Leap: a Model-Based Reinforcement Learning Framework for Fast Object Detection

BY

EDOARDO ROBA
M.S, Politecnico di Torino, Turin, Italy, 2020

THESIS

Submitted as partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Chicago, 2020

Chicago, Illinois

Defense Committee:

Amit Trivedi, Chair and Advisor
Natasha Devroye
Andrea Bottino, Politecnico di Torino

ACKNOWLEDGMENTS

.

A tutti coloro i quali mi hanno sopportato e supportato a 7229 km di distanza.

Vorrei ringraziare il professor Andrea Bottino, il quale mi ha sostenuto e seguito dall'Italia.

La mia famiglia, senza la quale non sarei stato in grado di affrontare un anno dall'altra parte del mondo. Grazie Richi che mi sei stato vicino anche a Chicago fisicamente.

Alessandra, che ha dato un senso alla lontananza, inseguendomi nei miei progetti sempre e comunque.

L'altra mia famiglia, gli amici, che seppur lontani ho sentito vicini.

Grazie ancora professor Amit Trivedi, il quale è stato il ponte tra l'Università dello studio e l'Università del lavoro.

E non ringrazio, ma cito la pandemia, la quale ha reso tutto questo progetto ancor più unico e raro.

ER

TABLE OF CONTENTS

CHAPTER			PAGE
1	**INTRODUCTION**		1
	1.1	Motivations	1
	1.2	Thesis structure	2
2	**BACKGROUND**		3
	2.1	Primitive methods	4
	2.1.1	Sliding window	4
	2.1.2	Region Proposal Network (RPN)	5
	2.1.3	RCNN	6
	2.1.4	Fast RCNN	7
	2.1.5	Faster RCNN	8
	2.2	Modern approaches	9
	2.2.1	SSD	10
	2.2.2	YOLO	11
	2.2.3	Object Detection with Deep Reinforcement Learning	12
	2.2.4	Leap	12
3	**MODEL-FREE AND MODEL-BASED OBJECT DETECTION**		14
	3.1	Previous and related work	14
	3.2	Recap on Reinforcement Learning	14
	3.3	Model-free approach	17
	3.3.1	Experiment	19
	3.4	Model-based approach	20
	3.4.1	Training principles	22
	3.4.2	Design	23
	3.4.3	Improving the design	26
4	**COMPATIBILITY OF PREDICTIVE MODEL WITH Q-LEARNING**		29
	4.1	Q-learning issues	29
	4.2	Moving the agent window	32
5	**RESULTS AND CONSIDERATIONS**		34
	5.1	Results on the dataset	34
	5.2	Some examples	38
	5.3	Considerations on the behavior of the algorithm	39
	5.3.1	Following the same path	39
	5.3.2	Convergence of the detector	40

TABLE OF CONTENTS (continued)

CHAPTER **PAGE**

5.3.3 Faster convergence with respect to the Model-free approach . . 42
5.3.4 Speed of the Leap algorithm 44

6 CONCLUSION AND FUTURE WORK 45

CITED LITERATURE 47

VITA 49

LIST OF FIGURES

FIGURE		PAGE
1	Sliding window algorithm	4
2	Anchor points and bounding boxes.	5
3	Example of proposals for RCNN.	6
4	Structure of Fast RCNN.	7
5	Process of the Faster RCNN algorithm.	8
6	SSD bounding boxes and ground truth.	10
7	YOLO process for 13x13 grid.	11
8	Reinforcement Loop loop.	15
9	Model-free algorithm structure.	17
10	Example of DRL for Object Detection.	19
11	Final schematic of the model-based algorithm.	21
12	Schematic of the network training process.	23
13	short	26
14	An example of the greedy policy on an 11-element vector.	29
15	Three cases where the algorithm gets stuck in a state are shown.	30
16	short	32
17	Average precisions for all the categories with 2-leap predictions.	35
18	Average precisions for all the categories with more than 2 leaps.	36
19	Average time spent to perform 200 steps per image.	37
20	Bus category.	38
21	Aero category.	38
22	Person category.	38
23	Sofa category.	38
24	Model-free approach.	40
25	It is shown 3-step ahead predictions algorithm; when the termination algorithm is predicted, the algorithm stops.	40
26	Model-free approach. The car on the left is found.	41
27	It is shown 2-step ahead predictions algorithm.	41
28	Model-based algorithm converges after three leaps in a forward pass to the car.	42
29	Model-free algorithm takes more steps to get to the final state.	43

SUMMARY

Thanks to the development of new technologies, Machine Learning has been a crucial turning point for the engineering research. It is a branch of Artificial Intelligence and it has become an efficient tool for problem solving in many fields: self-driven cars, biomedical research, software and even hardware problems can be solved through Machine Learning. In this project, we will show an application for object detection.

Almost all of the existing algorithms for object detection make use of a CNN to decrease the dimensionality of the processed data. However, this type of network has a very high computational cost usually, because of the large number of parameters. The purpose of this research is to design a model-based algorithm that tries to bypass the CNN. In other words, we created a predictive neural network which is able to predict future sliding windows of the detection algorithm. During the training of the Deep Q-Network, the transitions from one state to another are recorded and stored in a memory, which will be used as training data for the predictive network.

The final structure of the algorithm will be made up of two parts. The first one is the network described above, which forecasts the taken action by the object detection algorithm. The second part of the schematic is the Deep Q-network, which exploits the Markov Decision Process for the Reinforcement Learning algorithm.

At the end, we will show the results, the goals achieved and we will discuss about some future work.

CHAPTER 1

INTRODUCTION

Machine Learning is a wide area of research that was born in the middle of the XX century and it has been developing throughout the last years. In particular, in the 80s and 90s, engineers defined the basics of Machine Learning, such as backpropagation, SVM, recursive neural networks and so on. It is a branch of Artificial Intelligence and its goal is to give an algorithm the capability of thinking and having the consciousness of a human being. In particular, it has become possible to teach (or better, to train) a neural network how to recognize objects (Supervised Learning), how to distinguish classes of objects and find similirities (Unsupervised Learning) and make decisions based on the past experience (Reinforcement Learning). One of the most challenging aspects of Machine Learning is to translate a real scenario into the digital world. As a matter of fact, the human being is able to capture, analyze and make decisions using a very large number of data, even without being aware of it. However, during a computer simulation, the capability of a memory is still limited. The purpose of this project is to try to decrease the complexity of the operations during the object detection algorithm.

1.1 Motivations

This project comes from the personal interest in Machine Learning and the progressive research of the professor Amit Trivedi. It has been conducted in AEONLAB laboratory at University of Illinois at Chicago. One of the main reasons that motivated us to research on this

project is to try to overcome the computational complexity in the computer simulations that usually affect Computer Vision engineers. In fact, the project will be focused on the speed of the computations rather than the Reinforcement Learning algorithm itself. The starting question of this project was: in a real scenario, how can I reduce the complexity of the algorithm in order to make things go faster? How can we approach real-time simulations? This work can be adopted for many applications that make use of algorithms for object detection.

1.2 Thesis structure

In chapter 2, we will briefly introduce the already existing methods for Object Detection, starting from the most primitive ones.

In chapter 3, the model-free and model-based algorithms are introduced and the design of the new networks is shown.

In chapter 4, the compatibility of the new structure with the previous one is discussed.

In chapter 5, results are shown with respect the previous works and then a brief discussion on the behavior of the algorithm is shown.

In chapter 6, it is provided the conclusion of this project and some future works are discussed.

CHAPTER 2

BACKGROUND

In this chapter, we will discuss about Object Detection in general terms and then we will focus on the first methods that already exist.

Object Detection is a part of the science called Computer Vision. The goal of this area of engineering is to make a computer see as human beings do. Object Detection is made up of two parts: image classification and object localization. The first deals with image recognition and it can be developed through supervised learning methods; the goal is to recognize the object and classify it under a certain category. The second deals with localization of the object in the image: the goal is to identify the position by sorrounding it with a bounding box.

The consequences are huge: a car that is able to see in real-time simulations can bring to complete self-driving vehicles, an object detection algorithm can help doctors to identify cancer cells, videogames algorithms can interact better with the player as it was a 1v1 game, and so on. The applications are several and it can be a great step forward for Artificial Intelligence. Although it is a wide area of research and many valid algorithms have been developed, Object Detetion was born recently: in 2012 the algorithm RCNN was developed, in 2015 algorithms such as YOLO and Fast RCNN were published and in 2017 the code for SSD algorithm was written. In the next sections, we will briefly discuss about these algorithms.

2.1 Primitive methods

The very first algorithms are analyzed. They are very slow and have a very high computational cost. However, they have been the basis for the next generation algorithms.

2.1.1 Sliding window

This method is the first one ever developed for Object Detection. As we will see, it is very expensive in terms of computations.

The idea is to analyze the entire image, by sliding a window that scans every pixel of the image.



Figure 1: Sliding window algorithm

After scanning the entire image, the windows are fed to a CNN network (usually AlexNet) which creates feature vectors of each sliding window. AlexNet is a type of CNN which has 60 million parameters and it is used also for windows that do not contain any target. For this reason, the comptations take a very long time and the average speed was 20 seconds/image [1], which is not even near real-time applications.

2.1.2 Region Proposal Network (RPN)

This method approaches the problem in a totally different way. The algorithm creates anchor points on the entire image. For every point in the image, 4 or 6 bounding boxes with different sizes and shapes are created. Every box is then fed to a CNN which creates the extracted features. Then, the ground truths are compared to them.

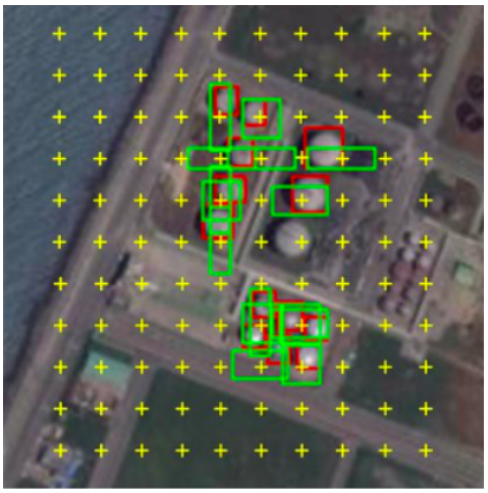


Figure 2: Anchor points and bounding boxes.

The speed of the computation of this method is not actually increased with resect to the previous one, since the entire image is processed by the CNN, even those boxes that do not contain any objects. However, this algorithm brought to develop algorithms such as RCNN.

2.1.3 RCNN

RCNN (Region-based Convolutional Neural Networks) was a huge step forward for Object Detection algorithms. The principle is very similar to RPN. However, there are a couple of differences: the first difference is that the number of regions created is fixed and it is equal to 2000. Then, the generated image proposals are fed to a CNN that extracts the feature vectors of each box. The convolutional network used was the one described at [2]. After that, the different outputs are classified by an SVM algorithm, which is able to classify the objects inside the proposals [4].

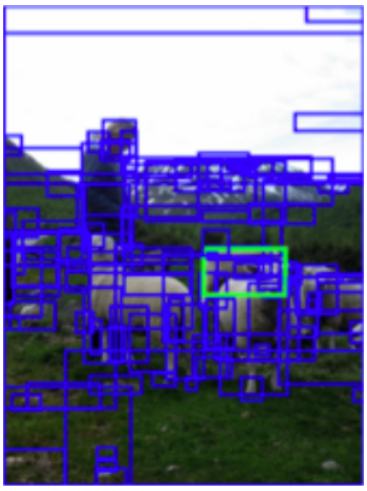


Figure 3: Example of proposals for RCNN.

In order to identify and to compare the different objects with the ground truth, the algorithm calculates the Intersection over Union (IoU) between the bounding boxes and the target (in Figure 3, an object was identified in green). The work at [4] shows a 30 % of improvements over the previous results on the dataset PASCAL VOC 2010.

2.1.4 Fast RCNN

The problem with RCNN is that the detection of an object is still slow. This is because the convolutional forward pass is computed for every proposal. Furthermore, since feature vectors are extracted for every proposal, the training takes several days and hundreds of GB of memory.

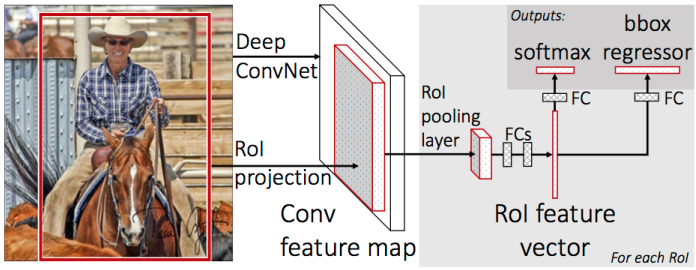


Figure 4: Structure of Fast RCNN.

Fast RCNN described in [6] proposes a new process of feature extraction. As it is shown in Figure 4, the entire image is extracted from a CNN, creating a feature map. The actual object detection is then computed with the feature vectores (proposals) extracted from the feature map. The output is made up of two elements: the percentage of confidence for every class and

a 4 value-vector which corresponds to the bounding box in the image for the corresponding object.

The results are very positive: in [6] it is shown that the training process is 18 times faster than RCNN trianing while testing is even 98 times faster than RCNN (0.1 s/image vs 9.8 s/image).

2.1.5 Faster RCNN

The Fast RCNN can be speeded up even more. The structure is the same: the regional proposals is computed after the CNN on the feature map, so also Faster RCNN is based on this structure, unlike the first architecture of RCNN.

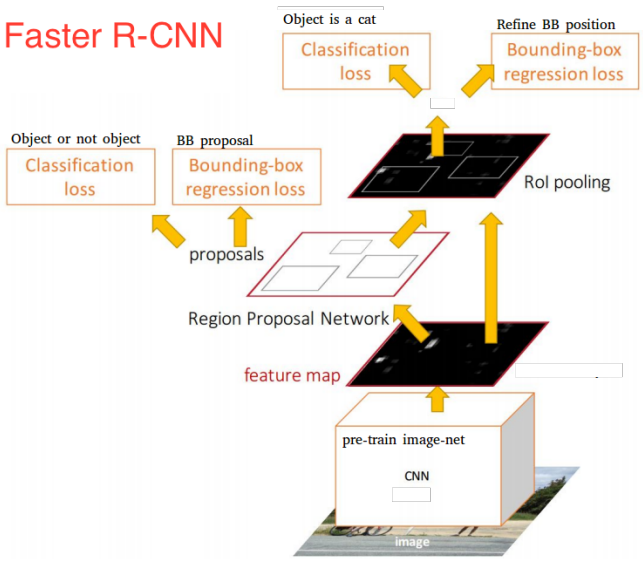


Figure 5: Process of the Faster RCNN algorithm.

As it is shown in Figure 5 [7], the difference is that on the extracted feature vectors several bounding box with different sizes and ratios for the same anchor are created. For each one of them, a classification loss determines if in the vector considered there is an object, even before the SVM classifies the category. In this manner, the SVM structure is bypassed for all the regional proposals that do not contain objects and it is used only for those regional proposals that contain objects.

Comparisons	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image (with proposals) [s]	50	2	0.2
Speedup [s]	1x	25x	25x
mAP [s]	66.0	66.9	66.9

The table was taken from [8].

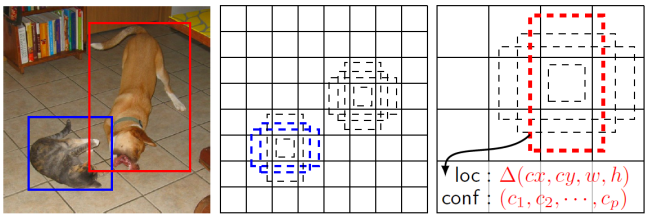
2.2 Modern approaches

Altough Faster RCNN increased the speed strongly, the previous works do not allow real-time simulations. This brings to consider object detection algorithm as an end in itself. The previous processes cannot be used for any realistic application.

Single shot detector methods represent the new family of algorithms for object detection. Two very similar algorithms called SSD (Single shot MultiBox Detector) and YOLO (You Only Look Once) represent a big step ahead in the Object Detection development and they have been developed at the same time more or less. They both can be used for real-time application: this is because of their very simple structures.

2.2.1 SSD

Single Shot MultiBox Detector (SSD) makes use of just one CNN and one deep neural network. For the training part, the algorithm only needs the input image and the bounding box defined a priori [9]. The image is then discretized to a set of default bounding boxes. The algorithm then chooses which set of bounding boxes is the closest to the ground truth.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 6: SSD bounding boxes and ground truth.

The results shown in [9] are very positive: the mAP (mean Average Precision, calculated with respect to the Intersection over Union function) achieved is even better than YOLO and it is equal to 77.2% for 300x300 input. However, even if this kind of algorithm can be used for real-time simulations, the speed of computation is slightly less than YOLO: 19 fps vs 21 fps.

2.2.2 YOLO

YOLO (You Only Look Once) algorithm is actually very similar to SSD: the principle is still to adjust the bounding box to the ground truth given a priori. However, the difference is that the image is subdivided into a grid.

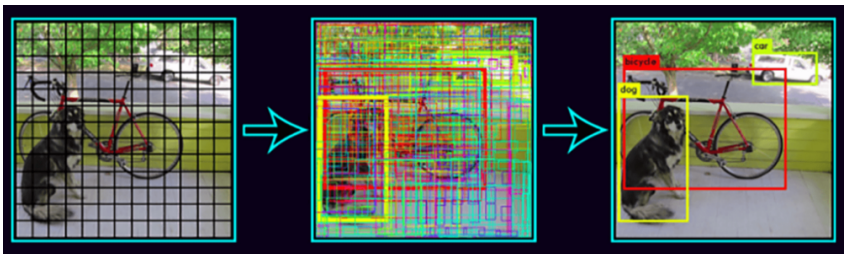


Figure 7: YOLO process for 13x13 grid.

The first YOLO experiments were considering 7x7 grid. The precision and speed of the computations strongly depend on the dimensions of the grid. As it is shown in the picture, after creating the grid, the square containing the center of the ground truth is highlighted and a bounding box around it is created. It is then reshaped in order to fit the object at the best and then the object is categorized.

The high speed is given by the fact that the algorithm makes use of a single network: a CNN, inspired by the GoogLeNet [5], where the fully connected layers predict the confidence of the class and the coordinates of the bounding box.

This method brings to have a very fast algorithm (21 fps and for fast YOLO the algorithm can even reach 155 fps); however, since the dimensions of the grid cell is fixed, the detection struggles with small object, since they might not be "visible" to the bounding boxes. As a matter of fact, as it was already mentioned in the previous paragraph, the mAP of YOLO is slightly less than the other real-time detectors: 66.4% mAP.
In order to show how light and fast this algorithm is, for the first time ever the team implemented this algorithm on mobile phones.

2.2.3 Object Detection with Deep Reinforcement Learning

At the end of 2015, a new approach for Object Detection was published [3] and the algorithm itself will be showed better in the next chapter. The principle is very different from the previous methods assumed previously. In this case, the computer takes an image as input and it is fed to a CNN, which has a simpler structure with respect to the previous CNN used. However, it still has more than 58 millions parameters, so this brings to have slow computations. The idea is to apply Deep Reinforcement Learning and teach the Agent to take the best action in order to approach the object. As it will be shown later in the next chapter, the first experiments with the primitive design reaches a good mAP (46.5%), close to the fast RCNN mAP.

2.2.4 Leap

Our approach starts from the algorithm showed in [3]. However, our goal is to speed up the process and bypass the CNN network, which is the heaviest part from a computational point of view. This improvement was already achieved with Fast RCNN. However, the purpose is not

to improve the object detection itself (it is part of future work), but to increase the speed and get closer to real-time simulations.

CHAPTER 3

MODEL-FREE AND MODEL-BASED OBJECT DETECTION

3.1 Previous and related work

As already mentioned before, for the purpose of our project a previous work concerning object detection was considered. In particular, we started from the model-free algorithm described by [3].

With the strong development of the Reinforcement Learning in the last few years, [3] shows a new idea behind object detection based on Deep-RL. As a matter of fact, the idea is to teach and train a Q-network that takes the best action to approach the object.

3.2 Recap on Reinforcement Learning

We will briefly introduce and recap the Reinforcement Learning principles, since it will be useful to understand the whole process better later.

Reinforcement Learning is one of the three categories fo Machine Learning: Supervised, Unsupervised and Reinforcement. This last category deals with the science of decision-making for computers. As a matter of fact, every Reinforcement Learning algorithm needs at least four components: an agent (or action), a Q-function (or V-function, but in our case we will consider a Q-function), a reward function and a policy. The process can be explained through the schematic in the Figure 8.

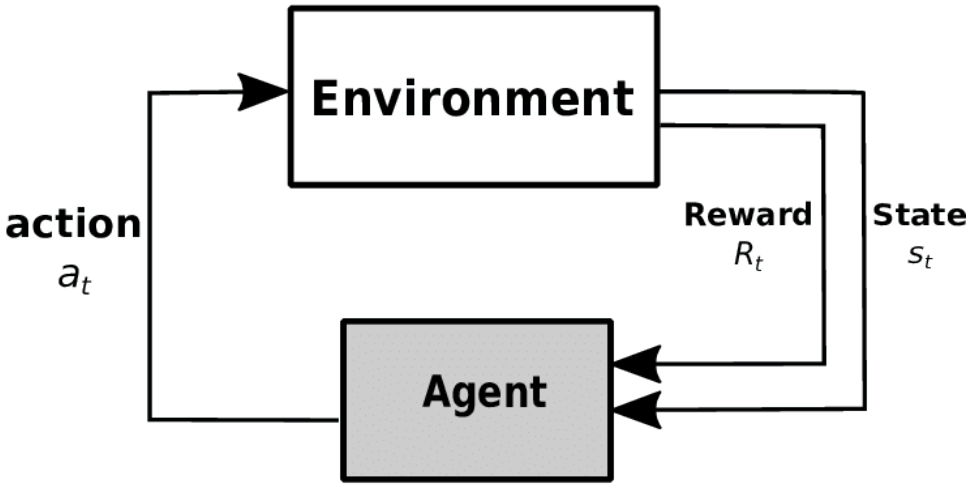


Figure 8: Reinforcement Loop loop.

The environment is the input, which can be an image, a map, an audio, etc. In our case it will be the image where we want to apply object detection. From the environment, the algorithm takes a random state, which is fed to the Agent. The Agent is then responsible to select the best action to take in order to get positive reward. As a matter of fact, after taking the best action, the next state is compared to the environment and a reward is obtained, which is defined a priori under a policy. The policy describes what are the best actions to be taken in order to get positive rewards. The loop is very similar to the human learning process: in fact, a human being takes action based on the past experience. The Q-function is useful because is the function that tracks the good and bad actions: the higher is the q-value, the better the corresponding action is and viceversa with bad actions. The only negative part of this process

is that it does not exist a list of predefined Q-function, so the designer of the algorithm must select the one that fits the process the best.

The Q-function is updated following the Markov Decision Process (MDP), which can be explained by the following formula:

$$Q(S, A) \leftarrow \alpha\Big[R + \gamma max_{A'} Q(S', A')\Big] \tag{3.1}$$

As it is possible to see, the formula shows that the Q-values depend on the reward, the next state and the action taken. More precisely, from one state S, a random action A between the possible ones is taken and the next state S' is observed. From this state, the best action A' (according to the policy, it is the one with the largest Q-value) is taken. γ is the discount factor and in our case it is equal to 0.99 while α is equal to 1.

This process will update the Q-function which will have to weigh the good and bad actions. At the end of the training the algorithm will know what the best action is for the corresponding state.

However, in [3], it used Deep RL. Usually, Deep Learning is that discipline where Learning is applied to extracted feature. As a matter of fact, Deep Learning is useful for applications such as Object Detection, Image Processing, Audio Processing, where a CNN is typically used. In fact, in our project, we will work with extracted features of the input image and the Reinforcement Learning algorithm is applied on them.

3.3 Model-free approach

In the previous work, it is used a model-free approach. This means that the algorithm does not know the transition from a state to another and it interacts with the next state only when the action is taken. Most of Object Detection and Reinforcement Learning algorithms follow this method. In this subsection, we will briefly show the structure of the previous model and its data-stream.

The structure is the following:

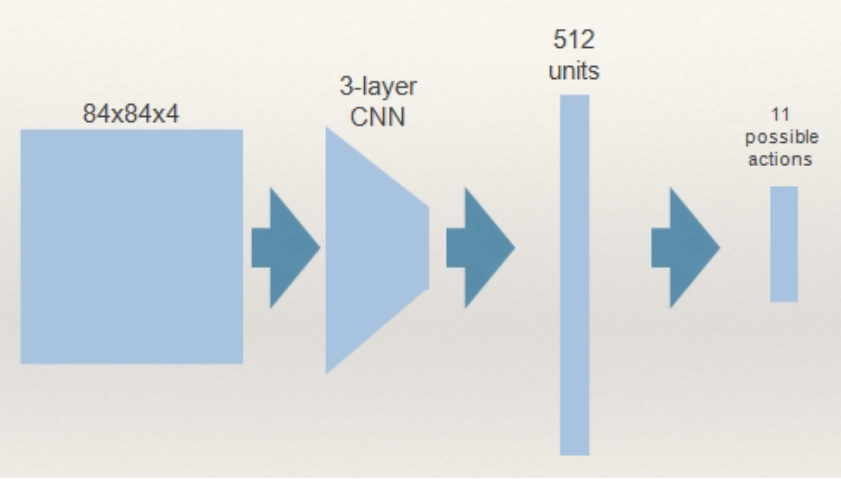


Figure 9: Model-free algorithm structure.

The input is a Tensor of 4 RGB frames of the bounding box of 84x84 pixels each (if the input boubding box is not square, it is adapted to be 84x84). Since the dimensionality is high, a

pretrained CNN is used. The network has 3 convolutional layers and a fully connected one. As already mentioned, the total amount of parameters of the CNN is 58 million parameters. The output of the CNN is a 512-element vector: this is the number of data on which Reinforcement Learning is applied. Since the ground truth is given a priori, the algorithm knows how to define good and bad actions. In fact, the reward is calculated depending on the IoU of the current state and the ground truth. This defines the Reward function, useful for the computations of (3.1). The parameters of the last layer are updated through gradient descent and backpropagation. The output of this final layer represents the q-values of DRL process. The higher these elements are, the better the action is for that state to be taken. However, if only one action was to be chosen for the current state, it would limit the detector (this concept will be analyzed better in chapter 4). The method in this case assumes to follow a ϵ-greedy policy. This means that during exploration, the algorithm understands which action is the best for each state, picking them up randomly. During exploitation, the algorithm takes the best action with the highest q-value. However, due to the ϵ-greedy policy (ϵ=0.2) approach, during exploitation, it is defined a probability density function, so the action with the largest q-value is more likely to be taken, but for some case, it may happen that the algorithm chooses some random action. The reason behind this approach will be more clear in the next chapter, where more specific situations will be analyzed.

The designed network outputs as many outputs as all the possible actions and this is a very good point since the input image is fed to the model only once and the output is computed in just one forward pass [3].

3.3.1 Experiment

The experiment conducted on the model-free algorithm makes use of the Pascal VOC2012 dataset. In the previous work, 19386 images were used, $\frac{4}{5}$ for the training, while a fifth for the testing. The images and their corresponding targets are uploaded by the function $VOC2012_npz_files_writter.py$. However, since the dataset is very large, the images are stored in a memory in an efficient way in the function $readingFileEfficiently.py$.

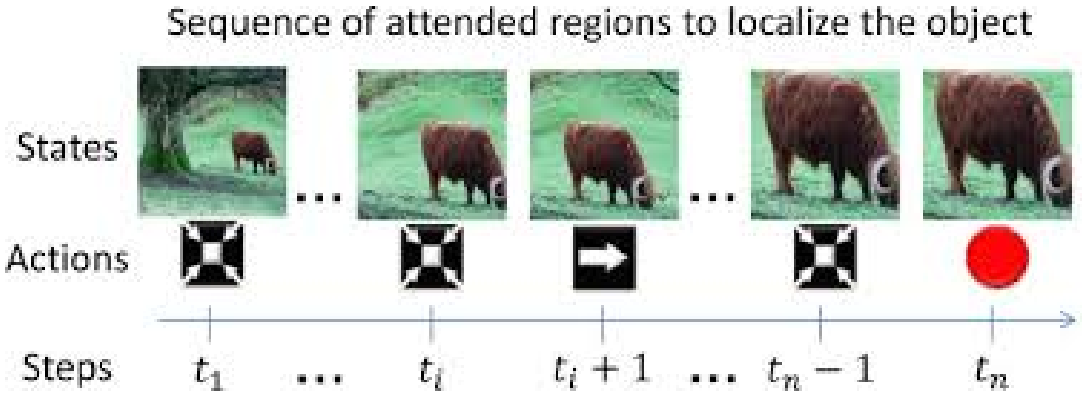


Figure 10: Example of DRL for Object Detection.

In [3], the experiment was conducted on 20 different classes. The obtained mAP (46.1%) is closed to the one obtained with RCNN (54.2%). On top of that, in order to better understand how good the algorthm is, the recall function is calculated. In image processing, recall is the metric that calculates the fraction between true positive and all the actual positive obtained.

In [3], it is shwon that recall for this kind of algorithm is equal to 71%. This is a good result, however all the other main object proposals algorithm (such as) reach at least 90%.

3.4 Model-based approach

The idea of our project comes from the fact that we wanted to see if this algorithm could get closer to real-time simulations. We were inspired to the works that brought to design Fast RCNN and Faster RCNN, which elude the CNN computational cost somehow. In fact, as already shown previously, these two methods play with the CNN and find ways to bypass it. The results are very positive and the average speed per image is increased.

The purpose of our work is to increase the speed of the computations of the algorithm. As a matter of fact, the project was mainly focused on the CNN.

As a matter of fact, the DRL algorithm has not been changed. The structure and the flow remain the same. However, there are still two big differences in the design. The first one is the presence of a neural network, which has the goal to predict future states and actions; the second one is that the output from the CNN is concatenated to another vector, which represents the past actions history (the reason will be more clear after the next chapter). In the following picture, it is shown the final structure of our proposed algorithm.

In the model-free approach, the Transitions (state,next-state,action) were not known and the agent knew the next state only after the interaction with the environment. However, for every Transition, the algorithm needed to go through the CNN and create the feature vector corresponding to the current state. In the model-based approach, as the name suggests, the goal is to create and work on an existing model, based on the Transitions recorded during the

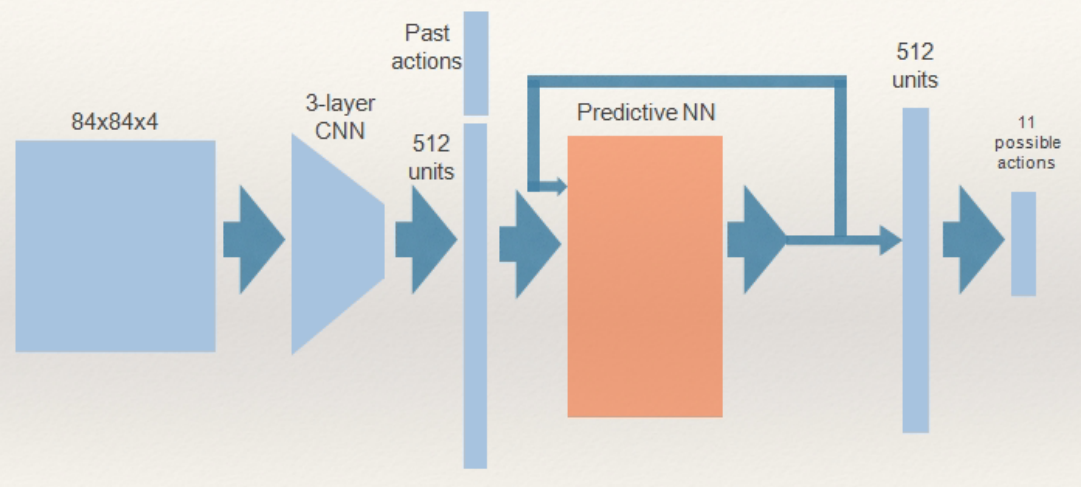


Figure 11: Final schematic of the model-based algorithm.

training of the Deep Network. In fact, the training of the entire Learning algorithm is divided into two parts. First of all, the backpropagation and Gradient Descent are applied on the second section of the network and using the Markov Decision Process (3.1) the weights are updated in order to select the best action. During this process, the Transitions (state, next-state and action) are all recorded in a replay memory. These data will be the training data used for the Predictive Neural Network. This additional Neural Network has the purpose to predict the next states in order to perform the leaps and bypass the CNN. As a matter of fact, the second part of the Learning process is the training of the Predictive Neural Network. It is well known that neural networks are useful for many applications and there exist many different typologies for different purposes. The kind of neural network used for our project is very useful for stock

market predictions, weather forecast or virus spreading. In general, for time series analysis, it is possible to use fully-connected layers networks, as it is shown in [10].

3.4.1 Training principles

The best way to teach a predictive neural network a sequence is to train it considering the flow of data in time. As a matter of fact, the only data we have is a sequence of data stored in a replay memory. There is no input-output relationship to train the network with. Nevertheless, the training desired outputs are intrinsic: since the goal is to learn a sequence, state and next-state pairs will be alternatively selected. In other words, in the replay memory, one Transition includes one next-state that will be the current state for next state. In informatic terms, the input training data will be the states from 0 to $n-1$ (where n is the length of the memory) and the output training data will be the next-states from 1 to n. In this manner, the network understands the sequence of Transitions. Figure 13 shows the schematic of the training process.

The design of the network will be showed better later.

The goal of this predictive neural network is to predict the future bounding boxes on the images. As the name of the algorithm suggests (Leap), this model performs leaps on the image in order to avoid to compare the state with the environment. In theory, with the correct design of the neural network, this should be already enough in order to make the algorithm work well. However, some more details must be considered.

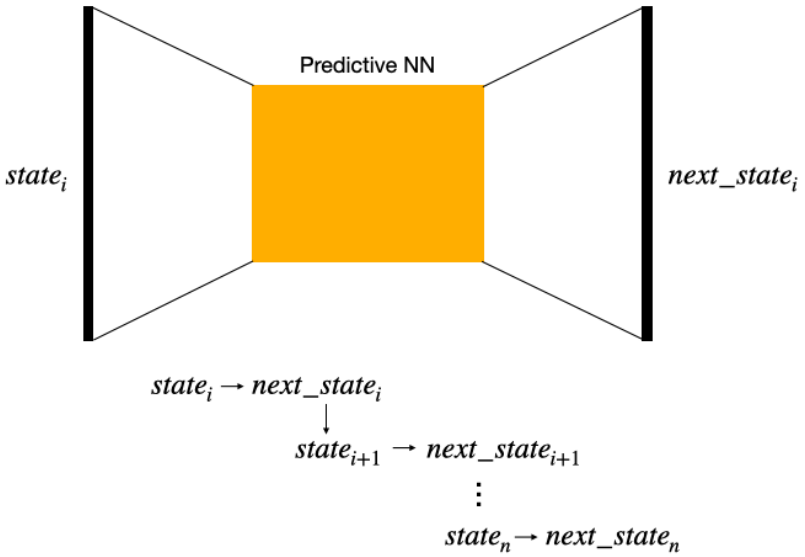


Figure 12: Schematic of the network training process.

3.4.2 Design

The goal of the project was to design another neural network lighter than the CNN, in order to decrease the complexity of the structure of the model. Nevertheless, since there is not a precise literature for designing this kind of networks, we needed more than one attempt in order to obtain the best trade-off between speed and accuracy.

The first network designed had three hidden layers between the input and output. It was a very complex and heavy design, but the starting point for the design was very close to the cost of the CNN on purpose. In this manner, the goal was to understand if a fully connected layer network was able to predict the Transitions. The three layer network has a symmetric design:

the first and final layers had 4000 neurons and the layer in the middle had 6000 neurons for a total of 52'116'000 parameters. Compared to the CNN, the complexity is very close. This network is able to reach an average of 85.1% of training accuracy over all the categories. The loss function that is optimized is:

$$L(w) = (y - out(w, x))^2 \tag{3.2}$$

where w are the weights of the network while the function $out(\cdot)$ defines the function between input and output. The accuracy considers those predictions that have the result of the equation (3.2) less or equal to 0.001. At first, there was no activation function, so the relationship between one layer and the other is linear. This is because at the beginning the nature of the feature vectors were unknown and we wanted to give more freedom to the computations of the network.

Since the computational cost and the average speed per image do not decrease, the final design could not be equal to the one that is just showed above. Anyways, designing a more complex network was useful to show that a fully connected layer network was able to predict and understand the sequence of the Transitions.

For the second design, the first step is to reduce the number of layers. Since the more neurons the network has, the more feature it is able to extract, the number of neurons per layer does not change much, but the number of layer is decreased to two. The two layers have 5000 neurons each and the total number of parameters is 30,130,000, which is almost half of the total of the CNN. With this design, the network was reaching less than 65% of training accuracy.

In order to improve the training process, the type of input and output data was analyzed. Since the data are extracted features, the main characteristics is that most of the elements are equal to 0. In order to improve the accuracy, it is needed to add an activation function that would output numbers greater or equal to 0 only. For this reason, the ReLU function is inserted in the last layer of the network, so the negative input data are converted to 0 while the positive elements are kept linear. In this way, the training accuracy reaches 80.4%.

At this point, the dimensionality of the network is decreased once more. The structure is still symmetric and the number of neurons for each layer is 2500. With this number of neurons, the total number of parameters is equal to 8,815,000. This is a huge step forward from a complexity point of view with respect to the structure of CNN. The number of training data used for the network is around 12k data, depending on how many steps are recorded per image. Since this dataset is particularly large, the learning rate parameter η during backpropagation is set to be equal to 10^{-11}, in order to make the updates slower. This process is very slow (it takes 12h of training more or less), but the convergence is more guaranteed. As a matter of fact, with these settings and the ReLU function on the last layer, the training accuracy reaches 78.6%. This is a good result, since it is not very far from the previous one.

It may seem that decreasing drastically the number of neurons does not affect the training. Nevertheless, with 2000 neurons per image the accuracy decreases strongly and it does not get bigger than 64%, which is not enough for our purpose.

3.4.3 Improving the design

The designed network is working well. However, it is possible to show that this designed network works for a little group of images, while for the rest is more or less random. At this point, the reader can debate on the fact that the network is predicting some Transitions, without knowing the context. As a matter of fact, until now the algorithm does not know anything about the data: it does not have any information between one state and the next one, the difference between one image and the next one, the different types of images that can be fed to the network and so on. The designed network is only giving outputs given some certain inputs, without differentiating the images.

If we think of the following images:



Figure 13: The same state for three completely different target positions.

In Figure 13, three very similar states are shown (in the top). However, the final target positions of the target (in this case cars) are very different. For the design described above and

the considerations discussed above, the algorithm should not distinguish the states to predict and the behavior of the algorithm is random. For example, in the first picture of Figure 14, the algorithm should go down until almost the last possible state, while for the second picture the bounding box should stop in the middl eof the picture more or less. For the third picture, the algorithm should predict to go right in order to get to the target. This result brings to make some changes in the network.

The question is: how to make the algorithm distinguish the different states for the different images? The solution comes from the idea that the more information is fed to the network, the better the predictions are. Exploiting this property of neural networks, we wanted to give a history for each state. The idea is to come to a state knowing which path the algorithm has already followed. For example, if the image contains the sky, the car will definitely be on the bottom of the image. In fact, if the the past actions are recorded, the network should predict that the first actions to be taken should be to go down until the road is reached, and so on for the rest of the images.

To do so, the past actions are recorded. The first memory that collected the past actions encoded them in an 11-element vector, which was set to 0 and every time an action was taken the corresponding element was increased by 0.1, since the elements of this further vector needed to have the same magnitude of the feature elements, avoiding to give numbers with very different magnitide, which would bring to computational issues.

Nevertheless, even if this would become the right way to follow, this starting design of the past actions history failed. The accuracy decreased a bit but the behavior of the algorithm

did not get better. The reason is that 11 elements are negligible with respect to 512 of the feature vector and the network is not sensible to this increase of the input data dimension. The solution to this problem is very simple: the actions are encoded in a 121-element vector (set to 0 at first) and every time an action i is taken, the corresponding ii_{th} element is increased by 0.1, in order to maintain the same shapes of the input data.

The resulting input dimensionality is equal to 633. This brings to have an increase to the ntotal number of parameters of the network: 9,117,000. This training accuracy of the designed network just described reaches 78.8%, which is very close to the previous case.

The final training process can be represented by the following pseudocode:

Algorithm 1 Training process

for $i = 0 : N$ **do**
  $TrainQ - functionwith(State, Action, Next_State)$
  $Record(State, Action, Next_State)$
end for
while $True$ **do**
  while $end_episode != True$ **do**
    $state_{i+1} = NN(state_i)$
    $parameters \leftarrow GD(state_{i+1} - Next_Statei^2)$
  end while
end while

CHAPTER 4

COMPATIBILITY OF PREDICTIVE MODEL WITH Q-LEARNING

4.1 Q-learning issues

As it was already mentioned, the algorithm creates an ϵ-greedy policy for the Q-learning process. This means that from the already trained Q-network 11 elements (which correspond to the Q-values) are extracted. The highest value corresponds to the best action to take, but on the basis of these data, a probability distribution function is created. The corresponding action with the highest q-value will have more probability to be taken, but it will not the only possible solution. It may happen that another action can be taken, as it is possible to see from the next picture.

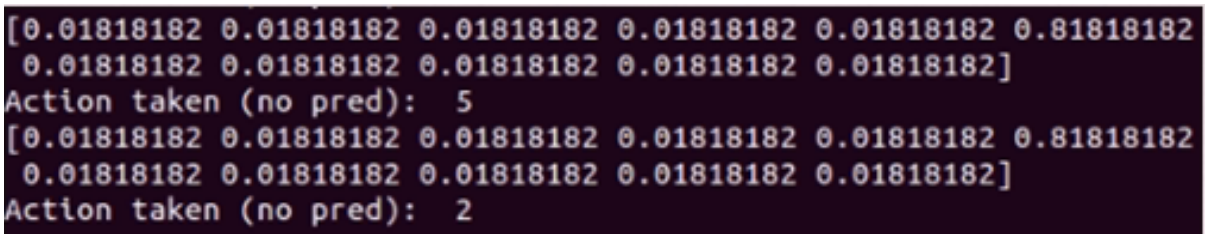


Figure 14: An example of the greedy policy on an 11-element vector.

It is easy to show that for most of the cases, the action with the highest probability is chosen. One further reason for the choice a ϵ-greedy policy is that it may happen that the algorithm gets to a certin state close to the boundaries. Furthermore, if the algorithm for that particular state selects an actions which would go outside of the boundaries, the algorithm would stop at that state. In other words, if the state does not change, the best action computed by the Q-learning will be always the same and the bounding box would remain the same.

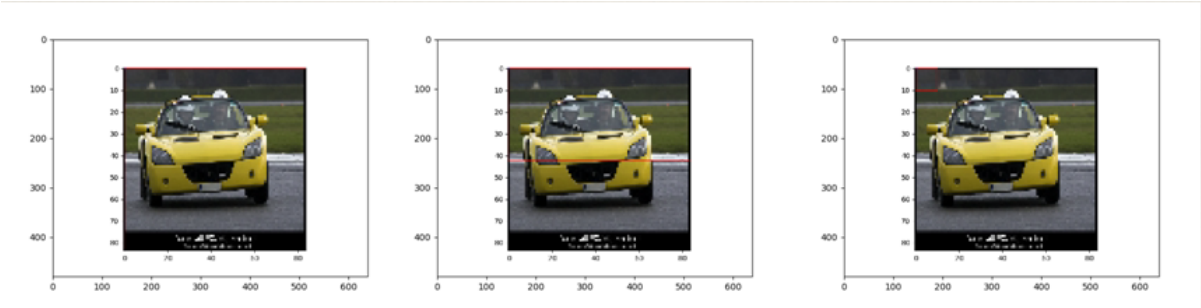


Figure 15: Three cases where the algorithm gets stuck in a state are shown.

The adoption of ϵ-greedy policy overcomes this problem. If the algorithm gets stuck on a particular state, the algorithm gives some freedom to choose other possible actions that would move the bounding box from that current state.

However, this brings to randomness, which cannot be predicted by a mathematical tool such as neural networks.

Given this situation, the question is: how to predict random behaviors? Or, at least, how to make the algorithm not lose the target even if one random action is taken?

The solutions that we adopted are two.

In order to decrease the probability of taking random actions, when one Transition is predicted, the action with the highest probability is chosen. In other words, the policy that selects the best action for the prediction is deterministic and one the best action is calculated, the action with the highest probabiliy is chosen.

As just mentioned, the second solution we adopted is that every n-steps the algorithm compares the predicted state with the environment. This method that we adopted is necessary because we lose less accuracy. If we really think about the process of object detection, the predictions of the bounding boxes are a result of a mathematical computation given by a neural network. Even if the predictions were perfect, it is better to compare the current state with the environment. The reason is that we wanted to make the algorithm not to lose the right path. As it is possible to imagine, the more steps ahead are made, the more inaccurate the path followed is. This result will be showed better in the next chapter.

Adopting these two solutions, the algorithm works better. An example of result is showed in the following sequence of pictures. In Figure 17, the blue states are the ones that are predicted (in this case it is 2-step ahead predictions) while the red bounding boxes are the states captured while comparing the current bounding box with the environment.

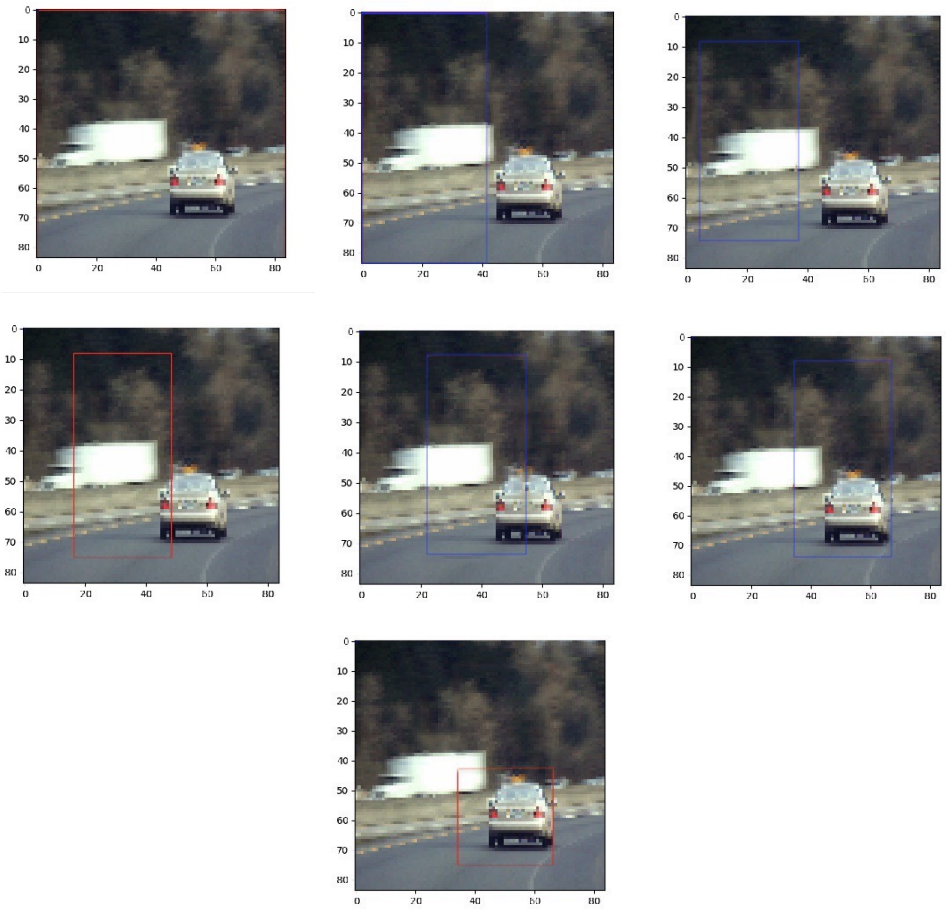


Figure 16: A detection process example. The blue bounding boxes are reported here, but they are the leaps that the algorithm makes.

4.2 Moving the agent window

There was a further issue when predicting the actions. The algorithm was predicting well but at first the leaps were not visible at first. The problem was that the network was able to

predict the future actions perfectly. However, the algorithm was understanding that the final action predicted had to be applied to the first initial state, since the window was not changed.

To solve this problem, the method adopted was to change the coordinates of the agent window. In other words, the algorithm was not knowing the image it was working on, but the agent window (which is given by 4 number, x_{min},y_{min},x_{max},y_{max}) was updated following the actions predicted. In fact, these actions were recorded in a vector of n-elements (depending on how many steps ahead the predictions were made for) and the agent window was moved according to the values of the actions. This process has a very low computational cost, since it is just adding, subtracting, multiplying and dividing numbers. As already mentioned, after n leaps the algorithm captures the image inside the updated bounding box, in order to not lose accuracy.

CHAPTER 5

RESULTS AND CONSIDERATIONS

In this chapter, we will analyze the different results obtained with the algorithm and solutions assumed before. There are 4 types of results. The detections do not improve strongly, the testing accuracy is the same as the one showed in [3]; however, we will see that there is a drastic improvement in the average speed.

5.1 Results on the dataset

There are different combinations we want to show. First of all, we wanted to test our algorithm on the entire dataset. However, we will discuss every result we obtained depending on the number of leaps the algorithm performs. This will bring differences in accuracy and speed. In the comparison, we show the results of the RCNN, the model-free algorithm used in [3] and the two algorithm we used in our project. We distinguished the two model-free approaches since in [3] they used the entire dataset (more than 19 thousand of images), while in our case, since the simulations were performed locally, we used part of the images (around 6000 images). For this reason, the model was trained "better" in the previous work and it performs better. However, the purpose of this project was to show the decrease of complexity when using our approach, so the comparisons are legit.

The first experiment was conducted with a small number of leaps, 2. The expectation is that the algorithm works as well as the model-free one and the speed is slightly higher than the model-free algorithm.

method	aero	bus	car	cat	chair	cow	dog	horse	person	sheep	sofa	train	mAP
RCNN	64.2	62.6	71.0	60.7	32.7	58.5	56.1	60.6	54.2	52.8	48.9	57.9	56.7
Model-free [3]	55.5	56.5	58.8	55.9	21.4	40.4	54.2	56.9	45.7	47.1	41.5	46.1	48.3
Our Model-free	45.5	47.0	44.0	48.1	13.4	18.3	36.7	25.3	39.4	12.1	24.4	33.9	32.4
Model-based	41.3	43.0	43.9	44.1	13.7	12.0	32.4	24.6	38.5	9.1	24.4	29.5	29.7

In the following, it is plotted the average precision with respect to the model-free. As it is possible to see, the mAP is sligthly inferior (2% less).

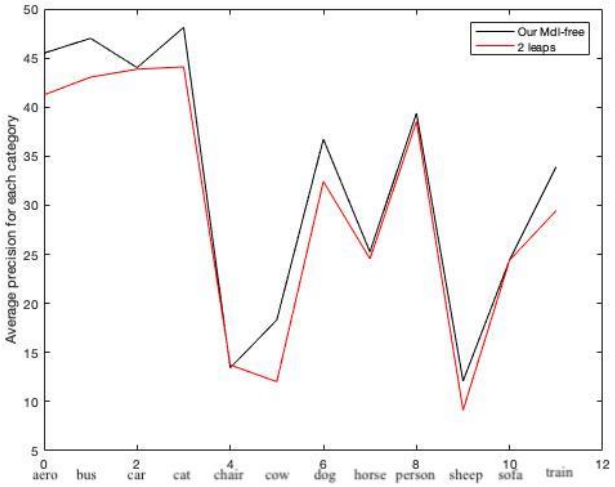


Figure 17: Average precisions for all the categories with 2-leap predictions.

However, the speed per image is already increased. In [3], it is shown that for 200 steps, the average time spent on one image is 1.54 s/image. However, in our case, the model-free algorithm has a testing processing time slightly higher, which is 1.88 s/image. If we repeat the same experiment performing 2 leaps, the average time spent on one image is 1.09 seconds per image.

In the following plot, it is shown the average precisions for more than 2 leaps. As it is possible to expect, the average precision starts decreasing, while the average speed increases.

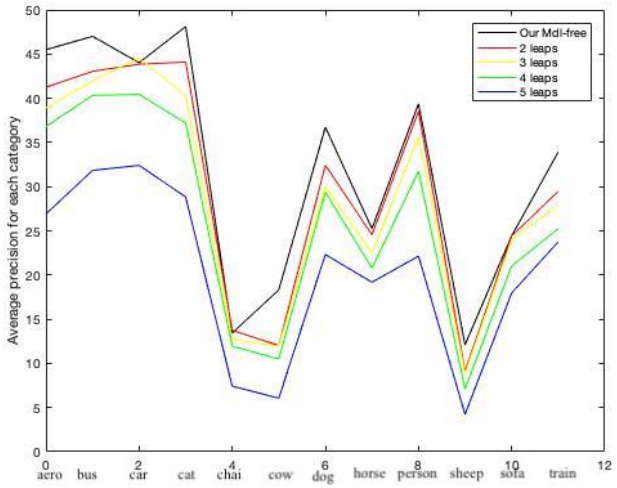


Figure 18: Average precisions for all the categories with more than 2 leaps.

Now we show how the speed of computations increases. The experiment that shows the speed of the algorithm is the same showed in [3]. In the following picture it is shown a graph of the average time spent for 200 steps on one image, considering all the categories. As one can expect, the average time decreases as the number of leaps increases. This result is actually the most important one. As one can expect, the more leaps the algorithm does, the less accurate the path followed will be. However, since the CNN is bypassed for most of the steps, the average speed of the computations per image increases. The speed changes a lot image by image, since it depends on how many steps per episode have to me made.

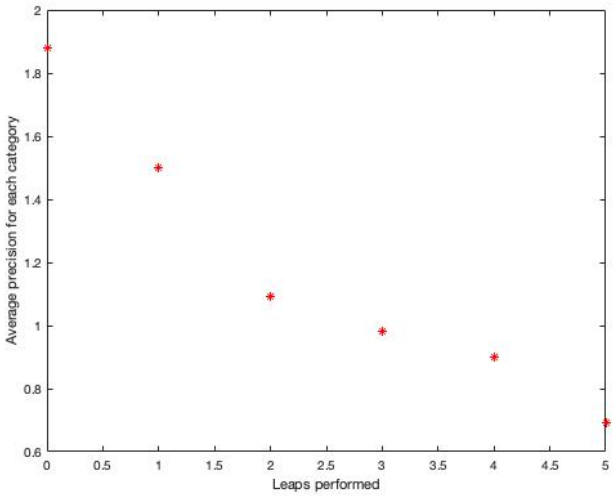


Figure 19: Average time spent to perform 200 steps per image.

5.2 Some examples

We show now some examples for different categories. In partiular, the categories bus, aero, person and sofa are shown. As it is possible to see, "jumps" are performed: more precisely, for these cases, 2 leaps are performed.

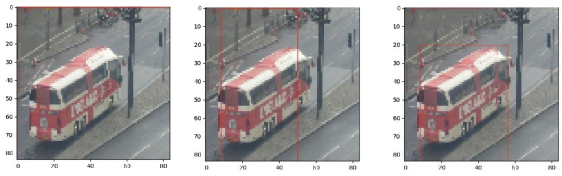


Figure 20: Bus category.

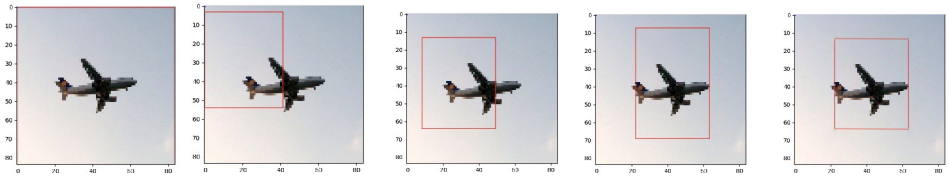


Figure 21: Aero category.

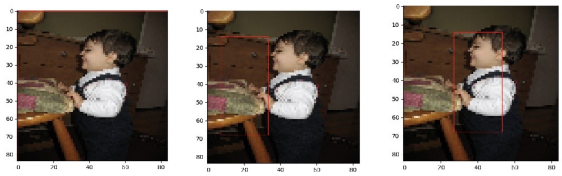


Figure 22: Person category.

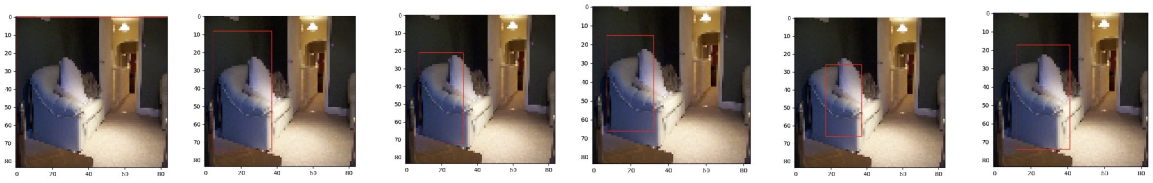


Figure 23: Sofa category.

5.3 Considerations on the behavior of the algorithm

5.3.1 Following the same path

The first result is the one that we expected: the path followed by the model-based algorithm is the same as the one followed by the model-free algorithm.

As it was already mentioned previously, the designed network reaches 78.8% of average training accuracy. This result actually guarantees good predictions of the next states.

As already mentioned previously, during predictions, the actions taken are the one corresponding to the highest q-value. In other words, in this case, the exploitation is used. This decreases the randomness of the algorithm and increases the probability of following the same path as the model-based one.

A very good example is showed below, where the predicted actions match perfectly the ones selected by the model-free algorithm.

In Figure 24 it is shown the detection algorithm based on the model-free approach. As it is possible to see from the pictures, the red bounding boxes are able to reach and identify the car on the left.

On the other hand, in Figure 25 it is shown the same image and the same path followed by the model-based approach. Differentely from Figure 16, the predicted states are not showed in order to give more the idea of the leaps. As already mentioned, the path in this case is exactly the same, so there will be no difference in the states shown with respect Figure 18. In this case, the algorithm reaches the same final state and the algorithm works as expected.

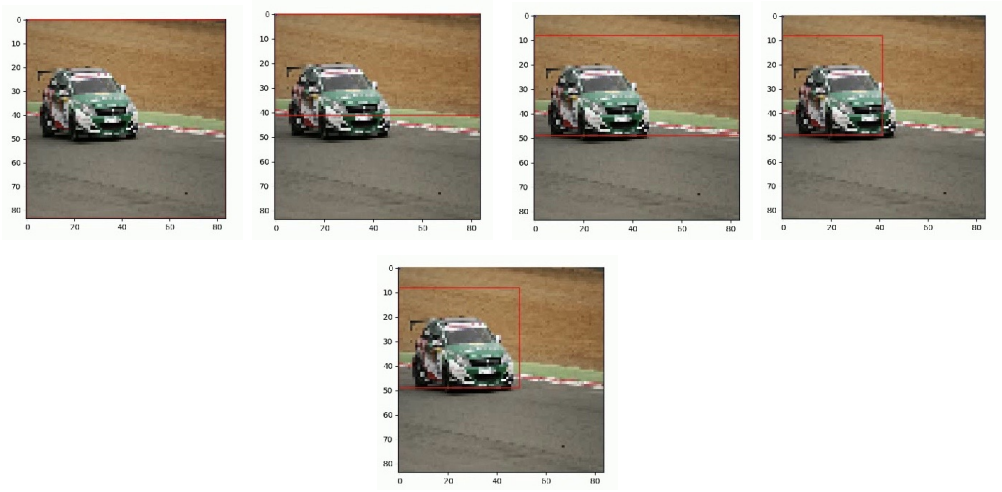


Figure 24: Model-free approach.

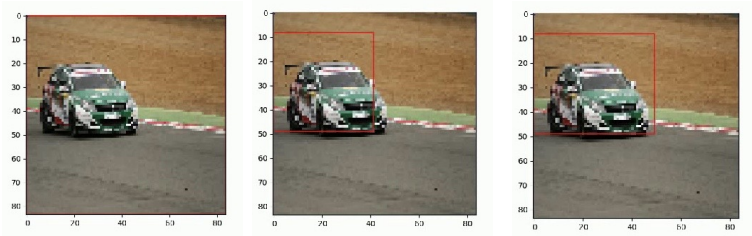


Figure 25: It is shown 3-step ahead predictions algorithm; when the termination algorithm is predicted, the algorithm stops.

5.3.2 Convergence of the detector

It can be shown that even if the predictions are not perfect for a particular current state, the algorithm converges to similar states. This is because of the solution adopted, described previously. In order to keep a good accuracy, every n steps, the algorithm compares the predicted state with the environment. This assumption guarantees a good track of the accuracy during the entire path. This result is shown in the Figures 20 and 21. As it is possible to see,

the two paths are different so are the final states. However, either the two methods are capable of detecting the target, which in this case is the car on the left.

This is a very positive result, since it shows how consistent our solution is and how robust the system is. In a more general term, this algorithm converges to the final state as well as the model-free algorithm.

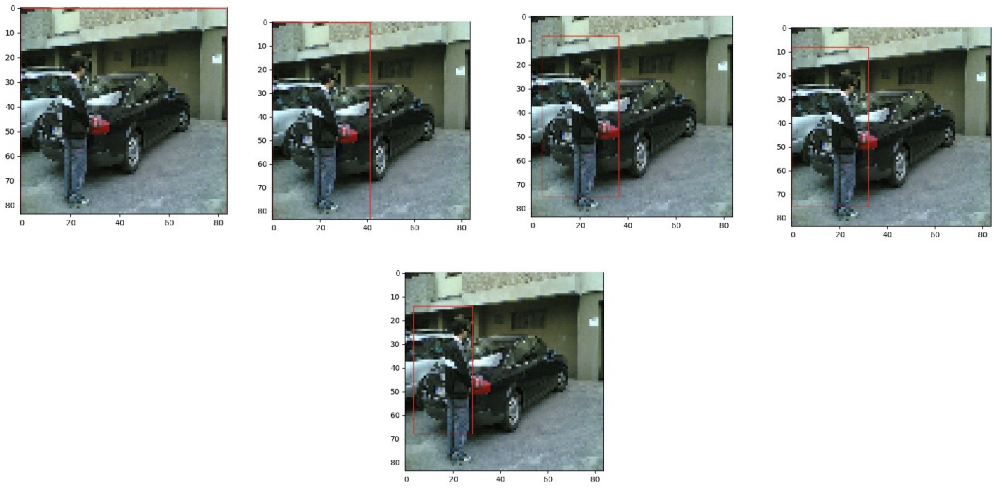


Figure 26: Model-free approach. The car on the left is found.

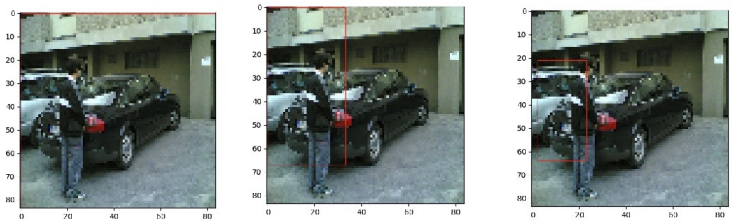


Figure 27: It is shown 2-step ahead predictions algorithm.

5.3.3 Faster convergence with respect to the Model-free approach

Previously, it was discussed the fact that the Q-learning process outputs a probability density function. The action with the largest q-value is more likely to be chosen but it may happen that a random action is taken. In the model-free method if some random action is selected, it will take longer to converge. For this further reason, the predicted actions are chosen picking the one with the largest q-value, as already discussed previously.

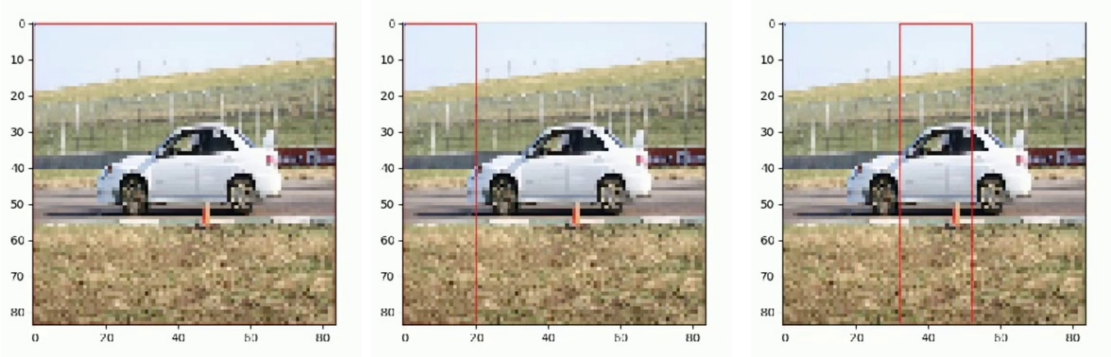


Figure 28: Model-based algorithm converges after three leaps in a forward pass to the car.

In Figures 28 and 29, it is shown an example: it is possible to see that the model-based algorithm reaches the final state before the model-free algorithm (it is 2-step ahead predictions).

Only for the purpose of the result, in the following pictures it is shown not a perfect detection, but the idea here is to show that the model-based algorithm may converge even before the model-free algorithm.



Figure 29: Model-free algorithm takes more steps to get to the final state.

5.3.4 Speed of the Leap algorithm

This result is actually the most important one. As one can expect, the more leaps the algorithm does, the less accurate the path followed will be. However, since the CNN is bypassed for most of the steps, the average speed of the computations per image increases. The speed changes a lot image by image, since it depends on how many steps per episode have to me made. However the average speed per image of paper [3] is 1.88 seconds/image during testing, using the CNN described above. If the number of leaps is increased to 4, the average speed is 0.9 seconds/image, but the average testing accuracy on (for example) the bus category is 31.84%, which is 16% lower than the model-free approach.

This result follows the expectations. It can also be considered the fact that depending on the application we are using this algorithm for, it is possible to prefer speed or accuracy. In our case, since our purpose is to approach real-time simulations, the result of higher speed is a good and positive result.

CHAPTER 6

CONCLUSION AND FUTURE WORK

In the past few years, object detection projects require more and more real-time simulations. The reason is that the goal of machine learning is to become part of the real world and replicate the human functionalites.

A related work was considered, which is the Active Object Detection based on Reinforcement Learning, published by University of Urbana Champaign. This project proposed a new object detection algorithm, which is based on Deep RL, as the name suggests. The agent in this case, was taking actions (move up, move down, scale up,...) based on past experience (exploration) and following what the algorithm has learned (exploitation). In fact, the algorithm follows an ϵ-greedy policy. However, the process is still quite slow (1.88 sec/image), because for every step, the algorithm compares the current state with the environment: for this reason, it is also called model-free algorithm.

The purpose of our work was to try to get this project as close as possible to real-time simulation. The goal was to use a model-based algorithm in order to bypss the CNN. Our motivation comes from the results obtained with Fast RCNN and Faster RCNN, which exploit the fact that the CNN used is the most computationally costly. As a matter of fact, they try to bypass the CNN in order to speed up the process.

Our model-based approach tries to overcome the computational issues of the CNN by predicting the future states. The algorithm needs a further network, which is less expensive than

CNN in terms of number of parameters (and in computations). For now, the fully connected layer network is the first design of this model. We think that our approach is the right way to get closer and closer to real-time simulations. However, better designs of the neural network, such as the use of GANs are kept as future work. We think that decreasing complexity on the neural network will bring to real-time simulations, built on Object Detection with Reinforcement Learning.

As expected, the predictions described above can make the algorithm go faster. However, it is possible to increase the speed by performing more leaps on the images. This brings to have less accuracy but it can be a good result for some applications where speed is more important than accuracy. As a possible future work it can be considered to improve the accuracy, independently from the number of leaps the algorithm performs.

CITED LITERATURE

1. Lee, J., Bang, J., and Yang, S.-I.: Object detection with sliding window in images including multiple similar objects. In 2017 International Conference on Information and Communication Technology Convergence (ICTC), pages 803–806. IEEE, 2017.

2. Krizhevsky, A., Sutskever, I., and Hinton, G. E.: Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097–1105, 2012.

3. Caicedo, J. C. and Lazebnik, S.: Active object localization with deep reinforcement learning. In Proceedings of the IEEE international conference on computer vision, pages 2488–2496, 2015.

4. Girshick, R., Donahue, J., Darrell, T., and Malik, J.: 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.

5. Redmon, J., Divvala, S., Girshick, R., and Farhadi, A.: 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.

6. Girshick, R.: Fast r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 1440–1448, 2015.

7. Ren, S., He, K., Girshick, R., and Sun, J.: Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.

8. Fukagai, T., Maeda, K., Tanabe, S., Shirahata, K., Tomita, Y., Ike, A., and Nakagawa, A.: Speed-up of object detection neural network with gpu. In 2018 25th IEEE International Conference on Image Processing (ICIP), pages 301–305. IEEE, 2018.

9. Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C.: Ssd: Single shot multibox detector. In European conference on computer vision, pages 21–37. Springer, 2016.

CITED LITERATURE (continued)

10. Lawrence, R.: Using neural networks to forecast stock market prices. 01 1998.

VITA

NAME	Edoardo Roba
EDUCATION	
	Master of Science in "Electrical and Computer Engineering, University of Illinois at Chicago, May 2020, USA
	Specialization Degree in " Mechatronic Engineering ", Jul 2020, Polytechnic of Turin, Italy
	Bachelor's Degree in Electrical and Computer Engineering -
	Electronic Engineering, Jul 2018, Polytechnic of Turin, Italy
LANGUAGE SKILLS	
Italian	Native speaker
English	Full working proficiency
	2019 - IELTS examination (7.0)
	A.Y. 2019/20 One Year of study abroad in Chicago, Illinois
	A.Y. 2018/19. Lessons and exams attended exclusively in English
SCHOLARSHIPS	
Spring 2020	Thesis Research
Fall 2019	Italian scholarship for final project (thesis) at UIC
Fall 2019	Italian scholarship for TOP-UIC students
TECHNICAL SKILLS	
Basic level	Hardware Security and Trust
Average level	Convex Optimization, Control Theory
Advanced level	Python, Matlab/Simulink, Model-Based Design
WORK EXPERIENCE AND PROJECTS	
2019/20	Knowledge of Machine Learning and Neural Networks

VITA (continued)

	Design of Reinforcement Learning algorithms (Checkers, Labyrinth escape) and Supervised Learning algorithms (SVM)
2017/18	Other Experiences:
	Programming WiFi modules ESP8266 on Arduino (server-client communication), Teachin Assistant at Assocam Scuola Camerana