Reinforcement Learning Course Project
Object Detection in an Image Applying Deep Q-Network

Tashrif Billah [1] **Alexander Loh** [*1]

Abstract

Q-learning and neural networks can be combined into a powerful tool for detecting objects in images. In (Caicedo & Lazebnik, 2015), an agent is trained to draw a bounding box around an object in the input image. We intend to replicate the results presented in the above paper by evaluating performance of the agent on the PASCAL VOC 2012 dataset. The proposed learning algorithm makes use of several components - a convolutional neural network (CNN) extracts features from an image, a deep Q-network (DQN) trains an agent to localize the object, and a support vector machine (SVM) categorizes the image inside the bounding boxes. The output of SVM is used to compare the accuracy of the learning algorithm against the results in (Caicedo & Lazebnik, 2015). Finally, we enhanced the DQN algorithm to make use of gradient clipping, prioritized experience replay and target Q-values.

1. Introduction

For many years, the scientific community has been interested in determining methods for accurately finding objects within images. The task of identifying an object in an image is known as recognition. In computer vision, detecting an object means bounding the object in an image by a box to show its location. Scientists have come up with novel techniques of correctly detecting objects in the image even in challenging scenarios i.e. occluded objects, multiple objects, and similar looking objects. The purpose of this project is to familiarize ourselves with state of the art techniques of object detection. An interesting advancement in this detection problem, and the subject of our study, was the application of deep Q-learning to object detection proposed by (Caicedo & Lazebnik, 2015). On top of implementing the method therein, we have also contributed to the model by implementing several modifications to the algorithm that have been proposed in various other deep reinforcement learning papers. In particular, while the basic deep Q-learning algorithm consists of replacing the Q-table with the output of a neural network and training the network based on the target value, we also implemented many of the suggestions proposed in (Mnih et al., 2015) as a novelty to the reference paper we followed. In particular, we used a custom loss function that clips the gradient of the loss to $[-1, 1]$, we implement a target Q-network used to determine the target value whose weights are copied from the main Q-network on an infrequent basis, and we implemented experience replay in order to reduce the correlation between consecutive experience updates. In addition, we implemented prioritized experience replay as proposed in (Schaul et al., 2015); we chose to use the rank-based algorithm described there.

[*]Equal contribution [1]PhD Student, Department of Electrical Engineering, Columbia University. Correspondence to: Alexander Loh <al3475@columbia.edu>, Tashrif Billah <tb2699@columbia.edu>.

Proceedings of the 34th International Conference on Machine Learning, Sydney, Australia, PMLR 70, 2017. Copyright 2017 by the author(s).

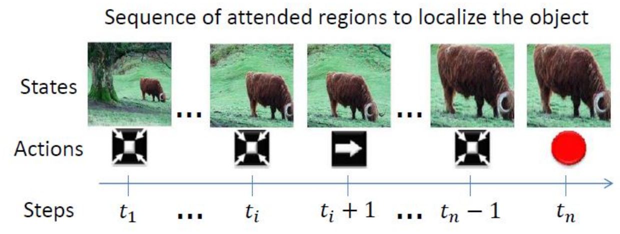

Figure 1. Illustration of the sequence of actions that detects a cow in an image. The agent evaluates different regions and decides how to transform a bounding box to sequentially localize the object.

2. Related Work

Recent works approach the problem of object detection in two steps - region proposal and object proposal. First, the agent looks at various regions in order to find ones that look interesting. The proposed regions are then evaluated by a classifier to determine possible existence of the desired object. Such a technique is presented in (Girshick

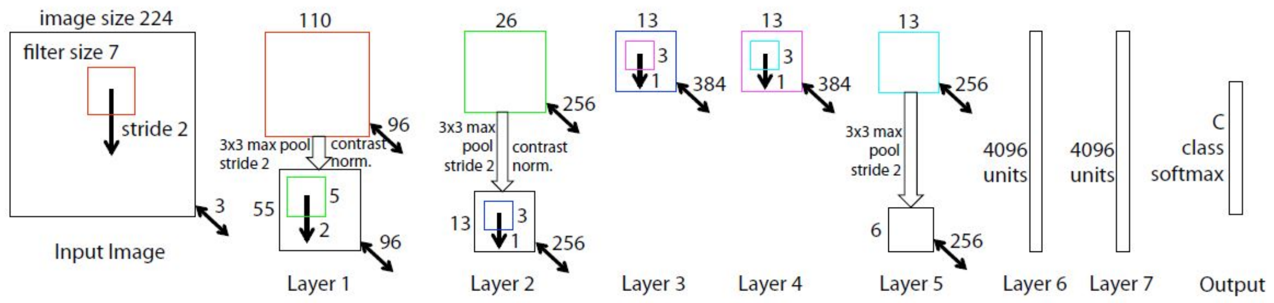

Figure 2. The backbone convolutional neural network (Zeiler & Fergus, 2014) for the object detection problem considered in (Caicedo & Lazebnik, 2015)

et al., 2014) under the acronym R-CNN that extracts around 2000 bottom-up region proposals, computes features for each proposal using a large CNN, and then classifies each region using class-specific linear SVMs. The R-CNN system still has the best performance and remains as a strong baseline. However, a major drawback of R-CNN is that it relies on a large number of object proposals to reach that performance, and demands significant computational power. Similar techniques are investigated in (Uijlings et al., 2013); (Wang et al., 2013). A fully connected CNN topology modified by adding in a Region-of-Interest (RoI) pooling layer is presented in (Jie et al., 2016). They propose position-sensitive score maps to address the dilemma between translation-invariance in image classification and translation-variance in object detection and thus avoid going over too many regions. They claim to achieve at least 2.5 times faster detection than the R-CNN counterpart. To circumvent heavy amount of region wise computation, a Tree-structured Reinforcement Learning (Tree-RL) approach is proposed in (Jie et al., 2016) that follows a sequential search for objects by fully exploiting both the current observation and historical search paths. The algorithm follows multiple near-optimal policies and is able to find multiple objects with a single feedforward pass. On the other hand (Lu et al., 2016), proposed a top-down search strategy to recursively divide a window into sub-windows. Then all the visited windows serve as anchors to regress the locations of object bounding boxes.

Meanwhile, the algorithm that we chose for implementation (Caicedo & Lazebnik, 2015), learns an optimal policy to localize a single object through deep Q-learning. Furthermore, MultiBox (Erhan et al., 2014) and DetNet (Szegedy et al., 2013) predict bounding boxes from input images using deep CNNs with a regression objective.

3. Methodology

The algorithm that we implemented performs object detection through deep Q-learning. In this environment, a state is defined as the combination of the history of actions along with a feature vector derived from the image, and an action is defined as the set of transformations that can be performed on an image, plus a 'trigger' action that indicates that the algorithm has found an object. The feature vector is computed by passing the portion of the image covered by the bounding box through a pre-trained CNN and observing the output at the top of its convolution layers. During the training phase, rewards are given after taking an action based upon the intersection-over-union (IoU) metric; these rewards are used to train a neural network that will learn to predict the value of the Q function. Since the training phase uses the ground truth boxes of the environment, this algorithm can also be seen as an apprenticeship learning algorithm, in which an expert knows the true bounding boxes and the agent's job is to learn to produce results that are consistent with the expert's. The steps are elaborated below.

3.1. CNN Architecture

The algorithm that we implemented, has followed the CNN architecture of (Zeiler & Fergus, 2014) as the backbone network. The CNN architecture (Fig. 2) has 8 layers in total. A 224×224 crop of an image with 3 color channels is presented as the input. The first hidden layer is a convolutional layer with 96 different filters for each channel. The first layer filter size is 7×7, using a stride of 2 in both the horizontal and vertical directions. The resulting feature maps are then: (i) passed through a ReLU (not shown), (ii) max pooled (within 3×3 regions, using stride 2) and (iii) contrast normalized across feature maps to give 96 different 55×55 feature maps. Similar operations with different dimensions/filter counts are repeated in layers 2,3,4, and 5 as shown in Fig. 2. The last two hidden layers are fully

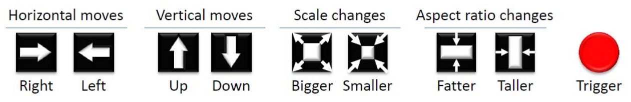

Figure 3. Illustration of the bounding box deformation actions proposed in (Caicedo & Lazebnik, 2015).

connected, taking features from Layer 5 as input. The final layer is a 20-way softmax function, with one output for each class. All filters and feature maps are square shaped.

3.2. State Representation

The state representation used for learning is a tuple (o, h), derived from the current bounding box and the history of past actions taken to reach this bounding box. The image feature vector $o \in \mathbb{R}^{4096}$, is obtained by first cropping the examined image by the current bounding box and resizing it to 224×224, then passing it through the pre-trained CNN in Fig. 2. Finally, o is obtained as the output of Layer 6 of the CNN. To obtain $h \in \mathbb{R}^{90}$, the action history, each action of the last 10 deformation actions made on the current bounding box is represented by a one-hot encoded vector in $\mathbb{R}^9$. These vectors are concatenated to obtain h. Although h is of low dimensionality compared to o, it can still help with stabilizing the search trajectory.

3.3. Bounding Box Deformation

The set of actions on the bounding box is composed of eight transformations that can be applied to the box and one action to terminate the search process, as illustrated in Fig. 3. The actions are organized in four sub-sets: actions to move the box in the horizontal and vertical axes, actions to change scale, and actions to modify aspect ratio. Each of these actions moves the corners of the bounding box by a fixed amount relative to its current dimensions; for instance, moving the box left is done by subtracting $\alpha(x_2 - x_1)$ from the upper left and lower right x-coordinates x_1 and x_2, while making it shorter is done by adding $\alpha(y_2 - y_1)$ to the upper left y-coordinate y_1 and subtracting it from the lower right y-coordinate y_2, where α is a fixed constant. Finally, the only action that does not transform the box is a trigger to indicate that the agent has chosen to stop at the current box. This action terminates the sequence of the current search, and restarts the box in an initial position to begin the search for a new object. By doing so, the algorithm can locate multiple objects in an image.

3.4. Region Proposal

The agent starts processing each image by initializing the current bounding box to the dimensions of the image. While training, it follows an ϵ-greedy policy in which ϵ is decreased with each pass through the training set. Since the environment knows the ground truth boxes, the reward function is calculated with respect to the IoU of the current box with the ground truth box. This allows the agent to determine which actions produce positive rewards. When the agent chooses to explore, instead of choosing a random action, it chooses an action from the set of actions that produce positive rewards, or any action if all actions give negative reward. In order to prevent the agent from proceeding forever, it is forced to terminate after 40 steps; (Caicedo & Lazebnik, 2015) report that most objects are able to be localized within this many steps.

3.5. Deep Q-Network

The above steps are put together to yield the DQN as shown in Fig. 4. It uses a neural network (represented by Layers 1, 2, and Output of the preceding figure) to estimate the $Q(s, a)$ state-action pair value function in place of the Q-table used in the conventional Q-learning algorithm as given in (Watkins, 1992).

Improvement in the steps of (Caicedo & Lazebnik, 2015) is measured using the Intersection-over-Union (IoU) between the target object and the predicted box, which is defined as follows: Let b be the box of an observable region, and g the ground truth box for a target object. Then, the IoU between b and g is defined as:

$$IoU(b, g) = area(b \cap g)/area(b \cup g) \tag{1}$$

Meanwhile, the reward function $R_a(s; s')$ for actions that transform the bounding boxes is then taken as follows:

$$R_a(s, s') = sign(IoU(b', g) - IoU(b, g)) \tag{2}$$

Intuitively, equation 2 implies that the reward is +1 if the IoU is improved when going from state s to state s', or -1 otherwise. Meanwhile, the trigger has a different reward scheme as follows:

$$R_\omega(s, s') = \begin{cases} +\eta, & \text{if } IoU(b, g) \geq \tau \\ -\eta, & \text{otherwise} \end{cases} \tag{3}$$

where ω is the trigger action, η is the trigger reward, and τ is a threshold that indicates the minimum IoU to mark a region as a positive detection.

As the agent traverses the state space, the neural network is trained to predict Q more accurately. This is done by defining the loss for an action taken at a particular state to be a function of $r + \gamma * max_{a'} Q(s', a') - Q(s, a)$, where r is the reward for going from state s to s', γ is the discount factor, $Q(s', :)$ and $Q(s, :)$ are outputs of the neural network to inputs s' and s respectively, and a is the action chosen based off of $Q(s, :)$. This loss is used to train the fully connected layer of the DQN (the layers labeled by Deep QNetwork in Fig. 4)

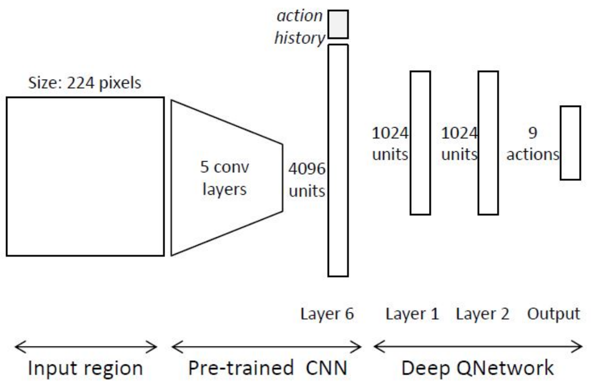

Figure 4. The architecture of the deep Q-network (Caicedo & Lazebnik, 2015): The input region is first scaled to 224×224 pixels and then transformed to a feature vector by a pre-trained CNN. The output of the CNN is concatenated with the action history vector to get a representation of the state. It is processed by the Q-network which predicts the optimal action on the bounding box so that the box progresses towards localizing the object.

3.6. Region Predicting

To predict the location of an object in a test image, the agent starts from a bounding box covering the dimensions of the image and takes the action that maximizes the value of $Q(s, :)$ at each step. Once the trigger action is hit, the agent then returns the last bounding box as its prediction. In order to prevent the agent from proceeding forever, we force the agent to take the trigger action after 40 iterations.

3.7. Region Scoring

In order to make our results comparable with those given by (Caicedo & Lazebnik, 2015), we implemented an evaluator for our results as a one-vs-all SVM. To do this, we took the training images and cropped out their objects using the ground truth bounding boxes for each image. We then resized each cropped image to 224×224 and passed it through the CNN to obtain feature vectors in the same way that we did for the DQN. We then trained a kernalized SVM for each object class by labeling all features corresponding to the same class as the SVM as $+1$, and all other features as -1. We then solved the SVM optimization problem (Cortes & Vapnik, 1995) for each class in MATLAB.

We evaluated the results of the DQN using the SVM by looking at the input images and their corresponding predicted bounding boxes. We cropped each image by its predicted bounding box, then predicted the labels of the resulting image from all of the 20 SVMs. Ideally, only one SVM class is supposed to give $+1$ on any given image, with all the others giving -1, allowing us to say that the image is of the class whose SVM reported $+1$. In the case that multiple SVMs report $+1$, we choose the class randomly among those SVMs, and in the case that they all report -1, we choose the class randomly. We consider a prediction to be correct if the label predicted by the SVM coincides with the ground truth label of the object in the image. Finally, we computed the per-class accuracy of the DQN as the proportion of images where the SVM predicted a correct class label out of all the images of a particular class.

3.8. Novelty

In addition to the standard algorithm laid out in the previous subsections, we implemented a number of improvements that have been shown to usually improve converge speed in practice. To simplify the explanation, we define the Q-learning difference for a particular set of s, a, r, s' values to be $r + \gamma * max_{a'} Q(s', a') - Q(s, a)$, the quantity first given in Section 3.5. First, we implemented gradient clipping by training the DQN using Huber loss (defined as $|x|$ when x is outside $[-1, 1]$ and x^2 otherwise, where x is the Q-learning difference) as opposed to mean squared error. This keeps the gradient of the loss function between -1 and 1. Next, we implemented two suggested improvements to DQN given in (Mnih et al., 2015): experience replay and target values. Experience replay was implemented by storing the values of (s, a, r, s') at each step that the algorithm took in a buffer. At each step, we selected 8 samples from this buffer and trained the DQN on this minibatch. We chose to keep the buffer to a size of $1,000$ samples. This method reduces the temporal correlation between updates. W then implemented target values by using two topologically identical DQNs to compute the Q-learning difference, one for $Q(s', a')$ and one for $Q(s, a)$. The network used to compute $Q(s, a)$ is the same as before, and is updated on each step by experience minibatches. However, the network used to compute $Q(s', a')$ has its weights copied from this network, but only on an infrequent basis; we chose to copy the weights once every 500 steps. Doing this will help to reduce oscillations in the weights of the DQN. Finally, we implemented the prioritized experience replay procedure given in (Schaul et al., 2015). This process selects experiences during experience replay in a weighted manner so as to emphasize training on the examples from which we can learn the most. We chose to use the rank-based method in (Schaul et al., 2015); this gives experience i a weight of $\frac{1}{rank(i)}$, where $rank(i)$ is the position of i in the list of experiences when it is sorted in decreasing order of the absolute value of their corresponding difference values so that the experience with the largest difference magnitude occurs at position 1, the next largest at position 2, etc.

4. Experiment and Results

Up to the milestone of the project, we trained the backbone CNN (Zeiler & Fergus, 2014) and a simplified version of the DQN (Mnih et al., 2015) building upon the CNN. The training was done with Keras library, using the Tensorflow as the backend, on Python. And finally, we used a category specific SVM to recognize the detected object. The implemented algorithm is tested on the PASCAL VOC 2012 (Everingham et al., 2010) dataset. The steps we took are outlined in the following subsections.

4.1. Database Description

The training data consists of images from 20 different classes in the PASCAL VOC 2012 dataset. Each image has an annotation file giving a bounding box and object class label for each object present in the image among the 20 classes. Multiple objects from multiple classes may be present in the same image. However, for simplicity, we stuck to the images with only a single label from this dataset. The full dataset consists of 17,125 images, of which 8,966 of them have unique labels. The number of images per class is graphically shown in Fig. 5. It should be noted that (Caicedo & Lazebnik, 2015) trained their DQN on the PASCAL VOC 2012 and PASCAL VOC 2007 datasets and evaluated their DQN on the PASCAL VOC 2007 dataset and thus their results may differ from ours.

4.2. CNN: Image Classification

Each RGB image in the dataset was pre-processed following the approach in (Zeiler & Fergus, 2014) and the architecture in Fig. 4 by resizing the smallest dimension to 256 and cropping the center 256×256 region. The per-pixel mean over all images was then subtracted from each image. The dataset was then augmented by generating several subimages based off each image; regions at the corners and center of the image were cropped out of each image and optionally flipped horizontally in order to create 10 new images of size 224×224 out of each image. We then split the dataset into training and test datasets, with training data making up 80% of the data, and test data making up the other 20%. This gave us a total of 71,128 subimages in the training set and 18,532 subimages in the testing set, both with approximately equal class distribution as in the original dataset. In training the CNN, stochastic gradient descent with a learning rate of 10^{-2} and a momentum term of 0.9 was used. Dropout was implemented in layers 6 and 7 at a rate of 0.5, and all weights were initialized to 10^{-2} and all biases were initialized to 0. We used mini-batch sizes of 128 and trained the dataset over 70 epochs. Due to the smaller size of the PASCAL VOC dataset as compared to ImageNet, our training procedure was much faster than that reported by (Zeiler & Fergus, 2014).

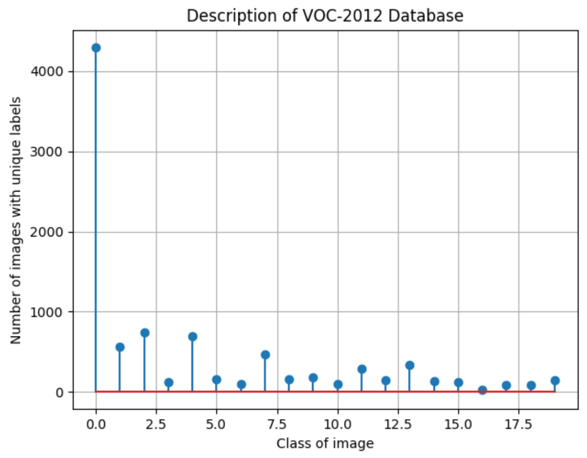

Figure 5. The database (Everingham et al., 2010) with 20 classes of images. Classes are, from left to right, person, bird, cat, cow, dog, horse, sheep, airplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, table, plant, sofa, tv.

4.3. DQN: Object Detection

In implementing the DQN described in the previous section, our choice of parameters were made to be as consistent as possible with those used by (Caicedo & Lazebnik, 2015). To be specific, we set $\alpha = 0.2$, where α is the amount that the bounding box can be deformed by in each dimension, as described in Section 3.3. We also manipulate ϵ, the exploration probability, on each epoch by initializing it to be 1.0 and decreasing it linearly until it reaches a value of 0.1 on the 5^{th} epoch, where an epoch is a single pass through the training data and we train the network over 15 epochs. Finally, we also set the trigger reward $\eta = 3$, the trigger reward threshold $\tau = 0.6$ and discount factor $\gamma = 0.2$, as described in Section 3.5. We processed the DQN on the PASCAL VOC 2012 dataset, where we used the same partition ratios as before: 80% of the images with unique labels as training data, and the remaining 20% as test data. This gave us $7,175$ training images and $1,793$ test images.

4.4. SVM: Detected Object Recognition

In implementing our SVM, we defined the kernel used as a polynomial kernel such that $K(x, y) = (x^T y + 1)^3$, and we set the penalty parameter $C = 1$ (see (Cortes & Vapnik, 1995) for more details on these parameters). The training and test partition ratios of the data are the same as for the DQN.

4.5. Resource and Computation

Due to the large amount of data associated with this work, we leverage the computing capacity of Google Cloud, courtesy of the Columbia EE department. We set up a virtual machine consisting of 16 n1-highmem CPUs, 4 NVIDIA Tesla K80 GPUs, and 104 GB memory. The machine is located in zone ”us-east1-d”. After setting up the machine and installing the necessary components, we were able to run our algorithms. When training the CNN on these machines, each epoch takes 54.28 seconds on average. We trained our neural network for 70 epochs which took a total time of 63.3 minutes. When training the deep Q-learning procedure on these machines, each epoch takes 2.55 hours on average. We trained the algorithm for 15 epochs which took a total time of 38.3 hours. When training the SVM on our local computers, the entire process took less than an hour.

4.6. Results

The complete object detection algorithm as in (Caicedo & Lazebnik, 2015) has three parts: CNN, DQN, and SVM. We wrote our own code for the whole implementation. Here, we present the results of the three separate parts to validate their correctness. Finally, we present the results for the whole system and compare them with the original work.

The obtained results for the CNN are shown in Table 1. By inspection, we see that our experimental results are comparable to those given in (Zeiler & Fergus, 2014). This is likely attributable to the fact that some image data was discarded due to having multiple labels. In particular, bottles, buses, chairs, cows, tables, and horses have slightly lower prediction accuracies. Looking at Fig. 5, we can see that there are comparatively fewer examples in these classes. Thus, it may be necessary to use all of the data if we want to hope to improve our accuracy on these classes.

The results for the DQN are shown in Table 2. The results are given in terms of IoU as defined in 1. Having the average IoU of the predicted bounding boxes in the range $0.5-0.8$ indicates that the algorithm is able to cover at least some portion of the bounding box most of the time.

The results for recognition of the predicted objects by

Table 1. CNN: Percentage classification accuracy comparison between our simulation and the reference (Zeiler & Fergus, 2014)

Class %	Ref	Ours	Class %	Ref	Ours
Airplane	96.0	86.3	Table	67.7	18.1
Bicycle	77.1	53.1	Dog	87.8	44.6
Bird	88.4	54.8	Horse	86.0	18.7
Boat	85.5	60.2	Motorbike	85.1	51.7
Bottle	55.8	29.0	Person	90.9	81.4
Bus	85.8	28.5	Plant	52.2	50.0
Car	78.6	56.0	Sheep	83.6	47.6
Cat	91.2	71.1	Sofa	61.1	38.2
Chair	65.0	20.8	Train	91.8	48.2
Cow	74.4	26.0	TV	76.1	71.4
Mean	Ref: 79.0%			Ours: 67.8%	

Table 2. DQN: Observed average IoU between the predicted boxes and the ground truth boxes produced by the algorithm in (Caicedo & Lazebnik, 2015)

Class	Avg. IoU	Class	Avg. IoU
Airplane	0.71	Table	0.43
Bicycle	0.56	Dog	0.69
Bird	0.63	Horse	0.70
Boat	0.73	Motorbike	0.70
Bottle	0.52	Person	0.55
Bus	0.61	Plant	0.58
Car	0.54	Sheep	0.71
Cat	0.64	Sofa	0.78
Chair	0.81	Train	0.68
Cow	0.67	TV	0.69

Table 3. SVM: Classification accuracy of our 20 category specific one-vs-all SVMs

Class	% Acc.	Class	% Acc.
Airplane	92.6	Table	40.0
Bicycle	87.5	Dog	79.8
Bird	84.2	Horse	78.1
Boat	66.6	Motorbike	68.9
Bottle	81.4	Person	98.0
Bus	85.7	Plant	52.9
Car	93.1	Sheep	61.9
Cat	87.2	Sofa	70.5
Chair	70.8	Train	92.4
Cow	40.0	TV	96.5
Mean		89.7%	

category-specific SVMs are shown in 3. The mean accuracy of detection is 89% which is a satisfactory result that can be used to validate our implementation.

Finally, we are now able to compute bounding boxes using the DQN and evaluate them on the SVM to determine the mean average precision of object detection (MAP). Though our CNN performs reasonably well, we eventually found and switched to a pre-trained VGG19 (Simonyan & Zisserman, 2014) CNN model following the original paper to get comparable results. The MAP is the metric used in (Caicedo & Lazebnik, 2015) to evaluate performance of the agent. The results are noted in Table 4. Also, a sample demonstration of agent progression is given in Fig. 6 that localizes the object in an image. Finally, detected bounding boxes along with the ground truth boxes are shown in Fig. 7. The visual observation of the above figures and tables corroborate the correctness of our implementation.

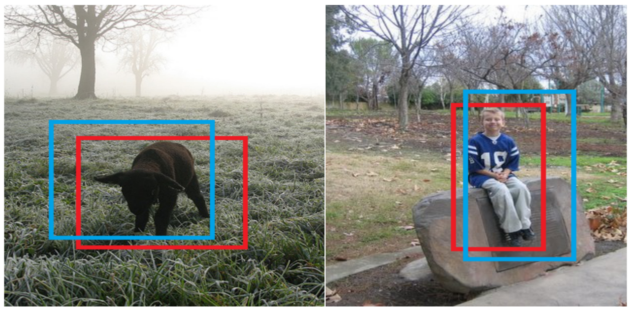

Figure 6. The detected bounding boxes in blue and the ground truth bounding boxes in red. The greater degree of overlap substantiates the correctness of our implementation.

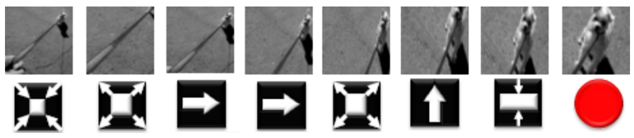

Figure 7. Agent sequentially performs the actions noted below the images to localize the object.

5. Conclusion

The goal of the project was to study the technique of object detection from a reinforcement learning perspective. Following the work of (Caicedo & Lazebnik, 2015), we have implemented the complete object detection system by ourselves. From algorithmic point of view, the project has three major aspects: training a CNN to extract features from images, training a DQN on top of the CNN for taking decision on bounding box deformation, and finally predicting the detected object as one of the 20 classes in the database using an SVM. All three aspects have been separately tested and corresponding results are noted. Finally, they have been put together to demonstrate comparable results to the original work (Caicedo & Lazebnik, 2015). The algorithm was implemented on VOC 2012 (Everingham et al., 2010) database. However, coarse results of CNN and DQN training were noted in the milestone. Building upon that, as committed, we have achieved the following:

1. We have fine tuned our own CNN. However, due to the limitation of computational resource, we could not achieve full training after running the algorithm on Google Cloud for two days. Also, the the local contrast normalization of the CNN was not clear in the paper to implement from scratch. That is why, we followed the pre-trained CNN model VGG19 like the one that has been followed in the original work (Caicedo & Lazebnik, 2015). Despite the above limitation, we have separately fine tuned the three major components of our project- CNN, DQN, and SVM a great deal to obtain comparable result as of the original work.

2. We have augmented our DQN with improvements such as prioritized experience replay, gradient clipping, and target values, which can help improve the convergence speed of the network.

3. Twenty category-specific SVMs have been implemented that predict the detected objects with good accuracy. Achieving this final step paved the way to show comparison of our results with the original work.

Implementing the above modification and fine tuning, our mean average precision of object detection is 44.9, which

Table 4. Average precision of detection per category from the DQN.

Method	Aero	Bicycle	Bird	Boat	Bottle	Bus	Car	Cat	Chair	Cow	Table	Dog	Horse	Motor	Person	Plant	Sheep	Sofa	Train	Tv	MAP
Ref	57.9	56.7	38.4	33.0	17.5	51.1	52.7	53.0	17.8	39.1	47.1	52.2	58.0	57.0	45.2	19.3	42.2	35.5	54.8	49.0	43.9
Ours	4.4	28.1	33.3	30.9	20.6	13.0	18.3	43.8	7.1	5.5	3.5	23.7	9.0	2.8	65.4	21.4	42.1	11.1	39.0	54.8	44.9

is higher than 43.9 as in the original work. However, it should be noted that our improvements come mostly from the higher MAP of the 'person' class, which dominates the dataset. For a few of the other classes, our precisions are somewhat lower. This could be attributed to the fact that our dataset was smaller than the one used by (Caicedo & Lazebnik, 2015); they used the entirety of the combination of the PASCAL VOC 2012 and PASCAL VOC 2007 datasets for training, while we only used the portion of the PASCAL VOC 2012 dataset that contained images with only one object. In future work, we may look forward to combining the PASCAL VOC 2007 dataset with the PASCAL VOC 2012 dataset in order to increase the amount of data available to us.

6. Acknowledgement

We would like to thank the courtesy of Columbia Electrical Engineering Department for providing us with Google Cloud credit that was used to train different stages of the algorithm. Moreover, we would like to thank TA Chen-Yu Yen for his timely email communication that helped us setting up our Google Cloud account. Notably, Dr. Chong Li has become a very popular instructor at Columbia after teaching here only two semesters. His engaging teaching style, quizzing homework, and fair grading policy attracts students to participate in effective learning. We wish him a successful industry and academic career.

References

Caicedo, Juan C. and Lazebnik, Svetlana. Active object localization with deep reinforcement learning. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

Cortes, Corinna and Vapnik, Vladimir. Support-vector networks. *Machine Learning*, 20(3):273–297, Sep 1995. ISSN 1573-0565. doi: 10.1007/BF00994018. URL `https://doi.org/10.1007/BF00994018`.

Erhan, Dumitru, Szegedy, Christian, Toshev, Alexander, and Anguelov, Dragomir. Scalable object detection using deep neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2147–2154, 2014.

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

Girshick, Ross, Donahue, Jeff, Darrell, Trevor, and Malik, Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

Jie, Zequn, Liang, Xiaodan, Feng, Jiashi, Jin, Xiaojie, Lu, Wen, and Yan, Shuicheng. Tree-structured reinforcement learning for sequential object localization. In *Advances in Neural Information Processing Systems*, pp. 127–135, 2016.

Lu, Yongxi, Javidi, Tara, and Lazebnik, Svetlana. Adaptive object detection using adjacency and zoom prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2351–2359, 2016.

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

Schaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015. URL `http://arxiv.org/abs/1511.05952`.

Simonyan, Karen and Zisserman, Andrew. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Szegedy, Christian, Toshev, Alexander, and Erhan, Dumitru. Deep neural networks for object detection. In *Advances in neural information processing systems*, pp. 2553–2561, 2013.

Uijlings, Jasper RR, Van De Sande, Koen EA, Gevers, Theo, and Smeulders, Arnold WM. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

Wang, Xiaoyu, Yang, Ming, Zhu, Shenghuo, and Lin, Yuanqing. Regionlets for generic object detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 17–24, 2013.

Watkins, Christopher H.C. Q-learning. *Machine Learning*, 8:279–292, 1992.

Zeiler, Matthew D. and Fergus, Rob. *Visualizing and Understanding Convolutional Networks*, pp. 818–833. Springer International Publishing, Cham, 2014. doi: 10.1007/978-3-319-10590-$1_5 3. URL$