YALE-NUS COLLEGE

CAPSTONE

Reinforcement Learning for Unsupervised Object Localization

Author:
Aaron PANG

Supervisor:
Prof. Robby TAN

March 26, 2018

Declaration of Authorship

I, Aaron PANG, declare that this thesis titled, "Reinforcement Learning for Unsupervised Object Localization" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

_____

Date:

_____

YALE-NUS COLLEGE

Abstract

Bachelor of Science (Hons.)

Reinforcement Learning for Unsupervised Object Localization

by Aaron PANG

Finding relevant objects in a given environment is an important task for both humans and artificial intelligence agents. Within a given scene a variety of objects can appear at a variety of sizes and locations, increasing the difficulty of the task. Traditional solutions thus usually used a sliding window approach or used features at several scales of the input of the image. Current state of the art deep learning solutions aim to use a variety of convolutional neural networks to solve the problem, yet these require vast amounts of hand labelled data to succeed. This capstone project thus attempts to solve both problems by applying reinforcement learning to the task of object detection. Our proposed agent thus takes in an input image and emits a proposed set of candidate bounding boxes by searching for regions it is most confident an objects exists within, rather than by using hand labelled bounding boxes as a comparison. This agent initially performed poorly when trained on synthesized MNIST data, but when trained on PASCAL VOC 2012 data it is able to produce qualitatively acceptable bounding boxes.

Acknowledgements

Thanks to my friends Silvia and Pratyush for always providing emotional and technical support over the course of this project.

Thanks to all my other friends I haven't listed for always listening.

Thanks to Robby for his advice and bringing my attention to COW PAPER.

Thanks to Maurice Cheung for always listening to us and being a helpful member of our community. Additionally I am also greatful to him for reviewing drafts of my capstone when my advisor was unable to.

Contents

To my family for always putting up with me...

Chapter 1

Introduction

1.1 Motivation

Society more than ever is governed by photographic images. Thus in order to better understand the billions of images uploaded daily, computer vision systems have also grown in relative importance. This means they must be able to extract exact locations of important features within an image. By doing so we can first individually identify multiple elements of a complex image, and then only extract its meaningful components. As a result we can improve the utility of surveillance cameras, autonomous vehicles and satellite cameras. Tasks such as reviewing security footage can be further automated by only including clips with humans or objects inside.

Current state of the art systems such as Faster R-CNN [1] and SSD [2] however require huge amounts of labelled data to train. Datasets such as MS-COCO [3] require thousands individuals to draw boxes around portions of an image that they deem relevant to be included in the training data. Additionally these images may not be able to fully capture all types of objects that are relevant, nor would they be able to capture the ways in which objects are placed or obscured in all sorts of images. On a whole training models for object detection and classification is incredibly difficult because of the required volume and type of data needed.

This project aims to address these issues by creating an unsupervised object localization methodology. This system will learn to draw bounding boxes over a given series of test data, without needing to be told either what is present inside our images or its set of bounding boxes. The aim of this system is not to outperform existing supervised solutions, but to provide a proof of concept that such an unsupervised

system can be created in the first place. Such a system would also reduce the number of proposed object candidates as compared to current systems.

1.2 Background

1.2.1 Deep Learning

Deep learning is a subset of machine learning that is primarily concerned with neural networks. The basics of a neural network are as follows. For a given input vector x with each of its columns representing some sort feature, we multiply it with a matrix of weights W, add it to a bias term b and then apply a non-linear function σ such as *sigmoid* or *ReLU*. Sigmoid is defined as $\sigma(x) = \frac{1}{1+e^{-x}}$, while relu is $ReLU(x) = \max(0, x)$. These non-linearities act as step functions for our network, allowing our network to model non-linear relationships in our data. This was largely inspired by how biological neurons also appear to utilize a step function.

$$f(x) = \sigma(Wx + b) \tag{1.1}$$

Each layer within a neural net represents an additional set of weights and bias we apply consecutively to our input. In order for this system to learn then we must thus use a process called Stochastic Gradient Descent (SGD). Stochastic gradient descent is method of non-convex optimization that can be summarized as follows. After applying our input vector to several layers of our neural network we will get some output vector y. We can then compare y with $\hat{y}$, also known as the ground truth vector through a loss function such as mean squared or cross entropy loss.

$$Loss_{MSE} = (y - \hat{y})^2 \tag{1.2}$$

$$Loss_{CE} = -\sum_i y_i log(\hat{y}_i) \tag{1.3}$$

We can take the derivatives of each weight and bias term in our network with respect to this loss function, finding which direction to move our weight matrices

and bias vectors to decrease this value for loss. We multiply this derivative with some scalar α, also called our learning rate, and minus it to our current value.

$$W_{new} = W_{old} - \alpha * \frac{\partial Loss}{\partial w} \tag{1.4}$$

Repeating this process over millions of time steps, we will eventually find values for weight and bias that minimize our loss. In order to make this process stochastic gradients are also calculated with respect to a randomly sampled batch of data. This induces a slight amount of noise into our model and minimizes the chance our network gets stuck at locally optimal solutions.

Other optimizers then such as RMSProp [4] and Adam [5] apply several other ideas to the optimization process. Both were chosen for this capstone for different experiments. One shortcoming both of these optimizers wanted to overcome was the selection of a suitable learning rate for training. Adam and RMSProp address by introducing learning rate annealing, which decreases our learning rate over time with respect to previous values either linearly or exponentially.

Another idea that these then aimed to introduce was momentum. This means that instead of only considering the gradients with respect to the current input, both these optimizers choose to keep a fraction of the previous gradients and sum them to our current value. This dampens the oscillations of directions in which our network chooses to go towards. Empirically this has resulted in better performance over time. Adam improves upon other optimizers by taking into account past values to calculate both first second-order moment of our gradients to better decay the value of its learning rate.

An improvement to neural networks is called dropout. This is a layer that randomly zeros out many of a layer's neurons during training. Our network learns a more robust mapping between each layer, relying on less of its parameters at each step of training. This both helps our network overfit less and increases our accuracy at evaluation time.

1.3 Image Classification and Convolutional Neural Nets

Image classification is one of the few tasks in supervised machine learning that has become largely solved. The idea is that for a given image a system will do its best to determine a class label, eg. whether its a dog, cat, car etc.

1.3.1 Kernels

Starting in 2012 with AlexNet[6] most modern classification systems heavily rely on the use of convolutional neural nets. These are similar to regular neural nets in that they consist of a series of weights and bias vectors and apply these to non-linear functions. The crucial difference is that they aim to apply a matrix of trainable values, also known as a kernel, across an image to extract features from it. Each kernel has a specified width and height, and sweep across our image at intervals determined by its stride. At each layer we stack many of these kernels together to create a volume of extracted features. This also means each kernel can learn a different mapping of our image.

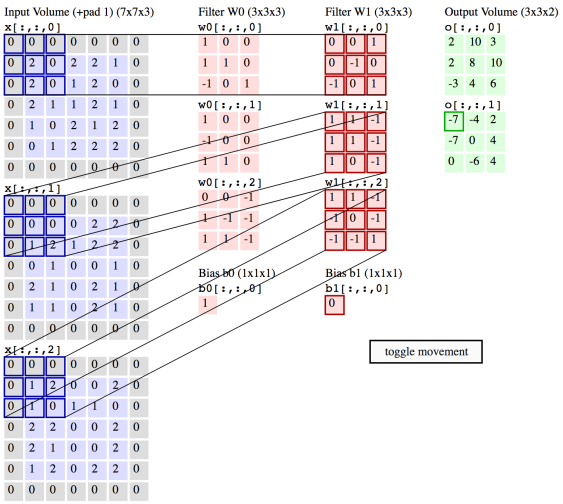

FIGURE 1.1: Example of a CNN [7]. Here the blue is a matrix representation of our input image. Each red square is one of our kernels that are slid across our input image. The green is the resultant output.

Through SGD we can then tune what exact values are in each kernel, which over time allows our system to learn to detect features such as edges, eyes and fur from a given image. These kernels are weights which are shared throughout the image, allowing us to detect the same items across all sections of the image. Convolutions

also allow us to reduce the size of our model. As compared to feeding each pixel value into a regular feed forward neural net, by sharing the same kernals

1.3.2 Pooling

Often pooling layers are also used in CNN architectures. These further reduce our number of features, for example by taking the maximum value from a sliding window of the feature map. After a series of these convolutional-pooling we then flatten our feature map into a 1-dimensional vector and pass it onto a regular feed forward neural network, leading us to predict the class label of the image.

Today's state of the art classification engines such as Xception [8] can easily differentiate between tens of thousands ambiguous classes. A network is therefore considered 'deep' when it contains many layers, sometimes as many as 1000.

1.4 Reinforcement Learning

Reinforcement learning(RL) is a field of machine learning that aims to create artificial intelligence agents capable of learning from experiences. (see section 2.1 for more details on formulas and derivations.) Most RL problems are formulated as Markov Decision Processes (MDP). These problems have a state s which our agent can observe. This state is markovian in the sense that every state fully determines the problem at hand without requiring a memory of previous states. At each state our agent can choose a series of actions a, which then will lead it to a particular set of rewards as determined by the environment. This is similar to how animals learn from trial and error.

Our here is to find a policy π that will maximise these rewards obtained by the agent. In our case we do so by accurately estimating the rewards obtained by taking a particular action at a given state, and choosing only the actions which maximize this value.

The advantage of RL based solutions is that they can solve problems without knowing the problem before hand. The same methods used to solve Atari games were also applied to play Go[9].

Examples of recent successes in RL include Atari games [10] and AlphaGO [9]. While games have been the drivers of RL, it also been applied to other domains such as ad pricing [11] and even e-commerce [12].

Compared with deep learning however RL faces an entirely different set of challenges. While the labelled datasets used in classification or speech recognition are highly structured, the observations made by an RL agent heavily depends on its current exploration and policy. For example an agent that immediately dies in the early rounds of a game can never hope to optimize for the final end game. The search space of state, action pairs also vast exceeds the data within object classification, requiring orders of magnitude higher compute in order to succeed.

1.5 Object Localization / Detection

Inspired by successes in classification other researchers applied these ideas in classification to object localization and detection tasks. Localization here is defined as drawing a bounding box over a given image and predicting its class label. Detection however involves drawing multiple boxes over multiple objects within the same image. This project focuses on the former.

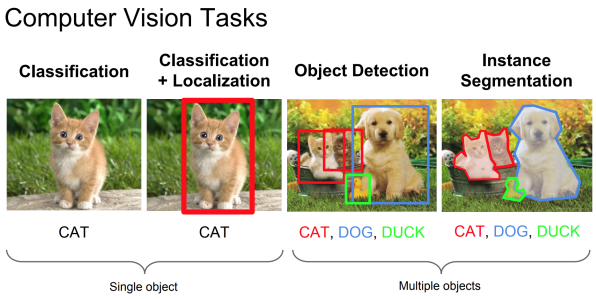

FIGURE 1.2: Overview of typical computer vision tasks [13]

Localization traditionally required an ensemble of techniques to be even tackled and hand crafted features such as SIFT or HOG [14], but modern day deep learning based solutions can solve the entire problem end to end. Through a series of convolutional networks and smart use of regressors, object detection today can also be done in real time.

1.6 Contributions

The contributions of this capstone are as follows. We have proposed an unsupervised RL-based agent that is capable of achieving relative success on the task of Object Localization. The idea of perhaps using RL-based agents in object detection was first proposed by myself after watching several discussions and lectures online on the topic. My advisor sent me a the following papers: Active Object Localization with Deep Reinforcement Learning[15] and Attention-Aware Deep Reinforcement Learning for Video Face Recognition[16] to survey.

I thus explored all other resources mentioned in this capstone and did my best to give an introduction to reinforcement learning to my advisor and give a basic overview of object detection to him. Both of these were fields he was not intimately familiar with.

Afterwards we both agreed that confidence intervals would be the best approach to solving the issue of object detection in an unsupervised manner, but it was left up to me on how to obtain these values for each time step our Q-Network would operate. It was left to me to determine the architecture, the environments and data passed to this network as well. Most importantly I had to also determine the training regime for the network and how best to implement it.

Chapter 2

Existing Solutions

2.1 Reinforcement Learning

In reinforcement learning we observe a particular state, take some action and get a reward and new state in return.

$$s \xrightarrow{a} r, s' \tag{2.1}$$

More formally this can be illustrated by the Bellman equation below [10].

$$\begin{aligned} Q(s,a) &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ Q(s,a) &= r_t + \gamma \max_a Q(s',a) \end{aligned} \tag{2.2}$$

Here Q is our computed table of all possible reward values for s, a state action pairs. The values stored within Q are also known as q-values. γ is a discount factor that is less than 1 used to make sure the q-network accounts for future consequences. It is less than 1 to ensure q-values do not approach infinity. A policy $\pi(s)$ is defined as a probability distribution of actions at any given state. Since storing such a table would be computationally intractable for large problems, but it can be approximated using a neural network.

Traditionally solutions simply use equation 2.2 to assign values within a table. In deep reinforcement learning we instead use backpropagation to make our network bend towards the same output $Q(s,a) \rightarrow r_t + \gamma \max_a Q(s',a)$

2.1.1 Deep Q-Learner

The Deep Q-Network (DQN) [10] architecture was what laid most of the groundwork of recent deep learning based reinforcement learning solutions.

Architecture

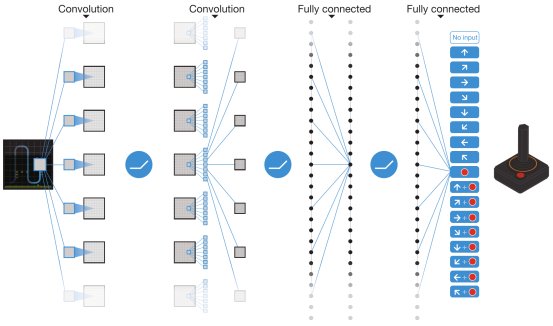

FIGURE 2.1: DQN architecture[10]

As presented in the original paper the DQN architecture takes in a series of 4 atari game frames that are resized and grayscaled and estimates the rewards obtained by selecting a particular action using a neural network.

It is also important to note that the DQN is an example of online-learning, in the sense that it learns as it experiences an environment rather than learning with respect to a history of labelled data.

During training this network uses what is known as an ϵ-greedy policy. At the beginning of training ϵ is set to be some parameter less than or equal to one. This determines with what probability our network choses a random action. Over time this value of ϵ is gradually decreased. When not choosing randomly our network will simply choose the action at a particular state with the highest estimated q-value. ϵ however is never set to 0 during training, ensuring that our network continues to have some degree of exploration. At inference time we would of course only act with respect to the q-values estimated by the network.

Training

Important training methods introduced by the DQN are experience replay module and target networks. Experience replay aims to offset the short sightedness of our

network by letting it see previous actions sampled from a memory module. This is because if we only train our network with the current examples it sees, these tend to be highly correlated with one another and makes our network overfit.

Target networks are thus a copy of the current estimate of the Q function that is updated at slower intervals. Since our Q-network is updated at every time step it has a tendency to fluctuate wildly. This means between consecutive timesteps its estimates of q-values for the same input state may greatly change and potentially lead to feedback loops that we do not want. As a result we use our target network as our estimate of $max_a Q(s', a)$ on the right hand side of 2.2 during training instead.

In the initial phases of training our network's estimates of rewards may also greatly diverge. As a result the authors also found that limiting the values of gradients that are backpropagated between $-1, 1$ reduces the variation of the network and allows it to converge faster.

All of these small improvements allowed the DQN to perform at above human levels on a series of 49 atari games.

Drawbacks

However drawbacks of the DQN are numerous and varied. It requires an immense amount of data from an environment, on the order in the tens of millions of frames to train a single agent. This translates to the network requiring nearly 50 hours of game time to learn to play a simple game such as pong or breakout[17]. Like many other neural network based reinforcement learning solutions it is also highly sensitive to initial conditions and hyper parameters. For example even setting the paddle position differently can severly hamper its performance [18]. It also means that without the right set of hyper parameters, our model will fail to learn an appropriate policy in the first place **Alexipan** More strangely multiplying the rewards of our network by a scalar value can change it behaviour [17].

2.1.2 Alternatives to the DQN

Other methods that are now commonly used in deep reinforcement learning research include policy gradients, and actor-critic methods. Neither improvements

were used within this capstone. Some improvements that were explored, such as prioritized experience replay and dueling q-networks, will be touched upon later.

In policy gradients, the network is used to estimated a policy $\pi(s)$ for our network. It ignores what the actual rewards will be obtained by taking a particular action. Over the possible starting states of the environment, we can assess a model by estimating the total rewards obtained by following our policy. While this quantity is difficult to estimate, calculating its gradient however is not. Policy gradients are an exploration of numerical optimization methods such that the rewards obtained by following policy are monotonically increasing[19].

Actor-Critic methods thus combine Q-networks with policy gradient methods[19]. For each state the critic network assesses the reward values for each potential action while the actor selects an action. Whenever rewards diverge from the expectations of the critic we prioritized this experience and train with respect to it. This difference is also known as the advantage. This method also allows us to utilize multiple actor networks simultaneously, allowing each network to explore its own set of actions that hopefully do not correlate with one another. This significantly decreases training time and total rewards obtained in Atari games.

Improvements that are utilized in this capstone are the ideas of the Double-DQN(DDQN)[20] and prioritized experience replay[21].

Prioritized experience replay improves on the original by looking only at the differences of our network and the rewards it obtains. The idea being Q-value estimates that greatly diverge from what our agent observes should be trained on more than estimates that are correct. For each experience we then calculate the error between the network and reality. This error is therefore used to balance a modified binary heap, allowing us to efficiently sample only experiences that have the highest error.

The DDQN aims to overcome systemic bias and value overestimation in the DQN. For example imagine a scenario in which the true q-values for a given state are identical. The network however is inherently noisy in its estimates. During an update step, because of the max in $\max_a Q(s, a)$ the action with the largest positive error will be selected. This over time can lead to that particular action to be favoured. The proposed solution is to instead use an alternative Q-Network to find the index of action that is maximised, and another one with which to calculate actual q-values.

$$Q_1(s,a) = r_t + \gamma Q_2(s', argmax_a Q_1(s', a)) \tag{2.3}$$

On atari this allows our network to achieve almost double the scores of the original DQN architecture. The target networks in the DQN can be used to estimate values of Q_2.

2.2 Object Detection

In order to understand the history of object detection and localization it is necessary to revisit the PASCAL VOC[22] challenge and ImageNet ILVRSC[23] of the early 2010s. The main metrics to watch out for in object detection are IOU scores, defined as the intersection between the proposed boxes and the actual ones. The other metric is mAP or mean average precision. This measures how many of the actual proposed objects are correctly classified.

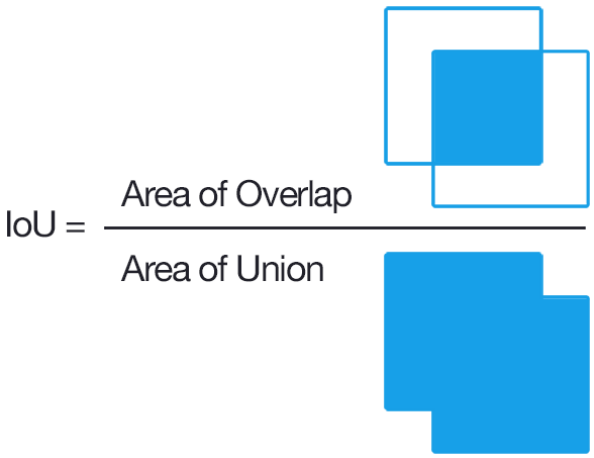

FIGURE 2.2: How IOU scores are calculated[24]

2.2.1 R-CNN

R-CNN was the first comprehensive solution to object detection that utilized deep learning. The main insight was to use a pretrained CNN to extract features and to perform classification over smaller windows. Potential regions were calculated using 'selective search'. Selective search groups together pixels which have similar colours and intensities. Running this several iterations generates 2000 regions

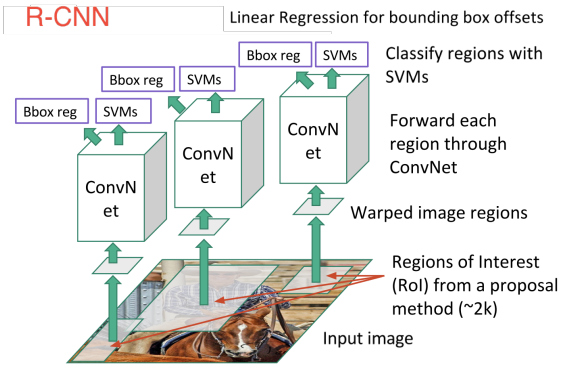

FIGURE 2.3: R-CNN Overview[25]

per image after a minute of run time. Each proposed image would be resized and ran through our pretrained CNN. These extracted features are then classified via an ensemble of SVMs.

FIGURE 2.4: Selective search in action [25]

Drawbacks to this approach was the difficulty in training. As it was composed of multiple components, each segment had to be trained on its own. The significant run time of selective search also prevented this network from training quickly.

2.2.2 Fast R-CNN

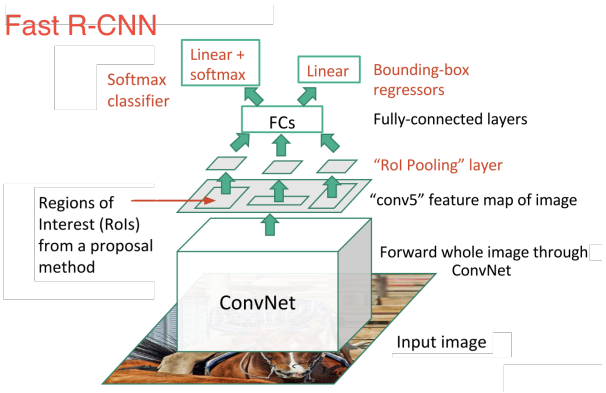

FIGURE 2.5: Fast R-CNN Overview[26]

Fast R-CNN attempted to improve things by sharing as many resources as possible. It does so by running the CNN only once and sharing that block of convolutional features. The old SVMs were now also replaced with fully connected dense layer, followed by a softmax for classification. This network however still relied upon selective search in the original image to find regions of interest. Each proposed region from search would then be projected onto a block from the convolutional block computed earlier. In order to deal with varying sized feature sizes, each block is pooled via ROI pooling into a fixed sized vector. We would then find the corresponding convolutional feature set for a given region of interest and pool that into a fixed sized vector. ROI pooling works by subdividing a block into equal sized areas as best as it can, then finding the max value in each block. Therefore all pooled feature blocks would become the same size afterwards. As a result this method greatly improved the speed of detection by limiting the times the CNN is used.

2.2.3 Faster R-CNN

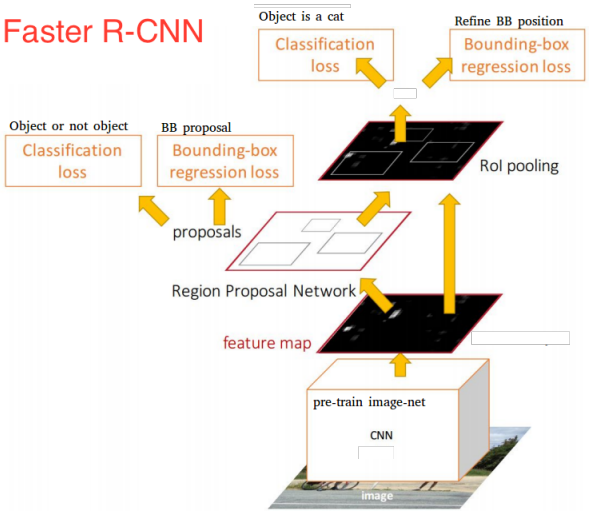

FIGURE 2.6: Faster R-CNN Overview[1]

Further developments sougth to get rid of selective search entirely. In Faster-RCNN selective search is replaced by a region proposal network (RPN). The RPN acts as follows, we first preselect a series of k possible shapes of bounding boxes of varying sizes. The RPN will then scan through a window of our block of convolutional features, rating the likelihood there existed an object within one of the k

possible bounding box shapes within the original image.

The most likely features are then once again ROI pooled and passed unto a fully connected layer.

Why Faster-R-CNN remains relevant to our discussion is because it serves as a baseline for most object detection task. While some newer developments may be faster, they rarely if ever outperform Faster-R-CNN's baseline performance. Our proposed RL solution to object localization can be used to replace the RPN while limiting the number of candidate windows.

2.2.4 YOLO/SSD

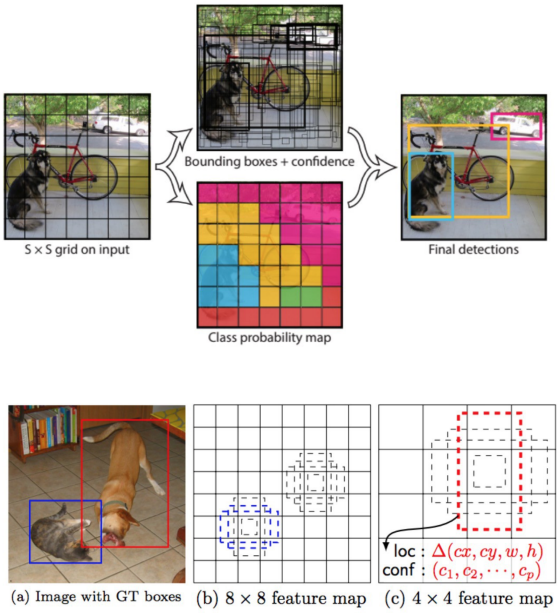

FIGURE 2.7: YOLO[27] and SSD[2] Overview

YOLO and SSD attempt to tackle the problem from an entirely different point of view. Previous systems would always generate proposed regions then pass them into a classifier. YOLO and SSD attempt to do both steps simultaneously. YOLO does so by dividing our image into a large grid. It then predicts a set of proposed bounding boxes with coordinates for each grid, alongside a class prediction for the box as a whole. The system then intelligently combines all this information together, drawing a box over entire sets of grids and corrects them slightly in order to localize them.

SSD achieves similar results by modifying the RPN step of Faster R-CNN. It does so by not only proposing a series of bounding boxes directly from the block of convolutional features, but also running each box through a classifier. This skips the need for ROI pooling layer. The main issue here is that there will be an excess of proposed boxes during training. Thus we only select the box with the highest IOU to be the correct one.

2.2.5 Limitations

As of 2018 state of the art models still only achieve unremarkable mAP (mean average precision scores) (40%)[27] on MS COCO object detction challenge. This means that out of all the times the model was confident it detected a particular image, half the time it has actually detected the class wrongly.

2.3 Reinforcement Learning and Object Detection

The focus of this capstone is on the two papers Active Object Localization by Caicedo [15] and Hierarchical Object Detection by Bellver [28] Both formulated the search for bounding boxes as a markov decision process and aimed at reducing the number of proposed boxes as compared to other object detection frameworks. While ultimately their performance falls short of other previously mentioned systems these are still worth revisiting for combining two diverging fields of machine learning.

The main difference between the two papers are in the sets of actions their network can undertake. In Caicedo's model the bounding box window is able to move vertically and horizontally across our canvas, while also shrinking and growing if need be. In Bellver's model every action aims to shrink the bounding box, this leads it to be considered a form of hierarchical search.

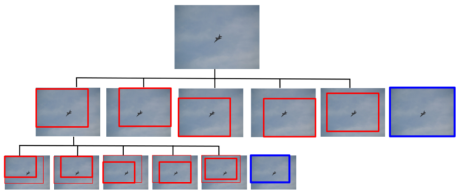

FIGURE 2.8: Sample of actions undertaken by Bellver[28]

The features of each network also slightly differ. Caicedo extracts a new set of features at each time step by resizing the current bounding box and running it through a pre trained CNN. Bellver on the other hand also gives the option of pre-computing a single block of convolutional features that are shared for each time step. This means for a proposed bounding box we have to extract the block of corresponding features from its computed convolutional block and use ROI pooling before passing it onto our Q-Network. Otherwise there still is the option of computing a new set of features for each time step as well.

From empirical evidence it appears that computing a new set of convolutional features at each time step yields better performance at the cost of run time. However as our RL agent reduces the search space for bounding boxes significantly this difference in compute is almost negligible.

Both also differ from the DQN by only keeping a state vector of the previous four actions as integers, rather than keeping a feature representation of the previous bounding boxes.

Both also structure their environments in a similar fashion. Rewards are kept to be binary values to improve predictability and accuracy of the q-network. This means whenever the q-network proposes a bounding box that has higher IOU values it is rewarded the same amount regardless of the magnitude of change. Once a certain IOU threshold is met both environments terminate and emit a significantly larger reward. Otherwise both have a cut off number of timesteps, after which the environment will terminate on its own.

Overall Bellver's model is able to find the majority of objects within 10 time steps, a two orders of magnitude reduction in bounding box proposals.

Chapter 3

Experiments

Each experimental section delves deeper into the methods, results and discussions of each trial. However in general all three share a similar overview.

Rather than rely on IOU scores or class labels from the training data we primarily rely on confidence signals from a pretrained CNN network. These are obtained by running a softmax layer over the output logits of the CNN.

$$softmax(x)_j = \frac{e^{x_j}}{\sum_i^K e^{x_i}}, j \in \{1, \ldots, K\} \tag{3.1}$$

Softmax aims to normalize a vector in an exponential fashion. We use the highest value of the softmax as our confidence score, regardless of its corresponding label.

The q-network functions similarly as to the previously discussed RL based object detection solutions, but instead used to maximise confidence scores.

Rewards were also kept consistent between experiments. Rewards can be obtained between each step of the q-network or when it terminates.

$$R_{step} = \begin{cases} +1 & \text{if } confidence_{new} > confidence_{old} \\ -1 & \text{otherwise} \end{cases} \tag{3.2}$$

$$R_{trigger} = \begin{cases} +3, & \text{if confidence} \geq \eta \\ -3, & \text{otherwise} \end{cases} \tag{3.3}$$

Where η is a preset confidence threshold.

The actions taken by the q-network are directly copied from Bellver. The scale

which our network shrinks its bounding box estimatesis also another hyperparameter that is empirically determined. All of these experiments used a scale of $\frac{3}{4}$ for their actions. This produces successive bounding boxes with some degree of overlap, which has shown to give better performance [28].

State history was also kept to a maximum of the immediate 4 previous frames or actions.

Like most reinforcement learning problems Huber Loss[29] was also used instead.

$$Loss_{Huber} = \begin{cases} 0.5 * x^2, & \text{if } |x| \leq \delta \\ 0.5 * \delta^2 + \delta * (|x| - \delta), & \text{otherwise} \end{cases} \tag{3.4}$$

Huber loss acts as linear loss function for small values, but a quadratic loss function for larger values. This is dependent on the value for δ, set to 1.0 in most cases.

Unless otherwise stated each layer of all networks utilized the ReLU activation function.

Additionally none of the proposed bounding boxes are pased unto trained regressors. In all other object detection frameworks these coordinates are pased onto a regressor in order to correct them. However this is only possible given a ground truth of values from which we want the values to be corrected into.

3.1 MNIST DQN

FIGURE 3.1: Sample images generated for the first experiment

To generate the data, a blank 75x75 canvas was generated. As each digit was of size 28x28, a random width and height position were chosen between $0 - 47$. Afterwards each image was copied over to that randomized coordinate.

The aim of this experiment was to create a q-network capable of localizing to a single MNIST digit on a blank canvas. Using a basic CNN that is trained to recognize digits from the original MNIST dataset, we pass a collection of 4 image crops to a modified version of the original atari DQN.

3.1.1 Methods

Our experience replay module is created at a size of $100,000$ experiences. However in this example these were not populated ahead of time. The architecture of our q-network is as follows. Its input in this case is a stacked 4 frames of resized candidate bounding box images. This initial network also lacks a target network with which we calculate our q-values upon training.

Type	Size	Filters	Kernel Size	Stride	Padding
Input	4x28x28	N.A.	N.A	N.A	N.A
Conv1	32x21x21	32	8x8	1	0
Conv2	64x9x9	64	4x4	2	0
Conv3	32x7x7	32	3x3	1	0
fc1	512	N.A	N.A	N.A	N.A
fc2	5	N.A	N.A	N.A	N.A

TABLE 3.1: Overview of the q-network

Each layer was initialized using random uniform weights.

RMSProp was chosen as the optimizer in this case.

Hyperparameter	Value
Learning Rate	0.0001
γ	0.99
ϵ_{start}	0.9
ϵ_{end}	0.05
ϵ_{decay}	60000
η	0.5
total moves	10
batch size	128

TABLE 3.2: Hyperparameters

Each image is treated as an episode of the game, lasting at most 10 time steps. Training time was about 1 hour per 1000 episodes on a single GTX 1080 GPU with PyTorch 0.3 and CudNN 7.0. The value for ϵ_{decay} was chosen as to allow our q-network to experience each image from the MNIST training dataset at least once.

3.1.2 Results

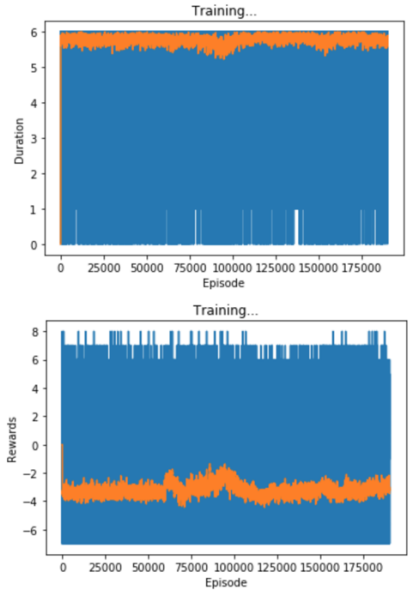

FIGURE 3.2: Length of actions and rewards per episode

This network appears to perform no different to a randomized agent after mild training. Both the number of actions taken per episode and total rewards earned are noisy graphs with no perceptiple trends.

3.1.3 Discussion

While this network performs poorly there are perhaps several reasons why. Firstly the features obtained by the q-network's internal CNN maybe insufficient to produce a localization policy. But as modified version of this network was used to play Atari games, this is highly unlikely.

Additionally this network stores a relative abundance of negative experiences. During the random seeding of the memory module more than 90% of experiences did not lead to a positive reward. This served as the underlying motivation for using a prioritized experienced replay module the next experiment.

3.2 MNIST DDQN + Experience Replay

The goal of this experiment was to see whether alternative training methods would yield more conclusive results. An addendum was that the prioritized experience replay module would be seeded with actions made by a fully randomized agent before hand. Additionally by training using the DDQN methodology the hope was that the q-value estimates would be more accurate.

3.2.1 Methods

The only changes to our hyperparameters was the reduction of total time steps before termination to 5. This was motivated by observations of how an agent is able to localization to a digit in less than 3. This would also hopefully reduce the number of negative experiences the q-network will experience. The target network was also updated at a frequency of 1000 timesteps.

3.2.2 Results

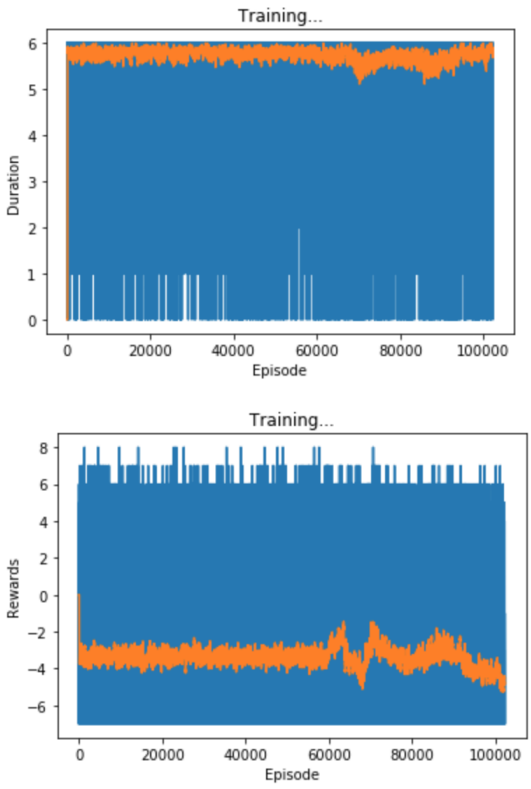

FIGURE 3.3: Length of actions and rewards per episode

As compared to the agent before, there is a clear uptick in performance when the q-network begins to take over. This is because we reach the number of episodes as stated by ϵ_{decay}. However this quickly diminishes after more episodes. This is potentially indicative of a network that forgets its values due to new experiences, thus a lower learning rate and longer exploration period may have proved to be more useful.

3.2.3 Discussion

In both experiments so far it was also clear that the network simply had too many negative examples. Even with a prioritized experience replay module it was not clear that the network fully learnt to localize to anything of interest upon the canvas.

3.3 Pascal VOC 2012+2007

Compared to the previous dataset, PASCAL VOC both contains richer information within each image and contains larger objects overall. This means a significant proportion of the images require little to no zooming in the first place for an object to be considered localized. Recognizing these factors, the following experiments adapts code from https://github.com/imatge-upc/detection-2016-nipsws/.

The main difference between the 2012 and 2007 dataset are the sizes of each image and the number of images for each class. Unlike previous experiments the q-network is also only trained to localize on a specific class, in our case airplanes.

3.3.1 Methods

Confidence scores were obtained via ResNet-50[30] pretrained on ImageNet. ResNet differs from other classification engines in that it approximates the *residuals* of a mapping, rather than a function itself. Through an added skip connection each layer is an output of some $F(x) + x$ rather than a mapping from one layer to the next. Empirically this yielded significantly better performance in classification and allowed for even deeper layers.

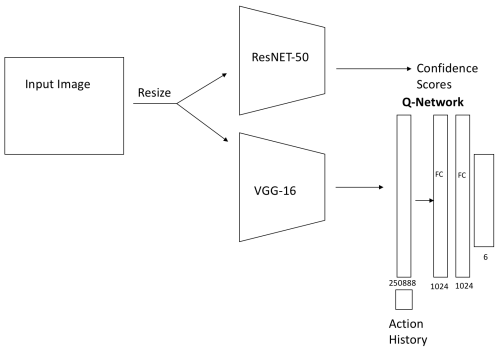

FIGURE 3.4: Architecture overview for this experiment

Since this network is trained only as a regular DQN, our experience replay module is only sampled randomly. Its size is also significantly smaller at only 1000 experiences.

For this experiment the proposed bounding box image was passed into VGG-16 for feature extract at each timestep as this yielded better performance.

Our optimizer in this case was Adam with a learning rate of 0.001. As we were only testing the localization capability of the network rather than its general performance, we restricted to only images with planes. The model was trained for 20 epochs, taking around 20 hours to do so.

This model was created using Keras with a TensorFlow backend. Training time was significantly slower at around $2 - 5$ images per second.

γ was also reduced to 0.9, meaning our network accounts for less of its future rewards. η was increased to 0.8, accounting for the number of images in which our network appears to not take any action.

3.3.2 Results

From a few small epochs it appears that our IOU scores of proposed bounding boxes appears to increase. This is inspite of the fact that this network was never told to optimize for this metric, nor did it ever recieve signals about the IOU scores of the bounding boxes it proposed.

FIGURE 3.5: IOU scores over time

3.3.3 Discussion

Qualitatively this network is largely a success. Looking at a few sample outputs it is clear the network has gained some ability to localize to what it believes to be a plane.

FIGURE 3.6: Examples of proposed bounding boxes

Here the proposed bounding boxes over the planes are of reasonable accuracy and do not ignore important components of the image.

Yet the network is not without its faults. Often times our q-network becomes distracted by other items in the scene. Objects in the foreground that are not planes occaisionally attracts its attention. This is perhaps because our confidence scores are generated indiscriminately by our classifier without regards to what object we actually want detected.

Compared to both Bellver and Caicedo this particular q-network underperforms significantly. The Bellver paper did not include IOU performance of its model.

Most of this performance difference arises because this q-network tends to emit

FIGURE 3.7: Examples of proposed bounding boxes

Model	Value
Ours	0.22
Caicedo	0.5

TABLE 3.3: Comparison of IOU Scores

bounding boxes which are significantly larger. The boxes it proposes need only to be big enough for ResNet-50 to classify them. Depending on the intended usage of this object detection system, finding a slightly wider bounding box may not hamper its performance significantly.

As shown in our results section the IOU score of our model also appears to increase with time. This shows that creating a network which optimizes for confidence in object classification correlates highly our goal of proposing more accurate bounding boxes.

Of note the category of planes in PASCAL is also exceptionally broad. It includes airliners, fighter jets and even rotrary planes. More often than not the most salient portion of the plane are its wheels. Thus our network occaisionally also localizes to only the wheels of a plane.

Improvements to this network would have involved a more rigorous hyperparameter search. As reinforcement learning problems are particularily sensitive to initial conditions and hyper parameters, values such as η could have been better tuned as to suite our needs.

While our network appears to perform well on planes, it is still unclear how it would perform on other classes within PASCAL VOC. This experiment also leaves it unclear whether a unsupervised class agnostic object detector is possible using RL based methods.

Improvements to speed up the network could also be reusing VGG-16 features

for the classification confidence. Since it is clear from Faster R-CNN that 'objectness' score can be directly estimated via convolutional feature maps, using these directly instead of a separate architecture would lead to a 2-3x speed up.

Other factors such as a prioritized experience replay module or dropout could have also been considered.

Chapter 4

Conclusions

This capstone thesis thus explores the applications of Deep RL agents in partially observed markov decision processes outside of videogames. It clearly illustrates that reinforcement learning algorithms have a much wider range of applications than just on videogames and can be applied to unsupervised machine learning tasks.

While the performance of this model may still leave much to be desired, there still remains immense potential in this algorithm that is yet to be explored. This was because the model does not directly have access to IOU scores with which optimize upon. Moreover the bounding boxes it tends to produce are usually larger than the ground truth labels, because this was deemed to be good enough for classification. The goal was to propose a system that could achieve some success in the domain of object detection and localization, not to neccessarily beat state of the art solutions to the problem. The fact that our model appears to improve its baseline IOU scores over time without directly optimizing the value is a success in of itself. The hope is that with such a model researcher can thus design object detection frameworks that no longer require vast amounts of labelled data in order to train their models. This would open up other avenues of research and data collection, without the need for humans to manually label bounding boxes or classes ahead of time.

4.1 Frontiers

Other related works in the field include a tree based approach to tackling reinforcement learnign and object detection[31]. This allowed this particular reinforcement learning agent to consider multiple bounding boxes with different degrees of confidence simultaneously.

Another important field that this capstone did not touch upon was the task of image segmentation. Models such as Mask R-CNN [32] are able to more accurately label an object on a per pixel level. Whether a reinforcement learning approach that is unsupervised remains to be seen.

Bibliography

[1] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 6, pp. 1137–1149, 2017, ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[2] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 9905 LNCS, 2016, pp. 21–37, ISBN: 9783319464473. DOI: 10.1007/978-3-319-46448-0_2. arXiv: 1512.02325.

[3] T. Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common objects in context", in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 8693 LNCS, 2014, pp. 740–755, ISBN: 978-3-319-10601-4. DOI: 10.1007/978-3-319-10602-1_48. arXiv: 1405.0312.

[4] G. E. Hinton, N. Srivastava, and K. Swersky, "Lecture 6e- rmsprop: Divide the gradient by a running average of its recent magnitude", *COURSERA: Neural Networks for Machine Learning*, pp. 26–31, 2012. [Online]. Available: https://goo.gl/RsQeis.

[5] D. P. Kingma and J. L. Ba, "Adam: a Method for Stochastic Optimization", *International Conference on Learning Representations 2015*, pp. 1–15, 2015, ISSN: 09252312. DOI: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. arXiv: 1412.6980.

[6] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Alexnet", *Advances In Neural Information Processing Systems*, pp. 1–9, 2012, ISSN: 10495258. DOI: http://dx.doi.org/10.1016/j.protcy.2014.09.007. arXiv: 1102.0183.

[7] A. Karpathy and F. F. Li, *CS231n Convolutional Neural Networks for Visual Recognition*. [Online]. Available: https://cs231n.github.io/convolutional-networks/ (visited on 03/25/2018).

[8] F. Chollet, "Xception: Deep Learning with Separable Convolutions", *arXiv preprint arXiv:1610.02357*, pp. 1–14, 2016, ISSN: 1063-6919. DOI: 10.1109/CVPR.2017.195. arXiv: 1610.02357. [Online]. Available: https://arxiv.org/abs/1610.02357.

[9] D. Silver and D. Hassabis, *AlphaGo: Mastering the ancient game of Go with Machine Learning*, 2016. [Online]. Available: https://research.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html.

[10] V. M. Deepmind, "Human-level control through deep reinforcement learning", *IJCAI International Joint Conference on Artificial Intelligence*, vol. 2016-Janua, no. 7540, pp. 2315–2321, 2016, ISSN: 10450823. DOI: 10.1038/nature14236. arXiv: 1604.03986. [Online]. Available: http://www.nature.com/doifinder/10.1038/nature14236.

[11] J. Zhao, G. Qiu, Z. Guan, W. Zhao, and X. He, "Deep Reinforcement Learning for Sponsored Search Real-time Bidding", 2018. arXiv: 1803.00259. [Online]. Available: https://arxiv.org/abs/1803.00259.

[12] Q. Cai, A. Filos-Ratsikas, P. Tang, and Y. Zhang, "Reinforcement Mechanism Design for e-commerce", 2017. arXiv: 1708.07607. [Online]. Available: http://arxiv.org/abs/1708.07607.

[13] F.-F. Li, A. Karpathy, and J. Johnson, "Lecture 8 - 1 Feb 2016 Lecture 8: Spatial Localization and Detection", [Online]. Available: http://cs231n.stanford.edu/slides/2016/winter1516{_}lecture8.pdf.

[14] A. Borji, M. M. Cheng, H. Jiang, and J. Li, "Salient Object Detection : A Survey", *eprint arXiv*, pp. 1–26, 2014, ISSN: 1941-0042. DOI: 10.1109/TIP.2015.2487833. arXiv: arXiv:1411.5878v1.

[15] J. C. Caicedo and S. Lazebnik, "Active Object Localization with Deep Reinforcement Learning", [Online]. Available: https://www.cv-foundation.

org/openaccess/content{_}iccv{_}2015/papers/Caicedo{_}Active{_}Object{_}Localization{_}ICCV{_}2015{_}paper.pdf.

[16] Y. Rao, J. Lu, and J. Zhou, "Attention-aware Deep Reinforcement Learning for Video Face Recognition", [Online]. Available: http://openaccess.thecvf.com/content{_}ICCV{_}2017/papers/Rao{_}Attention-Aware{_}Deep{_}Reinforcement{_}ICCV{_}2017{_}paper.pdf.

[17] Alexirpan, *Deep Reinforcement Learning Doesn't Work Yet*. [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html (visited on 03/26/2018).

[18] K. Kansky, T. Silver, D. A. Mély, M. Eldawy, M. Lázaro-Gredilla, X. Lou, N. Dorfman, S. Sidor, S. Phoenix, and D. George, "Schema Networks: Zero-shot Transfer with a Generative Causal Model of Intuitive Physics", [Online]. Available: https://www.vicarious.com/wp-content/uploads/2017/10/icml2017-schemas.pdf.

[19] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning", *IEEE Signal Processing Magazine Special Issue on Deep Learning for Image Understanding*, pp. 1–14, 2017, ISSN: 1701.07274. DOI: 10.1109/MSP.2017.2743240. arXiv: 1708.05866. [Online]. Available: https://arxiv.org/pdf/1708.05866v1.pdf{\%}0Ahttp://arxiv.org/abs/1708.05866.

[20] H. V. Hasselt, A. C. Group, and C. Wiskunde, "Double Q-learning", *Nips*, pp. 1–9, 2010.

[21] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, and J. Veness, "Prioritized Experience Replay", *International Conference in Machine Learning*, vol. 4, no. 7540, p. 14, 2015, ISSN: 0028-0836. DOI: 10.1038/nature14236. arXiv: 1502.04623. [Online]. Available: http://arxiv.org/abs/1507.01526{\%}5Cnhttp://dx.doi.org/10.1007/978-3-642-27645-3{_}2{\%}5Cnhttp://arxiv.org/abs/1112.6209{\%}5Cnhttp://arxiv.org/abs/1509.06461{\%}5Cnhttp://www.arxiv.org/pdf/1509.06461.pdf{\%}5Cnhttp://arxiv.org/abs/1511.06295{\%}5Cnhttp://arxiv.org/abs/151.

[22] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge", *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, 2010, ISSN: 09205691. DOI: 10.1007/s11263-009-0275-4.

[23] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei, "ImageNet: A large-scale hierarchical image database", in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255, ISBN: 978-1-4244-3992-8. DOI: 10.1109/CVPRW.2009.5206848. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5206848.

[24] A. Kovashka, *University of Pittsburgh: Error HTTP 404 - File not found*. [Online]. Available: https://people.cs.pitt.edu/{~}kovashka/cs1699/hw4.html (visited on 03/25/2018).

[25] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "R-CNN", *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 580–587, 2014, ISSN: 10636919. DOI: 10.1109/CVPR.2014.81. arXiv: 1311.2524.

[26] R. Girshick, "Fast R-CNN", in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2015 Inter, 2015, pp. 1440–1448, ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.169. arXiv: 1504.08083.

[27] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "(2016 YOLO)YOU ONLY LOOK ONCE: UNIFIED, REAL-TIME OBJECT DETECTION", *Cvpr 2016*, pp. 779–788, 2016, ISSN: 10636919. DOI: 10.1016/j.nima.2015.05.028. arXiv: 1506.02640v1.

[28] M. Bellver, X. Giro-i Nieto, F. Marques, and J. Torres, "Hierarchical Object Detection with Deep Reinforcement Learning", *Nips*, no. Nips, 2016. arXiv: 1611.03718. [Online]. Available: http://arxiv.org/abs/1611.03718.

[29] J. Cavazza and V. Murino, "Active Regression with Adaptive Huber Loss", *arXiv:1606.01568 [cs]*, pp. 1–14, 2016. arXiv: 1606.01568. [Online]. Available: http://arxiv.org/abs/1606.01568.

[30] K. He, X. Zhang, S. Ren, and J. Sun, "ResNet", *arXiv preprint arXiv:1512.03385v1*, vol. 7, no. 3, pp. 171–180, 2015, ISSN: 1664-1078. DOI: 10.3389/fpsyg.2013.00124. arXiv: 1512.03385. [Online]. Available: http://arxiv.org/pdf/1512.03385v1.pdf.

[31] J. ZEQUN, "SCALE-ROBUST DEEP LEARNING FOR VISUAL RECOGNITION", [Online]. Available: http://scholarbank.nus.edu.sg/handle/10635/134430.

[32] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN", in *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, 2017, pp. 2980–2988, ISBN: 9781538610329. DOI: 10.1109/ICCV.2017.322. arXiv: 1703.06870.