Artificial Intelligence Final Project: Object Localization with Deep Reinforcement Learning

Ruofeng Li
Electrical and Computer Engineering
McGill University
Montreal, Canada
ruofeng.li@mail.mcgill.ca

Manoosh Samiei
Electrical and Computer Engineering
McGill University
Montreal, Canada
manoosh.samiei@mail.mcgill.ca

***Abstract*—Object localization has been a common task in computer vision field. Methods of localizing objects in an image have been proposed based on the features of the attended pixels. As the computational resources developed, people started to use powerful computing techniques such as convolutional neural networks in this task. Moreover, it has been lately proposed that the process of localizing objects can be formulated as a dynamic decision process, and be solved by a reinforcement learning approach. In this project, we implement a novel active object localization algorithm based on deep reinforcement learning. We further investigate the performance of the model by conducting experiments on hyperparameter setting and network structure. In the end, we compare the performance of the algorithm with a random baseline, and propose some drawbacks of the model to be addressed in future work.**

***Index Terms*—object localization, neural networks, deep reinforcement learning, Markov decision process**

I. Introduction

There are two common tasks in computer vision: object recognition (classification) and object detection (localization). an object recognition (classification) algorithm identifies which objects are present in an image. It takes the entire image as an input and outputs class labels and class probabilities of objects present in that image. An object detection (localization) algorithm not only indicates which objects are present in the image, it also outputs bounding boxes $[x_{min}, y_{min}, x_{max}, y_{max}]$ to indicate the location of the objects inside the image. One approach to localize objects is to slide a box or window over an image to select a patch and classify each image patch covered by the window using the object classification model. But this method is an exhaustive search and is computationally expensive. Hence many state-of-the-art methods for object localization introduce region proposals, which indicate the regions in an image that are likely to contain an object. In this project we implement and enhance an active object detection model for localizing objects in scenes using deep reinforcement learning. This model is proposed by [1] and is class-specific, which allows an agent to focus on proposed regions found by following a reinforcement learning policy for identifying the correct location of a target object from a specific class; such as cat, dog, car, aeroplane etc. The agent uses a set of actions to deform a bounding box in the image to find the specific location of the target object. The agent is trained using a deep Q-network, and is trained on VOC2012 dataset. In the original paper [1] the agent is trained on both VOC2012 and VOC2007 dataset; but due to limited computational resources, we only trained and tested our agent on images from VOC2012 dataset. As the authors of the paper suggest, the implemented model is able to localize a single instance of an object after analyzing between 11 and 25 regions in an image.

This project is an extension to the reinforcement learning concepts discussed in class.

A. Background Knowledge

The first step in object detection task is to extract features from the input image, using convolutional neural networks. **Convolutional neural network or CNN** is Similar to fully connected neural network which was discussed in class, in that it also has learnable weights and biases, and its neurons receive some input, perform a dot product and follows it up with a non-linear activation function such as ReLU (Rectified Linear Unit). But the main drawback of fully connected layers while working with images is their huge number of tuning parameters (weights). For instance for a 224x224 RGB image we have 224x224x3= 150528 weights in the first hidden layer of a fully connected network. On the other hand, convolutional neural networks reduce the dimensions of input images by performing a convolution operation on them. In other words, it performs a convolution operation with a small part of the input matrix having same dimension. The sum of the products of the corresponding elements is the output of this layer. In classification we then use these features along with image labels (object categories) to train a network in a supervised manner.

These features along with the history of previous actions will be then used as our state representation in our reinforcement learning problem formulation, which We will elaborate on in later sections. In this formulation we use **deep q**

networks, proposed by [4], 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. The active combining of deep neural network with q-learning is called deep q-learning. In fact, DQN is just an artificial neural network which estimates the optimal q function (weight*feature vector). DQN accepts state from a given environment as input and for each given state input 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. Indeed, instead of using value iteration to solve the problem here we use deep neural network.

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. One difference in DQN structure to other neural networks is that 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.

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

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

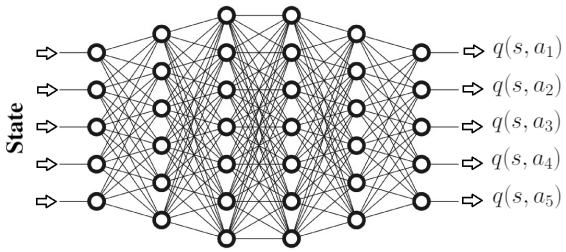

Fig. 1. Deep Q-network Architecture

Training a DQN needs a special setting called experience replay. With experience replay we store agent's experiences at each time step in a dataset called the replay memory. At time t the agent's experience e_t is defined as: e_t = (current state, action, reward, next state). In practice, the replay memory is set to some finite size limit N, and therefore it only stores the last N experiences. The act of sampling from the replay memory that stores these experiences is called experience replay. The reason of using replay memory, is to break the correlation between consecutive samples. If the network only learns from consecutive samples of experiences as they occurred sequentially in the environment, the samples would be highly correlated and therefore lead to inefficient learning. Using the estimated q values from deep q network, we can find an optimal policy for object region proposals to localize an specific category of objects in that image.

II. Related Work

A. R-CNN (Region-based cnns)

R-CNN model proposed in [2], first finds some region proposals using selective search algorithm. This algorithm proposes some regions by computing hierarchical grouping of similar regions based on color, texture, size and shape compatibility. In the next step, the model creates a feature vector representing each proposed region of image, in a much smaller dimension using a pre-trained Convolutional Neural Network. A pretrained CNN is previously trained on a big dataset such as ImageNet and its weights are tuned accordingly. To feed the image to the network we should resize them to match the CNN input. Then after extracting features of region proposals, we use an SVM classifier to classify the object in that region. We have one SVM for each object class, which means that for one feature vector we have n outputs, where n is the number of different objects we want to detect. The output is a confidence score, which means how confident we are that this particular feature vector represents this class. One drawback of RCNN and our implemented approach is that we need to use a pretrained cnn for feature extraction, as we cannot train both SVM (or in our case dqn) and cnn classifiers jointly.

B. Hierarchical object detection with Deep RL

In [3] authors train an agent that given an image window, is capable of deciding where to focus the attention among five different predefined region candidates (smaller windows). Then they repeat this procedure for the smaller regions until a tight box is found around the object. This iterative process provides a hierarchical image analysis. This approach is similar to our implemented model in this project except that their solution has a different action formulation by dividing the image into five sub-regions, which the agent need to choose from in each step. In our implemented model however, we have 9 actions to deform a window across the image with out dividing the image into sub-regions. We will further elaborate on our problem formulation in section III.

III. Method

A. Markov decision process formulation

The object localization problem can be viewed as a dynamic control process to transform the geometry and location of the bounding box with a sequence of steps, which can then be formulated as a dynamic Markov decision process (MDP). In this formulation, a single image will be viewed as the environment, and the states *S* are constructed based on the features of the current bounded pixels and a series of previous actions in history vector. The actions *A* are composed of the geometric transformations of the bounding box, and the

reward function R is defined based on the effects of these transformation actions. We will further clarify each component of MDP formulation as follows.

1) States S: It is expected that the state space of this problem formulation should be very large due to the complexity of dealing with the image data. As a result, generalization with feature extraction is necessary to represent the states efficiently and effectively. In this project, the states are constructed with a feature vector o, which summarizes the image information of the current found bounding box, and a history vector h of previous taken actions. The feature vector o is extracted by feeding the current bounded region of the image into the pre-trained VGG-16 network. Since the VGG-16 network requires for inputs with the same dimension, all the attended regions are transformed into a 224x224 image with the OpenCV library. The network then deals with the pixel values of the attended region and extracts corresponding features out of it. The output feature vector is set to be a 4096-dimensional vector. The action history vector h is a binary vector that encodes the actions applied in the past with one-hot encoding method. This vector stores 10 previous actions taken by the agent, and as we have 9 possible actions, the one-hot encoded history vector will have 90 dimensions. By adding this history vector as part of the state representation, the agent is expected to be informed about what has happened in the past and avoid getting stuck in repetitive search cycles.

2) Actions A: As stated above, the actions of this MDP are mainly the geometric transformations of the bounding boxes. There are nine transformation actions and one terminal action in this MDP formulation . The eight transformation actions can be divided into four categories, which modify the horizontal level, vertical level, scale and aspect ratio of the bounding box, respectively. A figure of all actions can be seen in figure 2. The transformation actions are executed by calculating a discrete change to the box by a relative factor α, using equations 3.

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

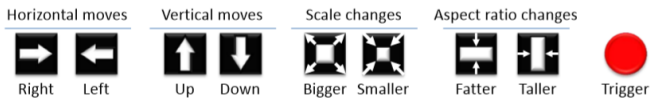

Fig. 2. Illustration of the actions in MDP. Image credit [1]

Where 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. With this calculated discrete change, the action can be carried out by adding this value to or subtract this value from the corresponding coordinates.

3) 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 detecting window 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:

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

4) 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 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 be 3 if the current IoU is larger than the threshold and -3 otherwise, as the formula below:

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

Where η was chosen to be 3 and τ (trigger threshold) was chosen to be 0.6, as proposed by the original paper [1].

B. Model and implementation

The architecture we used for our DQN model can be seen in figure 3. This model is similar to the model proposed by authors in [1]. However, to extract features from each region, instead of using a 5-layer convolutional neural network, we use a vgg16 network pretrained on imagenet dataset (inspired by [3]). To feed each region to vgg16, we should first resize it to 224x224 dimensions. Vgg16 network is composed of 13 convolutional layers, 5 max pooling layers, and 3 dense layers. We do not use the last dense layer as it compresses 4096 units of the previous dense layers to 1000 units and get rid of some useful feature information. Then, as previously mentioned, we concatenate these 4096 unit features with a 90 dimensional history vector of actions which contain one-hot encoded version of 10 past actions (we have 9 actions therefore: 9x10 =90 dimensions). This final array forms our state representation, and is fed to our 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 9 units and outputs the q-values for each of the 9 actions.

Also to calculate the loss of our DQN, we used smooth L1 loss. Smooth L1-loss can be interpreted as a combination of L1-loss and L2-loss. It behaves as L1-loss when the absolute

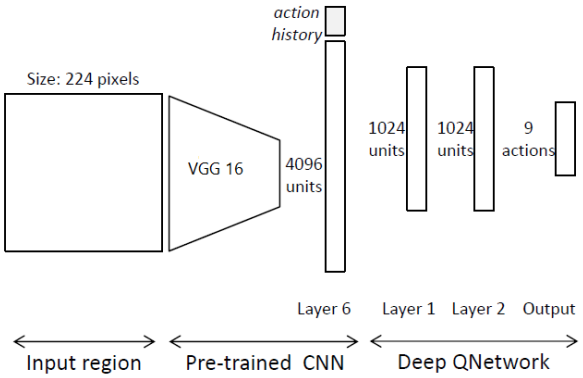

Fig. 3. Architecture of our Qnetwork. Image credit [1]

value of the argument is high, and it behaves like L2-loss when the absolute value of the argument is close to zero.

C. Dataset

The dataset applied in this project is PASCAL Visual Object Classes (VOC) 2012 dataset. It is a widely recognised dataset in image processing field. The dataset includes four different image sets: classification/detection image sets, segmentation image sets, action classification sets and person layout taster image sets. In this project, the experiments are conducted on images from the classification/detection image sets. For each class, the detection task is to predict a bounding box around each object of that class in a test image (if any). Ground truths are provided in classification/detection image sets, giving the correct bounding boxes around all target objects in images. It is also possible to have multiple objects from the same class in an image. This image set is composed of twenty object categories: person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dinning table, potted plant, sofa, tv/monitor. Figure 4 shows some images from all 20 classes of dataset with their (ground truth) bounding boxes.

In the dataset, there are several text files for each object category, which provides the name of all images along with a number (1 or -1) to show whether each image contains any object from that category. This helps the users to train the model with specific classes. In our project, since the model is class-dependent, we trained the model with images containing objects from one class at each time. Due to the limitation of the time budget, we conducted hyperparameter search experiments only on aeroplane class. We also evaluated the performance of our best model on three other classes to examine the generalization of our method, which we will describe in section IV. For each class, there are hundreds of images containing objects from that class in dataset. However, due to limited computational resources, to train our model for each class, we only use 30 images from that class.

Fig. 4. Images from 20 classes of dataset along with their ground truth bounding boxes.

D. Hyper-parameter tuning

In this section, we attempt to investigate the effect of multiple hyper-parameters on our model performance, including the size and setting of experience replay, IoU threshold for choosing trigger action, the size of action history, and the value of discount-rate for future rewards. These studies are introduced separately as follows.

1) Changing experience replay setting: As mentioned previously, we use experience replay to train our DQN to break the correlation between consecutive samples and prevent the network from being biased. The experience replay technique stores information of previous visited events in experience replay memory and randomly selects some samples from it to train the model. Given the mechanism of experience replay, it can be realised the number of samples taken from the experience memory can affect the overall performance of the model. We tested two different settings for our experience replay. First we changed the number of random samples chosen from a 1000-sample replay memory (the size of replay memory was kept constant). In this setting we tested two different sizes: 10, and 20, for our sample experience vector to be trained on DQN. In the other setting, we also changed the size of our replay memory to the size of our samples and removed randomization of samples. In this setting we tested 1, 10, and 20 consecutive samples instead of random samples to train our DQN. The results can be seen in table IV.

TABLE I
CHANGING EXPERIENCE REPLAY SETTING

experience replay setting	Average IoU
20 random samples	0.3325
10 random samples	0.3104
20 consecutive samples	0.1642
10 consecutive samples	0.1815
1 consecutive samples	0.0667

We realized that consecutive samples regardless of their size cause inefficient learning of DQN. In all of the consecutive training settings, the agent's IoU was lower than a random agent (IoU = 0.2), which always explores and never acts greedy wrt. q value. The number of random samples have a positive effect on agent's performance. The more random sample we have to train the network, the better the we can minimize the error between estimated q and true q value, unless we overfit the network to training data. A bigger experience vector can also lead to slower training. So we should be aware of the trade off here. We selected 20 random samples as a desirable value for our random experience sample size.

2) Changing the iou threshold for choosing trigger action: When the IoU of the predicted and ground truth bounding box is more than a specific threshold, denoted by τ, the agent chooses the trigger action to end the training on that image. Many object detection approaches choose $\tau = 0.5$ for their trigger action, but in the implemented paper [1] authors claim that $\tau = 0.6$ had a better performance for their method. The results for both $\tau = 0.5, 0.6$ can be seen in table II

TABLE II
CHANGING TRIGGER THRESHOLD

trigger threshold	Average IoU
τ = 0.6	0.3325
τ = 0.5	0.2739

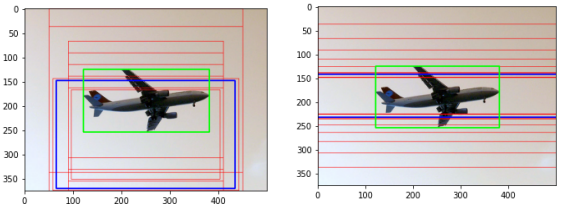

Fig. 5. The predicted box is shown in blue, ground truth box is shown in green, and search path is shown in red. Left picture is for the original model with $\tau = 0.6$, and right picture is for $\tau = 0.5$. The network is trained on 30 images for 50 epochs (each epoch with maximum 20 steps) and is tested on 100 images.

For some examples such as figure 5 above, $\tau = 0.5$ seems to work better. But in general the average IoU with $\tau = 0.6$ is higher.

3) Changing the size of action history vector: In the previous section, it was mentioned that one of the components of the state representation is the history action vector, which contains the past taken actions as additional information for the agent to avoid the repetitive search cycles. It is reasonable to consider the number of history actions stored in this vector to be a factor influencing the performance of the model, since it affects how far the agent can see through the past events. We conducted several experiments on the effect of the number of previous actions stored in history vector. We tested four values: 1, 5, 10 and 20, for the size of action history vector.

TABLE III
CHANGING THE SIZE OF HISTORY VECTOR

Size of history vector	Average IoU
1	0.4578
5	0.4191
10	0.3325
20	0.2640

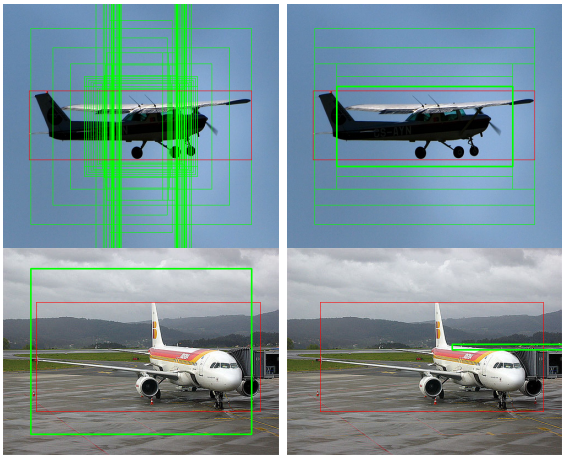

Fig. 6. The predicted box shown in thick green line, ground truth box shown in red, and search path shown in thin green line. Left two pictures are for history size = 1, right two pictures are for history = 20. The network is trained on 30 images for 50 epochs (each epoch with maximum 20 steps) and is tested on 100 images.

From the results, we can see that when the size of history vector is equal to 1, the model achieves better average IoU. This observation differs from what is stated in the original paper [1]. When we visualized the results, we realized that each size of history vector fits better to a set of images, while fails to perform well for other images. Particularly, we found that with large history vector, the agent tends to over-compress the window, while with small history vector, although the agent does not over-compress the window, it is likely to get stuck in the search process. The reason why the agent behaves like this could be due to the limited number of training samples we used. Clearly, neural networks are sensitive to the data volume, and as we decrease the number of training samples, it is possible that DQN fails to reduce the error between the actual and estimated q values sufficiently. Hence, the extracted policy that we follow during testing, by acting greedy with respect to q value, will be inaccurate.

4) Changing the value of γ (discount rate): One of the most important parameters in reinforcement learning problem set-up is discount rate γ, which indicates how important the future rewards are to the value of current state. With a higher γ, the agent tends to consider future rewards an important factor in earning high return, while with a lower γ, the agent focuses more on the immediate rewards. In order to find out how γ influences the performance of the model, we conducted experiment with γ = 0.1, 0.5 and 0.9.

From the results we can see that when γ = 0.1, the model

TABLE IV
CHANGING DISCOUNT RATE

Discount rate	Average IoU
0.1	0.3325
0.5	0.2321
0.9	0.2080

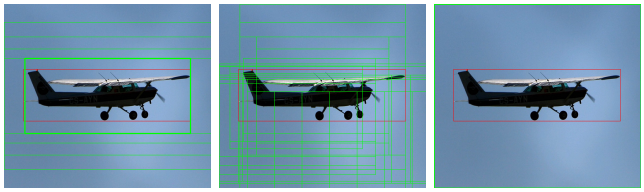

Fig. 7. The predicted box shown in thick green line, ground truth box shown in red, and search path shown in thin green line. Left picture is for γ = 0.1, center picture is for γ = 0.5, and right picture is for γ = 0.9. The network is trained on 30 images for 50 epochs (each epoch with maximum 20 steps) and is tested on 100 images.

achieves the best performance. We realized when γ = 0.5, the agent tends to get stuck in the search path, which hinders the agent from localizing the object efficiently. When γ = 0.9, the agent always chooses the trigger action at the very beginning of the search path, resulting in a low prediction accuracy. The reason of these behaviors could be due to the different extent of contribution of the trigger reward to the current state. When γ = 0.9, the trigger action will be contribute with a large value to the current state since it has 3 times the reward gained by a transformation action. As a result, the agent tends to always choose the trigger action. In the 0.5 case, the value of a trigger action would be approximately similar to a transformation action, so the agent might get stuck.

E. Changing model structure

As previously stated, the architecture for dqn proposed in the original paper [1] is composed of 3 dense layers with 1024,1024,9 units from top to bottom. We added one more dense layer with 1024 units before the last layer to investigate any possible improvement to their model. We realized that although adding one more layer slows down the training time of the agent, it leads to higher average IoU score and an improved accuracy in predicting bounding boxes. The results of this change can be seen in table V and figure 8.

TABLE V
CHANGING DQN STRUCTURE

DQN arcitecture	Average IoU
three dense layers	0.3325
four dense layers	0.4353

IV. RESULTS

A. Model performance

In general, the model proposed by [1] has not been able to exceed the baseline of R-CNN model, which is a pioneer

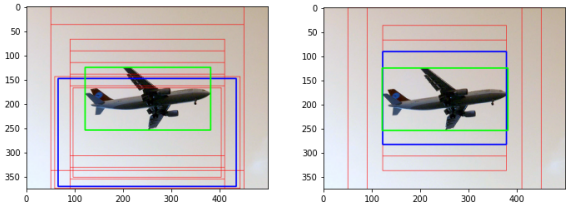

Fig. 8. The predicted box shown in blue, ground truth box shown in green, and search path shown in red. Left picture is for a dqn with 3 dense layers, and right picture is for a dqn with 4 dense layers. The network is trained on 30 images for 50 epochs (each epoch with maximum 20 steps) and is tested on 100 images.

solution to object localization (described in section II). However, this method has a benefit over R-CNN and that is its number of object proposals before finding the right bounding box. Based on authors claim in [1], the model we implemented finds the bounding box around a target object between 11 to 25 steps, while R-CNN method proposes much more number of proposals before finding a (possibly more accurate) bounding box. Hence, our model performs faster than R-CNN, as a trade-off for less accurate boxes. In our implementation, we were also able to detect objects in test images within 30 steps, however the predicted boxes were less accurate compared to the original model in paper [1], which was due to lack of sufficient training of our DQN.
We also tested our model on 3 more object categories in addition to aeroplane class, namely: boat, bird, and bicycle. The results can be seen in table VI.

TABLE VI
DIFFERENT OBJECT CATEGORIES

Class	Average IoU
boat	0.0919
bird	0.2665
bicycle	0.1039
aeroplane	0.3225

B. Comparison to random agent baseline

We implemented an agent that always explores and choose a random action at all time steps. It never acts greedy with respect to q values. This agent can form a baseline for our predictions. The average IoU on 100 test images for this agent is equal to 0.0.2098 although this number might vary for different runs but can indicate that an IoU of 0.2 is achievable by a random agent, hence if the average IoU of a trained model for a class is less than this value, the model is useless. Two examples of accidental good detections by a random agent can be seen in figure 9. We previously mentioned that due to lack of sufficient computational resources, we cut the number of training examples to 30 images, which caused the model to be under-trained for many classes and output unsatisfactory results. For instance, bicycle and boat classes both have a really low average IoU below the random agent baseline. Hence we suggest training the model with more training examples as a future work.

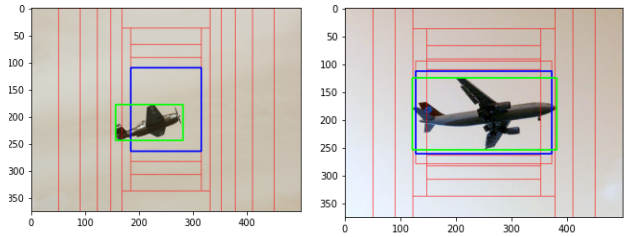

Fig. 9. Two examples of good target detections by a random agent.

V. Discussion and Conclusion

A. The failures of our implementation

As mentioned in earlier sections, for each class of objects, we trained our agent for 50 epochs of maximum 20 steps on only 30 images that contain that object. This insufficient amount of training caused some biased behaviors in our agent for some categories. For instance for bicycle class we realized most of the predicted boxes have a similar vertical form regardless of the shape of target object in image. This behavior can be due to having several objects with vertical boxes in the training data such as figure 10, that caused our agent to become biased and deform the window to a vertical form regardless of the shape of target in the test images, as seen in figure 11. The average IoU in this scenario was approximately 0.1039.

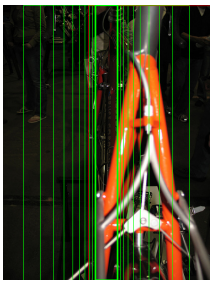

Fig. 10. Vertical target objects that might have biased our agent for other target objects in test images. (Predicted box in bold green)

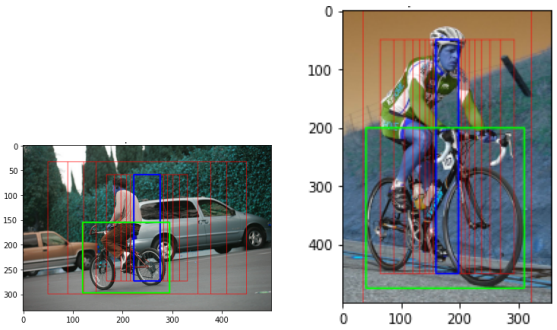

Fig. 11. Bias in agent's prediction of bounding boxes in test images of bicycle class. (Predicted box in bold green, ground truth in red, search path in blue)

Another case was when we had big target objects among our 30 training examples that led our agent to be biased in assuming big targets in test images and choose trigger action in the first steps. This scenario happened with boat category as seen in figure 12

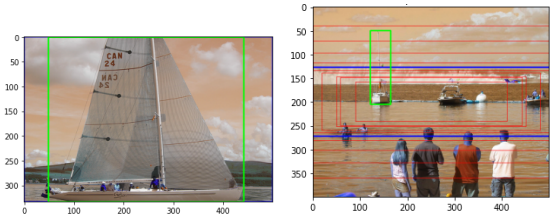

Fig. 12. Bias in agent's prediction of bounding boxes in test images of boat class. In many images the trigger action was chosen in the first step and agent predicts the whole image as the bounding box around the target. (Predicted box in bold green, ground truth in red, search path in blue)

B. Conclusions

In this project, we implemented the active object localization method proposed by Caidedo et al. [1] and conducted multiple experiments on Pascal VOC 2012 dataset. Through the experiments, we found that the agent is capable of localizing a target object with a bounding box. By tuning the important hyperparameters in the algorithm, we found that discount rate, the number of sampled experiences, the size of the history vector and IoU threshold for trigger action all affect the model performance. Besides, adding more layers to the structure of deep q network can lead to better performance, with the cost of increased training time.

Although due to the limitation of time and computational power, our results are not comparable to state-of-the-art solutions in object localization problem, it is still worthwhile to look into how localization problem can be formulated as a dynamic decision process and be solved by a reinforcement learning approach. Meanwhile, by examining the results across different classes, we realized that the model is very sensitive to the number trained samples, and as we greatly reduced the number of training samples, the model's performance was easily affected by the potential bias in training data, thus producing unsatisfying precision even lower than the random model for two of the tested classes.

C. The limitation of approach and future improvements

In our implementation, due to the lack of available resources, we exploited part of the training set and attempted to minimize the training time (still each of our experiments took around 7 hours with google colabratory GPU). With appropriate computational resources, we would like to implement this algorithm with all available training samples and find the optimal results that could be possibly obtained.

Another drawback of the algorithm is that it is only able to detect one object from a particular class at a time, since the MDP will be terminated after one object is found. In future, we would like to extend our implementation to be able to detect all objects from the same class in images.

Moreover, the algorithm is class-dependent, which is a main drawback of this approach. As a future work, we would like

to extend the algorithm to be class-independent to be able to find all objects regardless of their class.

REFERENCES

[1] Caicedo, Juan C. and Lazebnik, Svetlana, Active Object Localization with Deep Reinforcement Learning, IEEE Computer Society, USA, 2015, Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV), pp. 2488–2496

[2] Ross Girshick and Jeff Donahue and Trevor Darrell and Jitendra Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, 2013, arXiv

[3] Miriam Bellver and Xavier Giro-i-Nieto and Ferran Marques and Jordi Torres, Hierarchical Object Detection with Deep Reinforcement Learning, 2016, arXiv

[4] Volodymyr Mnih and Koray Kavukcuoglu and David Silver and Alex Graves and Ioannis Antonoglou and Daan Wierstra and Martin Riedmiller, Playing Atari with Deep Reinforcement Learning, 2013, arXiv