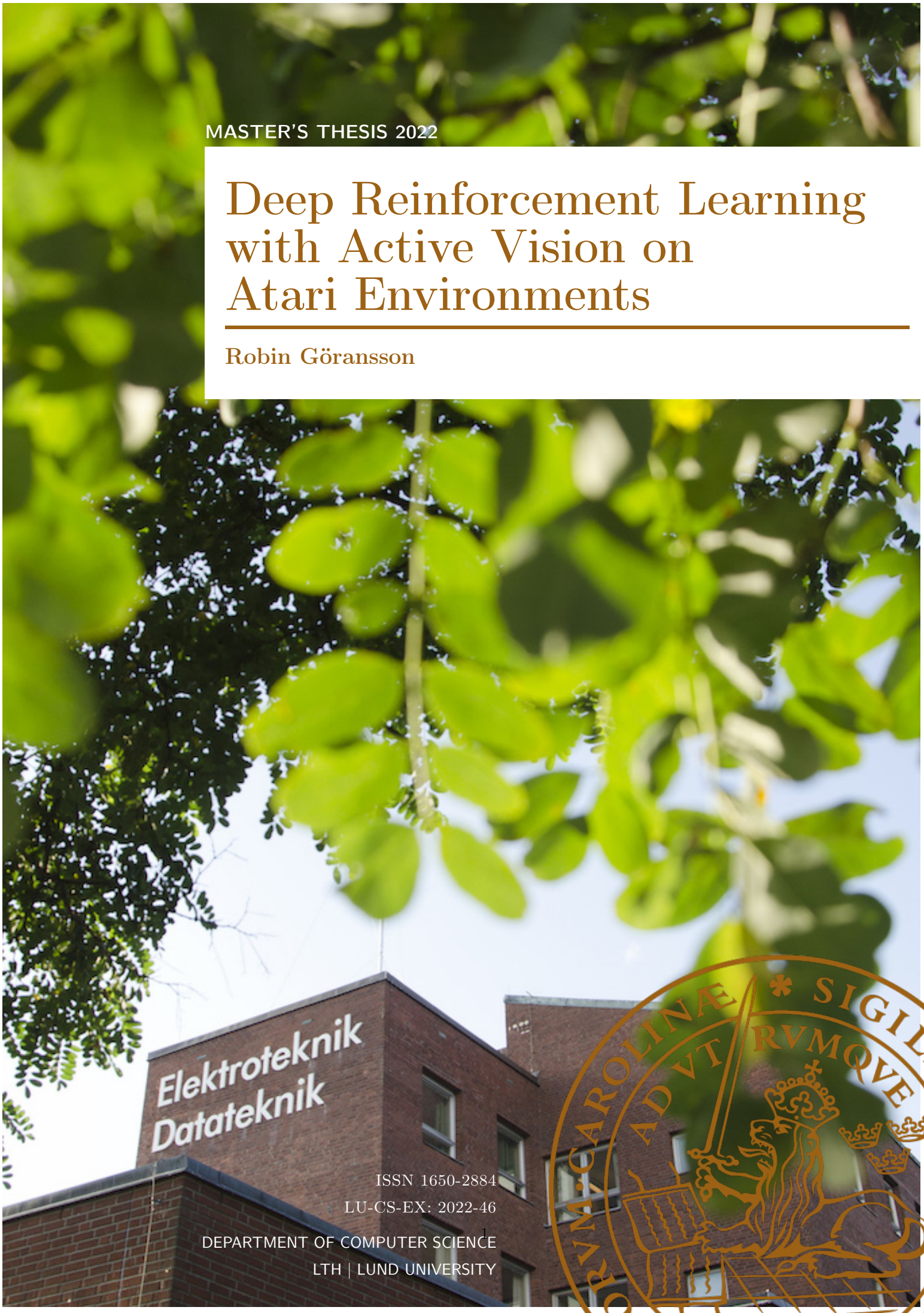


MASTER'S THESIS 2022

Deep Reinforcement Learning with Active Vision on Atari Environments

Robin Göransson

ISSN 1650-2884
LU-CS-EX: 2022-46

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-46

Deep Reinforcement Learning with Active Vision on Atari Environments

Djup Förstärkningsinlärning med Aktivt Seende på Atarimiljöer

Robin Göransson

Deep Reinforcement Learning with Active Vision on Atari Environments

(A Study on Resolution and Field of View)

Robin Göransson

ro0211go-s@student.lu.se

June 20, 2022

Master's thesis work carried out at
the Department of Computer Science, Lund University

Supervisor: Volker Krüger, volker.krueger@cs.lth.se

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

Reinforcement learning (RL) algorithms have throughout the years found most success on artificial domains where the state-space is fully observable. Atari 2600 games, where the full screen is used as the input state, are thus commonly used to evaluate the performance of an RL agent. Using the full screen as input could however be unnecessary as a significant amount of pixels on the screen do not contain any relevant information.

In this thesis a current RL algorithm is expanded using active vision. By restricting the visible portion of the screen and giving the agent means to control its focus-of-attention the state-space is made partially observable. With the addition of active vision an agent must simultaneously learn to play the game and learn to control its focus-of-attention. This more challenging task is solved using a modified version of the recurrent A3C-LSTM network which can handle active vision.

Throughout the thesis different models, that simulate the human visual system in different ways, were used. While all models used rectangular focal areas the models differed from each other in how resolution was handled. The first models used a single, constant resolution in the full focal area. By comparing these models it could be shown that a larger focal area using a lower resolution is preferable to a smaller focal area with higher resolution.

Next, the resolution was set to decrease with increased distance to the center of the focal area. The model using the larger focal area and a lower resolution was replaced with a model using the same large focal area but with higher resolution towards the center and lower towards the edges. The decreasing resolution did not improve performance.

As a final addition peripheral vision was added to the models. The peripheral vision was created using a very low resolution background outside the focal area. This addition did improve the performance of the models on Pong and Breakout while the addition barely affected the performance on Beam Rider. The model using decreasing resolution with the added peripheral was the best performing model on both Pong and Breakout while the most successful constant resolution model without peripheral achieved the best performance on Beam Rider.

Contents

1 Introduction

In reinforcement learning (RL) a fully observable environment is generally assumed. This means that reinforcement learning is most suitable on artificial domains such as games where the state-space is fully observable. A popular way to evaluate the performance of agents based on reinforcement learning has been through a suite of Atari 2600 games such as Breakout and Pong. In these games the agent uses the full screen for learning and thus does not have to deal with the problem of acting with incomplete information. However, using all the information available on the screen can be problematic as the state-space can get very large. Also, a significant amount of pixels on the screen do not contain any relevant information.

The goal for this thesis is to expand on the current RL algorithms using a *focus-of-attention* mechanism that would allow the agent to limit its field of view and focus on only the most important part of the screen. By omitting certain pixels the state-space becomes partially observable but, as we assume that the focus will be on relevant points, the information loss should be small. With only part of the screen visible for the agent it is being tasked with controlling its focus-of-attention and to effectively act with incomplete information. This approach is called *active vision* and will make the behavior of the agent more similar to human behavior.

In previous work [14] where the visual system of the agent was represented by a rectangle with constant resolution it was shown that this approach could be used to achieve near-optimal performance in Pong, Breakout and Space Invaders with only 35% of the screen visible. The model used in this approach was based on a Deep Q-Network (DQN) [9] with extensions such as the addition of an LSTM. The LSTM introduces recurrent layers in the network that allows the agent to store information about past inputs in a hidden state. This extended model shared many similarities with the model known as R2D2 [5]. Active vision was then introduced to the model forming the Myopic Deep Recurrent Q-Network (MyDRQN) [14]. This agent can remember past observations and can control both its movement and its focus-of-attention. An action that moves the player within the game will be referred to as a *natural action* while an action that moves the focus-of-attention will be referred to as a *visual action*. The introduction of active vision means that the network must find and optimize one policy for the player movement and one policy for controlling the focus-of-attention. At each time-step the screen is observed and the agent uses this information to simultaneously decide on the next natural action and the next visual action. An agent will need to learn both how to move and where to look in order to play the game successfully.

In this thesis the MyDRQN was replaced with a modified A3C-LSTM network that can handle the visual task. It was thus important to first make sure that good performance could be achieved using the rectangular *vision window* with constant resolution. The impact of lowering the resolution for a constant resolution vision window was then studied. By lowering the resolution it is possible to enlarge the agent's field of view while keeping the amount of *pixel-information* present in the input state small. Next, this vision window with homogeneous resolution was replaced with a vision window where the resolution decreases when the distance to the center of attention increases. This new vision window was created using three differently sized, concentric rectangles. The resolution in the innermost rectangle was kept high while the resolution in the other rectangles were lowered. Using the vision window with decreasing resolution the amount of pixel-information present in the modified game screen can be lowered by a considerable amount without shrinking the agent's field of view. As a last step the pixels outside the vision windows were replaced with values from a very low resolution version of the game screen. This new background can be considered to be the peripheral vision of the agent. The use of different resolutions can be motivated biologically as it mimics the human visual system to some extent.

2 Research Questions

- Does the introduction of active vision speed up training since it reduces the size of the state-space?
- Can the A3C-LSTM with focus-of-attention perform well on the different Atari games using the rectangular vision window with constant resolution?
- Is the size of the vision window more important than the resolution of the input state?
- How will the vision window with decreasing resolution impact the performance of the model?
- Will the addition of peripheral vision affect the performance of the models?
- How much will the agent actually move its focus-of-attention while playing the games?
- Can similarities between how the agent and a human plays the game and controls its focus-of-attention be found?

3 Reinforcement Learning

In Reinforcement Learning an agent that is able to observe the environment, takes actions that affects the environment and receives rewards from the environment, is defined. Based on the collected information the agent is tasked to learn a policy which maximizes some rewards received from the environment. An RL-agent has to learn through trial-and-error. The algorithm used in this project is based on an RL approach known as Actor-Critic. In order to explain this approach it is reasonable to first introduce Markov Decision Processes, as most RL tasks can be expressed as MDPs, and then continue with TD-Learning and the Policy Gradient.

3.1 Markov Process

A Markov Process is a stochastic process that describes a sequence of states S_1,S_2,... with the special property that the current state completely characterizes the process. This property is known as the Markov property and can formulated as:

$$\mathbb{P}[S_{t+1}|S_t] = \mathbb{P}[S_{t+1}|S_1, S_2, ..., S_t] \ . \tag{3.1}$$

A Markov Process can be expressed as a tuple $\langle \mathcal{S}, \mathcal{P} \rangle$ where $\mathcal{S}$ is a finite set of states and $\mathcal{P}$ is a state transition probability matrix with entries on the form $\mathcal{P}_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s]$ to describe the probability for the transition from state s to state s'.

3.2 Markov Reward Process

A Markov Reward Process (MRP) is a Markov Process where numerical rewards associated with the different transitions have been introduced. An MRP can be expressed as a tuple $\langle \mathcal{S}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is a finite set of states, $\mathcal{P}$ is a state transition probability matrix, $\mathcal{R}$ is a reward function and $\gamma \in [0, 1]$ is the discount factor. The reward function represents the next expected reward R_{t+1} given a current state and can be expressed as $\mathcal{R}_s = \mathbb{E}[R_{t+1}|S_t = s]$. The discount factor γ is used to discount future rewards in order to make immediate rewards more valuable than future rewards.

In general, the next reward is not very interesting by itself. A simple example would be an environment where non-zero rewards are only given out in terminal states. Thus, it is more interesting to study the total amount of (discounted) reward along a trajectory $\{S_t, R_{t+1}, S_{t+1}, R_{t+2}, ...\}$. This quantity is known as the *return* G_t and is given as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \ . \tag{3.2}$$

The inclusion of the discount factor γ prevents the return from diverging when no terminal states exist in the environment and the trajectory is of infinite length.

Using the definition of the return it is possible to formulate an expression for the value of a state s. The value of state s is defined as the expected return when starting in state s:

$$v(s) = \mathbb{E}[G_t|S_t = s] \ . \tag{3.3}$$

One should note that by this definition the value of a terminal state is always zero.

3.3 Markov Decision Process

A Markov Decision Process (MDP) is an MRP where the agent is able to make decisions and thus take actions. MDPs are used to formally describe environments in reinforcement learning and can be expressed as a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{P}$ is a state transition probability matrix, $\mathcal{R}$ is a reward function and $\gamma \in [0, 1]$ is the discount factor. As the agent is allowed to take actions the state transitions and the associated rewards now depend on both the current state and the chosen action. The entries in $\mathcal{P}$ and $\mathcal{R}$ can thus be expressed as:

$$\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s'|S_t = s, A_t = a] \ , \tag{3.4}$$

$$\mathcal{R}^a_s = \mathbb{E}[R_{t+1}|S_t = s, A_t = a] \ . \tag{3.5}$$

In an MDP the agent chooses actions and behaves in a certain way. This means that it is useful to describe this behavior with a *policy* π. A policy is a probability distribution over actions given states:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \ . \tag{3.6}$$

For a given policy π it is possible to define value functions for the MDP. The state-value function for an MDP is the expected return starting in state s and then following policy π:

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \ . \tag{3.7}$$

Further, the action-value function for an MDP is the expected return starting in state s, taking action a and the following policy π:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \ . \tag{3.8}$$

When using MDPs we want to find the best possible performance and thus "solve" the MDP. In other words we want to find the optimal value functions and the optimal policy. The optimal value functions are defined as:

$$v_*(s) = \max_\pi v_\pi(s) \ , \tag{3.9}$$

$$q_*(s, a) = \max_\pi q_\pi(s, a) \ . \tag{3.10}$$

and the optimal policy is defined as the policy π_* for which $\pi_* \geq \pi, \ \forall \pi$. Here it is useful to know that $\pi \geq \pi'$ if $v_\pi(s) \geq v_{\pi'}(s), \ \forall s$.

Further, it can be shown that there exists an optimal policy π_* for any MDP and that $v_{\pi_*}(s) = v_*(s)$ and $q_{\pi_*}(s,a) = q_*(s,a)$. It is also possible to generate the optimal policy if the optimal action-value function is known by maximizing over it:

$$\pi_*(a|s) = \begin{cases} 1 & \text{if } a = \arg\max_{a \in \mathcal{A}} q_*(s,a) \\ 0 & \text{otherwise} \end{cases} \quad . \tag{3.11}$$

To find the optimal value functions, and thus also the optimal policy, the *Bellman optimality equations* are typically used. These important equations can be written the following way:

$$v_*(s) = \max_a \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a v_*(s') \ , \tag{3.12}$$

$$q_*(s,a) = \mathcal{R}_s^a + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}_{ss'}^a \max_{a'} q_*(s',a') \ . \tag{3.13}$$

These equations are however non-linear and no closed-form solutions exist in general. This means that to solve these iterative methods are commonly used.

3.4 TD-Learning

The temporal-difference learning algorithm can be used to iteratively solve the Bellman optimality equation for either the state-value function, found in Equation 3.12, or the action-value function, found in Equation 3.13. This is thus an example of value-based reinforcement learning, in which a value function is computed in order to find the optimal policy. The simplest temporal-difference learning algorithm is known as TD(0) and can be described with the following update rule:

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \ . \tag{3.14}$$

In the equation above α is the learning rate, the estimated return $R_{t+1} + \gamma V(S_{t+1})$ is known as the *TD-target* and $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is known as the *TD-error*. The update rule thus moves the value $V(S_t)$ towards the the TD-target.

If TD(0) instead is applied on the action-value function the following update rule can be found:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t)) \ . \tag{3.15}$$

It should also be noted that TD(0) learns *on-policy*. This means that it learns about policy π from experience sampled from the same policy π.

3.5 Q-Learning

An alternative iterative value-based method is *Q-Learning*. This method works *off-policy* and solves the Bellman optimality equation for the action-value function, found in Equation 3.13. In an off-policy setting experience is sampled from behavior policy μ to learn about target policy π. In Q-Learning actions are chosen according to the behavior policy μ while the action-values are updated towards the value of an alternate action chosen by the target policy π. If the target policy is chosen to be greedy the update rule can be written as

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)) \tag{3.16}$$

where α is the learning rate, $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ is the target and $R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$ is the error.

3.6 Value Function Approximation

In environments where the state-space is too large it is not possible to learn the value for each state individually. Instead some function approximator is used to estimate the value functions for the different states. This function approximator can, for example, be a linear combination, a Gaussian mixture model or a neural network and is defined using parameters ω. When using value function approximations it is possible to generalize from seen states to unseen states. In the linear case the value functions are represented by a linear combination of features x:

$$v_\omega(s) = x(s)^{\mathrm{T}}\omega \ , \tag{3.17}$$

$$q_\omega(s,a) = x(s,a)^{\mathrm{T}}\omega \ . \tag{3.18}$$

In Equation 3.17 the approximated state-value function is expressed as $v_\omega(s)$ while the approximated action-value function in Equation 3.18 is expressed as $q_\omega(s,a)$. With a given function approximator the value function approximations can be expressed as:

$$v_\omega(s) \approx v_\pi(s) \ , \tag{3.19}$$

$$q_\omega(s,a) \approx q_\pi(s,a) \ . \tag{3.20}$$

A useful method to find good parameter choices is stochastic gradient descent. If a differentiable function $J(\omega)$ is defined as

$$J(\omega) = \mathbb{E}_\pi[(v_\pi(s) - v_\omega(s))^2] \tag{3.21}$$

or, using the action-value function instead,

$$J(\omega) = \mathbb{E}_\pi[(q_\pi(s,a) - q_\omega(s,a))^2] \tag{3.22}$$

a local minimum can be found by moving ω in the opposite direction of the gradient $\nabla_\omega J(\omega)$:

$$\Delta\omega = \alpha(v_\pi(s) - v_\omega(s))\nabla_\omega v_\omega(s) \tag{3.23}$$

or:

$$\Delta\omega = \alpha(q_\pi(s,a) - q_\omega(s,a))\nabla_\omega q_\omega(s,a) \tag{3.24}$$

where α is the step-size. Further, it should be noted that $v_\pi(s)$ is in practice replaced by some target; for example by $(R_{t+1} + \gamma v_\omega(s))$, if TD(0) is used.

3.7 Policy Gradient

In policy-based reinforcement learning the policy is directly parameterized using parameters θ according to:

$$\pi_\theta(a|s) = \mathbb{P}[A_t = a|S_t = s, \theta] \ . \tag{3.25}$$

Using a policy objective function $J(\theta)$ a policy gradient method can be used to search for a local maximum in $J(\theta)$. The parameters can then be updated by moving θ in the direction of the gradient:

$$\Delta\theta = \alpha\nabla_\theta J(\theta) \ . \tag{3.26}$$

To compute the policy gradient analytically the following identity is used:

$$\nabla_\theta \pi_\theta(a|s) = \pi_\theta(a|s)\frac{\nabla_\theta \pi_\theta(a|s)}{\pi_\theta(a|s)} = \pi_\theta(a|s)\nabla_\theta \log \pi_\theta(a|s) \tag{3.27}$$

where $\nabla_\theta \log \pi_\theta(s, a)$ is known as the score function. Using the identity above the policy gradient theorem can be derived:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\ Q_{\pi_\theta}(s, a)]\ . \tag{3.28}$$

A disadvantage with using the policy gradient is the high variance. One way to reduce this variance is to introduce the advantage function, which will be discussed in the section Advantage Function.

3.8 Actor-Critic

In actor-critic algorithms value-based reinforcement learning is combined with policy-based reinforcement learning. These algorithms maintain two sets of parameters; ω and θ. An example is an action-value based actor-critic. Here, the critic is used to estimate the action-value function $Q_\omega(s, a) \approx Q_{\pi_\theta}(s, a)$ by updating parameters ω and the actor updates the policy parameters θ in the direction suggested by the critic.

The parameters ω can, for example, be updated by the critic using the TD(0) algorithm. The TD(0) update rule found in Equation 3.15 is used to find the TD-target $R_{t+1} + \gamma Q_\omega(S_{t+1}, A_{t+1})$ and this target is then used to replace $q_\pi(s, a)$ in Equation 3.24 to get:

$$\Delta\omega = \alpha(R_{t+1} + \gamma Q_\omega(S_{t+1}, A_{t+1}) - Q_\omega(S_t, A_t))\nabla_\omega Q_\omega(S_t, A_t)\ . \tag{3.29}$$

The parameters ω found by the critic is used to estimate the action-value function as $Q_\omega(s, a) \approx Q_{\pi_\theta}(s, a)$ and is then inserted into the policy gradient theorem found in Equation 3.28:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s)\ Q_\omega(s, a)]\ . \tag{3.30}$$

With this result, the update rule for the actor can be written as:

$$\Delta\theta = \alpha\nabla_\theta \log \pi_\theta(a|s)\ Q_\omega(s, a)\ . \tag{3.31}$$

It should be noted that approximating the policy gradient can introduce bias. However, if the value function approximation in chosen carefully it can be shown that no bias will be introduced and that the exact policy gradient still can be followed.

3.9 Advantage Function

It is important to reduce the high variance of the policy gradient. This can be achieved by introducing the *advantage function*:

$$A_{\pi_\theta}(s, a) = Q_{\pi_\theta}(s, a) - V_{\pi_\theta}(s)\ . \tag{3.32}$$

While the action-value function estimates how good (or bad) it is to take a certain action from a certain state, the advantage function estimates how much better (or worse) it is to take a certain action than expected. As the advantage function can significantly reduce the variance the critic from the actor-critic algorithm should estimate $A_{\pi_\theta}(s, a)$ instead of $Q_{\pi_\theta}(s, a)$. This can be done by first approximating both $Q_\omega(s, a) \approx Q_{\pi_\theta}(s, a)$ and $V_\nu(s) \approx V_{\pi_\theta}(s)$ and then computing the estimated advantage function:

$$A(s,a) = Q_\omega(s,a) - V_\nu(s) \ . \tag{3.33}$$

Thus, the policy gradient can be rewritten as:

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(a|s) \ A(s,a)] \ . \tag{3.34}$$

3.10 Exploration vs. Exploitation

When training an RL-agent the trade-off between exploration and exploitation needs to be considered. The agent needs to both explore new states and trajectories and also exploit the previously gathered information to make the best choices and to follow the trajectories with the best rewards. If a policy is fully exploratory all decisions are made randomly and the agent might not reach the states with the best rewards. If a policy instead is fully greedy and always exploits the current information to take the action that is considered the best the agent will probably act sub-optimally as the best trajectories have not been found.

One way to deal with exploration and exploitation is to use an ϵ-greedy policy. This type of policy can be considered a combination between a fully greedy policy and a random policy. When using an ϵ-greedy policy the next action is chosen uniformly at random with probability ϵ and chosen greedily with probability $(1 - \epsilon)$. Thus, every state can be reached with a non-zero probability while the actions that are considered the best will be chosen with a higher probability.

A second way to deal with this is to let the action get chosen probabilistically from a multinomial distribution. If each possible action is given a value that estimates how good it is to take this action, a softmax function can be used to transform these values into probabilities. After the softmax function has been applied each action will be associated with a probability value between 0 and 1 and these values will sum up to 1.0. Then, an action is chosen by picking one from the probability distribution. An action that is considered good will be chosen with a higher probability but every action can be chosen with a non-zero probability. As more information is collected the probability for the best action will increase towards 1, while the probabilities for the other actions will decrease towards 0.

4 Deep Reinforcement Learning

In Deep Reinforcement Learning neural networks are used for value function approximation. While neural networks are powerful it can be problematic to combine a neural network with reinforcement learning. There are for examples issues with convergence and the fact that consecutively sampled experiences are highly correlated also causes problems. However, there are ways to deal with these problems and these solutions are presented below.

4.1 The DQN Algorithm

One of the first successful implementations of Deep Reinforcement Learning was the Deep Q-Network (DQN) [9]. This algorithm is based on Q-Learning and the policy used is ϵ-greedy. In order to deal with the problem with highly correlated samples a method known as *experience replay* is used. When using experience replay the N latest transitions $(S_t, A_t, R_{t+1}, S_{t+1})$ are stored in a replay memory $\mathcal{D}$. Then, after an action has been performed, a random mini-batch of transitions is sampled from $\mathcal{D}$ and used in the next optimization step. This means that consecutive parameter updates will not have these highly correlated samples [9]. A problem with using experience replay is that a lot of memory is required to keep a large replay memory. Thus, it is of interest to introduce another solution to the correlation problem; asynchronous algorithms.

4.2 The A3C Algorithm

The Asynchronous Advantage Actor-Critic (A3C) [8] algorithm is based on the Advantage Actor-Critic algorithm described in the section Advantage Function. The algorithm is asynchronous in the sense that many independent action-learners, with their own set of network parameters and local copy of the environment, sync up with the global network independently of each other. The A3C uses a convolutional neural networks with a softmax output for the policy $\pi_\theta(s)$ and a linear output for the value function $V_\nu(s)$, respectively. Note that even though the parameters θ for the policy and the parameters ν for the value function are shown as separate, some of the parameters are always shared in practice [8]. In this implementation the parameters for the convolutional torso are shared while the parameters for the dense output layers are separate. For simplicity $V_\nu(s) = V_\theta(s)$ is used in this project.

When using A3C the number of action-learners are decided and each action-learner is initialized with the parameters of the global network. Then, actions are taken according to the, now, local policy until a terminal state is reached or until t_{max} steps has been performed. The policy can be seen as a probability distribution over the actions and the action is thus chosen probabilistically. The total discounted reward R_{tot}, which is an estimate of $Q_{\pi_\theta}(s, a)$ [8] from Equation 3.32, is computed through bootstrapping and an estimate of the advantage function is computed as [8]:

$$A_\theta(s_t) = R_{tot} - V_\theta(s_t) = (\sum_{i=0}^{k-1} \gamma^i R_{t+1+i} + \gamma^k V_\theta(s_{t+k})) - V_\theta(s_t) \tag{4.1}$$

where k can vary but is upper bounded by t_{max} [8]. Using this advantage function both a value loss and a policy loss can be computed and combined with the entropy $H(\pi_\theta) = -\sum \pi_\theta \log \pi_\theta$ to compute the total loss:

$$L_{value} = \frac{1}{2} \sum_{i=0}^{k-1} A_\theta(s_{t+i})^2 \ , \tag{4.2}$$

$$L_{policy} = \sum_{i=0}^{k-1} (-\log \pi_\theta(a_{t+i}|s_{t+i}) \ A_\theta(s_{t+i}) - \beta H(\pi_\theta(s_{t+i}))) \ , \tag{4.3}$$

$$L_{total} = 0.5 \cdot L_{value} + L_{policy} \ . \tag{4.4}$$

In Equation 4.4 the 0.5 is used to make policy learning faster than value learning and β is a parameter that control the strength of the entropy term [8]. The entropy $H(\pi_\theta)$ is initially high when the choice of action is random but decreases towards 0 when the probability for a certain action increases towards 1. This way the entropy term lowers the loss the most when the prediction is uncertain making the agent more likely to explore and finding more diverse strategies. Further, note that $\pi_\theta(s_t)$ is a probability distribution while $\pi_\theta(a_t|s_t)$ is the conditional probability that action a_t is chosen in state s_t. Given the total loss the local gradient is computed with respect to θ and then, the local gradient is applied to the global network in an asynchronous fashion and the local parameters are reinitialized with the now updated global parameters. Pseudo-code for the A3C algorithm can be found in Algorithm 1. The pseudo-code is based on a figure in the original A3C paper [8].

The advantage with using asynchronous updates is that consecutively sampled experiences no longer are highly correlated as they are sampled from different independent action-learners. This is thus a replacement for experience replay but without the need to keep a large buffer of experiences in memory.

Algorithm 1 Pseudo-code for A3C.

```
Initialize global network with parameters Θ
Initialize global shared counter T
Choose amount of action-learners
for all action-learner threads do
    while T < T_max do
        Initialize local step-counter t = 0
        Synchronize local parameters θ = Θ
        Get state s_t
        while t < t_max and s_t is not terminal state do
            Perform action a_t according to probabilistic policy π_θ(s_t)
            Receive and save reward R_{t+1} and new state s_{t+1}
            t ← t + 1
            T ← T + 1
        end while
        Compute total discounted reward R_tot through bootstrapping
        Compute estimate of advantage function A_θ(s_t) = R_tot − V_θ(s_t)
        Compute total loss L_total = 0.5 · (1/2) Σ A_θ(s_t)^2 − Σ(log π_θ(a_t|s_t) A_θ(s_t) − βH(π_θ))
        Use loss to compute local gradient with respect to θ
        Update global Θ asynchronously using local gradient
    end while
end for
```

4.3 A3C with LSTM

In the Atari environments, presented in Sections 6.1-6.3, consecutive states are correlated and more than one state is needed to determine which way something is moving. In the original DQN paper [9] this is dealt with by representing each state with four consecutive frames. This quadruples the input size and thus increases the size of the state-space, which can slow down training. Another way to give the agent information about more than the current frame is to introduce some recurrent elements in the network. More specifically, a Long Short-Term Memory (LSTM) [4] is added to the network after the convolutional layers. The LSTM is fed with the output from the convolutional layers and the outputs from the LSTM from the previous time-step. This allows the agent to remember information from previous states and to thus make better decisions. The resulting model, the A3C-LSTM, uses smaller kernels and more filters in its convolutional part than suggested for the original DQN from 2013 [9] and the original A3C from 2016 [8]. The use of smaller kernels, usually of size 3×3 or 5×5, and more filters has become the standard in convolutional neural networks as these have been shown to be more cost-effective and to have better generalization properties [13]. Figure 1 is an overview of the model architecture.

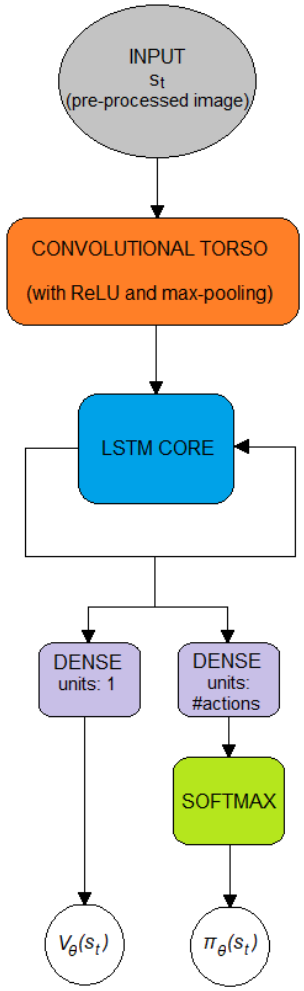


Figure 1: Overview of the architecture of the A3C algorithm with LSTM.

4.4 Other Extensions Included in A3C-LSTM

Below follows two important extensions that are included in the A3C-LSTM used in this project. First, the idea behind a shared optimizer is explained and the Shared Adam optimizer is introduced followed by an introduction of Generalized Advantage Estimation.

4.4.1 Shared Optimizer

The choice of optimizer can impact the performance and the behavior of a machine learning algorithm. Choosing a good optimizer is thus important. In the original A3C paper three different optimizers were used: SGD with momentum, RMSProp and Shared RMSProp. Of these three Shared RMSProp was the most robust optimizer, followed by the normal RMSProp optimizer [8]. The RMSProp optimizer is defined by the following updates

$$g_t \leftarrow \beta g_{t-1} + (1-\beta)\Delta\theta_{t-1}^2 \tag{4.5}$$

$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\Delta\theta_{t-1}}{\sqrt{g_t + \epsilon}} \tag{4.6}$$

$$\tag{4.7}$$

where g is the moving average of squared gradients, β is a moving average parameter, α is the learning rate and ϵ is a small constant for numerical stability. While θ are shared across threads in both the normal RMSProp and Shared RMSProp, the difference between the optimizers is whether or not the moving average of squared gradients g are also shared across threads. In Shared RMSProp, where g is shared across threads, g is updated asynchronously [8].

Adam is another popular optimization algorithm that can be used instead of RMSProp. The Adam optimizer can be defined with the following updates

$$g_t \leftarrow \nabla_\theta f_t(\theta_{t-1}) \tag{4.8}$$
$$m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t \tag{4.9}$$
$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{4.10}$$
$$\hat{m}_t \leftarrow \frac{m_t}{1-\beta_1^t} \tag{4.11}$$
$$\hat{v}_t \leftarrow \frac{v_t}{1-\beta_2^t} \tag{4.12}$$
$$\theta_t \leftarrow \theta_{t-1} - \alpha \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{4.13}$$

where f is the objective function, g is the gradients of the objective function, m is the biased first moment estimate ($\hat{m}$ is bias-corrected version), v is the biased second moment estimate ($\hat{v}$ is bias-corrected version), β_1 and β_2 are decay rates, α is the learning rate and ϵ is a small constant for numerical stability [6]. In the extended version of A3C a shared version of this optimization algorithm, called Shared Adam, is used. For Shared Adam the moment estimates m_t and v_t as well as the parameters θ are shared across threads and updated in an asynchronous fashion.

While Adam usually is a good choice, it has been shown that there exist convergence issues when using the optimization algorithm. A way to deal with these issues is AMSGrad, which is a modified version of the Adam algorithm [11]. The changes affect the second moment estimates v and can be expressed in the following way:

$$v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \tag{4.14}$$
$$\tilde{v}_t \leftarrow \max(\tilde{v}_{t-1}, v_t) \tag{4.15}$$
$$\hat{v}_t \leftarrow \frac{\tilde{v}_t}{1-\beta_2^t} \tag{4.16}$$

In other words; AMSGrad used the maximum of the second moment estimate instead of using the latest value. This algorithm has been shown to have better convergence properties than the normal Adam algorithm [11].

4.4.2 Generalized Advantage Estimation

In the original A3C algorithm a k-step TD-error was used to estimate the advantage function $A_\theta(s,a)$. The advantage function could also have been estimated using, for example, a 1-step or a 2-step TD-error. Generalized Advantage Estimation, or GAE, combines these different TD-errors to make a more robust estimation of the advantage function [12]. Below, the GAE is derived, starting with the definition of the so called TD(0)-error:

$$\delta_t^V = R_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \tag{4.17}$$

Next, TD(0)-errors for different time-steps ($\delta_t^V, \delta_{t+1}^V, \dots$) are combined to form estimators $\hat{A}_t^{(k)}$ [12]:

$$\hat{A}_t^{(1)} := \delta_t^V = -V_\theta(s_t) + R_{t+1} + \gamma V_\theta(s_{t+1}) \tag{4.18}$$

$$\hat{A}_t^{(2)} := \delta_t^V + \gamma\delta_{t+1}^V = -V_\theta(s_t) + R_{t+1} + \gamma R_{t+2} + \gamma^2 V_\theta(s_{t+2}) \tag{4.19}$$

$$\hat{A}_t^{(3)} := \delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V = -V_\theta(s_t) + R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 V_\theta(s_{t+3}) \tag{4.20}$$

$$\hat{A}_t^{(k)} := \sum_{l=0}^{k-1} \gamma^l \delta_{t+l}^V = -V_\theta(s_t) + R_{t+1} + \gamma R_{t+2} + ... + \gamma^{k-1} R_{t+k} + \gamma^k V_\theta(s_{t+k}) \tag{4.21}$$

It can be noted that the estimator $\hat{A}_t^{(k)}$ is equivalent to the advantage estimation, $A_\theta(s, a)$, used by the original A3C algorithm, as seen in Equation 4.1. The generalized advantage estimator $\hat{A}_t^{GAE(\gamma,\lambda)}$ can now be defined as the exponentially-weighted average of the $\hat{A}_t^{(k)}$ estimators

$$\begin{aligned}
\hat{A}_t^{GAE(\gamma,\lambda)} &:= (1-\lambda)(\hat{A}_t^{(1)} + \lambda\hat{A}_t^{(2)} + \lambda^2\hat{A}_t^{(3)} + ...) \\
&= (1-\lambda)(\delta_t^V + \lambda(\delta_t^V + \gamma\delta_{t+1}^V) + \lambda^2(\delta_t^V + \gamma\delta_{t+1}^V + \gamma^2\delta_{t+2}^V) + ...) \\
&= (1-\lambda)(\delta_t^V(1 + \lambda + \lambda^2 + ...) + \gamma\delta_{t+1}^V(\lambda + \lambda^2 + \lambda^3 + ...) + ...) \\
&= (1-\lambda)(\delta_t^V(\frac{1}{1-\lambda}) + \gamma\delta_{t+1}^V(\frac{\lambda}{1-\lambda}) + \gamma^2\delta_{t+2}^V(\frac{\lambda^2}{1-\lambda}) + ...) \\
&= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V
\end{aligned} \tag{4.22}$$

where λ is a parameter $0 \leq \lambda \leq 1$ [12]. There are two special cases of the generalized advantage estimator that should be addressed: when $\lambda = 0$ and when $\lambda = 1$. For these special cases the following holds:

$$\hat{A}_t^{GAE(\gamma,0)} = \delta_t^V = R_{t+1} + \gamma V_\theta(s_{t+1}) - V_\theta(s_t) \tag{4.23}$$

$$\hat{A}_t^{GAE(\gamma,1)} = \sum_{l=0}^{\infty} \gamma^l \delta_{t+l}^V = \sum_{l=0}^{\infty} \gamma^l R_{t+l+1} - V_\theta(s_t) \tag{4.24}$$

For the case when $\lambda = 1$ the variance is usually high and when $\lambda = 0$ the variance is lower but bias is usually introduced. The choice of the parameter $0 < \lambda < 1$ is thus a trade-off between bias and variance [12].

5 Focus-of-Attention

A focus-of-attention (FoA) mechanism allows an agent, or a human, to focus on a certain part of an image or a text while giving less attention to other parts [3]. For a human the focus-of-attention mechanism is important as the brain cannot process all the visual input data at once. By adding focus-of-attention to a reinforcement learning agent the input data should shrink in size and possibly speed up training.

When training an agent on the Atari environments full pre-processed screens are used as inputs to the network. The pre-processing of the environments will be described in the section Pre-processing and Modifications. With inputs of size 80×80 the state-space becomes very large. In fact, the state-space is unnecessarily large as many input pixels aren't important. A way to deal with this is to introduce focus-of-attention to these Atari games. By adding focus-of-attention the screens are modified in a way that focuses the view of the agent on the relevant part of the screen. *Visual actions* are also introduced that move the center of attention each time-step. With this set-up the agent needs to learn how to play the game and at the same time learn where to put its attention.

The introduction of focus-of-attention makes sure that pixels with no or little importance will, with enough training, be disregarded by the model. This will effectively decrease the size of the state-space, which could speed up training. The introduction of the visual actions, i.e., to move the center of attention up, down, left or right or to do nothing, can however slow down the training due to the added complexity to the task. Further, the introduction of focus-of-attention makes the true game state only partially observable, which to some degree violates the Markov property. However, by moving its focus-of-attention the agent does technically have access to the full true game state. This dynamic is similar to how the human visual system works where the human eyes are moving around and locating interesting parts in order to build up a mental map of a scene.

This model with focus-of-attention was created by duplicating the head in Figure 1 in order to create one *natural head*, which controls the actual player movements, and one *vision head*, which controls the visual movements. This results in a model with four outputs instead of the two in the model without the focus-of-attention mechanism. The natural head outputs its own value function $V^{nat}_{\nu^{nat}}(s_t)$ and its own softmax output for the policy $\pi^{nat}_{\theta^{nat}}(s_t)$. Further, the vision head similarly outputs $V^{vis}_{\nu^{vis}}(s_t)$ and $\pi^{vis}_{\theta^{vis}}(s_t)$. Note that the four sets of parameters θ^{nat}, θ^{vis}, ν^{nat} and ν^{vis} share the parameters for the convolutional torso and the LSTM. This means that most parameters are shared and because of this the expressions are simplified as $\theta = \theta^{nat} = \theta^{vis} = \nu^{nat} = \nu^{vis}$ in this project. An overview of the model can be seen in Figure 2 and a more detailed illustration of the convolutional torso can be seen in Appendix A.1.

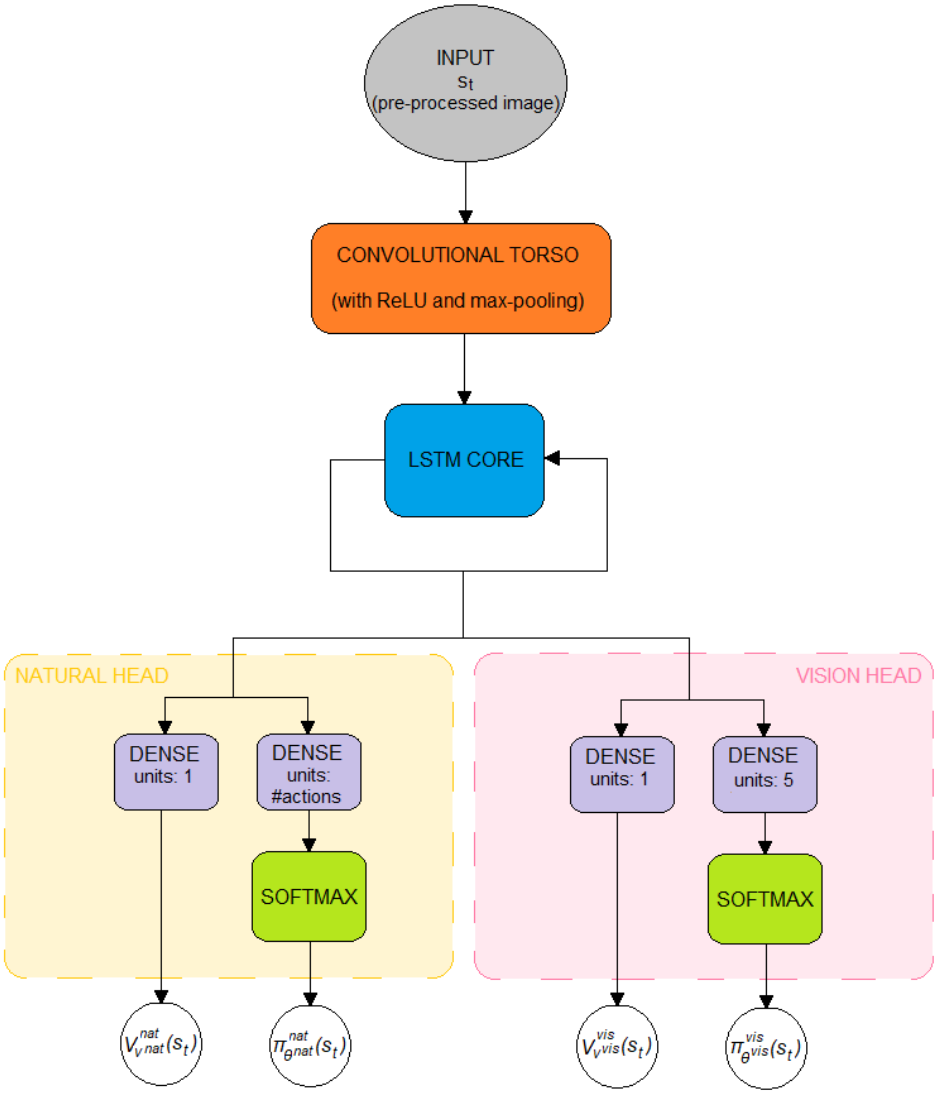


Figure 2: Overview of the architecture of the A3C-LSTM algorithm with focus-of-attention.

For this model a transition from state S_t to S_{t+1} is made through a natural action A_t^{nat} and a simultaneous visual action A_t^{vis}. The reward R_{t+1} from this transition depends only on the natural action A_t^{nat}, however. The visual action A_t^{vis} doesn't directly affect the state of the game, but instead affects what part of the information is accessible to the agent. This means that while R_{t+1} is not affected by A_t^{vis}, the state S_{t+1} is affected and thus also the following natural action A_{t+1}^{nat} and the next reward R_{t+2}. In order to account for this behavior the A3C loss function derived in Equations 4.1-4.4 is split in two parts. The first part is the total loss of the natural head and the second part is the total loss of the vision head. In the loss function for the natural head the reward R_{t+1} can be used as this reward is affected by the natural action A_t^{nat}. However, in the loss function for the visual head the reward R_{t+1} has to be replaced with R_{t+2} as this is the reward that is affected by the visual action A_t^{vis}. Both R_{t+1} and R_{t+2} are thus needed and a transition is represented by $(S_t, A_t^{nat}, A_t^{vis}, R_{t+1}, R_{t+2}, S_{t+1})$. A transition is illustrated in Figure 3.

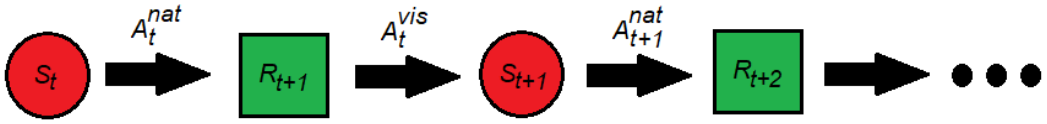


Figure 3: A transition of the A3C-LSTM algorithm with focus-of-attention.

5.1 Modifications of Loss Function

For this new model the total loss L_{total} can be divided into two parts: the total loss of the natural head L_{total}^{nat} and the total loss of the vision head L_{total}^{vis}. For the natural head the loss is defined in a similar way as the loss for the original A3C as shown in Equations 4.1-4.4 with the difference that generalized advantage estimation is used in the policy loss to make the advantage estimation more robust. This means that the value loss of the natural head can be defined as:

$$L_{value}^{nat} = \frac{1}{2}\sum_{i=0}^{k-1} A_\theta^{nat}(s_{t+i})^2 \ , \tag{5.1}$$

$$A_\theta^{nat}(s_t) = (\sum_{i=0}^{k-1}(\gamma^{nat})^i R_{t+1+i} + (\gamma^{nat})^k V_\theta^{nat}(s_{t+k})) - V_\theta^{nat}(s_t) \tag{5.2}$$

where $A_\theta^{nat}(s_t)$ is the simple (not generalized) estimate of the advantage function for the natural head. Next, the policy loss of the natural head with generalized advantage estimation can be defined as:

$$L_{policy}^{nat} = \sum_{i=0}^{k-1}(-\log \pi_\theta^{nat}(a_{t+i}|s_{t+i})\ \hat{A}_{t+i}^{GAE,nat} - \beta^{nat} H(\pi_\theta^{nat}(s_{t+i}))) \ , \tag{5.3}$$

$$\hat{A}_t^{GAE,nat} = \sum_{i=0}^{k-1}(\gamma^{nat}\lambda^{nat})^i \delta_{t+i}^{nat} \ , \tag{5.4}$$

$$\delta_t^{nat} = R_{t+1} + \gamma^{nat} V_\theta^{nat}(s_{t+1}) - V_\theta^{nat}(s_t) \tag{5.5}$$

where $\hat{A}_t^{GAE,nat}$ is the generalized advantage estimation for the natural head and δ_t^{nat} is the TD(0)-error for the natural head. Then, the value loss and the policy loss are combined as before in order to form the total loss for the natural head:

$$L_{total}^{nat} = 0.5 \cdot L_{value}^{nat} + L_{policy}^{nat} \ . \tag{5.6}$$

For the vision head R_{t+1} needs to be replaced with R_{t+2} as discussed above and visualized in Figure 3. With this small modification the value loss for the vision head can be defined as:

$$L_{value}^{vis} = \frac{1}{2}\sum_{i=0}^{k-2} A_\theta^{vis}(s_{t+i})^2 \ , \tag{5.7}$$

$$A_\theta^{vis}(s_t) = (\sum_{i=0}^{k-2}(\gamma^{vis})^i R_{t+2+i} + (\gamma^{vis})^k V_\theta^{vis}(s_{t+k})) - V_\theta^{vis}(s_t) \tag{5.8}$$

where $A_\theta^{vis}(s_t)$ is the simple estimate of the advantage function for the vision head. Further, the policy loss can in this case be defined as:

$$L_{policy}^{vis} = \sum_{i=0}^{k-2}(-\log \pi_\theta^{vis}(a_{t+i}|s_{t+i})\ \hat{A}_{t+i}^{GAE,vis} - \beta^{vis} H(\pi_\theta^{vis}(s_{t+i})))\ , \tag{5.9}$$

$$\hat{A}_t^{GAE,vis} = \sum_{i=0}^{k-2}(\gamma^{vis}\lambda^{vis})^i \delta_{t+i}^{vis}\ , \tag{5.10}$$

$$\delta_t^{vis} = R_{t+2} + \gamma^{vis} V_\theta^{vis}(s_{t+1}) - V_\theta^{vis}(s_t) \tag{5.11}$$

where $\hat{A}_t^{GAE,vis}$ is the generalized advantage estimation for the vision head and δ_t^{vis} is the TD(0)-error for the vision head. The losses is then combined in the same way as for the natural head:

$$L_{total}^{vis} = 0.5 \cdot L_{value}^{vis} + L_{policy}^{vis}\ . \tag{5.12}$$

Finally, the total loss of the natural head can be combined with the total loss of the vision head to find the total loss:

$$L_{total} = L_{total}^{nat} + L_{total}^{vis}\ . \tag{5.13}$$

Note that in the derivation above there exists one set of hyperparameters γ, λ and β for the natural head and one set for the vision head. This makes it possible to make one of the heads more short-sighted than the other or to introduce more variance in one of the heads and less in the other one. Throughout this project $\gamma^{nat} = \gamma^{vis}$, $\lambda^{nat} = \lambda^{vis}$ and $\beta^{nat} = \beta^{vis}$ are used for simplicity, however.

Pseudo-code for the A3C-LSTM algorithm with focus-of-attention can be found in Algorithm 2.

Algorithm 2 Pseudo-code for A3C-LSTM with focus-of-attention.

```
Initialize global network with parameters Θ
Initialize global shared counter T
Choose amount of action-learners
for all action-learner threads do
    while T < T_max do
        Initialize local step-counter t = 0
        Synchronize local parameters θ = Θ
        Get state s_t
        while t < t_max and s_t is not terminal state do
            Perform natural action a_t^nat according to probabilistic policy π_θ^nat(s_t)
            Perform visual action a_t^vis according to probabilistic policy π_θ^vis(s_t)
            Receive and save reward R_{t+1} and new state s_{t+1}
            t ← t + 1
            T ← T + 1
        end while
        Compute total discounted rewards R_tot^nat and R_tot^vis through bootstrapping
        Form estimate of advantage function for natural head A_θ^nat(s_t) = R_tot^nat − V_θ^nat(s_t)
        Form estimate of advantage function for vision head A_θ^vis(s_t) = R_tot^vis − V_θ^vis(s_t)
        Compute generalized advantage estimations Â_t^GAE,nat(s_t) and Â_t^GAE,vis(s_t)
        Form loss L_total^nat = 0.5 · ½ Σ A_θ^nat(s_t)^2 − Σ log π_θ^nat(a_t^nat|s_t) Â_t^GAE,nat(s_t) − βH(π_θ^nat)
        Form loss L_total^vis = 0.5 · ½ Σ A_θ^vis(s_t)^2 − Σ log π_θ^vis(a_t^vis|s_t) Â_t^GAE,vis(s_t) − βH(π_θ^vis)
        Form total loss by summing L_total^nat and L_total^vis
        Use loss to compute local gradient with respect to θ
        Update global Θ asynchronously using local gradient
    end while
end for
```

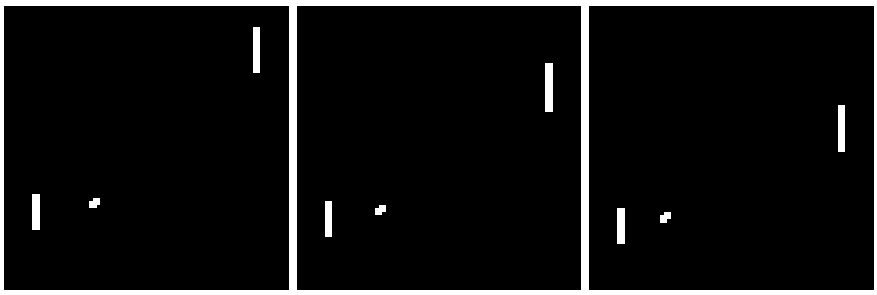
5.2 Pre-processing and Modifications

Before focus-of-attention, or a so called *vision window*, can be applied the game screen must be pre-processed and slightly modified. Each frame for the Atari games used in this project is an RGB image (values between 0 and 255) with a width of 160 pixels and a height of 210 pixels. These images are first cropped using game-specific information. This step removes pixels of no importance and leaves only the pixels representing the actual game area. Next, the size of the image is decreased to 80×80 pixels, converted to gray-scale and normalized to pixel values between 0 and 1. As a last step the state is normalized using an unbiased mean, μ, and an unbiased standard deviation, σ, of the pixel values before it is used as the input state to the convolutional network. The normalization is done as follows:

$$s_t \leftarrow \frac{s_t - \mu}{\sigma} \ . \tag{5.14}$$

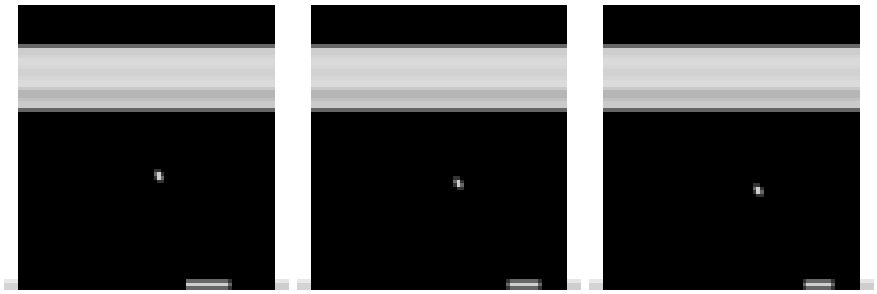
After this normalization the dark pixels will attain negative values while the light pixels will get positive pixel values.

It is also common to repeat each action for a few frames in order to reduce the size of the state-space. In this project each action is repeated for four frames. Further, each state is represented by a max-pooling operation of the three previous frames in order to represent some sense of direction. If the frames are represented by x_n, $n = 1, 2, ...$, and the states by $s_t = s_{n/4}$, $n = 4, 8, ...$, each state can be written as $s_t = s_{n/4} = \text{maxpool}(x_{n/4},\ x_{n/4-1},\ x_{n/4-2})$. Note that frame-stacking, as suggested in the original DQN paper, is not used to detect velocities and accelerations. Figures 4, 5 and 6 show examples of pre-processed states from the Atari games Pong, Breakout and Beam Rider.



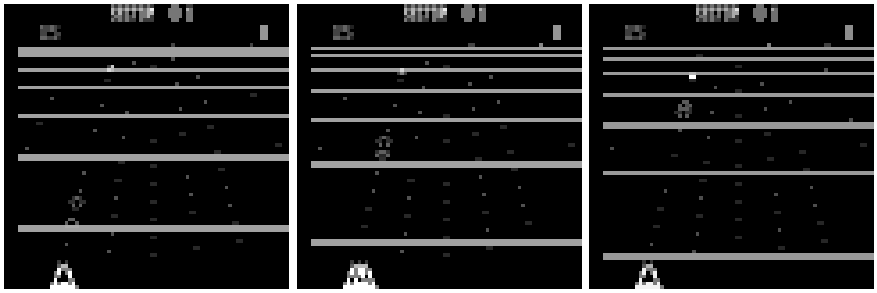
(a) State s_t. (b) State s_{t+1}. (c) State s_{t+2}.

Figure 4: Pre-processed consecutive states for the game Pong starting from some state s_t.



(a) State s_t. (b) State s_{t+1}. (c) State s_{t+2}.

Figure 5: Pre-processed consecutive states for the game Breakout starting from some state s_t.



(a) State s_t. (b) State s_{t+1}. (c) State s_{t+2}.

Figure 6: Pre-processed consecutive states for the game Beam Rider starting from some state s_t.

It is also standard to clip all rewards to either -1, 0 or $+1$. While this is done to generalize across different games it also makes the agent unable to differentiate between a small positive reward and a large positive reward.

Other modifications to the environment are to automatically use the reset action when a life is lost and to count a loss of a life as a terminal state. The first of the two modifications allows the agent to actually learn how to play the game without having to learn how to restart, while the second modification helps with value estimation.

A final modification is to take a random number of no-operation actions at the beginning of each episode. This is done in order to introduce some randomness into the otherwise deterministic Atari games. Without this added randomness the agent might not explore enough states to learn a good behavior.

5.3 Constant Resolution Vision Window

A straightforward way to modify the screen in order simulate the focus-of-attention of the agent is to use a rectangular area as the focal area. The focal area, which holds all the pixels that are visible to the agent, can be defined using a focal point (f_x, f_y), a focal width and a focal height. The focal point marks the center of attention and the focal width and focal height controls the size of the rectangular focal area. The resolution of focal area also needs to be decided. Different resolutions of the same input state can be seen in Figure 8. This type of vision window is referred to as a *Constant* resolution vision window and an overview of the window can be seen in Figure 7.

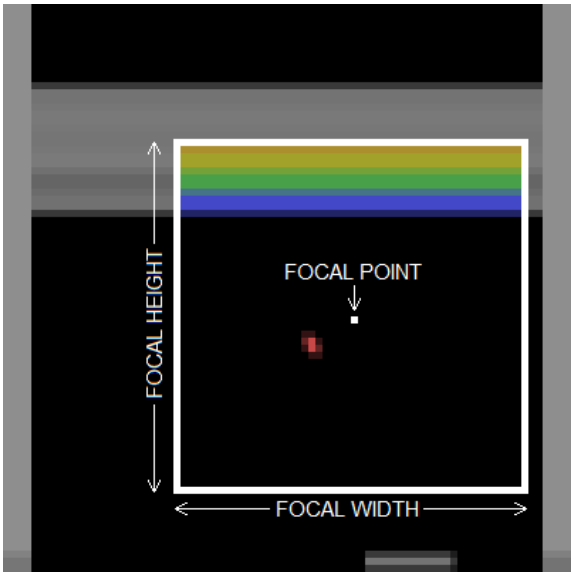
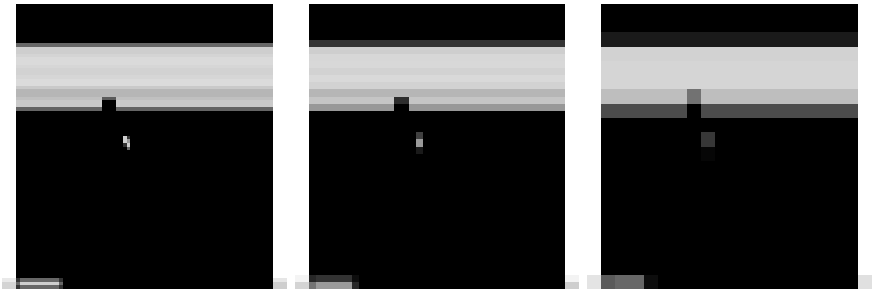


Figure 7: Overview of the *Constant* vision window.



(a) 80×80 resolution. (b) 40×40 resolution. (c) 20×20 resolution.

Figure 8: Input states in different resolutions.

When introducing focus-of-attention to the model a focal step-size also needs to be chosen. The focal step-size controls how many pixels the focal point will move when a visual action is taken. With the focal step-size set to 4 a visual action will move the focal point four pixels in the appropriate direction. Further, two other restrictions to the movement of the focal point are introduced. First, the focal point can not be moved off the screen. In a case where a visual action would move the focal point off-screen the focal point will not be moved. This means that the focal point can only reach a specific selection of locations and that these locations depend on the initialization of the focal point. In other words, the initialization affects which states that are reachable. This leads to the second restriction; the focal point can only be initialized in certain locations based on the focal step-size. With this restriction the same states can always be reached independently of the initialization. A visualization of the possible locations for the focal point can be seen in Figure 9. Note that the possible locations for the focal point, marked by green squares in the figure, are evenly spread out across the screen.

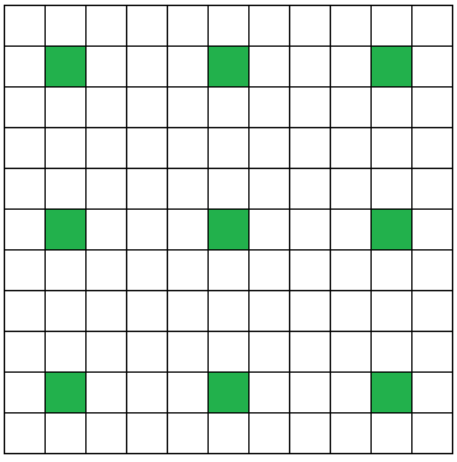


Figure 9: Possible locations for focal point with a focal step-size of 4.

The *Constant* vision window described above is applied on the original input image and simply replaces all pixel values outside of the focal area with zeros while keeping all pixels within the focal area unchanged. Note that this operation is applied after the normalization of the pixel values described in the section Pre-processing and Modifications, which means that a pixel value of 0 does not represent the color black. The normalization is done before applying the vision window to avoid having the focal point influence the normalization and to make sure that the pixels outside the vision window are kept constant zero. By replacing all pixels outside the focal area with zeros the input states for the focus-of-attention model will have the same size as the input states for the original model: 80×80 pixels. A few examples of input states can be seen in Figure 10 along with corresponding visualizations which makes it easier to follow what is actually happening when the agent is playing the game.

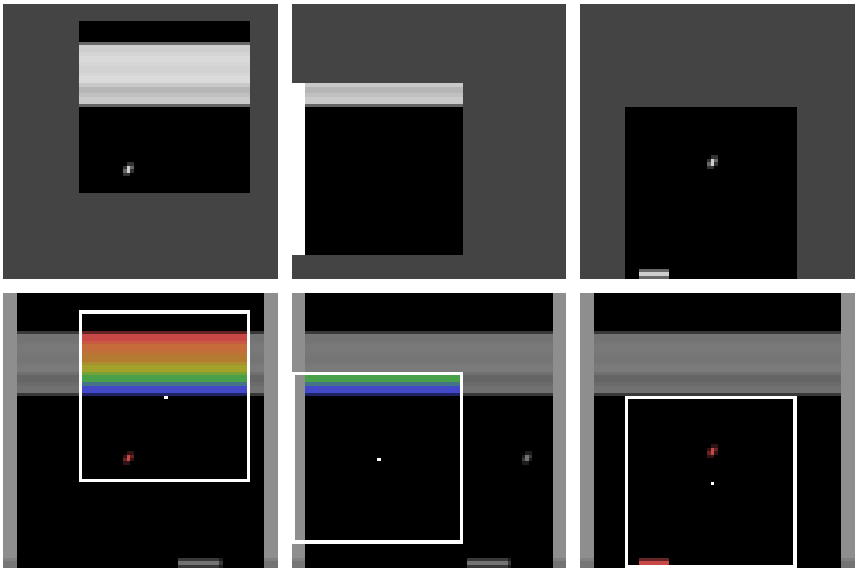


Figure 10: Input states in the top row and corresponding visualizations in the bottom row.

The main reason that the values of the pixels outside of the focal area is replaced with zeros as opposed to being cut out is that information about where the agent is looking is available to the model. If the pixels are cut out the input state will shrink in size but the agent will have to learn that the input is only a subspace and then learn to act accordingly.

5.4 Decreasing Resolution Vision Window

While the *Constant* vision window is a straight-forward way to simulate focus-of-attention it would be interesting to introduce a new type of vision window that better mimics the human visual system. The focal area of this new vision window is created using a focal point (f_x, f_y), three different resolutions and three rectangular areas with the focal point as their centers. The smallest rectangular area uses the best resolution while the largest rectangular area uses the worst available resolution. This way the resolution decreases when the distance to the focal point increases. The pixels outside the focal area are replaced with zeros in the same way as they were for the *Constant* vision window. The restrictions on the movement of the focal point described in the previous section is also applied on this vision window (see Figure 9). This type of vision window is referred to as a *Decreasing* resolution vision window. An overview of the window can be seen in Figure 11 and a comparison between three different resolutions can be found in Figure 8.

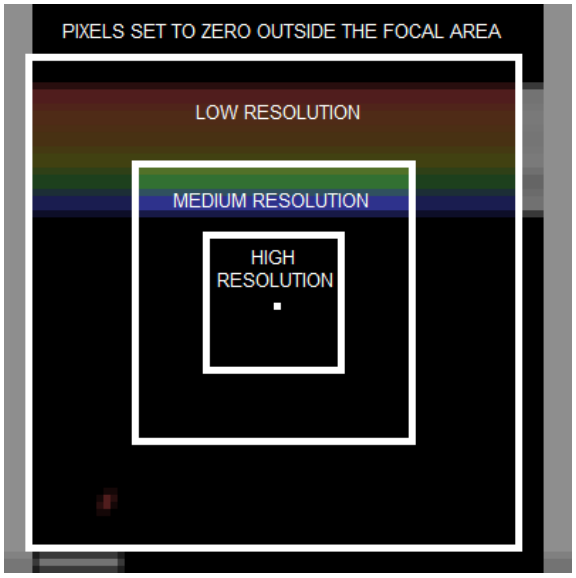


Figure 11: Overview of the *Decreasing* vision window.

When applying the *Decreasing* vision window an interesting metric is the amount of information, expressed in pixels, present in the input state. In the original 80×80 input state there is 6400 pixels worth of information present while only 400 pixels worth of information is present in the low resolution input state shown in Figure 8c. The amount of *pixel-information*, I_D, present after applying the *Decreasing* vision window, and thus combining the three different resolutions, can be computed the following way:

$$I_D = \frac{(f_w^{(1)} \cdot f_h^{(1)})}{(\frac{80^2}{R_v^{(1)} \cdot R_h^{(1)}})} + \frac{(f_w^{(2)} \cdot f_h^{(2)}) - (f_w^{(1)} \cdot f_h^{(1)})}{(\frac{80^2}{R_v^{(2)} \cdot R_h^{(2)}})} + \frac{(f_w^{(3)} \cdot f_h^{(3)}) - (f_w^{(2)} \cdot f_h^{(2)})}{(\frac{80^2}{R_v^{(3)} \cdot R_h^{(3)}})} \tag{5.15}$$

where $f_w^{(1)}$ and $f_h^{(1)}$ are the width and height of the innermost rectangle making up the focal area, $f_w^{(2)}$ and $f_h^{(2)}$ are the width and height of the middle rectangle, $f_w^{(3)}$ and $f_h^{(3)}$ are the width and height of the outermost rectangle, $R_v^{(1)}$, $R_v^{(2)}$ and $R_v^{(3)}$ are the three different vertical resolutions and $R_h^{(1)}$, $R_h^{(2)}$ and $R_h^{(3)}$ are the three different horizontal resolutions. Note that if the resolution i is 40×40 then $R_v^{(i)} = R_h^{(i)} = 40$. The formula computes the three different areas and divides them with certain factors to account for the lower resolutions. Note that Equation 5.15 computes the maximum amount of pixel-information present in the input state. If the focal point is moved to a corner of the screen only a fourth of the maximum amount of pixel-information is actually present in the input state.

In order to make comparisons between the *Decreasing* and the *Constant* vision windows the amount of pixel-information present after applying the *Constant* vision window can also be computed. This is more straight-forward as shown below:

$$I_C = \frac{(f_w \cdot f_h)}{(\frac{80^2}{R_v \cdot R_h})} \ . \tag{5.16}$$

Above f_w is the focal width, f_h is the focal height, R_v is the vertical resolution and R_h is the horizontal resolution. Again, this is actually an expression for the maximum amount of pixel-information present in the input state.

A few examples of input states with the *Decreasing* vision window applied can be seen in Figure 12 along with corresponding visualizations which makes it easier to follow what is actually happening when the agent is playing the game.

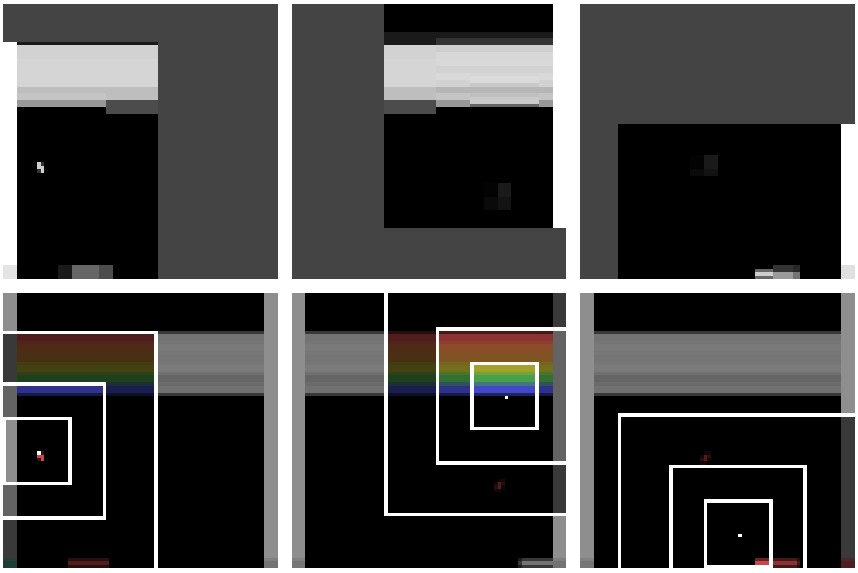


Figure 12: Input states in the top row and corresponding visualizations in the bottom row.

5.5 Peripheral Vision

To further mimic the human visual system it is interesting to introduce peripheral vision to the agent. The peripheral vision of a human is wide but blurry, making it useful for noticing movements but ill-suited for making out details. In order to extend the *Constant* and the *Decreasing* vision windows with something that simulates peripheral vision a small change is made. The pixels outside the focal area, that previously were set to zero, are now replaced with a very low resolution version of the input state. A very low resolution version of an input state can be seen alongside three higher resolution version in Figure 13. Note that the white paddle at the bottom makes the pixels slightly lighter and that the removed bricks makes the pixels slightly darker in the very low resolution input state, as seen in Figure 13d. Further, a visualization of how a focus-of-attention input is created from the RGB game screen can be seen in Appendix A.1.

(a) 80×80 resolution. (b) 40×40 resolution. (c) 20×20 resolution. (d) 5×5 resolution.

Figure 13: Input states in different resolutions.

The formula to compute the maximum amount of pixel-information present in the input state is also changed when the peripheral vision is added. The amount of pixel-information available when the peripheral is added to the *Constant* vision window, I_{C_P}, is computed as follows:

$$I_{C_P} = I_C + 2 \cdot \left(\lceil \frac{(\frac{80-f_w}{2})}{80/R_h^{(P)}} \rceil \cdot \lfloor \frac{80 - (\frac{80-f_h}{2})}{80/R_v^{(P)}} \rfloor + \lceil \frac{(\frac{80-f_h}{2})}{80/R_v^{(P)}} \rceil \cdot \lfloor \frac{80 - (\frac{80-f_w}{2})}{80/R_h^{(P)}} \rfloor \right) \tag{5.17}$$

where the amount of pixel-information I_C for the *Constant* window is computed as shown in Equation 5.16, f_w is the focal width, f_h is the focal height, $R_v^{(P)}$ is the vertical resolution of the

peripheral and $R_h^{(P)}$ is the horizontal resolution of the peripheral. If the peripheral instead is added to the *Decreasing* vision window the amount of pixel-information, I_{D_P}, can be computed the following way:

$$I_{D_P} = I_D + 2 \cdot \left(\lceil \frac{(\frac{80-f_w^{(3)}}{2})}{80/R_h^{(P)}} \rceil \cdot \lfloor \frac{80 - (\frac{80-f_h^{(3)}}{2})}{80/R_v^{(P)}} \rfloor + \lceil \frac{(\frac{80-f_h^{(3)}}{2})}{80/R_v^{(P)}} \rceil \cdot \lfloor \frac{80 - (\frac{80-f_w^{(3)}}{2})}{80/R_h^{(P)}} \rfloor \right) \tag{5.18}$$

where the amount of pixel-information I_D for the *Decreasing* window is computed as shown in Equation 5.15, $f_w^{(3)}$ is the outermost focal width, $f_h^{(3)}$ is the outermost focal height, $R_v^{(P)}$ is the vertical resolution of the peripheral and $R_h^{(P)}$ is the horizontal resolution of the peripheral. Note that $\lceil x \rceil$ is the ceiling function and $\lfloor x \rfloor$ is the floor function.

The Equations 5.17 and 5.18 compute the amount of pixel-information present in the input state when the focal point is located in the center on the screen. The expressions for I_{C_P} and I_{D_P} are somewhat complex, but this is needed to deal with non-square focal areas, non-square resolutions and situations where only a fraction of a peripheral pixel is visible behind the focal area. Two example calculations can be seen in Appendix A.2.

A few examples of input states using the *Decreasing* resolution vision window with peripheral vision can be seen in Figure 14 along with corresponding visualizations which makes it easier to follow what is actually happening when the agent is playing the game.

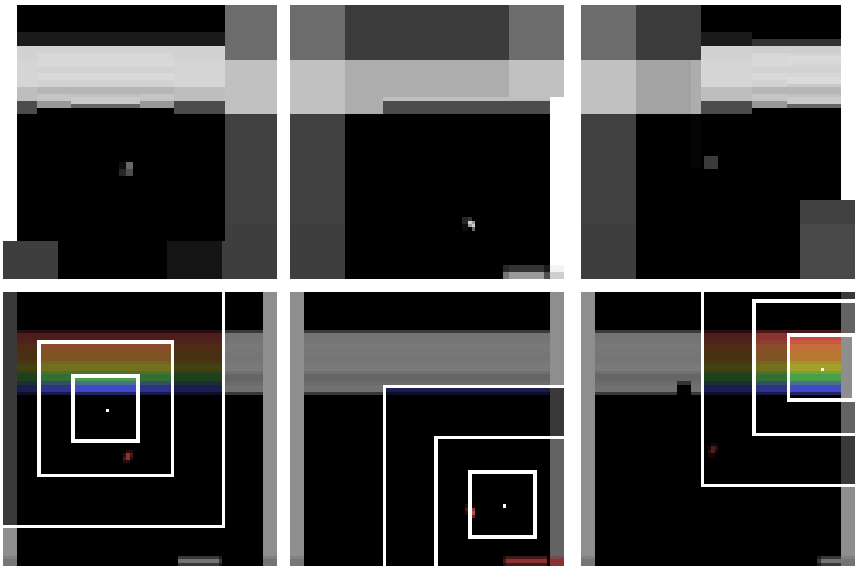


Figure 14: Input states in the top row and corresponding visualizations in the bottom row.

The use of peripheral vision mimics important aspects of the human visual system. The area with higher resolution around the focal point resembles a human's foveated vision while the very low resolution background resembles the peripheral vision of a human. The resolution of the peripheral vision is too low for the agent to make out details but high enough to track movements. The fact that the resolution decreases as the distance to the focal point increases is also true for the human visual system. One aspect that is not included in the vision windows used, is the fact that human vision is almost circular. The choice to instead use rectangular focal areas was made both for simplicity and due to the pixelated nature of the game states.

6 Experiments

Before running the different experiments it is important to know how the different Atari games work. The Atari environments used in this thesis are Atari 2600 games from the Arcade Learning Environment (ALE) suite [1], which is a popular choice for reinforcement learning related research. The games Pong, Breakout and Beam Rider are used in this project and are implemented using OpenAI Gym [2].

6.1 Pong

The game of Pong is very simple. The player controls the green paddle on the right using the following actions: stand still, move up and move down. The orange paddle on the left is the opponent. The goal of the game is to score more points than the opponent and to reach 21 points first. The player can score a point by hitting the white ball and make it go past the opponent on the left. The opponent scores in the same way: by hitting the ball past the player on the right. Every time a point has been scored the ball is reset in the middle of the screen. A typical game state in the game Pong can be seen in Figure 15.

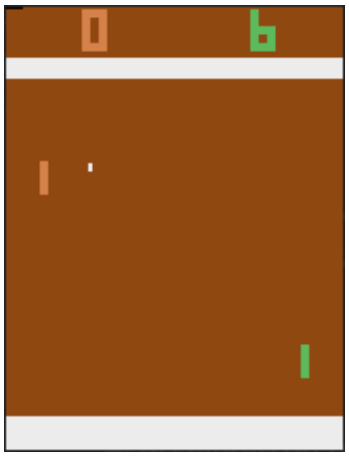


Figure 15: A typical screen in Pong.

The rewards returned from the environment are in accordance with the score of the game from the perspective of the agent: a reward of $+1$ is returned when the agent scores, a reward of -1 is returned when the opponent scores and for all other time-steps a reward of 0 is returned. As the game ends when either the player or the opponent reaches 21 points the maximum score of Pong is 21.

6.2 Breakout

In Breakout the player controls the red paddle on the bottom of the screen by either standing still, moving left or moving right. The goal of the game is to make the ball hit and destroy the bricks that can be seen on the top half of the screen. There are six rows of bricks, represented by different colors. When a brick is destroyed by the ball the ball rebounds and the player will have to hit it back up. If a brick in one of the middle rows is destroyed the speed of the ball increases, making the game more difficult. If the player cannot return the ball and it moves past the paddle at the bottom the player loses a life. The player starts with five lives and if all lives are lost the game is over. Two typical game states can be seen in Figure 16.

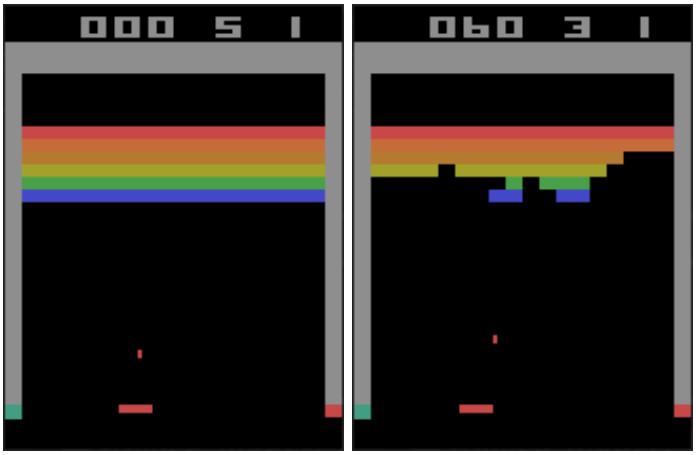


Figure 16: Two typical screens in Breakout.

In Breakout the rewards returned by the environment is in accordance with the game score. A reward of +1 is returned when a brick in the bottom two rows is destroyed, a reward of +4 is returned when a brick in the two middle rows is destroyed and a reward of +7 is returned when a brick in the top two rows is destroyed. As in Pong a reward of 0 is returned for all other time-steps. Using this information the total score for destroying all the bricks can be found to be 432 points. To beat the game of Breakout all bricks need to be destroyed twice. This means that the maximum score of the game is 864.

As mentioned before the three actions stand still, move left and move right can be chosen at each time-step in the game of Breakout. However, there is actually a fourth action available. This action is known as 'FIRE' and its only use within this game is to start the game after a life has been lost. If this action is chosen at any other time it has the exact same effect as choosing to stand still.

6.3 Beam Rider

In Beam Rider the player controls the yellow space-ship at the bottom of the screen and the goal is to destroy enemies by shooting them. The game is divided into sectors and in each sector the player must destroy 15 enemies in order to proceed within the game. As the player is moving through the sectors new enemy types, with unique movement patterns, are introduced. In Beam Rider the player has access to two different types of ammunition: laser shots and torpedoes. The laser shot is unlimited but can only destroy certain enemy types. The torpedoes can destroy everything but the player only has access to three torpedoes per sector. The amount of torpedoes available to the player is represented by purple rectangles at the top right of the screen. At the end of each sector a so called Sector Sentinel appears. This big, red enemy is worth a lot of points but can only be destroyed using torpedoes. It is thus smart to use the torpedoes sparingly during the sectors. The player starts with two extra lives and a life is lost if the player's space-ship is hit by an enemy. Typical game screens from Beam Rider can be found in Figure 17.

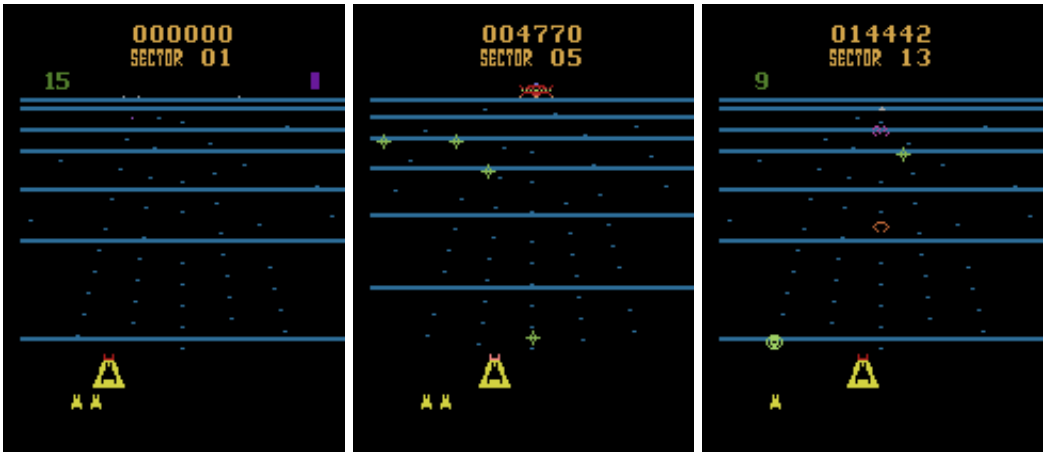


Figure 17: Three typical screens in Beam Rider.

The player gets points when enemies are destroyed. The normal enemies gives a smaller point reward while the Sector Sentinel at the end of each sector gives a larger reward if the player manages to destroy it. The point value per enemy grows as the player makes its way through the sectors. If no enemy is hit during a time-step the reward from the environment is 0. Beam Rider is a game that only ends when all lives are lost. Thus, there are no actual max score and the game can go on indefinitely.

There are nine actions in Beam Rider. The agent can choose to stand stand still, to move left or to move right. The agent can also choose to use the 'FIRE' action to shoot the laser shot or choose the action 'UP' to shoot a torpedo. In the Arcade Learning Environment suite the 'FIRE' and 'UP' actions can be combined with the move left and move right actions. Thus the following actions exist; 'NOOP', 'LEFT', 'RIGHT', 'FIRE', 'FIRE+LEFT', 'FIRE+RIGHT', 'UP', 'UP+LEFT' and 'UP+RIGHT'.

6.4 Experiment Set-up

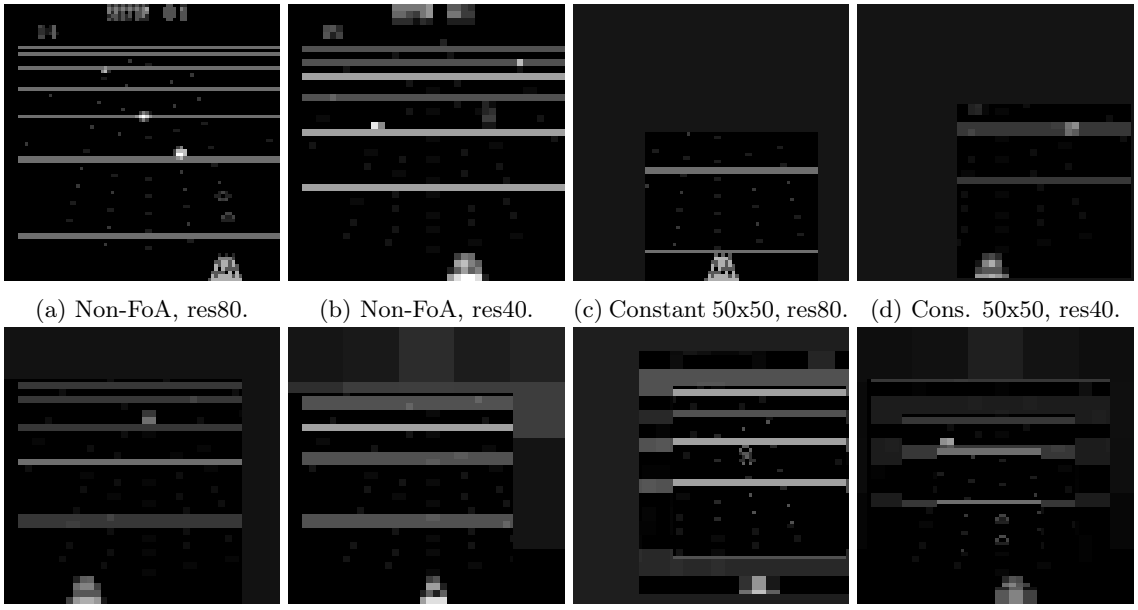
A number of different focus-of-attention models were tested on the Atari games Pong, Breakout and Beam Rider. For each focus-of-attention model the size(s) of the focal area(s), the resolution(s) and the focal step-size need to be chosen. Below the different models are listed:

- *Non-FoA, res80*: The A3C-LSTM model without the FoA task with full 80×80 resolution.
- *Non-FoA, res40*: The A3C-LSTM model without the FoA task with the lower 40×40 resolution.
- *Constant 50x50, res80*: A 50×50 focal area with constant resolution and a focal step-size of 4. Uses a resolution of 80×80. The pixels outside the focal area are set to zero.
- *Constant 50x50, res40*: A 50×50 focal area with constant resolution and a focal step-size of 4. Uses a resolution of 40×40. The pixels outside the focal area are set to zero.
- *Constant 70x70, res40*: A 70×70 focal area with constant resolution and a focal step-size of 4. Uses a resolution of 40×40. The pixels outside the focal area are set to zero.
- *Decreasing 30-50-70*: A focal area made up of a 30×30 area with 80×80 (high) resolution, a 50×50 area with 40×40 (medium) resolution and a 70×70 area with 20×20 (low) resolution. A focal step-size of 4 is used. The pixels outside the focal area are set to zero.
- *Constant(P) 70x70, res40*: The *Constant 70x70, res40* model with added peripheral. The pixels outside the focal area are replaced with a 5×5 resolution version of the game state.
- *Decreasing(P) 30-50-70*: The *Decreasing 30-50-70* model with added peripheral. The pixels outside the focal area are replaced with a 5×5 resolution version of the game state.

The sizes of the focal areas are decided in such a way that they are small enough to force the agent to learn a good policy for the visual actions, yet large enough to let the agent reach good scores on the games. The focal step-size is chosen to be 4 pixels per action in all cases. This value allows the agent to move its focus-of-attention fast enough to track movements while keeping some continuity between time-steps. A summary of the models can be seen in Table 1 and example states can be seen in Figure 18.

Table 1: Summary of the models used in this thesis.

Model name	Focal area(s)	Resolution(s)	Peripheral
Non-FoA, res80	-	80×80	-
Non-FoA, res40	-	40×40	-
Constant 50x50, res80	50×50	80×80	-
Constant 50x50, res40	50×50	40×40	-
Constant 70x70, res40	70×70	40×40	-
Decreasing 30-50-70	30×30, 50×50, 70×70	80×80, 40×40, 20×20	-
Constant(P) 70x70, res40	70×70	40×40	5×5
Decreasing(P) 30-50-70	30×30, 50×50, 70×70	80×80, 40×40, 20×20	5×5



(a) Non-FoA, res80. (b) Non-FoA, res40. (c) Constant 50x50, res80. (d) Cons. 50x50, res40.

(e) Cons. 70x70, res40. (f) Cons.(P) 70x70, res40. (g) Decreasing 30-50-70. (h) Decr.(P) 30-50-70.

Figure 18: Example states for the different models used in the thesis.

As mentioned in Sections 5.4 and 5.5, where the vision windows were introduced, the amount of pixel-information present in a model is an interesting metric to consider when comparing results. Thus, the Equations 5.15, 5.16, 5.18 and 5.17 are used to compute the amount of pixel-information present in the different models. The results of the computations are presented along with the full size of the focal area in Table 2.

Table 2: Model comparison.

Model name	Total focal area	Pixel-information
Non-FoA, res80	-	6400
Non-FoA, res40	-	1600
Constant 50x50, res80	50×50	2500
Constant 50x50, res40	50×50	625
Constant 70x70, res40	70×70	1225
Decreasing 30-50-70	70×70	1450
Constant(P) 70x70, res40	70×70	1241
Decreasing(P) 30-50-70	70×70	1466

The hyperparameters used by the models were initially set to values that have been shown to work well for the A3C-LSTM without focus-of-attention [7] and then tuned within a limited range. One important hyperparameter is the λ introduced in the section Generalized Advantage Estimation. This choice is a trade-off between bias and variance and as the Atari games are very deterministic (only the number of no-operation actions at the start of each episode and the initialization of the focal point introduces randomness to the game) the GAE-λ is chosen in a way that introduces quite a lot of variance to the model. Another hyperparameter of importance is the amount of action-learners used by the A3C. Increasing the amount of action-learners improves convergence properties but increasing it too far can slow down training if not enough computing resources are available. In this thesis 32 action-learners are used. A full list of hyperparameters can be found in Appendix A.3.

The experiments were run on a node on Berzelius, which is the premier AI/ML cluster at the National Supercomputer Centre (NSC) at Linköping University. Berzelius is made up of 60 NVIDIA® DGX-A100 compute nodes and each of these nodes is equipped with 8 NVIDIA® A100 Tensor Core GPUs [10].

7 Results

The results of the experiments are presented in this section. The performance of an agent throughout the training process is quantified as the score per episode. While the training agents in the 32 action-learner threads choose their actions probabilistically from the trained policies, a test agent, that runs on its own thread, evaluates the model by choosing actions greedily with respect to the policies. After each evaluation the score is saved and the thread is put to sleep for a pre-determined amount of seconds before starting the next evaluation. In this thesis the thread is set to sleep for 20 seconds between each evaluation.

During training the score can vary a lot between consecutive tests. In order to better visualize how the score changes over time, an exponential moving average, EMA, is introduced. By replacing the normal score with the EMA it is easier to see trends and to compare the performance between different models. The exponential moving average for time-step t can be computed the following way:

$$EMA_t = \kappa \cdot SCORE_t + (1 - \kappa) \cdot EMA_{t-1} \tag{7.1}$$

where κ controls how much previous scores will affect the next EMA. For the results below κ was chosen to be 0.05. For this choice of κ the latest score, $SCORE_t$, will make up for 5% (0.05) of the average, $SCORE_{t-1}$ for 4.75% ($0.05 \cdot 0.95$), $SCORE_{t-2}$ for 4.51% ($0.05 \cdot 0.95^2$) and so on. The latest score will thus always be the most important factor in the average.

The scores, or the EMAs, are plotted both against time and against the number of episodes. The score versus time plots can be used to compare the different models while the score versus episode plots are provided to allow for comparisons with experiments run on different hardware set-ups.

First, results from three models described in Section 6.4 are presented. The three models are *Non-FoA, res80*; *Non-FoA, res40* and *Constant 50x50, res80*. The first model is the standard model without focus-of-attention and full 80×80 resolution. This model is mostly used for comparisons. The other two models represent different ways of limiting the amount of pixel-information present in the input state. In the model *Non-FoA, res40* sub-sampling is used on the full image to decrease the amount of pixel-information from 6400 pixels to 1600 pixels (see Table 2). For *Constant 50x50, res80* the amount of pixel-information is decreased to 2500 pixels (or to a lower amount depending on the location of the focal point) by applying a vision window with constant resolution. This third model was the one used in previous work done on the subject [14]. All three models can be considered baselines in this thesis. The performance of the different models can be seen in Figures 19-21.

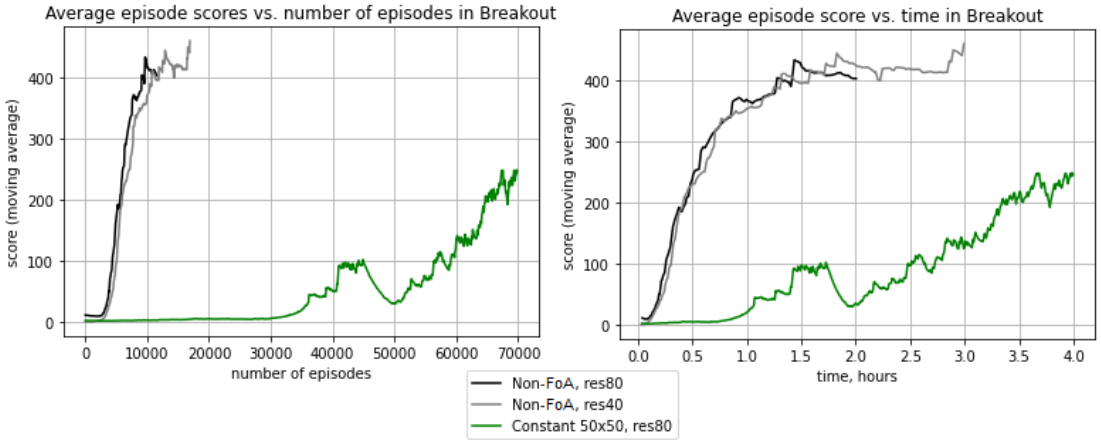


Figure 19: Performance in Breakout for baseline models.

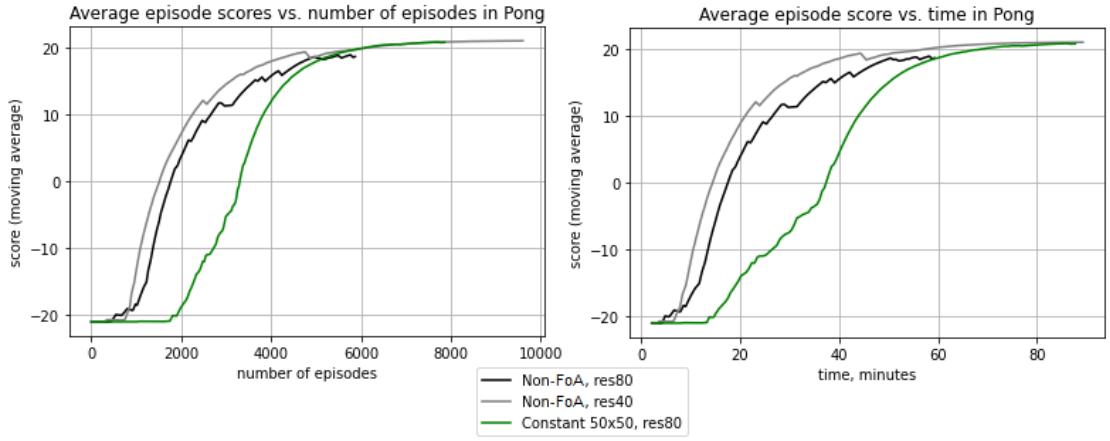


Figure 20: Performance in Pong for baseline models

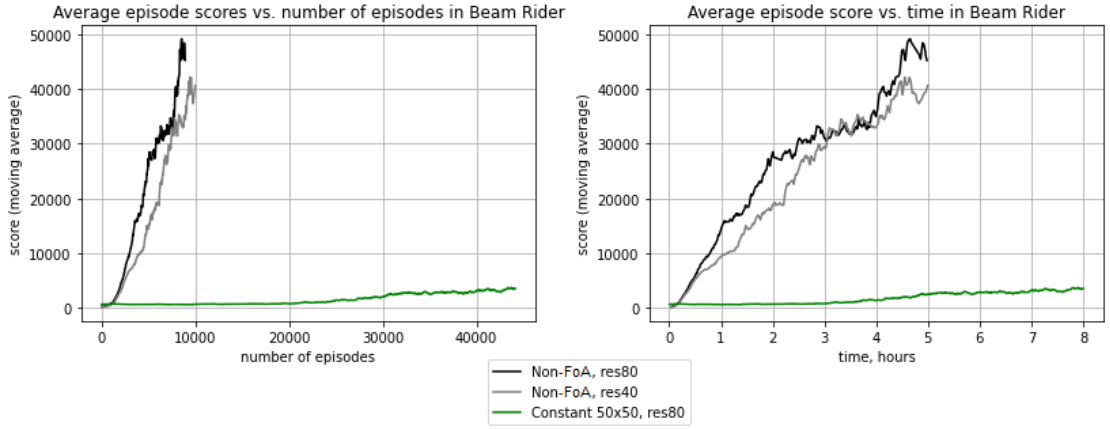


Figure 21: Performance in Beam Rider for baseline models

The Figures 19-21 showed that the sub-sampling done in the model *Non-FoA, res40* had little effect on the performance. The lower resolution caused small drops in performance on Breakout and Beam Rider while it actually slightly improved the performance on Pong. For all the games the addition of the focus-of-attention task slowed down performance significantly. This drop in performance was the biggest for Beam Rider and smallest for Pong.

As the lower resolution barely had any effect on the performance the next step involved constant vision windows with lowered resolution. In this step the models *Constant 50x50, res40* and *Constant 70x70, res40* were introduced and plotted alongside *Non-FoA, res80* and *Constant 50x50, res80*. The 50×50 window with lowered resolution obviously holds less pixel-information than the previously used 50×50 window. In fact it only used 625 pixels worth of information (see Table 2). More interesting is the fact that *Constant 70x70, res40* uses 1225 pixels worth of information compared to the 2500 of *Constant 50x50, res80*. This means that the larger window covers more of the screen while still using less pixel-information. The performance of the models can be seen in Figures 22-24.

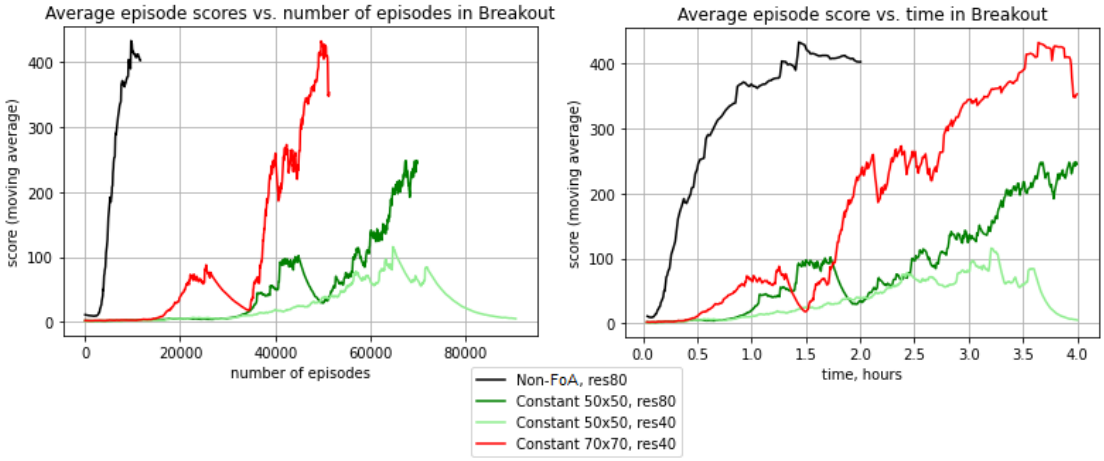


Figure 22: Performance in Breakout for low resolution constant vision windows.

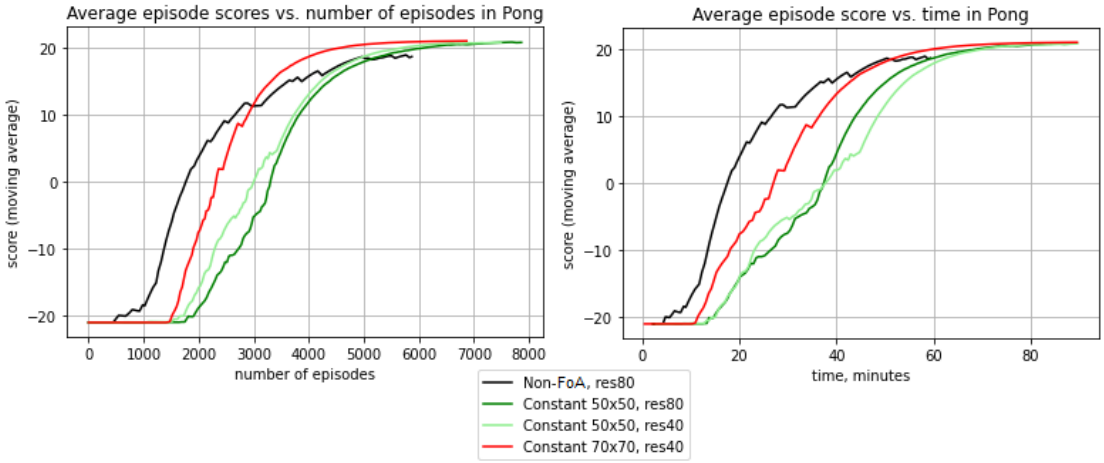


Figure 23: Performance in Pong for low resolution constant vision windows.

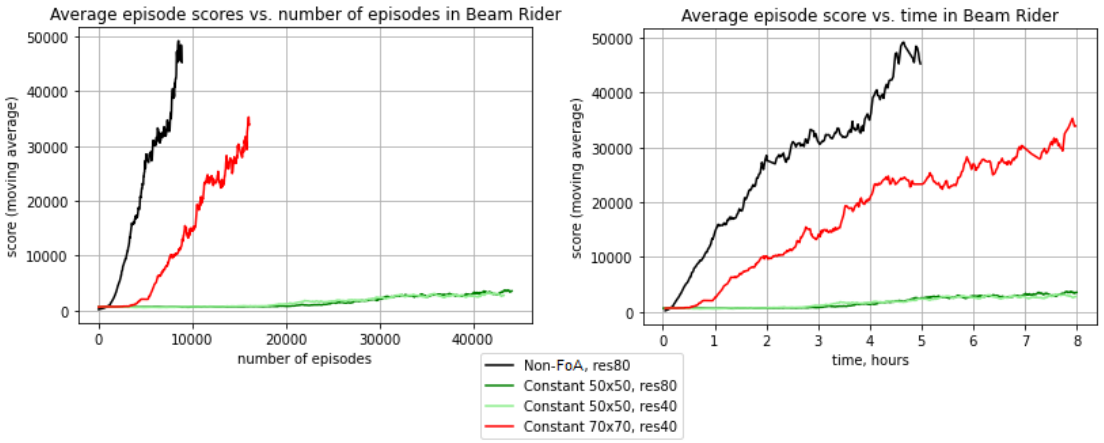


Figure 24: Performance in Beam Rider for low resolution constant vision windows.

Figures 22-24 show that the model *Constant 70x70, res40* performs better than the *Constant 50x50, res80* model for all games. The *Constant 70x70, res40* model even performs better (reaches the maximum score faster) than the *Non-FoA, res80* model in Pong. The lowered resolution in *Constant 50x50, res40* has some effect on performance. While the drop in performance is next to none for Pong and Beam Rider the performance drop is large for Breakout.

The third step introduced the *Decreasing 30-50-70* model. This model uses multiple resolutions and is compared with the *Constant 70x70, res40* model that uses the same total size of the focal area. The *Decreasing 30-50-70* model contains more pixel-information than the *Constant 70x70, res40* model close to the focal point while it contains less pixel-information further from the focal point. In total the *Decreasing 30-50-70* model uses a maximum of 1450 pixels worth of information while *Constant 70x70, res40* uses 1225 pixels (see Table 2). The two models thus uses a similar amount of pixel-information. Comparisons of the models can be found in Figures 25-27.

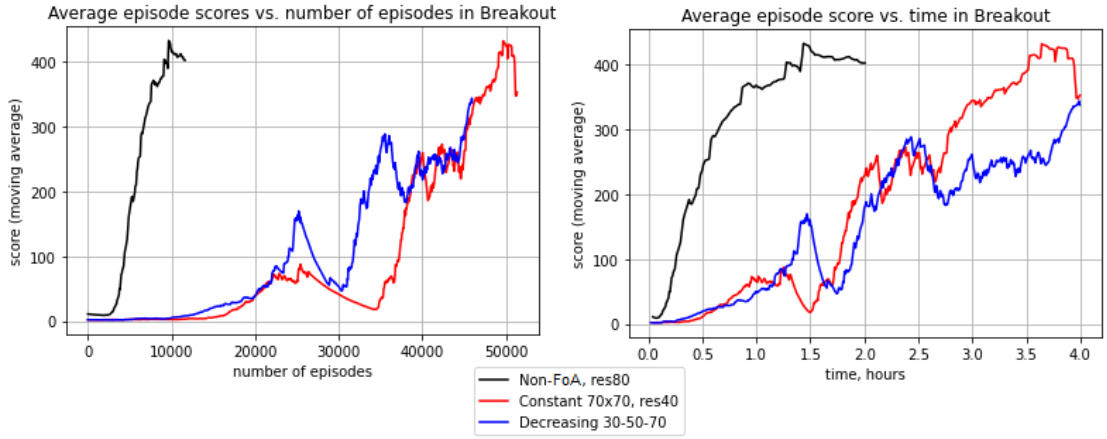


Figure 25: Comparison between constant and decreasing vision windows in Breakout.

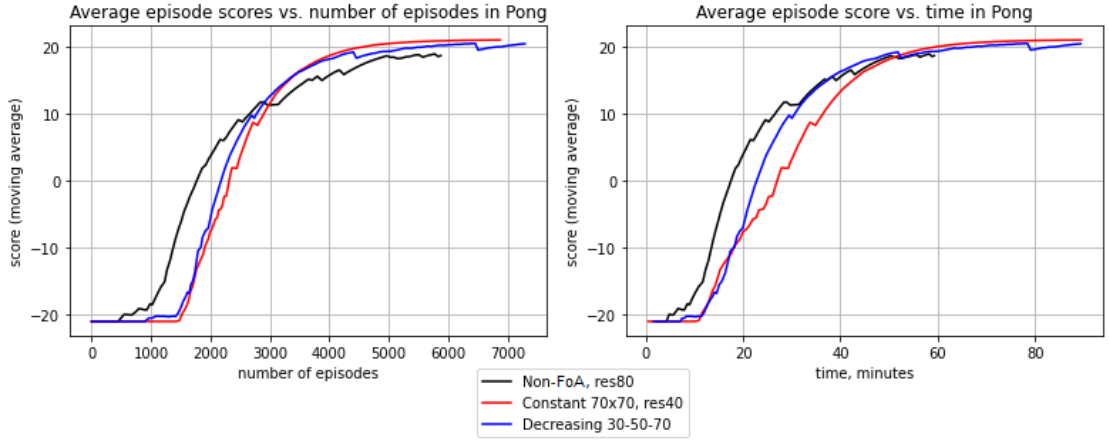


Figure 26: Comparison between constant and decreasing vision windows in Pong.

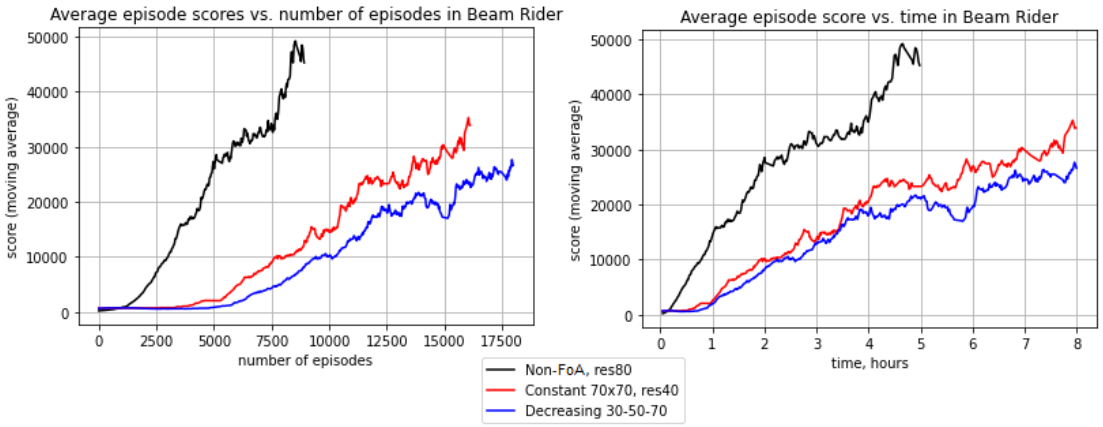


Figure 27: Comparison between constant and decreasing vision windows in Beam Rider.

The Figures 25-27 show that the introduction of the decreasing vision window does not, in general, improve performance. In Breakout the performance of the two models (*Constant 70x70, res40* and *Decreasing 30-50-70*) are somewhat similar but during the last hour the model with the constant resolution performs better. In Pong the decreasing resolution vision window performs slightly better and in Beam Rider the constant vision window would be preferable.

As a last step peripheral vision is added to the models. This means that the models *Constant(P) 70x70, res40* and *Decreasing(P) 30-50-70* are now in focus. By adding these 5×5 resolution backgrounds a maximum of only 16 pixels worth of information is added to the input states (see Table 2). These models are compared with the models *Constant 70x70, res40* and *Decreasing 30-50-70* in order to see what effect the addition of the peripheral has on the performance. The results are presented in Figures 28-30.

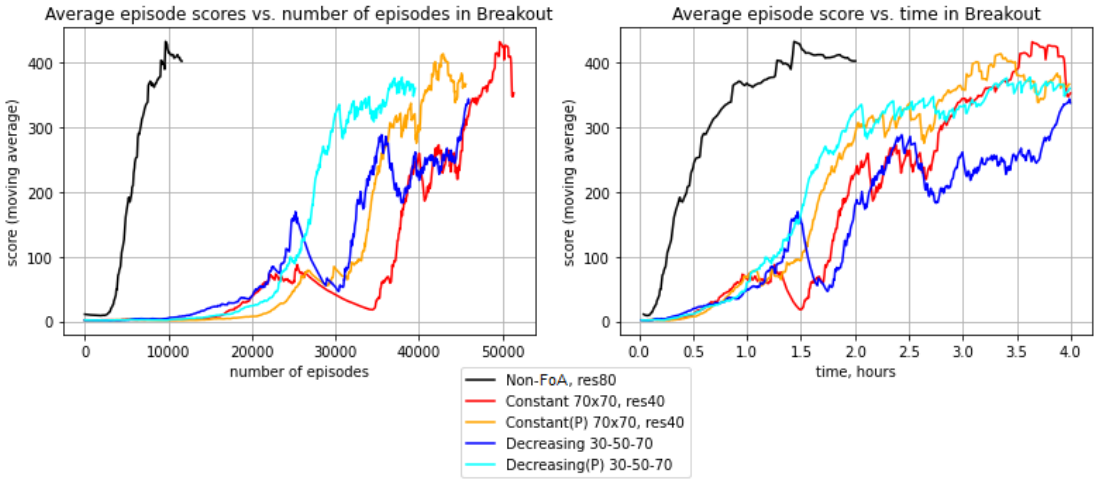


Figure 28: Performance of models with added peripheral in Breakout.

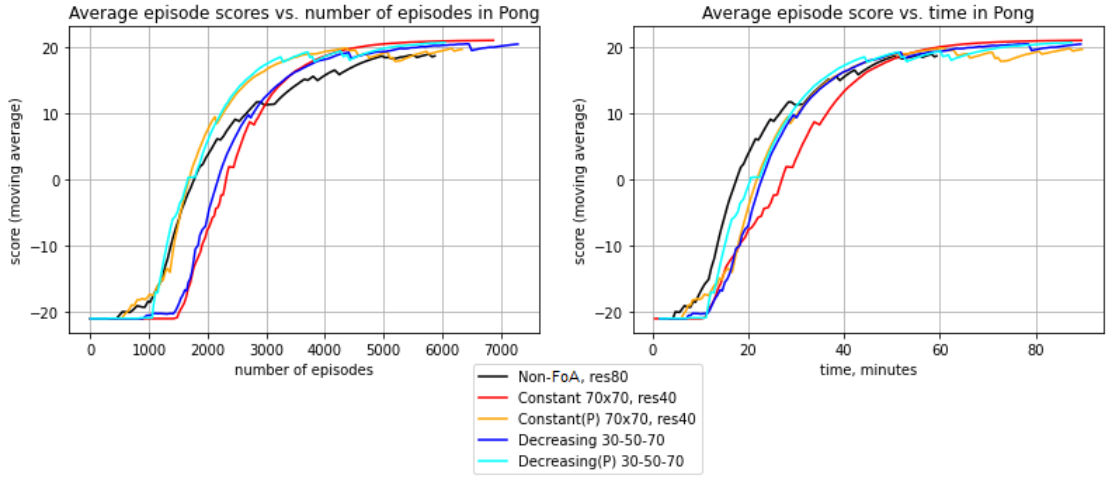


Figure 29: Performance of models with added peripheral in Pong.

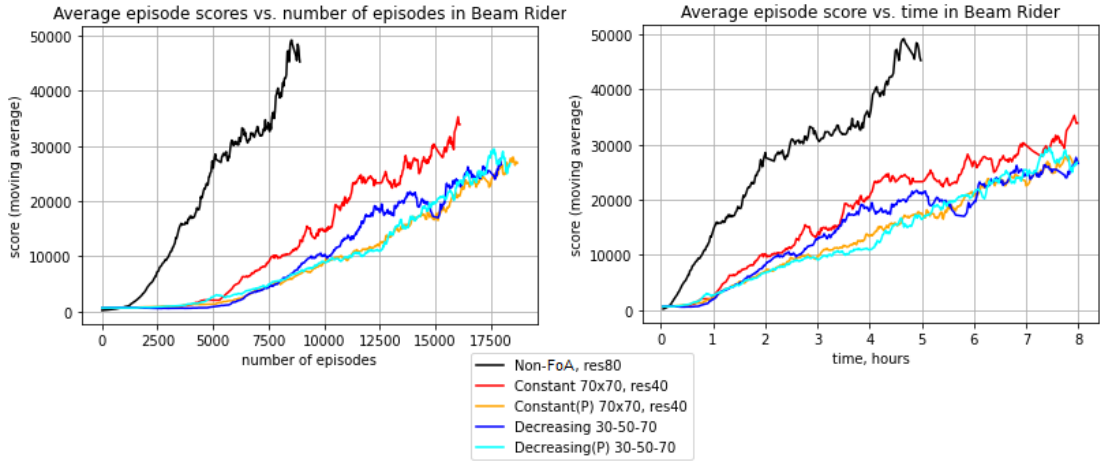
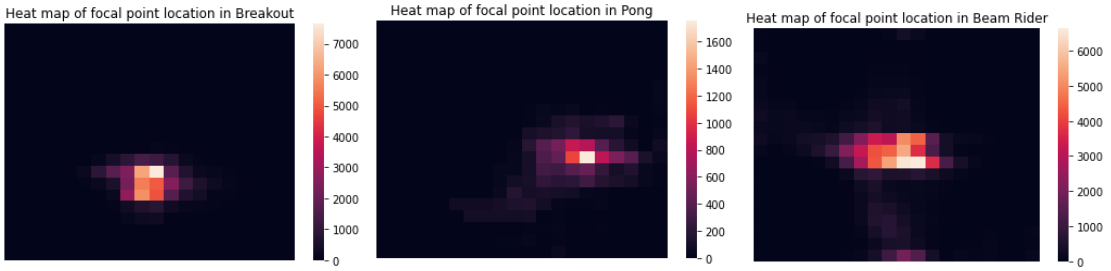


Figure 30: Performance of models with added peripheral in Beam Rider.

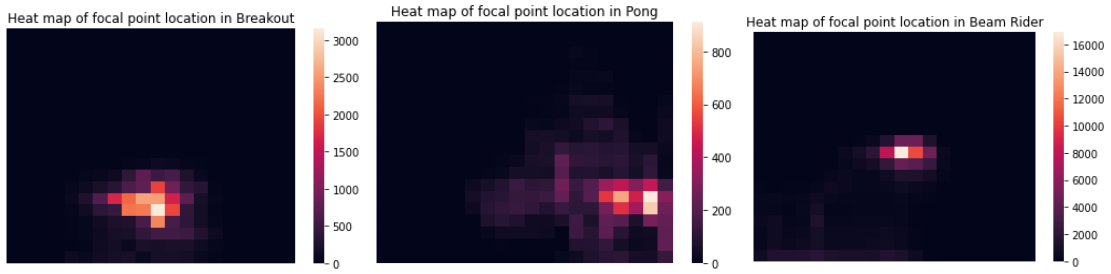
The results in Figures 28-30 show that the addition of the peripheral can have a big impact on the performance. On Breakout and Pong the addition of the peripheral improves both the constant and the decreasing resolution models significantly. On both of these games the *Decreasing(P) 30-50-70* model can be considered to perform the best out of all the focus-of-attention models. In the case of Pong, both the *Constant(P) 70x70, res40* and the *Decreasing(P) 30-50-70* models perform better than the standard *Non-FoA, res80* model. In the last game, Beam Rider, the addition of the peripheral does not improve the model. For the decreasing resolution model the performance remains more or less unchanged while the performance of the constant model gets slightly worse. On Beam Rider, the best focus-of-attention model is the *Constant 70x70, res40* model.

To visualize where the agent is looking while playing the games the movement of the focal point during 10 full games were recorded and turned into so-called heat maps. Common positions are bright while uncommon positions are dark. Heat maps for the different games can be found in Figure 31 and Figure 32 and the figures are related to the models *Constant 70x70, res40* and *Decreasing(P) 30-50-70*, respectively.



(a) Breakout. (b) Pong. (c) Beam Rider.

Figure 31: Heat maps of the focal point for the different games for model *Constant 70x70, res40*.



(a) Breakout. (b) Pong. (c) Beam Rider.

Figure 32: Heat maps of the focal point for the different games for model *Decreasing(P) 30-50-70*.

8 Discussion

8.1 Non-FoA Models

The first round of results, presented in Figures 19-21, show the performance of two non-FoA models. These plots show that lowering the resolution from 80×80 to 40×40 barely affects performance. In Pong, the performance is actually improved when lowering the resolution while the effect on Breakout is minimal and the performance on Beam Rider gets a little worse. By lowering the resolution some information is obviously lost but the size of the state-space is reduced at the same time. The loss of pixel-information can deteriorate the performance while the reduction of the state-space can improve performance. For a visually simple game like Pong (only two paddles and one ball are present) the lower resolution does not result in any real loss of information. This means that the reduction of the size of the state-space is enough to actually improve the model. The opposite happens for Beam Rider. This game is visually more complex and the lower resolution thus results in the loss of some important information. As a result the reduction of the state-space is not enough to speed up the learning.

8.2 Constant Resolution Models

In Figures 19-21 focus-of-attention is introduced with the *Constant 50x50, res80* model. This model is similar to the one used in the previous thesis on the subject [14]. The results show that the performance of the model is good on Pong and Breakout while the model struggles with Beam Rider. The performance on Pong and Breakout can be considered near-optimal; on Pong the maximum score is reached quite quickly and on Breakout all bricks of the first stage is almost cleared. The performance on Beam Rider, while not fast, can be considered okay as there is a steady but slow improvement. The learning of this model is however much slower than the learning of the non-FoA models.

As suggested in Section 8.1, a change in resolution from 80×80 to 40×40 does not affect the performance very much. Because of this two new constant resolution vision windows were added in the second round of results, as seen in Figures 22-24. The first new model, *Constant 50x50, res40*, is simply a lower resolution version of *Constant 50x50, res80* using 625 pixels worth of information instead of 2500. While there is no real effect of this lowered resolution on Pong and Beam Rider, the new model is struggling to play Breakout. The fact that there is no real effect on Pong is expected; the game is visually very simple and the lower resolution makes no real difference. The problems with Breakout are also not that surprising. The small window combined with lower resolution could cause problems for this visually more challenging game. It is more surprising that there is no real effect of the lowered resolution on Beam Rider. It was on this visually challenging game that the difference in performance between the models *Non-FoA, res80* and *Non-FoA, res40* was the largest. The similar performance of the models *Constant 50x50, res80* and *Constant 50x50, res40* on Beam Rider could be explained by the fact that small vision window is a larger problem than the lower resolution. In other words, the small vision window is the bottleneck for these models and this property causes both the models to perform badly.

The next model to consider is the *Constant 70x70, res40* model presented in Figures 22-24. This model uses a larger focal area than *Constant 50x50, res80* but a lower resolution. This setting makes sure that the new *Constant 70x70, res40* model uses less pixel-information than the *Constant 50x50, res80* model: 1225 pixels worth of information instead of 2500. For all games the new constant resolution model performs better than the previous constant resolution models used in this thesis. The improvement is very big for Breakout and Beam Rider, but the performance on these games is still worse for this model than for the standard *Non-FoA, res80* model. For Pong, the *Constant 70x70, res40* model performs better than the *Non-FoA, res80* model in the sense that it reaches the maximum score of 21 faster. The results of this new constant model with larger focal area suggest that the size of the vision window is more important than the resolution of the input state.

The Figures 22-24 also show that the introduction of the focus-of-attention task does not, in general, speed up the training process. The plots further show that the beginning of training is

especially slow for the FoA models. This is probably due to the fact that the size of the state-space is actually increased for untrained models when focus-of-attention is introduced. When applying the vision window the amount of actual game states remain unchanged and for each of these states there now exists many different input states based on the location of the focal point. It is not until the agent learns to control its focus-of-attention that the size of the state-space is effectively reduced. On top of this the size of the action-space is increased which further slows down training. These different effects can be seen in Figure 23 depicting the performances of models on the game of Pong. For the first couple of minutes the FoA models struggle to improve where the non-FoA model does. This can be due to the fact that the size of the state-space is initially increased for the FoA models. Then, as the models improve and the vision window is controlled with success, the performance of the FoA models improve faster than the non-FoA model as the size of the state-space effectively has been reduced.

8.3 Decreasing Resolution Models

In Figures 25-27 the *Constant 70x70, res40* model is compared to the *Decreasing 30-50-70* model. This model uses a decreasing resolution vision window which means that it uses three different resolutions within its focal area. In a 30×30 area around the focal point the resolution is 80×80. The resolution is then reduced to 40×40 for the middle focal area of size 50×50. The outer focal area is of size 70×70 and uses a resolution of 20×20. The two models used are using the same total focal area while the decreasing model uses higher resolution in its center and lower resolution further from the focal point. The changes in resolution changes the amount of pixel-information used by the model from 1225 to 1450 pixels (see Table 2). The amount of pixel-information is thus almost the same for the two models.

For Breakout and Beam Rider the use of the decreasing resolution deteriorates the performance slightly. The performance on Pong is more or less unchanged. As discussed earlier in both Section 8.1 and Section 8.2 the agent is not affected much when changing the resolution between 80×80 and 40×40, which means that the higher resolution in the center of the *Decreasing 30-50-70* model does not improve the model in any significant way. The drop in resolution from 40×40 to 20×20 in the outer focal area, however, seems to cause some problems for the agent when learning. With this low resolution (20×20) the information loss is large enough to make a difference. This is probably why the performance of *Decreasing 30-50-70* is slightly worse than the performance of *Constant 70x70, res40* for Breakout and Beam Rider. The changes in resolution does not, however, seem to affect the visually simple game of Pong. In Pong, the low resolution of 20×20 is probably enough for the agent to locate the ball and the paddles. This means that there is no real difference between the two models.

The different resolutions of the *Decreasing 30-50-70* model can possibly make the training process more challenging for the agent. When only using one resolution a certain game object always look the same but when using three different resolutions the same game object can take on multiple shapes. A ball can look like a sharp dot if located in the center of the focal area but it can also look like a blurry blob if located further from the focal point. Also, at some point in time a certain area of the screen can be given in high resolution and at another point in time this same area can be given in a lower resolution. In other words, the multiple resolutions of the decreasing vision window introduces a new challenge for the agent.

It should also be noted that the *Decreasing 30-50-70* model performs better than the constant resolution windows of size 50×50 presented in Figures 22-24, even though the decreasing model did not outperform the *Constant 70x70, res40*. The *Decreasing 30-50-70* model can be seen as a 50×50 window padded with a lower resolution area. It is thus clear that the low resolution (20×20) area adds valuable information to the model. Again, it would seem that the size of the focal area is more important than the resolution used.

8.4 Peripheral Vision

The final models are shown in Figures 28-30. Here, the *Constant 70x70, res40* model is extended with peripheral vision forming the *Constant(P) 70x70, res40* model and the *Decreasing 30-50-70*

model is extended the same way, creating the *Decreasing(P) 30-50-70* model. The addition of the 5×5 resolution background known as the peripheral only adds a maximum of 16 pixels worth of information to the model. The peripheral vision is too low resolution to make out details, but should be enough to track some movements and notice changes.

The results shown in Figures 28-30 are rather interesting; the addition of the peripheral when playing Breakout or Pong improves the performance while the addition of the peripheral when playing Beam Rider seems to cause problems for the agent. Starting with the *Constant(P) 70x70, res40* model it can be seen that the performance on Breakout and Pong improves while the performance on Beam Rider gets worse with the addition of the peripheral vision. Moving on to the *Decreasing(P) 30-50-70* model it can be seen that the addition of the peripheral improves the performance on Breakout and on Pong while the performance on Beam Rider remains unchanged. The fact that both models improve on Pong when adding the peripheral is probably due to the fact that the very low resolution is enough to track the objects in this visually simple game. Also, as discussed in Section 8.3, the two models *Constant 70x70, res40* and *Decreasing 30-50-70* are technically very similar when playing Pong which means that the two new models, *Constant(P) 70x70, res40* and *Decreasing(P) 30-50-70*, should also be very similar. On Pong, both models with the peripheral performs better than the standard *Non-FoA, res80* model.

The improvement on Breakout when adding the peripheral is significant, even though the peripheral only adds a maximum of 16 pixels worth of information to the model. When playing Breakout the focal area is generally located on the lower half on the screen, which is visualized by the heat maps in Figures 31a and 32a. This means that the agent usually has no information on the locations of the remaining bricks. The very low resolution peripheral can however help with this; it can give the agent enough information to know on which part of the screen the remaining bricks are located. Also, the improvement on Breakout is in accordance with the previous observation that the size of the focal area is more important than the resolution.

The results on Beam Rider is not, however, in accordance with this. The addition of the peripheral does not affect Beam Rider very much. There is even a drop in performance for the constant resolution model when adding the peripheral. The unchanged performance of Beam Rider for the decreasing resolution model can be explained by the fact that the agent usually keeps its focus-of-attention in the center of the screen, as seen in the heat maps in Figures 31c and 32c. This means that the peripheral only adds some pixels by the edges of the screen which barely hold any information. The deteriorated performance of the constant resolution model is more difficult to explain. One explanation could be the variance that is present in all the models. Another explanation could be the fact that the added peripheral makes the agent move its vision window less frequently. In the heat maps for Beam Rider (Figures 31c and 32c) it can be seen that for a model without peripheral the focal point moves more than for a model with peripheral. With the added peripheral, that mostly adds information along the edges, the agent seems to be less likely to move its focus-of-attention. This could cause the model to perform worse as this added information is not precise enough to base decisions on.

A final remark on the results is that the variance of the models can cause certain models to perform better or worse if re-trained. More specifically, it has been noticed that there is a certain chance that a model gets stuck in a local minima for some time when playing Breakout. If an agent gets stuck in a local minima too long this can greatly deteriorate performance. Thus, it would have been interesting to run each model multiple times to get more reliable statistics. However, due to constraints on time and on the available resources on Berzelius, this was not possible.

When comparing the different FoA models the *Decreasing(P) 30-50-70* model performs the best on Pong and Breakout, while the *Constant 70x70, res40* model performs the best on Beam Rider. It should also be noted that all the models with a total focal area of size 70×70 performs better than the *Constant 50x50, res80* model that was presented in previous work [14].

8.5 Analysis of Agent Behavior

When looking at the heat maps in Figure 31 it is easy to see that the focal point does not move much in a trained model. In Breakout, the most important area of the screen seems to be the

bottom half where the paddle is located while the right half is most important when playing Pong as this is the area where the player's paddle is located. In Beam Rider the vision window is usually quite centered but is occasionally moved down to the bottom. While it is true that there are not much movement of the vision window it should be noted that when comparing Figure 31 and Figure 32 it can be seen that there are more movement for the more successful model. For Breakout and Pong, the *Decreasing(P) 30-50-70* was the best model and the heat maps in Figures 32a and 32b suggest more movement than the heat maps in Figures 31a and 31b. The opposite is true for Beam Rider; *Constant 70x70, res40* was the better model and the heat map in Figure 31c shows more movement than the heat map in Figure 32c. The movement of the focus-of-attention thus seems to be of importance.

In Breakout the agent generally struggles to clear all the bricks of the first stage. The agent plays well until only a few bricks remain, at which point the agent can get stuck in an endless loop or just simply miss the ball. Due to the fact that the focal area of the agent usually covers the lower half of the screen it is not surprising that the agent struggles to hit the last couple of bricks as it cannot see them. This problem is to some extent solved by the added peripheral in the model *Decreasing(P) 30-50-70*. The agent also uses a rewarding tactic while playing Breakout; the agent creates a tunnel through the brick wall and shoots the ball through said tunnel allowing the ball to bounce on the top of the top row. This tactic quickly generates a lot of points without the need of hitting the ball a single time. When using this tactic the agent does not, however, move its focus-of-attention up to see what is going on. Instead it patiently waits until the ball comes back down. This is the main difference between how the agent and a human would play, as a human probably would follow the ball with its vision when it bounces on the top of the brick wall.

When the agent is playing Pong it usually keeps its focus-of-attention on the right side of the screen tracking the movements of the paddle and the ball. The agent tends to follow the ball slightly to the left with its focus-of-attention after a return but the focal area is very rarely moved all the way to the left side. The paddle of the opponent is thus almost never present within the focal area. This is however not needed as the agent knows how the opponent will return the ball based on the ball's trajectory. When training, the agent quickly learns how to score. Hitting the ball a certain way (from the correct angle and with the correct speed) will always result in a point as the opponent will not be able to return the ball. The agent usually learns a few different variations of this shot and simply uses those shots over and over during a match. This approach to the game almost always results in a 21 - 0 win. It should also be noted that after the agent has scored the ball is reinitialized in the center in a deterministic way which leads to a situation where the ball will always end up in the lower right corner. Because of this the agent tends to keep its focus-of-attention in the lower right corner, as seen in the heat map in Figure 32b. This is quite similar to how a human would play Pong. A human would, however, probably look at the opponent when it is hitting the ball. Also, the agent can hit the ball in the correct way to score with much higher precision than a human could making the agent a better player.

As mentioned before, the agent usually keeps its focus-of-attention centered when playing Beam Rider. The agent tries to make sure to keep the top pixels of the space-ship within the focal area to be able to track its movements. The agent also tends to move its focus-of-attention down to the space-ship when the Sector Sentinel appears and multiple projectiles are shot towards the player. The focal area is probably moved to make it easier to avoid all these projectiles. Further, the agent has some non-optimal behaviors. Firstly, the agent tends to shoot all its torpedoes in the beginning of the game, and secondly, the agent usually tries to shoot enemies with the laser that cannot be destroyed. The first non-optimal behavior is probably due to the long length of a sector. The agent needs to be very long-sighted in order to wait until the end of a sector before using the torpedoes. The second non-optimal behavior could partly be because the agent cannot tell the enemies apart with the low resolution and partly be because there is no harm in shooting the invulnerable enemies if the agent still avoids being hit by the enemy. To summarize, the agent plays the game quite similar to how a human player would and positions its focus-of-attention where most human players would look. A human player would however have an easier time with saving the torpedoes until they are actually needed.

9 Conclusions

In this thesis the A3C-LSTM network was extended with a focus-of-attention mechanism and trained on three different Atari 2600 games. An agent was required to both learn to control its movements and its focus-of-attention. Rectangular vision windows with constant resolution, decreasing resolution and with peripheral vision were used. In general, the introduction of the focus-of-attention task slowed down the training process. The drop in performance is probably due to the larger action-spaces and the larger state-spaces of the untrained FoA models. In Pong there are however signs that once the agent learns to control its focus-of-attention the state-space can effectively be reduced leading to a speed-up of the training process.

The A3C-LSTM with focus-of-attention was shown to perform near-optimally on the Atari games Pong and Breakout with a 50×50 constant resolution vision window. It struggled more on the visually challenging game Beam Rider but the performance was improving steadily, although slowly. By introducing a 70×70 constant vision window with lower resolution is could be shown that the size of the vision window is more important than the resolution of the input state. High resolution pixel-information is not that much more valuable than lower resolution pixel-information and lower resolution pixel-information is much better than no information at all. The larger vision window with lower resolution is using 1225 pixels worth of information and is performing better than the smaller vision window with high resolution that is using 2500 pixels worth of information.

Using a decreasing resolution vision window, which still uses a focal area of size 70×70 and almost uses the same amount of pixel-information but with varying resolution did not speed up performance further. The higher resolution closest to the focal point does not make up for the loss of information related to the lower resolution far from the focal point. The difference between a 40×40 resolution and a 20×20 resolution seems to be more significant than the difference between a 80×80 resolution and a 40×40 resolution. The decreasing resolution vision window does, however, perform better than the small constant window. The addition of the extra low resolution area does, in other words, improve the performance in a significant way. Again, it would seem like the size of the vision window is more important than having a high resolution.

The addition of the peripheral vision adds very little pixel-information to the model but affects the performance quite a lot. When the peripheral is added to the two vision windows the performance is improved for Breakout and Pong. In both these cases the peripheral seems to provide important information and the resulting model has the best overall performance on these games. The peripheral can probably be used to keep track of the opponent's paddle in Pong and on the location of the remaining bricks in Breakout. The same improvement can not be seen in Beam Rider. This is probably due to the fact that the agent keeps its focus-of-attention centered while playing Beam Rider, making the peripheral add only unimportant information at the edges of the screen.

In summary, a large focal area is more important than the use of high resolution. With a larger focal area the agent barely has to move its focus-of-attention and can instead focus on positioning the focal point in a good spot. The heat maps provided in the Results section reflect the most important areas of the screen when playing a specific game. This information could in principle be used to highlight important parts and to filter out information of no importance with hopes of reducing the state-space and speeding up the training process.

A Appendix

A.1 Network Architecture

The overview of the A3C-LSTM with focus-of-attention, first presented in the section Focus-of-Attention, can be seen below:

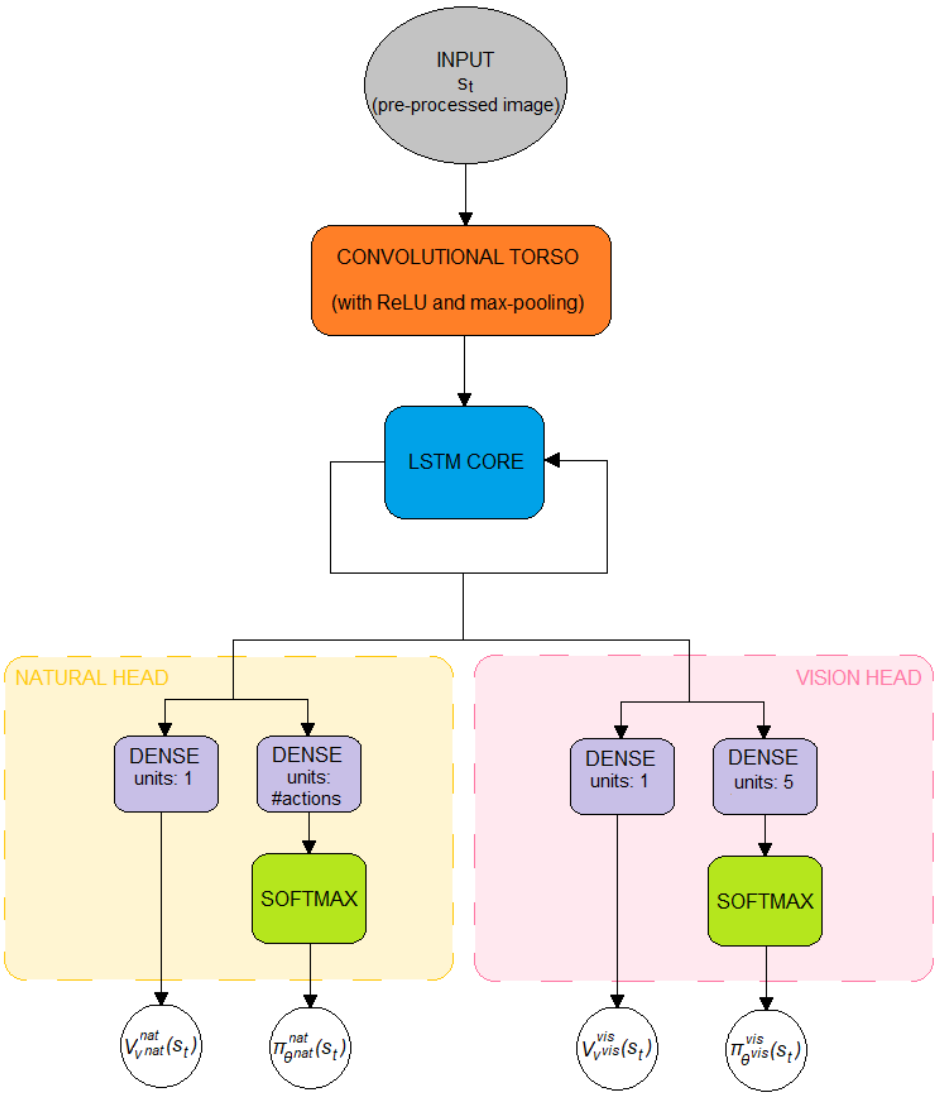


Figure 33: Overview of the architecture of the A3C-LSTM algorithm with focus-of-attention.

As the convolutional torso includes many important details that are not present in the figure above, a more detailed illustration of the convolutional torso is provided below:

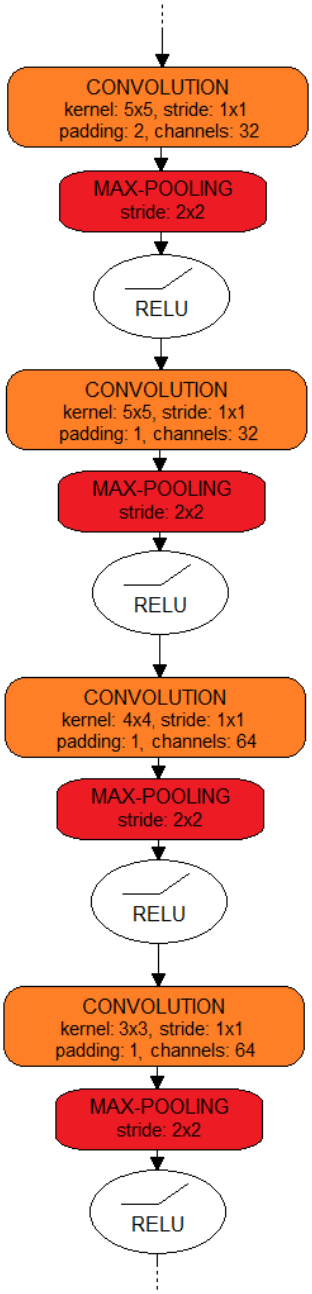


Figure 34: Detailed view of the convolutional torso of the A3C-LSTM algorithm with focus-of-attention.

A visualization of how a focus-of-attention input is created can be found below:

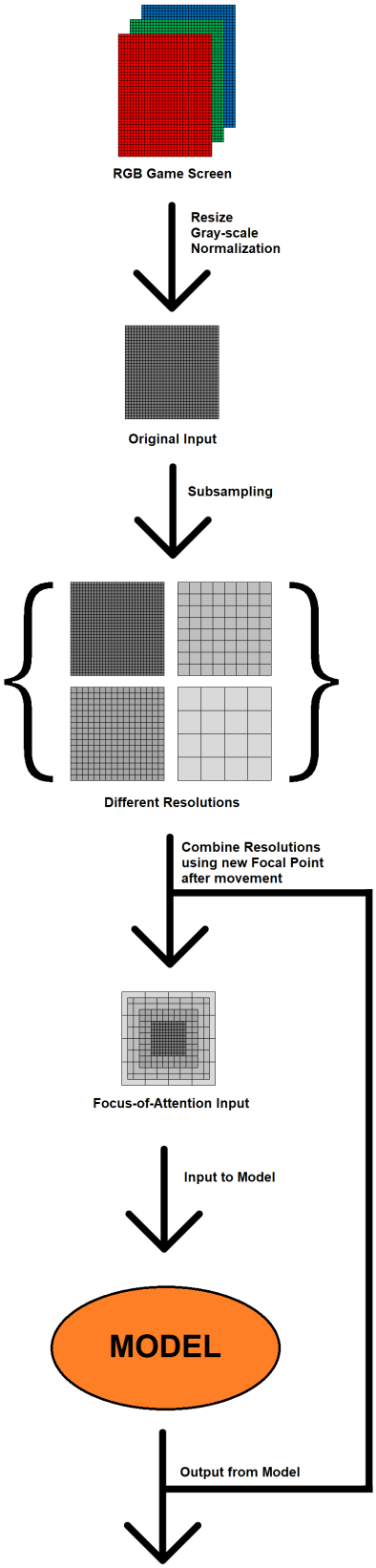


Figure 35: Overview of how a focus-of-attention input is created.

A.2 Pixel-information

When calculating the amount of pixel-information that the peripheral vision adds some low resolution pixels are partially covered by the focal area. Two example calculations can be found below. Note that the blue and the red areas are calculated separately and then added together and multiplied with 2.

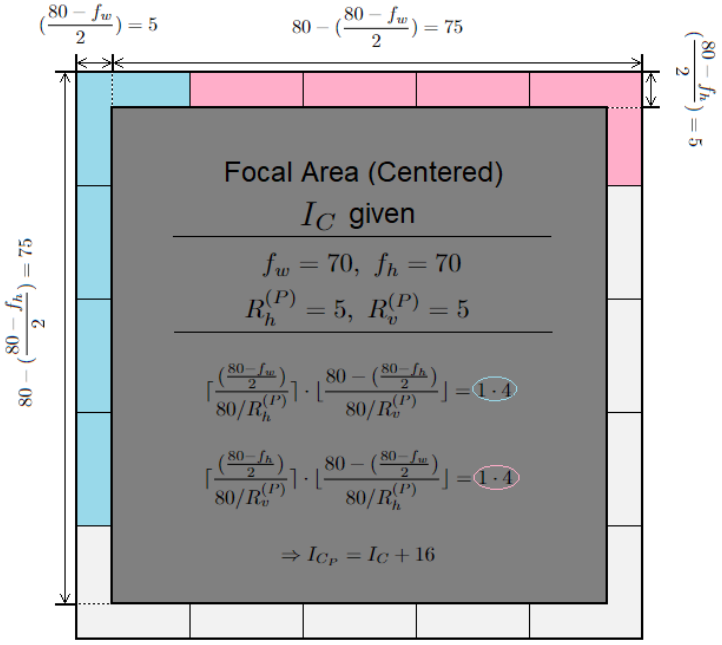


Figure 36: Example calculation of pixel-information for peripheral.

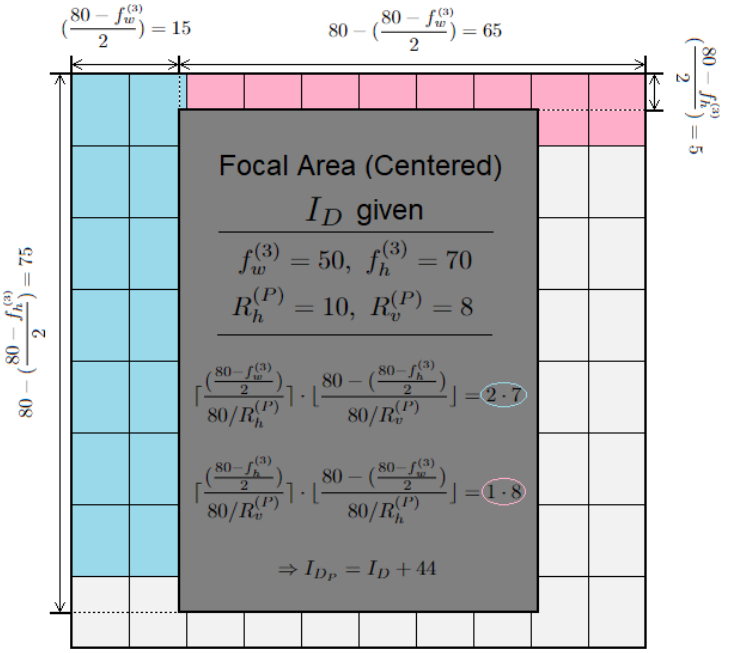


Figure 37: Example calculation of pixel-information for peripheral.

A.3 Hyperparameters

Table 3: Hyperparameters used in the project.

Hyperparameter	**Value**
Atari pre-processing	
Frame size	80×80
Gray-scale	Yes
Cropped images	Yes
Normalized images	Yes
Frame skip	4
Max-pool screens	Yes
Frame stack	No
Initial no-operations	$[0, 30]$
Clip rewards to $\{-1, 0, +1\}$	Yes
Loss of life terminal state	Yes
Fire on reset	Yes
Number of actions	Game specific
Training	
Learning rate α	$1.0 \cdot 10^{-4}$
Optimizer	Shared Adam (AMSGrad)
Action-learner threads	32
Max number of steps, t_{max}	20
Discount factor, γ	0.99
GAE parameter, λ	0.92
Entropy term, β	0.01

References

[1] Bellemare, Marc G.; Naddaf, Yavar; Veness, Joel; Bowling, Michael. *The Arcade Learning Environment: An Evaluation Platform for General Agents*. 2013. https://arxiv.org/abs/1207.4708

[2] Brockman, Greg; Cheung, Vicki; Pettersson, Ludwig; Schneider, Jonas; Schulman, John; Tang, Jie; Zaremba, Wojciech. *OpenAI Gym*. 2016. https://arxiv.org/abs/1606.01540

[3] Hernández, Adrián; Amigó, José M.. *Attention Mechanisms and Their Applications to Complex Systems*. 2021. https://doi.org/10.3390/e23030283

[4] Hochreiter, Sepp; Schmidhuber Jürgen. *Long Short-Term Memory*. 1997. https://www.researchgate.net/publication/13853244_Long_Short-term_Memory

[5] Kapturowski, Steven; Ostrovski, Georg; Quan, John; Munos, Remi; Dabney, Will. *Recurrent Experience Replay in Distributed Reinforcement Learning*. 2019. https://openreview.net/forum?id=r1lyTjAqYX

[6] Kingma, Diederik P.; Ba, Jimmy Lei. *Adam: A Method for Stochastic Optimization*. 2017. https://arxiv.org/abs/1412.6980

[7] Kostrikov, Ilya. *PyTorch Implementations of Asynchronous Advantage Actor Critic*. 2018. https://github.com/ikostrikov/pytorch-a3c

[8] Mnih, Volodymyr; Badia, Adriá Puigdomènech; Mirza, Mehdi; Graves, Alex; Harley, Tim; Lillicrap, Timothy P.; Silver, David; Kavukcuoglu, Koray. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. https://arxiv.org/abs/1602.01783

[9] Mnih, Volodymyr; Kavukcuoglu, Koray; Silver, David; Graves, Alex; Antonoglou, Ioannis; Wierstra, Daan; Riedmiller, Martin. *Playing Atari with Deep Reinforcement Learning*. 2013. https://arxiv.org/abs/1312.5602

[10] National Supercomputer Centre. *Berzelius*. 2022. https://www.nsc.liu.se/systems/berzelius/

[11] Reddi, Sashank J.; Kale, Satyen; Kumar, Sanjiv. *On the convergence of Adam and Beyond*. 2019. https://arxiv.org/abs/1904.09237v1

[12] Schulman, John; Moritz, Philipp; Levine, Sergey; Jordan, Michael I.; Abbeel, Pieter. *High-Dimensional Continuous Control Using Generalized Advantage Estimation*. 2018. https://arxiv.org/abs/1506.02438

[13] Simonyan, Karen; Zisserman, Andrew. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. https://arxiv.org/abs/1409.1556

[14] Öhman, Jim. *Application of Deep Q-learning for Vision Control on Atari Environments*. 2020. http://lup.lub.lu.se/student-papers/record/9041446

Robin Göransson
Lunds Tekniska Högskola

Simulating the visual system of a deep reinforcement learning agent in different ways on Atari games

When training an agent on Atari games not all information on the screen is relevant. In this thesis the agent is tasked with controlling an area of interest that effectively can be used to exclude irrelevant pixels. Different types of interest areas are proposed, some of which perform well with limited information.

In reinforcement learning an agent is tasked with taking actions in such a way that some rewards are maximized. A reinforcement learning agent learns through trial and error; if the agent performs well it receives a positive reward and if the agent performs badly it receives a negative reward. In deep reinforcement learning these ideas have been combined with deep learning and neural networks. This combination allows reinforcement learning to be used on larger environments and on more complex problems, making reinforcement learning more useful and a more popular choice for solving problems.

When it comes to evaluating the performance of deep reinforcement learning agents different Atari 2600 games are often used with the goal of learning to play the game. These games are useful as the problem solving ability of an agent can be tested quite easily on them. The Atari games are, however, fully observable: all the information on the screen is available to the agent at all times. A problem with this is that not all this information is relevant to the agent. In an attempt to exclude irrelevant pixels, the agent was in this thesis given a small region of interest (ROI) which could be moved across the screen. With this the agent had to learn where too look at the same time as it had to learn how to play the game.

Throughout the thesis the agent's "visual system" for controlling the region of interest was simulated in different ways. First, the ROI was created using a simple rectangle containing pixels of one chosen resolution. The pixels outside this rectangle were simply replaced with zeroes. By using ROIs of different sizes and different resolutions it could be shown that a larger ROI with lower resolution was preferable to a smaller ROI with a higher resolution. The agent's field of view is, in other words, more important than the resolution.

Next, a new ROI was created by letting the resolution decrease as the distance to the center of the ROI increased. This type of ROI is inspired by the human visual system. Keeping the size of the ROI fixed while the resolution in the center was increased and the resolution towards the edges was decreased, did not, in general, improve performance. The increased resolution towards the center was probably not enough to make up for the lowered resolution towards the edges of the ROI. It was, however, clear that the low resolution towards the edges did provide important information to the agent.

A final addition to the models used in the thesis was peripheral vision. This addition is also inspired by the human visual system. The agent's peripheral vision was simulated using a very low resolution background which added very little information to the model. When using peripheral vision, information from the whole screen is used, even though the information given by the peripheral is too blurry to make out details. The very low resolution of the peripheral is, however, enough to track movements and detect changes on the screen. This is made clear by the fact that the addition of the peripheral, in general, significantly improved the performance in the games.

Throughout the thesis it has been shown that an agent can perform well while controlling its ROI and also that it is possible to reduce the amount of information available to the agent quite a lot without hurting the performance. The successful agents from this thesis could possibly be modified to work on more complex real-world environments, where the use of a visual system is needed. The different ROIs using a reduced amount of information could also be useful in situations where there are limits to bandwidth or something similar.