CS230

# Deep-Q-Network-Facilitated Object Detection

**Xiao Zhou***
SCPD, Stanford University
xiao.zhou@stanford.edu

## Abstract

Object region detection is an essential step for object identification. In this project, the object region detection is modeled as a sliding window game, a Markov Decision Process solvable using the Deep Q-network (DQN). With the downstream object identification neuron network (INN), the DQN-INN system has the potential to do object detection and segmentation.

## 1 Introduction

In general, the objects are detected in two ways: the YOLO way and the sequential way. For the YOLO way, an image is divided by a grid. Each grid cell is responsible for detecting an object and its bounding box [8, 9, 10]. For the sequential way, the objection regions are proposed by a categorical-independent method which involves image segmentation, segment clustering, or ranking, and then these regions are sent to another algorithm for object identification and bounding box prediction. For example, in the R-CNN method, selective search [11] is used for region proposals [2, 1]. Although both methods have been used widely, neither is likely to work well on high resolution images with small fine details, such as pathological tissue images (whose resolutions reach $200,000 \times 10,000$ pixels and objects need to be identified at pixel level): for the YOLO-series algorithms, the object detection capability and its efficiency is restricted by the "resolution" of the grid and anchor boxes; for the R-CNN algorithms, the object region proposing is orthogonal to the object detection, resulting in useless object regions that undermines the efficiency.

In this project, I experimented with a sequential method: (1) using deep Q network (DQN) to propose object regions, (2) these regions are then sent to a object identification neural network (INN) for object identification, (3) the DQN and INN can be co-trained to increase the accuracy of the proposed object regions and the object identification (not yet performed). This is inspired by how human eyes work and my personal experience of identifying cellular structures on biological images. The retina of human eyes has a important region called macula which is responsible for high resolution vision. The retina is the DQN and the INN is the macula, and this method is named rolis for "rolling eyes". This idea is not new. Several studies have explored using reinforcement learning for object detection [5, 7, 12].

DQN usually requires a huge amount of training, especially when the state and action spaces are large. With limited computational resources, this project applies the proposed method on a simplified problem: identifying fixed-size handwritten digits from gray-scale images. Concretely, (1) 0-2 $16 \times 16$ handwritten digits will be drawn on a $80 \times 80$ background image, (2) the DQN detects $16 \times 16$ digit regions and send to INN for identification. You can imagine these images are biological samples: the digits are different cell types.

*LinkedIn: https://www.linkedin.com/in/xiaozhou-1/

Project Milestone of CS230: Deep Learning, Spring 2021, Stanford University, CA.

## 2 Dataset

The following written digit dataset are used: MINST (kaggle), USPS (kaggle) and ARDIS [4]. Each image of these dataset has been (1) converted to 8-bit gray-scale image, (2) filtered cleaned using a 3-by-3 median filter, (3) resized to 16-by-16 image using bicubic, and (4) collected in a single file. The digit classes are balanced, and their counts are 0: 9216, 1: 9906, 2: 8679, 3: 8725, 4: 8436, 5: 7789, 6: 8470, 7: 8845, 8: 8293, 9: 8539. The flickr30k image dataset is used as background. The images were converted to 8-bit gray-scale, center-cropped to $80 \times 80$, and then collected in a single file.

The dataset are split to train, dev, and test by random indexing: the indices are shuffled and divided into these three group in the ratio of 80%, 10%, 10%, respectively. The images are queried by a data Handler, and the gray-scale images containing digits are created by ImageComposers.

## 3 Method and Results

### 3.1 Overview

The structure of rolis is shown in Figure 1. For this project, finding digits in images is modeled as a game: (1) a random cursor box of $16 \times 16$ pixels starting at a random location in the image (see the left inset in Figure 1); (2) the goal of the agent is to move the cursor to the digits and mark them. The marked regions are sent to the INN for identification. The INN not only predicts the digit label but also the pixels belong to the digit (see the right inset in Figure 1).

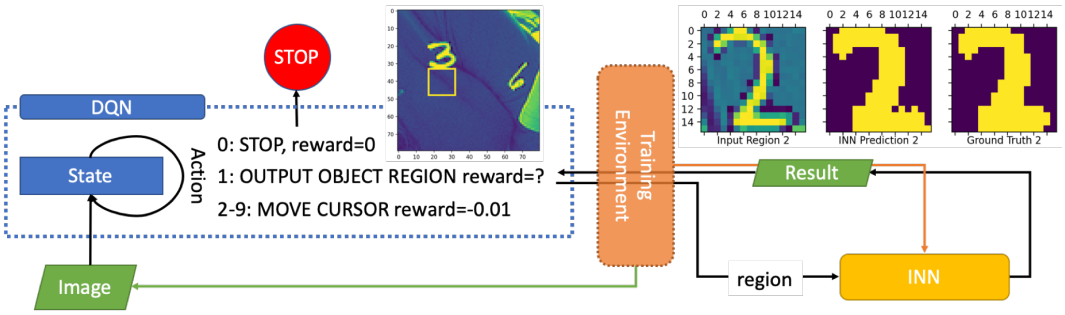


**Figure 1: Overview of rolis.** *The black lines passing through "Training Environment" mean that the data will be handled by the "Training Environment" during training. The green line represent the images are produced by the "Training Environment" during training, and the orange line means that the loss function uses the true label from the "Training Environment" during training.*

The DQN and INN are trained separately at first and then a co-training in the end. **DQN training**: the "Training Environment" generates $80 \times 80$ images containing 0-2 digits of size $16 \times 16$; the output object regions is judged by the environment and the highest Intersection over Union (IoU) with the ground truth is used as a reward. **INN training**: the "Training Environment" generates $16 \times 16$ digit images with backgrounds (or just the background images without digits), the digit labels, and the binary pixel map (indicating which pixel belongs to the digit). Then, the INN is trained in a supervised way using these data. **Co-training** see 4.2.

### 3.2 DQN

I first tried Google's DQN setup, since it has been successfully applied to Atari games [6]. Concretely,

- **state**: a 4-frame history (frames filled with 0s are used to substitute any non-existent history frames, e.g. at the beginning of each episode, 3 frames are non-existent).
- **action**: stop, strike (output the cursor region), and 8 moves: $\leftarrow \uparrow \rightarrow \downarrow \nwarrow \nearrow \searrow \swarrow$.
- **transition**: "stop" ends the episode; "strike" paints the whole cursor box region with 0.5 and outputs the region to the environment for judgement; each of the 8 directions moves the cursor box with stride 1, unless moving against boarders in which case the stride is 0.

- **reward**: "stop" receives 0 reward; "strike" receives an IoU as a reward, if the output region has a highest IoU>2/3 with a the ground-truth region. That ground-truth region also become unavailable. Otherwise, -0.01 as a reward. Each directiona move has a reward of -0.01.
- **network architecture**: same as Google's DQN [6].
- **loss function**: a MSE loss, same as Google's DQN

$$\mathcal{L} = (y_t - Q(s_t, a_t; w))^2 \quad \text{where } y_t = \begin{cases} r, & \text{episode ends at } t+1 \\ r + \gamma max_{a'} Q(s_{t+1}, a'; w^-), & \text{otherwise} \end{cases}$$

This setup was tried on images containing 0-5 digits with random backgrounds. However, after 4 million steps, it was hard to tell whether the model is learning anything. Therefore, images containing 1 digit with no background were used instead, and several modifications was also made to improve performance:

- **state**: 1-frame history is used. Unlike Atari games which usually have moving characters at any moment, in this problem, everything is stationary unless a moving action is taken. Therefore, >1 frames only unnecessarily enlarge the state space.
- **network architecture**: ResNet-18 is modified and used in this problem. It is deeper than Google's DQN and has been designed for image classification.
- **loss function**: Smooth L1 Loss is used to reduce sensitivity to outliers. Meanwhile, double DQN is implemented to reduce bias. (Prioritized Experience Replay was also implemented, but the result was worse than before, suggesting a bug in the code, and it was then removed.)

**Result** After training for 4 million steps using the improved model, in 6 out of 10 evaluations, the cursor was moved in close proximity to the digits, confirming the model was working. However, the agent hardly made the strike move. Therefore, the model was trained for another 4 million steps. Although 7 out of 10 cases showing the cursor being moved to the digits, no strike moves were made. An example is shown in Supplemental Material 5.1. This suggests that this DQN model probably needs far more training steps to converge to an optimal solution. To make the Q-function converge fast, I experimented with a supervised learning method (see 4.1).

### 3.3 INN

The INN is designed to not only identify the digits (or a "non-digit"), but also identify the pixels that belong to the digits. Inspired by YOLO, the $16x16$ input image, after the INN network, outputs (1) a 11-element vector $c$ representing the likelihood of the 11 digit classes (0-9 and non-digit), and (2) a $16x16$ matrix $S$ probability map indicating how likely the pixel belongs to the predicted digit. The architecture is shown in Figure 2 which use a modified ResNet block shown in the inset.

The loss function is comprised of two parts: a cross entropy loss measuring the digit prediction and a hinge loss measuring the belonging prediction. Formally,

$$\mathcal{L} = -\sum_{j=0}^{10} c_j \log \frac{e^{\hat{c}_j}}{\sum_{i=0}^{10} e^{\hat{c}_i}} + \begin{cases} \mathbb{1}_{\text{argmax}(\hat{c})\neq 10} \max(0, 1 - \hat{S}S) & \text{argmax}(\hat{c}) = \text{argmax}(c) \\ 2 & \text{argmax}(\hat{c}) \neq \text{argmax}(c) \end{cases} \quad (1)$$

where, $c$ is a one-hot coding vector for the ground truth class, $\hat{S} \in [-1, 1]$ (tanh activation), $S \in \{-1, 1\}$ (1 for yes, -1 for no), $\mathbb{1}_{\text{argmax}(\hat{c})\neq 10}$ means that the predicted class is a digit.

The reason hinge loss is used is that it is bounded between 0 and 2. The cross entropy loss $\in [0, \infty)$. When the class is predicted correctly, the cross entropy loss $\leq -\log(\frac{1}{11}) \approx 2.4$. Therefore, the model prefers to make the class prediction correct first, and then make the belonging prediction accurate. In this way, there is no need to use a hyper-parameter to balance the two losses.

**Metric** Since both the digit classes and the belonging classes are well balanced, accuracy was used as metric to measure the performance. I tested my digit classification accuracy on 100 samples of dev data, and the result is 95% which was used as a reference.

**Result** One million iterations were performed using a batch size of 32 and a learning rate starting of 1e-3. The learning rate was linearly decreased to 1e-5 at 500,000th iteration. The final INN reached 96% accuracy of digit classification, and 95% accuracy of belonging classification on 32 samples of

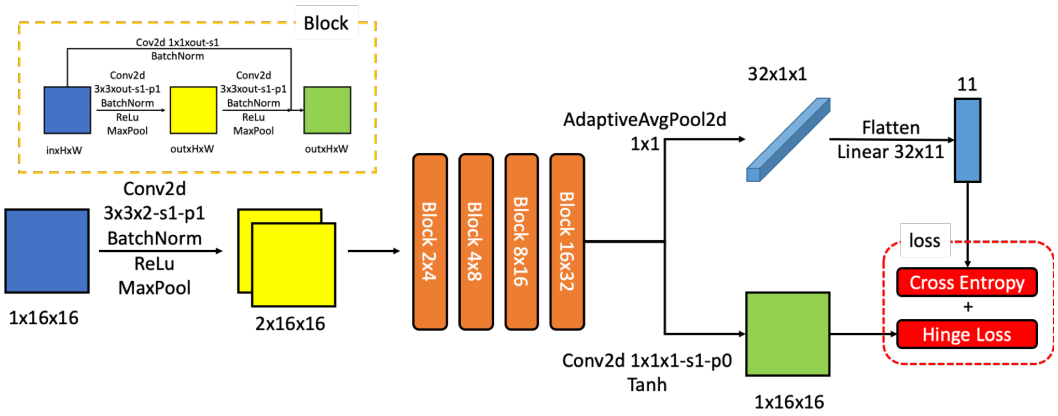


**Figure 2: Network Architecture of INN.** *The text under the colored shape represent the shape of the data. For example, 1x16x16 represents 1 layer of 16x16 matrix. The texts along arrows represent the configuration of that part of the network. For example, Conv2d 3x3x2-s1-p1 BatchNorm Relu MaxPool means Conv2d with two 3x3 kernel, stride 1 and padding 1, followed by BatchNorm, ReLu and MaxPool. AdaptiveAvgPool2d 1x1 is an average pooling which takes an mxWxH input and output mx1x1 data.*

either "train" or "dev" data, suggesting that the model has been well trained. On 10000 "test" data, the final INN also achieved 96% accuracy of digit classification, and 95% accuracy of belonging classification. Four examples are shown in Figure 3, showing the accuracy of the model's prediction.

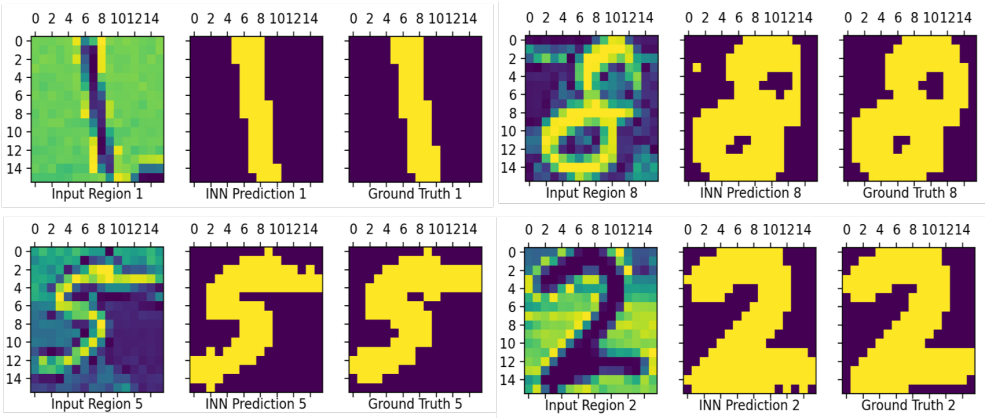


**Figure 3: Results of INN.** *Four typical examples of INN's prediction. For each panel, left image: input image; middle image: INN prediction; right image: ground truth.* The images are all gray-scale. The color is matploblib viridis color gradient: 0-dark, 0.5-green, 1-yellow.

## 4 Discussion and Future Plans

### 4.1 Supervised Learning of Q-function

In Q-Learning, the agent needs to explore the environment and update the Q-function according to the experience. Hopefully, the Q-function will converge to an optimal solution, i.e. converge to $Q^{\pi^*}(s,a)$. In contrast, in this digit finding game, (1) an optimal policy $\pi^*$ can be deduced: find the shortest path connecting the digits and strike them, and (2) the $V^{\pi^*}(s)$ (the value of a state under the optimal policy) can be calculated. Therefore, instead of using Q-Learning, given $(s,a,r,s')$, the $Q^{\pi^*}(s,a)$ can be directly calculated by

$$Q^{\pi^*}(s,a) = r + \gamma V^{\pi^*}(s')$$

Then, the $\hat{Q}^{\pi^*}(s, a; w)$ approximated by the neuron network can be learned by supervised learning. Due to the limitation of time, I directly tried on images containing 0-2 digits with random background. However, the model still behaves similar to DQN: it is obvious that the model is learning to move the cursor to the digits, but it fails to strike the digits or stop on time. If this problem persists with more training iterations, the possible causes are:

- the strike and stop actions are rare compare to the moving actions in the replay buffer. This is similar to the "activation word" problem discussed in this course. This can be solved by prioritized experienced replay.
- Accumulated errors. The policy is derived from $\text{argmax}_a \hat{Q}(s, a; w)$. The $\max$ operation leads to bias, and the bias accumulated step by step, especially when the error of $\hat{Q}(s, a; w)$ is high and when the episode is long. This may be remedied by a depth-first tree search for future $n$ steps. At each step take the highest $k$ actions from $\hat{Q}(s_t, a; w)$ and track the sum of $\hat{Q}$ values of each path. At the current step, choose the action has the highest sum. The intuition is that the negative and positive error will cancel out by the sum.
- ResNet-18 is insufficient to generalized the finding digit problem. Besides the confusing background, the optimal solution requires finding the shortest path to visit all the digits. This is the Travelling Salesman Problem, and it is also the reason that I choose 2 as the maximum number of digits (it can be easily solved using greedy). May be use a deeper network would help.

The finding digit game is a delayed reward problem which is hard for DQN. I plan to examine the above possibilities and try to make the DQN work. It would also be interesting to see how the neural network handles the Travelling Salesman Problem. Another possibility is to replace the DQN with a recurrent neuron network as suggested by Ayush Kanodia.

### 4.2 Co-train DQN and INN

Co-training, which has not been performed yet, is to let the two network learn each other's data distribution. For example, the the digit 1 of DQN output images may be not positioned at the center, and the INN should learn to identify these images. Meanwhile, DQN should learn to position digits at the center for easy identification. A well-trained DQN may hardly output non-digit regions, and DQN should learn this distribution. In co-training, the output regions are directly used as input for INN. The environment still supervise the learning of INN with digit label and belonging map. The strike reward for DQN can be

$$\mathbb{1}\{\text{output region is a digit}\}\mathbb{1} \cdot \{\text{INN correctly identified the digit}\}$$

If GAN can generate good results, two cooperative networks are also likely to perform well.

### 4.3 Compare with YOLO

YOLO uses one network to do both object detection and identification. It has the advantage of being fast and compact, because (1) the neurons are shared between object detection and identification, and (2) the world is fractal: neurons work on big scale can be reused on small scale. Although I have not made rolis work yet, assuming it works, the advantage of rolis is:

- the number of objects to detect is not constraint by the grid and anchor box number as in YOLO.
- the DQN is a reinforcement learning method which can self-adjust to data drift.
- the downstream INN can be more powerful. As demonstrated here, it can do both segmentation and identification
- the separated DQN and INN may scarify speed, but distributed computing can be naturally applied. A DQN-powered camera can send small images back to powerful cloud for INN to compute complex features.
- can be used on robotic vision. For the finding digits problem, imagine the cursor is always at the center. When the agent perform move actions, it actually moves a camera. Such idea has been studied in [3].

## References


[1] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[3] X. Han, H. Liu, F. Sun, and D. Yang. Active object detection using double dqn and prioritized experience replay. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–7. IEEE, 2018.

[4] H. Kusetogullari, A. Yavariabdi, A. Cheddad, H. Grahn, and J. Hall. Ardis: a swedish historical handwritten digit dataset. *Neural Computing and Applications*, pages 1–14, 2019.

[5] S. Mathe, A. Pirinen, and C. Sminchisescu. Reinforcement learning for visual object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[7] A. Pirinen and C. Sminchisescu. Deep reinforcement learning of region proposal networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[9] J. Redmon and A. Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[10] J. Redmon and A. Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[11] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[12] B. Uzkent, C. Yeh, and S. Ermon. Efficient object detection in large images using deep reinforcement learning. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, March 2020.

# 5 Supplemental Material

## 5.1 Result of DQN

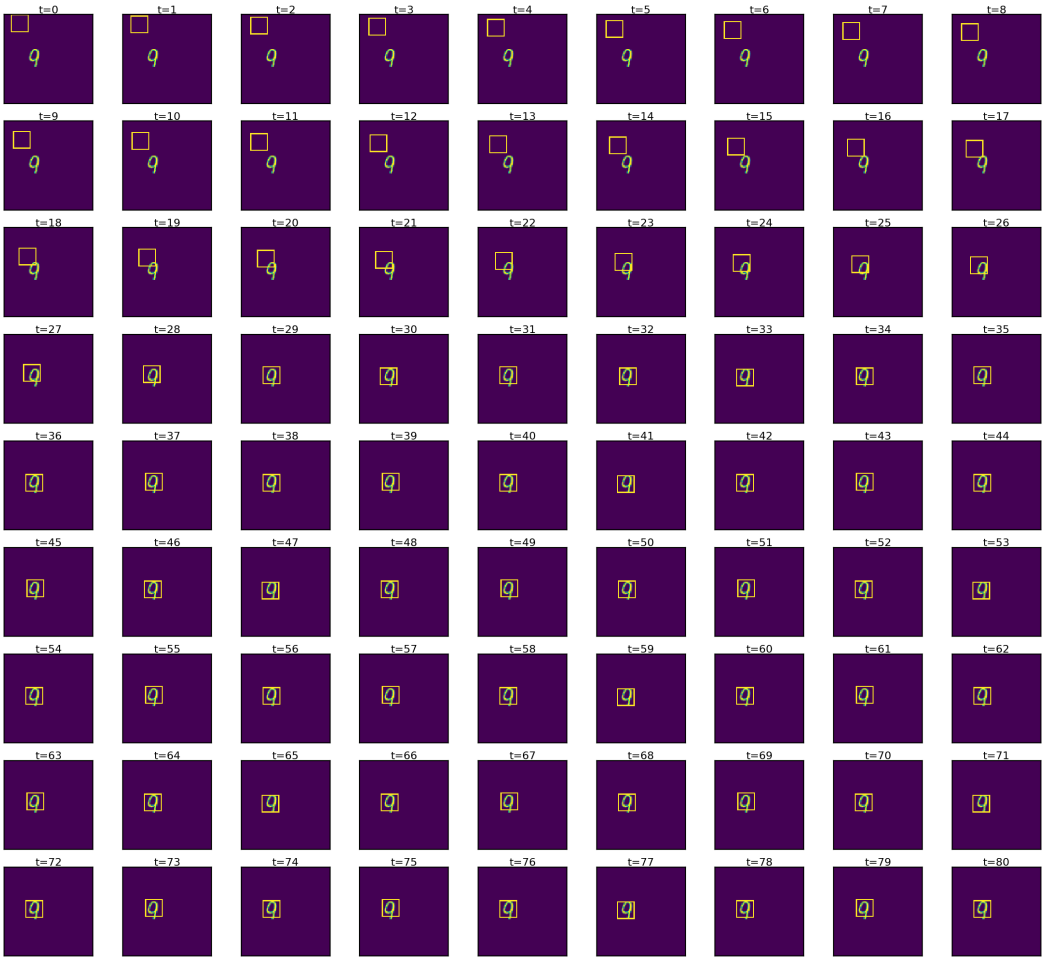


**Figure 4: Promising Example of DQN's Result.** *The result is after 8 million training steps using ResNet-18 and double DQN. The cursor was moved to the digit quickly, however no strike action was made. There are total 161 steps and only the first 81 on are shown. In the rest steps, the cursor is jiggling around the digit.*