Reinforcement Learning Final Project: Active Object Localization with Deep Reinforcement Learning

Weiting Tan Shuhao Lai
Johns Hopkins University
{wtan12, slai16}@jhu.edu

Abstract

In this project, we reformulated object detection into a Markov Decision Process in order to apply reinforcement learning algorithms to the problem. We reproduced the paper "Active Object Localization with Deep Reinforcement Learning" (Caicedo and Lazebnik, 2015) by using a DQN to generate actions that transform the predicted bounding box until it tightly bounds the region of interest. Further, we provide two extensions: fine-tuning the feature extractor by pretraining a VGG16 network on a classification task and implementing Dueling DQN. The code is available here; this repo contains scripts to create your virtual environment, download and parse data, and run the agents.

1 Background and Motivation

The problem of object localization, or detection, is important for many downstream vision tasks and have been dominated by end-to-end convolution or transformer based approaches. However, with the success of reinforcement learning in Atari games as shown by (Mnih et al., 2013), many researchers are reconsidering the feasibility of leveraging reinforcement learning in traditional computer vision tasks.

Of particular interests is the use of reinforcement learning in object localization as introduced by (Caicedo and Lazebnik, 2015), which uses a top-down approach by attending to a large region of interest and incrementally refining it until the desired object is localized. This approach is fundamentally different from CNNs since no fixed path is used to search for objects (i.e. the fixed path of a sliding window).

To further understand the advantages and disadvantages of reinforcement learning on object localization, we will work to reproduce the methods introduced by (Caicedo and Lazebnik, 2015) and extend the work in two ways: pretrain the feature extractor to improve domain adaption and use dueling DQN. Please view section 3 for more details.

2 Related Works

In the paper "Playing Atari with Deep Reinforcement Learning", Mnih et al. (2013) used a deep learning model to generate Q values for actions to use in Atari. The proposed algorithm used a replay memory to store experiences when the agent plays the Atari games. The DQN model can then sample mini batches from the replay buffer to train on; random samples of the replay buffer is also necessary to de-correlate the time dependent, highly correlated samples generated from the Atari games. The mini batches can be used to train the DQN model using a 1-step bootstrapping approach to estimating the return for a given state after taking an action.

As an extension, Wang et al. (2016) proposed Dueling Networks, which is a DQN network that produces value and advantage estimates for a given input state; these two estimates are then aggregated to produce Q-values. The training procedure of Dueling DQNs closely follow the procedure introduced by (Mnih et al., 2013).

In addition to reinforcement learning, CNNs have played a significant role in the success of object detection. Namely, YOLO models were able to provide multiple object detection in real time by exclusively leveraging convolution layers and making predictions at multiple scales. However, YOLO and many other CNN approaches use very specialized designs (like anchor boxes and bounding box calculations) to achieve their exceptional results.

3 Problem Formulation

In this project, we frame the object detection problem as a Markov Decision Process so that Reinforcement Learning agents can be used. In partic-

ular, we define the states, actions, and rewards as follows:

1. **States:** Since we are using Deep Reinforcement Learning (DRL), the state is represented as a pair (o, h) where o is the feature vector extracted from the predicted ROI using a convolutional neural network (Bengio and Lecun, 1997), which is normally in dimensional space. h is the history of previous 10 actions. Since we have 9 actions and each action is represented as a one-hot vector, $h \in R^{90}$.

2. **Actions:** For object detection, the available actions are transformations that can be applied to the current bounding box until it tightly bounds the object of interest. In this project, the actions are defined as

   (a) Horizontal moves (left and right)
   (b) Vertical moves (up and down)
   (c) Scale changes (maintaining aspect ratio)
   (d) Aspect ratio changes
   (e) Trigger, which is a special action that indicates that an object has been found

   These actions transform the box by a factor relative to its current size. Given the bottom left coordinates (x_1, y_1) and top right coordinates (x_2, y_2), the size we change the bounding box by is $\alpha_w = \alpha \cdot (x_2 - x_1), \alpha_h = \alpha \cdot (y_2 - y_1)$. Here α is chosen to be 0.2 from the author's empirical exploration.

3. **Rewards:** The reward is defined based on Intersection-over-Union (IoU) between the target box and the predicted box. Suppose b is the area covered by predicted box and g is the area from the target, then $IoU = area(b \cap g)/area(b \cup g)$ and the reward is defined to be:

$$R_a(s, s') = sign(IoU(b', g) - IoU(b, g))$$

   Where b' corresponds to the box after transitioning from state s to s' by an action.

4 Methods

In the original paper, a pretrained CNN (trained on a different dataset) is used and only two fully connected layers are used to map image feature retrieved from CNN into Q-values for the 9 available actions. The DQN is trained in the same manner

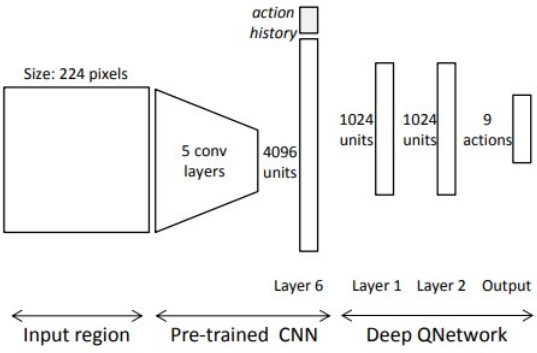

Figure 1: Pretrained CNN with RL agent

as proposed by Mnih et al. (2013); the training procedure will incorporate a replay memory to break short-term correlations between states and to reuse experiences; further, the training objective is to directly approximate $Q^*(s, a)$ via 1-step bootstrapping. Training requires a policy DQN to select actions and a target DQN used to generate target Q values; please view the original paper (Mnih et al., 2013) for more information on training. Note that there is a trained agent for each class and a vector containing a history of taken actions is concatenated to the extracted feature vector (Figure 1).

We will reprodouce this approach to localize objects but we also propose the following extensions:

1. Pretrain the CNN on the training set so the model is better able to extract features in the new domain.

2. Employ dueling network architecture: inspired from Wang et al. (2016), we replaced the single stream Q-Learning (Mnih et al., 2013) by a double stream Q-Learning as shown in Figure 2.

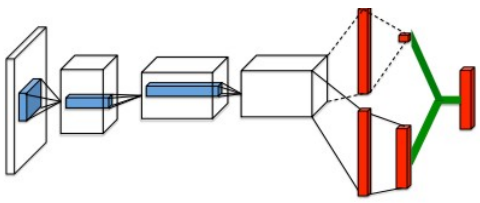

Figure 2: Dueling Network Architecture

We have two stream, one that produces value function $V(s; \theta; \beta)$ and another that produces a $|\mathcal{A}|$ dimensional vector $\mathcal{A}(s, a; \theta, \alpha)$ that represents advantage function $\mathcal{A}^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$,

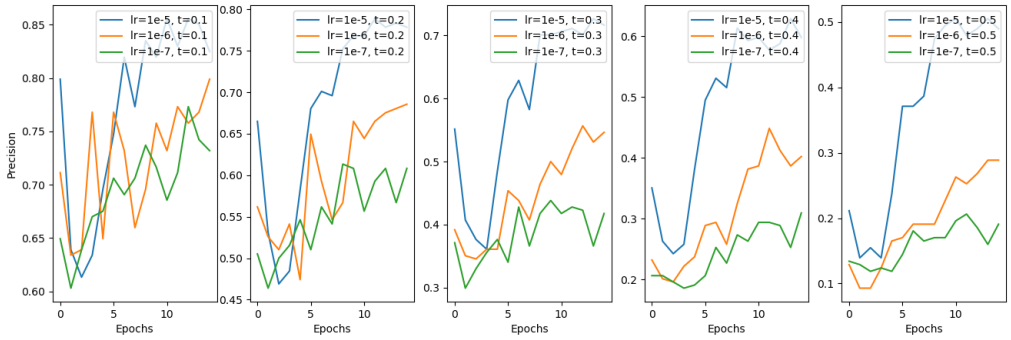

Figure 3: Precision of aeroplane bounding box by different learning rates. For the 5 subplots, each one has a different threshold t. True positives are those predictions that give bounding box with Intersection-Over-Union larger than threshold. The higher the threshold t, the smaller the true postives, which gives smaller precision value.

where α, β are the unique trainable parameters for the two streams. Then we can combine the two streams to generate Q-values using $Q(s,a;\theta,\alpha,\beta) = V(s;\theta,\beta) + \left(\mathcal{A}(s,a;\theta,\alpha) - \frac{1}{|\mathcal{A}|}\sum_{a}'\mathcal{A}(s,a';\theta,\alpha)\right)$. This dueling DQN model can be trained similarly to a vanilla DQN model.

5 Dataset

We use the Pascal VOC 2012 and VOC 2017 dataset (Everingham et al., 2010). Combined, this dataset has 20 classes and contains approximately 27k images, each with one or more objects. We split the data so 70 percent is in the train set, 15 percent is in the validation set, and 15 percent is in the test set. The data is randomly split and we have verified an approximately even class distribution between the train, validation, and test dataset.

5.1 Preprocess

During training for both the reinforcement learning models and pretrained CNN model, the input images are preprocessed by resizing the images such that they maintain their aspect ratios (padding is used instead of cropping in this process).

To pretrain the feature extractor, we use the same dataset (VOC2012 and 2017) but ignore the bounding box labels. We prepare the data as a classification task where each image is associated with one label. When there are multiple classes per image (rare), we perform a weighted sampling from the objects. For example, if an image has three cats and one dog, then we have 75% chance to label it as "cat" and 15% chance to label it as "dog". Class assignment is done each time a mini batch is curated.

5.2 Training

We have trained three models for this project: DQN, Pretrained DQN, and dueling DQN. All three models are trained with the same hyper-parameters [1] as Caicedo and Lazebnik (2015) except for the learning rate. We used learning rate 1^{-5} instead of 1^{-6} because it gives better result from our grid search, shown in figure 3. For Pretrained DQN, we used pretrained/finetuned VGG16 as the image feature extractor. We finetuned VGG16 on VOC datasets and achieved accuracy of about 75% (for the 20-class classification problem) as shown in figure 4. Then, we freeze the feature network and use it as image feature extractor to train RL agent.

Note that we train one RL agent for one class (20 classes of object in total) to improve agent stability during training. Given the time and hardware constraints for this project and the fact that each agent takes several hours to train, we will mainly examine results from five classes (five DQN models).

6 Results

For our results, we used metrics precision and cumulative rewards. We generate results per class because each RL agent is only trained for a single class. Precision is calculated as the number of correct bounding boxes divided by the number of predicted bounding box ($TP/(TP+FP)$). A bounding box prediction is considered correct if its IoU with the ground truth is greater than a preselected threshold. For average cumulative rewards,

[1] Code: https://github.com/steventan0110/ObjectDetectionRL

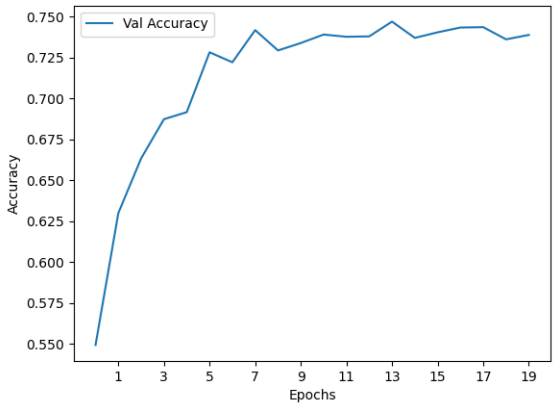

Figure 4: Accuracy of fine-tuned VGG16

we simply accumulate rewards made by RL agents per epoch and divide it by total number of images.

We show plots of precision (evaluated on validation set throughout the training epochs) in figure 5 and plots of rewards in figure 6 for two classes: aeroplane and bird. We also calculate the metrics on the test data for five class in table 1. After analyzing the plots we have several findings:

1. Training converges in a few epochs: from the reward plot, we see that for both classes, each of the three models converges to a reward value after 20 epochs of training. We also checked plots of other classes as well as training loss and the result confirms that training converges in a few epochs. This is intuitive because we don't have too many images for each class and the image feature extractor (vgg16) is freezed during training. Therefore there are not many parameters to train.

2. Precision fluctuate with low correlation to reward: When we compare precision (figure 5) and reward plots (figure 6), we find the model that converges to the highest reward value does not outperform other models in terms of precision. This is not expected because we want reward to be a good proxy of bounding box's quality in order to train RL agent. However, since reward is an artificial metric (where we compare the change of IoU after each action), it's not directly related to the final objective, thus it's possible that precision value does not behave the same as reward.

3. Precision changes drastically between classes: As we see from plots and result table, we find that RL agents for different classes have drastically different results, where aeroplane performs much better than the other classes we experimented with. This could be explained by two reasons. First, we spend most of our time in this project with the aeroplane class because we use it to do hyperparameter tuning on learning rate. We also use it to test our implementation. That's why we have run experiments for it many times and the learning rate/other implementation detail we chose gives the best performance for it. However, for the other classes, we only trained them once with learning rate 1^{-5} without experimenting any other settings. Second, the aeroplane class is one of the easy class to localize because aeroplanes are usually rectangle shape and they usually occupy a large portion of the image with sparse background features (like skies). Other classes like chair can be much harder to detect due to their small size, amount of noise in the background, diverse features, and occlusions.

Model (Aeroplane)	**Precision%**	**Avg Reward**
DQN	53.2	6.48
Pretrain DQN	42.0	10.71
Dueling DQN	46.8	9.52
Model (Bird)	**Precision%**	**Avg Reward**
DQN	13.5	9.06
Pretrain DQN	18.7	9.41
Dueling DQN	13.8	10.28
Model (Bicycle)	**Precision%**	**Avg Reward**
DQN	15.6	7.47
Pretrain DQN	18.8	9.46
Dueling DQN	10.8	9.23
Model (Chair)	**Precision%**	**Avg Reward**
DQN	4.2	6.98
Pretrain DQN	9.7	8.59
Dueling DQN	4.2	9.82
Model (Car)	**Precision%**	**Avg Reward**
DQN	24.5	8.42
Pretrain DQN	22.2	9.35
Dueling DQN	33.8	10.34

Table 1: Evaluation of trained RL agent on Aeroplane, Bird, Bicycle, Chair, and Car classes. Threshold=0.5 is used to calculate the precision.

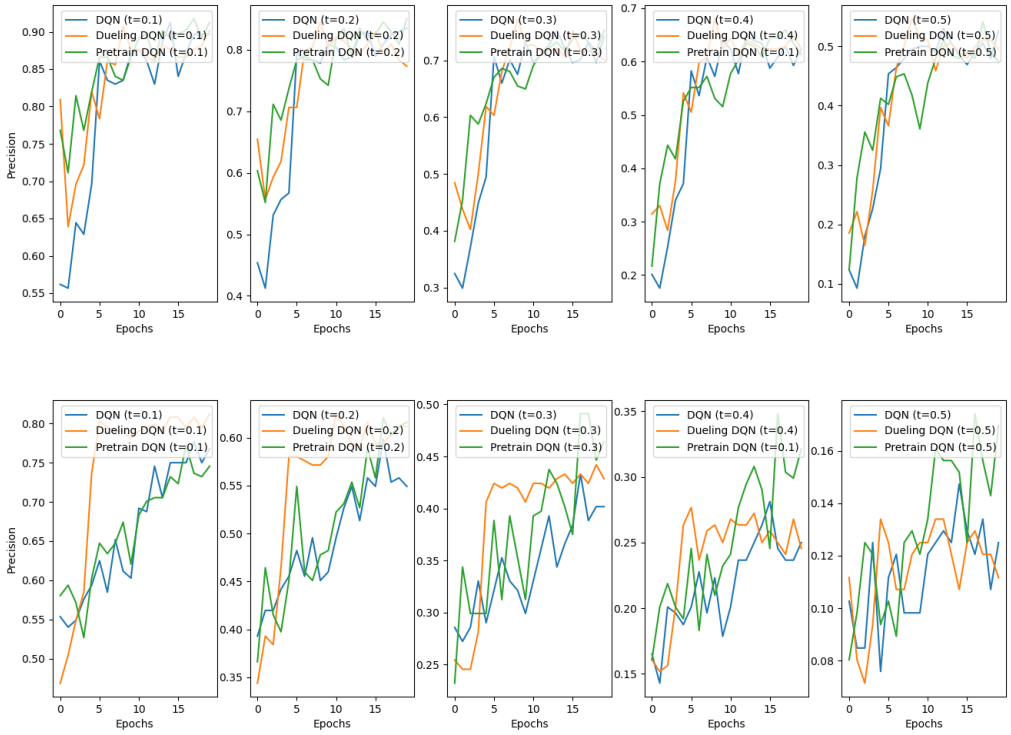

Figure 5: Precision plot for aeroplane and bird class evaluated on validation set

7 Conclusion

For this project, we were able to reproduce the paper "Active Object Localization with Deep Reinforcement Learning" (Caicedo and Lazebnik, 2015) to detect objects using DQNs. We also provided extensions by pretraining the feature extractor and implementing Dueling DQN. We observe that the pretrained model and the Dueling DQN model produce better rewards and precision than the vanilla DQN model with the pretrained model producing the best qualitative bounding boxes.

7.1 Limitations and Future Works

The current design does not have an action denoting "no object present". There are several approaches to resolve this problem:

1. Train another classifier to detect if an object is present before using the reinforcement learning algorithm.

2. Add a "no object" action to the model (this would require negative samples and can introduce instability to the training).

3. Track the trajectory of predicted bounding box to predict if the box is not localizing an object (i.e. oscillations can be a sign that an object is well localized).

Further, the current formulation does not easily enable the DQN to detect multiple objects for a given image. One solution is to simply cross out the detected regions and restart the agent on the edited image; the marked off regions will hopeful encourage the agent to detect the next object. Note, however, we will then need to know when an object is not present, otherwise we will not know when to stop marking off detected objects.

Further, the current formulation of the rewards encourage the agent to take the longest path possible while improving IoU since each action has the potential to incur a positive reward. This may be undesirable since each additional action incurs additional computations, so taking the shortest path is ideal.

Lastly, there exists many DQN models that we can explore for this problem, like Double DQN. There is also a lot of promise with replacing the feature extractor model and using a more robust pretraining task (perhaps using techniques from semi-supervision for learning representations).

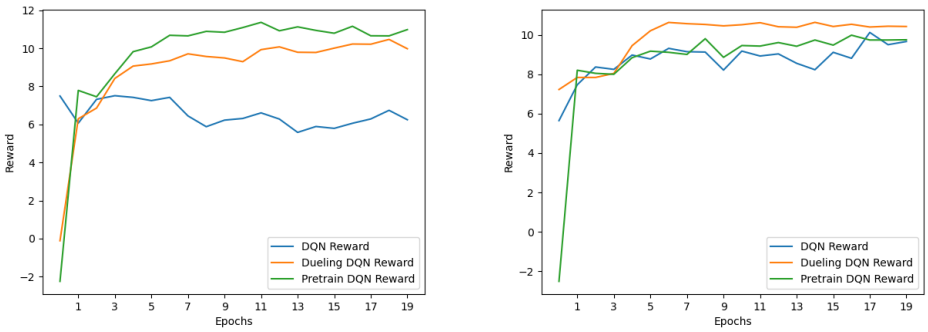

Figure 6: Average cumulative reward plot for aeroplane and bird class evaluated on validation set

8 Contributions

For this project, Steven and Shuhao equally contributed to each part of the project; this includes reviewing the papers, writing and debugging the core models and code, training the models, generating the plots and examples, etc.. We communicated often throughout the project to make this possible.

References

Y. Bengio and Yann Lecun. 1997. Convoevering-ham10lutional networks for images, speech, and time-series.

Juan C. Caicedo and Svetlana Lazebnik. 2015. Active object localization with deep reinforcement learning.

M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. 2010. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado van Hasselt, Marc Lanctot, and Nando de Freitas. 2016. Dueling network architectures for deep reinforcement learning.

A Example Results

This appendix shows some sample results from each model. The results for a given class are laid in a 3 x 3 grid where each row represents a different model: samples from the first row are from a vanilla DQN; samples from second row are from a pretrained DQN; samples from the third row are from a Dueling DQN. The green box denotes the ground truth while the red box denotes the prediction made by the model.

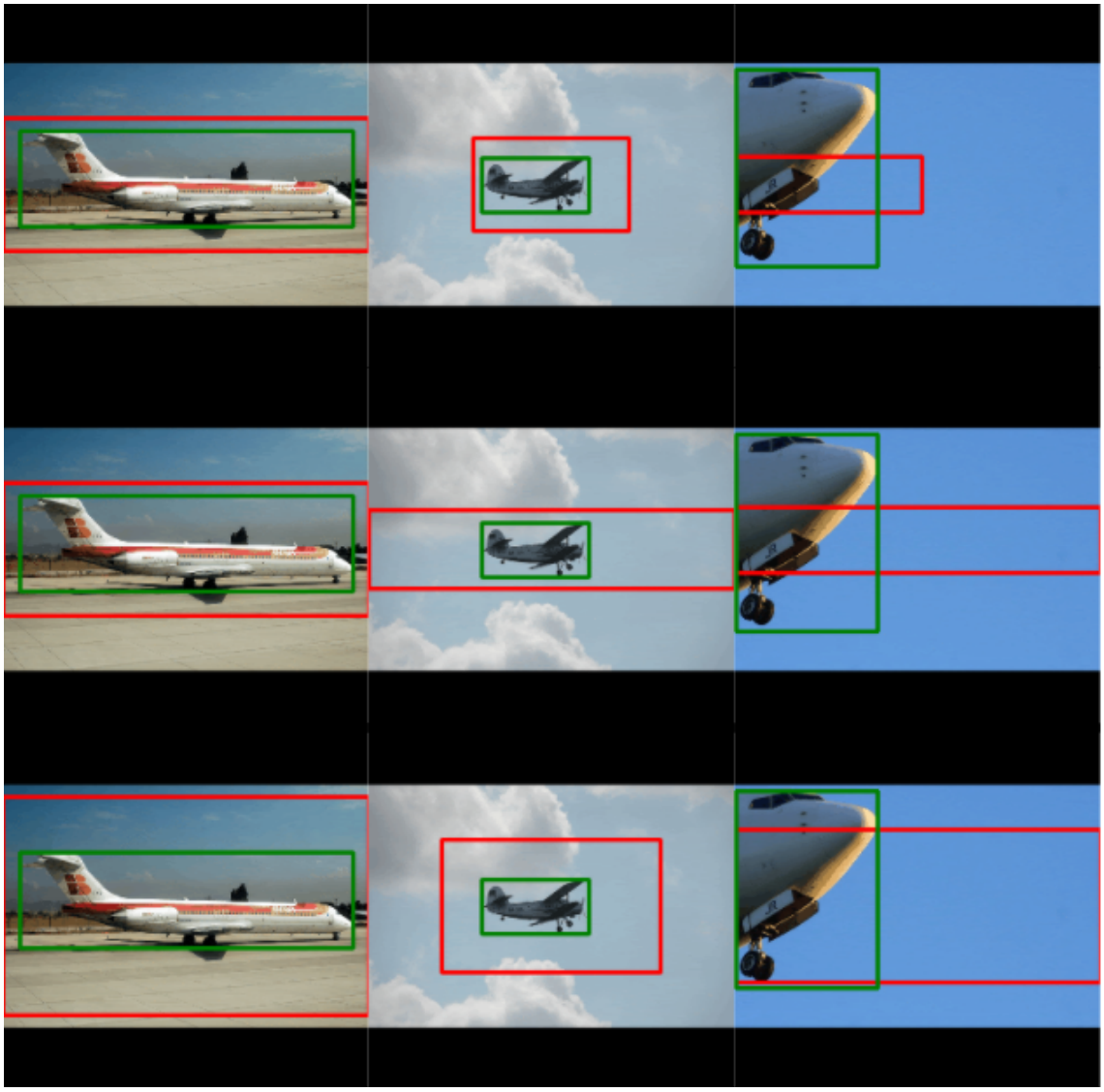

Figure 7: Sample results for the aeroplane class.

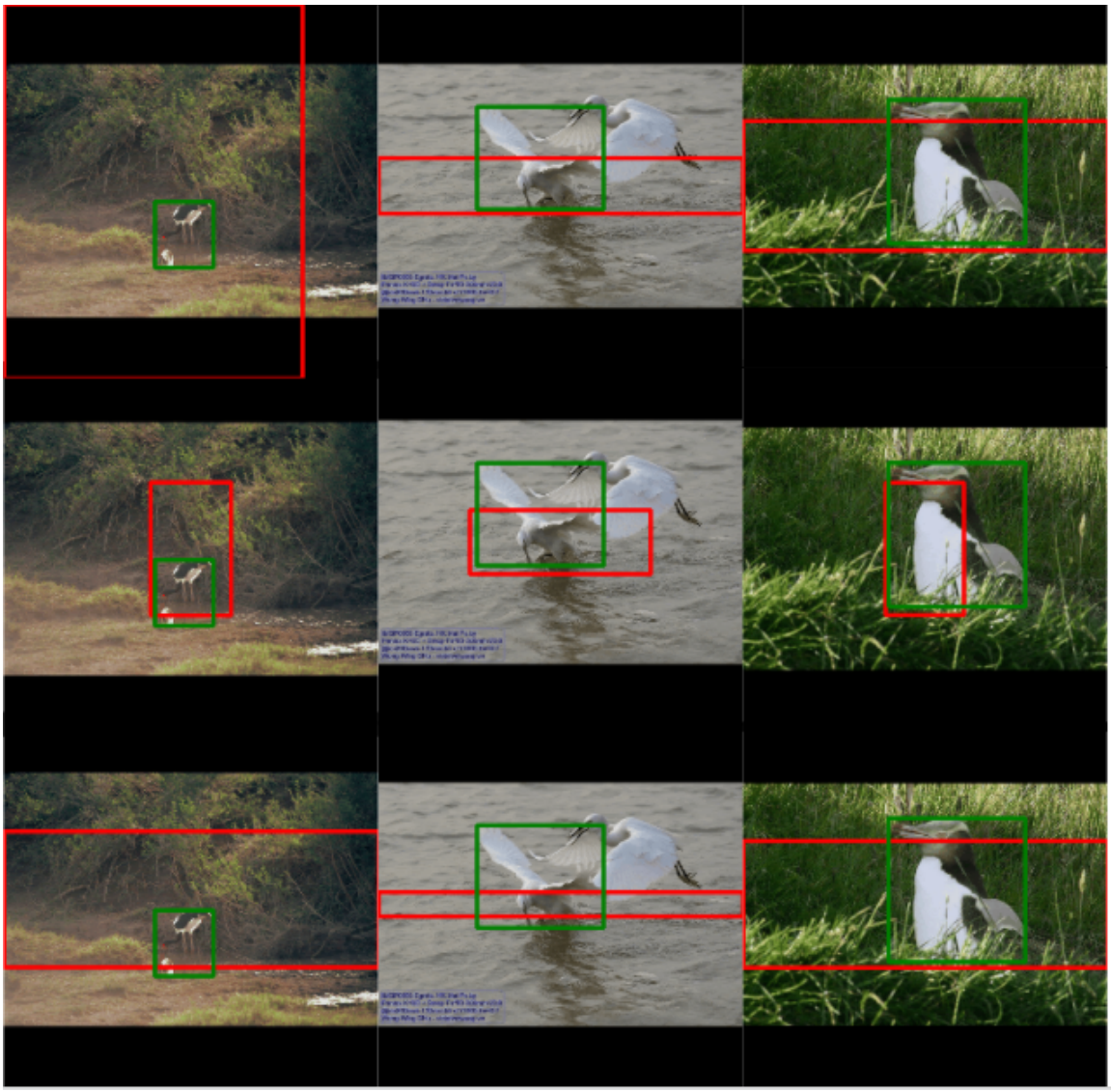

Figure 8: Sample results for the bird class.

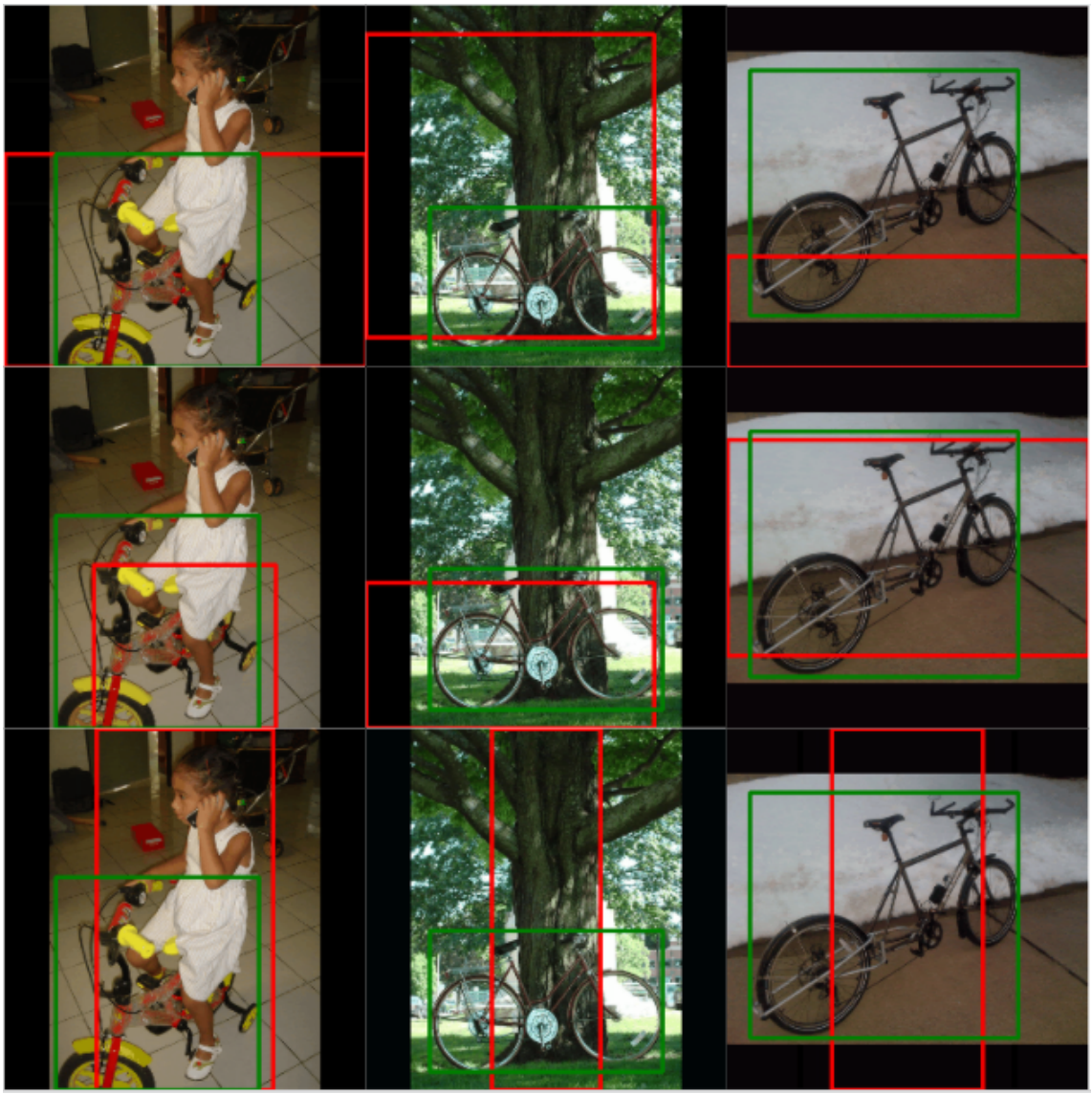

Figure 9: Sample results for the bicycle class

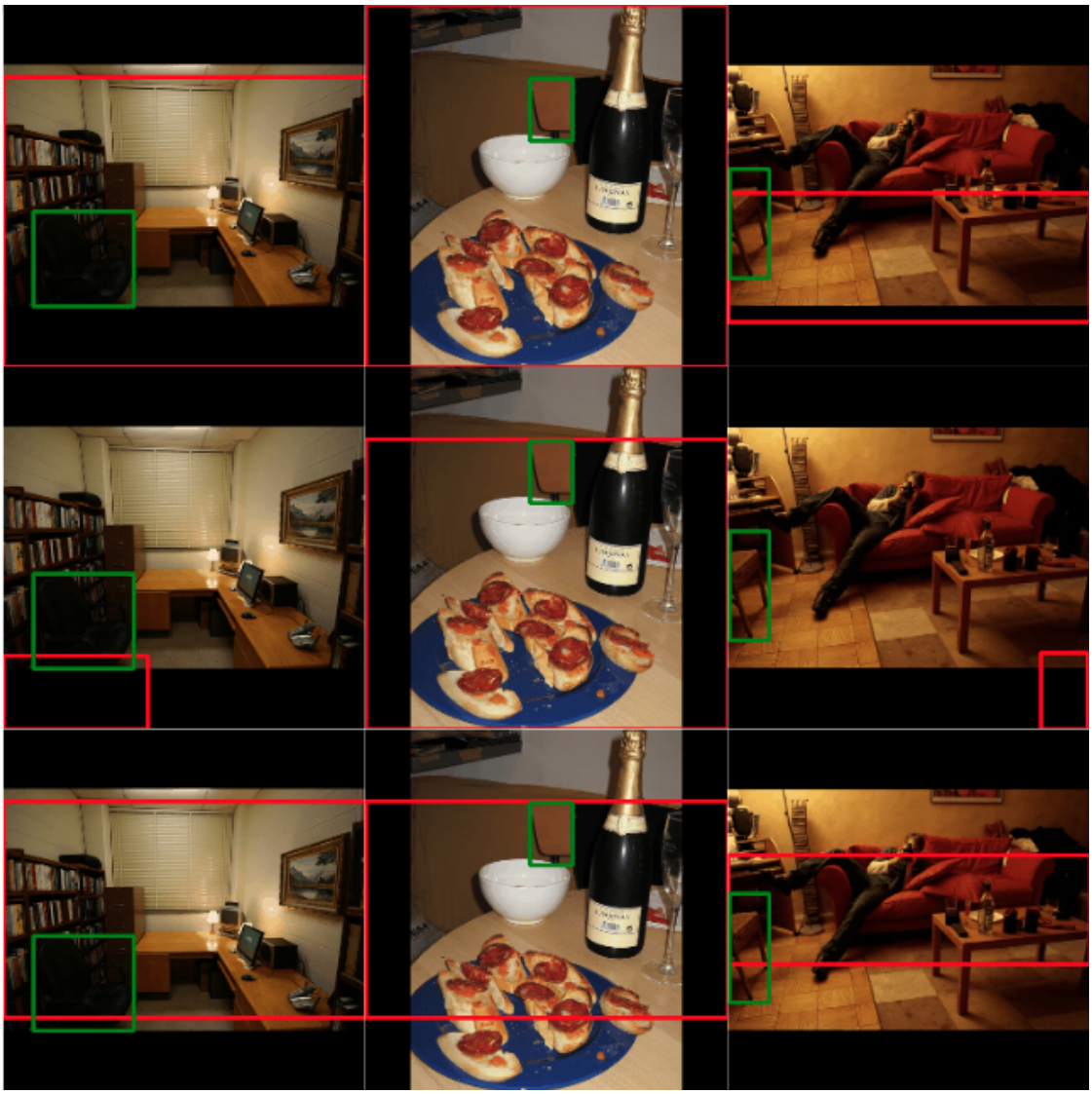

Figure 10: Sample results for the chair class

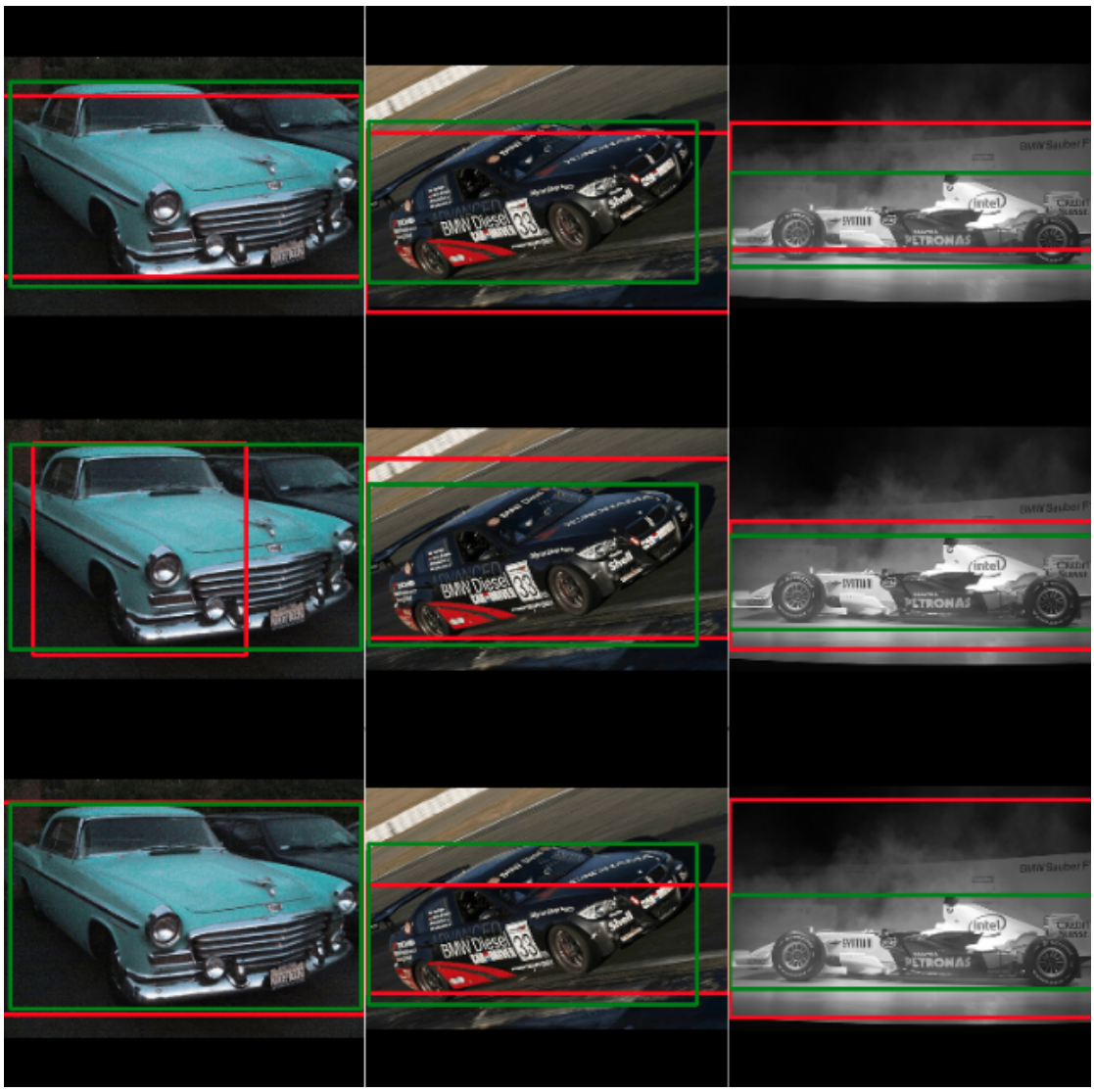

Figure 11: Sample results for the car class