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. Labeled data means that when the data is being trained, we already have told the learning algorithm about given inputs and expected outputs. A few examples of supervised machine learning are of linear regression, naïve bayes, logistic regression, K-Nearest Neighbor, Support-Vector Machine, decision tree, neural networks etc. So the main requirement for this type of machine learning is labeled data.
- Unsupervised Learning [2], mainly uses machine learning algorithms to analyze and cluster unlabeled datasets. These algorithms try to discover hidden patterns or groupings of data without the need for human interference. These abilities to discover similarities and differences in information makes it the ideal solution for different approaches like exploratory data analysis, bio-informatics, clustering, analysis of sequences, segmentation of customers, image recognition and many more. Few of the unsupervised learning approach are 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 historical backdrop of reinforcement learning generally deals with two main ideologies. One ideology is concerns itself when one learns by the method of trial and error which initiated with the understanding of animal psychology of how they learnt and responded. This idea runs along with some of the earlier works in the field of artificial intelligence which proceeded to the resurrection of the field known as reinforcement learning in the 1980s. The next ideology it matters itself is with the problem of optimal control and whose solution is by generally found by using the approach of dynamic programming and value function. For the most part, this idea has not considered learning. Although both the ideas have been quite independent, though there are omissions revolving around a third and less transparent idea concerned with temporal-difference methods. All the three ideologies culminated together in the late 1980s to produce what we know as today's modern field of reinforcement learning.

The term known "optimal control" arose in existence as of late 1950s which was used to describe a design problem dealing with a controller which minimizes the parameter of a dynamic system and its behavior a period of time. One among many of such ways to this problem was thought of in the mid-1950s by Richard Bellman and other researchers which was just the extension of a theory proposed by Hamilton and Jacobi in the 19th century [4]. The particular approach used the concept of the state of the dynamic system and that of a value function, or in other words an "optimal return function," [4] which was used to define an equation, now famously known as the "Bellman equation". These types of approaches that aim at solving the control problem optimality by solving the "Bellman equation" is widely to known as "dynamic programming". Bellman also popularized the stochastic as well as the discrete version of the control problem optimality famously known as Markovian decision processes (MDPs), and Ron Howard was the one who concluded the policy iteration method for MDPs [14]. All of the idea discussed above are the most essential elements which underlie in the theory and algorithms of today's modern day reinforcement learning.

Dynamic programming still to date is widely considered the only feasible way to solve general stochastic optimal control problems. Such optimal problems suffer from what Bellman used to identify as "the curse of dimensionality," which means that as the number of state variable in the given problem increased the computational requirement needed to actually compute the values of the variable increase exponentially. But even after these setbacks it is still way more efficient, accepted and more widely applicable than any other general method. Dynamic programming has a very extensive arc of development starting since the late 1950s, which also includes and extension to the new and interesting field of partially observable DPs.

Now again focusing back on the other major idea which currently leads us into the current and new area of reinforcement learning revolves around the idea to learn from trial and error. This specific idea find its root in psychology, where "reinforcement" theories of learning are abundant and common. Perhaps the first one to express the essence of trial-and-error learning succinctly was Edward Thorndike, who in a nutshell said that any action rewarded with good i.e. positive or bad i.e. negative outcomes have a tendency to be re-elected and changed accordingly. Thorndike proposed this phenomenon to be known as the "Law of Effect" because it described how the tendency of selecting different actions were actually effected by that of reinforcing events.

Now understanding the third idea, dealing with the history of reinforcement learning, concerned with the idea of learning known as temporal-difference. Temporal-difference learning methods are very unique in the idea that they are driven by the differences in between temporally consecutive estimates of the same quantity, like for example the probability to win a game of tic-tac-toe . This approach is less distinctly used than the other two approaches, but it has played a particularly important role in this field mostly because temporal-difference methods seemingly are quite new and unique in the current field of reinforcement learning.

The origin dealing with that of temporal-difference learning can again be in part be attributed to psychology of animal behavior and learning notably in the concept known as "*secondary reinforcers"*. A *secondary reinforcer* is a stimuli that is generally paired with a *primary reinforcer* that can be food or pain and due to this as a result lead to a interchange in the reinforcing properties. The optimal control and temporal-difference ideas fully coincided together in 1989 when Chris Watkins's proposed and developed a novel concept knows as "Q-learning" [7].

Literature Survey

(I) Components of Reinforcement Learning [4]

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 [4]:** 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 [4]:** 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 [4]:** 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 [4]:** 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 [4]:** 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 [4]:** A *policy* defines learning the behavior of an agent and it's way at a given instance of time. In other words, a policy can be defined as a mapping technique which maps the agent's behavior from perceived states of the environment to actions it must take when it is present in one of those states. This can be correlated to psychological terms know as associations or stimulus-response rule. In some of the cases the policy involved can be a very simple function or it can be a lookup table, while in other instances it can involve very in-depth computation, for example a process to search. The policy is at the core of a reinforcement learning agent which means that all by itself, it is sufficient enough on it's own to determine the behavior of agents. In general sence, policies can be stochastic.

2) **Reward Function [4]:** A *reward function* is the one which gives the definition of a goal in a problem of reinforcement learning. In other words, it maps each state-action pair, in other words knows as perceived state, of the environment in which the agent is present to a unique countable number called a *reward*, which indicates intrinsically how desirable a given state is. The sole purpose and objective of a reinforcement learning agent is to maximize the total reward it will receive in the long run of the experiment instead of short term gain. The reward function also defines what good or bad events are for the agent in terms of the reward it gains. In a system of biology, it would only be appropriate to relate and analyze biological rewards with either pleasure or pain. These are the instantaneous and crucial features of the problem the agent faces in an experimental setting. The reward function should mandatorily be unchanged by the agent. It can instead be use as be a basis to alter the agent's 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. In general, reward functions can be stochastic.

3) **Value function [4]:** A *value function* is the one that tells the agent specifically about the goodness of what it is following for it's own in the distant future. In other words we can say that the *value* of a particular state is the total amount of reward an agent expectedly accumulates over the future run of it's experiment, starting from that specific state. Rewards are the parameters which helps the agent figure out the states immediate underlying desirability, whereas values specify the *long-term* appeal of states after taking into consideration of all those states that are likely to appear in future and the related rewards that might be available for those states. Let's say for example a state may always yield a low immediate reward but still yield a high value because the future states that follows the current state tends to earn quite high rewards or vice versa. Let's take for example human analogy. Rewards can be equated to pleasures if the reward is high and alternatively pain if the reward is low, whereas values correlating to a more refined and futuristic judgment of how satisfied or dissatisfied we are that the environment is present in that particular state.

4) **Model [4]:** *Model* can be thought of as one that tries to copy the behavior of the environment. Let's say for example, we know the state and the action, the model might be able to predict the expected next state and corresponding next reward. Models are generally used for *planning*, which decides on the course of action the agent must take by taking into consideration all the future situations that are possible before they actually come to fruition. The amalgamation of

models and planning in reinforcement learning systems is quite relatively a new development. Early systems of reinforcement learning were straightforward learners of trial and error , which is actually the total *opposite* of planning. Nonetheless, slowly and gradually it became clearer that reinforcement learning methods are quite closely related to dynamic programming methods which in-turn does use models and are in turn closely related to planning methods involving state and space. Modern reinforcement learning covers the whole spectrum starting from low-level trial and error learning to high-level deliberative planning.

Rewards in a way are primary parameters, whereas the values, as predictions of rewards, can be understood as secondary parameters. Without rewards there would not be any values and the one and only purpose to estimate the values is to nevertheless achieve the most reward. Regardless it is values with one should be most concerned about when making and evaluating the decisions. Value judgements are the one's based on which actions are executed. The aim is to try to seek out actions that convey around the states of highest value but not the most reward mainly because such actions procure the greatest amount of reward for the agent over the long term.

In planning and decision-making the borrowed quantity is called value and it is one of the main parameters. Unfortunately, it is easier to determine the rewards rather than determining values. Rewards are directly given by the environment the agent is present in but values are the one's that needs to be estimated and re-estimated from the sequences of observations an agent makes over its entire lifetime. As a matter of fact the most important constituent of nearly all the reinforcement learning algorithms is a method on how to effectively estimate values. The pivotal role of estimation of values is arguably the most essential thing that is known about reinforcement learning over the past couple of decades. Although it's not necessary to estimate value functions when solving a reinforcement learning problem. Like for example methods used for searching such as genetic algorithms or genetic programming and much more, methods for optimization have already been well-used to solve problems dealing with reinforcement learning. These methods search precisely in the policy space without ever engaging into value functions. These are known as *evolutionary* methods for the very reason because the way they operate is similar to the way evolution biologically produces living organisms with skilled behavior even though they have not learnt it during their individual lifetimes. If the policy space is adequately small or can in some way be structured so that good policy implementations become common or effortless to find then evolutionary methods can be very effective. Also in addition to this, evolutionary methods have several advantages on problems in which the learning agent cannot accurately understand the state of its environment.

Nonetheless, what it means is that reinforcement learning learns during it's interaction with the environment which evolutionary methods do not do. Evolutionary methods generally tends to discount much of the useful structures of a problem dealing in reinforcement learning. Such methods do not consider the fact that the policy being searched for is actually a function mapping from states to actions and do not consider which states an agent visits through during its lifetime or which actions the agent selects. In some cases such information can mislead (e.g., when states are misinterpreted).

(II) Markov Decision Process [4]

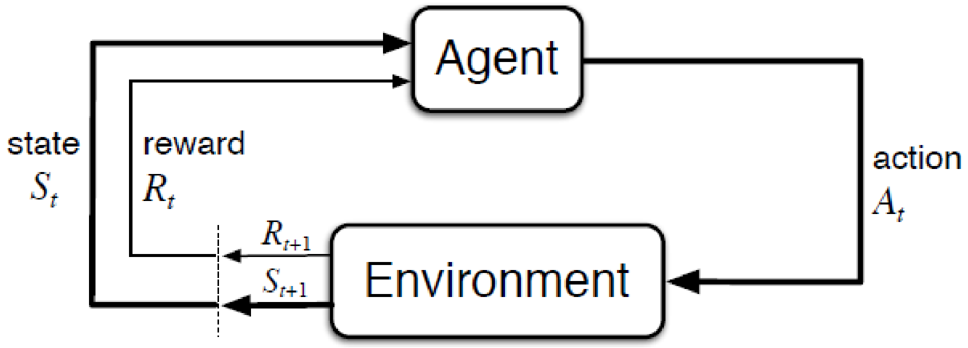


Fig. Agent-Environment Interaction in a Markov Decision Process [4]

Markov Decision Process (MDPs) [4] in simple words are nothing more but a direct framing up of a learning problem from interactions to achieve a specific goal. The main entity or learner and also the decision maker is known as an *agent*. The thing *agent* interacts with every single time which comprises everything outside the *agent*, is called an *environment*. They both interact again and again with the agent selecting actions and the environment responding to these *actions* and presenting new situations before the *agent*. The *environment* also gives rise to *rewards*, which are nothing but some special numerical values that the *agent* tries to *maximize* over time through its choice of *actions*.

If the states and action spaces are finite, then the problem so formed is known as a finite markov decision process (fMDP). Finite MDPs are very important for reinforcement learning [3] problems and most of literatures out in the scholarly world have assumed that the environment is a finite MDP in their works.

Any reinforcement learning problem can be modeled as a Markov Decision Process [14]. Markov Decision Processes are a classic formulation of sequential decision making, where actions tend to influence not just immediate rewards, but also subsequent situations or states, and through those, future rewards [4].

MDPs involves delayed reward and puts on a heavy emphasis on the need to adjust between immediate and delayed rewards. Conversely in problems contextualized as bandit problems, the value *q*(a)* of each action *a* is estimated, whereas in MDPs we try to predict the value *q*(s, a)* of each action *a* in each state *s*, or we tend to predict the value *v*(s)* of each state given optimal preference of actions. These state-dependent quantities are very crucial in order to accurately assign credit for long-term importance to individual election of actions.

The agent and environment interact with each other at each of the discrete time steps denoted by t = 0, 1, 2, 3,… At each discrete time step t, the agent receives some sort of representational information of the environment's current *state* given by $S_t \in S$, and on the basis of the state information it then selects an *action* given by $A_t \in A(s)$. Now after one time step reflecting the repercussion of an *action*, a reward is presented to the agent which is given by $Rt+1 \in R$ and thus lands itself in a new state denoted by $St+1$. Thus all this information above gives rise to a sequence that looks like $S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$ [4]

Now in a *finite* MDP, the multiple set of states, actions, and rewards (S,A,R) have all but *finite* number of elements in them. In such cases, the reward R_t and state St both have well defined discrete probability distributions which is reliant only on the preceding state and corresponding *action*. Now for a specific value of the random variables, s' and r, there is a probability of these values occurring at a given time t due to the values particularly given from that of the preceding states and corresponding actions which can be given by the equation mentioned below:

$$p(s', r|s, a) \equiv \Pr\{S_t = s' R_t = r|\, S_{t-1} = s\, , A_{t-1} = a\} \text{ [4]}$$

for all $s', s \in S, r \in R$, and $a \in A(s)$.

The function p here represents the system dynamics or the measurements of the specific MDP. The dot on the equals sign in the equation conveys the information that it is a definition, particularly in this case, the function p rather than a fact that follows from given previous definitions. The function of dynamics given by $p\colon S\ x\ N\ xS\ x\ A \rightarrow$ [0, 1] is an ordinary deterministic function which consists of four elements. The '|' is borrowed from the notation which denotes conditional probability, but in this context it just points out to us that p indicates a probability distribution for each choice of s and a, that is given by below [4],

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

In a Markov decision process, the probabilities conveyed by p comprehensively characterizes the environment's dynamic nature which in other words can be understood as the probability of every achievable value for S_t and R_t depending only on the immediate preceding state and it's corresponding action, given by S_{t-1} and A_{t-1} respectively and given both of the values and not at all on earlier states and/or actions. This can be viewed as a restriction on the state, but not on the decision process itself. The state generally includes all the necessary information about all aspects of all the previous agent-environment interactions which makes always makes a difference in the future. So if the all of the above conditions are met and followed, then the state is understood as to follow a *Markov Property.* [4] Discount factors are associated with time horizons as given below [4].

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

The MDP framework is quite conceptual but as well as very malleable and finds it's application in many distinct problems in varied ways. Like the time steps are not needed to have fixed intervals of real time; they instead can have arbitrary consecutive decision making stages and performing an action. The actions can be of low-level controls like the voltages applied to the motors of a robotic arm, or high-level decisions like whether or not to go and watch a movie or to go to a seminar or not. Similar to actions, states can take a lot of different forms. They can either be completely resolved by low-level sensations, like direct sensor readings, or they can be high-level and abstract in a sense as that of a symbolic description of objects present in a room.

The MDP framework can be thought of as a problem that deals with a directed method of learning with the goal to learn from various interactions. It comes up with the idea that whatever the details of the sensors, memory, and control setup are there and whichever objective one is trying to solve, any such problem of learning a goal-specific actions can always be reformulated as an ensemble three signals communicating in between an agent and its environment: one of the signal will represent the choice an agent makes (i.e. actions), one signal will represent the measure on which the choices are being made (i.e. states), and one more signal will define of what that agent will yield based on the choice and the measure on which choices are being made (i.e. rewards). This framework may not be sufficient to represent all decision-learning problems usefully, but it has proved to be widely useful and applicable. Just for a fun example below is a typical workday of a person.

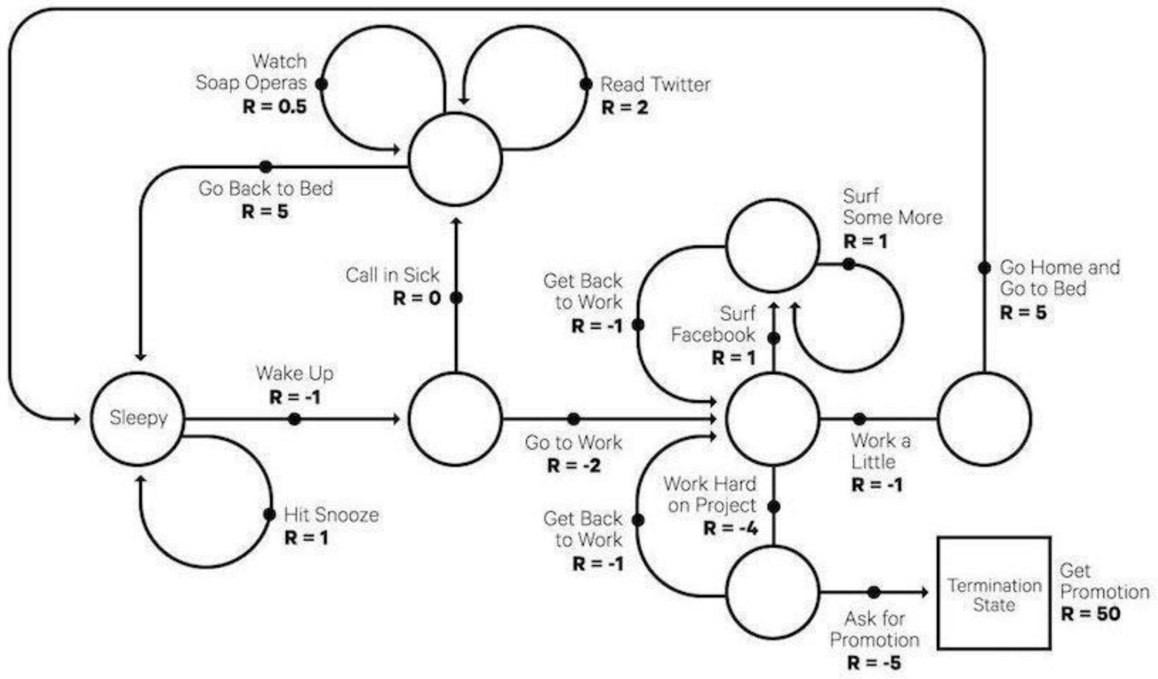


Fig. *A simple Markov Decision Process (MDP) that represents a typical workday* [16]

(III) Bellman Equations

Bellman equations are the equations that help us find value functions and optimal policies. Now from the knowledge one knows that in an environment the policy is bound to change with different experience and correspondingly different policies are bound to have different value functions. Thus the value function which gives the maximum value compared to all the other value functions is called the optimal value function. Below is the derivations of the Bellman Equation [4]

Starting:

$$V^{\pi}(s) = E[G_t \mid S_t = s] \quad \text{—— ①}$$

Here $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$

$$(\text{Return})\ G_t = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \quad 0 \leq \gamma \leq 1 \text{ Discount Rate}$$

$V^{\pi}(s)$ = value function of state 's' following policy 'π'

$\pi \rightarrow$ policy : $[S \xrightarrow{Map} A]$

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$$

$$= R_{t+1} + \gamma [R_{t+2} + \gamma R_{t+3} + \cdots]$$

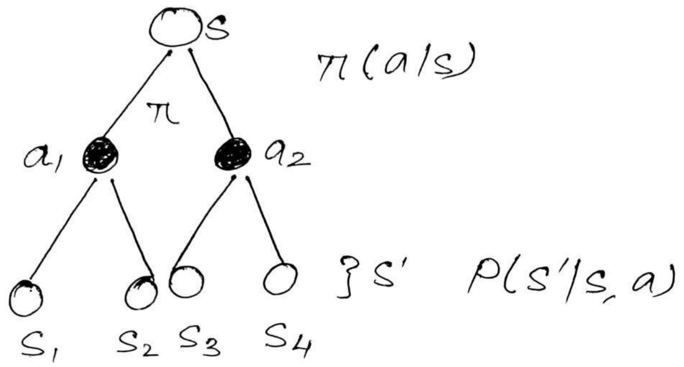
$$G_t = R_{t+1} + \gamma G_{t+1} \quad \text{—— ②}$$

Substituting ② in ①, we get

$$V^{\pi}(s) = E[R_{t+1} + \gamma (G_{t+1}) \mid S_t = s]$$

Consider a backup diagram:

State transition $$\begin{tikzcd} \textcircled{s} \arrow[r, bend left] & \textcircled{s'} \end{tikzcd}$$



$\pi(a|s) \to$ probability of taking action 'a' following policy 'π'

$P(s'|s,a) \to$ transition probability which represents enviroment dynamics.

For example: $P_{s_5 a_2} = \pi(a_2/s)\cdot P(s_5/s, a_2)$

(Probability of moving from '$s \to s_5'$', taking action 'a_2', following policy 'π')

Generalizing,

$$\mathbb{P} = \sum_{a\in A} \pi(a|s) * \sum_{s'\in S} P(s'|s,a) \quad \longleftarrow ③$$

$$\boxed{V^{\pi}(s) = \mathbb{P}\cdot E\left[R_{t+1} + \gamma G_{t+1} \mid S_{t+1} = s'\right]} \quad — ④$$

Substituting ③ in ④

$$\boxed{V^{\pi}(s) = \sum_{a\in A} \pi(a|s) \sum_{s'\in S} P(s'|s,a)\, E\left[R_{t+1} + \gamma G_{t+1} \mid S_{t+1} = s'\right]}$$

$$E[R_{t+1} + \gamma G_{t+1} | S_{t+1} = s']$$

$$\Rightarrow \{ E[r | s, a, s'] + \gamma E[G_{t+1} | S_{t+1} = s'] \}$$

We know that

$$V^{\pi}(s') = E[G_{t+1} / S_{t+1} = s']$$

$$\Rightarrow \{ E[r / s, a, s'] + \gamma V^{\pi}(s') \}$$

$$\boxed{\begin{aligned} V^{\pi}(s) = \sum_{a} \pi(a/s) \sum_{s'} p(s'/s, a) \times \\ \{ E[r / s, a, s' + \gamma V^{\pi}(s') \} \end{aligned}}$$

This above equation is known as "Bellman" Equation." for policy 'π'

Now, optimal policy, $\pi^* \rightarrow$ highest reward.

Optimal Value function, $V^*(s) = \max_{\pi} V^{\pi}(s)$

Bellman Optimality Equation,

$$V^*(s) = \sum_a \pi(a/s) \sum_{s'} p(s'/s,a)\left\{E[r/s,a,s'] + \gamma V^*(s')\right\} \quad \text{— (a)}$$

$q^*(s,a) \rightarrow$ optimal q - function

$$\because V^*(s) = \sum_a \pi(a/s)\, q^*(s,a)$$

$$\therefore q^*(s,a) = \sum_{s'} p(s'/s,a)\left\{E[r/s,a,s'] + \gamma \sum_a \pi(a/s')\, q^*(s',a)\right\} \quad \text{— (b)}$$

Both (a) & (b) are forms of Bellman Optimality Equation, but (a) is the most used form, i.e for value iteration or function

(IV) Model Based and Model Free RL Algorithm [5]:

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

Model Free Algorithms are the one's that try to learn through various experience gained during it's interactions with the environment, i.e. this algorithm tries to estimate the optimal policy without using or predicting the dynamics (reward and transition 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).

(V) Q-Learning [6] :

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 [6] and adopted & proposed by Watkins and Dayan [7].

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 by executing the last improvement of the policy.

The term 'Q' in Q-learning [7] is an abbreviation for quality. Quality in this setting represents how useful a given action is in regards of gaining some sort of futurereward.

Speaking about reward, when considering rewards the following the reward tends to follow the same pattern as that of a MDP. A reward $Ra\ (s_j, s_k)$ is obtained when executing an action ai in state s_j and the environment or system transforms to state s_k after the decision executer takes action a_i. The decision maker tend to follows a policy, π such that $\pi(\cdot)$:S→A, that for each state $s_j \in S$ takes an action $a_i \in A$. So that it is the policy which tells the decision maker which specific actions to take in each state. The policy π may be shuffled and randomized as well. Longer time experiences have more variance as compared to smaller one's as longer time includes more irrelevant information, while short time experiences 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 tells us the information on how much the reinforcement learning agents actually cares about it's rewards in the distant future compared to those in the immediate future of it's course of actions. Thus if γ=0, the agent will only focus on learning the actions that produce an immediate reward and if γ=1, the agent will calculate each of its actions based on the total sum of all of its future rewards.

(VI) Bounding Box

Bounding box in it's heart and soul is mostly just a rectangle that surround an object present in an image and tries to specify mainly it's position in the image, it's class and maybe confidence (probability of how likely that object is in that location). Bounding box are one of the most prominent techniques which draw the attention mainly when it comes to object detection and localization. They are generally used in the task of object detections where the main aim is to identify the position and type of maybe multiple objects in the image. A example's ensues this line.

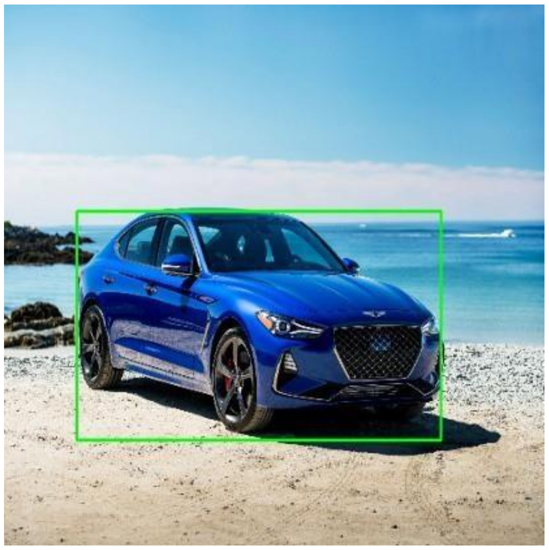
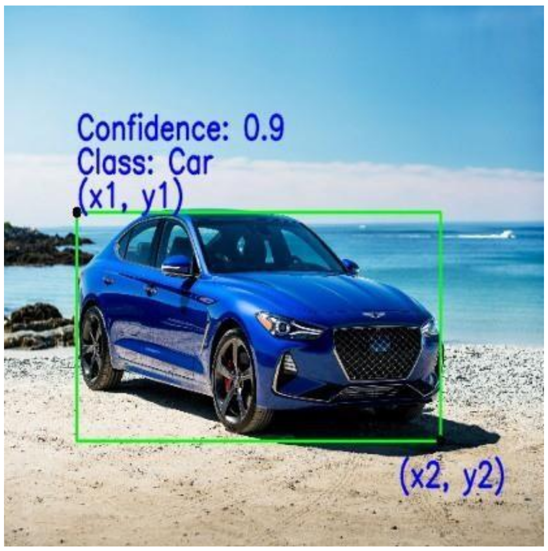


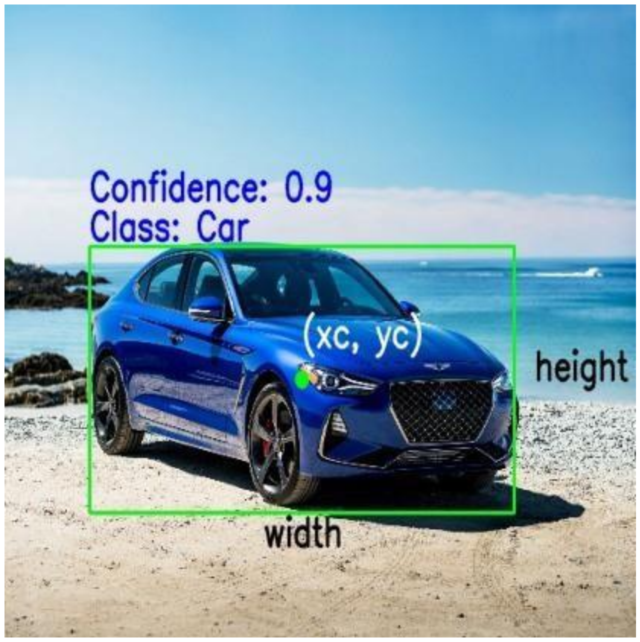
Fig. A Bounding-Box example

Now coming to the conventions used when specifying a bounding box are:

1) To specify the box with respect to the respective coordinates of it's top left and bottom right points.



2) To specify the box with respect to it’s center, height and width.



Now below are the list of parameters that are generally used to define the specifications of the bounding boxes:

1) Class: The first and foremost parameter is denoting the class to which a specific object in an image belongs to. For example car, person, truck, aeroplane etc.
2) (x1, y1): This corresponds to the x and y co-ordinates of the top left corner of the given rectangular box also can be written as xmin and ymin.
3) (x2, y2): This corresponds to the x and y co-ordinates of the bottom right corner of the rectangle also written as xmax and ymax.
4) (xc, yc): This corresponds to the x and y co-ordinates which denotes the center of the bounding box.
5) Width: This represents the width of bounding box.
6) Height: This represents the height of bounding box.
7) Confidence: This indicates the probability of the presence of the particular class of object present in that image. For example a confidence of 0.8 indicates that there is a 80% chance that the particular class of object actually exists in that box.

(VII) Resnet50[11]

Deep residual networks have become popular and most being the infamous ResNet-50[11]. This deep neural network is a convolutional neural network (CNN) which is as deep as 50 layers. A residual neural network (ResNet [11]) is a special type of artificial neural network (ANN) which is known to stack residual blocks one over the other to form a deep neural network.

In the past few years, the fascinating field dealing computer vision has went into very extensive research and transformation with the advent of very fast changing landscapes and technologies. As a result of this due to such improvement, it is now possible for different models dealing with computer vision to easily outperform humans and in a very efficient way solving a multitude of problems specifically dealing with object detection, image recognition, object localization, face recognition, image classification and much more.

With this in mind, the launch of such deep convolutional neural networks deserve a round of applause. These neural networks has been used greatly and in-depth for analysis of images with exceptional precision.

Even though having the flexibility to just keep on stacking layers after layers to our CNN's which might help us to solve even more complicated endeavors, they themselves come with their own sets of pretty big problems and issues. Most of the time it is observed that training a neural network becomes challenging and tough as we keep on increasing the number of layers. It is no guarantee that increasing the number of layers will be able to generate more feature and hence in a way increasing our accuracy, but instead the opposite happens. The training accuracy instead of improving actually deteriorates.

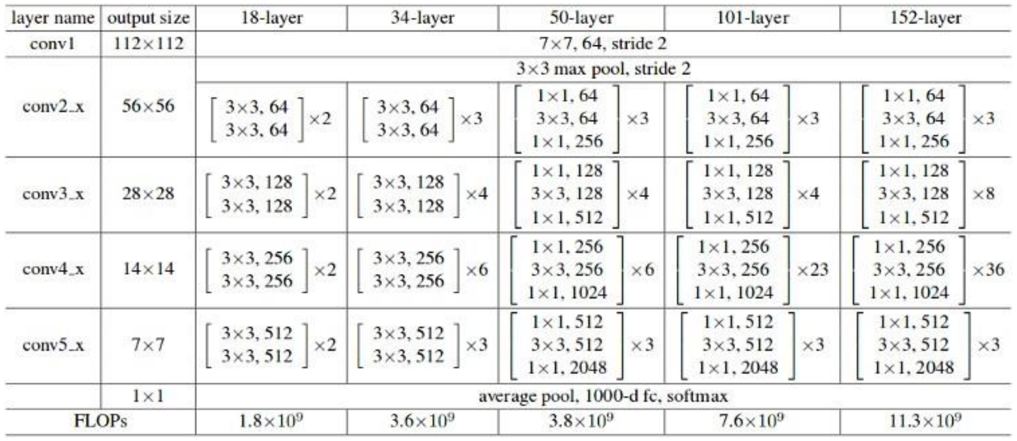
So to address the issue of deteriorating accuracy with the increase of layers the introduction and use of ResNet [11] became very important. The idea of ResNet [11] actually stemmed from the winning solution of ILSVRC 2012 known as "Imagenet classification with deep convolutional neural networks" authored by Alex Krizhevksy, Ilya Sutskever and Geroffrey E. Hinton [22]. This paper introduced the world to "AlexNet" named after the first author of [22]. This lead to a fierce battle among the researchers to build deeper CNN's. But even after increasing the number of layers there was no significant increase in the training accuracy. Even after if the training accuracy increased due to increasing depth it saturated after some layers. At that moment ResNet dropped as bomb in the research community of computer vision.

Now what is ResNet? ResNet[11] is actually an acronym of Residual Network. It was an innovative deep neural network approach that was first made known by the team of researchers from Microsoft Research lead by Kaiming He in 2015 who proposed the research paper with the title "Deep Residual Learning for Image Recognition" [11].

The reason of this model to be extensively studied used and popularity can be known from the fact is that the culmination of this convolutional model won the winning accolade at the ILSVRC 2015 image classification competition with a minimal error of 3.57%. In addition to that it also took the top spot in various other competitions held in 2015 like ImageNet, COCO and ILSVRC.

ResNet [11] in itself has many variants consisting of a number of different layer embedded in it that run on the same concept initially proposed by the research group. Resnet50 is one of the most influential networks where the number '50' denotes that the variant is working with 50 layer deep neural network.

When solving a problem related to areas like computer vision involving the use of deep convolutional neural networks, researchers engaged in stacking one layer after the other. But then came along the problem of vanishing/exploding gradients. This problem was then addressed by the used of normalized initialization of the layers. Thought the thought was that such additional layers might be able to help solve more and more complex problems with better efficiency as the different such layers can be trained for different tasks eventually leading and getting higher accuracy in results.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Fig. The different layer architectures of ResNet[11] variants

But the thought of multiple layers stacked one after the other can help enrich the features of the model, a network that is deep enough, ran into a problem. The problem of "degradation". When such deep- networks start to converge, with the now the ever increasing depth of the network, accuracy gets saturated and then degrades rapidly.

Now from the common notion we might think that this degradation might be a result of overfitting, but it is not the case. On the contrary, this might be a result of problems caused due to varying initialization of the networks, different optimization functions and most importantly, the ever known problem of vanishing/exploding gradients.

ResNet[11] was formulated keeping the intent of addressing this exact problem of degradation. Deep residual nets makes use of such residual blocks [11] to improve the accuracy of the models. The concept of “skip connections,” [11] which lies at the core of the residual blocks, is the strength of these type of deep neural networks.

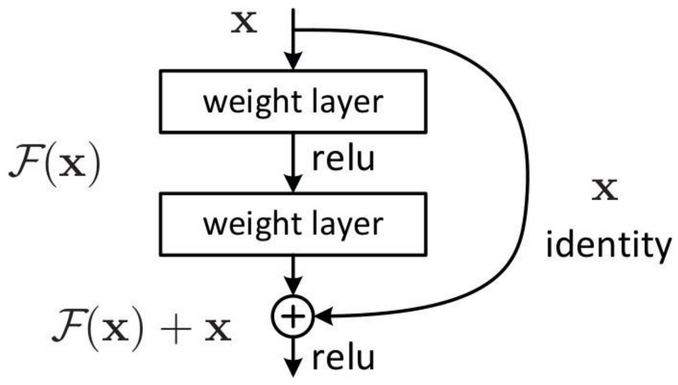


Fig. Skip Connection as mentioned in “Deep Residual Learning for Image Recognition” [11]

Now why skip connections? Skip connections as mentioned in the above diagram works in two ways. First of all they all mitigate the problem of vanishing gradient by creating an alternative route for the gradient to go via. Secondly they enable the neural network to learn an identity function. The aim of the identity function is to ensure that the upper layers of the neural network performs better that the lower layers.

Thus the presence of residual blocks helps to make it substantially easier for the network layers below to learn identity functions easily and efficiently. As a result, ResNet[11] enhances the efficiency of such deep neural networks which contains way more neural layers than vanilla CNN’s while having the ability to minimize the percentage of errors. In other words we can say that, the presence of skipping connections and adding the output from previous layers to the outputs of consecutive stacked layers skipping some of the layer in between making it easier to train much deeper networks than what was possible earlier.

The very first ResNet[11] architecture was the Resnet-34 which was initially mentioned in the research paper first introducing ResNet[11], which introduced shortcut connections so as to turn a simple neural network model into analogous residual network. In this case, the simple neural network drew inspiration from VGG [16] namely VGG-16 & VGG-19, with 3 X 3 filters making up the convolutional networks. But when drawing a comparison between VGGNets[16] and ResNet [11], the latter have lesser number of filters and comparatively less complex. The ResNet [11] having 34 layer achieved a batter performance of 3.6 billion floating point operations compared to 1.8 billion floating point operations of a smaller ResNet [11] with 18 layers.

There were two very simple rules of designing such a neural network – first being the layers when having the identical size of output feature maps and that of filters in them, and the second being that size of feature map be halved corresponding to the number of filters being doubled in order to preserve the time complexity of each and every layer. It was made up of 34 weighted layers.

Then the simple neural network is infused with the skipping connection. Keeping the dimensions of input and output same for the network, the identity functions are used directly. With the expected increase in the dimensions, two options are needed to be taken into consideration. The first of such option was to pad extra zeros to increase the dimension keeping the shortcut intact and letting it perform identity mapping. The other option was to match dimensions using projection shortcut. Below is the image of the ResNet[11] architecture.

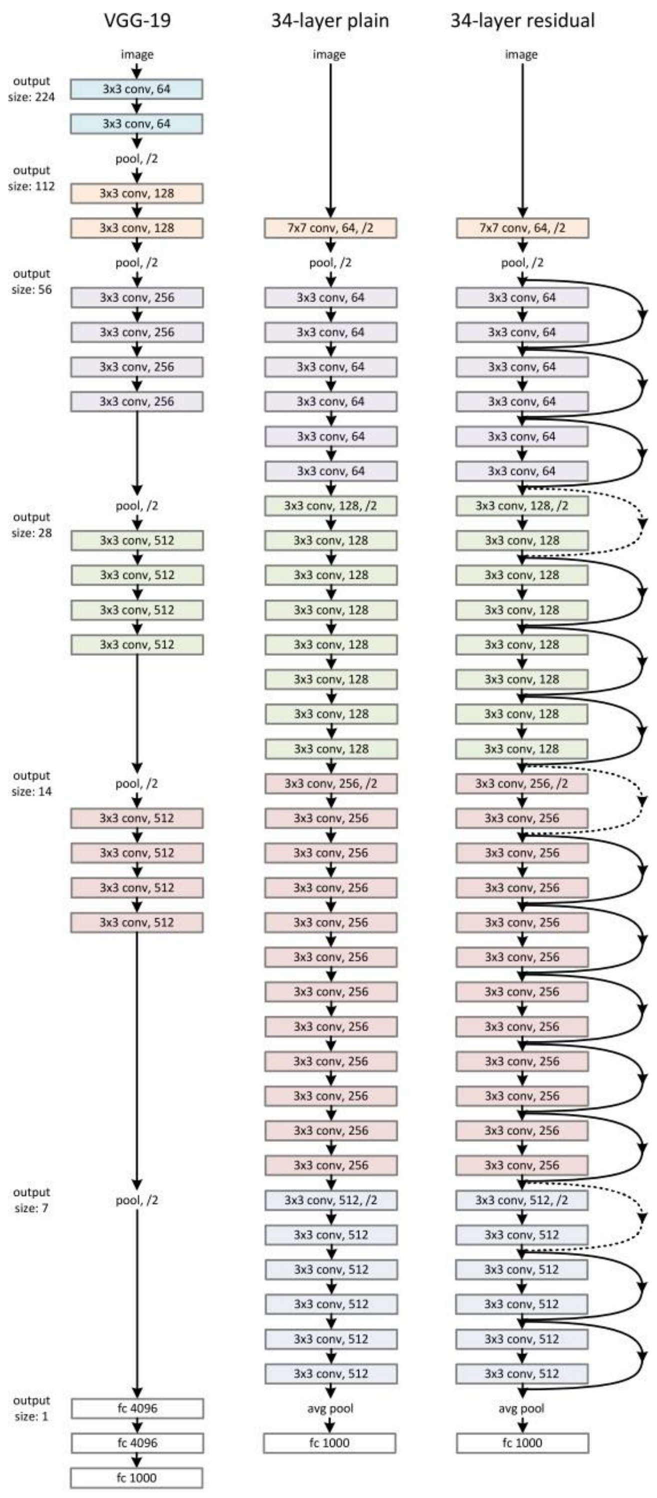


Fig. ResNet Architecture [11]

(VIII) Deep Q Network (DQN) [8]:

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" [8] 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 $(s, a, st+1, r)$ 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.

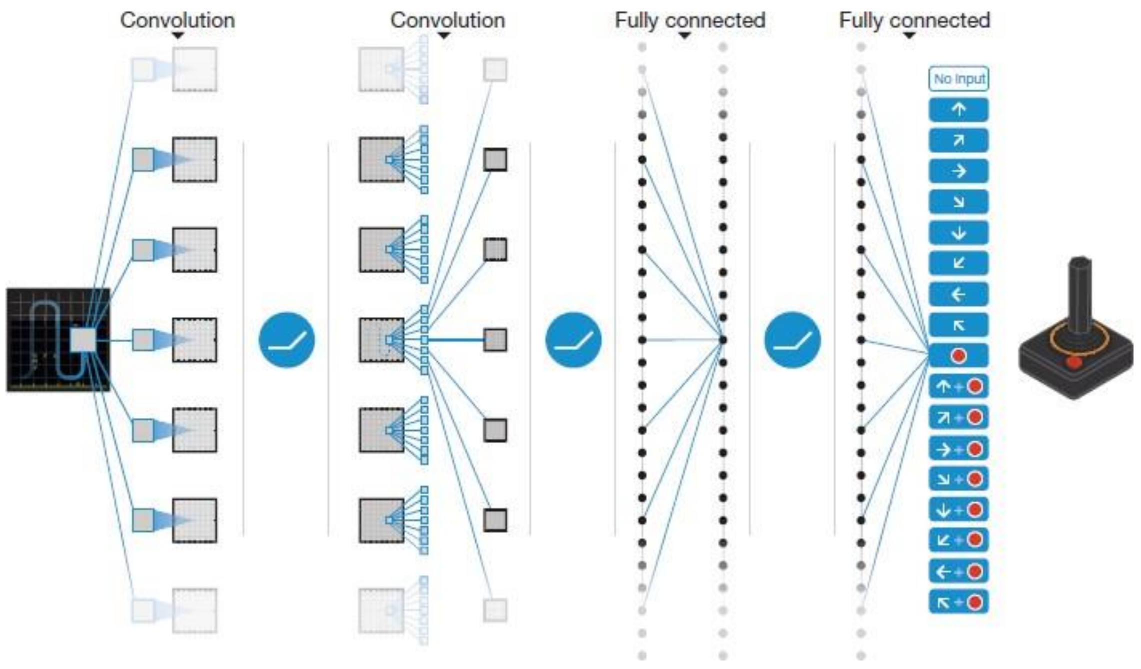


Fig. Network structure of Deep Q-Network (DQN), where Q-values Q(s, a) are generated for all actions for a given state [8]

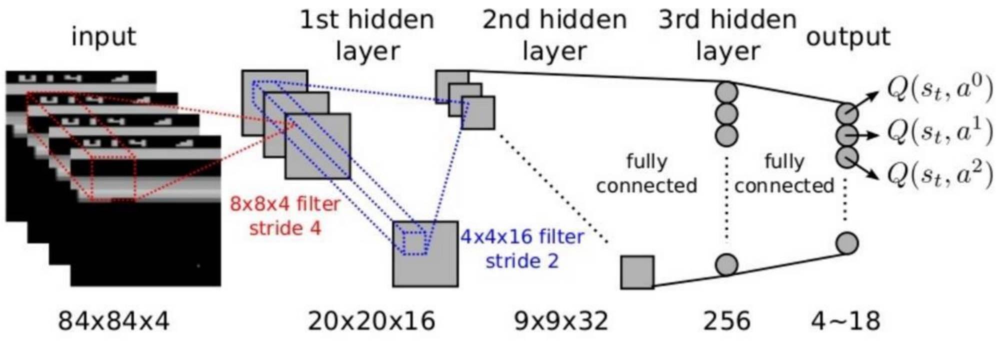


Fig. Exploded View of the above DQN [9]

(IX) YOLO - "'You Only Look Once"[19]

YOLO [19] (You Only Look Once) real-time object detection algorithm, in recent time has become one of the most intriguing and innovative ideas coming out of the computer vision research community especially of those object detection algorithms which is extremely effective that also encloses many of the shortcoming of various state of the art methods. Object detection from a long time is a critical capability especially in the field of autonomous vehicle technology. It is an area of computer vision that's exploding and working so much better than just a couple of years ago. YOLO [19] along with it's continuous updates has tend to fascinate everyone especially in the computer vision community with it's ease of implementation following better results every single time.

YOLO [19] appeared on the radar of computer vision when it was introduced in a paper in 2015 by Joseph Redmon known as "You Only Look Once: Unified, Real-Time Object Detection"[19] and in a small time grabbed a lot of attention of fellow researchers especially in the field of computer vision. He gave a small introduction of this in one of his TED talk organized by University of Washington in 2017 by highlighting this state of the art approach in a real time approach.

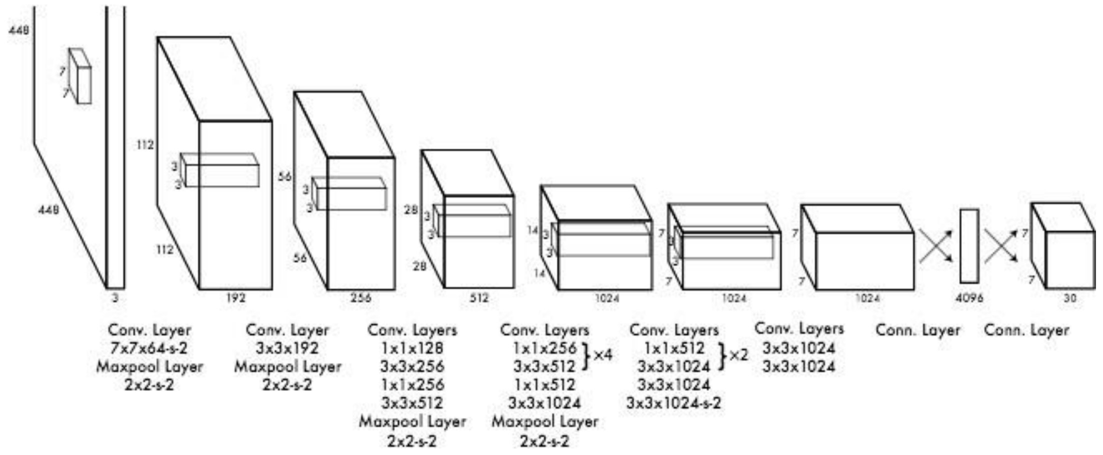
Object detection is amongst that category of perennial question in computer vision in which we all work to understand the where and what, especially where objects are present inside a given image and also what are they in the image or in other words what class do the object belong to. Object detection problem is comparatively more complex than classification, which also tends to perceive objects and not necessarily indicate where the location of object is in that specific image. In addition to this if there are more than one object present in the image then the classification fails.

In comes YOLO [19]. YOLO [19] completely changed to a approach that is a little different that the previous one's. It is a very intelligent CNN which does object detection in real-time. The algorithm uses a full image, applies it to a single neural network, bifurcates the whole image into multiple boxes and regions and then bounding boxes are predicted and for each such region probabilities are calculated. The bounding boxes used are assigned their respective weights based on the probabilities calculated for such bounding boxes.

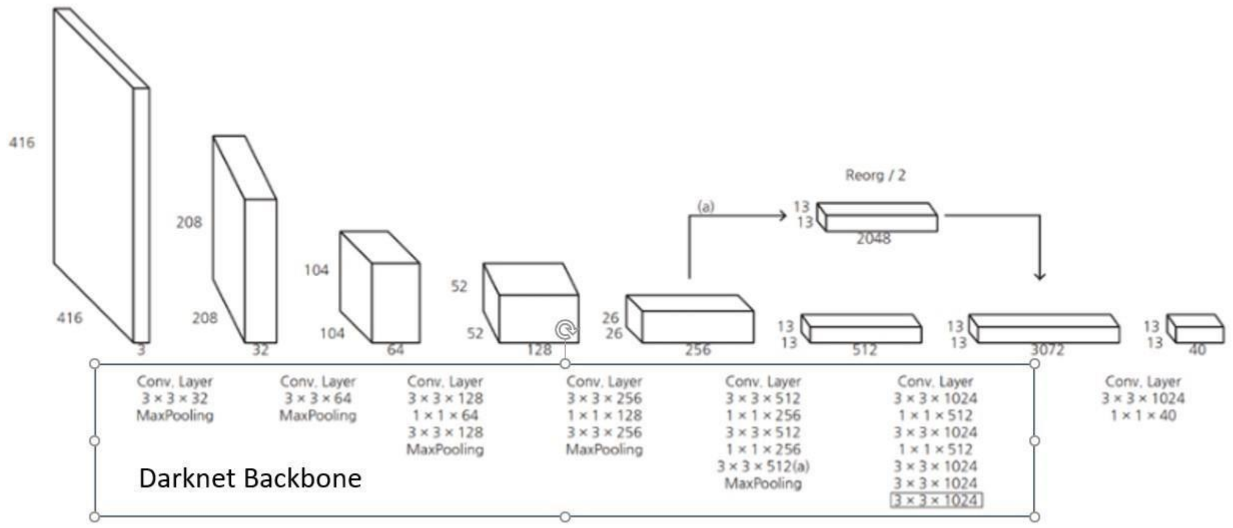
YOLO [19] is very popular because it has yielded very high accuracy compared to other SOTA methods while also having the ability to be executed in real-time. This algorithm considers the image for just once and in the way that it only requires forward propagation just once throughout the whole neural network to give predictions for that specific image. For multiple bounding box suggestions it uses non-max suppression method (which tries to make sure that it detects each object just once), it then outputs the recognized objects together with the corresponding bounding boxes.

Using YOLO [19], a single CNN consecutively predicts multiple bounding boxes and corresponding class probabilities for these bounding boxes. YOLO [19] trains on full images and directly optimizes the performance of it's detections. YOLO [19] has a number of advantages over other object detection methods:

- The speed of YOLO [19] is it's main selling point. Written in native CUDA and C, YOLO is blazing fast
- The entire image is fed to the YOLO [19] network as input while training and testing so that it on it's own can encode the information about it's classes context as well as appearance of it.
- YOLO [19] learns to generalize representations for the objects so that when it is trained on natural images and tested on other images, it outperforms nearly every other top detection methods.


448
448
7
7
3
112
112
3
3
192
56
56
3
3
256
28
28
3
3
512
14
14
3
3
1024
7
7
3
3
1024
7
7
1024
4096
7
7
30
Conv. Layer 7x7x64-s-2 Maxpool Layer 2x2-s-2
Conv. Layer 3x3x192 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x128 3x3x256 1x1x256 3x3x512 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x256 3x3x512 }x4 1x1x512 3x3x1024 Maxpool Layer 2x2-s-2
Conv. Layers 1x1x512 3x3x1024 }x2 3x3x1024 3x3x1024-s-2
Conv. Layers 3x3x1024 3x3x1024
Conn. Layer
Conn. Layer
Fig. YOLOv1 Architecture

Further research involving YOLO [19] has been conducted which resulted in the paper published in the month December of 2016 namely “YOLO9000: Better, Faster, Stronger,” [20] otherwise known as YOLOv2 by the same researchers, both of them Redmon and Farhadi, belonging to the University of Washington, which in itself provided a number of enhancements to the existing YOLO detection approach so as to include the detection of over 9,000 object categories by classification optimization and detection performed jointly.



416
416
3
208
208
32
104
104
64
52
52
128
26
26
256
13
13
512
(a)
Reorg / 2
13
13
2048
13
13
3072
13
13
40
Conv. Layer 3 × 3 × 32 MaxPooling
Conv. Layer 3 × 3 × 64 MaxPooling
Conv. Layer 3 × 3 × 128 1 × 1 × 64 3 × 3 × 128 MaxPooling
Conv. Layer 3 × 3 × 256 1 × 1 × 128 3 × 3 × 256 MaxPooling
Conv. Layer 3 × 3 × 512 1 × 1 × 256 3 × 3 × 512 1 × 1 × 256 3 × 3 × 512(a) MaxPooling
Conv. Layer 3 × 3 × 1024 1 × 1 × 512 3 × 3 × 1024 1 × 1 × 512 3 × 3 × 1024 3 × 3 × 1024 3 × 3 × 1024
Conv. Layer 3 × 3 × 1024 1 × 1 × 40
Darknet Backbone

Fig. YOLOv2 Architecture

Recently, the very same researchers published yet another paper in the month April 2018 on their improved progress by developing the YOLO [19] detection algorithm even further, “YOLOv3: An Incremental Improvement” [21].

As with a nearly every area of research especially in deep learning, huge amount of effort goes into trial and error. In improvement of YOLOv3, this effect of trial and error was in full force as the research team tried to a huge number of various ideas, many of them just failing straight out. Just a few of the ideas that stuck included a novel network to perform the process of feature extraction which contained of convolutional layers 53 to be exact, a novel metric for detection, having the ability to predict the score of object for each new bounding box using the well-known concept of logistic regression, and the loss metric to be used for class prediction as binary cross-entropy during training. In the end the novel approaches resulted in what we know today as YOLOv3 which runs significantly faster than other detection methods with generally the same performance. In addition the current version of YOLO no longer struggles with small objects present in an image.

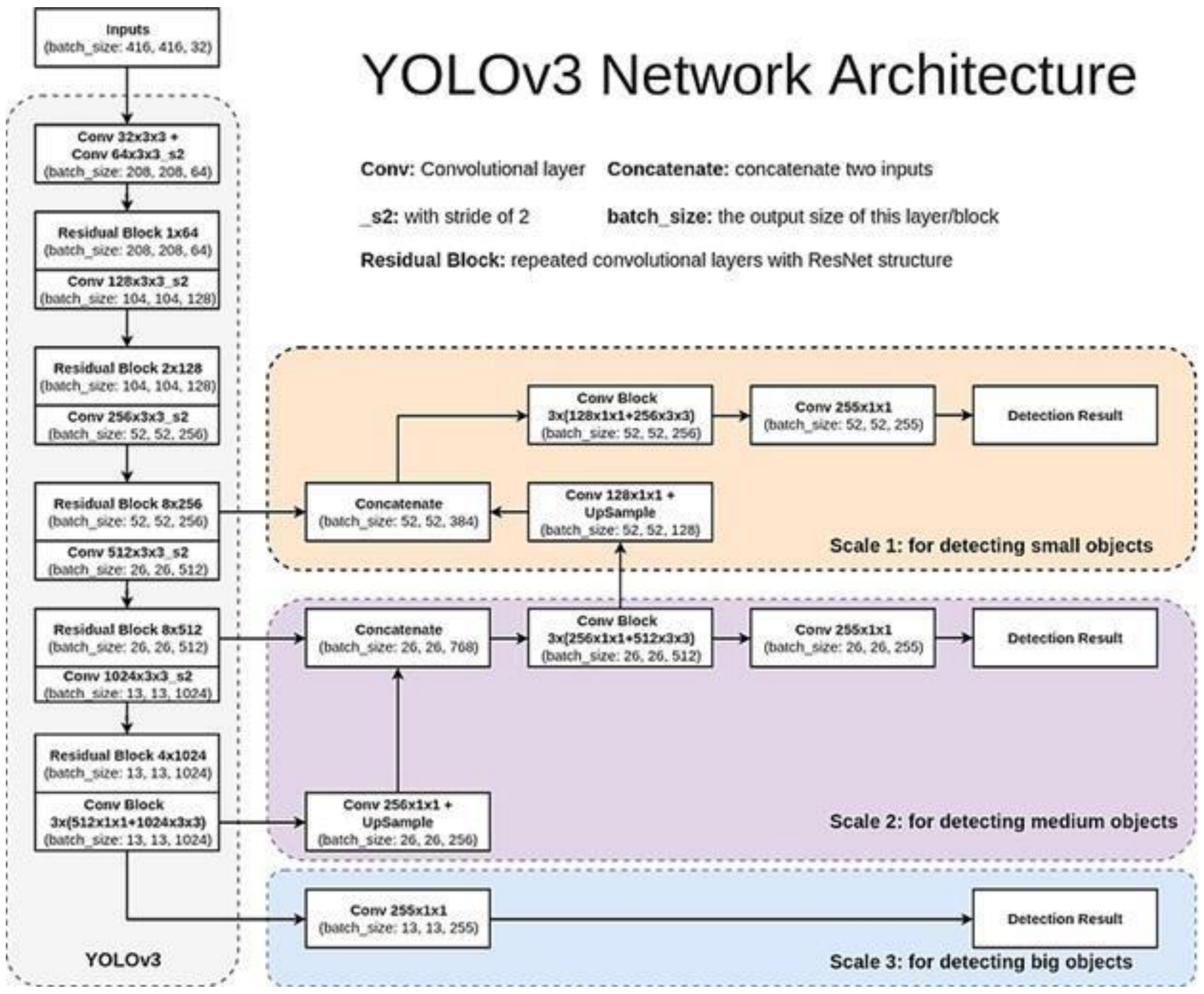

YOLOv3 Network Architecture
Conv: Convolutional layer
Concatenate: concatenate two inputs
_s2: with stride of 2
batch_size: the output size of this layer/block
Residual Block: repeated convolutional layers with ResNet structure
Inputs (batch_size: 416, 416, 32)
Conv 32x3x3 + Conv 64x3x3_s2 (batch_size: 208, 208, 64)
Residual Block 1x64 (batch_size: 208, 208, 64)
Conv 128x3x3_s2 (batch_size: 104, 104, 128)
Residual Block 2x128 (batch_size: 104, 104, 128)
Conv 256x3x3_s2 (batch_size: 52, 52, 256)
Residual Block 8x256 (batch_size: 52, 52, 256)
Conv 512x3x3_s2 (batch_size: 26, 26, 512)
Residual Block 8x512 (batch_size: 26, 26, 512)
Conv 1024x3x3_s2 (batch_size: 13, 13, 1024)
Residual Block 4x1024 (batch_size: 13, 13, 1024)
Conv Block 3x(512x1x1+1024x3x3) (batch_size: 13, 13, 1024)
YOLOv3
Conv Block 3x(128x1x1+256x3x3) (batch_size: 52, 52, 256)
Conv 255x1x1 (batch_size: 52, 52, 255)
Detection Result
Concatenate (batch_size: 52, 52, 384)
Conv 128x1x1 + UpSample (batch_size: 52, 52, 128)
Scale 1: for detecting small objects
Concatenate (batch_size: 26, 26, 768)
Conv Block 3x(256x1x1+512x3x3) (batch_size: 26, 26, 512)
Conv 255x1x1 (batch_size: 26, 26, 255)
Detection Result
Conv 256x1x1 + UpSample (batch_size: 26, 26, 256)
Scale 2: for detecting medium objects
Conv 255x1x1 (batch_size: 13, 13, 255)
Detection Result
Scale 3: for detecting big objects

Fig. YOLOv3 Architecture

Now the current version of YOLO that is YOLOv4 and in development of YOLOv5. Redmon reportedly stopped working on YOLOv4 after the possibility of possible misuse of this technology specifically for data collection and military uses resulting into serious ethical violation.

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” [10]. 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 [10].

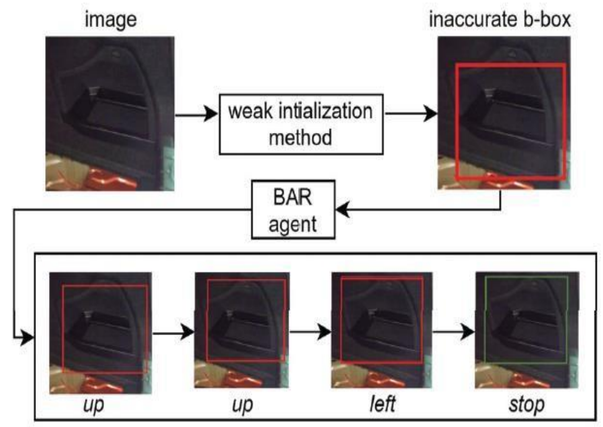


Fig. BAR agent workflow during the testing phase. Given an image and an inaccurate b-box enclosing the target object, BAR chooses the path *TE* with *T = {up,up,left}* [11]

Every image contains exactly one annotated target object whose b-box is represented by its upper-left corner ($xmin$, $ymin$) and its lower- right corner ($xmax$, $ymax$). 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:

$$Q_{t+1}(s, a) = (\alpha - 1)Q_t(s, a) + \alpha(r + \gamma \max_{a'} Q_t(s', a')) \text{ [10]}$$

The three main components of BAR-DRL are:

1. State: The state is composed of a feature vector $\in R^{1238}$ extracted using ResNet50 [11] pre-trained 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 : : $\Gamma = \frac{10-r}{10}$, *r*1 = 6, *c* = 4, *r*2 = 3 [10].

Problem Statement: Most of the literature tends to create bounding boxes, be it for single objects or be it for multiple objects. The approach in this paper can be extended to multiple objects. But what happens if there are overlapping objects in the image and the methodology fails to correctly identify and create the bounding boxes in the image. This sort of application can be huge for the industrial applications and the scenarios where it's bound to be a lot of overlapping objects. So to formally define my problem statement **"Bounding Box Refinement Agent for Overlapping Object Detection"**.

Methodology

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 [13].

For execution, multiple experiments involving images of “aeroplane” class was chosen from PASCALVOC 2007 [13]. Since the aim of my experiment is to reduce the manual effort of correcting the bounding boxes by the proposal produced by the reinforcement learning agent below will contain the table and list of all the experimental parameters that I created for my own testing. Wrong manual annotations where done on these 200 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 parameter for the metric used is Average IoU when testing on images.

The deep q 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 [10].