

Learning to Perceive in Deep Model-Free Reinforcement Learning

Gonçalo Querido
Instituto Superior Técnico
Lisbon, Portugal
goncalo.querido@tecnico.ulisboa.pt

Alberto Sardinha
INESC-ID & Instituto Superior
Técnico & PUC-Rio
Lisbon, Portugal
jose.alberto.sardinha@tecnico.ulisboa.pt

Francisco S. Melo
INESC-ID & Instituto Superior
Técnico
Lisbon, Portugal
fmelo@inesc-id.pt

ABSTRACT

This work proposes a novel model-free Reinforcement Learning (RL) agent that is able to learn how to complete an unknown task having access to only a part of the input observation. We take inspiration from the concepts of visual attention and active perception that are characteristic of humans and tried to apply them to our agent, creating a hard attention mechanism. In this mechanism, the model decides first which region of the input image it should look at, and only after that it has access to the pixels of that region. Current RL agents do not follow this principle and we have not seen these mechanisms applied to the same purpose as this work. In our architecture, we adapt an existing model called *recurrent attention model* (RAM) and combine it with the *proximal policy optimization* (PPO) algorithm. We investigate whether a model with these characteristics is capable of achieving similar performance to state-of-the-art model-free RL agents that access the full input observation. This analysis is made in two Atari games, Pong and SpaceInvaders, which have a discrete action space, and in CarRacing, which has a continuous action space. Besides assessing its performance, we also analyze the movement of the attention of our model and compare it with what would be an example of the human behavior. Even with such visual limitation, we show that our model matches the performance of PPO+LSTM in two of the three games tested.

KEYWORDS

Reinforcement Learning, Model-Free, Attention Mechanism, Hard Attention, Active Perception, Visual Attention

1 INTRODUCTION

In our everyday lives, even though we are constantly being flooded with visual stimuli, we do not give the same importance to everything in our field of view. Instead, we focus on small regions that attract us the most. In those moments, we take advantage of a cognitive process called *visual attention* [4]. Unconsciously, we interpret those regions and extract meaning from them using another mental process named *perception* [15]. The combination of both these processes allows us to solve complex tasks because, from all the visual information we receive, we filter the most important to perform our activities and not pay attention to irrelevant elements in our surroundings.

Current Reinforcement Learning (RL) models, even though they achieve excellent performance in a broad range of tasks, do not follow this behavior typical of humans. For example, when learning to play a video game, RL algorithms typically process the whole input image, giving the same importance to every region of the

input game frame. Such design results in models that rely on large convolutional neural networks (CNNs) that process a large number of pixels, making the model take too long to train, requiring high computational power, and potentially limiting their applicability [8]. To keep the training time reasonable, images are often preprocessed to reduce the size of the input, losing some of its details. Using these low-resolution images can hamper the models from completely understanding what is present in their input, which lowers their performance [19].

To overcome these limitations, in this paper we contribute the first RL architecture that implements an attention mechanism similar to the one humans have. Applying such a mechanism allows the model to only process the pixels it perceives as the most useful, which makes it much more computationally efficient and able to use the original images without resizing them.

Attention mechanisms such as the one described above recently started appearing in the literature, but none of them were applied to the same purpose as this work. The closest to our work is, perhaps, the model proposed by Mnih et al., called *recurrent attention model* (RAM) [12], which implements the same attention mechanism we use in our work in the context of image classification. The authors introduce the concept of *glimpse*, a retina-like representation of a portion of an image centered around a location l . The region of the image around l has high resolution; regions further away from l have increasingly lower resolution. Such representation is crucial to the performance of the agent and is what makes the complexity of the model not dependent on the size of the input images. Instead, it depends on the size of the glimpses. The RAM paper uses four networks: one responsible for extracting the glimpses from an image and learning features from them; a Long Short Term Memory (LSTM) network that builds an internal representation of the environment combining the information from all the previous glimpses; one network responsible for the classification of the image received, and another that chooses the coordinates of the next glimpse.

There is also a number of other models that apply different types of attention mechanisms. For example, the work of Tang et al. [18] is an example of an RL agent that is capable of achieving top performance in games such as CarRacing [9] and DoomTakeCover [14]. Their agent starts by dividing the input video frame into patches, assigning a level of importance to each one of them. Then, it selects the K most important patches and extracts features from them. This information is later used by a controller which selects the actions to take. Although this model only has around 3600 parameters, since it is trained using a computational intensive evolutionary strategy called Covariance-Matrix Adaptation Evolution Strategy (CMA-ES), it takes a long time to train.

In this paper, we address the following research questions:

- (1) Is it possible to attain state-of-the-art performance in complex control tasks with limited (but active) perception?
- (2) Is there any similarity between the attention movements of the model and human behavior?

To address these questions, the paper proposes a novel architecture that combines a glimpse-based attention mechanism with a model-free reinforcement learning algorithm (PPO). Our results show our model can match the performance of PPO+LSTM in two of the three games tested while processing a significantly smaller number of pixels from the input images. Our model has fewer training parameters than its vanilla counterparts and is, therefore, more efficient than existing models that apply attention mechanisms.

In the remainder of this paper, we go over the modifications we have made to the RAM [12] architecture, showing how our model compares against multiple versions of the PPO algorithm when playing video games such as Pong, SpaceInvaders, and CarRacing. In the end, we analyze the movement of the attention of our model and compare it with what would be an example of human behavior.

2 BACKGROUND

This section introduces the core concepts and notation used in the remainder of the document.

2.1 Active Perception and Attention

The concept of *active perception* was defined by Bajcsy as the intelligent acquisition of information about an environment in order to understand it better [2]. In comparison to a passive perception agent that statically senses its environment and takes actions accordingly, an active perception agent can improve its performance by actively moving its sensors to better reason about its environment [10].

In artificial intelligence, active perception models can be implemented using deep learning methods such as CNNs [10]. Since the selection of what an agent should sense can take inspiration from the human visual attention process, an attention mechanism can be used to replicate such behavior. In machine learning, *attention* is a technique that consists in choosing which parts of the input are the most important, resulting in more computational power being allocated to them. In the literature [6, 20, 21], we can find two categories of attention mechanisms:

- **Soft attention:** is a mechanism that splits the input into multiple parts, assigning a weight to each one of them. The weights represent the importance associated with each portion of the image, and since this model is differentiable, they can be updated using backpropagation.
- **Hard attention:** is a mechanism where the model does not go through some parts of its input because the neural network itself stochastically decides which are the parts it should pay attention to. However, this mechanism is not differentiable but can be trained using RL.

The model from Tang et al. [18] used a soft attention mechanism, while in our work, we use a hard attention mechanism.

2.2 Reinforcement Learning

In RL, an agent interacts with its environment and learns, by trial and error, an action-selection rule (a *policy*) that maximizes the agent’s reward over time.

In our problem, the agent does not have full observability of the environment, so the best mathematical framework to formally represent this problem is a Partially Observable Markov Decision Process (POMDP). A POMDP is defined as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{Z}, \mathcal{P}, \mathcal{O}, r)$, where \mathcal{S} is a discrete set of states, \mathcal{A} is a discrete set of actions, \mathcal{Z} is a discrete set of observations, $\mathcal{P}(s' | s, a)$ represents the transition probability of going from state s to s' performing action a , $\mathcal{O}(z | s', a)$ represent the probability of receiving the observation z , while being in state s' after taking action a , and $r(s, a)$ is the reward function $r : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$.

In our work, we decided to propose a model-free RL architecture instead of a model-based. We did not choose the latter because we decided to understand first if an RL model was capable of having good performance with such visual restriction while having a simpler, model-free approach. Building a complete model of the environment while just seeing a region of it, is another challenge that we leave for future work. Since we made that decision, our agent is not able to know either the transition probabilities \mathcal{P} or the reward function r of the environment. Therefore, it has to learn, explicitly by trial and error, the optimal policy $\pi : \mathcal{H} \rightarrow \Delta(\mathcal{A})$, which is a mapping from the history of past observations to a distribution over actions. One way of learning that policy is using an actor-critic policy-gradient method. This algorithm learns a parameterized policy (*an actor*) and estimates a value function (*a critic*). For our problem, we need two actors: one to learn the action policy $\pi_\theta(a | z)$ (Actor), and the other to learn the policy $\pi_\mu(l | z)$ that chooses the coordinates of the image to look at (Locator).

The specific actor-critic method used in this work was the PPO algorithm, presented by Schulman et al. [17]. PPO is simpler than other policy gradient methods, well-studied, and was already tried alongside the RAM architecture [22]. For these reasons, we decided to have it as the base of our model.

In PPO, we alternate between interacting with the environment to get data and updating the policy using stochastic gradient ascent. In every policy update, this policy gradient method guarantees that the difference between the new policy and the old policy is small, which prevents the algorithm from having a high variance during training. Having the probability ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t | s_t)}{\pi_{\theta_{\text{old}}}(a_t | s_t)} \quad (1)$$

and \hat{A}_t , an estimator of the advantage function at timestep t , PPO maximizes the following surrogate objective:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)\hat{A}_t) \right] \quad (2)$$

where ϵ is a hyperparameter. The clipping prevents large policy updates and penalizes the probability ratio $r_t(\theta)$ when it tries to move far away from 1.

3 GLIMPSE-BASED ACTOR-CRITIC (GBAC)

In this section, we introduce a novel model called Glimpse-Based Actor-Critic (GBAC) that combines a hard attention mechanism with a model-free RL algorithm. When compared to other RL models, GBAC processes much fewer pixels, and its training parameters do not depend on the size of the input, which makes it more efficient.

In our problem, the game environment E gives, at each timestep, a frame s_t of the game. Since our model cannot have access to all the information of s_t , it has to select only a portion of it to be its observation z_t . That observation has the coordinates $l_{t-1} = (x_{t-1}, y_{t-1})$ that were chosen by the model in the previous timestep. During its interaction with E , our agent has to learn the best policy $\pi_\theta(a_t | z_t)$ that selects the action a_t to be performed in the game. While doing this, the model also has to understand which regions of s_t have the most valuable information and learn a policy $\pi_\mu(l_t | z_t)$ to choose the set of coordinates (x_t, y_t) to be taken in the next timestep.

As we have seen in Section 2.2, this problem can be seen as an instance of a POMDP. In our case, the observations the agent takes are represented by the glimpses, and the history of past interactions with the environment is stored in the hidden state of two LSTMs. When combining the memory, the current glimpse, and the previously chosen location, the LSTMs have all the information needed to learn the policies that select the actions and the locations to take next.

3.1 Architecture

The architecture of RAM [12] was the basis for our proposal, as a result, we present it in Figure 1 and will briefly describe it next.

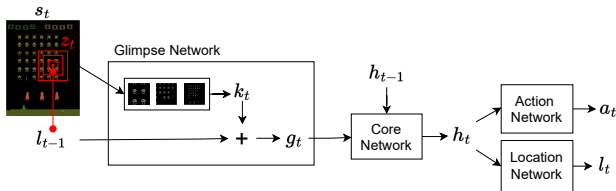


Figure 1: Architecture of RAM. Image adapted from the original paper of Mnih et al. [12]

Every timestep, RAM receives the game frame s_t and a set of coordinates l_{t-1} . Its Glimpse Network takes a glimpse z_t centered at l_{t-1} and extracts features, not only from z_t , which are represented in Figure 1 by k_t , but also from l_{t-1} . Using fully connected layers the two features are merged into the vector g_t . Next, this vector is fed to an LSTM, the Core Network, which stores all the previous information from the glimpses and the coordinates chosen, building its internal memory h_t . After that, h_t is the input for two other networks, the Action Network and the Location Network, which are also fully connected layers that select the next action a_t and coordinates l_t , respectively. Since the Location Network was non-differentiable, it was trained using the policy gradient method REINFORCE, while the other components used backpropagation.

After the presentation of RAM, some papers proposing improvements to the image classification capabilities of the model were written, such as the publications of Ba et al. [1] and Zuur [22]. After the presentation of our architecture, we will describe which refinements we took into consideration and how we have adapted them to our problem.

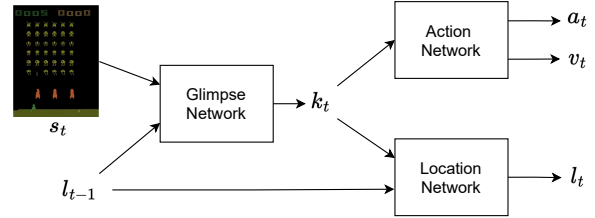


Figure 2: Overview of the architecture of the Glimpse-Based Actor-Critic

Figure 2 presents an overview of the GBAC architecture. We start by receiving a game frame s_t and the set of coordinates l_{t-1} our agent chose at the end of the previous timestep. The Glimpse Network saves into k_t the features extracted just from the glimpse taken from s_t and centered at l_{t-1} . After that, the vector k_t is used as the input of the Action Network. This network outputs not only the action a_t the agent will take next, but also an estimate v_t of the value function. k_t is also used by the Location Network, which merges it with the features extracted from the location l_{t-1} to select the next location coordinates l_t . In opposition to RAM, instead of doing this merging for the input of both networks, we only do it for the Location Network.

We now present the key refinements of RAM that we took into consideration to address our problem. To start, we followed Ba et al. [1] and added a CNN to the Glimpse Network (represented in Figure 3) because it was necessary for the model to extract better features from the video frames. As one of the experiments done by Zuur [22] suggested, we also found that it was beneficial to provide separate inputs for the Action and Location Networks (Figure 2). Therefore, the configuration which performed best was the one that fed to the Action Network only the features (k_t) extracted from the glimpse, and to the Location Network the vector resultant from the merge between k_t and the features extracted from l_{t-1} . Still with this idea of separating the processes of choosing the next action and the next glimpse coordinates, we added an extra LSTM, making the Action Network and the Location Network use their separate LSTM (Figures 4 and 5). This separation destroyed the need for having an explicit Core Network like in RAM, avoiding the mix of information that is necessary for each task and allowing us to fine-tune the parameters for each LSTM. The last change we made was the selection of PPO instead of REINFORCE to train the model since it achieves better results in harder environments [22] and we found it simpler to train. This change meant that in each timestep, our Action Network also needed to make a prediction v_t about the quality of performing the action a_t in the current state of the environment.

3.1.1 Glimpse Network. The Glimpse Network, represented in Figure 3, is the module responsible for extracting from the game frame,

the region the agent chose to focus its attention. With the image coordinates l_{t-1} chosen in the previous timestep, this network extracts an observation z_t from s_t , which is called a *glimpse*. This glimpse can have multiple patches, each having double the size of the previous. For example, Figure 3 presents a glimpse with three patches. However, to simulate peripheral vision, all the larger patches are downsampled to the dimension of the smallest. That smallest patch will have the highest resolution, making it the focal point. Regarding the other patches, the larger they are, the further away they are from the focus point, so the lower their resolution is. When compared to the original image, this process results in a vector with fewer pixels.

After rescaling, the resultant vector passes through a set of convolutional layers and a fully connected layer to extract a vector of features k_t . The number of convolutional layers and their respective kernel sizes and stride are changed depending on the size of the glimpse.

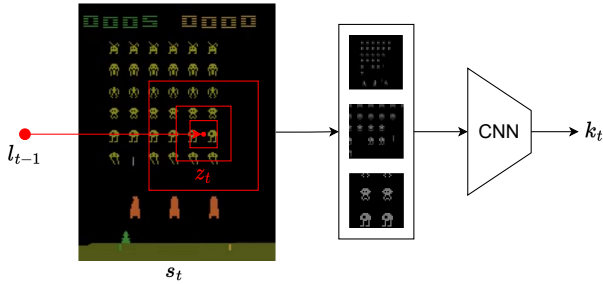


Figure 3: Detailed diagram of the Glimpse Network of GBAC

During this work, we assumed that the glimpses are always squares. If the coordinates of l_t make a patch catch pixels that are out of the bounds of s_t , that patch will be moved in order to fit inside the frame. This mechanism proved to achieve better results than simply filling the pixels out of bounds with a value.

3.1.2 Action Network. The Action Network, depicted in Figure 4, is responsible to choose the game action a_t the agent should perform in each timestep. Since we chose an actor-critic algorithm to train GBAC, the Action Network also estimates the value function, which helps the agent to understand if it is performing well or not.

In this network, its LSTM saves the information z_t extracted previously from the glimpse and combines it with its hidden memory h_{t-1}^g , which stores all the information gathered in earlier timesteps to build an internal representation of the game environment. The new hidden state h_t^g is fed to two fully connected layers, the Actor and the Critic, that output the action a_t and the value v_t , respectively.

3.1.3 Location Network. The Location Network, illustrated in Figure 5, is the module behind the hard attention mechanism and is responsible to choose the image coordinates l_t where the agent should look in the next timestep. Those coordinates are sampled from a truncated normal distribution whose mean is given by this network, being the standard deviation a fixed value.

In order to choose the mean value, the Location Network has a neural network that extracts features from the coordinates l_{t-1} ,

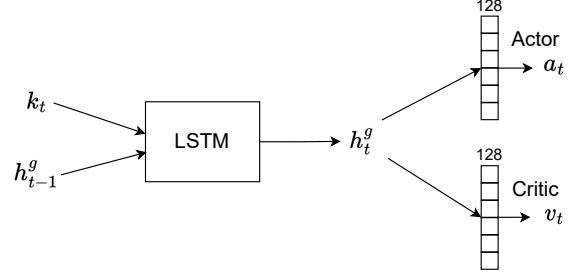


Figure 4: Detailed diagram of the Action Network of GBAC

merging them with the features k_t . The resultant vector g_t is then used to update the internal state of an LSTM. Its internal state h_t^l is fed to the Locator, which is a neural network with two fully connected layers and ReLU and Tanh activation functions, respectively. The Locator chooses the mean value used in the truncated normal distribution from which the next set of coordinates l_t is extracted.

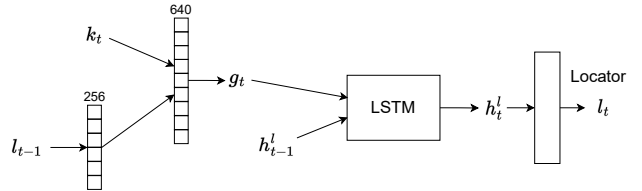


Figure 5: Detailed diagram of the Location Network of GBAC

Since this model only uses a small portion of an image, its complexity is not dependent on the size of the input image but rather on the size of its glimpses. This can be an advantage if the model has to handle large images.

3.2 Training

The training process of our model is very similar to the one presented by Schulman et al. in the PPO paper [17]. We follow the suggestions they proposed, and the only difference is that we have two policy losses, instead of just one. Like PPO, our model alternates between interacting with the environment to get new information and updating both its policies with that new data.

When exploring the environment, in each timestep, the model saves the action and coordinates of the glimpse it chose, the value function estimated by the critic, the reward received from the game, and a flag that indicates if the current episode has finished.

After a predetermined number of timesteps, the model uses the collected data to update its policies. For both policies, we use the "surrogate" objective already presented in Section 2.2:

$$L^{CLIP\pi}(\theta) = \hat{\mathbb{E}}_t \left[\min(r_t^\pi(\theta) \hat{A}_t, \text{clip}(r_t^\pi(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t) \right] \quad (3)$$

where ϵ is a hyperparameter and the advantage function \hat{A}_t is computed recurring to Generalized Advantage Estimation [16].

Besides the two policy losses, the objective has two more terms: an entropy bonus B that promotes the exploration of the environment, and the squared-error loss of the value function, L_t^{VF} . With these additions, the objective’s formula is the following:

$$L_t(\theta) = \hat{\mathbb{E}}_t \left[L_t^{CLIP_a}(\theta) + L_t^{CLIP_g}(\theta) - \alpha L_t^{VF}(\theta) + \beta B[\pi_a](s_t) \right] \quad (4)$$

where α and β are coefficients. Note that the entropy bonus is only calculated for the action policy, not for both policies. Adding the same bonus for the location policy did not seem to improve the results, thus we decided to keep the objective simpler.

4 EXPERIMENTAL EVALUATION

In this chapter, we present all the experiments that enabled us to test our model, establishing which base PPO algorithm is the fairest choice to compare our model with, and discovering which size of the glimpses gives better results. In the end, we have all the information necessary to answer the questions in Section 1. Our code is available on GitHub.

4.1 Evaluation Process

4.1.1 Game Environments. In order to measure the performance of GBAC and compare it against the state-of-the-art RL agents, we selected three game environments with different characteristics. The first two are both games from the Atari 2600 and have a discrete action space, while the third, is CarRacing [9] from OpenAI Gym [3] and has a continuous action space. We tried to select three games that were not the easiest ones available and that required the agents to learn different sets of skills, such that we could see how they were capable to adapt to each type of task.

In order to obtain the best performance in the Atari games, we used the same environment modifications proposed by Mnih et al. [13] and Machado et al. [11], thus leading to better results for these games. Those modifications are well accepted and used in the literature. For example, we resize the original frame from 210x160 to 84x84 pixels, clip the rewards, and scale the pixel values to $[0, 1]$.

4.1.2 Comparison Models. We decided to compare GBAC with three different versions of PPO. The first one is the original PPO agent presented by Schulman et al. [17] because it was used as the base for our model. However, since our implementation uses LSTMs in its architecture and the version of PPO with an LSTM is also quite common in the RL literature, we decided to choose the PPO+LSTM agent for comparison as well. The third version of PPO we used is a modification of the PPO+LSTM algorithm where the restriction of only using a small portion of the game frame was imposed. But, different from GBAC, the coordinates where the agent looks are chosen completely randomly. This allows us to test if the perception mechanism implemented in our model is better than one that makes its choices randomly.

The PPO implementation used in this work was not the original one provided by Schulman et al. [17] in OpenAI Baselines [5], but rather a revised implementation presented by Shengyi et al. [7] that closely follows the performance of the original. This implementation provides many versions of PPO including one with an LSTM. To make the PPO agents capable of playing the selected games, we

added to their architecture a CNN with the same layout as the one presented in the DQN paper [13].

By comparing GBAC with the first two versions of PPO, we can discover if it is possible to achieve state-of-the-art performance playing video games, despite having a limited (but active) perception of the environment, which was our first question.

4.1.3 Evaluation Metrics. Evaluating RL models is never a straightforward task due to their high variance. For this reason, during training and testing, we average the episodic return each agent achieved over the last 100 episodes.

In training, Pong, SpaceInvaders, and CarRacing learned during 15 million, 20 million, and 5 million timesteps, respectively. Then, the agent that achieved the best performance over the last 100 episodes is tested for another 100 episodes. For each configuration, the results are always the average over three runs and their respective standard deviations are also presented.

Seeding is another aspect that can have an impact on the performance of the agent. Certain seeds can make the agent perform significantly better or worse. Thus, in each run, the seed used in the environment is chosen arbitrarily.

Regarding the policy that chooses the locations of the glimpses, in order to understand if our model learns a behavior that resembles the human vision, we analyze the evolution of the policy throughout the training phase and present a visual representation of the results. This is the information we need to answer our second question.

Now that the entire evaluation process is detailed, we present, in the next sections, the results we obtained.

4.2 Base Models Selection

In contrast to one of the Atari optimizations proposed by Mnih et al. [13] and Machado et al. [11], our model does not perform better when the original image is resized. In fact, it never learned a way to beat its adversary in Pong when a glimpse smaller than the resized size of 84x84 was used. This meant that with that setup, we could not take advantage of the glimpse’s structure because the model needed the entire image to perform well, which does not follow the restriction of our problem. An explanation for this result can be the fact that since the glimpses taken by GBAC have multiple patches that end up being resized to a lower resolution, and the input frame fed to the model was also resized, the loss in information might be so much that our agent is not capable of solving the game.

Therefore, to make the comparisons fair, we decided that the base PPO models should also use the entire image as input. In order to discover how much this decision could impact the performance of the base agents when playing the two Atari games, we studied the difference in performance between using the original 210x160 image and the resized 84x84 input. In CarRacing, since the original image is just 96x96, we decided not to compare the base models with a resized version of the input. In addition, since GBAC uses LSTMs, we compared PPO with PPO+LSTM to find out if they perform any differently.

In Table 1, we show the training and testing performances that PPO and PPO+LSTM had when using either the full image frame or the resized input. The results shown do not allow us to conclude with absolute certainty that one way is better than the other. In Pong, there are not any significant differences in performance either

Model	Full Img.	PongNoFrameskip-v4		SpaceInvadersNoFrameskip-v4		CarRacing-v0	
		Max. Train Avg.	Test Avg.	Max. Train Avg.	Test Avg.	Max. Train Avg.	Test Avg.
PPO	No	21.00 ± 0.01	20.98 ± 0.02	2090.00 ± 254.16	2013.07 ± 244.56	-	-
PPO	Yes	20.91 ± 0.11	20.83 ± 0.13	2261.90 ± 295.87	2221.62 ± 201.31	867.16 ± 6.64	824.31 ± 8.04
PPO + LSTM	No	20.11 ± 0.25	20.00 ± 0.31	1182.57 ± 259.03	1077.63 ± 245.82	-	-
PPO + LSTM	Yes	20.03 ± 0.23	19.85 ± 0.39	900.20 ± 79.16	812.58 ± 111.51	783.62 ± 11.58	659.73 ± 24.42

Table 1: Comparison between the training and testing performance of PPO and PPO+LSTM, using a resized frame (84x84) of the input and also the full game frame (210x160)

in regular PPO or in PPO+LSTM. In SpaceInvaders, for the regular PPO, the agent that uses the full image achieves average returns that are around 200 points better than the agent that resizes the input. This result indicates that, for this game, some useful information is lost during the resizing of the image. However, for the model that uses PPO+LSTM, the opposite is verified. Since the size of the LSTMs was the same in both cases, this suggests that for the full image, the LSTM needed to be bigger because it could extract better data from fewer pixels.

From the same table, when comparing the PPO agent against PPO+LSTM for the same type of input, we can see that the use of an LSTM deteriorates the performance of the agent. Having in mind that in the PPO game, a match ends when a player reaches 21 points and the reward of the agent is the difference between the points scored and scored against, the difference between the two models is marginal. It only means that, on average, the PPO+LSTM agent let the opponent score one point, while PPO did not. In SpaceInvaders, the performance drops by half, which is a significant reduction. In CarRacing, there is also a drop in performance, even though it is less accentuated than in SpaceInvaders. The major difference between the PPO and PPO+LSTM models is that the former uses a stack of four frames as input, while the latter receives just one frame at a time, counting on its LSTM to discover and store the information that is useful to the agent. Therefore, in SpaceInvaders and in CarRacing, the LSTM is not able to store all the information needed, like the velocity and direction of the objects from the game, which would allow the agent to perform better.

In short, from these results, we cannot conclude that using the resized frame is better than using the full frame, and since our model utilizes the full image, in order to make the comparisons fairer, with as many equal variables as possible, from here onwards, we will be referring to the version that uses an LSTM and the entire frame as input when mentioning the base model.

4.3 GBAC Performance Analysis

In this section, we study not only the impact that different sizes of glimpses and different numbers of patches have on the performance of our agent but also how the best configuration performs against the three versions of PPO.

In our architecture, each glimpse can have one or more patches. Since we stipulated that each new patch has double the size of the previous, increasing the number of patches results in glimpses with a smaller focus region. This means that if we want glimpses with two patches, the largest possible size for the smallest patch is 80x80, with three patches it will be 40x40, and with four patches 20x20.

The higher the number of patches, the larger the "peripheral vision" of our model. Nonetheless, this increase in information comes with the price of it not being as detailed as the portions of the image closer to the focal point.

With this in mind, besides their architectures, using glimpses with just one patch makes our model no different from the base PPO agents. Therefore, the results from the agents that use glimpses with one patch are just useful to compare the two architectures, and to discover how large the input image has to be, in order for the agent to maintain its performance. Therefore, the results that are relevant to understand the performance of our model are the ones given by configurations that use more than one patch.

4.3.1 Pong. In relation to the Atari game Pong, Table 2 presents the maximum training average and the testing performance of the three PPO versions, as well as the best glimpse size for each number of patches of our agent. Since the returns for this game are bounded between -21 and +21, when analyzing the performance of all the agents, we can see that one of three scenarios occurred: the agent learned how to beat its adversary and ended up with scores close to +20; the agent did not manage to learn anything useful, resulting in scores around -19; or the timesteps were not enough for the agent to learn a good policy so it achieved a score between the previous two.

Model	Glimpse	PongNoFrameskip-v4	
		Max. Train Avg.	Test Avg.
PPO	-	20.91 ± 0.11	20.83 ± 0.13
PPO+LSTM	-	20.03 ± 0.23	19.85 ± 0.39
GBAC	1p, 160x160	7.61 ± 10.42	6.95 ± 10.89
GBAC	2p, 80x80	7.15 ± 20.80	6.64 ± 21.14
GBAC	3p, 40x40	20.06 ± 0.44	19.82 ± 0.88
GBAC	4p, 20x20	-14.86 ± 5.39	-15.77 ± 4.82
PPO Random	3p, 40x40	-19.87 ± 0.05	-20.16 ± 0.11

Table 2: Training and testing performance in Pong

Regarding the glimpses with one patch, our agent did not manage to achieve scores near +20 with the larger glimpse consistently because, in two of the three runs, the agent was still improving its performance when the timesteps finished. Therefore in this game, our agent was not capable of matching the performances of PPO. With two patches, the results were slightly better than the previous because the standard deviation is much higher. The model achieved a score of around +19 in two of the three runs. Using three patches, GBAC was capable of matching the performance of the PPO+LSTM

agent, consistently beating its opponent by 21-1. The difference to the regular PPO agent, only means that our agents let the opponent score a point, while the other did not. After seeing the results until this point, we might think that the performance keeps improving while we increase the number of patches of each glimpse. However, this is not the case when we look at the scores achieved using four patches. The size of the patches was so small that our model was not capable of achieving a performance of +20 in, at least, one of the three runs, which was true in any of the previous results.

Regarding the PPO agent that took glimpses at random locations, we see that it performed very poorly, not being able to learn an optimal policy to play Pong. Therefore, we can say that, in this game, choosing good glimpse locations matters.

4.3.2 *SpaceInvaders*. Table 3 shows the training and testing results of all the agents for the SpaceInvaders game.

SpaceInvadersNoFrameskip-v4			
Model	Glimpse	Max. Train Avg.	Test Avg.
PPO	-	2261.90 ± 295.87	2221.62 ± 201.31
PPO+LSTM	-	900.20 ± 79.16	812.58 ± 111.51
GBAC	1p, 160x160	741.45 ± 392.54	607.42 ± 287.84
GBAC	2p, 80x80	444.18 ± 40.78	341.47 ± 25.71
GBAC	3p, 40x40	596.50 ± 182.35	544.43 ± 166.79
GBAC	4p, 20x20	439.72 ± 26.27	378.00 ± 28.70
PPO Random	3p, 40x40	516.62 ± 60.59	467.38 ± 50.53

Table 3: Training and testing performance in SpaceInvaders

Starting with the results of our model for one patch, we found that this time, the average score of GBAC was closer to the one achieved by the PPO+LSTM agent. However, when considering the results for the glimpses with two patches, the performance dropped almost by half. With three patches, GBAC achieves the best performance when considering the use of more than one patch. Nonetheless, the model is not able to match the performance achieved when it used the bigger glimpse with just one patch. With four patches, we have the same decrease in performance, already seen in Pong. However, this time it was not as severe and slightly beat the results from two patches while having the smallest standard deviation in both training and testing of any of the experiments.

In this game, our model was not capable of matching the performance of the PPO+LSTM agent, however, we still consider it an interesting result, considering the viewing restrictions of our problem. While processing 86% fewer pixels than PPO+LSTM (4.800 vs. 33.600) in each timestep, GBAC only had a performance drop of 33%.

In this game, the random agent has a performance that is on par with our model, which possibly means that in SpaceInvaders the location of a glimpse is not that important. One possible reason behind this proximity could be the fact that, in this game, GBAC can pay attention to a lot more things that can improve the return received from the environment. As opposed to Pong, where our agent only had three objects (two paddles and one ball) to keep track of.

4.3.3 *CarRacing*. Lastly, Table 4 shows the performance that GBAC and the other agents achieved during training and testing, when playing the CarRacing game, which has a game frame of size 96x96.

CarRacing-v0			
Model	Glimpse	Max. Train Avg.	Test Avg.
PPO	-	867.16 ± 6.64	824.31 ± 8.04
PPO+LSTM	-	783.62 ± 11.58	659.73 ± 24.42
GBAC	1p, 96x96	815.12 ± 5.75	660.02 ± 69.38
GBAC	2p, 40x40	694.50 ± 107.94	641.11 ± 57.42
GBAC	3p, 20x20	676.82 ± 84.89	564.00 ± 56.42
PPO Random	2p, 40x40	622.41 ± 17.89	589.87 ± 13.19

Table 4: Training and testing performance in CarRacing

Like in the other two games, here, the largest glimpse size is also the one that achieves the better result for a glimpse with one patch. In addition, this time our model was capable of matching the performance of the PPO+LSTM agent in testing and beating it during training. For two patches, the achieved results maintained the performance seen in testing when using one patch, even though the score in training decreased and had a much higher standard deviation. With three patches, which in CarRacing was the maximum number we tested, we start to see the performance drop slightly.

In this game, GBAC was capable of maintaining its performance across all number of patches, which is a good result since the scores closely match the PPO+LSTM agent.

Regarding the PPO agent with random glimpses, the performance difference is again not that far behind our model. Since the generated track occupies a good portion of the image, it may be easier for the random model to select good actions from every glimpse it receives.

From this study, we can conclude that even when using the entire frame (glimpses with one patch) the performance of our model is capable of matching the performance of PPO+LSTM, although not consistently. This shows our architecture could compete with PPO+LSTM if we did not impose our restrictions. Additionally, we discovered that the performance of GBAC does not increase linearly with the number of patches. It reaches a point where the information lost with the reduction of the glimpse size is more significant than the information gained with the addition of another patch. The optimal number of patches is three for the Atari games and two for CarRacing. Those numbers of patches proved to be the right balance between having a patch size that discarded the irrelevant information, allowing the model to just focus on the most important, and not being too small such that after rescaling the large patches, it was still possible to understand what was present in the "peripheral vision" of the agent. Finally, we saw that in most cases, the largest glimpse size possible for each number of patches is the one that produces the best results. In CarRacing, for two and three patches this was not the case, and the sizes 48x48 and 24x24, respectively, did not give the best results, even though they were very close.

An important fact we should mention is that, since the complexity of our model is independent of the size of the input, we can achieve these results with a model that, in the Atari games,

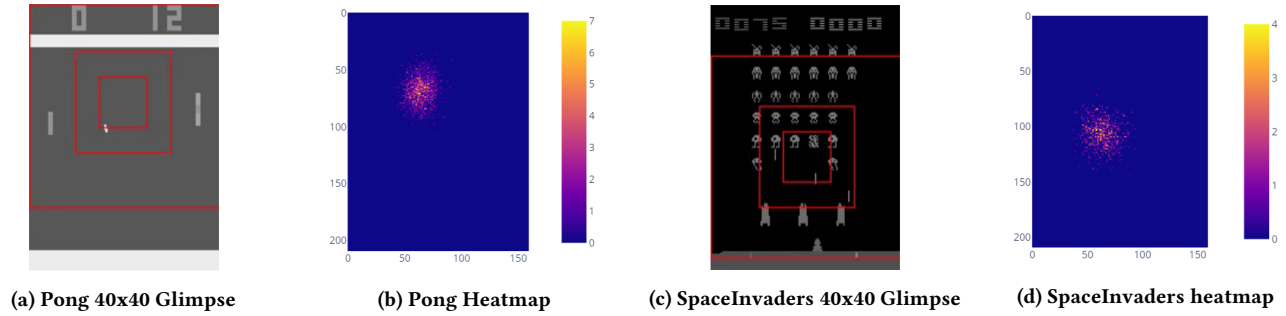


Figure 6: Frames with glimpses of 3 patches and heatmaps representing the number of times during an episode that the agent chose a specific set of image coordinates to be the center of the glimpse, for the Pong and SpaceInvaders games respectively

has 15% fewer total training parameters ($\sim 1.7M$ vs. $\sim 2.0M$) than the PPO+LSTM model, and almost the same number as PPO. In CarRacing, since the actions are continuous, the model is bigger, but it still has 10% fewer parameters than PPO+LSTM ($\sim 2.2M$ vs. $\sim 2.5M$) and only more 70k than PPO. If we use bigger environments having frames with many more pixels, this difference between the number of training parameters required will only keep increasing.

4.4 Glimpse Movements Analysis

After discovering how GBAC performs against the other models, it is also important to understand how the location of its glimpses is evolving throughout training and which behavior the model finds out to be the best. While making this analysis, we also compare the agent’s decisions with the choices we would consider when playing the games.

Regarding the regions that our best agents chose to look at during the episodes, we can see in Figure 6b and Figure 6d that, for the Atari games, their distribution is relatively similar in both games. In SpaceInvaders, our agent keeps its focus near the center of the image, while in Pong, it chooses locations slightly upwards from the center. In general, the distribution of locations in both games has more choices closer to the center and becomes more sparse the further they are from it.

Having the focus point almost near the center of the image, as we can see in Figure 6b, means that, in Pong, the agent gets the location of each paddle from its "peripheral vision" (Figure 6a). We think that this choice is different from what a human would select to focus on in this particular game. We believe that a human would pay more attention to the position of its own paddle and the ball.

In relation to SpaceInvaders, in our opinion, the choices are much closer to what a human would do because the agent keeps its focus on the lower rows of enemies (Figure 6c), which are the ones that need to be destroyed first.

Regarding Car Racing, our model presents quite unusual behavior during the training process. It starts similarly to the other two games with a circular normal distribution for the location coordinates (Figure 7a). However, after a few training epochs, that distribution starts dispersing over multiple parts of the entire image, ending up with the region shown in Figure 7b, creating some kind of borders, which constrain the choice of coordinates, and that do

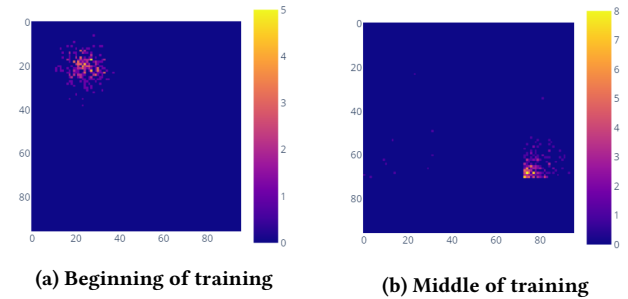


Figure 7: Heatmaps representing the evolution of glimpse locations in the CarRacing game

not correspond to the limits of the image. This last behavior is the one that is present in the best solution at the end of training.

In our opinion, even though in two of the three games, our model does not follow exactly what we think is the human vision behavior when playing those video games, we would need a more systematic way of analyzing the regions that we select to focus our attention, using, for example, an eye-tracking device, to better compare the agent’s behavior in relation to humans.

5 CONCLUSION

This work proposed a solution for the problem of an agent that has limited vision, and for that reason, besides deciding which action it has to take in the environment, it also has to choose which part of the environment it should look at. To solve this problem, we proposed GBAC, a model that combines a glimpse-based hard attention mechanism with a model-free RL algorithm.

We started by proving that, for some games like Pong and CarRacing, our model is already capable of achieving similar performance to the PPO version that more closely resembles our model, that is, the variant that also uses an LSTM. On other games like SpaceInvaders, a drop in performance is verified, with means, there still is room for improvement. We finished concluding that our model does not necessarily choose the same regions of the image that we selected to look at when we played the video games ourselves. Only in SpaceInvaders we considered this was verified. In addition, in

CarRacing, we are not able to explain the reason behind the unusual behavior of our model.

ACKNOWLEDGMENTS

This work was partially supported by Portuguese national funds through Fundação para a Ciência e a Tecnologia (FCT) under projects UIDB/50021/2020 (INESC-ID multi-annual funding), PTDC/CCI-COM/5060/2021 (RELEvaNT), PTDC/CCI-COM/7203/2020 (HOTSPOT). In addition, this research was partially supported by the Air Force Office of Scientific Research under award number FA9550-22-1-0475 and an EU Horizon 2020 project (TAILOR) under GA No. 952215.

REFERENCES

- [1] Jimmy Ba, Volodymyr Mnih, and Koray Kavukcuoglu. 2015. Multiple Object Recognition with Visual Attention. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.
- [2] Ruzena Bajcsy. 1988. Active perception. *Proc. IEEE* 76, 8 (1988), 966–1005.
- [3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. OpenAI Gym.
- [4] Charles E. Connor, Howard E. Egeth, and Steven Yantis. 2004. Visual Attention: Bottom-Up Versus Top-Down. *Current Biology* 14, 19 (2004), R850–R852.
- [5] Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, Yuhuai Wu, and Peter Zhokhov. 2017. OpenAI Baselines. <https://github.com/openai/baselines>.
- [6] Saman Ghaffarian, João Valente, Mariska van der Voort, and Bedir Tekinerdogan. 2021. Effect of Attention Mechanism in Deep Learning-Based Remote Sensing Image Processing: A Systematic Literature Review. *Remote Sensing* 13, 15 (2021).
- [7] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto, and Weixun Wang. 2022. The 37 Implementation Details of Proximal Policy Optimization. In *ICLR Blog Track*. <https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/>
- [8] Angelos Katharopoulos and Francois Fleuret. 2019. Processing Megapixel Images with Deep Attention-Sampling Models. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 97)*. PMLR, 3282–3291.
- [9] Oleg Klimov. 2016. *CarRacing-v0*. <https://gym.openai.com/envs/CarRacing-v0/>
- [10] Elijah S. Lee. 2021. Active Perception with Neural Networks. *arXiv preprint arXiv:2109.02744* (9 2021). <https://doi.org/10.48550/arxiv.2109.02744>
- [11] Marlos C. Machado, Marc G. Bellemare, Erik Talvitie, Joel Veness, Matthew Hausknecht, and Michael Bowling. 2018. Revisiting the Arcade Learning Environment: Evaluation Protocols and Open Problems for General Agents. *Journal of Artificial Intelligence Research* 61, 1 (jan 2018), 523–562.
- [12] Volodymyr Mnih, Nicolas Heess, Alex Graves, and Koray Kavukcuoglu. 2014. Recurrent Models of Visual Attention. In *Proceedings of the 27th International Conference on Neural Information Processing Systems (Montreal, Canada) (NIPS'14, Vol. 2)*. MIT Press, Cambridge, MA, USA, 2204–2212.
- [13] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fiedjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dhharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
- [14] Philip Paquette. 2016. *DoomTakeCover-v0*. <https://gym.openai.com/envs/DoomTakeCover-v0/>
- [15] Daniel L. Schacter, Daniel Todd Gilbert, Daniel M. Wegner, and Bruce M. Hood. 2016. *Psychology* (2nd european ed.). Palgrave. 133 pages.
- [16] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and Pieter Abbeel. 2015. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings* (6 2015).
- [17] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. *arXiv preprint arXiv:1707.06347* (7 2017). <https://doi.org/10.48550/arxiv.1707.06347>
- [18] Yujin Tang, Duong Nguyen, and David Ha. 2020. Neuroevolution of Self-Interpretable Agents. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (Cancún, Mexico) (GECCO '20)*. Association for Computing Machinery, New York, NY, USA, 414–424.
- [19] Vajira Thambawita, Inga Strümke, Steven A. Hicks, Pål Halvorsen, Sravanthi Parasa, and Michael A. Riegler. 2021. Impact of Image Resolution on Deep Learning Performance in Endoscopy Image Classification: An Experimental Study Using a Large Dataset of Endoscopic Images. *Diagnostics* 11, 12 (2021).
- [20] Feng Wang and David M. J. Tax. 2016. Survey on the attention based RNN model and its applications in computer vision. *arXiv preprint arXiv:1601.06823* (1 2016). <https://doi.org/10.48550/arxiv.1601.06823>
- [21] Xiao Yang. 2020. An Overview of the Attention Mechanisms in Computer Vision. *Journal of Physics: Conference Series* 1693, 1 (dec 2020), 012173.
- [22] Marcel Zuur. 2019. *Deep Reinforcement Learning of Active Sensing Strategies with POMDPs*. Master's thesis. Radboud University - Faculteit der Sociale Wetenschappen.

A EXTENDED RESULTS

No. Patches	Glimpse Size	PongNoFrameskip-v4		SpaceInvadersNoFrameskip-v4	
		Max. Train Avg.	Test Avg.	Max. Train Avg.	Test Avg.
1	160	7.61 ± 10.42	6.95 ± 10.89	741.45 ± 392.54	607.42 ± 287.84
1	150	-18.96 ± 1.67	-19.32 ± 1.38	384.43 ± 72.19	338.45 ± 59.50
1	140	-18.93 ± 1.80	-19.36 ± 1.47	386.50 ± 75.94	347.08 ± 99.27
1	130	-19.92 ± 0.04	-20.19 ± 0.03	330.76 ± 77.16	265.00 ± 53.71
1	120	-19.86 ± 0.04	-20.06 ± 0.05	343.98 ± 137.35	273.00 ± 113.99
1	110	-19.89 ± 0.11	-20.22 ± 0.08	381.66 ± 85.91	340.12 ± 77.90
1	100	-19.89 ± 0.05	-20.24 ± 0.11	321.91 ± 51.51	256.78 ± 77.16
1	90	-19.65 ± 0.42	-19.90 ± 0.36	285.76 ± 61.01	214.35 ± 23.69
1	80	-19.90 ± 0.01	-20.28 ± 0.05	244.53 ± 7.94	190.83 ± 26.97
1	70	-18.47 ± 1.30	-18.88 ± 1.08	243.08 ± 23.85	186.83 ± 19.25
1	60	-19.89 ± 0.05	-20.16 ± 0.09	232.95 ± 17.10	214.58 ± 13.55
1	50	-19.63 ± 0.54	-19.80 ± 0.56	251.46 ± 33.61	190.62 ± 22.40
1	40	-19.93 ± 0.04	-20.19 ± 0.11	218.78 ± 2.93	163.42 ± 14.12
1	30	-19.95 ± 0.03	-20.17 ± 0.17	221.73 ± 4.20	170.33 ± 21.35
1	20	-19.75 ± 0.32	-20.11 ± 0.25	244.51 ± 4.90	203.97 ± 23.26
1	10	-19.87 ± 0.07	-20.23 ± 0.06	238.90 ± 15.14	188.23 ± 32.58
2	80	7.15 ± 20.80	6.64 ± 21.14	444.18 ± 40.78	341.47 ± 25.71
2	70	-6.68 ± 22.92	-6.99 ± 22.80	371.68 ± 58.30	296.10 ± 36.34
2	60	-19.86 ± 0.05	-20.27 ± 0.18	371.21 ± 6.30	350.15 ± 10.13
2	50	-16.72 ± 5.48	-17.17 ± 5.20	283.81 ± 76.94	217.58 ± 62.06
2	40	-19.91 ± 0.03	-20.19 ± 0.07	252.58 ± 22.31	194.52 ± 26.93
2	30	-17.24 ± 2.34	-17.52 ± 2.43	248.98 ± 5.78	208.03 ± 7.23
2	20	-19.87 ± 0.07	-20.17 ± 0.16	241.50 ± 10.81	190.52 ± 31.40
2	10	-19.91 ± 0.06	-20.24 ± 0.09	243.51 ± 5.16	209.43 ± 32.67
3	40	20.06 ± 0.44	19.82 ± 0.88	596.50 ± 182.35	544.43 ± 166.79
3	30	-10.99 ± 10.42	-11.80 ± 9.05	274.38 ± 5.24	212.07 ± 34.62
3	20	-19.92 ± 0.01	-20.12 ± 0.10	293.12 ± 28.46	228.63 ± 28.06
3	10	-19.93 ± 0.03	-20.35 ± 0.39	251.60 ± 12.27	194.13 ± 31.28
4	20	-14.86 ± 5.39	-15.77 ± 4.82	439.72 ± 26.27	378.00 ± 28.70
4	10	-19.86 ± 0.05	-20.23 ± 0.08	297.45 ± 9.98	252.77 ± 13.72

Table 5: Training and testing performance of every glimpse size for every number of patches tested in Pong and SpaceInvaders.

No. Patches	Glimpse Size	CarRacing-v0	
		Max. Train Avg.	Test Avg.
1	96	815.12 ± 5.75	660.02 ± 69.38
1	90	675.12 ± 84.89	607.46 ± 39.81
1	80	741.03 ± 112.73	534.56 ± 72.77
1	70	747.12 ± 103.91	258.70 ± 274.03
1	60	692.40 ± 121.21	546.35 ± 40.01
1	50	559.05 ± 19.45	361.30 ± 73.27
1	40	539.68 ± 11.35	485.85 ± 9.80
1	30	187.67 ± 136.04	32.64 ± 81.82
1	20	155.15 ± 77.05	152.99 ± 75.43
1	10	138.51 ± 88.40	128.90 ± 94.17
2	48	748.70 ± 74.90	544.58 ± 132.04
2	40	694.50 ± 107.94	641.11 ± 57.42
2	30	616.74 ± 84.43	347.20 ± 243.31
2	20	465.76 ± 74.51	375.30 ± 164.83
2	10	161.44 ± 36.96	145.80 ± 48.40
3	24	700.53 ± 28.06	472.35 ± 295.73
3	20	676.82 ± 84.89	564.00 ± 56.42
3	10	545.18 ± 92.57	509.85 ± 70.74

Table 6: Training and testing performance of every glimpse size for every number of patches tested in CarRacing.

B ALGORITHMS HYPERPARAMETERS

	Hyperparameter	Value(s)
PPO	Advantage normalization	True
	Annealing learning rate	True
	Batch size	[1024, 2048]
	Clipping coefficient - action	[0.1, 0.2]
	Clipping coefficient - location	0.2
	Clipped value loss	True
	Entropy coefficient	[0.01, 0]
	GAE lambda	0.95
	Gamma	0.99
	Grayscale	True
	Learning rate - action	[2.5e-4, 3e-4]
	Learning rate - location	3e-5
	Locator normal distrib. variance	0.1
	Maximum gradient clipping norm	0.5
	Minibatch size	[256, 64]
	No. environments	[8, 1]
	No. minibatches	[4, 32]
	No. steps	[128, 2048]
	Optimizer	Adam
	Update k epochs	[4, 10]
Value function coefficient	0.5	
Glimpse	Glimpse scale	2
	Glimpse FC layer size	[384, 512]
	Location FC layer size	256
	LSTM size	128
	No. glimpses	1

Table 7: List of parameters used in GBAC, in the Atari games and CarRacing, respectively