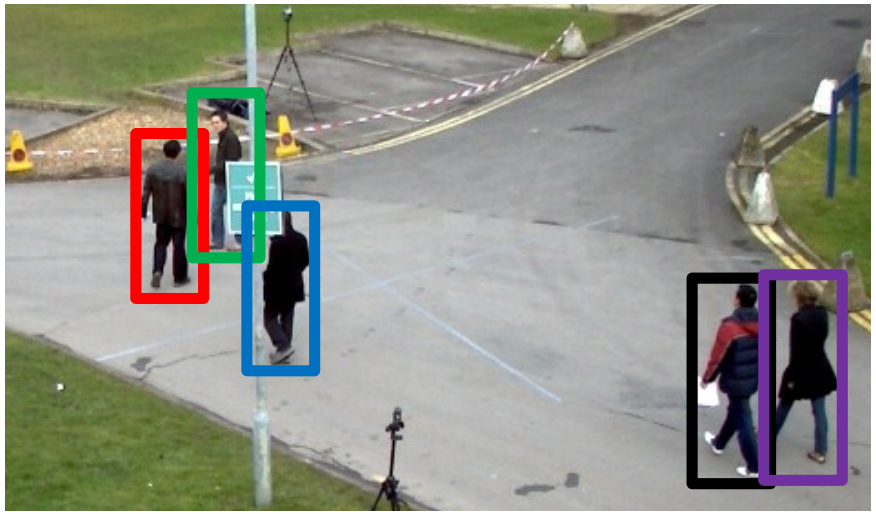


1st Training instance

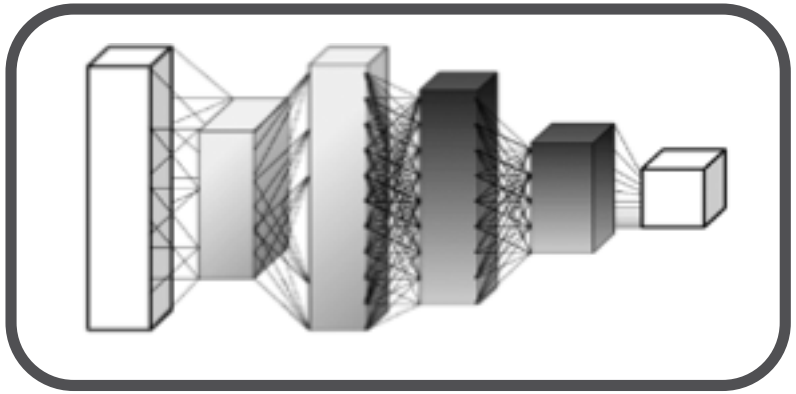


# Application Examples

- ✓ Object Detection and Localization



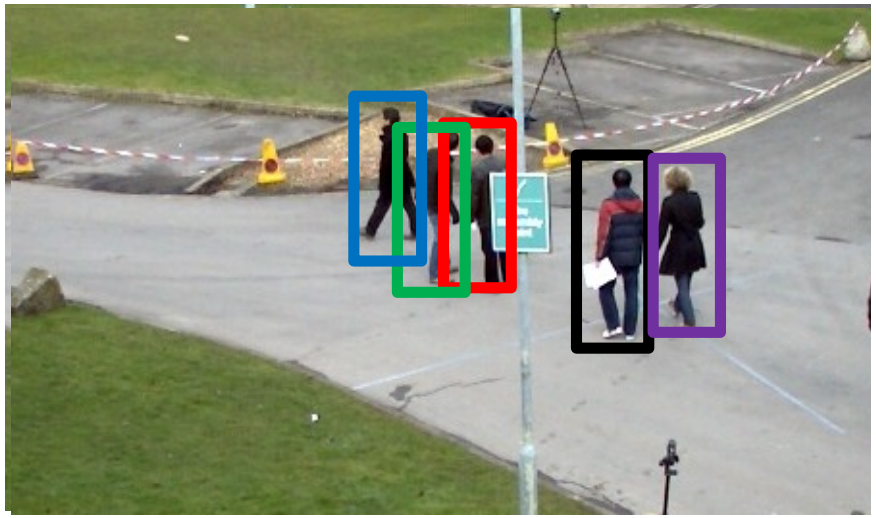
1st Training instance



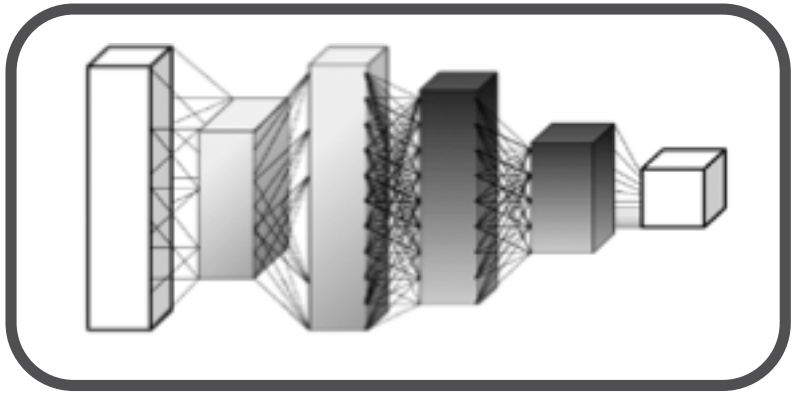
$$Y = \begin{bmatrix} x_1 & y_1 & w_1 & h_1 \\ x_2 & y_2 & w_2 & h_2 \\ x_3 & y_3 & w_3 & h_3 \\ x_4 & y_4 & w_4 & h_4 \\ x_5 & y_5 & w_5 & h_5 \end{bmatrix}$$

# Application Examples

- ✓ Object Detection and Localization



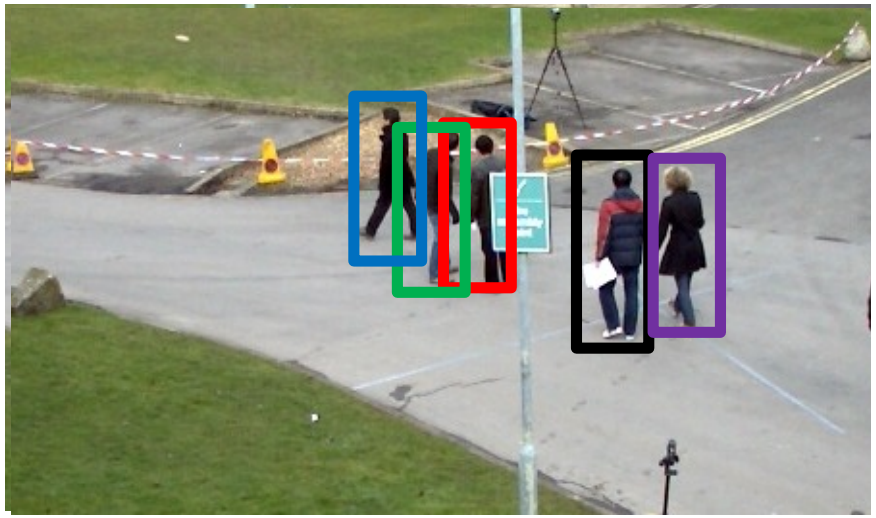
2nd Training instance



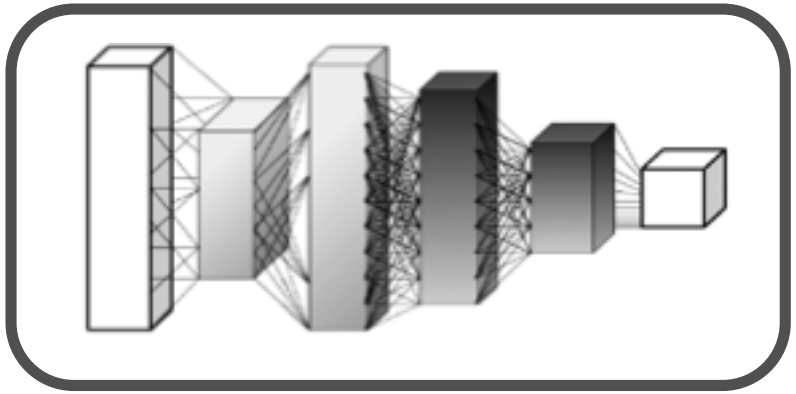
$$Y = \begin{bmatrix} x_3 & y_3 & w_3 & h_3 \\ x_2 & y_2 & w_2 & h_2 \\ x_1 & y_1 & w_1 & h_1 \\ x_4 & y_4 & w_4 & h_4 \\ x_5 & y_5 & w_5 & h_5 \end{bmatrix}$$

# Application Examples

- ✓ Object Detection and Localization



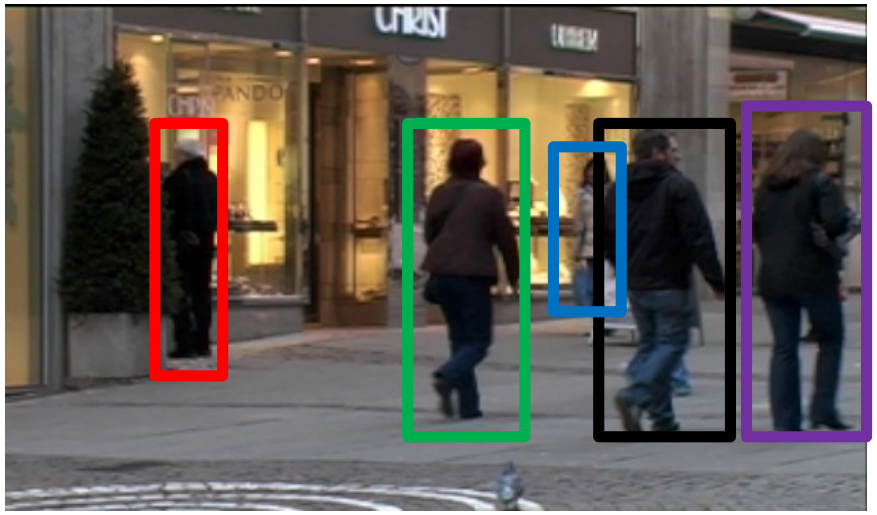
2nd Training instance



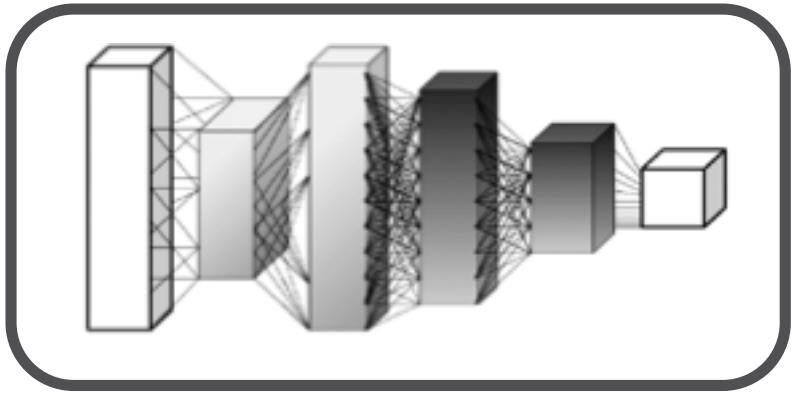
$$Y = \begin{bmatrix} x_3 & y_3 & w_3 & h_3 \\ x_2 & y_2 & w_2 & h_2 \\ x_1 & y_1 & w_1 & h_1 \\ x_4 & y_4 & w_4 & h_4 \\ x_5 & y_5 & w_5 & h_5 \end{bmatrix}$$

# Application Examples

- ✓ Object Detection and Localization



3rd Training instance

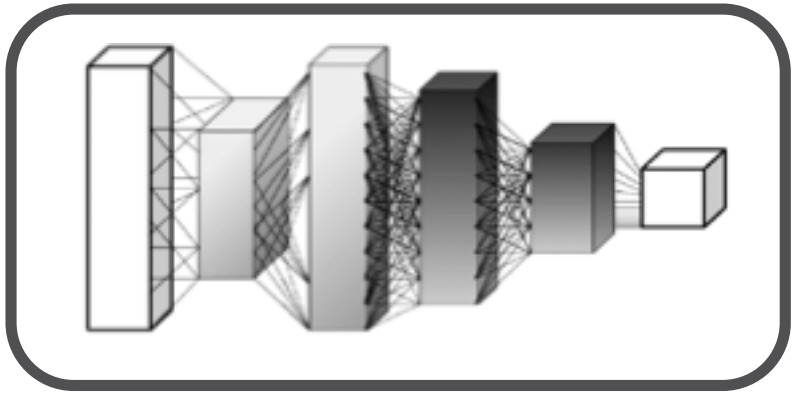


$$Y = \begin{bmatrix} x_1 & y_1 & w_1 & h_1 \\ x_2 & y_2 & w_2 & h_2 \\ x_3 & y_3 & w_3 & h_3 \\ x_4 & y_4 & w_4 & h_4 \\ x_5 & y_5 & w_5 & h_5 \end{bmatrix} ?$$

# Application Examples

- ✓ Object Detection and Localization

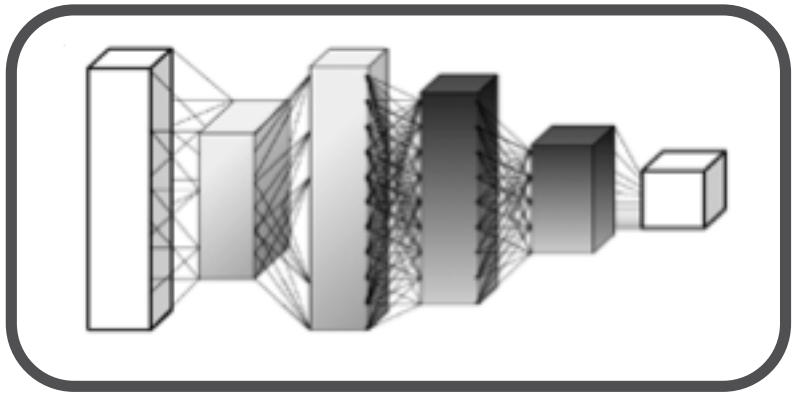
4th Training instance



$$Y = \begin{bmatrix} x_1 & y_1 & w_1 & h_1 \\ x_2 & y_2 & w_2 & h_2 \\ x_3 & y_3 & w_3 & h_3 \\ x_4 & y_4 & w_4 & h_4 \\ x_5 & y_5 & w_5 & h_5 \\ x_6 & y_6 & w_6 & h_6 \end{bmatrix}$$

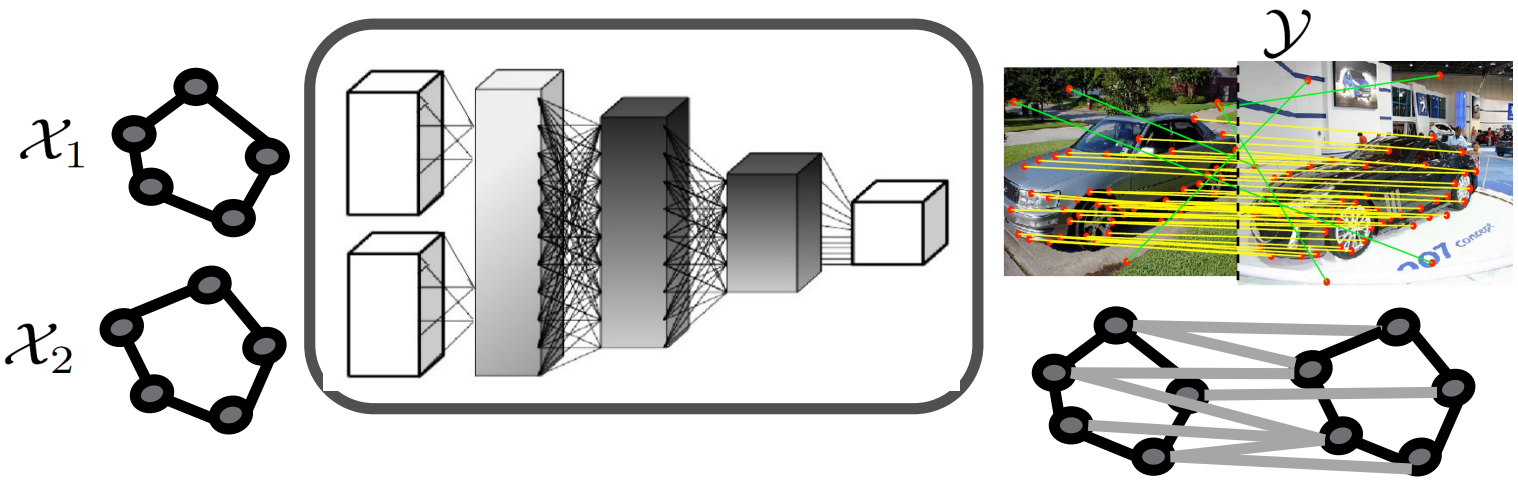
# Application Examples

- ✓ Object Detection and Localization
- ✓ Instance Level Segmentation



# Application Examples

- ✓ Object Detection and Localization
- ✓ Instance Level Segmentation
- ✓ Multi-Object Tracking
- ✓ Data Clustering
- ✓ Feature/Graph Matching or any Graph problem



# Set Learning

It is intuitive that **conventional machine learning approaches** cannot deal with the **set inputs and/or outputs**, where the dimensions are unknown and unfixed and their elements can be permuted.

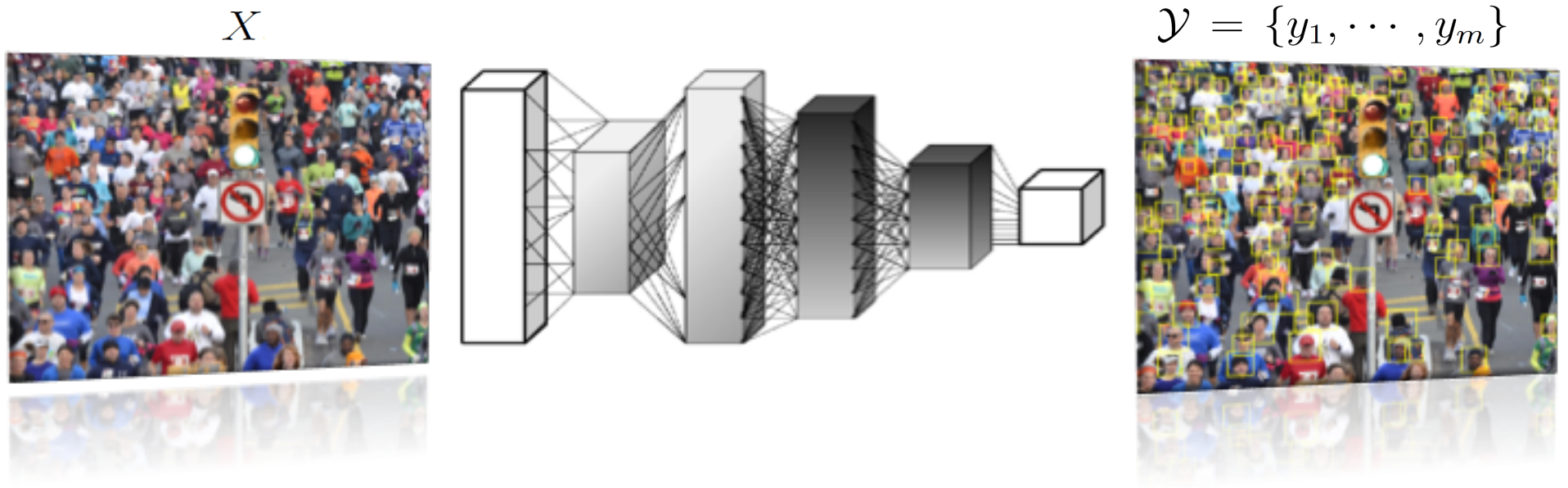
## Set learning an emerging field of study

- O. Vinyals *et al.* **Order matters**, *ICLR* 2015: using RNN to input and output an ordered set
- Rezatofighi *et al.* **DeepSetNet**, *ICCV 2017* (Spotlight): using CNN to output a set
- A. Smola *et al.* **Deep Sets**, *NIPS* 2017 (Oral): using CNN to input a set
- Rezatofighi *et al.* **Joint DeepSetNet**, *AAAI 2018*, using CNN to output a set
- Rezatofighi *et al.* **Deep Perm-Set Net**, *submitted to NIPS 2018*, using CNN to output a set with unknown permutation

# Technical Difficulties & Developments

## 1. Structured Input, Set output

**Potential Applications:** Object Detection, Instance level segmentation and *etc*



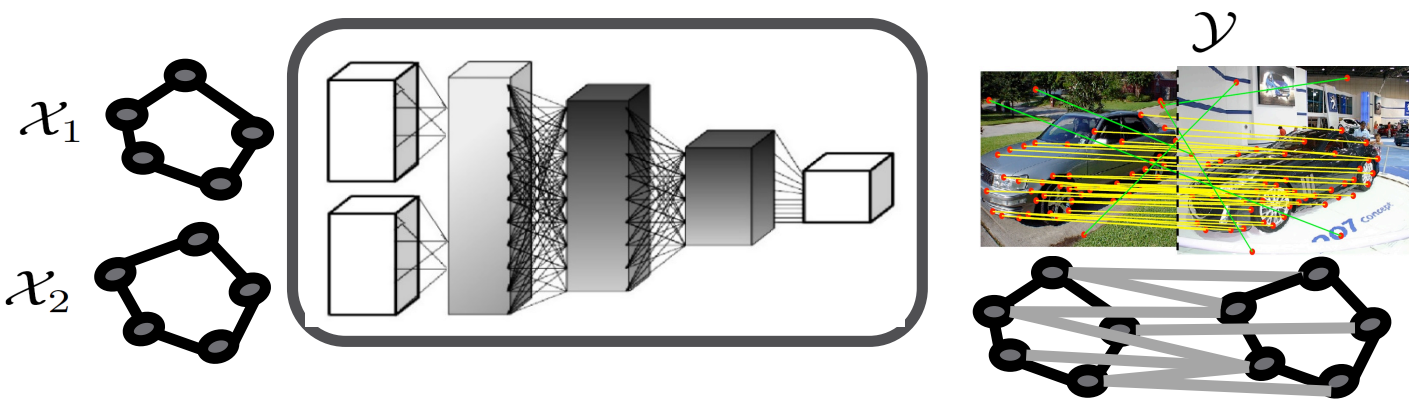
# Technical Difficulties & Developments

## 1. Structured Input, Set output

**Potential Applications:** Object Detection, Instance level segmentation and *etc*

## 2. Set Input, Set output

**Potential Applications:** Graph problems, *e.g.* matching



# Technical Difficulties & Developments

1. **Structured Input, Set output**

**Potential Applications:** Object Detection, Instance level segmentation and *etc*

2. **Set Input, Set output**

**Potential Applications:** Graph problems, *e.g.* matching

3. **Recurrent Set Network**

**Potential Application:** Real-time multiple object tracking

# Technical Difficulties & Developments

## 1. Structured Input, Set output

- ✓ Learning set cardinality as a discrete distribution, using the backpropagation algorithm [1]
- ✓ Joint learning of set cardinality and state distributions [2]
- ✓ Learning to make the outputs permutation invariants [3]


1. Rezatofighi *et al*. DeepSetNet: Prediciting Sets With Deep Neural Networks, ICCV 2017
2. Rezatofighi *et al*. Joint Learning of Set Cardinality and State Distribution, AAAI 2018
3. Rezatofighi *et al*. Deep Perm-Set Net, *submitted* to NIPS 2018

# Deep Perm-Set Net

## Some Descriptions

$$\mathcal{Y} = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}$$ ⇐ a set with **unknown** cardinality $m$ and permutation

# Deep Perm-Set Net

## Some Descriptions

$$\mathcal{Y} = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}$$

$$\mathcal{Y}^m = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}^m$$

a set with **known** cardinality $m$
but **unknown** permutation

# Deep Perm-Set Net

## Some Descriptions

$$\mathcal{Y} = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}$$

$$\mathcal{Y}^m = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}^m$$

$$\mathbf{Y}_{\boldsymbol{\pi}}^m = (\mathbf{y}_{\pi_1}, \cdots, \mathbf{y}_{\pi_m})$$

a set with **known** cardinality $m$ and permutation $\boldsymbol{\pi}$, where

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$$

# Deep Perm-Set Net

## Some Descriptions

$$\mathcal{Y} = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}$$

$$\mathcal{Y}^m = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}^m$$

$$\mathbf{Y}_{\boldsymbol{\pi}}^m = (\mathbf{y}_{\pi_1}, \cdots, \mathbf{y}_{\pi_m}) \Leftarrow$$

a set with **known** cardinality $m$ and permutation $\boldsymbol{\pi}$, where

$$\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_m)$$

It is simply a tensor !!

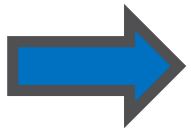
# Deep Perm-Set Net

## Some Descriptions

$$\mathcal{Y} = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}$$

$$\mathcal{Y}^m = \{\mathbf{y}_1, \cdots, \mathbf{y}_m\}^m$$

$$\mathbf{Y}_{\boldsymbol{\pi}}^m = (\mathbf{y}_{\pi_1}, \cdots, \mathbf{y}_{\pi_m})$$



$$\mathcal{Y}^m := \{\mathbf{Y}_{\boldsymbol{\pi}}^m | \forall \boldsymbol{\pi} \in \boldsymbol{\Pi}\}$$

It can be expressed by an ordered set with any arbitrary permutation, where

$$\boldsymbol{\Pi} = \{\boldsymbol{\pi}_1, \boldsymbol{\pi}_2, \cdots, \boldsymbol{\pi}_{m!}\} \text{ and } |\boldsymbol{\Pi}| := m!$$

# Deep Perm-Set Net

The probability density function for a set

$$p(\mathcal{Y}|\mathbf{x}, \mathbf{w}) = p(m|\mathbf{x}, \mathbf{w}) \times U^m \times p_m(\mathcal{Y}^m|\mathbf{x}, \mathbf{w})$$

Output as a set

# Deep Perm-Set Net

The probability density function for a set

$$p(\mathcal{Y}|\mathbf{x}, \mathbf{w}) = p(m|\mathbf{x}, \mathbf{w}) \times U^m \times p_m(\mathcal{Y}^m|\mathbf{x}, \mathbf{w})$$

Input as a tensor,
*e.g.* Image

# Deep Perm-Set Net

The probability density function for a set

$$p(\mathcal{Y}|\mathbf{x}, \mathbf{w}) = p(m|\mathbf{x}, \mathbf{w}) \times U^m \times p_m(\mathcal{Y}^m|\mathbf{x}, \mathbf{w})$$

Sets of parameters,
*e.g.* CNN parameters

# Deep Perm-Set Net

The probability density function for a set

$$p(\mathcal{Y}|\mathbf{x}, \mathbf{w}) = p(m|\mathbf{x}, \mathbf{w}) \times U^m \times p_m(\mathcal{Y}^m|\mathbf{x}, \mathbf{w})$$

Cardinality Distribution

# Deep Perm-Set Net

The probability density function for a set

$$p(\mathcal{Y}|\mathbf{x}, \mathbf{w}) = p(m|\mathbf{x}, \mathbf{w}) \times U^m \times p_m(\mathcal{Y}^m|\mathbf{x}, \mathbf{w})$$

The normalizer to avoid the unit mismatch across the different dimensions

# Deep Perm-Set Net

The probability density function for a set

$$p(\mathcal{Y}|\mathbf{x}, \mathbf{w}) = p(m|\mathbf{x}, \mathbf{w}) \times U^m \times p_m(\mathcal{Y}^m|\mathbf{x}, \mathbf{w})$$

Joint distribution of
$m$ elements of the set

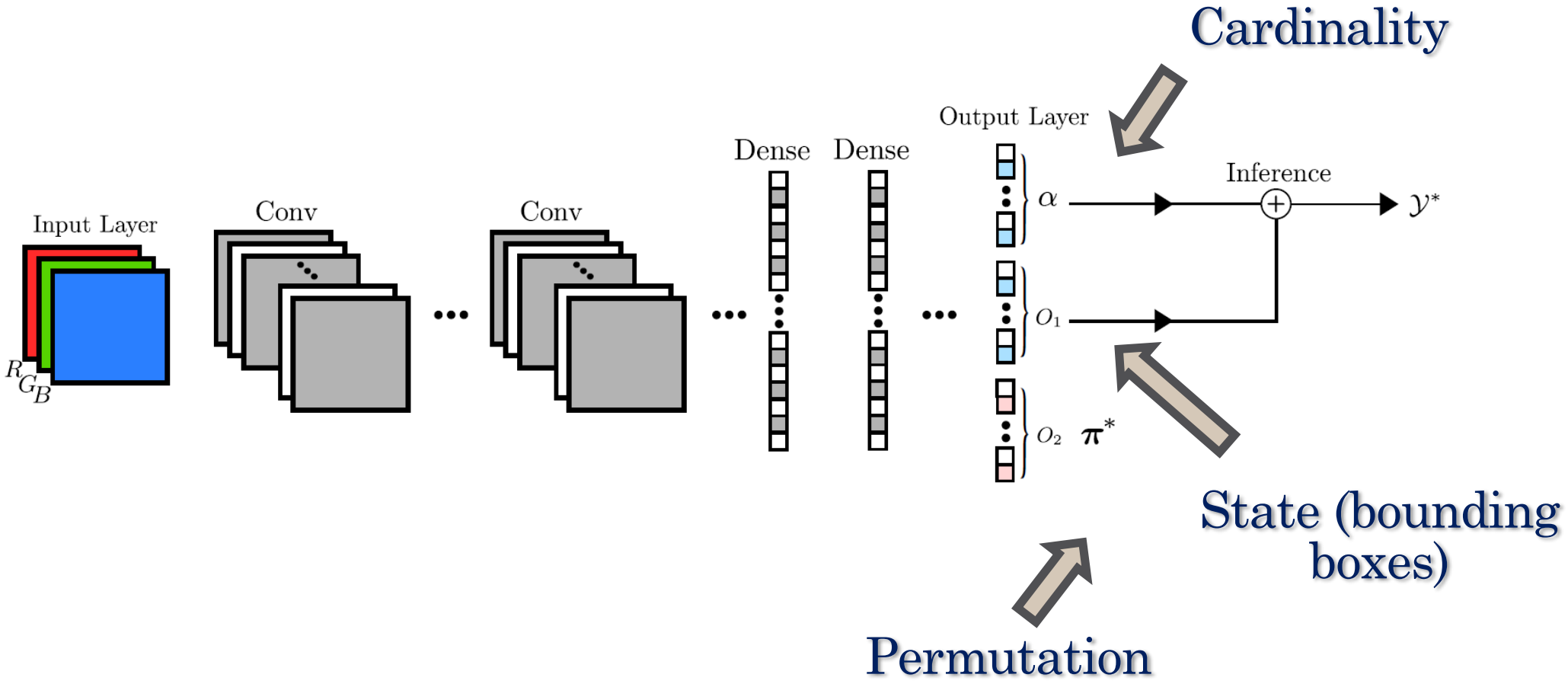
# Deep Perm-Set Net

The probability density function for a set

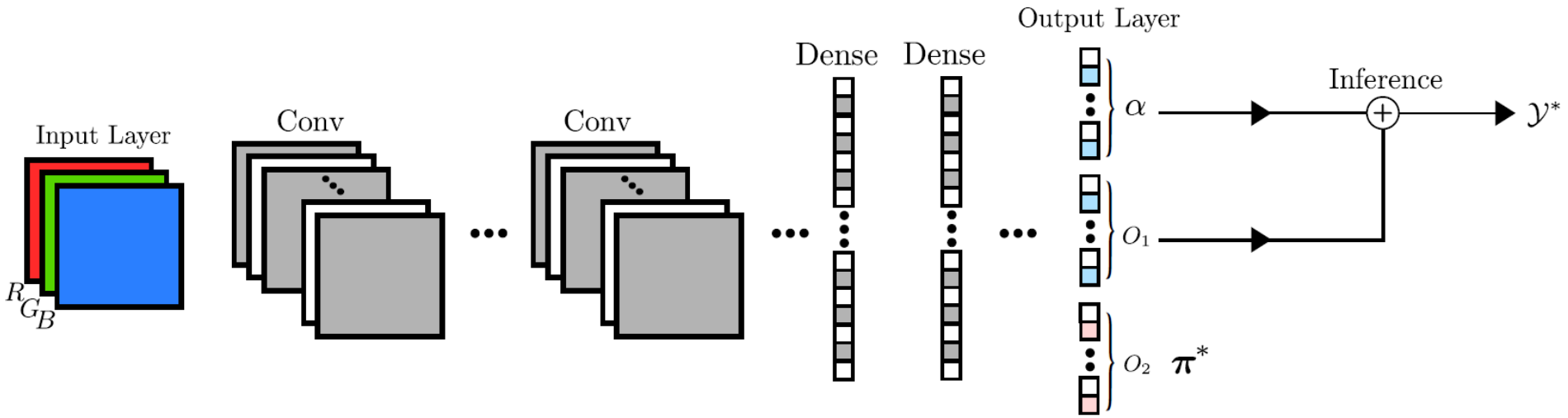
$$p(\mathcal{Y}|\mathbf{x},\mathbf{w}) = p(m|\mathbf{x},\mathbf{w}) \times U^m \times \sum_{\forall \boldsymbol{\pi} \in \boldsymbol{\Pi}} p_m(\mathbf{Y}^m_{\boldsymbol{\pi}}, \boldsymbol{\pi}|\mathbf{x},\mathbf{w})$$

Marginalization over the permutations

# Deep Perm-Set Net

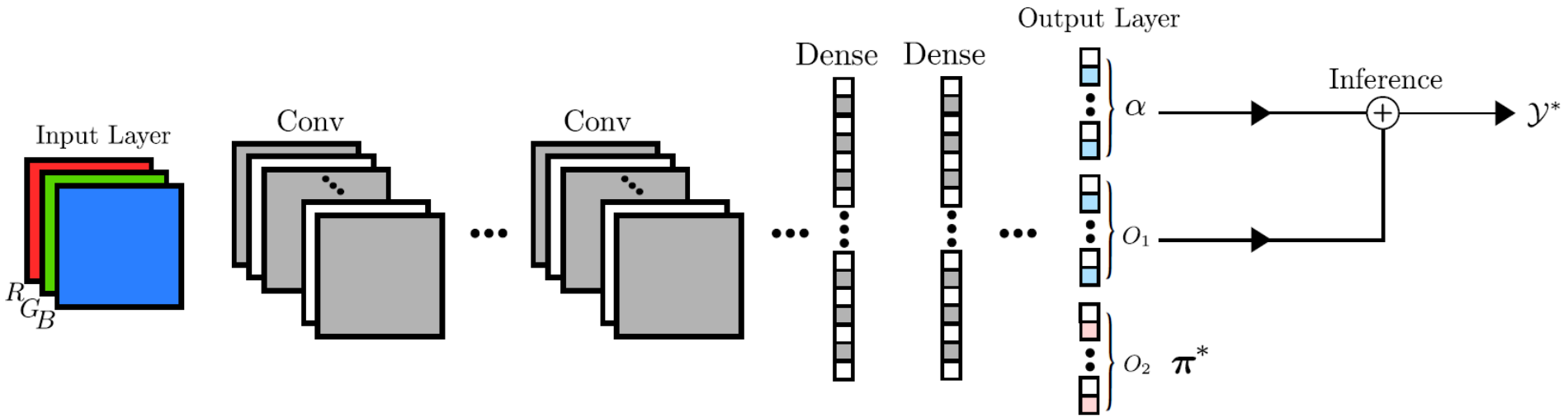


# Deep Perm-Set Net




Training: you compute the loss on the state ordered by a given permutation → Hungarian matching algorithm

# Deep Perm-Set Net




Testing: you only need to use the estimated cardinality and state.

# Experiments

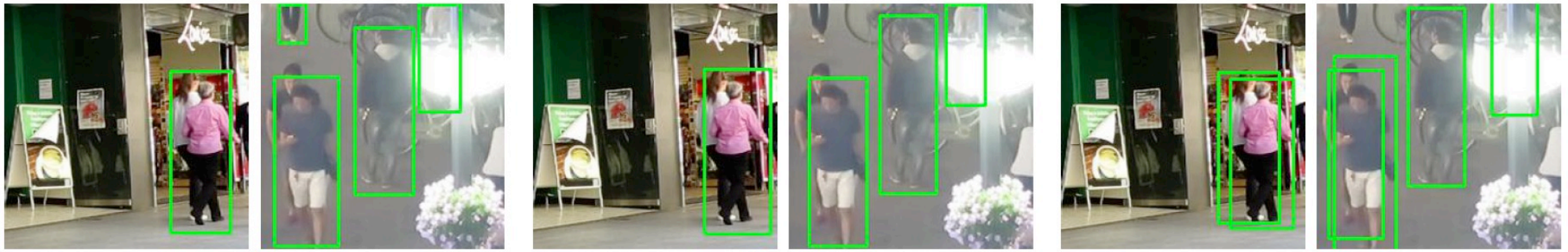
## Pedestrian Detection - MOTChallenge

Table 1: Detection results on the real data measured by average precision, the best F1 scores (higher is better) and log-average miss rate (lower is better).

| Method | AP ↑ | F1-score↑ | MR ↓ |
|---|---|---|---|
| Faster-RCNN | 0.68 | 0.76 | 0.48 |
| YOLO v2 | 0.68 | 0.76 | 0.48 |
| **Our Detector (w/o card.)** | **0.75** | **0.80** | **0.47** |
| **Our Detector** | – | 0.80 | |

# Experiments

## Pedestrian Detection - MOTChallenge

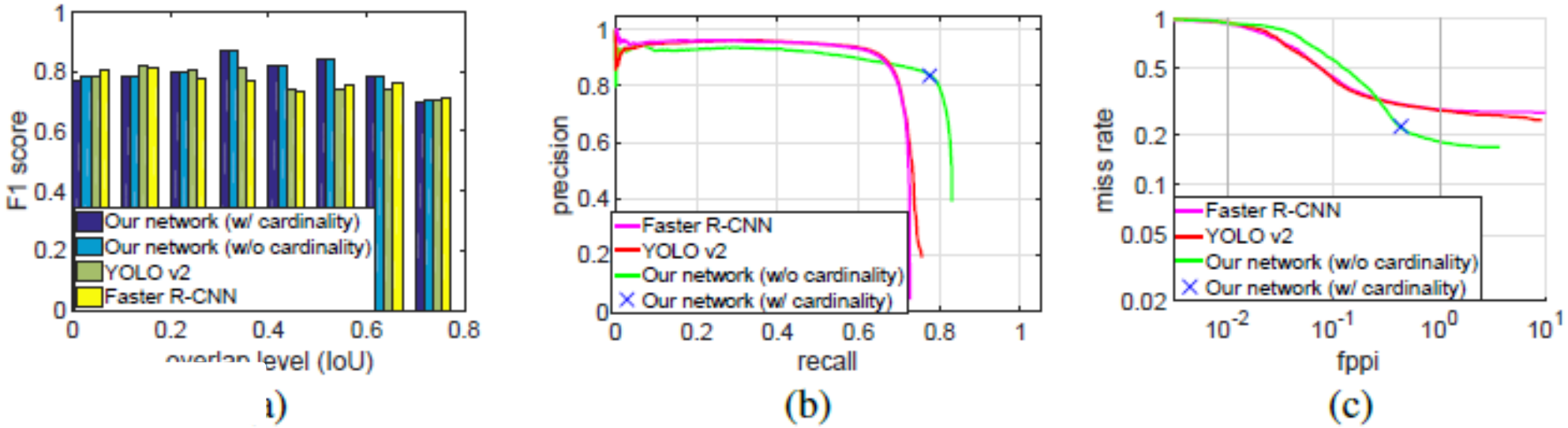


(a) (b) (c)

A comparison between the detection performance of (a) Faster-RCNN, (b) YOLO v2 and (c) our set detector on heavily overlapping pedestrians from *MOTChallenge* benchmark. Both Faster-RCNN and YOLO v2 fail to properly detect heavily occluded pedestrians due to the inevitable NMS heuristic.

# Experiments

## Pedestrian Detection - MOTChallenge




a) (b) (c)

Figure 3: (a) The best F1 scores against the level of object occlusions calculated by intersection of union (IoU), (b) Precision-Recall curve, and (c) ROC (miss rate-false positive per image) curve on pedestrian detection data for the competing detectors: Faster-RCNN, YOLO v2, our network (w/o cardinality) and our network (w/ cardinality). Our final detection results are also shown as a single point in the curves.

# Experiments

## Complex CAPTCHA test – De-Sum

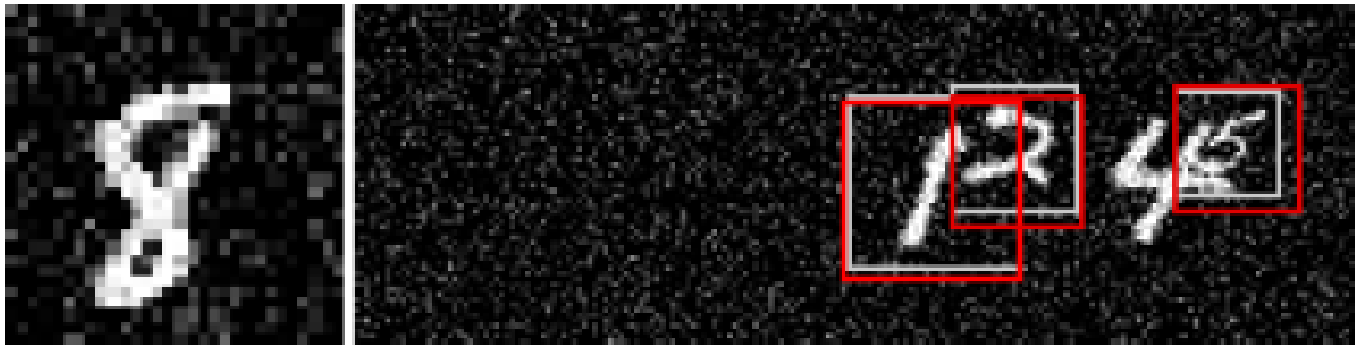


Figure 4: A *query digit* (left) and a *set of digits* (right) for the proposed CAPTCHA test. The ground truth and our predicted solutions are shown by white and red boxes respectively.

# Experiments

## Complex CAPTCHA test – De-Sum

Far more interesting outcome!! The ability of mimicking arithmetics without any rules being coded.

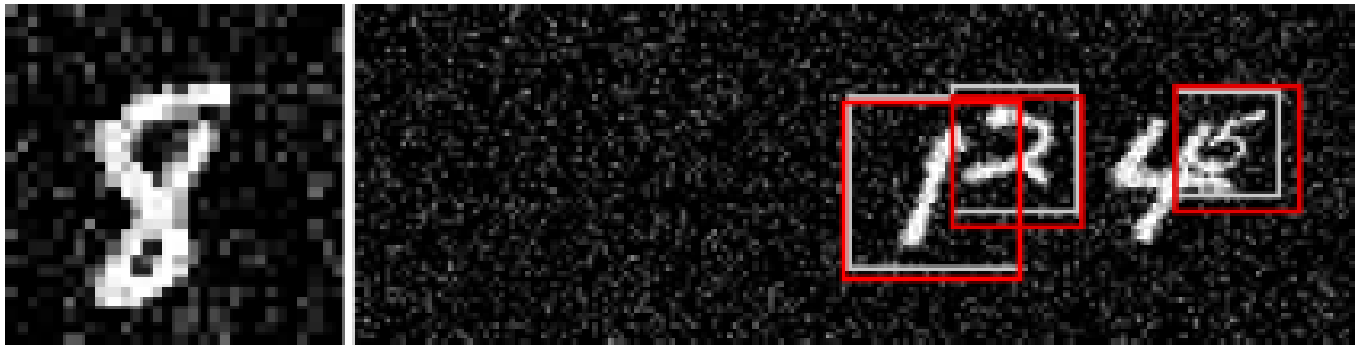
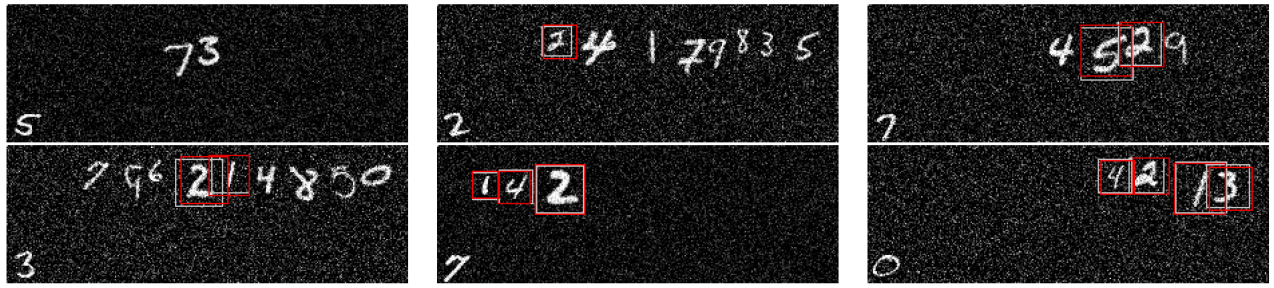


Figure 4: A *query digit* (left) and a *set of digits* (right) for the proposed CAPTCHA test. The ground truth and our predicted solutions are shown by white and red boxes respectively.

# Experiments

## Deep Perm-Set Net Vs. Faster-RCNN



Further examples showing a perfect prediction of the solution using our Deep Perm-Set Network. Each image contains a *query digit* (bottom left corner) and a *set of digits*. The ground truth and our predicted solutions are shown by white and red boxes respectively. Note that *zero* represents number 10 for our experiment.

Further examples showing the solution of Faster-RCNN for the test. Each image contains a *query digit* (bottom left corner) and a *set of digits*. The ground truth and the predicted solutions for Faster-RCNN are shown by white and red boxes respectively. Faster-RCNN simply learns to detect almost all the digits while ignoring the logical relationship between the *query digit* and the *set of digits*.

# Questions?