Object Detection with Deep Reinforcement Learning

Manoosh Samiei
Electrical and Computer Eng. Department
McGill University
manoosh.samiei@mail.mcgill.ca

Ruofeng Li
Electrical and Computer Eng. Department
McGill University
ruofeng.li@mail.mcgill.ca

ABSTRACT

Object localization has been a crucial task in computer vision field. Methods of localizing objects in an image have been proposed based on the features of the attended pixels. Recently researchers have proposed methods to formulate object localization as a dynamic decision process, which can be solved by a reinforcement learning approach. In this project, we implement a novel active object localization algorithm based on deep reinforcement learning. We compare two different action settings for this MDP: a hierarchical method and a dynamic method. We further perform some ablation studies on the performance of the models by investigating different hyperparameters and various architecture changes.

***K*eywords** object localization · neural networks · deep reinforcement learning · Markov decision process

1 Introduction

Object classification and object detection are two of the most important tasks in computer vision. An object classification algorithm identifies which objects are present in an image. While, an object detection (localization) algorithm not only indicates which objects are present in the image, but also outputs bounding boxes to indicate the location of the objects inside the image. Object detection has significant applications in real life. In self-driving vehicles, object detection is used to detect cars, pedestrians, traffic lights, road signs, etc. around the vehicle to help it decide its next actions. Object tracking is another application that is vastly used in traffic monitoring, activity recognition, ball tracking in sports, surveillance and security i.e. tracking a person in a video. Face and iris detection are used as powerful means of identity verification. Also, detection of organs is crucial in medical imaging to assist the clinicians in diagnosis, therapy planning and image-guided interventions, by detecting and analyzing the organs of a patient.

In this project we train a model to localize objects in images using a deep reinforcement learning approach. We formulate object detection problem as a Markov-Decision Process(MDP), and try to find the salient parts of an image that are more probable to contain a target object, and then zoom on them. This process is repeated until a tight bounding box is found around the target object.

We implement two different action settings for this MDP. In the first setting, a hierarchical method proposed by [1], the agent chooses to focus on one of the 5 sub-regions of the image (ie. top-left, top-right, bottom-left, bottom-right, center) at each time step. In the second setting, a dynamic method proposed by [2], the agent deforms a bounding box using simple transformation actions (horizontal moves, vertical moves, scale changes, and aspect ratio changes) at each step to find the specific location of an object in the image. These methods are class-specific and can find a specific class of object at a time. Both methods are able to localize objects within 10 steps (region proposals). As suggested by [2], attending smaller number of regions proposals is a superiority of deep reinforcement learning methods over R-CNN method which is a baseline model in object detection. We also attempt to enhance the hierarchial method, by changing the network's architecture, multiple setups, and hyperparameters. To train and evaluate the model, we use Pascal VOC2012 dataset.

1.1 Background

The first step in object detection task is to extract features from the input image, using convolutional neural networks or CNN. Convolutional layers reduce the dimensions of input data by performing a convolution operation on them, which makes them appropriate for handling image data. We use a pre-trained VGG16 network to extract the features from the input images as suggested by [1]. We use **deep q networks**, proposed by [3], instead of basic q-learning updates to estimate state-action values (q), since DQN fits better to large state spaces where simple q-learning algorithm fails to

perform efficiently. DQN is just an artificial neural network which estimates the optimal q function. DQN accepts state from a given environment as input and for each given state, the network outputs estimated q values for each action that can be taken from that sate. The objective of this network is to approximate the optimal q function that is derived from bellman equation 1.

$$q_*(s,a) = E[R_{t+1} + \gamma \max_{a'} q_*(s', a')] \tag{1}$$

The loss from the network is calculated by comparing the outputted Q values to the true optimal Q values. The objective of network is to minimize this loss. After the loss is calculated, the weights of the network are updates via stochastic gradient descent and back propagation. This process is done repeatedly for all states of the environment until the losses are sufficiently minimized and an approximate optimal q function is obtained. This process process is similar to one-step Q-learning update rule in 2.

$$w_{t+1} = w_t + \alpha[R_{t+1} + \gamma \max_{a} q(s_{t+1}, a, w_t) - q(s_t, a_t, w_t)]\nabla q(s_t, a_t, w_t) \tag{2}$$

Training a DQN needs a special setting called experience replay. With experience replay we store agent's experiences, defined as: $e_t = (current\ state, action, reward, next\ state)$, at each time step in replay memory. If the network only learns from consecutive samples of experiences as occurred sequentially in the environment, the samples would be highly correlated and lead to inefficient learning. Random samples from replay memory are then used to train DQN in order to break the correlation between consecutive samples. Using the estimated q values from deep q network, we can find an optimal policy for object region proposals to localize a specific category of objects in that image.

2 Method

2.1 Markov decision process formulation

We define object detection problem as a Markov decision process, in which our goal-oriented agent interacts with a visual environment that is our image. At each time step in training phase, the agent applies a transformation to a bounding box in image, a positive or negative reward is assigned to agent based on the new bounding box, q values are updated based on the the state, action and reward, and DQN is trained. During testing, the agent does not receive rewards and does not update the model either, it just follows the learned policy. We implemented two different formulations from two papers: a dynamic method proposed by [2] and a hierarchical method [1]. The MDP formulation of these two methods differ in their action setting and to some extent their state representation. We will describe the components of these MDPs as follows.

2.1.1 Actions for Hierarchical Method

Generally in both hierarchical and dynamic method, there are several **movement** actions and a **terminal** action. In hierarchical method, there are five movement actions which correspond to five sub-regions in the current bounding box: the four regions representing the four quadrants plus a central region. There is also a terminal action called trigger, which should be chosen when the agent reaches its goal of finding a right bounding box around the target object. Figure 1 can further clarify why this method is called hierarchical.

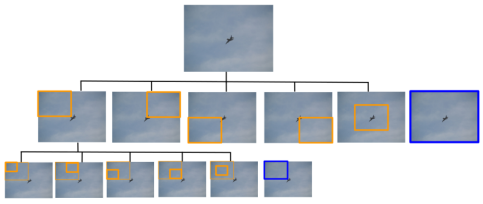

Figure 1: Illustration of hierarchical actions in MDP. Image credit [1]

2.1.2 Actions for Dynamic Method

In dynamic method, there are eight transformation actions that change the geometry and location of the bounding boxes, and one trigger action. The eight transformations can be divided into four categories, which modify the horizontal

level, vertical level, scale and aspect ratio of the bounding box. Figure 5 can further clarify these transformations. Any transformation makes a discrete change to the box by a relative factor (α) to its current size, using equations 3.

$$\alpha_w = \alpha(x_2 - x_1) \quad \alpha_h = \alpha(y_2 - y_1) \tag{3}$$

Figure 2: Illustration of dynamic actions in MDP. Image credit [2]

Where $\alpha \in [0, 1]$ and x_1, y_1, x_2, and y_2 are the coordinates of the top-left corner and the right-bottom corner of the current placed bounding box, respectively. Then according to the type of transformation action taken, α_w and α_h are added to or subtracted from the current bounding box coordinates.

2.1.3 States

The states of this MDP, are constructed of a feature vector which summarizes the image information of the current found bounding box, and a history vector of several previous taken actions. In both hierarchical and dynamic method, the feature vector is extracted by feeding the current bounded region of the image into a pre-trained VGG-16 network. Since the VGG-16 network requires for inputs with the same dimension, all the attended regions are resized to 224x224 dimension. In hierarchical method, the output of the last pooling layer which is 25088-dimensional is used as feature vector; while in dynamic method, the output of the first dense layer which is 4096-dimensional is used as feature vector. In hierarchical method, the action history vector *h* contains 4 past actions of agent in a one-hot encoded format, and as we have 6 actions, the binary one-hot encoded history vector has 24 elements. In dynamic method, the history vector *h* contains 10 past actions of agent, and as we have 9 actions, the history vector has 90 elements. By adding this history vector as part of the state representation, the agent is expected to avoid repetitive actions in its search paths.

2.1.4 Intersection-over-Union (IoU)

Before explaining the reward formulation of our MDP, we need to define a metric called Intersection-over-Union (IoU). We mainly exploit the Intersection over Union (IoU) between the predicted bounding box and the ground truth mask to evaluate the performance of our model. The IoU is defined as the ratio of the overlapped area between the predicted bounding box and the ground-truth bounding box over the area encompassed by both of them. More specifically, given the predicted bounding box b and the ground truth bounding box g, the IoU between b and g is expressed as the following formula:

$$IoU(b, g) = \frac{area(b \cap g)}{area(b \cup g)} \tag{4}$$

2.1.5 Reward function R

The reward function needs to be defined corresponding to the actions taken. In the proposed formulation, the reward function R is given based on the improvement of the Intersection-over-Union (IoU) between the predicted and the ground truth bounding box, during the transition between states. Thus, for non-terminal actions, the reward is set to 1 if the state transition improves the IoU and -1 otherwise, as suggested by below formula:

$$R_a(s, s') = sign(IoU(b', g) - IoU(b, g)) \tag{5}$$

For the terminal action (trigger), the reward scheme is different since the terminal action does not change the geometry of the bounding box. As a result, the reward is defined based on whether the current IoU exceeds the pre-set threshold (τ) or not. The terminal reward is set to 3 if the current IoU is larger than the threshold and -3 otherwise, as suggested by below formula:

$$R_w(s, s') = \begin{cases} +\eta \text{ if } IoU(b, g) \geq \tau \\ -\eta \text{ otherwise} \end{cases} \tag{6}$$

Where η was chosen to be 3 for both dynamic and hierarchical methods and τ (trigger threshold) was chosen to be 0.6 in dynamic method [2] and 0.5 for hierarchical method [1].

2.2 Model Architecture

The architecture of both hierarchical and dynamic models that we implemented can be seen in 3. In the original paper of dynamic method [2], the authors use a 5-layer pre-trained convolutional neural network instead of a VGG16 network to extract features; however, for better comparability of the two approaches we used a pretrained VGG16 for both models. VGG16 network is composed of 13 convolutional layers, 5 max pooling layers, and 3 dense layers. In hierarchical method, the output of the last max pooling layer is concatenated with 24-dimensional action history vector which forms our state representation. In dynamic method, the output of first dense layer in vgg16 is concatenated with a 90-dimensional action history vector and forms our state representation. This final array is then fed to a deep q network which is composed of 3 dense layers. The first two layers are composed of 1024 units and the last dense layer contains q-values for each of our actions, 6 units in hierarchical method and 9 units in dynamic method. It is worth mentioning that in DQN structure the output layer is not followed by any non-linear activation function since we need the raw non-transformed q values from the network. In our implementation, we used a linear activation function for the output layer.

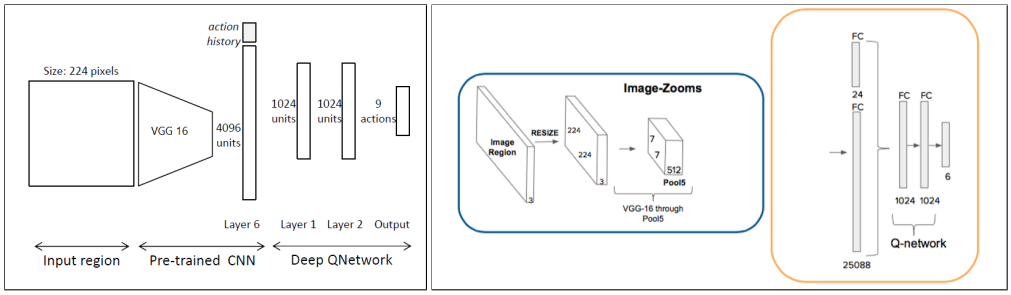

Figure 3: Architecture of dynamic model in the left [2] and hierarchical model in the right [1]

2.3 Dataset

The dataset applied in this project is PASCAL Visual Object Classes (VOC) 2012 dataset. This dataset is composed of images for twenty object categories, such as person, bird, cat, cow, dog, etc. There are around 12,000 images in this dataset, for which the ground truth bounding boxes for model training and testing are provided.

In this dataset, there are several text files for each object category, which provide the name of all images along with a number (1 or -1) to show whether each image contains an object from that category. This helps the users to train the model with specific classes. Since the model (in both dynamic and hierarchical method) is class-dependent, we trained the model with images containing objects from one class at each time.

3 Results and Discussions

3.1 Hierarchical Method

In [1] authors test another variation of hierarchical model, called pool45-Crops,that only extracts features for the whole image, and then reuses some regions of feature map instead of extracting new features for each predicted bounding box. However, they discuss in their paper that this change degrades the performance. Therefore we did not implemented this approach and we only focused on improving the better model called image-zooms by performing hyper-parameter search, and changing model structure and set-ups. Due to the limitation of the time, we conducted hyperparameter search experiments only on aeroplane class. We also evaluated the performance of our best model on seven other classes to examine the generalization of our method.

Also in all experiments, the initial value for exploration parameter, ϵ, is set to 0.9 (in dynamic method it is set to 1), and at the end of each epoch it is decreased by 0.1, until its value reaches 0.1 when it stays constant for the rest the epochs.

To evaluate the performance of different settings we use average IoU (Intersection over Union) over all testing images. For images with multiple objects from the target class, we measured the IoU between the prediction and all ground truth boxes and picked the maximum IoU.

In all experiments the agent is at least trained for 10 epochs, and the models of all epochs are stored separately and tested. The highest average IoU over these epochs is used for comparison.

The highest average IoU we obtain using the default settings, proposed by [1], is **0.3997**. In the following we investigate the effect of several changes in the original setting of the model.

3.1.1 Double Q Learning

As suggested by [4], one problem associated with deep q networks, or generally q learning algorithms, is the over estimation of Q values. In other words, as our Q-values move closer to their targets, the targets continue to move further because we are using the same network to calculate both of these values. To solve this issue, we implemented two separate network, one to estimate the target value at each step called target network and one to learn policy (q values) called policy network. The target network is a clone of the policy network. Its weights are frozen with the original policy network's weights, and we update the weights in the target network to the policy network's new weights every certain amount of time steps. We tested updating the weights of target network to policy network with three different number of time steps, i.e. every 5 steps, every 10 steps, and every 1 step. As it can be seen in table 1, updating weights of target network with bigger time intervals, leads to better results.

Table 1: Number of steps between updating target network weights

Updating weights interval	**Highest Average IoU**
every 10 step	0.4151
every 5 step	0.3899
every step	0.3882

3.1.2 Overlapping vs Non-overlapping Sub-regions

In the original setting of model, each sub-region is 3/4 of its ancestor bounding box, in both width and height. This setting leads all 5 subregions to have overlap with each other. We investigated a non-overalpping setting, in which each subregion's width and height is 1/2 of its ancestor box, and therefore there is no overlap between the subregions. Image 1 is showing a non-overlapping case of hierarchical method. To make this change in the code, we set a parameter called 'scale subregion' to 1/2. There is another parameter in the code, called 'scale mask', that controls the movements of sub-regions to 5 different locations in image, i.e. top-left, top-right, bottom-left, bottom-right, and center. The amount of movement of subregion, is determined by the multiplication of scale mask and scale subregion parameters. Therefore, in non-overlapping case, 'scale mask' should be equal to 1, so that by multiplying it with 1/2, we move each mask by 1/2 and cover the whole area of ancestor box. We tested three other modifications to these two parameters, that can be seen in table 2. When scale subregion is 1/2 and scale mask is 1/2 the masks move by 1/4 in ancestor box and therefore do not cover the whole surface of the box and are concentrated in the middle of the box. We also tested bigger subregions, each 4/5 of the previous box, both with full coverage of box and concentrated in the middle of box. Two results were obtained: first smaller windows (1/2) have better performance, second when windows are concentrated in the middle of the image they might have better performance due to most objects being presents relatively in the middle. In [1] authors suggest that overlapping windows perform better than non-overlapping case; while in our experiments we obtained the best result with the non-overlapping case.

Table 2: Changing the size and movement of sub-regions in ancestor box

Setting	**Scale subregion**	**Scale mask**	**Highest Average IoU**
Non-overlapping	$1/2$	1	**0.4599**
Smaller, Concentrated	$1/2$	$1/2$	0.4527
Default	$3/4$	$1/3$	0.3997
Bigger, Concentrated	$4/5$	$5/16$	0.3629
Bigger	$4/5$	$1/4$	0.3438

3.1.3 Reward Setting

To investigate the effect of different reward settings, we modified the original reward set up in two ways. One problem that we observed was that for some images the agent fails to choose trigger action on time; it continues making window smaller up to the maximum number of steps allowed in testing. Thus we decided to encourage agent to choose trigger action earlier, by increasing the value of reward assigned for the trigger action (η in formula 6). The results can be seen in table 3.

Table 3: Changing the value of trigger reward

Trigger action reward (η)	**Highest average IoU**
3 (default)	0.3997
6	0.2081
10	0.2252

In the second setting, we used another definition called recall instead of IoU for reward measurement. The formula for the recall is:

$$Recall(b, g) = \frac{area(b \cap g)}{area(g)} \tag{7}$$

where, compared to IoU formula, the denominator is changed from the union of the predicted bounding box (b) and the ground truth (g) to just the area of the ground truth box (g). We experimented with three different values for recall threshold of choosing trigger action (τ). Although we are training agent to maximize its recall, we measured its average IoU to be able to compare its performance with other settings. The highest average IoU up to 10 epochs corresponding to each experiment is shown in table 4.

Table 4: Changing the threshold of trigger recall

Recall threshold	**Average IoU**
0.5	0.3916
0.6	0.3793
0.7	0.3786

From the results, we observe that the default setting with IoU as the evaluation method and trigger reward (η) equal to 3 performs the best. With higher trigger reward, the agent tends to trigger at the very early steps, thus resulting in a relatively low IoU. By changing the reward evaluation method from IoU to recall, the agent tends to search for actions that maximize the recall, i.e. it tends to predict larger windows instead of an accurate one around the object, giving a lower IoU in the end.

3.2 Fixing the target object during training

In VOC2012 dataset, some images contain more than one object of the same class. In the original code published by the authors [1], for multiple-target images, the agent changes the target during training based on the location of the current predicted box. In other words, the agent tries to find the object, with which it currently has higher IoU, and if it is searching for another object further away from its current bounding box in image, it will change its target object to a closer one. We tested the agent performance while fixing its target object, to investigate the effect of this behavior in agent's training. As a result of this change the agent's highest average IoU was increased to **0.4217**, an increase of around 0.02 compared to its original setting. We believe that changing target object can cause confusion for agent during training, and consequently fixing the target object improves its performance.

3.2.1 Training and test steps

We tested two other values: 7 and 15 for the number of steps per epoch during training and testing time to investigate its effect on performance. We realized that the default setting with 10 number of steps per epoch works best during training. However during testing, decreasing the number of steps, leads to higher performance for some experiments. We argue that limiting the number of steps during test prevents the agent from making the predicted bounding box smaller, keeping a higher IoU (intersection) with the ground truth box. The results for the number of training steps can be seen in table 5.

Table 5: Changing the number of training steps per epoch

Training Steps per epoch	**Average IoU**
10 (default)	0.3997
15	0.3340
7	0.3497

3.2.2 Discount rate γ

We tested three different values for the discount rate (γ). The results of these experiments are shown in table 6.

Table 6: Changing the value of discount rate γ

Value of γ	**Average IoU**
0.1	0.2785
0.5	0.3044
0.9 (default)	0.3997

From the results we observe that with higher γ, the average IoU is higher. With a higher γ, the agent is more far-sighted, meaning that it gives great importance to the future rewards and acts in a way to maximize its long-term reward. Therefore, it chooses regions that in the end lead to higher intersection with the target object in image. While with a smaller γ, the agent tries to maximize its short-term reward and continues searching for the object, which causes the predicted window to become smaller and smaller, leading to a lower IoU in the end.

3.2.3 Batch size

During training, at each time step, we extract a fixed number of random experiences, called batch size, from replay memory to train our deep q network. We tested three different values for batch size, to investigate its effect on training. The results are shown in table 7.

Table 7: Changing the number of samples from replay memory

batch size	**Average IoU**
50	0.4041
100 (default)	0.3997
200	0.3170

From the results, we observe that with higher random training samples, the performance degrades. This sounds surprising as we expected that with larger training samples at each step, DQN is trained better. One probable reason might be the higher probability of extracting correlated samples from replay memory. In these experiments, the size of replay memory was 1000 samples, and when extracting higher number of samples from memory (200 vs. 50), it is more probable to extract correlated samples, which as discussed earlier leads to inefficient training.

3.2.4 Trigger threshold

We modified the IoU threshold (τ) for choosing the trigger action illustrated in formula 6. The results are shown in table 8.

Table 8: Changing the value of trigger threshold

Trigger threshold	**Average IoU**
0.5 (default)	0.3997
0.6	0.3585
0.7	0.3835

From the results we observe that increasing the threshold of trigger action decreases the performance of the model. This could be due to that it is hard for the agent to find IoU scores of 0.6 and above, during training, and thus many times choosing trigger action leads to negative rewards, according to formula 6. Consequently, choosing trigger action is discouraged in training and in the test phase, the agent fails to choose the trigger action at the right time, causing overly compressed predicted windows.

3.2.5 Size of action history vector

The action history vector, as introduced in the Method section, is used to prevent the agent from repeating its previous actions during the search. We tested multiple values for the size of action history vector, in the setting of not changing the target object. In particular, we considered the state representation without action history, with one action history,

and four action history. The results for these experiments are shown in table 9. surprisingly, the case without any action history works the best. We argue that the action history vector in the state representation may not be well-understood by the agent. Action history vector contains only binary values and is concatenated to the output of vgg16, which contains numbers in a great range. Probably, the agent cannot interpret the action history part of the state representation.

Table 9: Changing the size of action history vector

Size of action history vector	Average IoU
without history	0.4385
1 action history	0.4245
4 action history (default)	0.4217
6 action history	0.4053

3.2.6 Adding layer to DQN

We also added one extra dense layer to our deep q network (i.e. 4 layer DQN) in double learning mode. The model performance was degraded with its best average IoU being **0.3524**.

3.2.7 Best-performance hierarchical model

Among all experiments we performed, the non-overlapping windows had the highest average IoU (0.4599). However, when we looked at the images of different settings, we observed more reasonable predicted bounding boxes with the **fixed target object** setting. Sometimes, choosing trigger action in early steps maximizes the IoU while does not predict the location of object accurately. Due to time limitation, we could not test the combination of both of these methods, i.e. non-overlap + fixed target object, and we suggest this to be considered as a future work.

3.2.8 Other object categories

We tested the performance of the model, with the setting of fixing the target object, on 7 other object classes. The results can be seen in table 10.

Table 10: Different Object Categories

Object class	Average IoU
aeroplane	0.4217
bicycle	0.4225
car	0.2762
cat	0.5172
bird	0.2628
bus	0.4437
dog	0.4518
horse	0.4149

3.3 Dynamic Method

As an extra work we also implemented another object detection model proposed by [2], and tested its performance on 8 categories. However due to time limitation, we could not perform a broad hyper-parameter study on this approach. We trained the dynamic model on 8 object classes of VOC2012 dataset, for 7 epochs (each epoch 20 steps) and tested the model (with maximum 10 steps for each image). The results can be seen in table 11.

4 Conclusions

In this project, we implemented two algorithms of object detection with deep reinforcement learning: a hierarchical model and a dynamic model. By adjusting hyperparameters and some setups, we achieved a hierarchical model with a reasonable performance in object detection. We also implemented a dynamic model, which has more freedom in its action setting (shape of bounding boxes) compared to hierarchical model.

Table 11: Different Object Categories

Object class	Average IoU
aeroplane	0.3723
bicycle	0.1990
car	0.1952
cat	0.4115
bird	0.2059
bus	0.3858
dog	0.2828
horse	0.3375

Comparing the two models, hierarchical method trains and performs faster than dynamic method, as it has fewer number of actions (6 vs. 9). It also fits the smaller objects better than dynamic method, since at each step the size of predicted box becomes smaller. However, making windows smaller at each time step is a weakness of this method, especially for bigger targets. Indeed, hierarchical model cannot recover from a bad region choice in the first step. It stays in the same region and choose its descendent sub-regions at each step. In Dynamic method on the other hand, the predicted box is able to move freely in image and is not bounded to any predefined region. The window is also able to become bigger at any step. One other weakness of hierarchical method is that the predicted box should always keep the shape of the original image, as all predefined sub-regions are a ratio of original image. In fact, if the image is vertical, the predicted boxes are all vertical, which may not fit the target object shape in image. This problem can be clearly seen in images included in the appendix.

In terms of better accuracy, we are not able to make a firm statement that which model is more accurate, as there are some differences in the default settings of the two model and due to time limitation we could not build a fair comparison setting to compare the two methods.

Both methods are dependent on the hyperparameters and variables chosen. As a result, to achieve good performance with these reinforcement learning object detection models, a careful design of hyperparameters should be considered. Besides, we are surprised to find that some of the techniques proposed by the papers, such as the action history vector included in the state representation, do not contribute to the performance of model in our case.

We also observed a peculiarity in hierarchical method training, that the best models are usually obtained in the first 5 to 7 epochs, and then the performance was decreased. This has conflict with the authors claim that they trained the model for 50 epochs. We suspect that our model is possibly under-trained and need to be trained on more data. In the original papers [2] and [1], the agent is trained on both VOC2012 and VOC2007 dataset; but due to limited computational resources, we only used VOC2012. Hence, we would like to combine the two datasets and retrain our models on them in future.

The algorithms we implemented in this project are the first algorithms that attempt to treat the object detection problem as a dynamic decision process. The idea of formulating the process of locating the bounding box as a Markov decision process (MDP) is a breakthrough, which is quite different from the conventional neural network approaches applied in the computer vision field. As a result, even though the accuracy of our models cannot reach that of the current state-of-art algorithms on this task, it is still worthwhile to look into and understand these models to examine the potential of reinforcement learning in the future of computer vision.

Moreover, as we discussed in the method section, these models are currently class-dependent, which means that the agent can only search for one class of objects at a time. As a future direction, it would be meaningful to extend these models to be class-independent so that it can be trained with arbitrary images and find object of all classes at the same time. Furthermore, the current algorithms are not robust in finding multiple objects in image at the same time; this also can be addressed as a future work.

5 Links to code base and video

We included the notebooks of our experiments, our trained models, and other contents of reproducibility checklist in Mega cloud storage, which is accessible through this link: `https://mega.nz/folder/7UMRmKAQ#PiPJ6RquyeWX48G3R0Eo-Q`

Our spotlight video is also available at: `https://youtu.be/dcGP9mDnFf0`

6 Statement of Contribution

Both Ruofeng and Manoosh worked on hyperparameter search, improving the models, interpreting the results, and writing the report. Manoosh also worked on coding and visualizing the results. Ruofeng also worked on preparing slides and spotlight video.

References

[1] Miriam Bellver, Xavier Giró, Ferran Marqués, and Jordi Torres. Hierarchical object detection with deep reinforcement learning. *ArXiv*, abs/1611.03718, 2016.

[2] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 2488–2496, 2015.

[3] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. 2013. cite arxiv:1312.5602Comment: NIPS Deep Learning Workshop 2013.

[4] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. *CoRR*, abs/1509.06461, 2015.

7 Appendix

Figure 4: Some good predictions by hierarchial model, all of the predictions are obtained by a model with fixed target object setting. The blue bounding box is a sample ground truth box for a target object, the red bounding boxes are the search path and the bold red bounding box is the final predicted box.

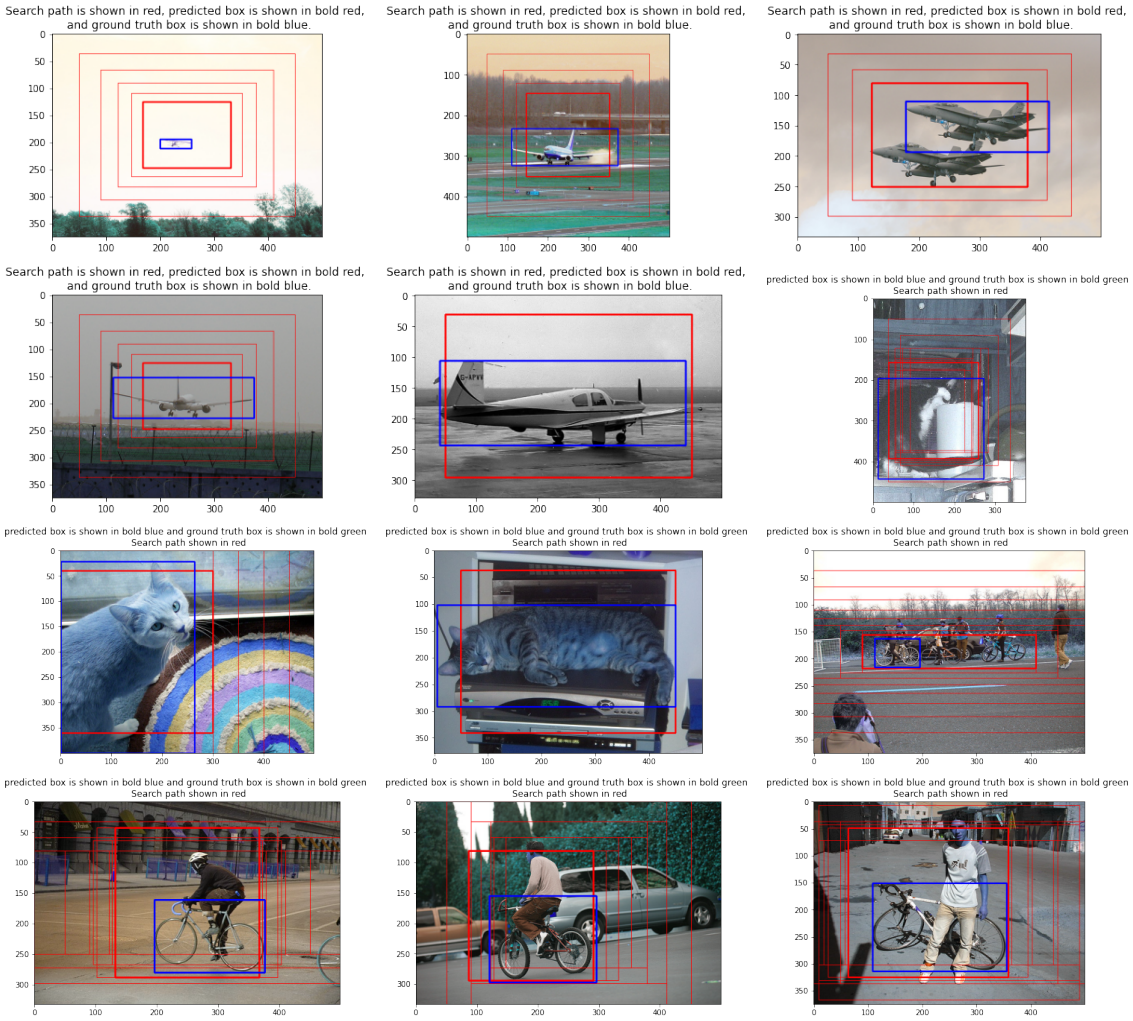

Figure 5: Some good predictions by dynamic model. The blue bounding box is a sample ground truth box for a target object, the red bounding boxes are the search path and the bold red bounding box is the final predicted box.