CONTINUOUS TRANSLATION ACTION AGENT FOR BOUNDING BOX REFINEMENT

A progress report for mid-term dissertation evaluation

Of

M. Tech (Signal Processing) 2020-2022

By

Anindya Ghosal (2020PSP3007)

Under the supervision

of

Prof. Jyotsna Singh

Division of Electronics & Communication Engineering

NETAJI SUBHAS UNIVERSITY OFTECHNOLOGY

(Formerly NSIT)

NEW DELHI-110078

CERTIFICATION

This is to certify that this report titled **"Continuous Translation Action Agent for bounding box refinement"** being submitted by ANINDYA GHOSAL (2020PSP3007) to the Division of Electronics and Telecommunication Engineering, Netaji Subhas University of Technology (NSUT) for the mid-semester dissertation evaluation for the award of M. Tech. degree, is record of bona-fide work carried out by him under the supervision and guidance. The matter embodied in this report has not been submitted for the award of any other degree.

Supervisor
Prof. Jyotnsa Singh
Division of ECE,
NSUT, New Delhi, India

Introduction

With the amount of computing power and the huge amount of data available for the researchers and companies to draw out insights and providing us with information about how, what, when and where has totally lifted the use of AI and Machine Learning to new heights.

Machine Learning on a very coarse level has been divided into 3 major areas, namely Supervised Learning [1], Unsupervised Learning [2], and Reinforcement Learning [3].

- Supervised Learning [1], it mainly deals with labeled data. Expanding a little, labeled data means when we train the data we already have told our machine learning algorithm about the given input and our expected output. A few examples of supervised machine learning consists of regression, logistic regression, naïve bayes, decision tree, K-Nearest Neighbor, Support-Vector Machine, Neural Networks. So the main requirement for this type of machine learning is labeled data.
- Unsupervised Learning [2], uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms discover hidden patterns or data groupings without the need for human intervention. Its ability to discover similarities and differences in information make it the ideal solution for exploratory data analysis, bio-informatics, genetic clustering, sequence analysis, customer segmentation, and image recognition. Few of the unsupervised learning approach consists of K-Means Clustering, Association Rules, Hierarchical Clustering, Neural Networks.
- Reinforcement Learning [3] finds its root in Psychology. As in day to day life we come across many situations which forces us to take a some actions. After taking such actions we get either positive reward or negative reward. This is the same idea and thought process behind reinforcement learning. You train a agent such that based on certain state it's present in, it takes few actions and gets rewards. Now based on what we want to do with those rewards is the main idea that is being explored nowadays. Reinforcement learning has started to gain recognition mainly in the field of robotics, supply chain management, self-driving cars, computer vision, finance and trading and much more.

So starting with reinforcement learning, the first ever idea that started was mainly embedded in psychological journals. Taking notes and inspiration from how humans react to different methods of learning. For example, if a child is running a race and he/she wins the race, the child is rewarded with a chocolate or a medal or any sort of appreciation. This is known as "Positive Reinforcement". If a child is playing in a playground and he accidently touches thorny bush, he immediately starts crying because the thorn is causing a pain. This is an example of "Negative Reinforcement".

The history of reinforcement learning has two main ideas. One idea concerns learning by trial and error and started in the psychology of animal learning. This idea runs through some of the earliest work in artificial intelligence and led to the revival of reinforcement learning in the early 1980s.

The other idea concerns the problem of optimal control and its solution using value functions and dynamic programming. For the most part, this idea did not involve learning. Although the two ideas have been largely independent, the exceptions revolve around a third, less distinct idea concerning temporal-difference methods. All of the three ideas came together in the late 1980s to produce the modern field of reinforcement learning as we know today.

The term "optimal control" came into use in the late 1950s to describe the problem of designing a controller to minimize a measure of a dynamical system's behavior over time. One of the approaches to this problem was developed in the mid-1950s by Richard Bellman and others through extending a nineteenth century theory of Hamilton and Jacobi. This particular approach uses the concepts of a dynamical system's state and that of a value function, or "optimal return function," to define a functional equation, now known as the “Bellman equation”. This class of methods for solving optimal control problems by solving this equation came to be known as dynamic programming. Bellman also introduced the discrete stochastic version of the optimal control problem known as Markovian decision processes (MDPs), and Ron Howard was the one who devised the policy iteration method for MDPs. All of these are the most essential elements underlying the theory and algorithms of modern reinforcement learning.

Dynamic programming is still widely considered the only feasible way of solving general stochastic optimal control problems. It suffers from what Bellman has called "the curse of dimensionality," meaning that its computational requirements of the algorithm grows exponentially with the number of state variables, but it is still far more efficient and more widely applicable than any other general method. Dynamic programming has been extensively developed since the late 1950s, including extensions to partially observable MDPs.

Returning now to the other major idea leading to the modern field of reinforcement learning that centered on the idea of trial-and-error learning. This idea began in psychology, where "reinforcement" theories of learning are common. Perhaps the first to succinctly express the essence of trial-and-error learning was Edward Thorndike, who in a nutshell said that action followed by good (positive) or bad (negative) outcomes have their tendency to be reselected altered accordingly. Thorndike called this the "Law of Effect" because it described the effect of reinforcing events on the tendency to select actions.

We turn now to the third idea, to the history of reinforcement learning, that concerning temporal-difference learning. Temporal-difference learning methods are distinctive in being driven by the difference between temporally successive estimates of the same quantity--for example, of the probability of winning in the tic-tac-toe example. This approach is smaller and less distinct than the other two, but it has played a particularly important role in the field, in part because temporal-difference methods seem to be new and unique to reinforcement learning.

The origins of temporal-difference learning are in part in animal learning psychology, in particular, in the notion of “*secondary reinforcers”*. A *secondary reinforcer* is a stimulus that has been paired

with a *primary reinforcer* such as food or pain and, as a result, has come to take on similar reinforcing properties. The temporal-difference and optimal control ideas were fully brought together in 1989 with Chris Watkins's development of Q-learning [8].

Literature Review

(I) Components of Reinforcement Learning [6]

After the brief introduction of reinforcement learning, there are a few key concepts that form an integral part of reinforcement learning ecosystem. For that here are the main or key components of a reinforcement learning algorithm:

1) **Agent:** The main component of a reinforcement learning algorithm is agent. This is the entity that take actions in a particular environment. For example: An industrial robot picking up non-biodegradable items in a recycling plant.
2) **Environment:** The surrounding in which the above mentioned agent will work or perform its activities. For example for an agent in computer vision the part of image enclosed by the bounding box can be the environment.
3) **State:** This is the defined as the present status of the interacting agent in a particular environment. For example is the robot arm in motion or it is stationary or is it going up or is it going down.
4) **Action:** Action is the steps the agent takes. More often than not in practical applications collection of actions called as action space is discrete in nature. For example the robotic arm moving left, right, up, down, close, open etc.
5) **Reward:** This is defined as the objective that is implicitly or explicitly defined for the agent to take actions and maximize.

After discussing and going through a couple of main building blocks, there has to be components that bring these building blocks together because they alone can't act or execute independently. So connection all the above components here are the binding concepts of Reinforcement Learning. They are:

1) **Policy:** A *policy* defines the learning agent's way of behaving at a given time. In other words, a policy is a mapping from perceived states of the environment to actions to be taken when present in those states. It corresponds to what in psychology known a set of stimulus-response rules or associations. In some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computations like a search process. The policy is the core of a reinforcement learning agent in the sense that it alone is sufficient to determine behavior. Speaking generally, policies may be stochastic.
2) **Reward Function:** A *reward function* defines the goal in a reinforcement learning problem. In other words, it maps each perceived state (or state-action pair) of the environment to a single number, a *reward*, indicating the intrinsic desirability of that state. A reinforcement learning agent's sole objective is to maximize the total reward it receives in the long run instead of short term. The reward function defines what are good and bad events for the agent. In a

biological system, it would not be inappropriate to identify rewards with pleasure and pain. They are the immediate and defining features of the problem faced by the agent. The reward function must necessarily be unalterable by the agent. It may, however, be a basis for altering the policy. For example, if an action selected by the policy is followed by low reward, then the policy may be changed to select some other action in that situation in the future. Generally, reward functions may be stochastic.

3) **Value function:** A *value function* specifies what is good in the long run. In other words, the *value* of a state is the total amount of reward an agent can be expected to accumulate over the future, starting from that state. Rewards determine the immediate, intrinsic desirability of environmental states, Values however indicate the *long-term* desirability of states after taking into account the states that are likely to follow, and the rewards available in those states. For example, a state might always yield a low immediate reward but still have a high value because it is regularly followed by other states that yield high rewards. Or the reverse could be true. Let's make a human analogy. Rewards are like pleasure (if high) and pain (if low), whereas values correspond to a more refined and farsighted judgment of how pleased or displeased we are that our environment is in a particular state.
4) **Model:** *Model* is something that mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for *planning*, which deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively new development. Early reinforcement learning systems were explicitly trial-and-error learners; which was almost the *opposite* of planning. Nevertheless, it gradually became clear that reinforcement learning methods are closely related to dynamic programming methods, which do use models, and that they in turn are closely related to state-space planning methods. Modern reinforcement learning spans the spectrum from low-level, trial-and-error learning to high-level, deliberative planning.

Rewards are in a sense primary, whereas values, as predictions of rewards, are secondary. Without rewards there could be no values, and the only purpose of estimating values is to achieve more reward. Nonetheless, it is values with which we are most concerned when making and evaluating decisions. Action choices are made based on value judgments. We seek actions that bring about states of highest value, not highest reward, because these actions obtain the greatest amount of reward for us over the long run.

In decision-making and planning, the derived quantity called value is the one with which we are most concerned. Unfortunately, it is much harder to determine values than it is to determine

rewards. Rewards are basically given directly by the environment, but values must be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. In fact, the most important component of almost all reinforcement learning algorithms is a method for efficiently estimating values. The central role of value estimation is arguably the most important thing we have learned about reinforcement learning over the last few decades.

Although estimating value functions is not strictly necessary while solving a reinforcement learning problem. For example, search methods such as genetic algorithms, genetic programming, simulated annealing, and other function optimization methods have been used to solve reinforcement learning problems. These methods search directly in the space of policies without ever appealing to value functions. We call these *evolutionary* methods because their operation is analogous to the way biological evolution produces organisms with skilled behavior even when they do not learn during their individual lifetimes. If the space of policies is sufficiently small, or can be structured so that good policies are common or easy to find, then evolutionary methods can be effective. In addition, evolutionary methods have advantages on problems in which the learning agent cannot accurately sense the state of its environment.

Nevertheless, what is mean by reinforcement learning involves learning while interacting with the environment, which evolutionary methods do not do. Evolutionary methods tend to ignore much of the useful structure of the reinforcement learning problem: they do not use the fact that the policy they are searching for is a function from states to actions; they do not notice which states an individual passes through during its lifetime, or which actions it selects. In some cases this information can be misleading (e.g., when states are misperceived), but more often it should enable more efficient search.

(II) Markov Decision Process [6]

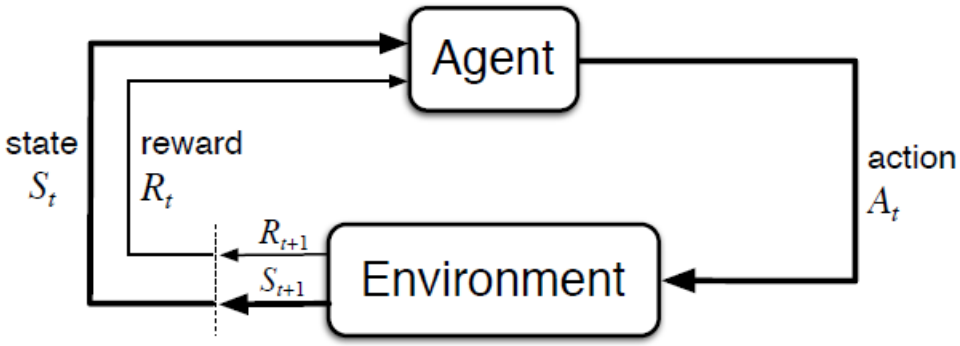


Fig. Agent-Environment Interaction in a Markov Decision Process [6]

Markov Decision Process (MDPs) [5] are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The learner and decision maker is called *agent*. The thing it interacts with, comprising everything outside the *agent*, is called *environment*. They interact continually, the agent selecting actions and the environment responding to these *actions* and presenting new situations to the *agent*. The *environment* also gives rise to *rewards*, special numerical values that the *agent* seeks to *maximize* over time through its choice of *actions*.

If the states and action spaces are finite, then the problem is called a finite MDP. Finite MDPs are very important for RL problems and much of literatures have assumed the environment is a finite MDP in their works.

Any reinforcement learning problem can be modeled as a Markov Decision Process. MDP's are a classic formalization of sequential decision making, where actions influence not just immediate rewards, but also subsequent situations, or states, and through those, future rewards.

MDPs involves delayed reward and the need to tradeoff immediate and delayed reward. Whereas in bandit problems we estimated the value $q_*(a)$ of each action a, while in MDPs we estimate the value $q_*(s, a)$ of each action a in each state s, or we estimate the value $v_*(s)$ of each state given optimal action selections. These state-dependent quantities are essential to accurately assign credit for long-term consequences to individual selection of actions.

More specifically, the agent and environment interact at each of a sequence of discrete time steps, t = 0, 1, 2, 3,… At each time step t, the agent receives some representation of the environment's *state*, $S_t \in S$, and on the basis of that it selects and *action*, $A_t \in A(s)$. After one time step, as a consequence of action, the agent receives a reward, $R_{t+1} \in R$ and finds a new state for itself, S_{t+1}. Thus this gives rise to a sequence that looks like $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots\ldots$

In a *finite* MDP, the sets of states, actions, and rewards (*S,A,R)* all have finite number of elements in them. In this case, the random variables R_t and S_t have well defined discrete probability distributions dependent only on the preceding state and action. For a particular value of these random variables, *s'* and *r*, there is a probability of those values occurring at time *t,* , for given particular values of the preceding states and action:

$$p(s', r|s, a) \equiv \Pr\{S_t = s' R_t = r | S_{t-1} = s, A_{t-1} = a\}$$

for all *s', s* ∈ *S, r* ∈ *R,* and *a* ∈ *A(s).* The function p defines the dynamics of the MDP.
The dot over the equals sign in the equation reminds us that it is a definition (in this case of the function p) rather than a fact that follows from previous definitions. The dynamics function $p: S\ x\ N\ xS\ x\ A \rightarrow$ [0, 1] is an ordinary deterministic function of four arguments. The '|' comes from the notation of conditional probability, but here it just reminds us that *p* specifies a probability distribution for each choice of *s* and *a,* that is,

$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r | s, a) = 1, \text{ for all } s \in \mathcal{S}, a \in \mathcal{A}(s).$$

In a Markov decision process, the probabilities given by *p* completely characterize the environment's dynamics which means that the probability of each possible value for S_t and R_t depends only on the immediately preceding state and action, S_{t-1} and A_{t-1}, and, given them, not at all on all earlier states and actions. This can be viewed as a restriction not on the decision process, but just on the *state*. The state generally includes information about all aspects of the past agent-environment interactions which makes a difference in the future. So if the above is followed, then the state is said to have a *Markov Property.*

The MDP framework is abstract and flexible and can be applied to many different problems in different ways. For example, the time steps need not refer to fixed intervals of real time; they can be arbitrary successive stages of decision making and acting. The actions can be low-level controls, such as the voltages applied to the motors of a robot arm, or high-level decisions, such as whether or not to have lunch or to go to graduate school. Just like actions, states can take a wide variety of forms. They can either be completely determined by low-level sensations, such as direct sensor readings, or they can be high-level and abstract, such as symbolic descriptions of objects in a room.

The MDP framework is a considerable abstraction of the problem of goal-directed learning from interaction. It proposes that whatever the details of the sensory, memory, and control apparatus, and whatever objective one is trying to achieve, any problem of learning goal-directed behavior can be reduced to three signals passing back and forth between an agent and its environment: one signal to represent the choices made by the agent (the actions), one signal to represent the basis on which the choices are made (the states), and one signal to define the agent's goal (the rewards). This framework may not be sufficient to represent all decision-learning problems usefully, but it has proved to be widely useful and applicable.

(III) Model Based and Model Free RL Algorithm [7]:

When going about different approaches to a RL problem, they can be broken broadly in 2 categories. Model-Free and Model-Based algorithms.

Model Free Algorithms that tend to learn through the experience gained from interactions with the environment, i.e. this algorithm tries to estimate the optimal policy without using or estimating the dynamics (transition and reward functions) of the environment.

Model Based Algorithms on the other hand is an approach that uses a learnt-model i.e. transition probabilities and reward function to predict the future action (optimal policy).

(IV) Q-Learning [4] :

Q-learning in its all glory and form was brought as a result of a specific variant of temporal difference learning (TD) initially a part of PhD Thesis by Watkins [8] and adopted & proposed by Watkins and Dayan [4].

Q-learning is a *value-based* model free learning algorithm. Value based algorithms updates the value function based on an equation (particularly Bellman equation). Whereas the other type, *policy-based* estimates the value function with a greedy policy obtained from the last policy improvement.

The 'Q' in Q-learning stands for quality. Quality here represents how useful a given action is in gaining some future reward.

Speaking about reward, when considering rewards the following the reward tends to follow the same pattern as that of a MDP. A reward $R_a(s_j,s_k)$ is obtained when taking action a_i in state s_j and the environment/system changes to state s_k after the decision maker takes action a_i. The decision maker follows a policy, π such that $\pi(\cdot):S\rightarrow A$, that for each state $s_j \in S$ takes an action $a_i \in A$. So that the policy is what tells the decision maker which actions to take in each state. The policy π may be randomized as well. Discount factors are associated with time horizons.

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1},$$

Longer time horizons have much more variance as they include more irrelevant information, while short time horizons are biased towards only short-term gains. The discounted reward phenomenon tends to make an infinite series finite. Thus aiming to maximize the long term reward instead of short term rewards. The discount factor essentially determines how much the reinforcement learning agents cares about rewards in the distant future relative to those in the immediate future.

If γ=0, the agent will only learn about actions that produce an immediate reward. If γ=1, the agent will evaluate each of its actions based on the sum total of all of its future rewards.

(V) Deep Q Network (DQN) [9]:

One of the breakthrough paper on implementation of deep learning for reinforcement learning can be credited to researchers from Deepmind for publishing their paper "Human-level control through deep reinforcement learning" [9] in 2015 which they trained agents on multiple Atari games which the only input being the screen on the game. This laid the foundation for the merging together of two fields Reinforcement Leaning and Deep Learning and birthed a new field of Deep Reinforcement Learning or DRL. Every time the agent makes a move in the environment, in this case the images of the game, it created a tuple of 4 variables called experience which consists current state, action it performed, the state it landed in and the reward it got for performing that operation *(s, a, s_{t+1}, r)* and then store these experiences by creating it's own data set in a replay memory. *Replay Memory* sounds same as you might think. It stores all the past experiences of the RL agent and then replays it again and again and learns from it. Now here the Q values from the Q Learning comes into picture helping the agent to take future action.

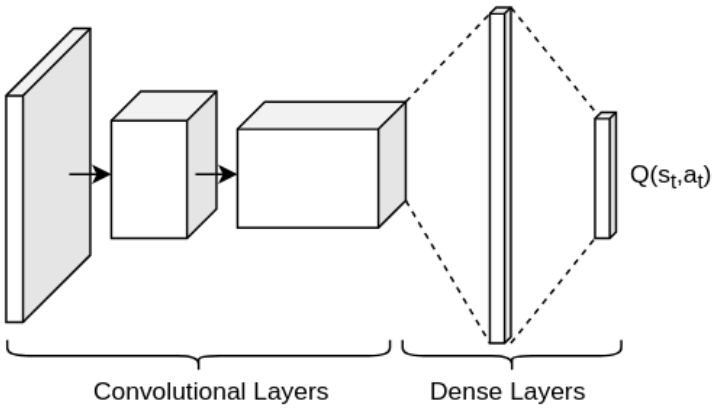


Fig. Network structure of Deep Q-Network (DQN), where Q-values Q(s,a) are generated for all actions for a given state [7]

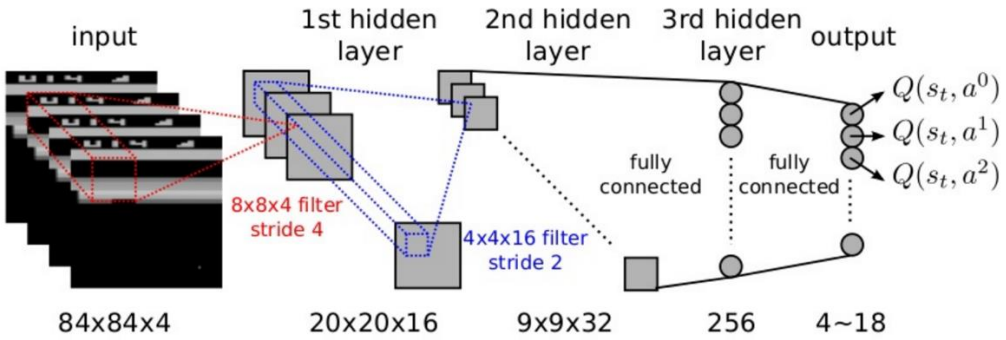
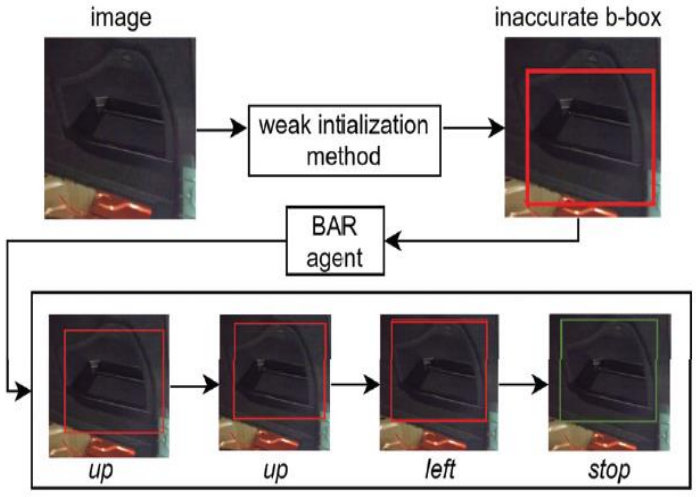


Fig. Exploded View of the above DQN [10]

Problem Statement

The base paper that I am aiming to work on and improve is titled "BAR- A Reinforcement Learning Agent for Bounding-Box Automated Refinement" [11]. The paper starts with the brief introduction of how deep learning has basically taken over the world due to its wide ranging applications mainly in the field of computer vision. Object detection and recognition deep learning techniques allowing machines to visualize their environment with pretty high accuracy. Thus we are seeing a high number of industries using computer vision in their day to day operations be it inventory management, process control, quality control and much more. But there is a drawback, deep learning still being a mix of supervised and unsupervised learning needs amount of labeled data with a high degree of accuracy which takes considerable amount of effort, time and money. Image annotations also are highly prone to human error and it the might happen that the images being biased towards one side, the human annotator might label the image wrongly. Even thought the whole dataset is labeled, it might happen that the Region of Interest (RoI) of the target object might change because of industrial circumstances. The cost of re-labelling is just not feasible. To the best of knowledge existing literature tends to generate bounding boxes around the target object but no attempts have been made to correct these bounding boxes. In summary, existing work focuses on reducing the time spent by human annotators on manual labeling. While, in the literature, this goal is achieved by finding alternative ways to generate b-boxes proposals, our work focuses on learning to correct inaccurately generated b-boxes to later refine annotations regardless of their initialization [11].



Every image contains exactly one annotated target object whose b-box is represented by its upper-left corner (x_{min}, y_{min}) and its lower-right corner (x_{max}, y_{max}). This b-box is considered inaccurate if its IoU with the groundtruth is below a threshold, denoted by β. Given an image and an inaccurate b-box enclosing the target object, the goal of the agent is to correct the b-box as shown in figure here. The agent achieves this goal by executing a series of actions that modify the position and aspect-ratio of the b-box. This series of actions corresponds to an episode that ends with the final correction of the agent [11]. At time step t, the agent updates its Q-value estimate in the following manner:

$$Q_{t+1}(s, a) = (\alpha - 1)Q_t(s, a) + \alpha(r + \gamma \max_{a'} Q_t(s', a'))$$

The three main components of BAR-DRL are:

1. State: The state is composed of a feature vector $\in R^{1238}$ extracted using ResNet50 [13] pretrained on ImageNet [12] and a history vector. The b-box enclosed region is resized to 224×224, then fed to the feature extractor that outputs a vector of size 512. The history vector encodes the 10 actions of the episode, each of which is represented as a one hot encoder.
2. Actions: These are the eight translation actions shown in table below and the *stop* action.

action	corresponding equations	action	corresponding equations
up	$x_{min} - c_1 \times height$ $x_{max} - c_1 \times height$	*wider*	$y_{min} - c_2 \times width$ $y_{max} + c_2 \times width$
down	$x_{min} + c_1 \times height$ $x_{max} + c_1 \times height$	*taller*	$x_{min} - c_2 \times height$ $x_{max} + c_2 \times height$
left	$y_{min} - c_1 \times width$ $y_{max} - c_1 \times width$	*fatter*	$x_{min} + c_2 \times height$ $x_{max} - c_2 \times height$
right	$y_{min} + c_1 \times width$ $y_{max} + c_1 \times width$	*thinner*	$y_{min} + c_2 \times width$ $y_{max} - c_2 \times width$

3. Reward: The reward for a translation action *a* at step *t* is:

$$r(a_t) = \begin{cases} 1, & \text{if } IoU_t > IoU_{t-1} \\ -3, & \text{otherwise} \end{cases}$$

A higher negative value is necessary when the IoU decreases to prevent the agent from worsening the initial b-box, which defeats the purpose of a correcting agent. The reward for the *stop* action is:

$$r(a_t) = \begin{cases} +r_1 + c * \Gamma, & \text{if } IoU_t \geq \beta \\ -r_2, & \text{otherwise} \end{cases}$$

where: $\Gamma = \frac{10-r}{10}$, $r1 = 6$, $c = 4$, $r2 = 3$ [11].

Problem Statement: As from the above explanation we can see that the translation actions that is being taken is discrete in nature. A possible future work as mentioned in the said paper and the main are of proposed work for my research work is instead of using discrete action space for translations action what happens when we use continuous translation action and what are the improvements one can observe in the said method. So for the continuous translation action my proposed name is "ConTra- Continuous Translation Action Agent for bounding box refinement".

Work Done Till Now

For simulation of the paper I am working on Pycharm with Tensorflow 2.7, Python 3.9, cuDNN 8.2.1 and cuda 11.5 on my laptop having 8GB RAM and Nvidia GTX1650 GDDR6 having 4 GB of VRAM. Since the authors used a private dataset, I have instead used PASCAL VOC 2007.

For execution, 20 images of "aeroplane" class was chosen from PASCALVOC 2007. 15 images form the training dataset and 5 form testing dataset. Wrong manual annotations where done on these 20 images and below contains the IoU of those images for different epochs starting from 50, 80,100 and thresholds varying from 0.50 to 0.80.

The neural network consists of two fully-connected layers of 500 neurons each with ReLu activation and random normal initialization, and an output layer of 9 neurons with linear activation. Mean square error loss with Adam optimizer and learning rate of 0.001 are used, and the discount factor γ for the Q-function is set to 0.90.

Image Name	Epochs	Threshold	Initial IoU	Final IoU	Max IoU
009365.jpg	50	50	0.36	0.33	0.4
	80	50	0.36	0.33	0.4
	100	50	0.36	0.33	0.4
	50	70	0.36	0.33	0.4
	80	70	0.36	0.33	0.4
	100	70	0.36	0.36	0.4
	50	75	0.36	0.33	0.4
	80	75	0.36	0.33	0.4
	100	75	0.36	0.48	0.48
	50	80	0.36	0.33	0.4
	80	80	0.36	0.33	0.4
	100	80	0.36	0.33	0.4
009461.jpg	50	50	0	0	0
	80	50	0	0.03	0.03
	100	50	0	0.03	0.03
	50	70	0	0.03	0.03
	80	70	0	0.01	0.01
	100	70	0	0	0.01
	50	75	0	0	0
	80	75	0	0.03	0.03
	100	75	0	0.05	0.05
	50	80	0	0.03	0.03
	80	80	0	0.03	0.03

	100	80	0	0.03	0.03
009480.jpg	50	50	0.58	0.73	0.76
	80	50	0.58	0.56	0.6
	100	50	0.58	0.65	0.74
	50	70	0.58	0.48	0.6
	80	70	0.58	0.73	0.76
	100	70	0.58	0.53	0.6
	50	75	0.58	0.73	0.76
	80	75	0.58	0.48	0.6
	100	75	0.58	0.51	0.6
	50	80	0.58	0.58	0.58
	80	80	0.58	0.54	0.6
	100	80	0.58	0.48	0.6
009615.jpg	50	50	0.91	0.56	0.91
	80	50	0.91	0.61	0.93
	100	50	0.91	0.56	0.91
	50	70	0.91	0.9	0.93
	80	70	0.91	0.82	0.91
	100	70	0.91	0.61	0.93
	50	75	0.91	0.44	0.93
	80	75	0.91	0.82	0.93
	100	75	0.91	0.61	0.93
	50	80	0.91	0.9	0.93
	80	80	0.91	0.93	0.93
	100	80	0.91	0.75	0.93
009702.jpg	50	50	0.04	0.02	0.04
	80	50	0.04	0.03	0.04
	100	50	0.04	0.03	0.04
	50	70	0.04	0.03	0.04
	80	70	0.04	0.02	0.04
	100	70	0.04	0.05	0.05
	50	75	0.04	0.02	0.04
	80	75	0.04	0.03	0.04
	100	75	0.04	0.05	0.05
	50	80	0.04	0.03	0.04
	80	80	0.04	0.03	0.04
	100	80	0.04	0.03	0.04

Future work to be done is increasing the dataset size and analyzing the parameter tuning and it's value on the final average IoU.

References

[1] Nasteski, Vladimir. "An overview of the supervised machine learning methods." *Horizons. b* 4 (2017): 51-62.

[2] Celebi, M. Emre, and Kemal Aydin, eds. *Unsupervised learning algorithms*. Berlin: Springer International Publishing, 2016.

[3] Sutton, Richard S., and Andrew G. Barto, "Reinforcement learning", *Journal of Cognitive Neuroscience* 11.1 (1999): 126-134.

[4] Christopher JCH Watkins and Peter Dayan. Q-learning. Machine learning, 8(3-4):279–292, 1992.

[5] White III, Chelsea C., and Douglas J. White. "Markov decision processes." *European Journal of Operational Research* 39.1 (1989): 1-16.

[6] Sutton, Richard S., and Andrew G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[7] Le, N., Rathour, V. S., Yamazaki, K., Luu, K., & Savvides, M. (2021). Deep reinforcement learning in computer vision: a comprehensive survey. *Artificial Intelligence Review*, 1-87.

[8] Watkins, C. J. C. H. (1989). Learning from delayed rewards.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. Nature, 518(7540):529–533, 2015.

[10] Artificial Intelligence, Leonardo Araujo dos Santos.

[11] Ayle, M., Tekli, J., El-Zini, J., El-Asmar, B., & Awad, M. (2020). BAR — A Reinforcement Learning Agent for Bounding-Box Automated Refinement. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(03), 2561-2568

[12] J. Deng, W. Dong, R. Socher, L. Li, Kai Li and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database," 2009 IEEE Conference on Computer Vision and Pattern Recognition, 2009, pp. 248-255,

[13] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).

[14] Everingham, M., Van Gool, L., Williams, C.K.I. *et al.* The PASCAL Visual Object Classes (VOC) Challenge. *Int J Comput Vis* **88,** 303–338 (2010).