University of Glasgow | School of Computing Science

Object localization using deep reinforcement learning

Mohammad Otoofi

School of Computing Science
Sir Alwyn Williams Building
University of Glasgow
G12 8QQ

A dissertation presented in part fulfilment of the requirements of the Degree of Master of Science at The University of Glasgow

September 8 2018

Abstract

The aim of this project is to learn a policy to localize objects in images by turning visual attention to the salient parts of images. In order to achieve this goal, the popular Reinforcement Learning (RL) algorithm, Q-learning, is adopted by incorporating the approximation method, Convolutional Neural Networks (CNN)s. Deep Q-Learning (DQL) [1] is the method resulting from cooperating Q-learning and CNNs. While using this method for object localization is not new and was tried before in [2], in this project despite implementing the algorithm with the novel deep learning framework, Tensorflow, a new set of experiments were conducted by using a new neural network architecture to show that representation learning can happen by Q-learning. More specifically, the original paper uses a pre-trained CNN as a feature extractor. However, in this project, the model was trained without using a pre-trained network for feature extraction. Two sets of experiments were conducted in this project. In the first one a model was trained with objects from the same super-category. It showed that the network is able to generalize information within a super-category. Furthermore, a new class form different super-category was added. It was determined that the network is able to learn high level conceptual information between two super-categories in order to detect the locations in images where there is an object, regardless of the type of object. In the second experiment, a model is trained using a shuffled data from all categories that achieved a smaller precision than the incremental model. Finally, it was demonstrated that using smaller dataset, one can achieve results near to the ones reported in the original paper.

Education Use Consent

Name: Mohammad Otoofi Signature:

Acknowledgements

This is the culmination of a very intensive but fruitful year. First of all, I am especially grateful to Dr. Jan Paul Siebert for giving me the opportunity to participate in this rewarding project. I would also like to thank him for his very valuable supervision and all the advise he provided me.

Finally, thanks to my family for their unconditional support, love and motivation. None of this would have been possible without them.

Contents

Chapter 1

Introduction

1.1 Overview

The aim of this project is to design a localization algorithm which does not need to process an entire scene to localize objects in order to reduce computation cost. For this, a method inspired by the human gaze vision system was applied. The basic idea is to localize objects in a scene by finding dense regions in the scene and guide the visual attention to that part. To achieve this goal, the problem is formulated as a sequence of decision making tasks. Formulating the problem as a decision making task led to applying a variant of RL [3] algorithms called DQL [1] to solve it. Using Deep Learning (DL) and Q-learning, an attempt was made to learn representation for objects from different categories and guide an intelligent agent to focus on an object in an image. This method was used previously in [2]. However, in [2] , a large base network trained on a separate dataset is used as a feature extractor. In this project, despite using a new neural network architecture and implementing the algorithm with the novel DL framework Tensorflow, it was attempted to train a model without using a feature extractor with an end-to-end training approach.

One of the domains in computer vision is object detection and localization. The aim here is to find and localize objects in a scene or in an image. This is a crucial task used in a variety of important applications like self-driving cars where it is required to see and track pedestrians, cars, motorcycles, trees and other objects in the scene, or in robotics where a robot arm needs to localize and pick up an object among many items on a table.

Popular algorithms in object detection have mostly concentrated on methods where localization and detection are conducted separately. The basic idea of all these methods is first to localize objects in an image and then classify them. While the main idea is similar, different approaches have been used to increase efficiency and effectiveness [4, 5, 6]. However, recent methods have emphasised the approaches where end-to-end learning is applied. In this way both localization and classification can be done with a single pass through an image. These models merge the two basic steps into one model to conduct localization and classification simultaneously [7, 8, 9]. However, to find objects in an image,these algorithms cover every sub-region in an image. Unlike these algorithms, where every patch in an image is processed, human vision doesn't search for objects in each region. It is shown in [10] that the human vision system localizes objects by perceiving the whole scene and successively searching dense regions by sequentially turning visual attention to the important local areas.

In contrast to the common algorithms for object localization, in this project the attention is focused on the methods inspired by the human vision system. To achieve this goal, the object localization problem is formulated as a control problem with a sequence of actions in order to tighten a boundary box around a target object. This project follows an active object localization method based on the human vision system proposed by C. Caicedo et al.[2] that follows a top-down search in an image. This method processes the whole image at the first stage and narrows down the exact location of an object. This requires designing an intelligent agent that can learn a policy for finding objects. The agent sees the environment, i.e. images, through a window. Initially, the window covers the whole scene. The agent tries to localize an object by performing a sequence of transformations on the initial window. The agent learns to decide what transformation is the best, given the current visible window. The agent needs to learn patterns and objects representation in order to be able to discriminate background from objects and to focus on the target. Given the complexity of the problem, i.e. the number of state growing exponentially and the environments with which the agent interacts are images or raw pixel values, the method used to tackle the problem is DQL.

However, C. Caicedo et al. uses a large pre-trained CNN trained on a separate dataset as a feature extractor. This imposes a computation cost and makes the method dependent on another dataset. This project aims to take further steps and to conduct experiments without using a pre-trained model. The idea is to test whether the agent is able to learn the representation of objects independently. Moreover, this method was proposed back in 2015 when Caffe was the common deep learning framework. One of this project's contributions is to implement the algorithm using the popular deep learning framework Tensorflow.

1.2 Outlines

The following chapters are organized in this way: Chapter 2 is dedicated to introducing recent advances in object detection algorithms since the advent of DL. It is divided into two parts. The first part addresses the object detection algorithms that conduct object detection and localization jointly. The second part deals with active object localization algorithms applied RL to simulate human vision system. Also, a brief introduction to RL is given in that chapter. Furthermore, in the first section of chapter 3, a theoretical framework for the method used in this project is explained and then the architecture of the neural network and tricks that were applied to implement the algorithm are described. At the end of chapter 3 the experiments conducted, along with their results are reported. Finally, in chapter 4 a summary of the project and its results are given. While the time of this project was limited, there are many ideas left to be the subjects of experiment as future works to extend this project. They are mentioned at the end of the final chapter.

Chapter 2

Background

Object localization algorithms consist of two groups: the algorithms that conduct object localization and detection jointly and the algorithms that mainly focus on object localization. In this section, first object detection algorithms, which perform object localization and detection jointly, are reviewed. The common characteristic between these categories of algorithms is that they have to process all sub-regions of an image in order to detect objects. In other words, these algorithms have to analyze large amounts of region proposals which has caused them to slow down. In the second part, the methods based on RL algorithms are explained. Before describing these algorithms, an introduction to RL is given. The background of RL based object localization algorithms comes from the human vision system that finds objects in a scene with a top-down search and turning visual attention to the densest part of the scene [10].

2.1 Object detection methods

In this section, the main subject is the recent object detection algorithms that apply deep learning methods to overcome the problem. Each method has used deep learning in a different way to enhance effectiveness and efficiency. As the methods evolve, they move closer to the idea of end-to-end learning. Since the subject of this project is object localization, the focus is mostly on the localization part of these methods.

2.1.1 Region-based Convolutional Network

Region-based Convolutional Network (R-CNN) [5] is the first work that applied the deep learning method in object detection problems. The main idea of the paper is that the algorithm finds all objects in an image using an exhaustive search algorithm and then classifies the proposed objects using CNN. The search algorithm to localize objects in an image is called Selective Search [9]. This search algorithm was designed to localize objects in images. Selective Search algorithm is able to deal with a variety of image conditions. The basis of Selective Search algorithm is hierarchical grouping algorithms. Using bottom-up grouping, Selective Search algorithm is able to generate locations of objects at all scales. The grouping process continues until the whole image becomes a single region. The detected regions then are further processed using a variety of colour spaces with different invariance properties, different similarity measures, and by varying starting regions. The output of the selective search algorithm is a set of region proposals which may contain an object. The R-CNN model combines selective search and CNN methods to localize and classify objects. R-CNN consists of three modules. The first one generates a set of proposal regions using selective search. The second is a CNN to extract a fixed-length 4096-dimensional feature vector from each region. The third module is a set of linear SVM classifiers whose input is the feature vector and its output is the probability of belonging to an object category. The architecture of R-CNN is shown in the fig 2.1.

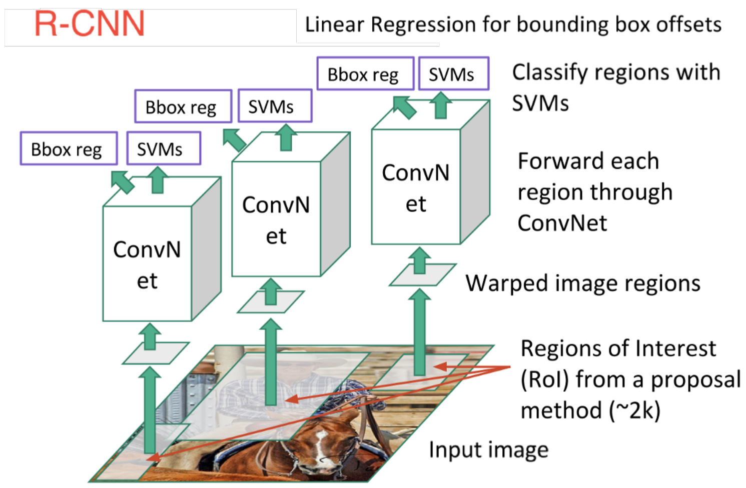

R-CNN
Linear Regression for bounding box offsets
Bbox reg
SVMs
Bbox reg
SVMs
Bbox reg
SVMs
Classify regions with SVMs
ConvNet
ConvNet
ConvNet
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)
Input image

Figure 2.1: Region-based Convolution Network (R-CNN). Source: J. Xus Blog

Fast R-CNN

Fast R-CNN [5] is a variant of R-CNN aimed to speed up object detection. R-CNN suffers from three major drawbacks. The first is that the algorithm consists of multiple stages which are learnt and tuned separately. The second is training time. It is reported that for 5K images of the VOC 2007 training takes 2.5 GPU-days [5]. The last problem is at testing time, where it is needed to perform in real time applications however, each image takes 47s to be processed [5]. Fast R-CNN is designed to reduce the amount of computation and memory needed by R-CNN by using a multi-task loss to train the whole network in a single pass and update all network layers. Fast R-CNN takes the whole image as input and feeds it to the main CNN. Using several convolutions and pooling layers, a feature vector is extracted from the input. The feature vector is consumed by a Region Of Interest (RoI) pooling layer to extract a fixed-length feature vector for each object proposal. The selective search method is applied to find RoI regions. Each feature vector is then flattened to be fed into fully connected layers which finally generate two sibling output layers. The first one is a one-hot-encoding vector passed through a softmax layer to indicate the probability of belonging to K object classes for each proposal object. The second output is a four-real valued vector for each of the K object classes which encodes the coordinates of predicted bounding boxes for the objects detected.

Faster R-CNN

Faster R-CNN [6] is designed to replace the selective search algorithm used in the previous versions of R-CNN. The problem with the selective search is that it is computationally expensive. While Fast R-CNN has introduced new features to reduce training and testing time, the selective search was still a bottleneck for R-CNN algorithms. In faster R-CNN a new network called Region Proposal Network (RPN) [6] has been introduced to replace the selective search algorithm. This network aims to propose regions which are later used by Fast R-CNN network to predict bounding boxes and to detect objects. RPN uses a pre-trained model trained over ImageNet dataset for classification. Specifically, firstly the RPN network, which is a deep convolutional network that proposes regions, takes an image as input and as output produces a feature map. The feature map is then used by a small network. The small network takes as input a $n \times n$ sliding window over the feature map. The output of the small network is a sibling output, a box-regression layer and a box-classification layer. Over each window location the small network predicts multiple region proposals. The number of region proposals is set by a parameter called K. The K proposed regions determine the number of reference boxes applied to all window locations to create region proposals. These boxes have different scales and aspect ratios to capture all possible objects at current sliding position, and they are called anchors. In this way, for every sliding window there is a K anchor. Using anchors, Faster R-CNN can address multiple scales and aspect ratios. The box-classification layer outputs a vector of probabilities which show an objectness score for each anchor box. Detected anchor boxes then are selected based on the objectness score. The anchor boxes exceed a predefined threshold then are fed to Fast R-CNN. It is noteworthy that Faster R-CNN merges RPN network with Fast R-CNN by using a so-called “attention mechanism” [11]. The RPN network guides Fast R-CNN network where to look. To share the computation, convolutional features are shared between RPN and Fast R-CNN. The rest of algorithm is similar to Fast R-CNN.

Mask Region-based Convolutional Network

Mask Region-based Convolutional Network (Mask R-CNN) [12] extends Faster R-CNN by adding a new part to the bounding box recognition in order to predict an object mask. Mask R-CNN uses the same two stages as Faster R-CNN applies. In the first stage Mask R-CNN adopts an identical RPN architecture however, the second stage comes with an extension to the original Faster R-CNN. In the second stage, in addition to bounding boxes and class predictions Mask R-CNN produces a binary mask for each RoI. Mask representation is used to demonstrate the object’s spacial layout. Furthermore, the RoI pooling layer in Faster R-CNN which was initially inherited from Fast R-CNN is also replaced by a RoIAlign layer. The problem with a traditional RoI pooling layer is that it quantifies a floating-number RoI which causes misunderstanding between the RoI and the extracted features. This quantification doesn’t affect classification but has significant negative impact on predicting pixel-accurate masks. The idea of adding a pixel mask to the algorithm means the segmentation task improves the localization and thus the classification. To achieve this goal three loss functions corresponding to each task are defined and totaled. The totaled error is then used to optimize and train the network.

2.1.2 Region-based Fully Convolutional Network

While R-CNN multi-stage algorithms consist of detecting region proposals and recognizing an object in each region, Region-based Fully Convolutional Network (R-FCN) [7] is a model with only convolutional layers which can be trained end-to-end with back-propagation. This brings the advantage of using Residual Nets (ResNets) for the first part of the algorithm which detects objects. The difference of R-FCN with previous algorithms is that every part of the algorithm can be trained as one network end-to-end while in the past, each stage of algorithms consisted of different parts and were trained separately. This model can take into consideration simultaneously the object detection and object localization. R-FCN applies RPN network to extract proposal regions (RoIs). The last layer of RPN outputs the feature maps, each of which detects a category of an object in a specific location. Having extracted the region proposals, R-FCN classifies the regions into object categories and backgrounds. The last layer of R-FCN produces position-sensitive score maps which vote for the location of objects in an image. At the end, a position-sensitive RoI pooling layer is used to aggregate the output of the last convolutional layer and produces scores for each RoI region. The position-sensitive RoI pooling layer conducts a selective pooling and returns a score from bank of $K \times K$ score maps. The whole process is shown in the fig 2.2.

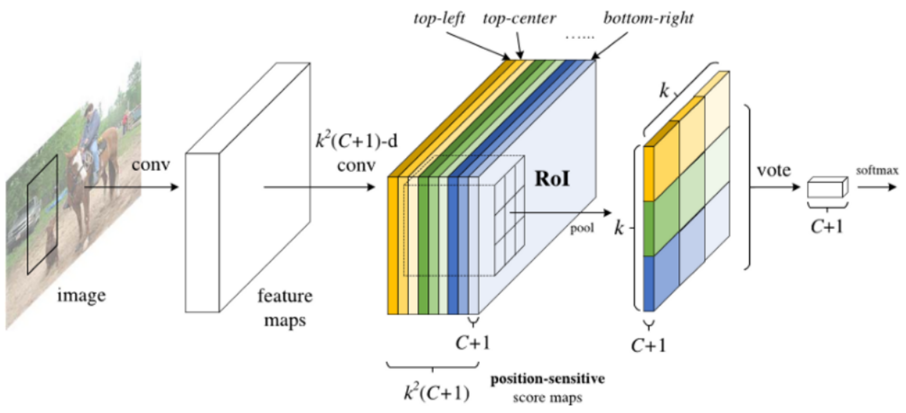


Figure 2.2: The input image is fed to a ResNet model and then its output is used by RPN model to detect RoI and compute probabilities corresponding to every object classes. Source: J. Dai and al. (2016)

2.1.3 You Only Look Once

You Only Look Once (YOLO) [8] is a model which consists of a single neural network which can be trained end-to-end by back-propagation. It merges the two steps of previous algorithms, object detection and localization, into one model. However, YOLO formulates the object detection problem differently, as a regression task which spatially separates bounding boxes and associates class probabilities. The network is trained to learn very general representations of objects. It takes as input the entire image and predicts the bounding boxes across all classes for an image simultaneously. The image is divided into $S \times S$ grid and then B bounding boxes and confidence scores are predicted for those boxes. If an object's centre falls within a grid cell, the grid cell will predict B bounding boxes along with a confidence score. To compute the confidence score, Intersection-over-Union (IoU) is needed to be computed which is difference between the predicted box and the ground truth. In this way, the confidence score which is $Pr(Object) \times$ IoU can be computed. The advantage of YOLO over other methods is its speed. It is so fast in comparison to other methods and this makes it suitable for real time applications. However, this comes with a trade-off in accuracy. YOLO makes more localization errors. The network architecture is shown in the fig 2.3.

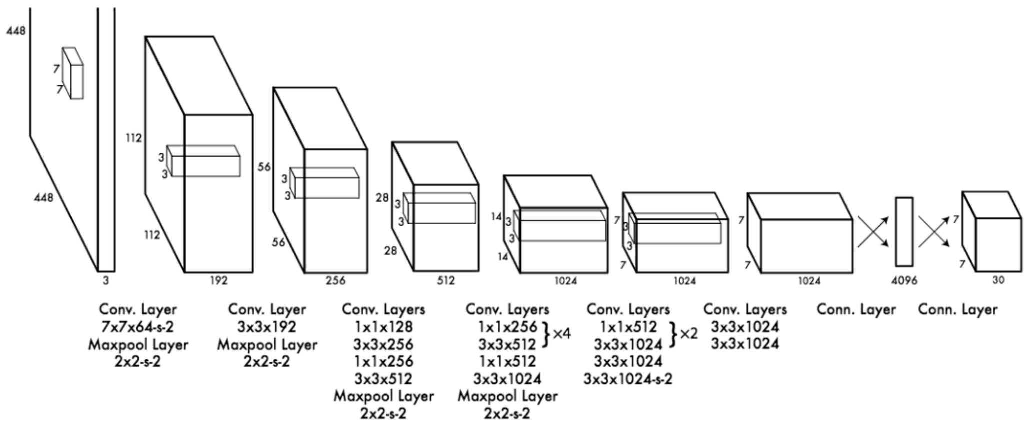


Figure 2.3: YOLO architecture consists of 24 convolutional layers and 2 fully-connected layers. Source: J. Redmon and al. (2016)

2.1.4 Single-Shot Detector

While the previous approaches are accurate, they are too slow and computationally expensive for embedded systems and real-time applications. In these cases, often speed comes at the cost of accuracy. To improve speed, Single-Shot Detector (SSD) [9] eliminated the region proposal step from the object detection pipeline. Thus, the neural network doesn't need to re-sample features to produce bounding boxes hypotheses. Using a convolutional filter, SSD predicts object categories and offsets in bounding box locations. In addition, another convolutional filter is used to perform object detection at different scales. The filters are applied over the feature maps of the first part of the neural network. This leads to a faster and more accurate algorithm than the previous ones [9]. Similar to YOLO and R-FCN, SSD proposes a model which consists of a single convolutional neural network that can be trained end-to-end. More specifically, SSD is based on a feed forward convolutional neural network which outputs a set of bounding boxes and scores for the presence of object classes. The first part of the neural network called base network follows a standard architecture (VGG-16 architecture [13]) and is responsible for feature extraction. The second part of the network produces a set of predictions. These predictions consist of predicted bounding boxes coordinates including the coordinates of the center, the width and the height of the box. Besides that, the network outputs a vector of probabilities related to confidence over each class of object. In addition, two other methods are used in training time. To keep the most relevant boxes, a method called Non-Maximum Suppression is used and then the output of that is consumed by the Hard Negative Mining method. Hard Negative Mining lists the negative predicted boxes by the confidence score and selects a subset of them to be used for computing the error. This is because lots of negative boxes are being predicted during training and could have a destructive effect in training the network. SSD network architecture is presented in the fig 2.4.

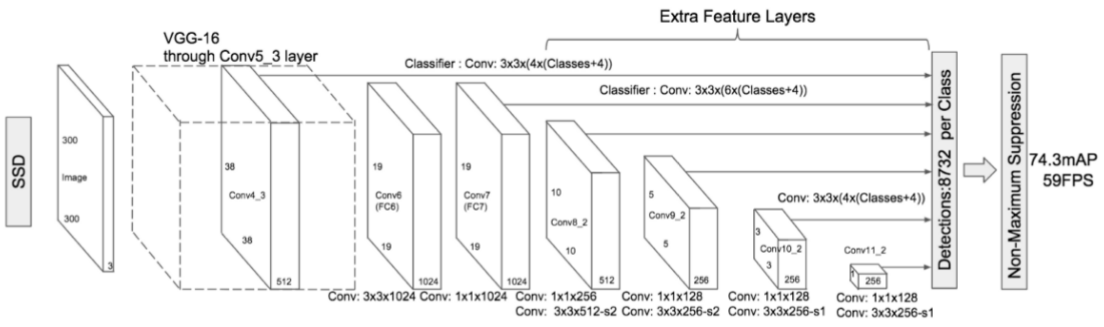


Figure 2.4: SSD model applies several feature layers to the feature maps produced by the base network to increase the number of relevant bounding boxes. Source: W. Liu and al. (2016)

2.1.5 Neural Architecture Search Net

Neural Architecture Search Net (NASNet) [14] follows a totally different paradigm for object detection. The idea of NASNet is to learn the architecture of neural networks. While researchers spend a lot of time designing an optimized neural network architecture, NASNet applies a Recurrent Neural Network (RNN) which can learn an optimized architecture for a specific goal over a dataset. NASNet learns how many layers, filters, and neurons are appropriate for a given problem. The network is also capable of learning small details of a network like strides height, strides width, and filter size in the case of CNNs. More specifically, NASNet is trained to use RL algorithm and the accuracy over a given dataset is used as a signal to train NASNet. The authors of [14] have created a neural network architecture learned using NASNet on CIFAR-10 and then trained the network on 2012 ImageNet. This model was later used as feature maps generator and stacked with Faster R-CNN. Finally, the entire pipeline were retrained with the COCO dataset.

2.2 Object localization methods

In this part, the focus is mainly on object localization methods using RL. Therefore, before explaining these methods RL context will be briefly introduced. It is noteworthy to emphasize that unlike previous explained methods these algorithms just concentrate on localizing objects and not actually detecting them. This project is based on the work "Active Object Localization with Deep Reinforcement Learning" explained in the following sections.

2.2.1 Reinforcement learning

RL is considered as a mathematical framework for experience-driven autonomous learning. Using RL, an agent is able to learn the structure of an environment by trial and error over time to achieve its goal(s) [15]. RL challenges the idea of machine learning approaches which assume there always should be a teacher (annotated data) to learn a task. Instead of having annotated data, RL provides a framework that enables an agent to learn a task by maximizing long-term rewards through interacting with an incompletely-known environment. Based on the consequence of interactions with the environment, the agent learns how to change its behaviour in response to received rewards.

In RL context, an agent observes state S_t at time step t. It interacts with the environment by taking action a from a set of predefined actions A. Although, a variant of the RL framework also supports continuing action space. Having taken action a, the environment brings the agent to a new state S_{t+1} at time step $t + 1$ based on a transition function. Also, the environment gives feedback r_{t+1} to the agent by a reward function to show the agent how good the taken action was. This process is shown in fig 2.5. Each state consists of sufficient information about the environment so that the agent can decide what action is best to take in a given state [16]. The goal of the agent is to learn a policy π that maximizes the expected discounted reward. A policy is a map from state space to action space. Optimal policy refers to a map that returns a sequence of actions that maximizes the expected reward [3].

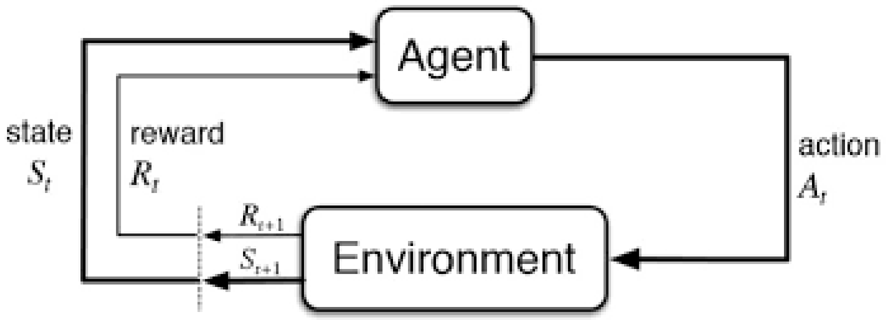


Figure 2.5: The perception-action-learning loop. Source: Reinforcement Learning: An Introduction 2nd Edition

Theoretical framework

RL algorithms are defined by Markov Decision Process (MDP) framework. According to MDP definition, it provides a mathematical framework for a sequence decision-making process where outcomes are partly random and partly under the control of the agent. MDP is defined for problems where there are a finite set of states and actions and the environment is fully observable.

A MDP is a tuple $\langle S, A, P, R, \gamma \rangle$:

- S is a finite set of states.
- A is a finite set of actions.
- P is a state transition probability matrix, $P^a_{ss'} = P[S_{t+1} = s' | S_t = s, A_t = a]$.
- R is a reward function, $R^a_s = E[R_{t+1} | S_t = s, A_t = a]$.
- γ is a discount factor $\gamma \in [0, 1]$.

Using MDP definition, a policy is a mapping from state space to probability distribution over actions $\pi : S \to p(A = a|s)$. Pursuant to the definition above, the probability of moving to a new state S' is conditionally dependent on current state s and the taken action a, but is independent from past states and actions. It means given s and a, the transition probability is conditionally independent of all previous states and actions which implies an MDP satisfies the Markov property [16, 3].

There are two variants of MDP. The first one is episodic MDP where the agent will be brought to the initial state after each episode with length T ends. The cumulative reward in this case is computed as $R = \Sigma_{t=0}^{T-1} \gamma^t r^{t+1}$ where γ gives higher weight to the rewards received earlier in an episode, and in this way the agent is encouraged to take the minimum number of actions in order to achieve its goal. The second variant is for infinite episode where $T = \infty$ [3]. However, in the context of object localization episodic MDP is used. Each image is considered as a new environment and the agent has several episodes (it is a design choice) to learn from each image. The final goal is to learn an optimal policy that returns the maximum expected return from all states:

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R|\pi] \tag{2.1}$$

Solving a MDP

RL algorithms are generally divided into two groups of tabular and approximate methods. Tabular methods are used in the situations where an agent needs to act in a finite state space. Within these sorts of problems it is computationally feasible to maintain a map between states and their corresponding values. Tabular solutions consist of Dynamic Programming (DP), Monte Carlo (MC) methods, and Temporal-Difference (TD) learning [3]. In contrast to DP, MC and TD methods, model-free algorithms can solve an MDP without needing to have any knowledge of transition and reward functions. They also can solve an MDP without needing to learn a model of the environment. Learning a model of the environment can help with planning, however it is computationally expensive. This has made model-free RL algorithms popular in the literature. Q-learning, which is popular due to its simplicity, is a model-free RL algorithm which is the foundation of many state-of-the-art methods [17, 1, 18, 19]. It learns the action-values in a given state. In fact, Q-learning is an off-policy TD control algorithm. After every step Q-values are updated using the formula below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{2.2}$$

In this way, by learning Q function, optimal action-value function can be obtained independent of the policy being followed. The reason why it is independent of current policy is Q-learning updates Q-values using the Q-value of the next state s' and the greedy action a'. The statement 2.2 indicates that state-action values are updated based on the previous computed Q-values and the actual Q-values faced after taking an action.

Q-learning can be used in two ways depending on the application domain: tabular or approximate solution. Tabular Q-learning needs to know the state space in advance. This method is sufficient for simple problems with small state space. However, in many real world problems the number of states grows exponentially and it is no longer possible to maintain action-values [16]. In these situations approximate solutions are applied. The goal in approximation solution is to learn a parameterized function which can predict Q-value for each action in every state.

Deep Q-Learning

The current state-of-the-art non-linear approximation method is neural networks, i.e. DL [20]. DL is able to learn representation of inputs from raw high dimensional data, e.g. camera inputs. Advantage of neural networks is their ability to generalize from past experiences to unseen situations. DQL [1], which combines Q-learning and DL, applies neural networks to approximate Q values. DQL is showed that can learn control policy for a range of tasks with minimum prior knowledge [1]. In general, Deep Reinforcement Learning (DRL) refers to the methods that use neural networks coupled to RL algorithms.

2.2.2 Tree-Structured Reinforcement Learning for Sequential Object Localization

Using a tree structured search strategy, a new method called Tree-structured Reinforcement Learning (Tree-RL) [21] was proposed to localize multiple objects in an image. Tree-RL is a method based on tree search where at each step using RL decides how the tree search should be divided. The search is in the form of top-down and begins from the entire image down to localizing an object in a leaf. The state is defined as the current window feature vector, the feature vector of the entire image and the history of taken actions. The agent is allowed to take an action from two predefined set of actions: one for scaling the current window to a sub-window, and the other for translating the current window locally [21]. Given a state, the agent selects the best actions from both groups of actions and then the selected actions are performed in two separate branches of a tree search. In this way, the agent takes both scaling action and local translation action simultaneously at each state. Each path from the root to a leaf of the search tree provides a near-optimal search. This is expected to improve localization accuracy for objects in different scales. To compute the reward for each action, IoU is used. IoU computes the proportion corresponding to the area of overlap between the current agent window and the bounding box provided by the ground truth over the area of union of the agent window and the ground truth. If the action taken results in increasing IoU compared to the IoU before the action was taken then the agent will receive a positive reward. Due to high dimensional continuous image input data and the model-free environment DQL is applied to learn an optimal policy [21].

2.2.3 Active Object Localization with Deep Reinforcement Learning

Active Object Localization [2] is the main paper by which this project is inspired. The main idea of the paper is to formulate the object localization task as a sequential decision making process where the agent's playground is images to localize objects. The agent is going to learn focusing on dense regions in images where there might be some objects. The agent is expected to determine the coordinates of bounding boxes tied to the objects in an image. More specifically, the agent starts by analyzing the entire scene and narrowing its current observed window down to an object by taking a sequence of actions. In the paper, the object localization task is formulated as a sequential decision making process and thus it can be modeled by MDP framework. The states of the MDP are a representation of the agent's window. To obtain state representation, the agent's window is processed by a pre-trained model and its features are extracted to be used along with a history of past taken actions. The pre-trained model used in [2] follows the architecture proposed in [22] and trained on ImageNet 2012 dataset for the classification task. It is said that using a pre-trained model for classification can help to extract spatial information in a scene and reduce the training effort

[2]. Regarding actions, the agent can transform its window in eight ways. They include moving left, right, up, down, reducing and increasing box scale, and making the window fatter or taller. There is also a termination action. The agent learns when to propose the current box as an object. More specifically, the agent begins its search by looking at the entire image. Each action changes the agent window height and width by a fixed factor. In this way, unlike the previous method the agent has four degrees of freedom to change the location of its box. Having freedom in moving the window has made the active localization method different from the other two approaches. To address the problem of designing a reward mechanism, active object localization uses a reward function based on increasing IoU. In the case of an action increasing IoU the agent would get a positive reward $+1$ and otherwise -1. If the agent successfully localizes an object it will receive $+3$. Finally, to obtain an optimal policy for object detection, DQL is used to solve the MDP [1] which makes it possible to use high dimensional inputs, i.e. images.

2.2.4 Hierarchical Object Detection with Deep Reinforcement Learning

Hierarchical object detection [23] considers the object detection task as a sequential decision-making problem where the agent's environment is an image. It formulates the problem as an MDP and tries to address it using Q-learning. Unlike "Active Object Localization" and "Hierarchical Object Detection" the agent is not free to move its window wherever it decides over the input image, instead there are a set of pre-defined windows that when the agent takes an action its window will be moved to the one of the pre-determined regions corresponding to the taken action. The agent starts by analyzing the entire image and decides where to focus among the sub-region windows. There are five candidate windows for the agent including four quadrants of the image plus a central region. Regarding candidate windows, two strategies are proposed to determine the sub-regions. The first strategy is dividing the image into non-overlapping sub-regions and the second, intersecting it into overlapping parts. Furthermore, the actions that the agent can take are categorized into two groups from which the agent can either decide to move its window and change observed region or terminate searching for objects. Similar to other works applied RL for object detection, the reward function is based on IoU. If an action leads to increasing IoU the agent will receive a positive reward. In addition, before a current observed window being used as an input to Q-network, using a pre-trained model, the state features are extracted. In hierarchical object detection, Image-Zooms and the Pool45-Crops models are used to extract features [23]. Each of them is used separately to propose a new model. Having extracted features using either of the models, the features are fed to a network consisting of a convolutional layer with 512 filters and size 7×7. The convolutional layer then is flattened and over three fully-connected layers its dimension reduces to the pre-defined number of actions.

2.3 Summary

This chapter provided a cohesive literature review for object detection algorithms that partly or mainly applied DL methods. These algorithms were divided into two basic groups, object detection and object localization. Object detection algorithms consisted of multiple stages. Two common steps in all of them were first localizing an object and then classifying it using a classifier, e.g. CNNs. However, the problem with these methods were the computational cost caused by the method used to produce region proposals and this fact that the algorithms had multiple stages to process an image. In other words, all these methods were classifiers re-purposed for object detection.

For this, in the most recent algorithms it was attempted to have only one network that was specifically designed for object detection and predicting bounding boxes. Using different approaches, object localization algorithms formulate the problem as a decision-making issue. The algorithms that follow this paradigm apply RL algorithms to overcome the problem. Inspired by human gaze control, these methods try to learn a representation for salient regions in images and use that as a guide to turn visual attention to the parts that are more likely to see an object there. This can significantly decrease the amount of computation. For this, in this project active object localization described in the section 2.2.3 is followed. This method uses the most similar algorithms to the way human vision works to localize an object in a scene.

Chapter 3

Approach

In this chapter, the approach used in this project is going to be explained in more detail. This project is mainly inspired by active object localization and DQL papers [2, 1]. As mentioned earlier, in this project the object localization task is formulated as a MDP which will be scrutinized in this chapter. In addition, experimental setup including the dataset used in this project, the training process, the conducted experiments, and their results are explained.

3.1 Methodology

Inspired by the human vision system, the aim of this project is to extract visual information from an image and turn the visual attention to the most salient part of the image. More specifically, the human vision system searches for a target in a scene in a top-down manner so that it first analyzes the entire scene and then narrows down its attention to focus on the target [21, 2, 23]. This process can be simulated by taking a sequence of actions in order to determine a tight box around a target.

To achieve this goal, MDP as a mathematical framework for modelling sequential decision-making is used. As mentioned in section 2.2.1, every MDP is a tuple $\langle S, A, P, R, \gamma \rangle$. However, the problem in the context of object localization is that transition probability, P, is not known. This implies the need for using a model-free RL approach. In addition, in the context of object localization the input images are high dimensional. All these requirements lead us to use DQL method. According to active object localization paper [2], the MDP elements are defined as below:

States

In the context of this project, the environment is considered as a set of images with which the agent interacts. The agent starts with observing the entire image and through interaction and by taking a sequence of actions tries to focus on dense regions of the image where it is more likely to see an object. The agent uses a window to look at a specific region covered by its window. In this way, at every step the agent just reads the pixels under its window. It is noteworthy that at the beginning of the interaction the agent's window covers the entire image.

The images have three channels, red, green, and blue. It is shown in [1] that colour information doesn't provide useful information for an agent to find its target in a scene. For this, in order to reduce the amount of memory and GPU/CPU usage, a pre-processing step is performed. The pre-processing step converts 128-colour palette images to a luminance channel which emphasizes more the information about the objects in an image and their boundaries. Not only does this prevent intensive memory and GPU/CPU usage but also makes distinguishing features more tangible for the purpose of object detection. Having converted images to the luminance channel, for the purpose of training, each state is stacked with the last three states. This is to make training process more efficient and allow the agent to learn through several steps.

Actions

While in hierarchical object detection [23] and Tree-RL [21] the agent can only move its window to certain parts of an image, in active object localization the agent is free to move the window to any region of the image. Giving the agent this degree of freedom is closer to how human vision works. This is one of the reasons the active object localization [2] method was followed in this project.

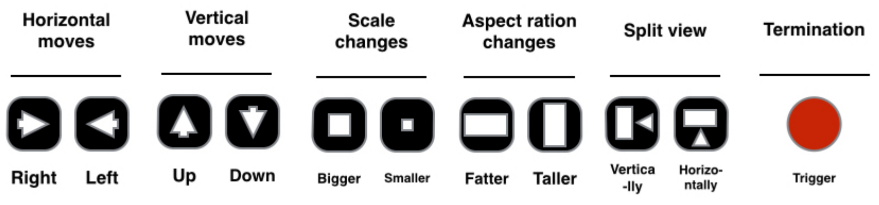


Figure 3.1: Set of possible actions

More specifically, there are 11 actions that the agent can perform. These include 6 categories of actions : horizontal moves, vertical moves, scale changes, aspect ratio change, split view, and trigger. Using these actions the agent can change the position of its window to move up, down, left, and right or make the window bigger, smaller, fatter, and taller or split it horizontally or vertically and finally the trigger action which terminates a search. The actions are shown in the fig 3.1. According to [2], this set of actions gives enough degree of freedom to the agent to be able to localize any object in a scene. Although, in the original paper split view actions are not used and they are just added in this project in order to enhance the agent ability to maneuver better. Using these actions, the agent can transform and change its window. At every step, the agent holds the coordinates of its window, $b = [x_1, y_1, x_2, y_2]$, which are the coordinates of the upper left corner and the bottom right corner. Every action makes a discrete change to the window coordinates in the following way [2]:

$$\alpha_w = \alpha * (x_2 - x_1) \qquad \alpha_h = \alpha * (y_2 - y_1) \tag{3.1}$$

Where α is set to 0.2 following the same setting in [2]. Having obtained the transformations, α_w and α_h are added or subtracted from the window coordinates b according to the taken action.

In addition, to address the exploration and exploitation dilemma, ϵ-greedy strategy is used. Since, at the beginning of learning, the agent needs to explore more and towards the end of learning it should exploit more what it has learned, an epsilon decay schedule is adopted. In this way, the agent starts interacting with the environment by taking random actions and as it learns more in the environment it takes actions based on the learned Q-values. Thus, in the following experiments ϵ is scheduled to increase from 0.1 to 0.8.

Rewards

In comparison to the other machine learning methods where annotated data is used to provide a feedback and optimize a loss function, in RL domain it attempts to avoid having annotated data. However, in RL there is still a loss function which needs to be optimized. For this, RL algorithms use an intrinsic signal which provides a feedback to the agent regarding whether the taken action has taken the agent close to its goal. In domains like arcade learning environments, the rewards are given to the agent based on the score the agent achieves. However, this feature is not available in an object detection problem. For this reason object detection using RL methods still need to use annotated data to provide a reward for the agent. In object detection problems a common way to compute reward for actions is using IoU between agent window and ground truth [23, 2, 21]. More specifically, in this project I followed the same reward scheme as suggested in [2].

If it is assumed that at state s the box b is the agent window at time step t, and at state s' after taking action a box b' is the agent window at time step $t+1$, and g is the ground truth then the agent will receive a positive reward, $+1$, if the statement 3.2 is positive. Otherwise the agent will receive a negative reward, -1.

$$RewSign_a(s, s') = sign(\text{IoU}(b', g) - \text{IoU}(b, g)) \tag{3.2}$$

The idea behind this reward scheme is that the agent will be given a positive reward if the taken action improves IoU. The reward is binary, $r \in \{-1, +1\}$. While IoU differences would be small and couldn't be used as a reward for the agent, a binary reward can both penalize and encourage the agent better by a bigger reward. The reward scheme for all actions excluding the trigger action is shown in the statement 3.3.

$$R_a(s, s') = \begin{cases} +1, & \text{if } RewSign_a(s, s') > 0 \\ -1, & \text{otherwise} \end{cases} \tag{3.3}$$

In terms of trigger action, the reward scheme is different. This is because the trigger action finishes the search and doesn't change the agent window, thus the statement 3.2 is always zero for the trigger action. After the trigger action is taken IoU is computed between the agent's window and the ground truth. If the IoU was bigger than a threshold, 0.5, the localization is considered successfully and the agent would get a positive reward, $+3$. The trigger action reward scheme is shown in the statement 3.4.

$$R_{trigger}(s, s') = \begin{cases} +3, & \text{if IoU}(b, g) > 0.5 \\ -3, & \text{otherwise} \end{cases} \tag{3.4}$$

Deep Q-Learning

As mentioned earlier in this chapter, to solve the MDP it is required to know transition probability, P, which is unknown in this problem. In addition to the transition probability, since the state and action space grow exponentially, it is not feasible to use model-based RL algorithms that require a model of the environment. All these clues lead us to the Q-learning method which is a model-free RL algorithm. Besides the large state and action space, the input data is high dimensional images by which it is aiming to learn concepts such as object representation directly from the raw input data. Thus, to incorporate the two ideas of Q-learning and learning from raw input data the method called DQL is applied [23, 2, 21, 1].

According to DQL method, by applying deep convolutional neural networks, it is attempting to learn a representation of the environment and approximate optimal action-value function. Based on the learned representation, the goal of the agent is to take an action that maximizes the expected cumulative rewards. This is formally shown in the following statement:

$$Q^*(s,a) = \max_{\pi} \mathbb{E}[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + ...|s_t = s, a_t = a, \pi] \tag{3.5}$$

The statement above shows that optimal Q value is the maximum sum of rewards r_t discounted by γ at each time step t which is obtained under a behaviour policy $\pi = P(a|s)$.

Based on DQL method, a convolutional neural network is used to approximate action-value function. To find the optimal approximate value function $Q(s, a; \theta_i)$[1], the loss function 3.6, which is based on TD difference, is optimized. The idea of TD difference is to compare predicted Q values, $Q(s, a; \theta_i)$, with the actual value that the agent will receive after taking the action, $r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$.

$$L_i(\theta_i) = \mathbb{E}_{(s,a,r,s')}[(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i))^2] \tag{3.6}$$

In the loss function above γ is the discount factor to specify the agent's horizon. In the experiments γ was set to 0.99 as suggested in [2]. The symbol θ_i determines the model parameters at iteration i used to predict Q values and θ_i^- is the model parameters used to compute the real Q value after taking action a and observing state s'.

3.2 Network architecture and implementation

One of the contributions of this work is implementing the algorithm proposed in [2] with Tensorflow. Since the original paper was published back in 2015 the algorithm was implemented with Caffe. However, in recent years, using computational graph abstraction, Google has introduced an efficient deep-learning framework called Tensorflow. While Caffe is mostly used for computer vision problems, Tensorflow is designed to support a wide range of machine learning methods including RL. The algorithm is re-implemented with Tensorflow with thanks to C. Caicedo for providing his implementation with Caffe and also with getting help from this github account for

[1]θ is the model parameters which in this case is neural network weights.

implementing DQL with Tensorflow. In addition, user manuals and documentation are provided in my github account.

In the original paper of active object localization, a pre-trained model was used to extract features from the images (base network) then the extracted features were used as inputs to a convolutional neural network. During training, only the second part of the network was trained according to the TD difference based on the similar loss function as 3.6. The first part of the network, the base network, has 5 Convolutional (Conv) layers pre-trained on a different dataset, the second part, which is an Q-network, has one Conv layer followed by 3 fully-connected layers.

However, in this project I followed the architecture proposed in [1]. The reason is that it is showed in [1] that this network has enough free parameters to capture the complexity of 49 video games which have much more complex environment than the object localization task. In addition, it has shown the network works effectively even without a pre-trained model. The details of the network architecture is shown in the table 3.1 and fig 3.2.

Layer	Input	Filter size	Stride	Num filters	Activations	Output
conv1	84 × 84×4	8×8	4	32	ReLU	20×20×32
conv2	20×20×32	4×4	2	64	ReLU	9×9×64
conv3	9×9×64	3×3	1	64	ReLU	7×7×64
fc4	7×7×64	-	-	512	ReLU	512
fc5	512	-	-	11	Linear	11

Table 3.1: Deep Q-Learning architecture

As suggested in [1], two tricks were used to make the learning process more stable and effective. One common problem that happens in methods where a function approximator such as neural networks is used to predict Q-values is the instability and divergence of predictions. To overcome this problem two tricks, memory replay [24] and freezing target networks [1], were applied.

To perform a training step with experience replay method, having taken action a_t at time t, the agent's experience is stored in a buffer with the format of $e_t = (s_t, a_t, r_t, s_{t+1})$. The buffer, $D_t = \{e_1, e_2, ..., e_t\}$, is a list of experiences that the agent has faced during its interaction with the environment. In this way, during the learning process, instead of using one experience per step, which is an inefficient use of data, a batch of experiences is sampled from the buffer and the network is trained by those experiences. Despite efficient use of data, experience replay also separates correlation between sequence of repeated experiences in order to prevent bias training towards repeated experiences which is the result of the agent being trapped in a part of the environment.

Another trick being used to prevent divergence in predictions is freezing target networks. According to [1], in order to prevent fluctuations in predictions, two separate networks were employed. The first one was used to compute the target statement showed in loss function 3.6, $r + \gamma \max_{a'} Q(s', a'; \theta_i^-)$ and second one was applied to compute the predictions, $Q(s, a; \theta_i)$. After every C updates, the second network parameters are copied to the first network used for computing the targets values.

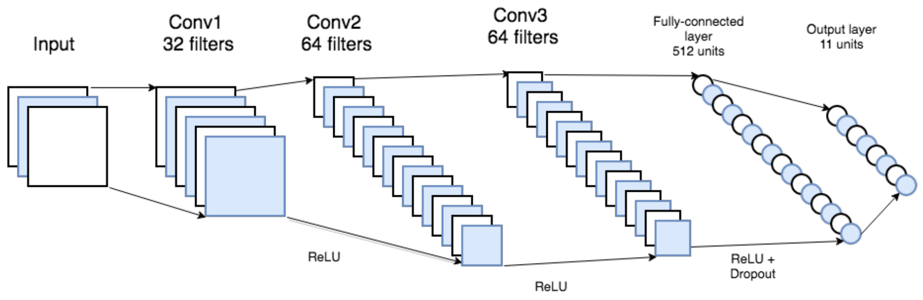


Figure 3.2: Set of possible actions

3.3 Analysis

In these experiments, it was attempted to train a model without using a pre-trained model for feature extraction. In addition, the experiments are designed to investigate the effect of tuning the model for one image category and then use it for bootstrapping. While in the original paper the experiments were conducted by training a model on a shuffled dataset from all categories, in the following experiments besides omitting the feature extractor part it is proposed to use one network and dataset to obtain a feature extractor and show how tuning the model for one category can improve the performance of the network for other classes.

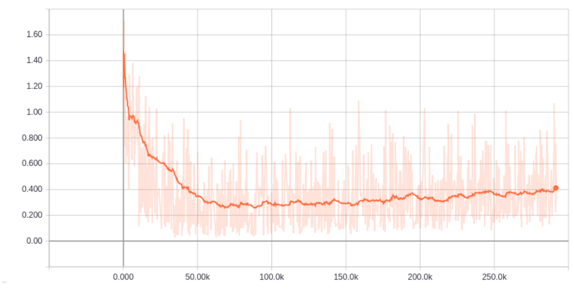
3.3.1 Dataset

The data set used in this project is Pascal VOC 2012. The dataset consists of 19386 images with 20 classes including person, bird, cat, cow, dog, horse, sheep, aeroplane, bicycle, boat, bus, car, motorbike, train, bottle, chair, dining table, potted plant, sofa, and tv/monitor. However, distribution over classes is not even. For instance, while number of cat images is 924, number of cow images is 273. Since in this project, the aim is to omit the pre-trained feature extractor and instead tuning the model for one category and use it for the next class there are 5 classes selected from one super-category including “cat”, “dog”, “cow”, and “horse”. Besides those categories, “person” class is selected as well which is from a different super-class. As mentioned before, this is to test whether the network is able to generalize information within a super-category and also between two different super-categories.

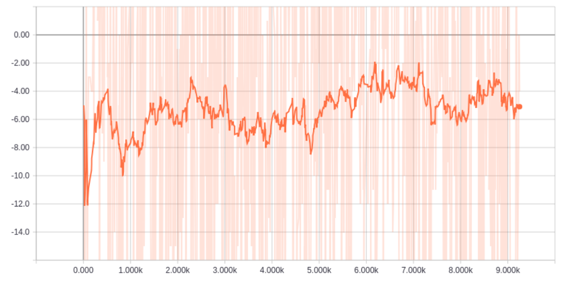
3.3.2 Experiments

First experiment is to find optimal hyper-parameters and architecture only for one category. The experiments started with cat images. First the same architecture as mentioned in the table 3.1 was followed. The agent could interact with each image in 15 episodes. However, as shown in 3.3a, despite the increase in cumulative reward per episode shown in the plot 3.3b, the loss function increased after $100k$ steps. This result motivated me to apply dropout technique [25] to prevent overfitting. Small dataset and excess number of episodes can be counted as the reasons of overfitting.

Having observed the above results, the experiment was repeated with two differences. First applying dropout technique on the third Conv layer output and second decreasing number of episodes from



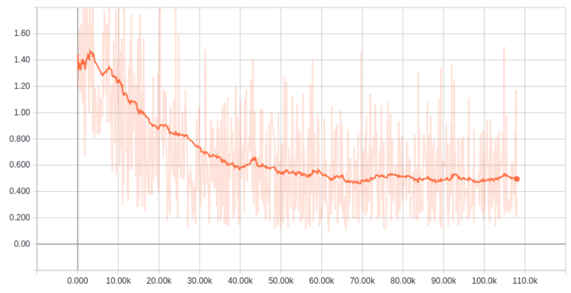
(a) Loss function. x-axis: step number, y-axis: error.



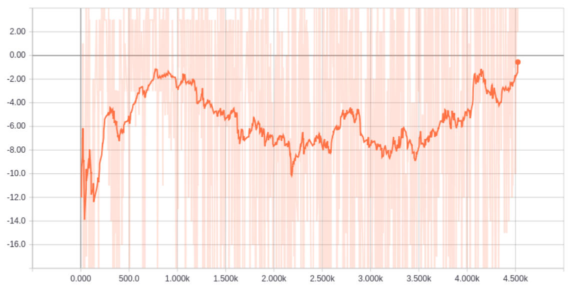
(b) Reward function. x-axis: episode number, y-axis: mean reward.

Figure 3.3: The model trained on cat images. The plots are smoothed.

15 to 5. These changes leaded to prevent overfitting and have a higher cumulative rewards. The results are shown in fig 3.4. Adding a dropout layer to the model caused the loss function decreased smoothly which shows overfitting has significantly reduced. In addition, a reward function achieved better scores with the same training data. Although the model without dropout shown in the fig 3.3b has a consistent trend, but the model with dropout, the fig 3.4b, despite a drop in the middle achieved higher rewards at the last 70 episodes with average 25.



(a) Loss function. x-axis: step number, y-axis: error.

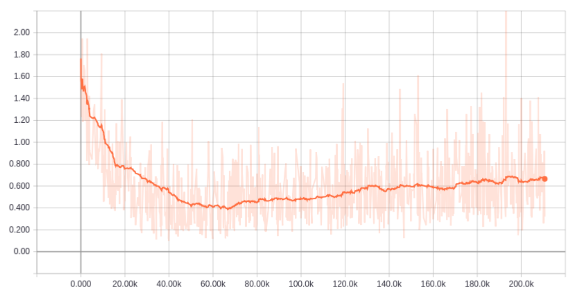


(b) Reward function. x-axis: episode number, y-axis: mean reward.

Figure 3.4: The model trained on cat images with dropout. The plots are smoothed.

In addition, to make sure that increasing the complexity of the network will not lead to a better accuracy, a new Conv and fully-connected layer was added to the neural network. With the new architecture the experiment above was repeated. However, as shown in the fig 3.5, in terms of loss and reward function similar result was achieved. The loss function at the minimum point was 0.19 which is about at the same level as the original architecture, 0.15. It is noteworthy that the pick at the beginning of the training is due to random actions. As mentioned earlier, to address the exploration and exploitation dilemma, a smart epsilon greedy strategy was applied. In addition, its mean precision on the testset was less than the original model. The evaluation result is shown in the table 3.2.

For evaluation purpose, Pascal VOC 2012 dataset was divided into 80% training and 20% testing data and the evaluation was conducted on testset. The performance of the agent was measured based on Mean Average Precision (MAP). For each object category the agent was given 200 images. Further, the agent could propose 15 regions for each image. The precision for successful regions, which had IoU above 0.5, then was computed. It is noteworthy that if the agent moves its window over an target object but doesn't use the trigger action to indicate the search finished while the agent has reached the threshold 50 actions, that region is considered as unsuccessful attempt. Thus, the


(a) Loss function. x-axis: step number, y-axis: error.

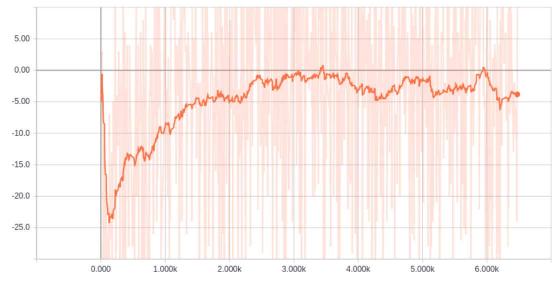

(b) Reward function. x-axis: episode number, y-axis: mean reward.

Figure 3.5: The model with new architecture trained with cat images. The plots are smoothed.

agent should explicitly use the trigger action so then that region will be evaluated.

Model Name	Best Model Precision
without dropout	0.34
with dropout	0.44
New Architecture	0.38
Baseline	0.23

Table 3.2: Results from the models trained on cat images and evaluated with IoU 0.5. The baseline is referred to an agent taking random actions.

As shown in the table 3.2, the original architecture based on the table 3.1 with dropout achieved the best MAP. It was expecting to get a better result from the new architecture however, since adding new layers introduces more free parameters and new free parameters require more training data to be tuned, going more over the data could only cause overfitting. For this reason, the model with dropout was used to continue the experiments.

In the following experiments, an incremental approach was used to test whether tuning the model for one category and using it for a bootstrapping training process for the next category works better than training on a shuffled dataset with all categories. At each step a trained model from the previous step was used as a bootstrap for the next model. More specifically, the trained model on “cat” images was used as a bootstrap to train a model on “dog” images then the trained model on “dog” and “cat” images was adopted for “cow” images and so on. To determine the effect of bootstrapping, another model was built and designed. The new model was trained directly on shuffled images from all the categories. That model is referred to as the ’shuffled model’ in the rest of this chapter. The results are shown in the table 3.3.

Table 3.3 shows the evaluation results for the models bootstrapping method used and the shuffled model. For comparison purposes, a baseline model and the results from active object localization paper [2] are also reported. In order to bootstrap, first cat images were used to train the model. The accuracy achieved for cat images was 0.46. The model trained on cat images then was used to train a model for localizing dogs in images. The accuracy achieved for dog class was 0.36 while precision for cat images decreased only 1 percent. Tuning the trained model for “cow” and “horse” images resulted in 0.24 and 0.38 respectively for these two groups. Finally, adding “person” category, which has come under a different super-category, caused a decrease in all categories. This can be explained by the similarity in animal shapes where all are quadruped but for detecting humans it is required to un-tune the networks weights and this causes decrease in classes coming under a

Model Name	Cat	Dog	Cow	Horse	Person	MAP
Cat	0.46	-	-	-	-	0.46
CatDog	0.45	0.36	-	-	-	0.40
CatDogCow	0.45	0.36	0.24	-	-	0.35
CatDogCowHorse	0.45	0.36	0.22	0.38	-	0.35
CatDogCowHorsePerson	0.44	0.34	0.21	0.37	0.23	0.32
ShuffledModel	0.27	0.18	0.12	0.18	0.14	0.18
ActObjLoc [2]	0.53	0.52	0.39	0.58	0.45	0.44
Baseline	0.22	0.15	0.10	0.15	0.11	0.14

Table 3.3: Evaluation results

different super-category. Despite the fact that the “person” and other categories are from different super-categories, still the network was able to use its learned information. This implies that the model can learn a general representation to find dense patters in images. To show that, another experiment was conducted with only “person” class however, the precision was 0.13. This indicates that network could generalize information learned from super-category animal to localize “persons” in images.

In addition, for comparison purposes, the results form three baseline models are indicated in the table 3.3. The first one is a model trained on the same dataset that the incremental model was trained on, with the difference that the whole data was used at once to train the model- the “shuffled” model. While MAP achieved for the incremental model was 0.32, with a considerable difference, the shuffled model achieved 0.18. This indicates that fine tuning a model for a specific class leads to a higher accuracy than training the model with random images. The second model results are brought from the original paper, “Active Object Localization with Deep Reinforcement Learning” [2]. It is noteworthy that this model was trained on both Pascal VOC 2012 and 2007. It is stated in [2] that amount of training data significantly affects the model effectiveness. In addition, the experiments were conducted by going over the whole 2012 and 2007 datasets 15 times. For this, the results from this project’s experiments are not comparable to the original paper results. However, the aim was to show that even with much smaller dataset and epoch numbers it was managed to achieve MAP 0.32 with 0.12 difference with the original paper. Further, the final model called random agent is reported to indicate that the trained agent behaves better than taking random actions.

Finally, the best model was evaluated with different IoU rates. The results reported in the previous experiments were with IoU 0.5. Shown in the fig 3.6 is precision at different IoU. It is reported for both the final model from the incremental method and each class separately. The agent could produce 15 proposals for each image and total number of 200 images were used for evaluation for each category. The cat category had higher precision at every IoU rates and the lowest precision trend belonged to “person” class. In overall, as it was expecting by increasing IoU rates the model precision gradually decreased.

3.4 Summary

In summary, in this chapter, first the mathematical framework of the method, MDP, was explained. More specifically, elements of MDP were described in more detail including what are considered as states, the actions that the agent can take, and the reward scheme. Further, it was justified

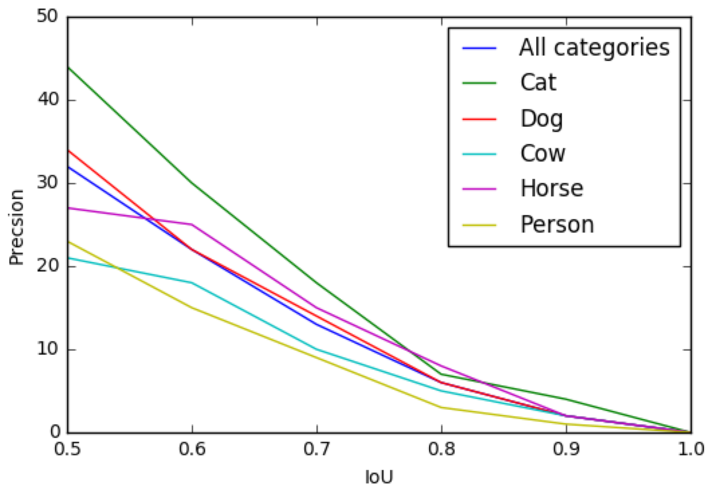


Figure 3.6: Precision/IoU graph for the best model

why DQL was selected to solve the MDP in comparison to other RL methods. Moreover, the network architecture along with the two tricks used to stabilize RL algorithm, experience replay and freezing target network, were described. Finally, with the purpose of omitting the pre-trained feature extractor, two main experiments were conducted. In the first experiment the model was trained with one category at each step and incrementally new object classes were added. Then, in the second experiment a model on shuffled data from all categories was trained. However, it was indicated that the model trained incrementally could obtain the ability of generalizing information within a super-category and also between two different super-categories. To show the ability of the model in generalizing its information to unseen category but from the same super-category, the first and second Conv layers filters of the best model are shown in appendix A for a sheep image which is not seen in the training process. Although, the results were behind the ones achieved in the original paper but by considering this fact that those results were attained by using a much bigger dataset and taking more time for training it is expecting that using a bigger dataset like COCO or combination of two dataset could result a better precision and reach to the level of the original paper.

Chapter 4

Discussion

This chapter first gives a brief summary of the project and then based on the observations and investigations during the project, possible extensions are suggested.

4.1 Summary

To summarize, this project investigated an object localization method based on the human gaze vision system. The aim was to prevent processing an entire image in order to find objects. Currently, the-state-of-the-art methods divide an image into sub-regions and search for an object in all the sub-parts. This is inefficient for applications like embedded systems where the computation power is limited or the resolution of the images are high. To address this issue, an object localization task was formulated as a decision-making problem. Followed the method proposed by [2], popular RL algorithm DQL was used to learn a policy from raw input data, i.e. images, to localize objects in a scene. In this way, using the policy learned, a set of actions that transforms a window was taken in order to tighten a bounding box around the target object.

While the algorithm proposed in [2] was implemented in 2015 using the Caffe framework, the first contribution of this project is to re-implement the algorithm with the novel deep learning framework, Tensorflow. This implementation can easily be modified by future researchers in order to conduct further experiments with different datasets, neural network architecture, or the agent architecture.

Furthermore, a set of experiments was conducted in order to show learning a localization policy is possible even without using a pre-trained model. While in [2] a pre-trained network following the architecture suggested in [22] was adopted to extract features, in this project by using the architecture proposed in [1], which could reach human level in arcade game environments, managed to achieve the results close to the original paper. More specifically, first a model trained on images from the same super-category including “cat”, “dog”, “cow”, and “horse”. Having trained the model on “cat” images, because of spatial similarity between “cat” and other classes from animal super-category, the network was able to generalize its information for other classes. The generalization ability was even extended beyond one super-category. By adding “person” class from a different super-category, the network could use this information to enhance its ability to localize persons in images. It seems the network was able to detect dense regions in images no matter what the object was in that region. However, the results achieved in this project were still behind those of the original paper. This can be explained by the amount of data and time used for training. In the original paper the combination of Pascal VOC 2012 and 2007 was used for training which has a significant effect on the results. However, in this project, due to time constraints, using a large dataset for training was not possible. Thus, training with a larger dataset remains for future experiments.

4.2 Future work

In the light of the above results, and given the potential of the project, a number of suggestions for taking this project to the next level are provided. While the scope of the MSc project didn't allow me to implement and conduct more experiments, it is predicted that these improvements could boost efficiency and effectiveness significantly.

The first idea that can affect the results of the experiments is using a pre-processing method to reduce the amount of computation. In this project, in order to reduce training time the input images were re-sized to 84×84 which results in loosing information in images. However, using Software Retina [26] it is possible to reduce resolution of the images and simultaneously keep important information for training.

Furthermore, the experiments can be extended by trying to learn the coefficient of box transformation, α. The formula is shown in the statement 3.1. Although active object localization gives the highest degree of freedom among the other works to the agent , the agent can only transform its window by multiplying the coordination of the window with a fixed coefficient. However, if the agent was able to learn the coefficient along with objects representations this could help the agent to move the window to its desired direction by a learned factor proportionate to the situation. This could give a full control to the agent to search an image for objects.

In addition, adding a termination action to find all objects in an image brings the agent close to the idea of having a complete object localization algorithm. However, in this project only detecting one object in an image without using a pre-trained model was investigated. This can be extended so that the agent would be able to count the number of objects and continues to search until it has localized all. It is noteworthy that adding a new action will introduce a new error and it is predicted that much more training would be needed in order to achieve acceptable results. Regarding counting the number of objects in images, a new variant of neural networks called Neural Arithmetic Logic Units (NALU) [27] have been recently introduced. While previously neural networks weren't able to generalize outside the range of numerical values confronted during training, NALU can learn systematic numerical computation. It can be used in partnership with Conv or RNN to learn counting the number of objects in images.

Regarding the replay memory trick, there is proposed a new variant of this algorithm called prioritized experience replay [28] which, instead of giving an equal significance to all experiences, tries to prioritize some important ones. In this way, important transitions could be used more frequently for updating the network parameters and therefore learning becomes more efficient.

Finally, when the object localization task is formulated as a decision making problem, it becomes crucial to capture the dependency between the states observed at each time step and the correspondence between actions over time. However, convolutional neural networks are not able to learn relations in time series data. In addition, convolutional neural networks are still computationally expensive and the cost increase is commensurate with the number of image pixels. That is the reason that, in this project, all images were re-sized to 84×84. Instead of using CNNs, RNN with Long Short-Term Memory (LSTM) units, which can processes inputs sequentially, can be adopted. RNN can bring the advantage of using high resolution images without worrying about linearly increasing computation cost with regard to the images' resolution. In addition, it is showed [29] that RNNs specifically with LSTM units are the most appropriate architecture now to learn a dynamic internal representation of the scene by incrementally combining information over time series data. Furthermore, to find the best architecture neural network specifically for object detection it is suggested to follow NASNet method introduced in section 2.1.5. Using this method instead of applying a specific architecture designed by a researcher the agent itself would be able to learn and design the best architecture which is specifically optimized for the object localization task.

Appendix A

Conv layer visualization

In this appendix, it is shown that the model is able to generalize its information to unseen categories. While the model was trained on animal super-class, the model was still able to detect sheep in the input image, the fig A.1, even though the model was not explicitly trained to detect sheep. As it is demonstrated in the figures A.2 and A.3, the layer 1 and 2 filters could detect and localize sheep in the input image. This implies the model could learn a general representation for the super-category animal. The sequence of actions taken to localize the sheep in the image is shown on the github account.

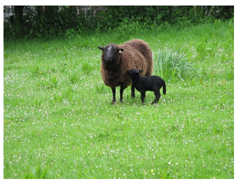


Figure A.1: Input image

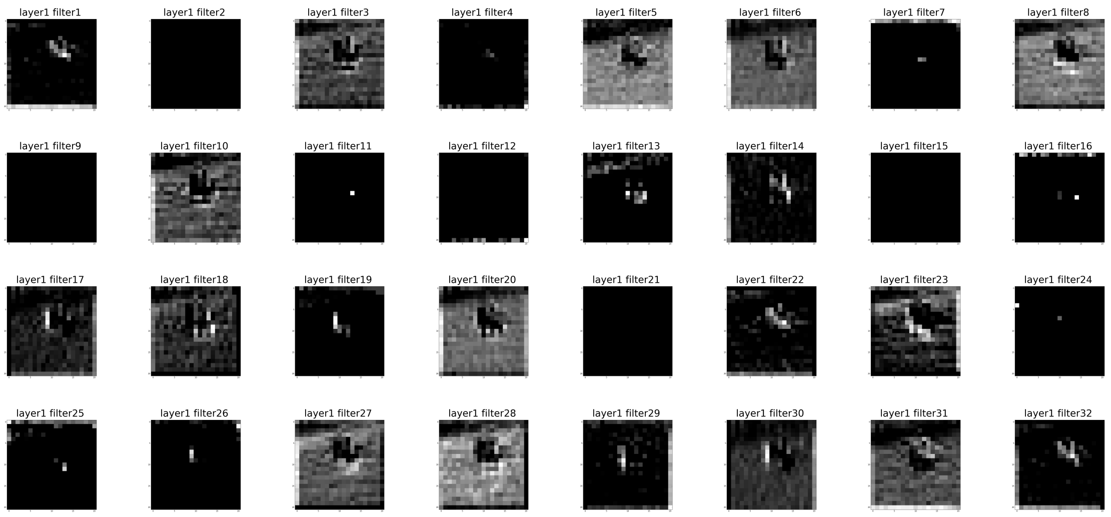


Figure A.2: Conv layer 1 filters

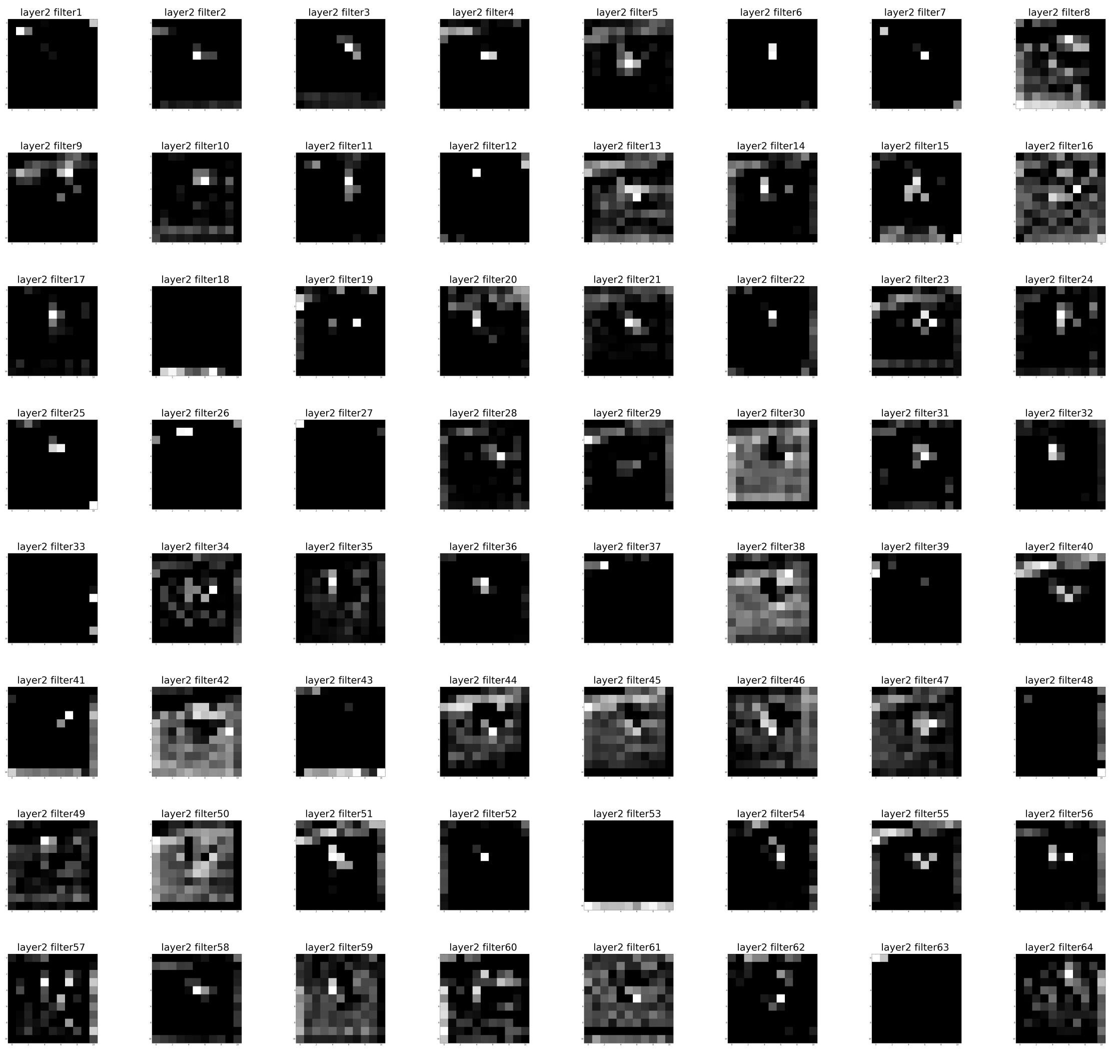


Figure A.3: Conv layer 2 filters

Bibliography

[1] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, February 2015.

[2] Juan C Caicedo and Svetlana Lazebnik. Active object localization with deep reinforcement learning. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2488–2496, 2015.

[3] Richard S Sutton, Andrew G Barto, et al. *Reinforcement learning: An introduction*. MIT press, 1998.

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

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

[6] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (6):1137–1149, 2017.

[7] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[8] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali 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] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[10] Jiri Najemnik and Wilson S Geisler. Optimal eye movement strategies in visual search. *Nature*, 434(7031):387, 2005.

[11] Jan K Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, and Yoshua Bengio. Attention-based models for speech recognition. In *Advances in neural information processing systems*, pages 577–585, 2015.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Computer Vision (ICCV), 2017 IEEE International Conference on*, pages 2980–2988. IEEE, 2017.

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

[14] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

[15] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866*, 2017.

[16] Lucian Busoniu, Robert Babuska, and Bart De Schutter. A comprehensive survey of multiagent reinforcement learning. *IEEE Transactions on Systems, Man, And Cybernetics-Part C: Applications and Reviews, 38 (2), 2008*, 2008.

[17] Zhang-Wei Hong, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, and Chun-Yi Lee. A deep policy inference q-network for multi-agent systems. *arXiv preprint arXiv:1712.07893*, 2017.

[18] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI*, pages 2094–2100, 2016.

[19] Alvaro Ovalle Castaneda. *Deep Reinforcement Learning Variants of Multi-Agent Learning Algorithms*. PhD thesis, Masters thesis, School of Informatics, University of Edinburgh, 2016.

[20] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, May 2015.

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

[22] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[23] Miriam Bellver Buenoa, Xavier Giro-i Nietob, Ferran Marquesb, and Jordi Torresa. Hierarchical object detection with deep reinforcement learning. *Deep Learning for Image Processing Applications*, 31:164, 2017.

[24] Long-Ji Lin. Reinforcement learning for robots using neural networks. Technical report, Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, 1993.

[25] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[26] Piotr Ozimek and J Paul Siebert. Integrating a non-uniformly sampled software retina with a deep cnn model. 2017.

[27] Andrew Trask, Felix Hill, Scott Reed, Jack Rae, Chris Dyer, and Phil Blunsom. Neural arithmetic logic units. *arXiv preprint arXiv:1808.00508*, 2018.

[28] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015.

[29] Volodymyr Mnih, Nicolas Heess, Alex Graves, et al. Recurrent models of visual attention. In *Advances in neural information processing systems*, pages 2204–2212, 2014.