Bounding Box Improvement With Reinforcement Learning

Andrea Cleland

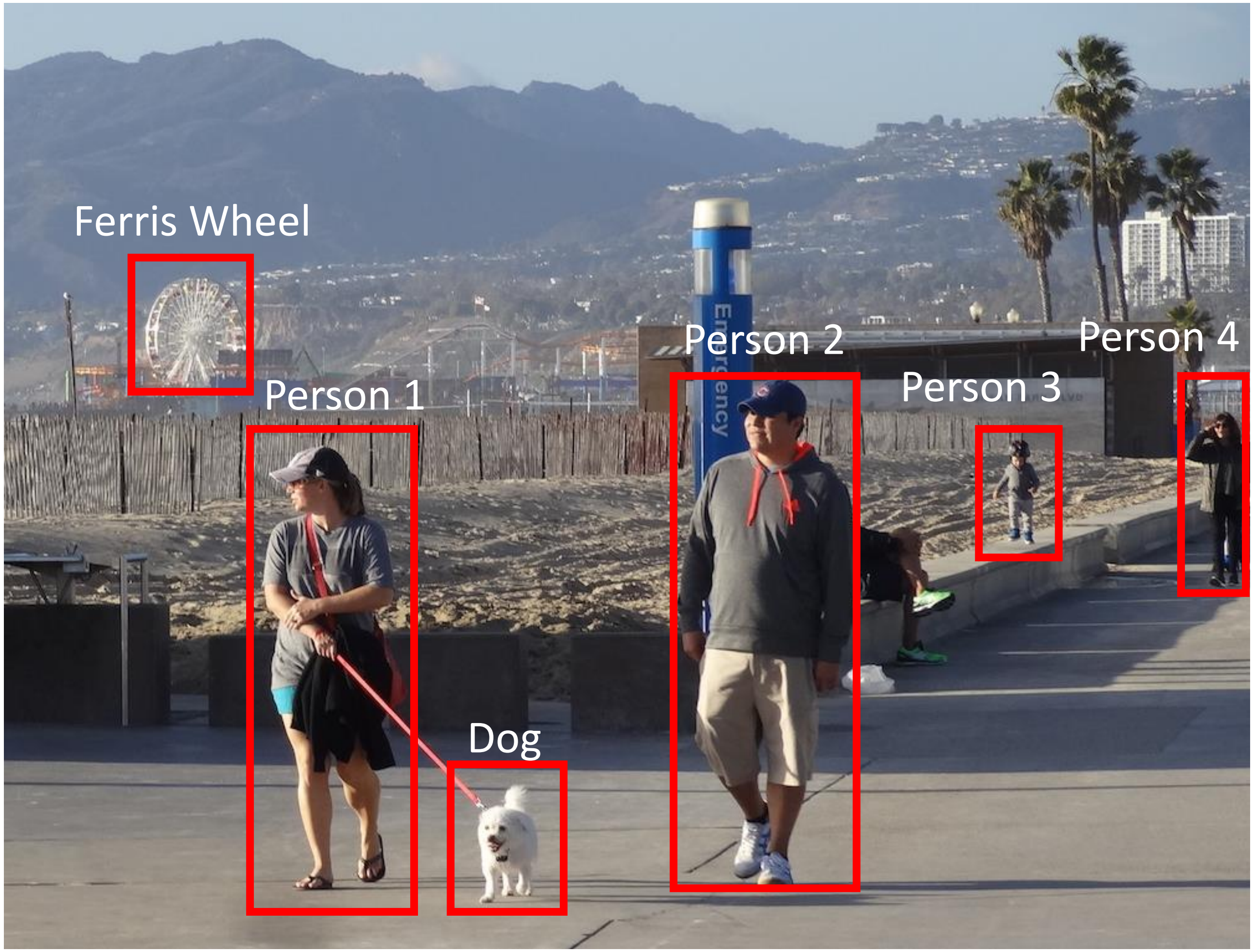
Master's Thesis Defense

Portland State University

2018

1. Introduction

Object localization

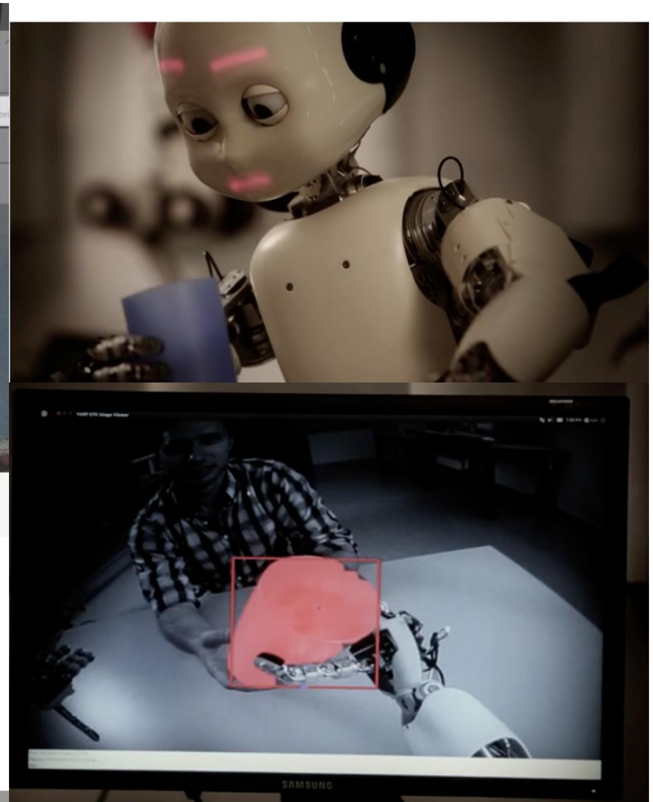


- Task: identify and locate objects in images

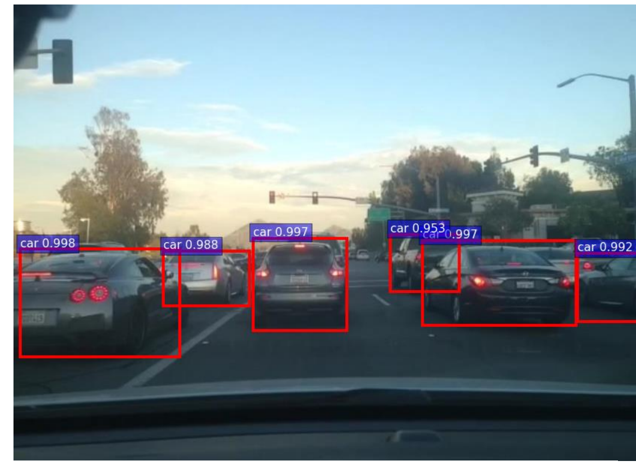
Object Localization Applications

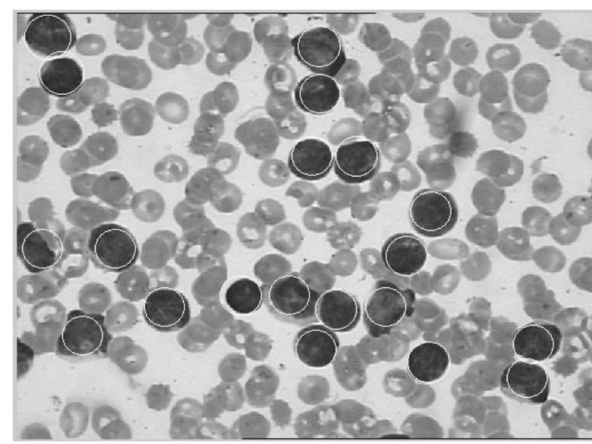
Cell Phone Apps

Robot Control



Self-Driving Cars





Medical Diagnosis

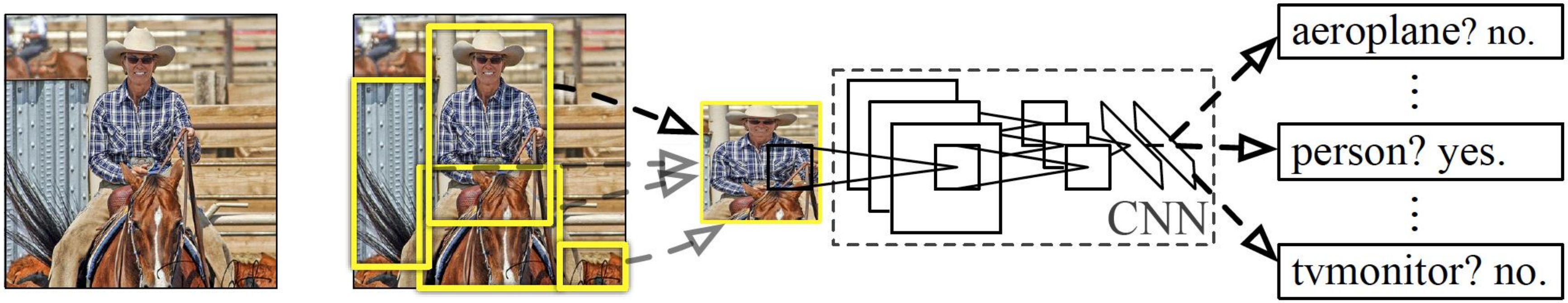
Object Localization

- Convolutional Neural Networks are very good at identifying objects, but localization is still a challenge

RCNN False Positives [Source]

Object Search

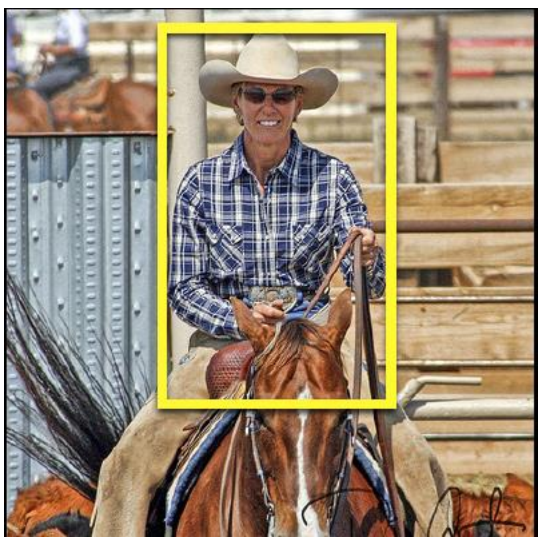
- Exhaustive sliding window approach is too slow
- Need to economize search:
- Generate object proposals based on likely locations
- Then do local search for object
  - When CNN detector has a positive identification, the bounding box may be a poor fit.
  - Need way to adjust box



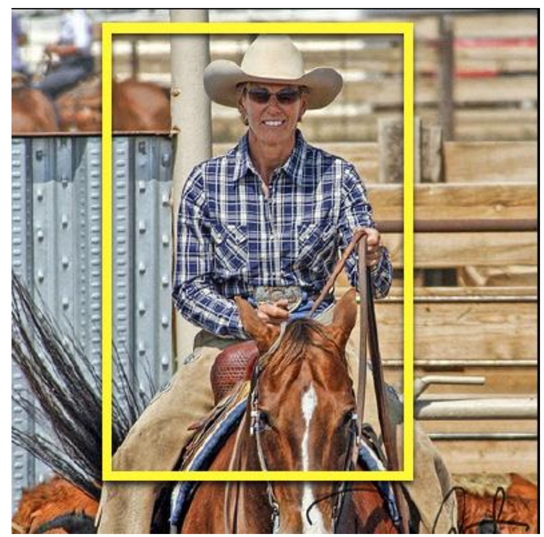
Bounding Box Regression (BBR)

- Extract CNN Features from proposed bounding box
- Estimate location and dimensions of true box through statistical regression on CNN features.

Bounding Box Proposal



Result of Bounding Box Regression



Ways to Improve Bounding Box Regression?

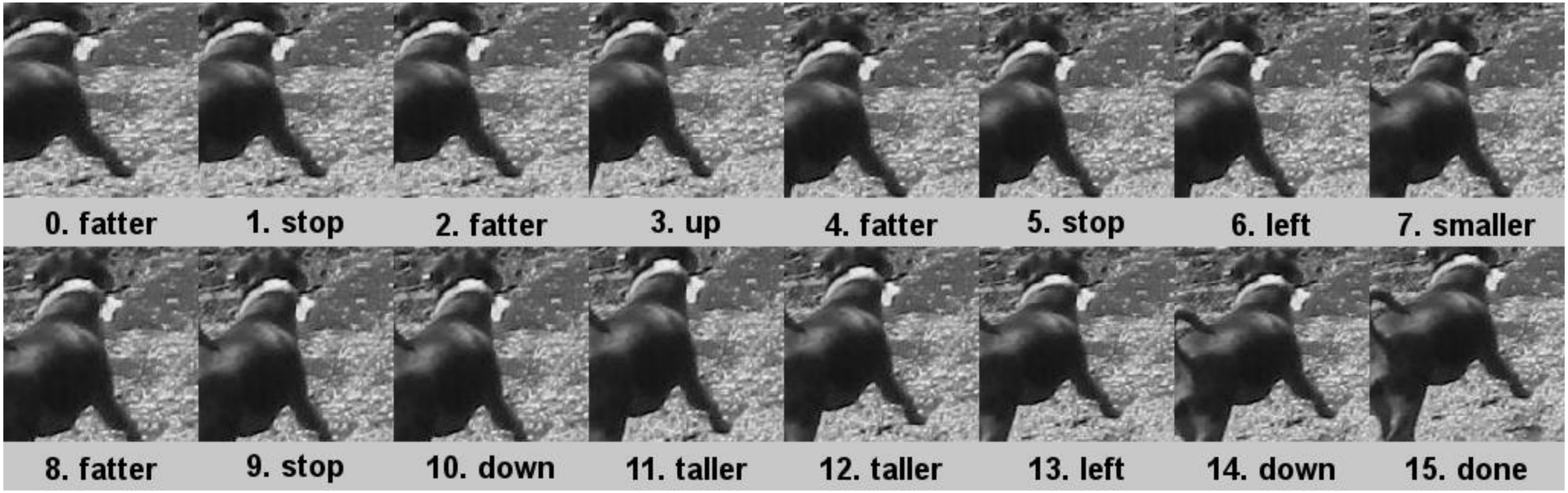
- BB Regression is only applied once – based on static analysis of features.

- Maybe an iterative active approach could work better?

My Algorithm

- Search policy aims to improve bounding box proposal through a sequence of transformative actions: {*up, down, left, right, bigger smaller, fatter, taller, stop*}

- Search policy is learned using reinforcement learning.



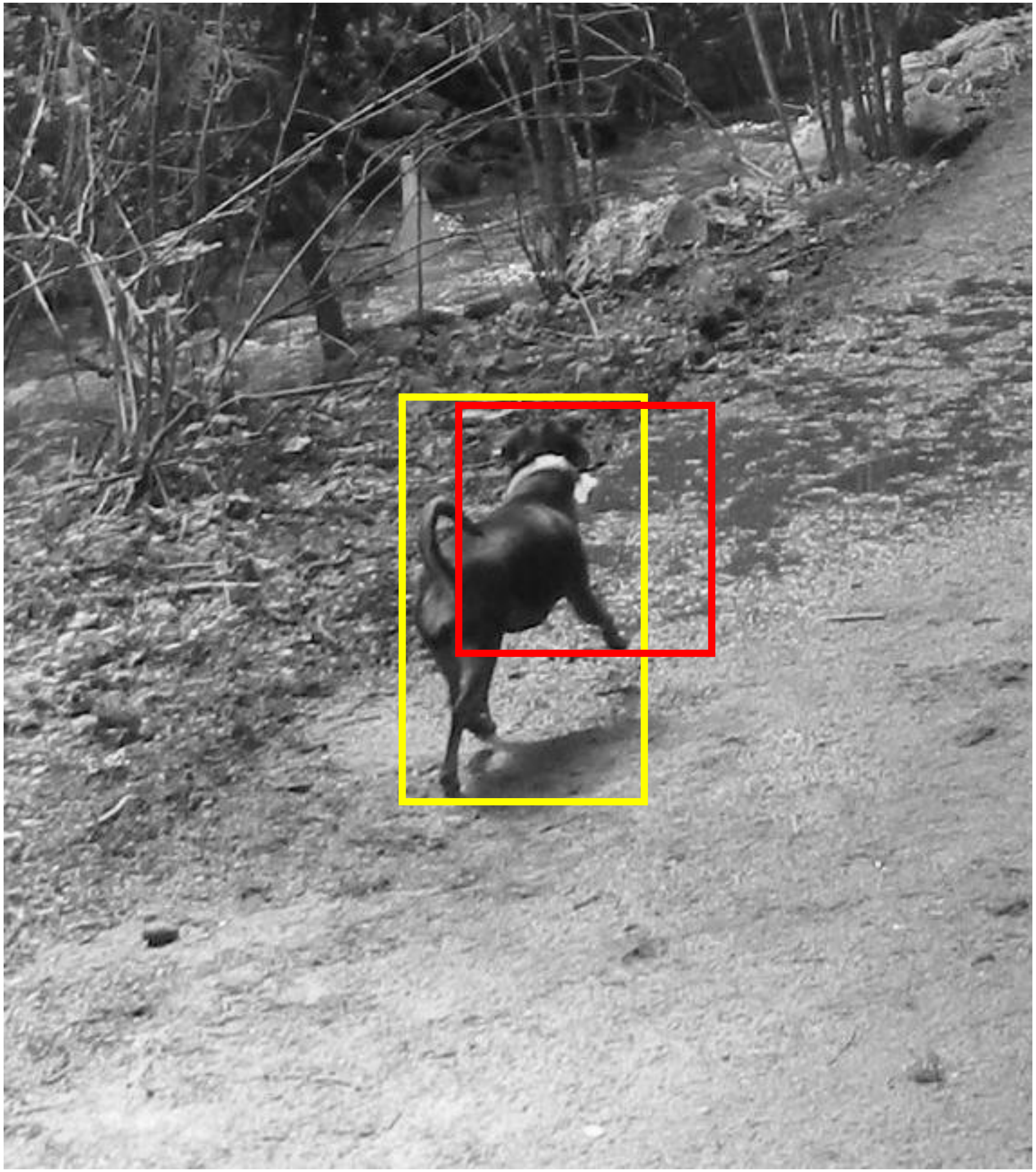
Initial box

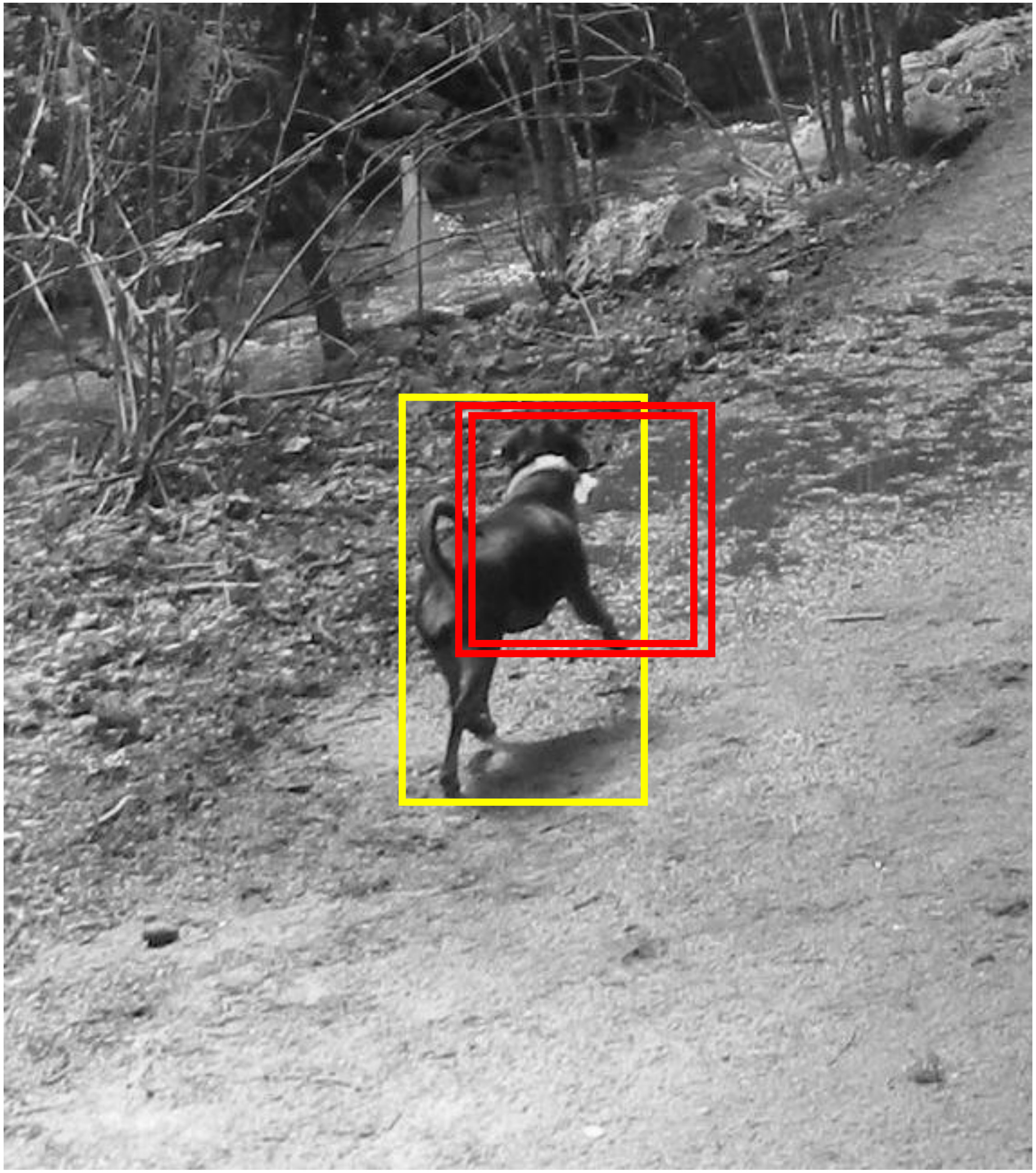
Fatter

Up

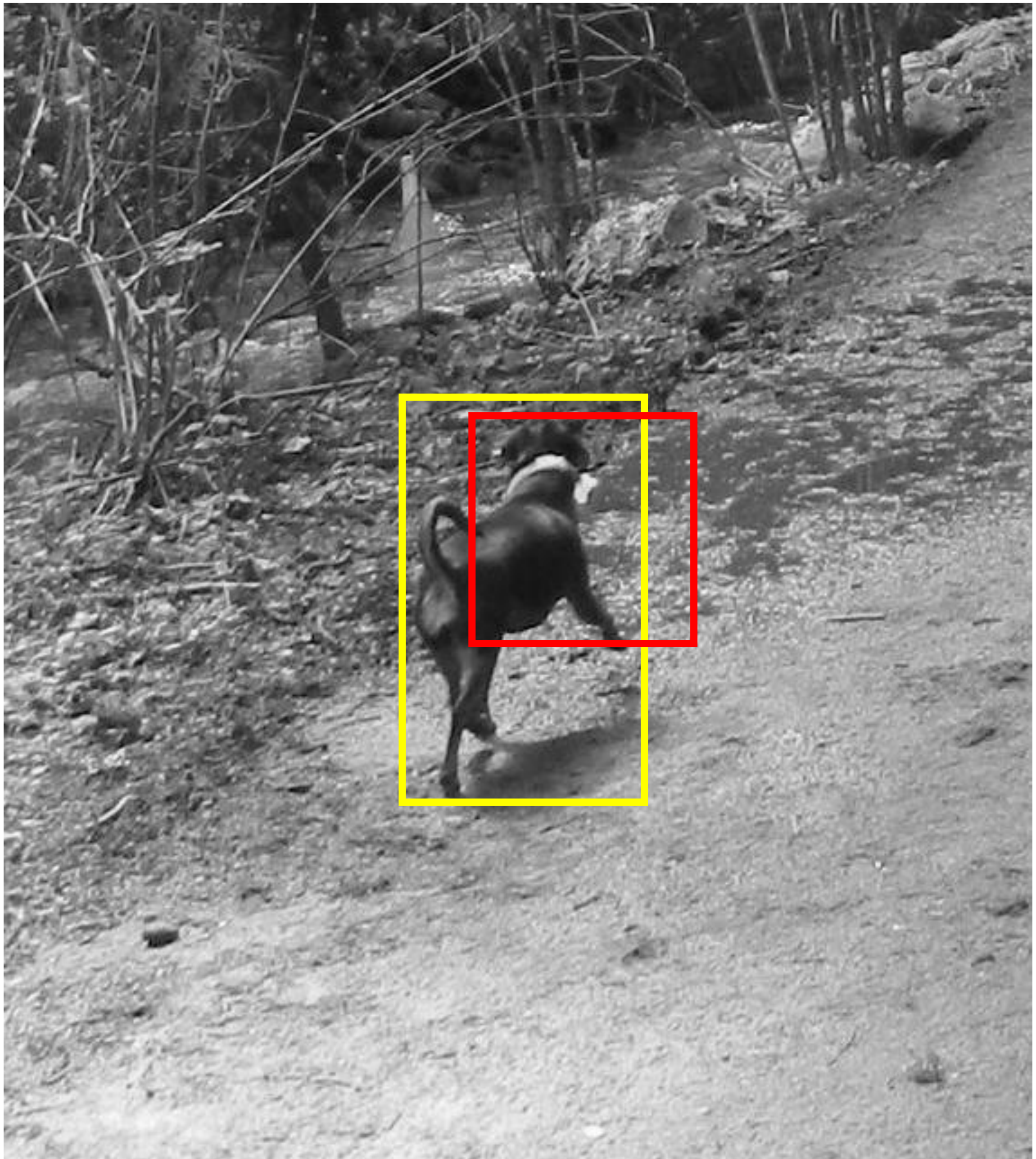
Fatter

Left

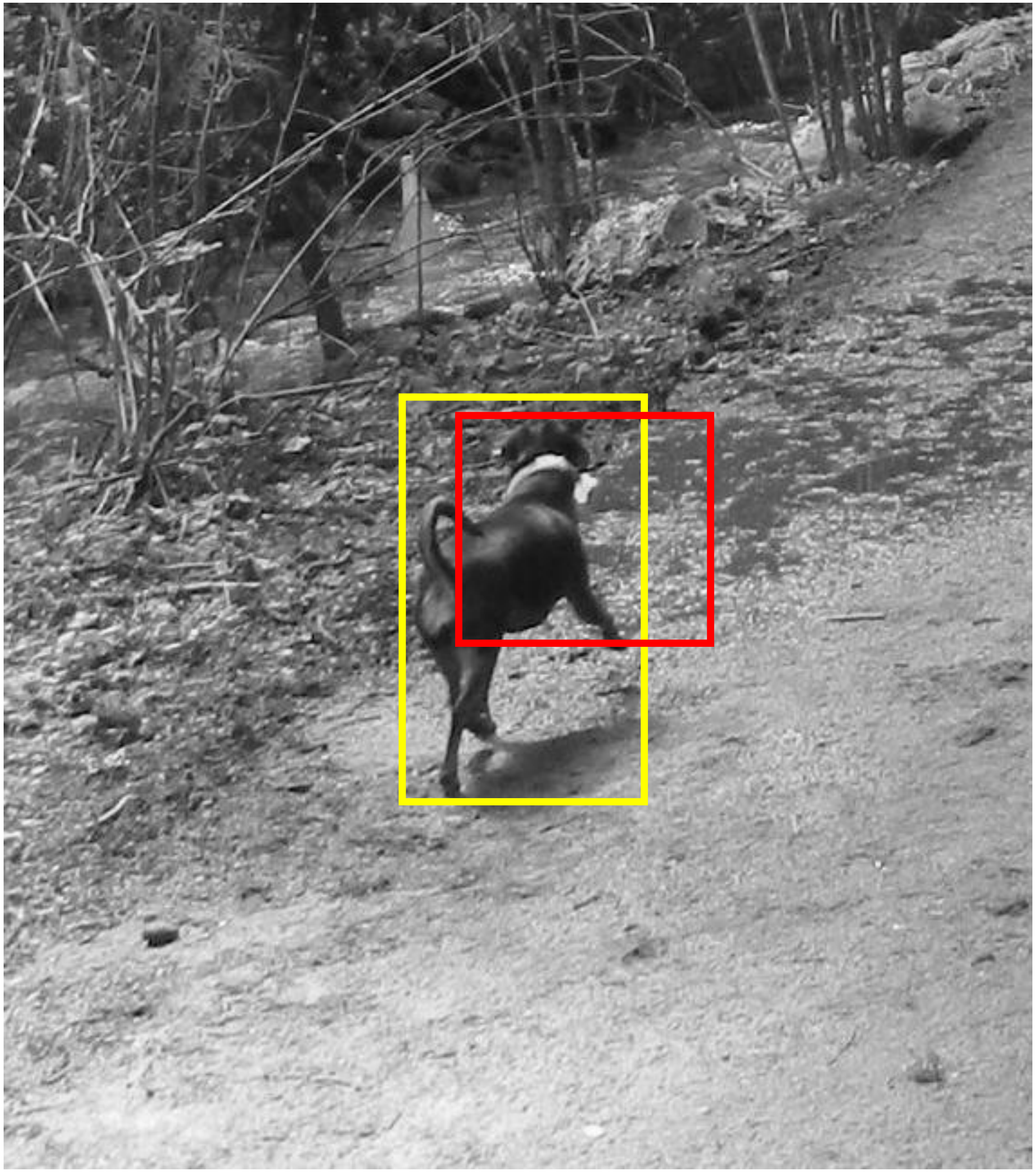


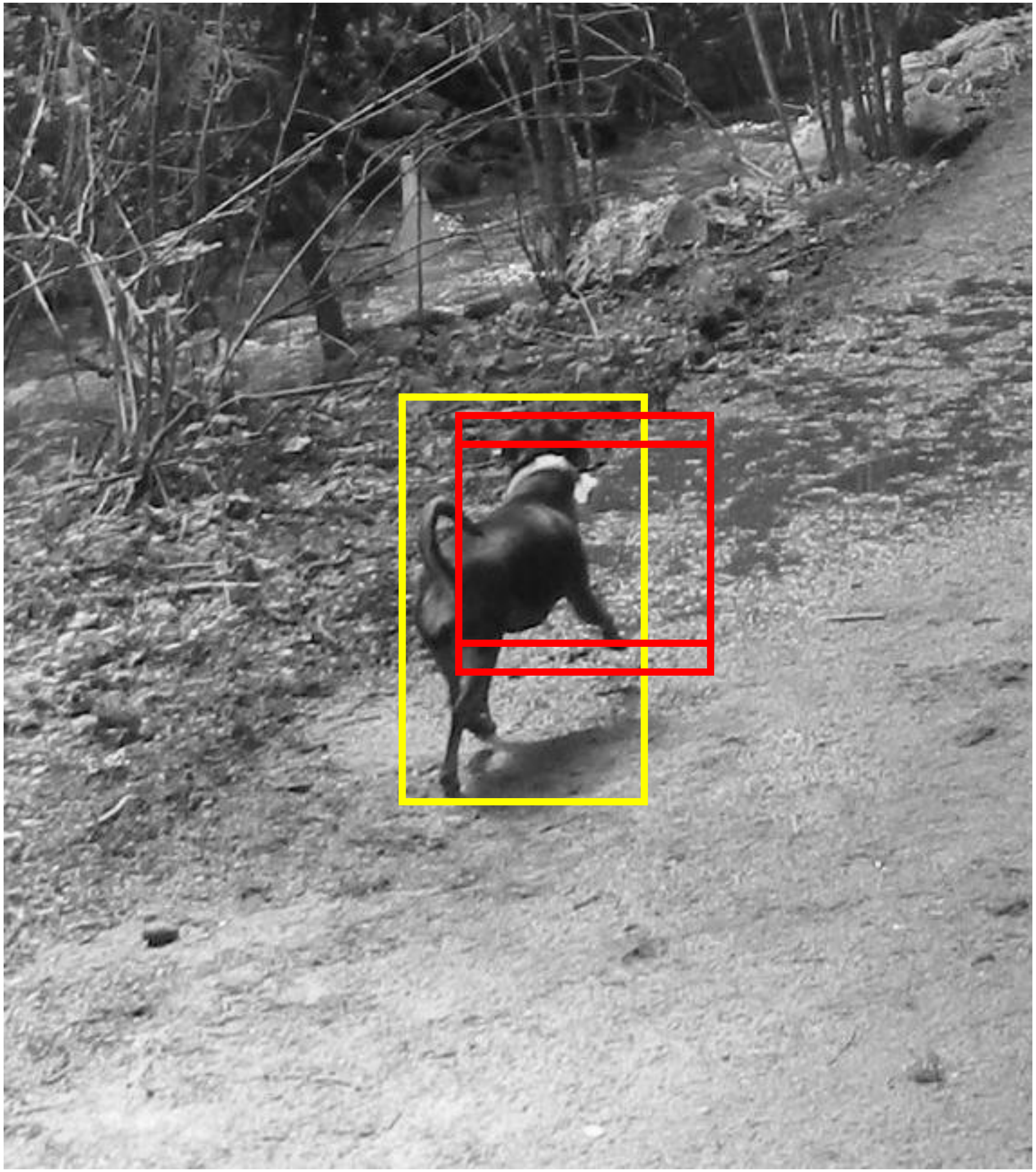


Smaller

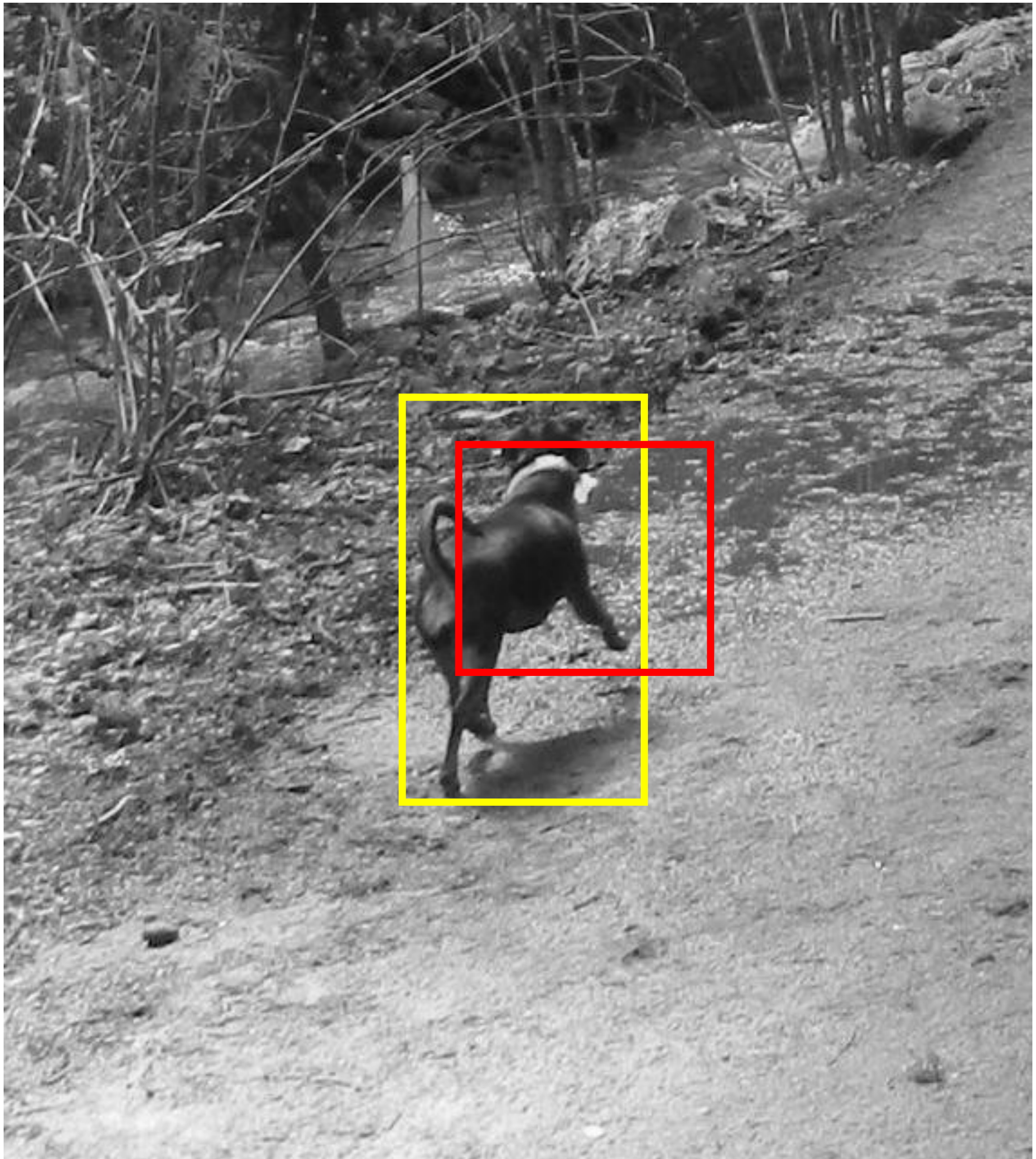


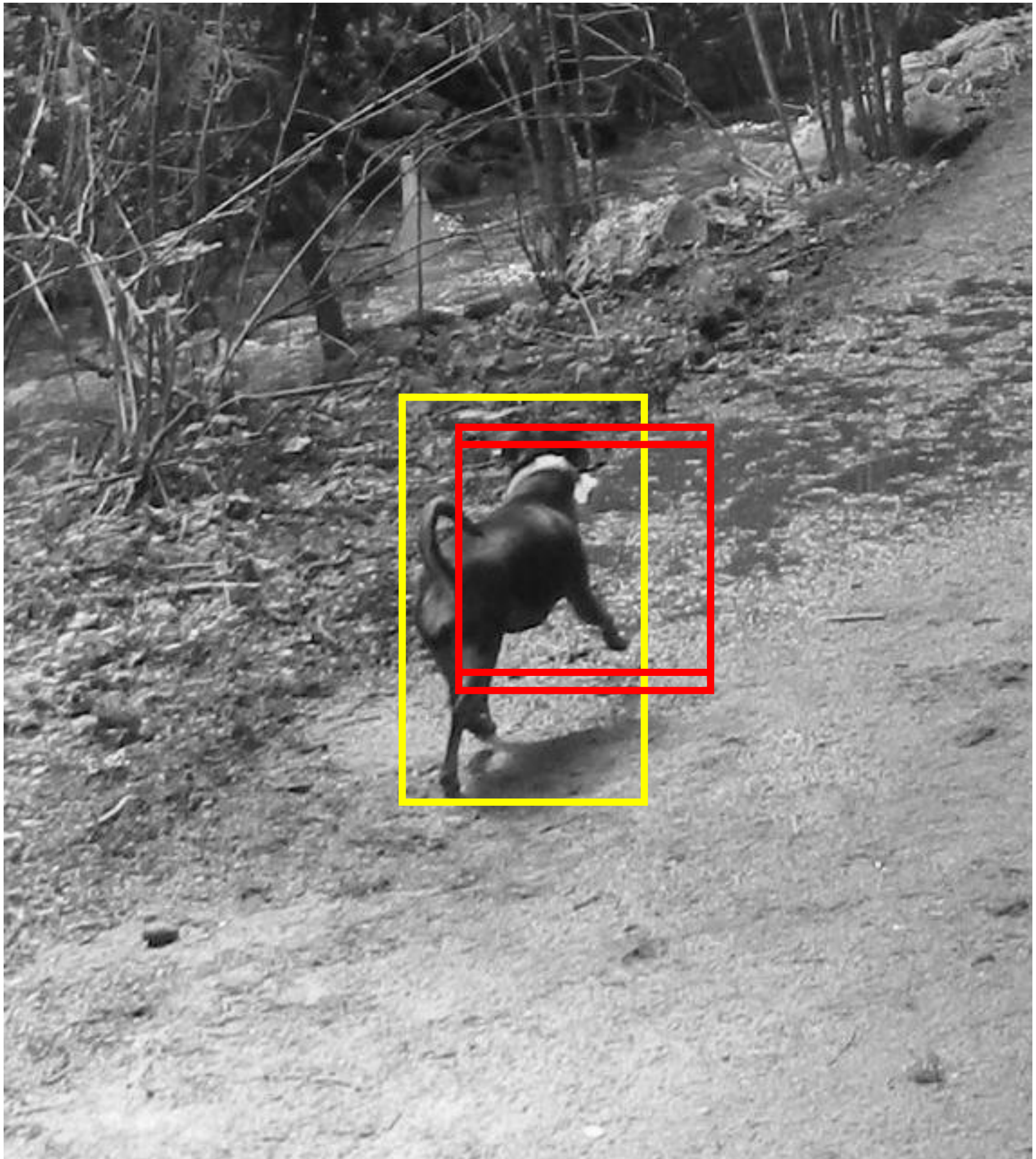
Fatter



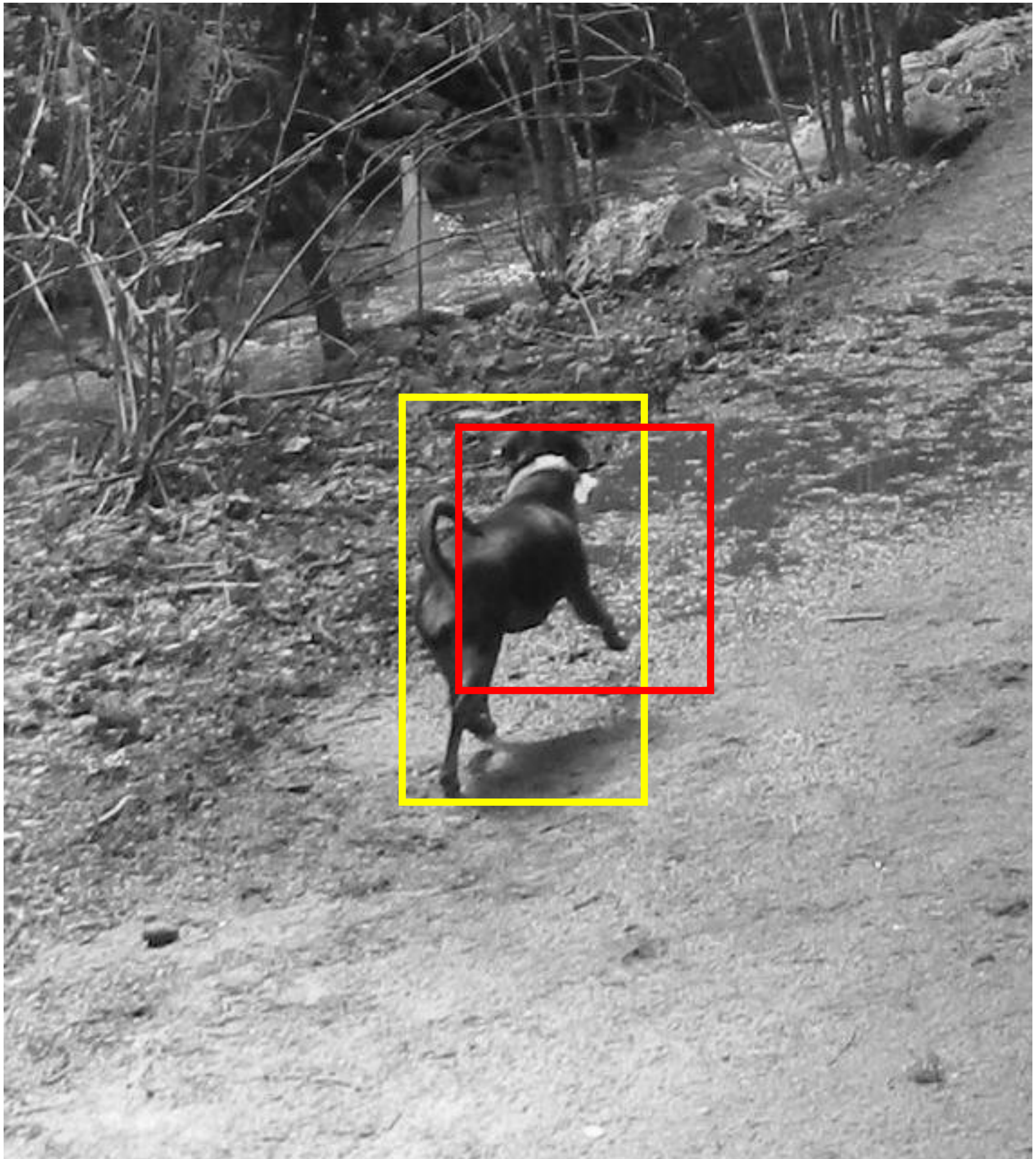


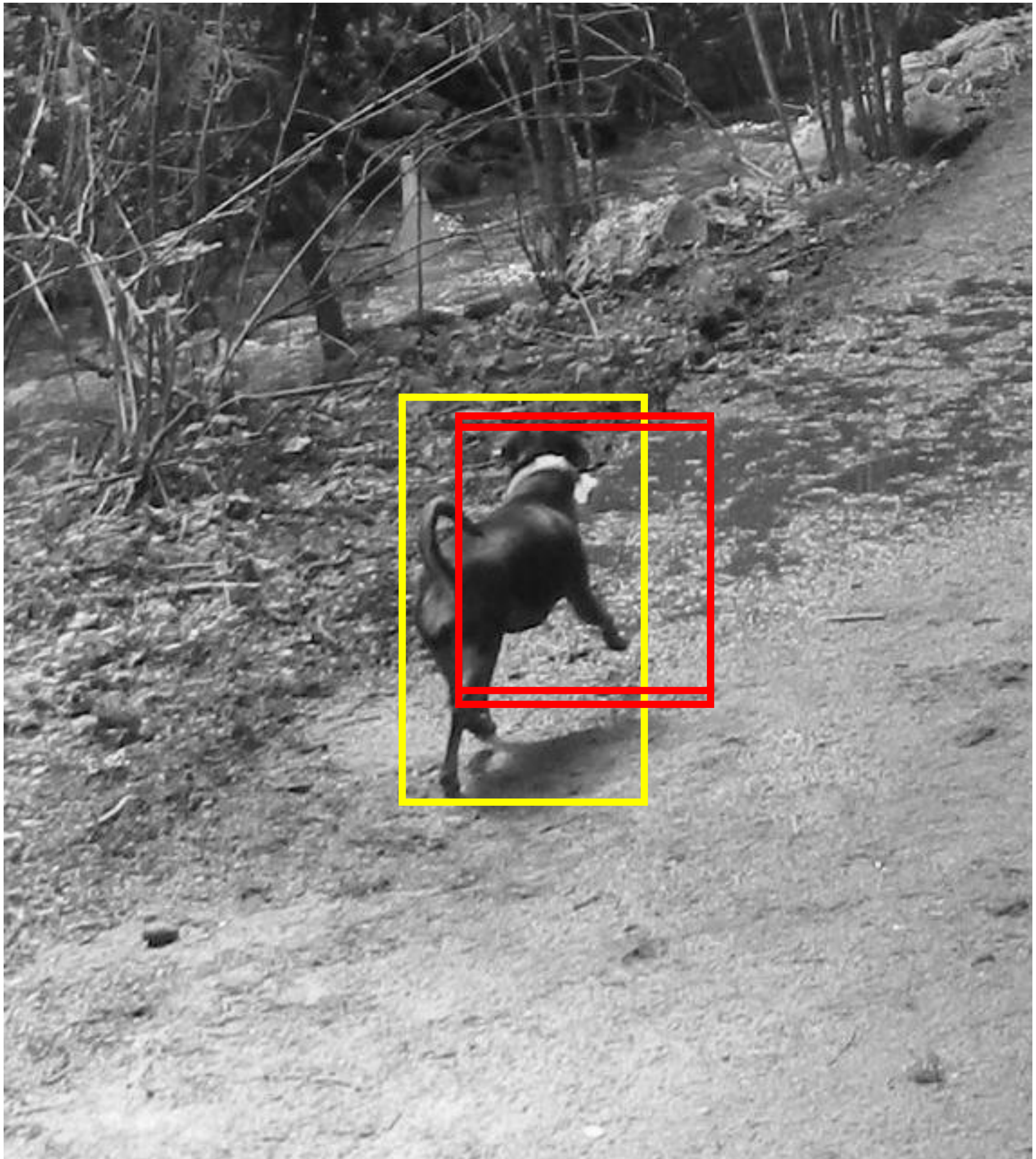
Down



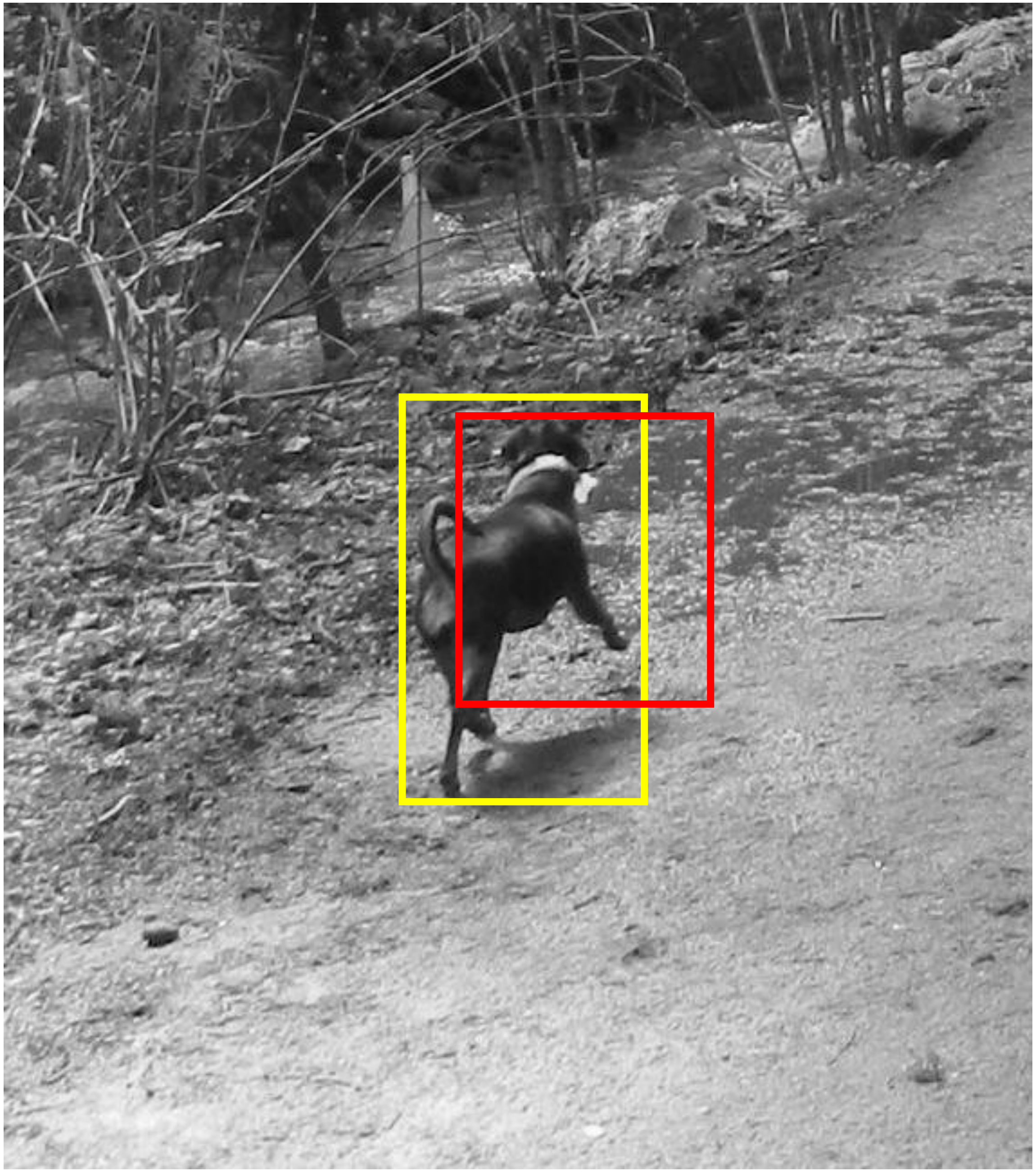


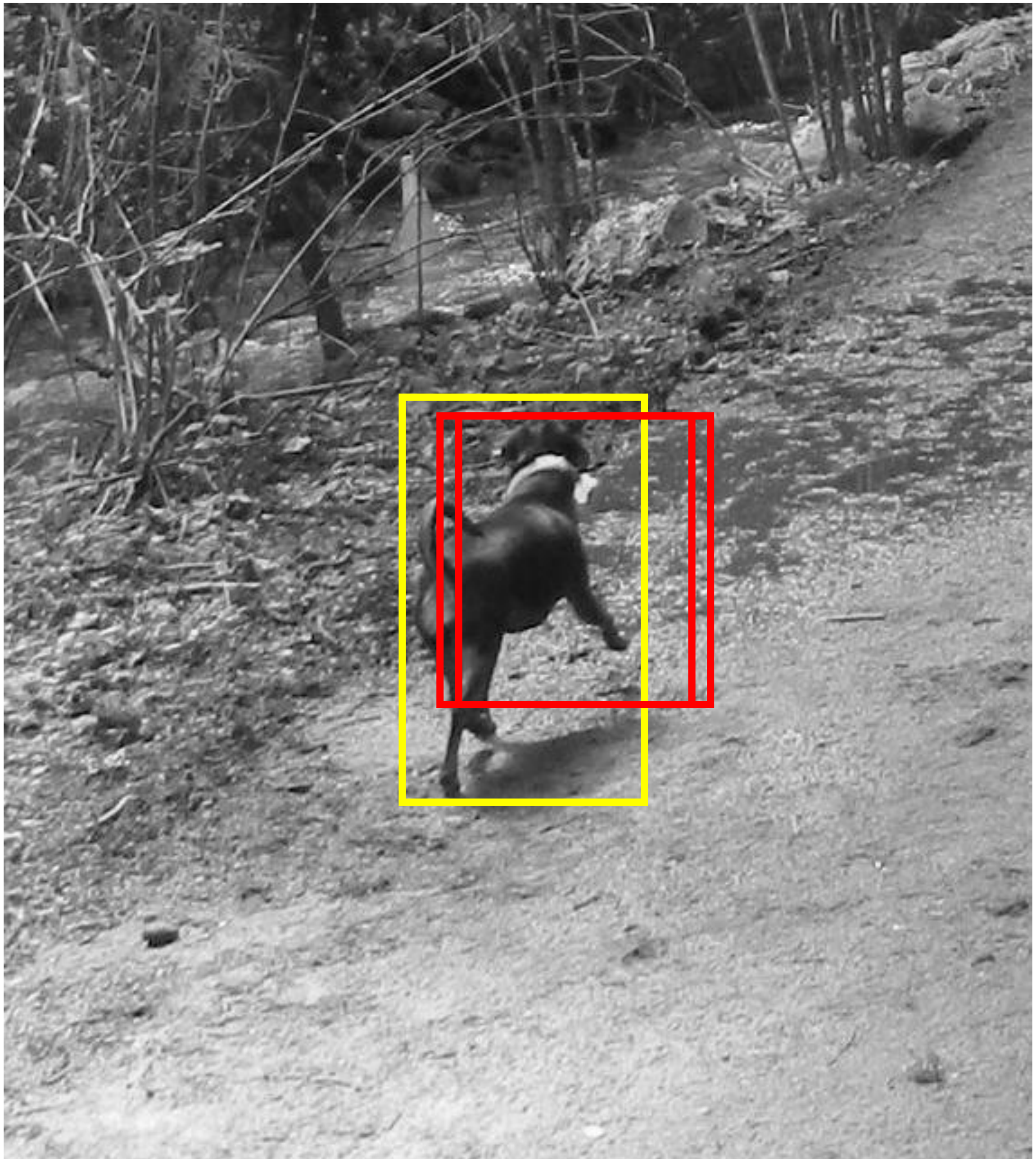
Taller





Taller





Left

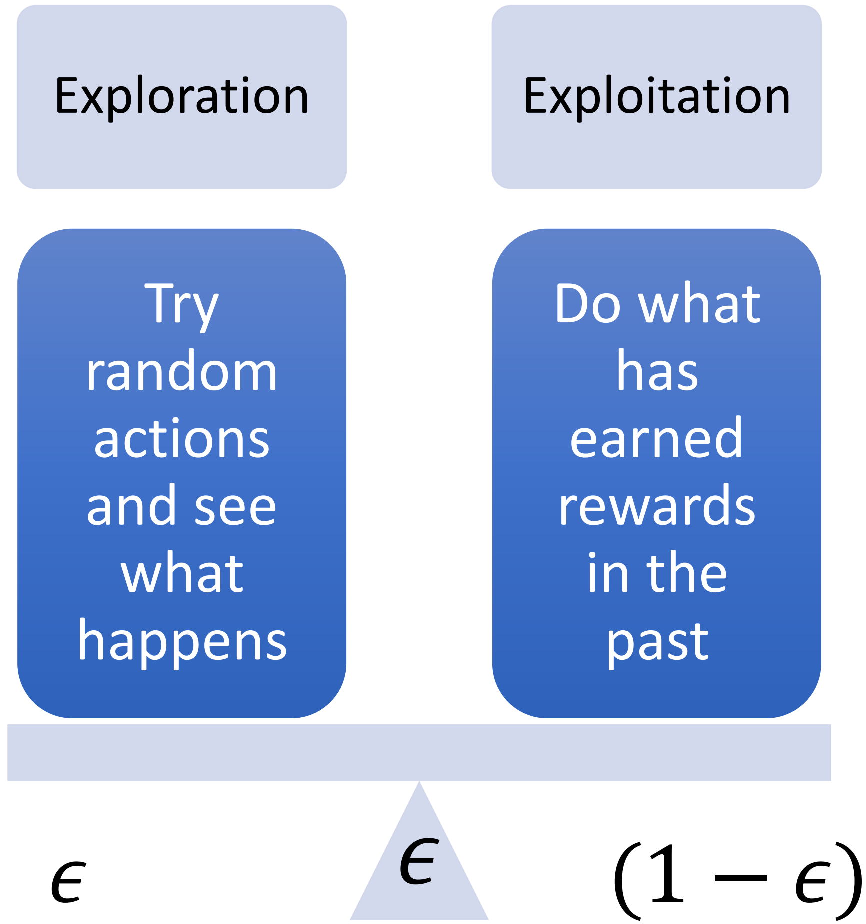
Down

Done

Reinforcement Learning

- Machine Learning method that works by trial and error (like the way we learn)
- Agent tries actions to complete a task
- Positive rewards for advantageous behavior
- Negative rewards for disadvantageous behavior
- Repeat

Epsilon-Greedy Algorithm



Epsilon-Greedy Variations

- Constant
- Annealing – epsilon policy where epsilon is gradually reduced over the course of training
  - Early in training exploration emphasized, exploitation later in training.
- Adaptive/Contextual – epsilon changes tied to learning progress or context.

Thesis Hypothesis:

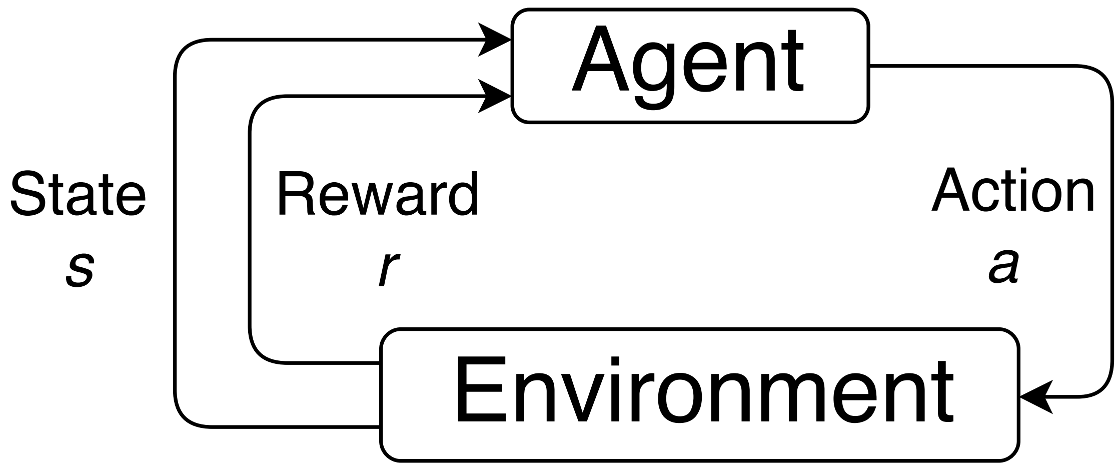
- I hypothesize that the Epsilon-greedy policy used during training matters for the performance of the search algorithm.

- I perform experiments to compare performance between 4 different epsilon policies.
  - 3 constant value: 0.75, 0.5, 0.25
  - 1 linear annealing policy. ~0.9 in beginning to ~0.1 at end of training

- I also explore the effect of the length of training (number of epochs)

2. Background

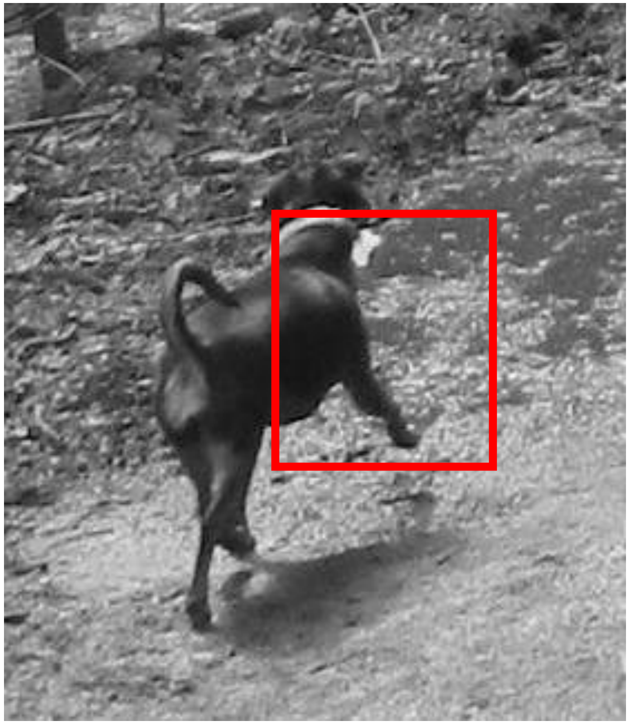
Reinforcement Learning (again)



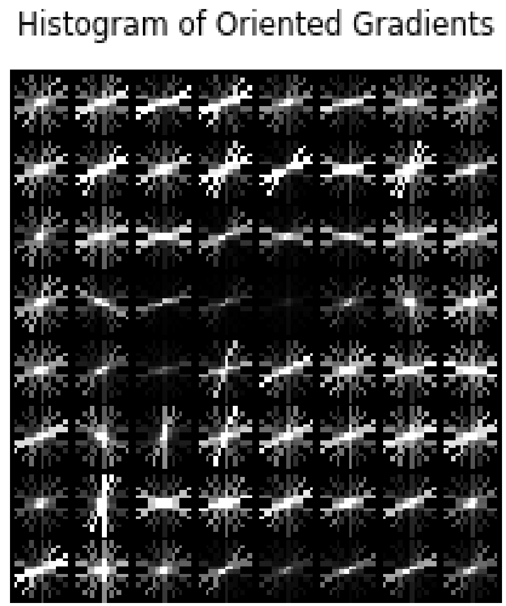
- Cycle repeats until terminal state is reached.
- One sequence of states from an initial state to the final state is referred to as an *episode*
- Agent's Goal: learn policy $\pi(s)$ to maximize cumulative discounted rewards over course of episode.

States in my algorithm

- Image, bounding box
- Features extracted from box to inform the algorithm.
- Action history
  - Last 10 actions taken
  - [left, left, up, fatter, smaller,…]

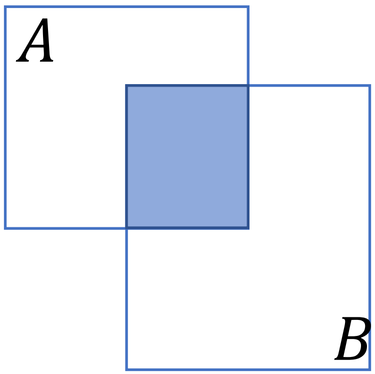


State Features - HOG



- Histogram of Oriented Gradients (HOG) features.
- Slopes of edges in images are computed
- Organized into histograms binned by slope orientation.
- Compiled (in my case) into a 2916-length vector

Reward Function: Intersection over Union (IOU



$$IOU(A,B) = \frac{A \cap B}{A \cup B}$$

- IOU = 0 => no overlap.
- IOU =1 => (A = B)
- IOU of bounding box to the ground truth used as goodness of fit measure.
- $r = \begin{cases} +1, & \Delta IOU > 0 \\ -1, & \Delta IOU < 0 \\ 0, & \Delta IOU = 0 \end{cases}$

Q-Learning

- In Q-Learning, the agent learns action-value function $Q(s, a)$, which is an estimate of 'value' of taking action a in state s.

- $Q(s, a) \leftarrow Q(s, a) + \underbrace{\eta}_{\text{Learning rate:}}[\underbrace{r + \gamma \max_{a'} Q(s', a')}_{\text{target}} - Q(s, a)]$

- Bracketed portion = difference between old estimate $Q(s, a)$ and the new 'target' estimate $r + \gamma \max_{a'} Q(s', a')$

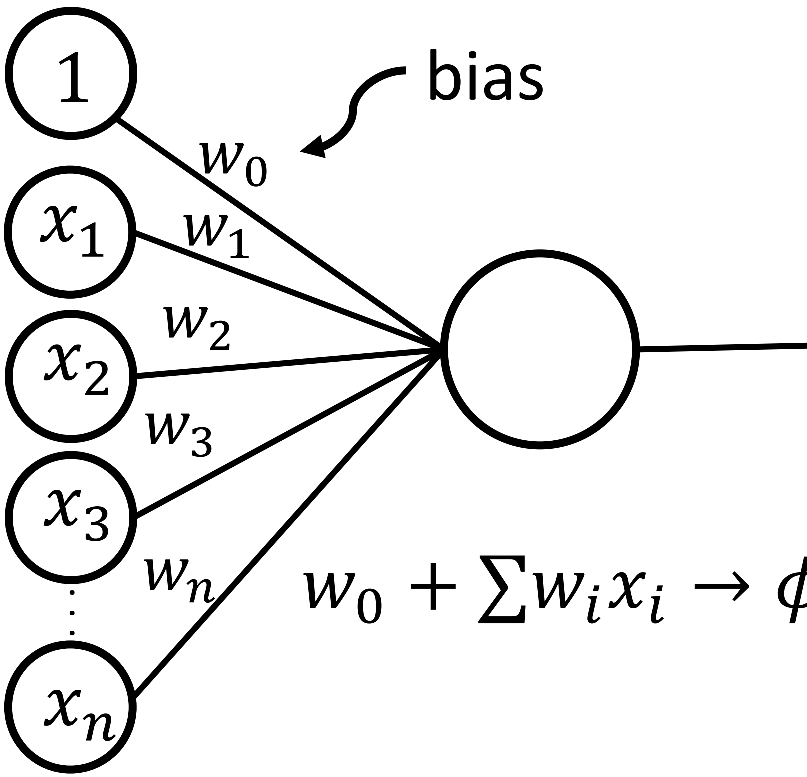
- Learning rate η is the rate at which the model updates to new information.

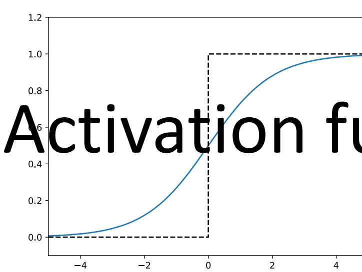
Q-Learning with Perceptrons

- Sometimes state space is prohibitively large for agent to explore all possible states.
- In these cases, instead of learning what to do in a **specific** state s, we want to learn a policy for what to do in states **similar to** s.
- To accomplish this, I approximate the Q-function using an ensemble of perceptrons.
- Q-values for each action determined by a linear function of state features.

Perceptron

- A *perceptron* is an artificial neuron that takes an input vector $x = (x_1, x_2, \ldots, x_n)$ and returns an *activation* based on a linear application of weights.





Activation function

$$w_0 + \sum w_i x_i \rightarrow \phi(w_0 + w \cdot x) \rightarrow y \text{ output}$$

Activation Function

- Traditional step function: $\phi(z) = \begin{cases} 0, & z < 0 \\ 1, & z \geq 0 \end{cases}$
  - Useful for binary classifications
  - Sometimes, the discontinuity at 0 is not desirable because a small change in weights causes a reversal in classification.
- Sigmoid function: $\sigma(z) = \frac{1}{1+e^{-z}}$
  - Continuous approximation of step function
  - My model uses sigmoid activation

Update Rule for Q-Learning with Perceptrons

- Perceptron weights updated according to

$$w_i \leftarrow w_i + \eta(\underbrace{t}_{\text{target}} - \underbrace{y}_{\text{perceptron output}})x_i$$

- Q-values updated according to

$$Q(s,a) \leftarrow Q(s,a) + \eta[\underbrace{r + \gamma \max_{a'} Q(s',a')}_{\text{target}} - \underbrace{Q(s,a)}_{\text{old Q-value}}]$$

- Agent takes action a_k in state s, weight W_{ki} is updated according to

$$W_{ki} \leftarrow W_{ki} + \eta(\underbrace{\sigma(r + \gamma \max_{a'} Q(s',a'))}_{\text{target}} - \underbrace{Q(s,a_k)}_{\text{perceptron output}})s_i$$

Back to Q-Learning

- States represented as an input vector $s = (s_1, \ldots s_n)$ to a perceptron. → $n + 1$ weights (including bias)
- Let there be m actions, with one perceptron per action.
- Weights organized into a $m \times (n + 1)$ matrix W.
- Q-values computed as below

$$\begin{pmatrix} w_{10} & w_{11} & w_{12} & \cdots & w_{1n} \\ w_{20} & w_{11} & w_{12} & \cdots & w_{1n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ w_{m0} & w_{m1} & w_{m2} & \cdots & w_{mn} \end{pmatrix} \begin{pmatrix} 1 \\ s_1 \\ \vdots \\ s_n \end{pmatrix} = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_m \end{pmatrix} \rightarrow \sigma(\cdot) = \begin{pmatrix} Q(s, a_1) \\ Q(s, a_2) \\ \vdots \\ Q(s, a_m) \end{pmatrix}$$

3. Methods

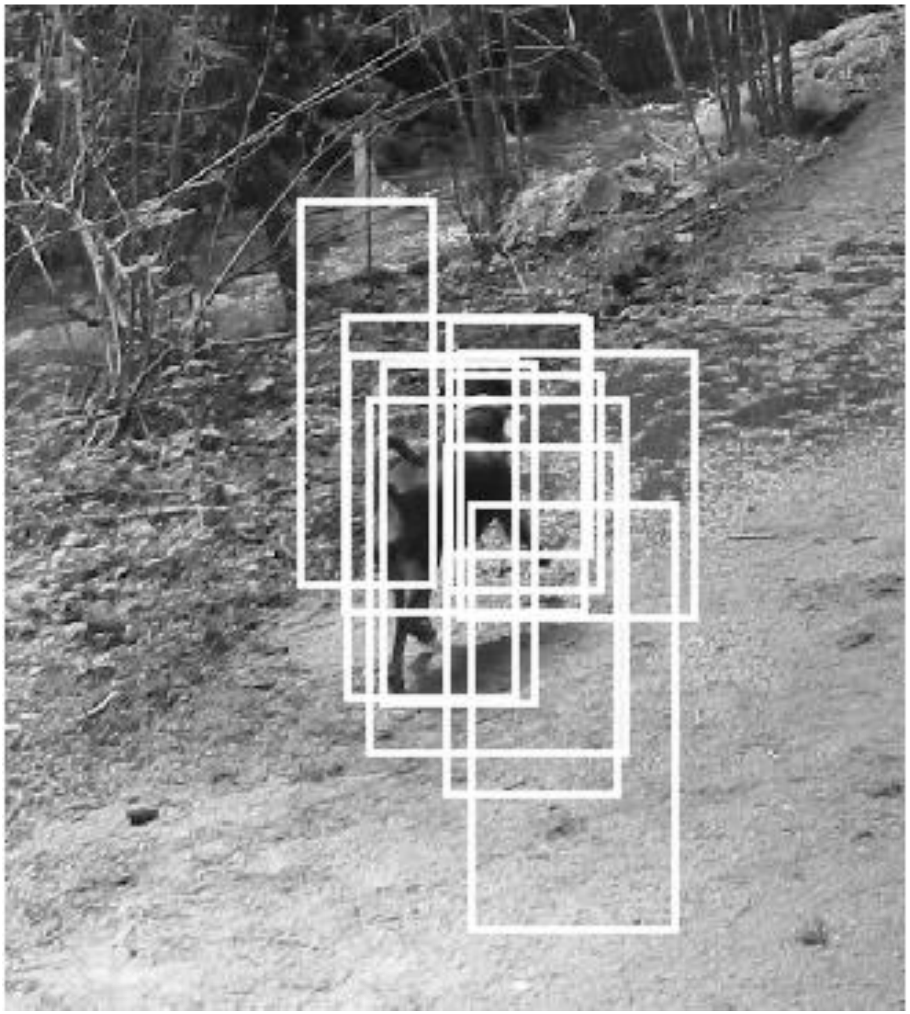
Dataset:

- Portland State Dog Walking Images
- Contains human-drawn ground truth labels for dogs, and humans.
- For each object category (dogs, humans), I split images into training set of size 400, and a test set of size 100

Bounding Box representation

- Box = (x, y, w, h)
- $(x, y) =$ bounding box's **center** location
- $(w, h) =$ box's width, height

Generating Initial Bounding Boxes (skews)



- 10 skews created per object.
- Bounding box components (x, y, w, h) shifted from ground truth according to random normal distribution.
- Standard deviation proportionate to width or height of ground truth box.

Parameters

- Learning rate $\eta = 0.2$
- Discount Factor $\gamma = 0.9$
- Actions Per Episode $= 15$
- Number of Epochs = 200 (and lower)
- Epsilon (varied)

Experiment Design

- Constant epsilon values: 0.75, 0.5, 0.25

(mostly exploration), (evenly balanced), (mostly exploitation)

- Annealing: epsilon $\epsilon = \ 0.904 \ - \ 0.004x$

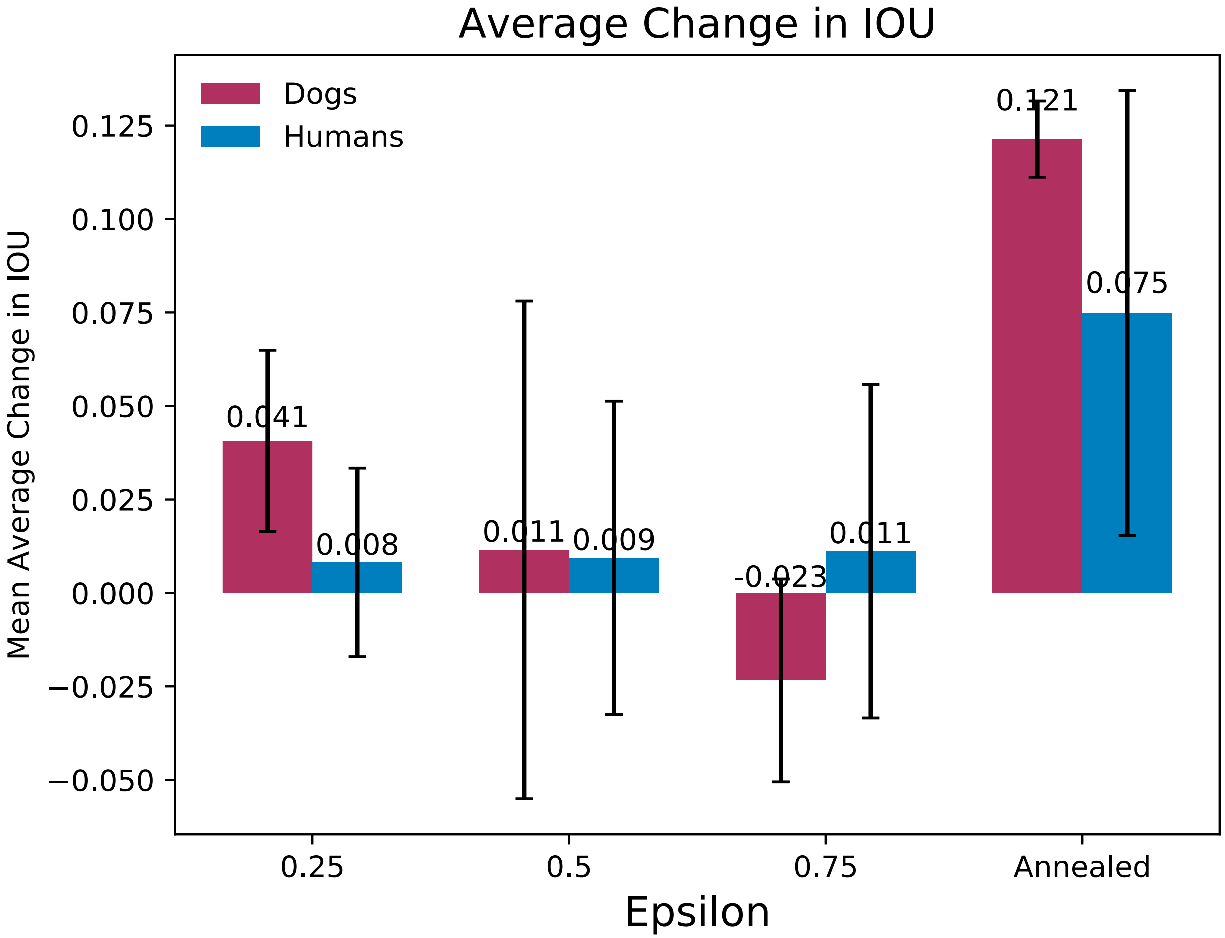
- 5 runs for each epsilon-greedy policy.
- Done for both ‘dog’ and ‘human’ categories.

Testing

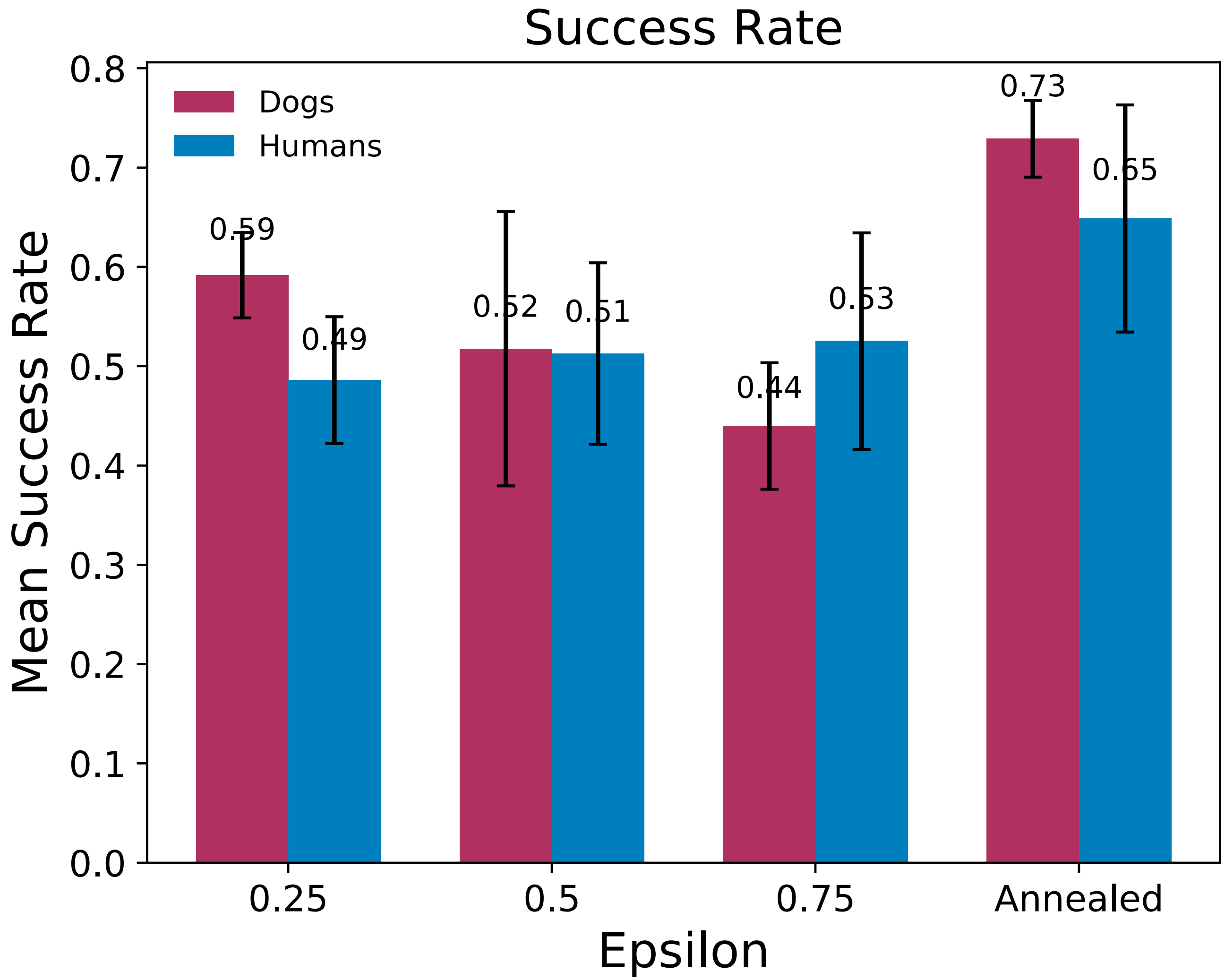
- 100 images x 10 skews/image = 1000 examples
- Algorithm mostly same as training: 15 actions per episode
- Actions chosen solely on Q-value (epsilon = 0)
- Weights are not updated (no need to compute rewards)
- Performance measures:
  - Average Change in IOU
  - Success Rate = Fraction of bounding boxes improved.

4. Results

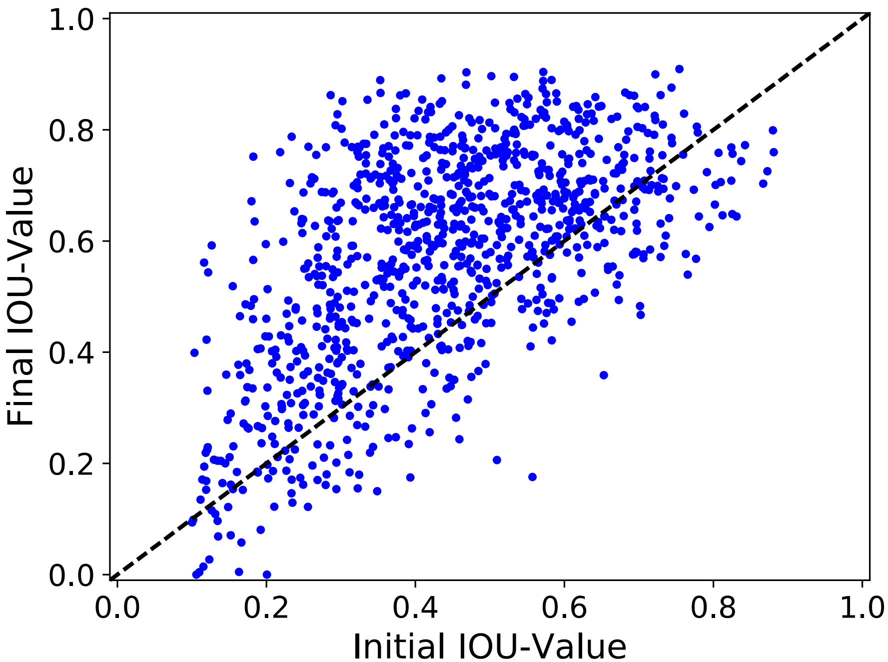
Effect of Epsilon 1 – Average change in IOU

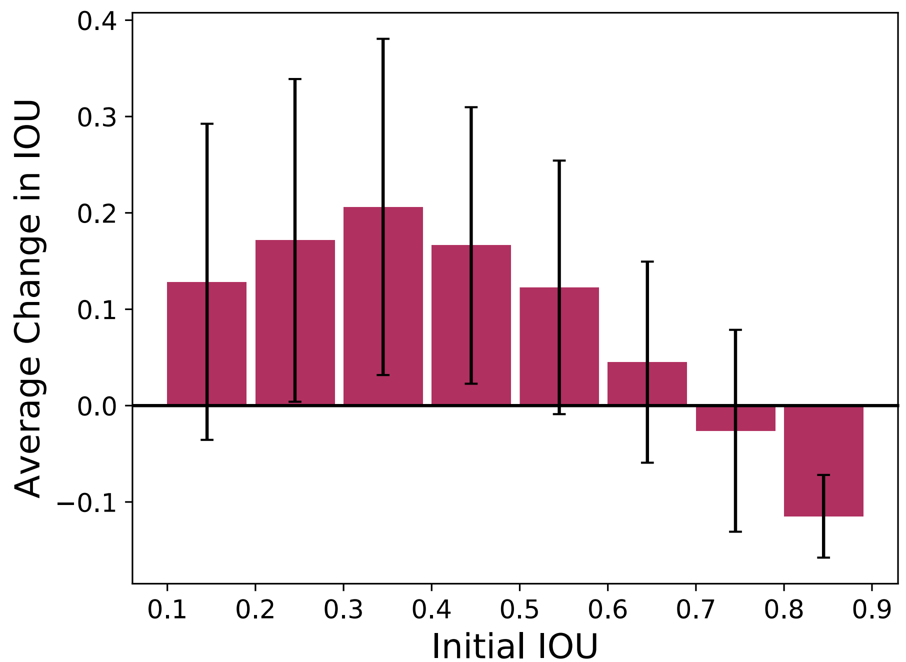


Effect of Epsilon 2 – Success Rate (percent improved)



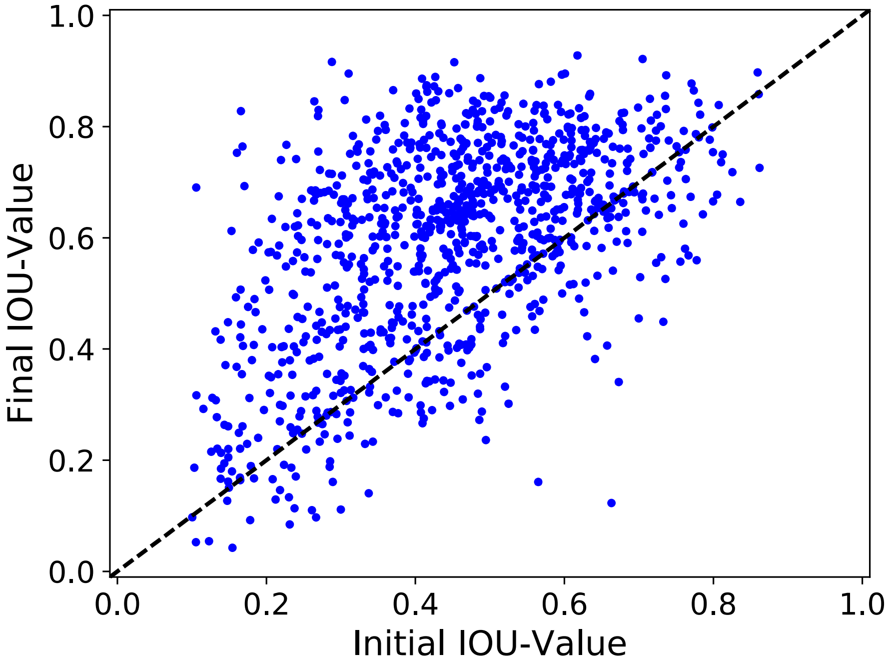
Effect of Initial Bounding Box IOU-Value Dogs

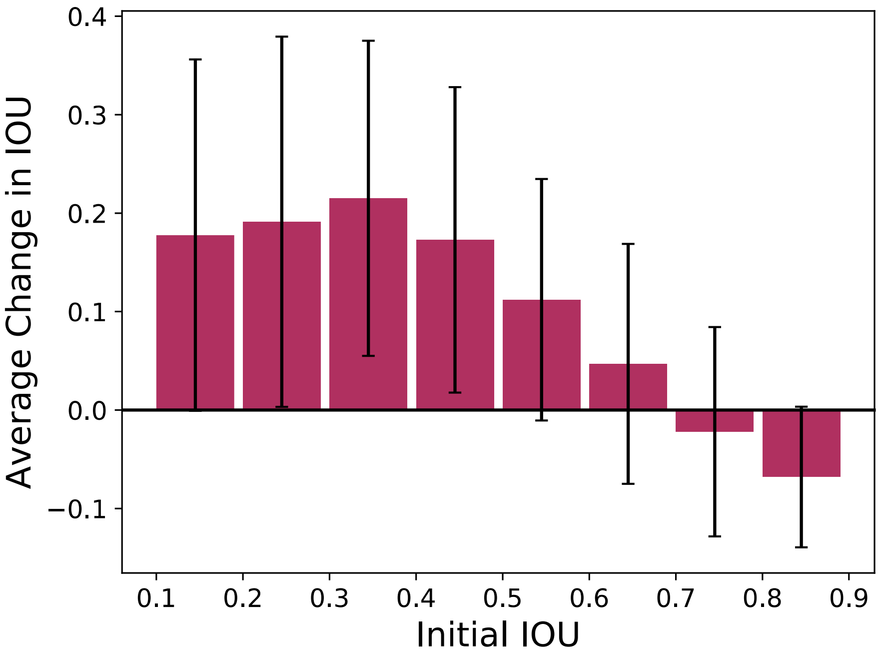




Highest performing ‘dog’ category run with annealing:
Mean IOU Improvement = 0.135,
Success Rate = 78.9%

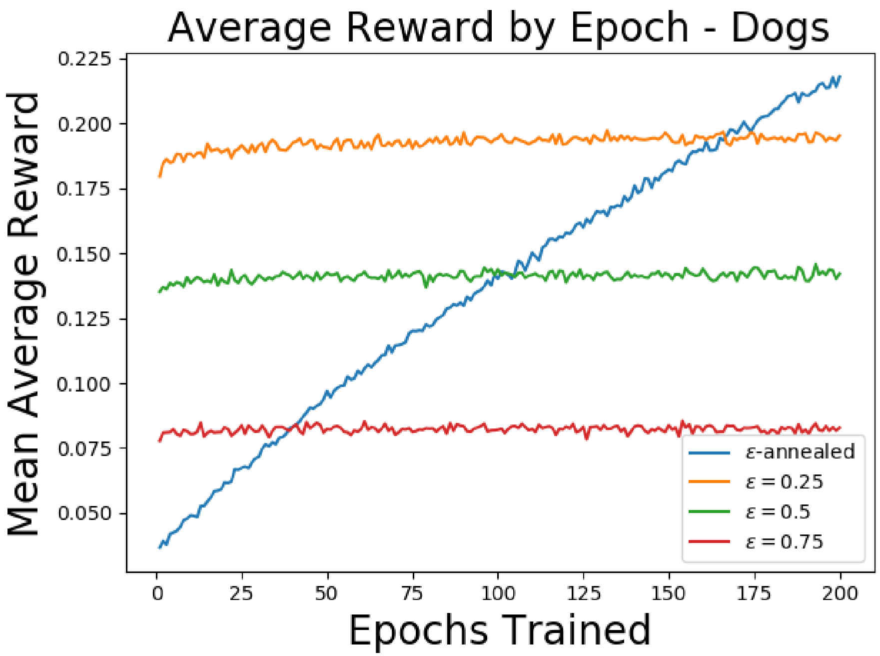
Effect of Initial Bounding Box IOU-Value – Humans

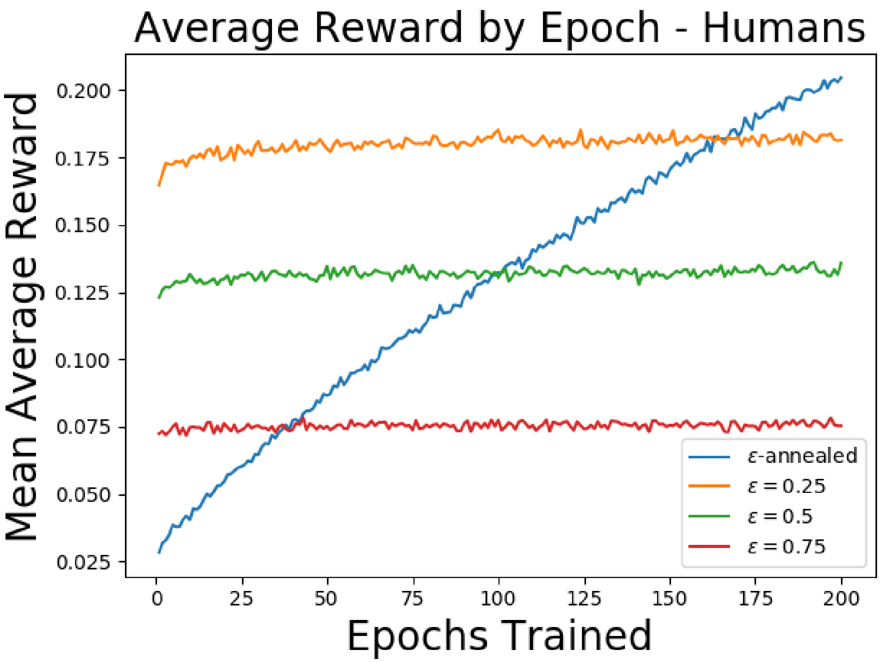




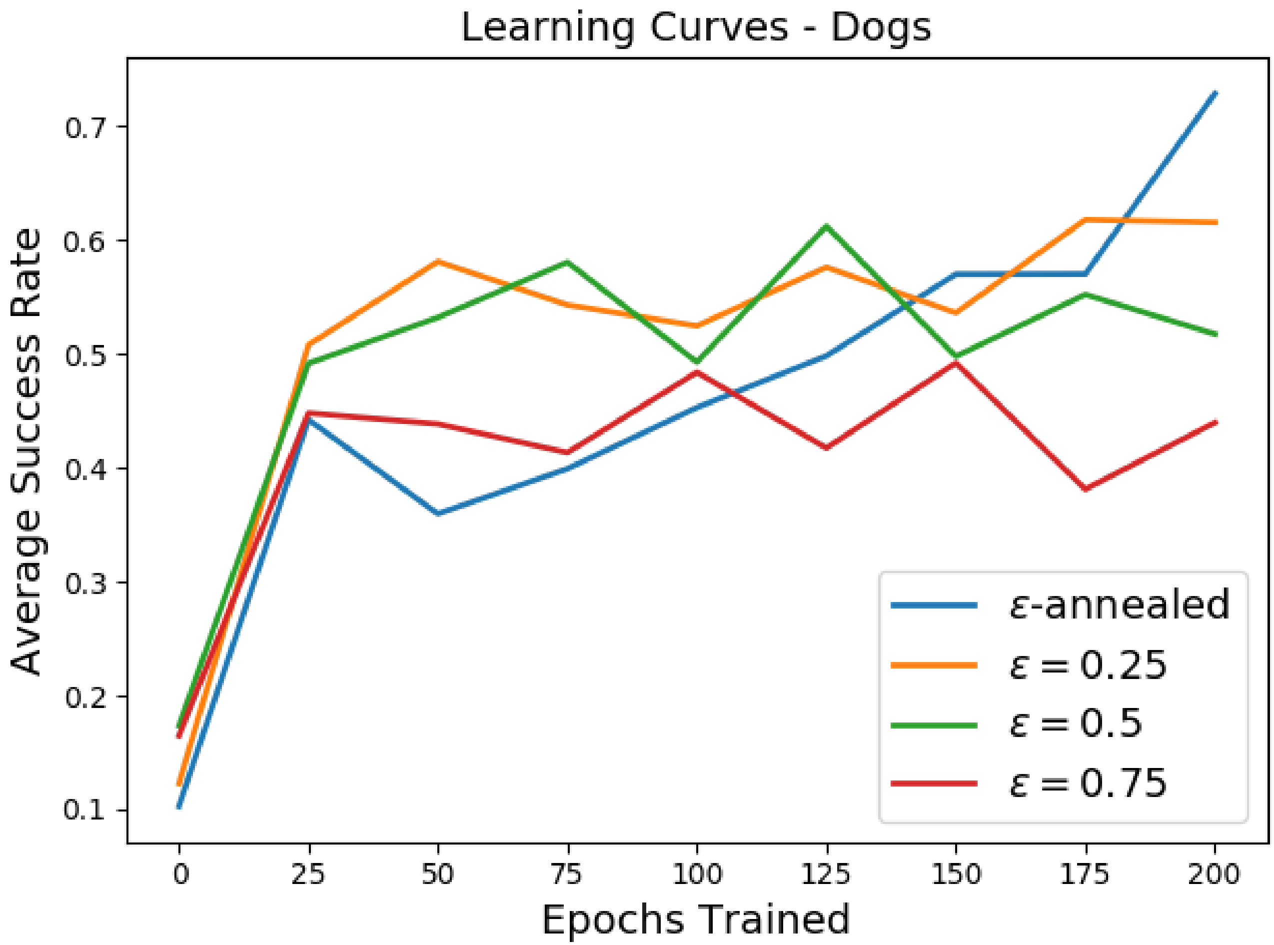
Highest performing 'human' category run with annealing:
Mean IOU Improvement = 0.143,
Success Rate = 78.4%

Average Reward Per Episode

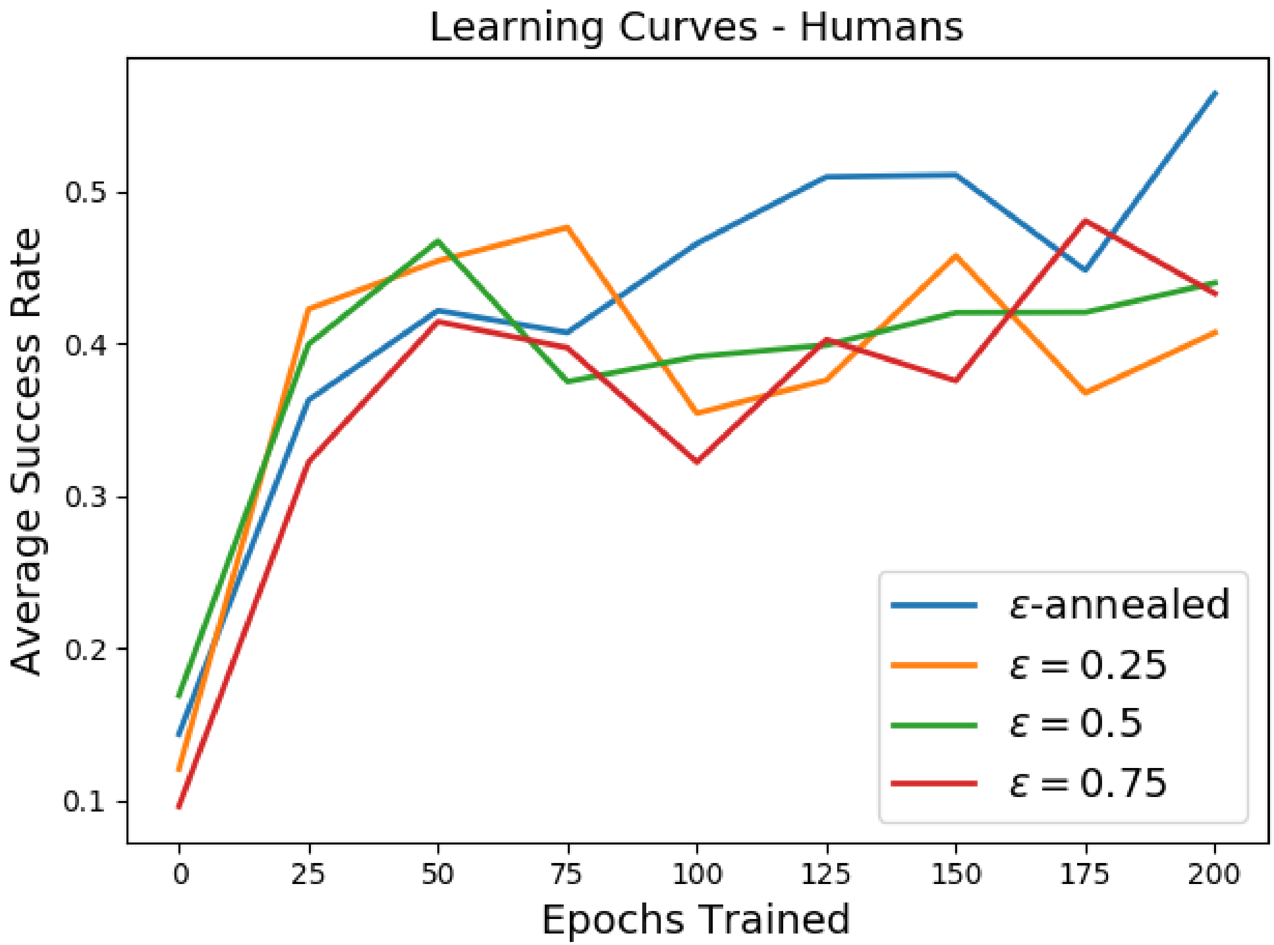




Effect of Number of Epochs
Success Rate by Epoch - Dogs



Effect of Number of Epochs
Success Rate by Epoch - Humans



5. Conclusion and Future Work

Conclusions and Future Work

- Annealing appears to work best
- Annealing runs may be under trained.
- Performance higher for dogs than humans- why?
- Other future work:
  - Use CNN features
  - More sophisticated stopping mechanism – stop action triggers end of episode.
- Rigorous comparison with bounding box regression.

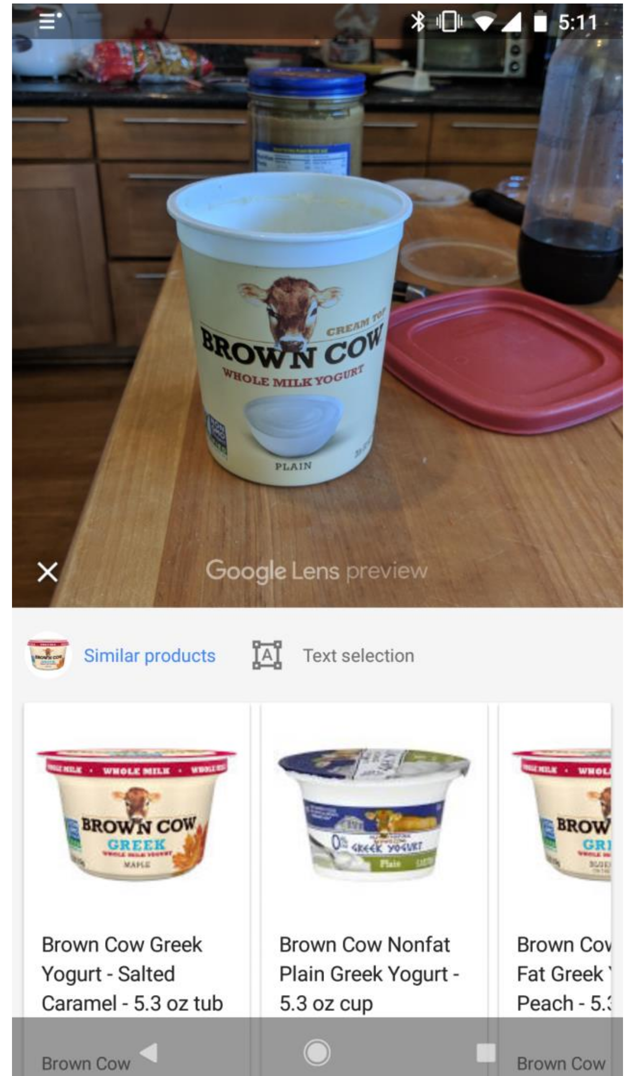
End

Additional Slides

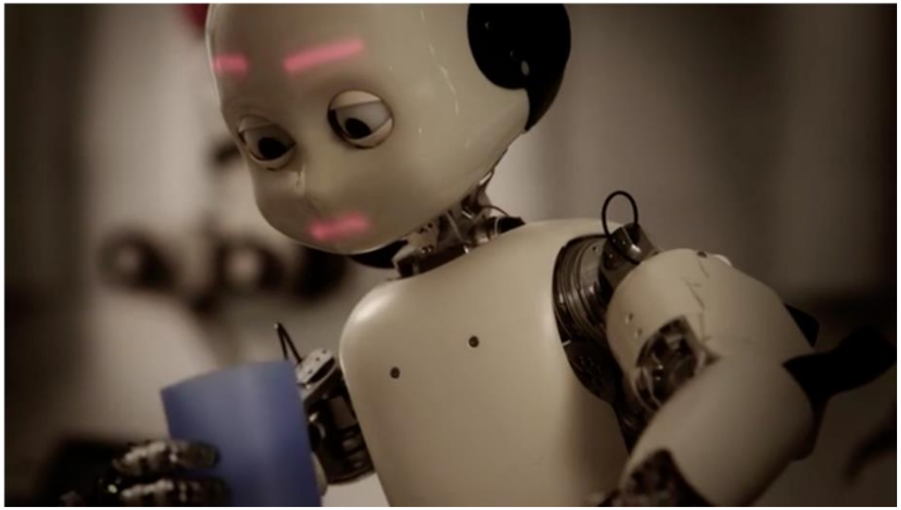
Training Algorithm

- For *epoch*=1 to 200:
  - Shuffle the training set;
  - For each (*img*, *skew*) in training set:
    - *current_box* ← *skew*;
    - Initialize state s ← HOG features from *skew*, 0 history vector;
    - For *step* = 1 to 15:
      - Select action a according to epsilon-greedy.;
      - Take action a to obtain *new_box*;
      - Add a to history vector;
      - Extract HOG features from *new_box* and combine with history vector to obtain state s';
      - Compute change in IOU-value to obtain reward r;
      - Compute $\max_{a'} Q(s', a')$;
      - Update perceptron weights according to
        - $w_i \leftarrow w_i + \eta(\sigma(r + \gamma \max_{a'} Q(s', a')) \; - Q(s, a))s_i$
      - *current_box* ← *new_box*;
      - $s \leftarrow s'$;

Cell Phone Apps (Google Lens)

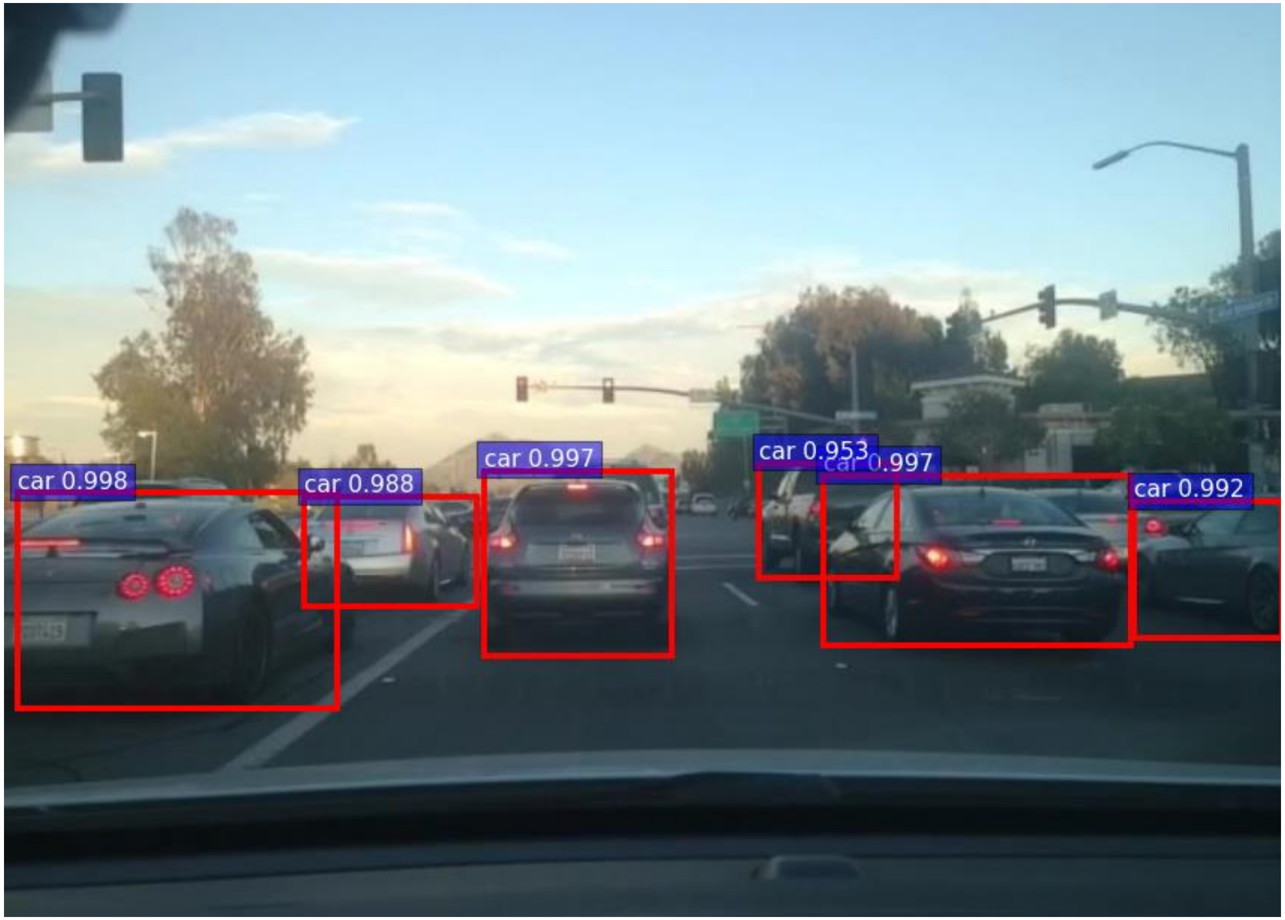


Robot Control

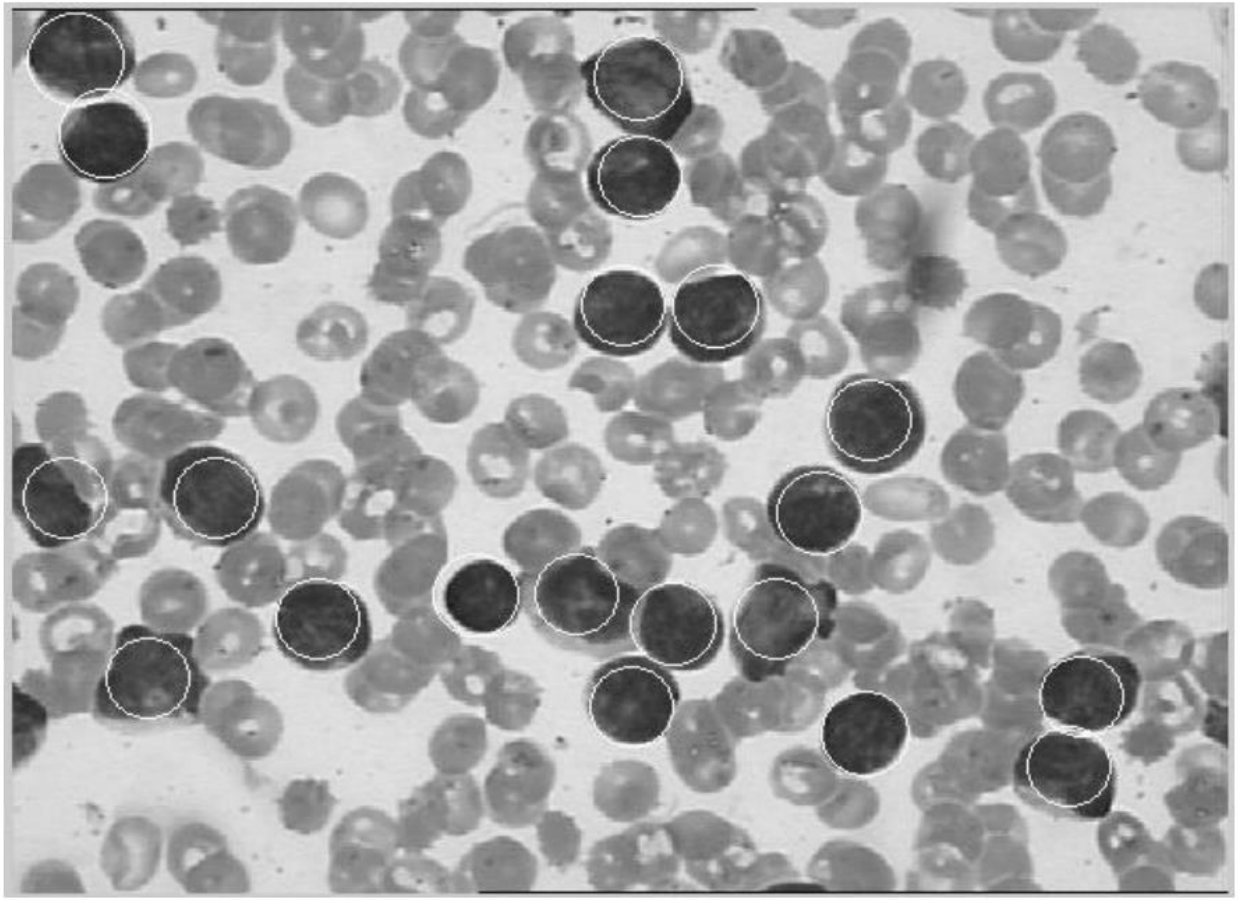
- [Source]

Self-Driving Cars



[Source](#)

Medical Imaging



Blood cell classification from:

[G. Karkavitsas, M. Rangoussi Object localization in medical images using genetic algorithms]

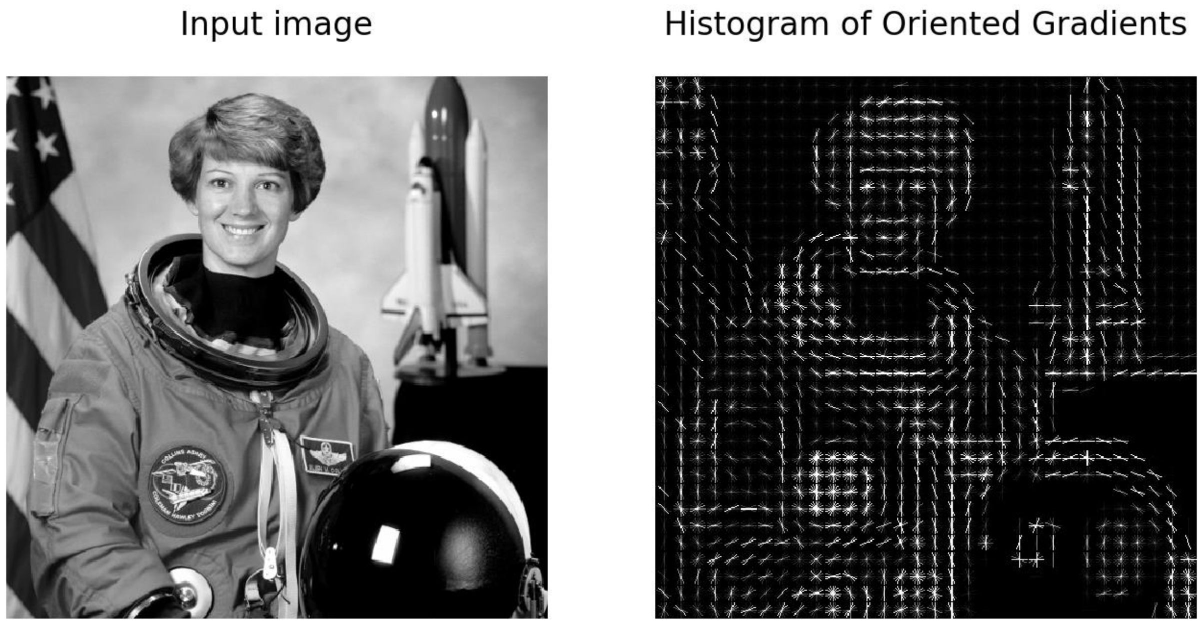
Markov Decision Processes

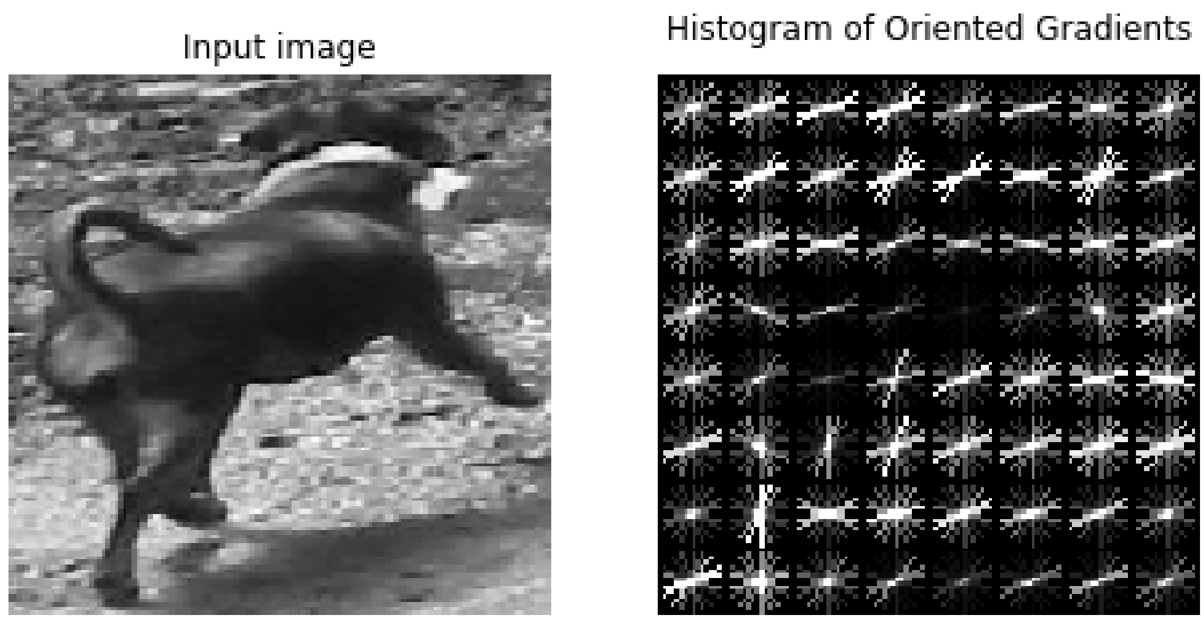
- RL Models are typically represented as *Markov Decision Processes*
- *Markov Decision Processes* (MDP) have the following components:
  - $S =$ Set of states, including initial state s_0 and terminal state s_T
  - $A =$ Set of actions agent may take
  - *Transition rules* that determine the next state given the previous state and the action taken by the agent. The transition rules may be probabalistic: $P(s_{t+1} = s' | s_t = s, a_t = a)$
  - Reward function $r(s, a)$
  - γ =discount factor. Weighs value of future rewards against present reward.

Histogram of Oriented Gradients

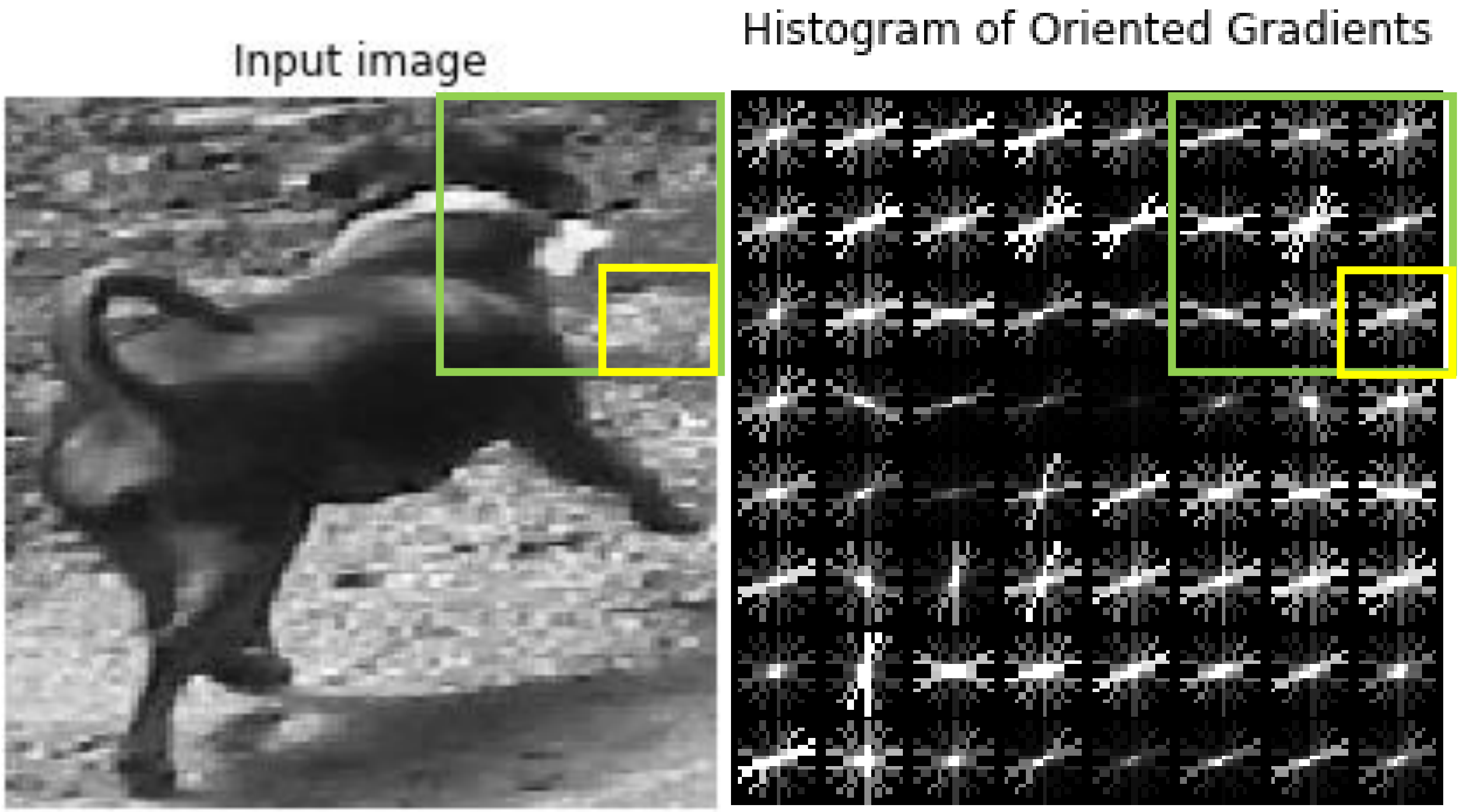
- Image region divided into cells,
- Within each cell, gradients are computed, (change in intensity with respect to x and y)
- Gradients compiled into histograms organized by cell.
  - Bins separated by orientation
  - 0-180° for unsigned gradients (which I use)
  - 0-360 ° signed gradients
  - Gradients normalized by “block”, which is a larger region encompassing each cell.
- scikit-image Library used to compute HOG [2].

HOG Examples





HOG, continued



Block: 3x3 cells

Cell:
16x16 pixels

- 180° divided into 9 histogram bins

HOG array shape: (6, 6, 3, 3, 9) → HOG Feature vector with 6 x 6 x 3 x 3 x 9 = 2916 features

blocks cells bins

State Definition

- State vectors are a concatenation of HOG features drawn from the bounding box, and history features.
- HOG array shape: (6, 6, 3, 3, 9) → HOG Feature vector with 6 x 6 x 3 x 3 x 9 = 2916 features
- Action history vector:
  - Each action encoded as a length-9 bit vector.
  - Last ten actions are recorded
  - So history vector has $9 \times 10 = 90$ features
- Combined state vector: $2916 + 90 = 3006$ features

Action definitions

Given a bounding box $b = (x, y, w, h)$:

- *left/right:* $x \leftarrow x \pm \alpha x$
- *up/down*: $y \leftarrow y \pm \alpha h$
- *bigger/smaller:* $w \leftarrow w \pm \alpha w,\ h \leftarrow h \pm \alpha h$
- *fatter:* $w \leftarrow w + \alpha w$
- *taller:* $h \leftarrow h + \alpha h$
- *stop:* no change in b

- Shift factor $\alpha = 0.1$

Training Algorithm

- Repeat for N epochs:
  - For each (*img*, *skew*) in training set:
    - *current_box* ← *skew*;
    - Initialize state s ← HOG features from *skew*, 0 history vector;
    - For *action* = 1 to 15:
      - Agent adjusts the box according to to epsilon-greedy. State s' obtained;
      - Compute change in IOU-value to obtain reward r;
      - Update perceptron weights;
        - $w_i \leftarrow w_i + \eta(\sigma(r + \gamma \max_{a'} Q(s', a')) - Q(s, a))s_i$
      - *current_box* ← *new_box*;
      - $s \leftarrow s'$;