ulm university universität
uulm

Faculty of Engineering,
Computer Science and Psychology
Institute of Neural Information Processing

Playing Atari with Deep Reinforcement Learning: Comparing Individual Models with Ensembles and Soups

Master's Project at Ulm University

Author:
Tim Wibiral
tim.wibiral@uni-ulm.de
Student Number: [redacted]

Supervisors:
Zeqiang Zhang
Prof. Dr. Daniel Braun

2023

Version from September 30, 2023

Abstract

TODO

1. Introduction

In recent years, the field of artificial intelligence has witnessed remarkable advancements in the domain of reinforcement learning (RL), particularly in the context of training agents to perform tasks in complex environments. Among the various paradigms of RL, Deep Q-learning (DQL), which combines deep neural networks with Q-learning algorithms, has emerged as a powerful approach capable of achieving superhuman performance in multiple video games [20, 2, 22]. Atari 2600 games, in particular, have served as a benchmark suite for evaluating the capabilities of RL agents due to their diverse gameplay mechanics and inherent complexity [21, 20, 9]

This project aims to investigate whether the combination of multiple trained models (Ensembles, Soups) can achieve higher scores than individual models for Deep Reinforcement Learning (DRL) for Atari 2600 games. While isolated models represent individual neural networks independently trained on a specific task, ensembles and soups combine multiple individual models to achieve a collective decision or combine their weights to create a new individual model. In other domains, like natural language tasks [24, 13], time series classification [12], or visual tasks [23, 6, 19, 5], model combinations like ensembles and soups demonstrated better performance than individual models. To provide a comprehensive evaluation, the focus lies on three diverse Atari games: Breakout, Enduro, and Seaquest.

To address these questions, Deep Q-learning is employed to train multiple deep convolutional neural networks (CNNs) on each game. The performance of these is compared to the performance of ensembles and soups that are generated using the individual models. Wilcoxon signed-rank tests are carried out to statistically analyze the performance differences between the three approaches, providing robust insights into the relative merits of isolated models, ensembles, and soups.

2. Background

This section gives a short theoretical overview on the methods used in this project.

2.1. Reinforcement Learning

Reinforcement learning (RL) is a branch of machine learning that focuses on training agents to interact with an environment and learn optimal actions to maximize a cumulative reward. In this project, RL is studied within the context of the Atari 2600 Simulator, a platform where agents are tasked with playing Atari games. This simulator serves as the environment $\mathcal{E}$ in which the agent operates. Below, the Deep Q-Learning (DQN) algorithm by Mnih et al. [21] is summarized.

The central entity in the RL framework is the agent, which is responsible for selecting actions (a_t) at each time step to navigate the Atari game environment. The set of legal game actions ($\mathcal{A}$) available to the agent is defined as $1, \ldots, K$. The agent's goal is to choose actions that lead to the highest cumulative reward. The agent's perception of the environment is limited to partial observations. At each time step, it receives an observation in the form of an image ($x_t \in \mathbb{R}^d$), representing the current game screen as a matrix of raw pixel values. This partial observation makes the problem a challenging one, as the agent lacks complete information about the environment's internal state.

The interaction between the agent and the Atari Simulator can still be modeled as a Markov decision process (MDP) by representing a state s_t of the MDP as the complete sequence of actions and observations: $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$. This sequence is finite but large and evolves over time, with each state being distinct due to the addition of new actions and observations.

The agent's objective is to maximize future rewards, where rewards (r_t) are defined as the change in game score. These rewards are discounted by a factor γ per time step. The future discounted return (R_t) at time t is calculated as the sum of rewards over time steps, with T denoting the termination time step. The optimal

action-value function Q^* represents the maximum expected return achievable for a given state-action pair (s, a) when following a policy π that maps a state to an action.

$$Q^*(s, a) = \max_\pi \mathbb{E}[R_t | s_t = s, a_t = a, \pi]$$

To approximate the action-value function, a function approximator in the form of a Deep Neural Network, called the Deep Q-Network (DQN), is employed. The DQN's weights θ are trained to estimate Q-values for state-action pairs. DQN training involves minimizing a sequence of loss functions $L_i(\theta_i)$ that change at each iteration i. These losses quantify the difference between the predicted Q-values ($Q(s, a; \theta_i)$) and the target values y_i generated by the DQN.

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[(y_i - Q(s, a; \theta_i))^2\right]$$

DQL is a model-free RL algorithm; it directly learns optimal strategies from samples gathered within the Atari Simulator, without explicitly constructing a model of the environment. Moreover, DQN employs an off-policy learning strategy, which involves learning about a greedy strategy while following a behavior distribution that ensures adequate exploration of the state space.

2.2. Ensembles

In contrast to deterministic algorithms, machine learning approaches contain a probabilistic element. While the deterministic algorithms always return the same result for the same input, machine learning algorithms are less stable and minor changes in the learning process (e.g. changes in the training dataset) can lead to vastly different results [3].

Smaller models trained on smaller datasets are more vulnerable to the effects of randomness: Removing a few samples from a small datasets can lead to a greater shift of the dataset's distribution than in a large dataset [4]. Larger Deep Neural Network architectures (i.e. models with more layers) are more resistant to the effects of randomness, but need more data and training time than smaller models. In the DQN scenario, the models seem to be especially vulnerable to distribution shifts:

They are much smaller than other model architectures created at the same time, like AlexNet [15], and the training process starts based on data created by random play [21].

Using Ensembles of models is one possible way to improve the predictions of the DQN. Instead of training a larger model, multiple shallow models are trained and used together as an Ensemble by combining their predictions [16, 8, 3]. Ensembles were first introduced for decision trees [16], followed later by the first Neural Network Ensembles [8].

Ensembles can be created ad-hoc, during the training, or post-hoc, with models that already finished training. Some methods change the way the models for the ensembles are trained, e.g. for Snapshot Ensembles the learning rate is varied during training, but all models used are snapshots of the same model [10] while Bagging algorithm vary the composition of the training data [3]. SUNRISE [17] is a variation of Bagging created for DRL. Post-hoc ensembles train a selection of classifiers (either homogenously or heterogeneously) independent of each other and combine them after the training into one Ensemble.

2.3. Soups

Similar to Ensembles, Soups aggregate multiple models (called ingredients) to achieve a higher performance. In contrast to Ensembles, all used models must have the same architecture and the Soup profits more, if the models used are very homogenous [24].

Soups are created by creating a new model from the ingredient models by averaging each weight of the ingredient models [24]. Wortsman et al. [24] report improved results when creating Soups from models that are fine-tunings of the same base model.

3. Methods

Based on the theoretical overview given above, this section describes how the methods were adapted for the practical application.

3.1. Deep Q-Learning

The application of deep learning and Q-learning called Deep Q-Learning (DQL) as described by Mnih et al. [21] applies the theoretical foundation provided above in a practical manner. The experience of the agent at each time step is stored as quadruple $e_t = (s_t, a_t, r_t, s_{t+1})$ in the *replay memory* dataset $\mathcal{D}$. The replay memory $\mathcal{D}$ contains N quadruples. After each time step, the DQN is updated using a randomly sampled batch from $\mathcal{D}$. While the MDP described by Mnih et al. [21] uses states consisting of all preceding observations of the episode, the practical application uses a fixed length representation for simplicity. This approach has the advantages, that each experience is potentially used for multiple weight updates and the random sampling improves the learning because the learning profits from uncorrelated samples. This also reduces the probability of unwanted feedback loops that can arise and keep the agent in a local minimum [21].

The discount factor γ is set to 0.99. The weight updates are performed using the RMSProp algorithm with minibatches of size 32. Learning rate and other hyperparameters of the RMSProp were not given by Mnih et al. [21], so it was assumed they are identical to the ones given in Mnih et al. [20]. The exploration rate ϵ is linearly decreased from 1 to 0.1 over the span of the first million frames. Afterward, it is fixed for the remaining 9 million frames. All in all, the training takes 10 million frames. The size of the replay memory is fixed to the most recent 1 million frames [21].

In this project, the training duration had to be scaled down from 10 million frames to 3 million frames. The training is run independently multiple times to create Ensembles of independent models. This would not have been feasible for training with 10 million frames in the time scope of the project. Proportionally, the replay memory size was reduced to $300,000$ samples. The span of reducing ϵ to 0 was kept 1 million frames.

The screen image of the Atari simulator was resized to 84×84 pixels and grey-scaled. Frame-skipping is applied and the agent acts every $4th$ frame. The selected action is repeated until action is selected 4 frames later [21].

Mnih et al. [20] published an improved version of the algorithm two years later. The authors changed two things: First, they used a larger model. Second, they introduced a target network, that stabilizes the training process. The target network is used to calculate the Q-values and the other model, the policy model is trained. This way, the other model "chases" the future reward predicted by the target model. Every $10,000$ update steps the parameters of the policy model are copied to the target model [20]. While Mnih et al. [20] still reduce ϵ over the first 1 million frames, they trained their model "for a total of 50 million frames (that is, around 38 days of game experience in total)" ([20], p. 6).

In this project, the update frequency was reduced to 3,000 frames, proportional to the 3 million frames of training. The model architectures used are described in more detail below.

3.2. Deep Neural Network Architectures

Mnih et al. [21] and Mnih et al. [20] used Convolution Neural Networks (CNN) that are described below. This project also introduces slight variations; one is to try to achieve slightly better performance, and the other tries to make the agents behavior better interpretable.

3.3. Mnih et al. [21]

The model proposed by Mnih et al. [21] consists of two convolutional layers and two linear layers. Both convolutional layers and the first fully-connected layer use ReLU [7] as activation function. The second fully-connected layer is the last layer of the model and activated linearly. The first convolutional layer consists of 16 filters of size 8×8 and a stride of 4 and the second convolutional layer consists of 32 filters of size 4×4 and a stride of 2. The first fully-connected layer has 256 units, but the

number of units in second fully-connected layer varies depending on the game. It is equal to the number of valid actions for the game at hand and varies from 4 to 18.

3.4. Mnih et al. [20]

Two years later, Mnih et al. [20] proposed a very similar model, but larger. The first convolution layer has 32 kernels (instead of 16) and the second convolutional layer has 64 kernels (instead of 32). The first fully-connected layer also doubles the number of units to 512. Other than that, the model is analogue to the previous model [21, 20].

3.5. Adam Optimizer and Huber Loss

This slight modification of the model proposed by Mnih et al. [20] uses variations suggested by the official code examples of Keras^[`https://keras.io/examples/rl/deep_q_network_breakout/`] : Adam [14] is used instead of RMSProp as is claimed that this improves the training time and the Huber function [11] is used as loss function as is claimed that it improves the stability.

3.6. Interpretable CNN

Convolutional layers have the property, that they can preserve spatial information. Assume that some information that is in the upper left corner of the input image. When a convolution is applied to that image, a filter is shifted over the image and applied to squares of the size of the filter. By the application of the filter, pieces of information get perturbed, but they keep their spatial position; if the filter is applied to the upper left corner of the input image, then the information is preserved in the upper left corner of the convoluted matrix. Transitively, this also holds for the sequential application of multiple filters.

This can property can be exploited using deep neural network that consist of only convolutional layers, and no fully-connected layers. Arandjelović and Zisserman

[1] demonstrated that this technique works for convolutional neural networks when combining images and sound data. Their model architecture inspired the architecture used in this project and is described below. The model was created with the goal of exploiting the spatial stability of convolution to interpret the decisions of the model. It will further be called Interpretable CNN.

The Interpretable CNN consists of three convolutional layers. The first two are identical to the convolutional layers of the model proposed by Mnih et al. [20], except that the first layer only gets one screen image as input, instead of four. The Interpretable CNN is also trained using the Huber loss and the Adam optimizer.

The third convolutional layer combined with a two-dimensional maximum pooling takes over the purpose of the linear fully-connected layers above. The number of filters of the third convolutional layer is equal to the number of valid actions n for the game at hand. This means that the output of the third convolutional layer has the shape $9 \times 9 \times n$. This cannot be used for classification, so the pooling operation over the first and second dimension reduces the $9 \times 9 \times n$ tensor to a vector of size n which is used as the output y for the optimization of the model parameters. This way, each of the n matrices shaped 9×9 stands in for one of the n possible actions. The higher the values in the 9×9 matrix, the higher is the corresponding value in the vector y after pooling.

After training the model, the parts of each 9×9 matrix have higher values / are more activated at positions that correlate to positions in the input image that favor the action that correlates to this 9×9 matrix. Put simply, when overlaying the 9×9 matrix of a selected action with the input image, the regions of the input image that influenced the selection of the given action are highlighted. This enables an interpretation of the learning process of the model, for example by comparing which regions are highlighted by different snapshots of the model.

An important disadvantage is that the model relies on a single input image, instead of multiple, making it hard for the model to interpret movement on the screen. Hence, the Interpretable CNN will most likely achieve lower scores than the other models.

3.7. Ensembles

The predictions of the independent classifiers of an ensemble can be combined in different ways, dependent on the output of the models. Reinforcement Learning for Atari games is a discrete learning task, so the methods used in this project are the following:

- Averaging: The prediction of the ensemble is the average of all the predictions of classifiers in the ensemble. This method is based on [12]
- Logistic Averaging: Similar to averaging, but the sigmoid function is applied to each individual classifier prediction before averaging. This way, the impact of classifiers that have very high outputs for one or more classes is modulated. Also based on [12].
- Majority Voting: Each classifier votes for one class. The prediction of the Ensemble is the class that got the most votes. This method is based on [18].

For this work, the Ensembles are created after training a few models independent of each other. Three different kinds of Ensembles are created this way:

- Top-3 Ensemble: An Ensemble that contains three models of the same architecture. From all models of that architecture the three best are selected.
- Snapshot Ensemble: This method is based on [10]. Huang et al. [10] altered the training process and save a snapshot whenever the model arrives at a local minimum, then increase the learning rate. The resulting M snapshots are used to create an Ensemble. As local minima are hard to pin down in RL, this work uses the last M snapshots of a model instead.
- Mixed Ensemble: Contains the best three models of each model architecture (nine models total).

Because of the high variance between the training data in different runs of the training, I expect the Ensembles to perform better than the individual models. However, I do not expect the different Ensemble methods (like averaging and voting) to perform significantly different.

3.8. Soups

The Soups used in this work are not created based on fine-tunings of the same base model as proposed by Wortsman et al. [24], but instead by using either three models trained independently or three snapshots of the same model at different stages of the training. Similar to the Ensembles, two kinds of Soup are cooked:

- Top-3 Soup: A Soup created using the three best performing models of an architecture.
- Snapshot Soup: A Soup created using three snapshots of a models training.

I expect the Snapshot Soup to perform better, as it created from more homogenous models. Like observed by Wortsman et al. [24], I expect them to not perform significantly better or worse than the Ensembles.

4. Experiments

This section describes how the training experiments were run and the resulting models were evaluated.

4.1. Used Atari Games

Within the scope of the project it was not possible to run the DQL algorithm on all available Atari games. Instead, three games were selected with the goal to be able to compare the results to the works of Mnih et al. [21] and Mnih et al. [20].

Mnih et al. [21] observed, that their algorithm achieved on some games much better scores and on some games much worse scores than humans. Based on these results the games Breakout, Enduro, and Seaquest were selected, to cover a range

of different performances compared to humans. The DQN achieved much higher scores on Breakout than humans, similar scores on Enduro, and much worse scores on Seaquest.

4.2. Procedure

One Interpretable CNN model is trained for each game, but 3 models of each of the three other architectures are trained. One model is enough for the analysis of the Interpretable CNN, but for the other models multiple models are necessary to build ensembles.

Each model is trained for 3 million frames, starting with an exploration rate ϵ of 1 that is reduced linearly to 0.01 over the first million frames. Afterward, the models are evaluated. For the evaluation, ϵ is set to 0.05, like in the work of Mnih et al. [21]. This is necessary to fight a bug in breakout, where the game "freezes" until a certain button is pressed. Just adding some randomness is a simple way of getting rid of this bug.

For the evaluation, a random baseline is created to compare the model performances to by playing 250 random games. Each of the individual models plays 10 games with $\epsilon = 0.05$, resulting in 30 data points per model architecture (except for the Interpretable CNN, which has only 10).

The ensembles and soups are created like described. Each Ensemble is evaluated with each ensemble method described above (majority vote and (logarithmic) averaging) for 10 games each. Due to the time constraints, only three models of each architecture were trained, hence the ensembles and soups not only use the *best* 3 models, but *all* 3 models.

5. Results and Discussion

This section describes the results of the experiments using box plots and Critical Difference Diagrams (CDDs). The CCDs were created as propsed by Ismail Fawaz et al. [12], using a Wilcoxon-Holm analysis with a significance level α of to determine pairwise significant differences between the models. The results of models that are connected via a black bar do not have a significant difference in their performance, but models that are not connected do have significant differences. Table A.1 in the Appendix shows the exact results for all models, ensembles, and soups. The + symbol in the box plots indicates the mean of the data.

5.1. Analysing the Interpretable CNN

An examination of the output of the last convolutional layer grants insights into the learning process of the Interpretable CNN. Figures 5.1 to 5.3 show heatmaps created from the last convolutional layer overlaid onto the screen image that was inputted into the model. The highlighted areas in darker red are the areas show higher activation in last convolutional layer and therefore had more impact on the models decision. For one input image, the model generates n different output heatmaps (with n being the number of valid actions for the game) of shape 9×9. The heatmap shown is always the heatmap with the highest maximum activation, and therefore the heatmap correlated to the action with the highest expected Q-value.

Figure 5.1 shows the last frame before the agent gets a reward. The darker red colored areas are the areas the Interpretable CNN deems the most important for the decision. The ball is in the darkest colored square. This indicates that the Interpretable CNN learned a representation of the ball.

The same frame is shown in Figure 5.2, but four different heatmaps were created based on four different snapshots taken during the training of a model. The upper left was created before training; at this point the models weights were initialized randomly and hence the heatmap was random. After 1 million frames, the heatmap looks already similar to the heatmap after 3 million frames.

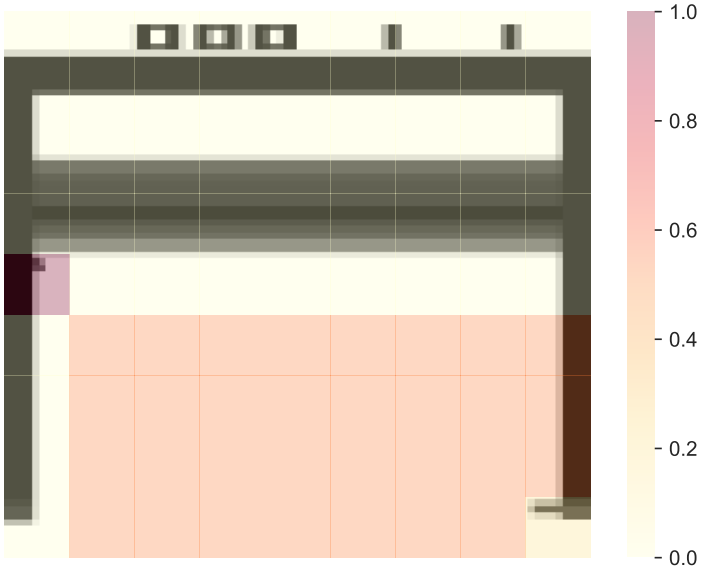

Figure 5.1.: Screenshot directly before the frame hits the tiles and a reward is given. Darker areas are areas with more influence on the decision. The ball is in the darkest square.

The ball bounces back to the player after the reward. Figure 5.3 shows how the Interpretable CNN has learned to locate the ball. The activation at the position of the ball is very low, but the activation around the ball is high, leading to a circle around the ball.

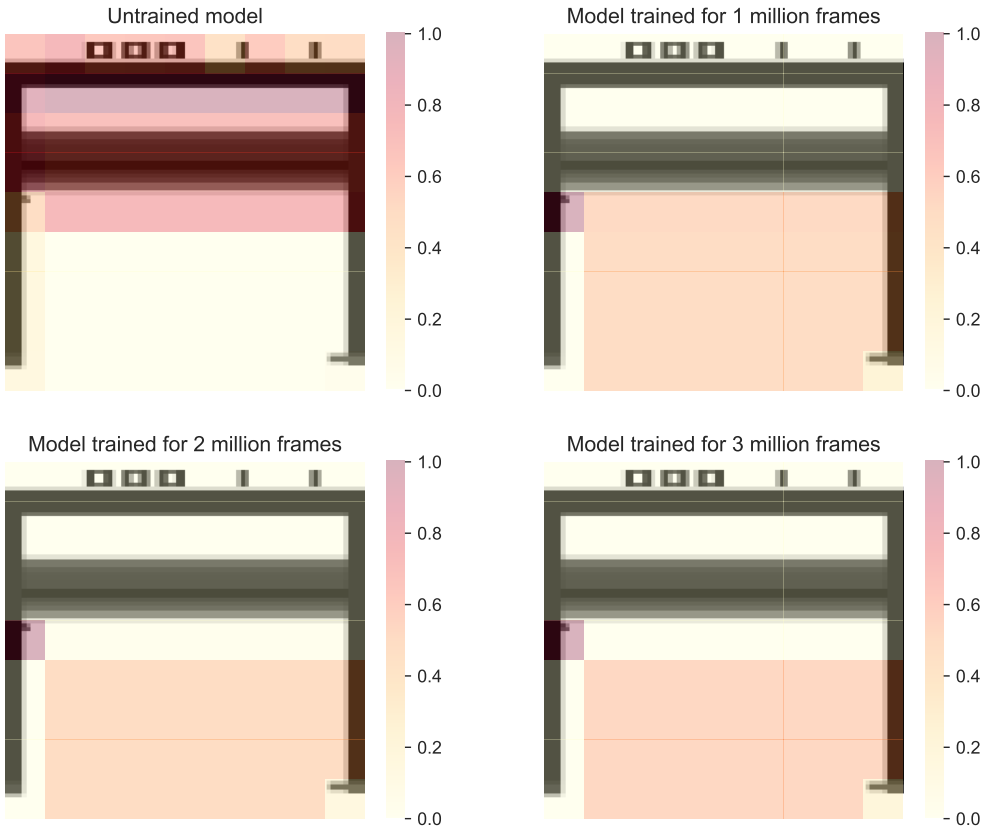

Figure 5.2.: Screenshot directly before the frame hits the tiles and a reward is given. The dark areas are randomly spread out before training, but quickly become more expressive.

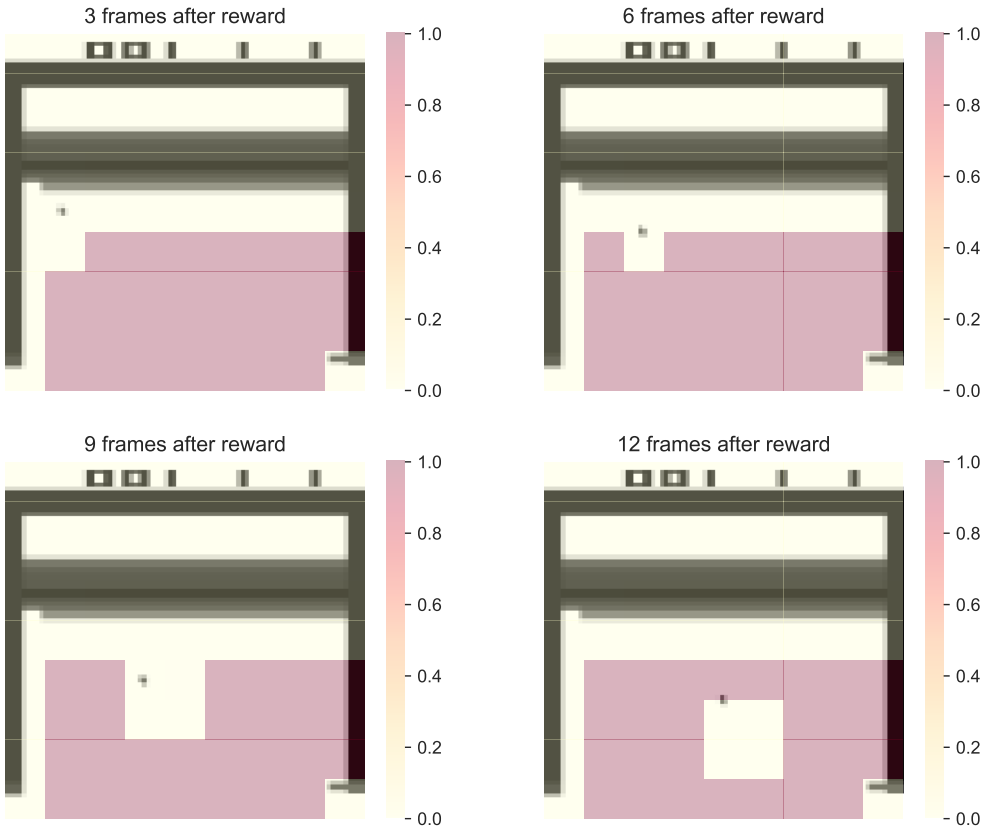

Figure 5.3.: Screenshots taken several frames after the reward was given. The ball is encircled, indicating that the Interpretable CNN learned to locate the ball.

5.2. Individual Models

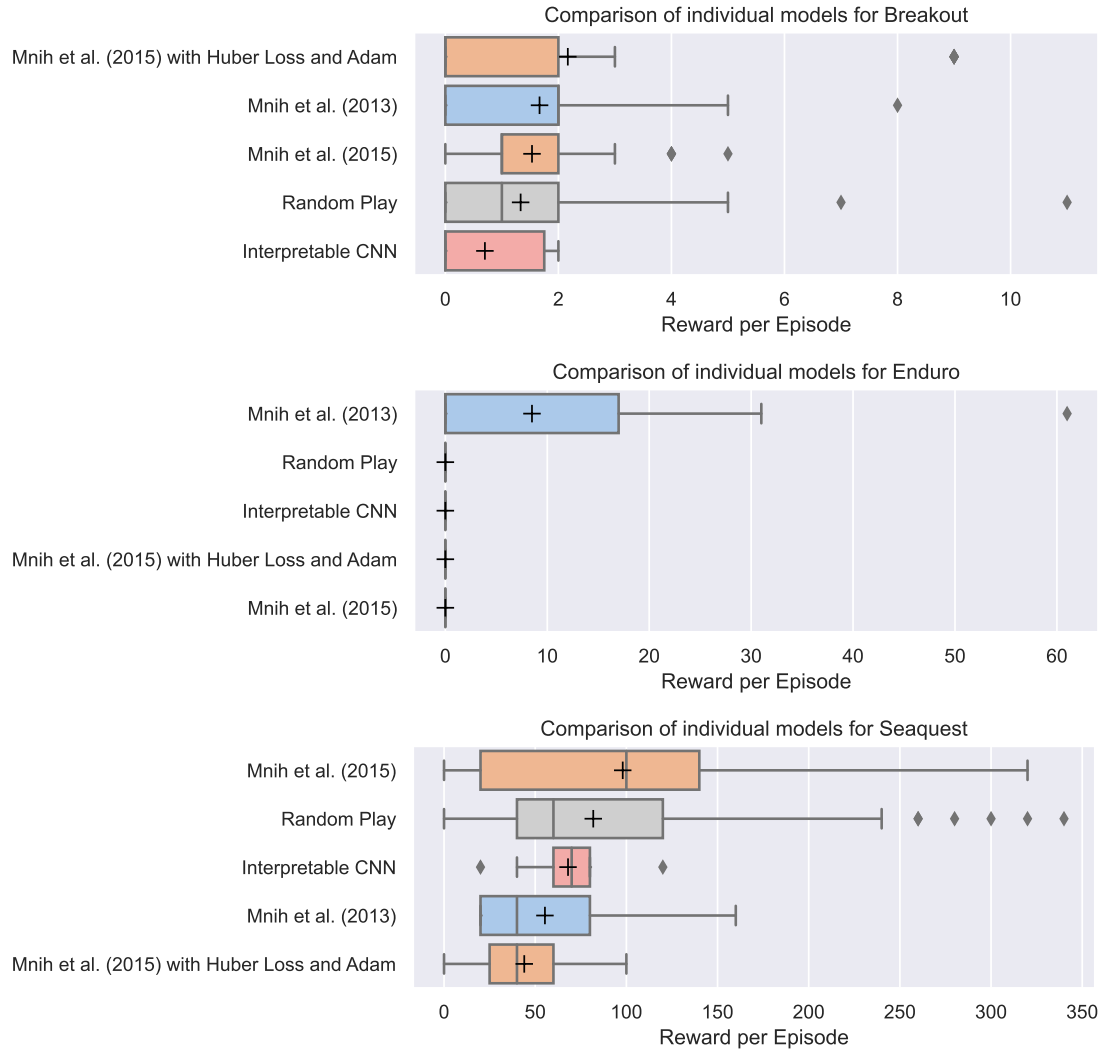

Figure 5.4.: Performance of individual models on Breakout, Enduro, and Seaquest.

Figure 5.4 shows the performance of individual models. All of the individual models, except for the Interpretable CNN, performed better than random play on Breakout. Interpretable CNN performs worse than the others, as sees only one frame (instead of four) making it impossible to learn the movement of the ball.

Only the model architecture by Mnih et al. [21] achieved a score higher than 0 on Enduro. Enduro is a racing game where the player has to stay on the button for

driving and steer to overtake cars. The DRL algorithm starts out playing random, but random play almost never leads to a long enough repetition of pressing the button to accelerate, hence only seldom receiving a reward signal from which the model can learn. That the model architecture by Mnih et al. [21] could be a product of better random play, but also of the model architecture being smaller and learning faster from sparse reward. The results of Mnih et al. [21, 20] tained for much longer, but also achieved much better results on Enduro. Running the DRL algorithm for longer could lead to more rewards and therefore better performance.

On Seaquest, the model architecture by Mnih et al. [20] performed the best, but the same architecture but trained using Huber loss and Adam optimizer performed the worst. While Huber loss and Adam optimizer improved the performance for Breakout, they did not help for Seaquest. Mnih et al. [21] reported that the DQN performance on Seaquest was much worse compared to humans, so it is no surprise that the models trained in this project performed worse compared to random play in Seaquest than compared to random play in other games.

5.3. Ensembles

In general, the mixed Ensembles (colored yellow in Figure 5.5) that combine multiple different architectures seem to perform better than other Ensembles. Even though the architecture by Mnih et al. [21] was the only individual achieving a score higher than 0 on Enduro, the mixed Ensemble of the architecture that involves all three architectures performed better than most of the Ensembles only containing models of the architecture by Mnih et al. [21]. Interestingly, the Ensemble of 3 Snapshots of the model architecture by Mnih et al. [20] trained using Huber loss and Adam optimizer achieved scores higher than 0 in Enduro, even though the individual models of the same architecture did not achieve a score higher than 0.

Based on this limited data, there cannot be given a clear answer which of the ensemble methods is the best one for DRL on Atari games. The Wilcoxon pairwise significance test (α = 0.05) resulted in no significant difference between the different ensembles. Training the models on more different games could reveal if any of the Ensembles is superior to the others.

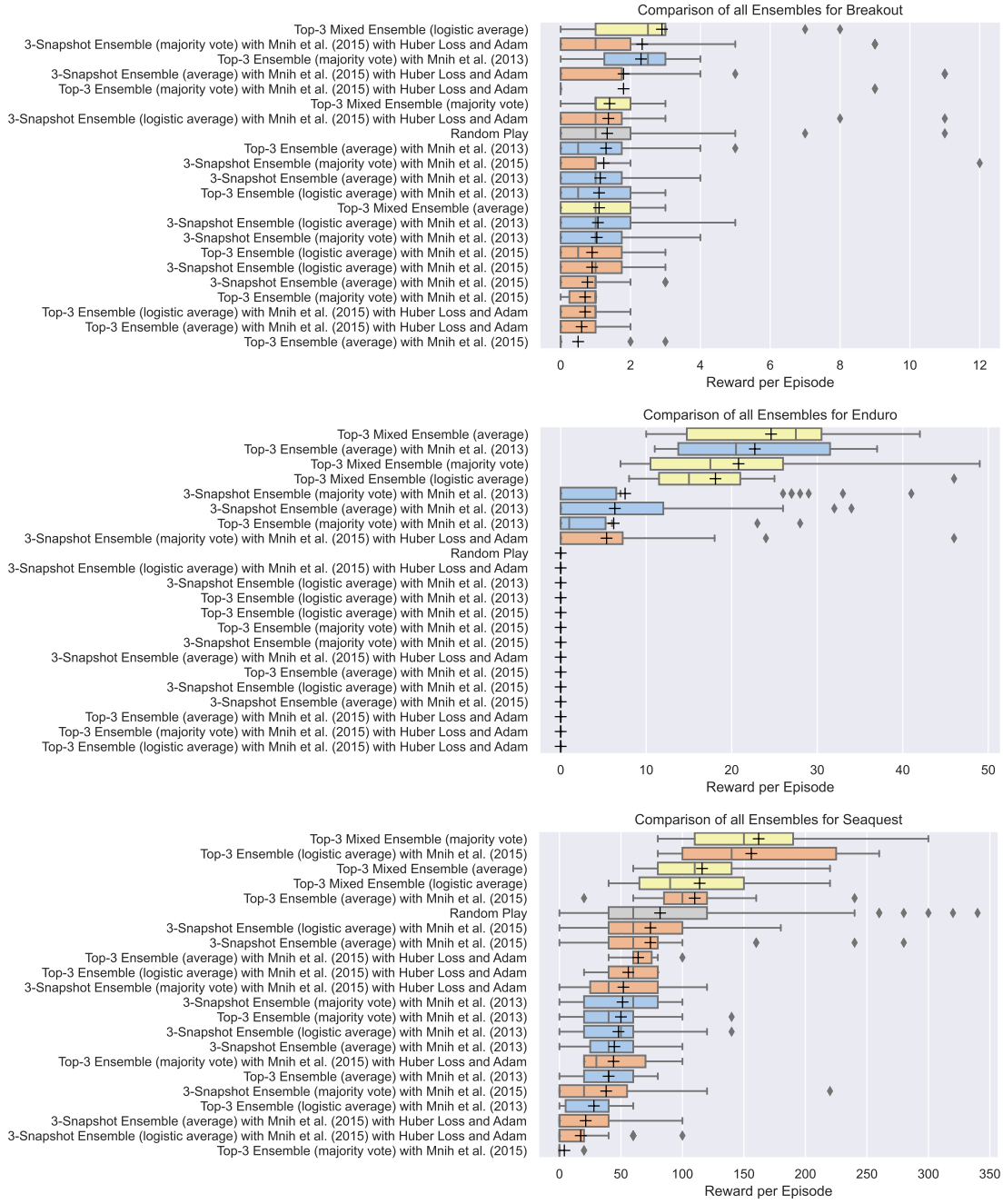

Figure 5.5.: Performance of Ensembles on Breakout, Enduro, and Seaquest.

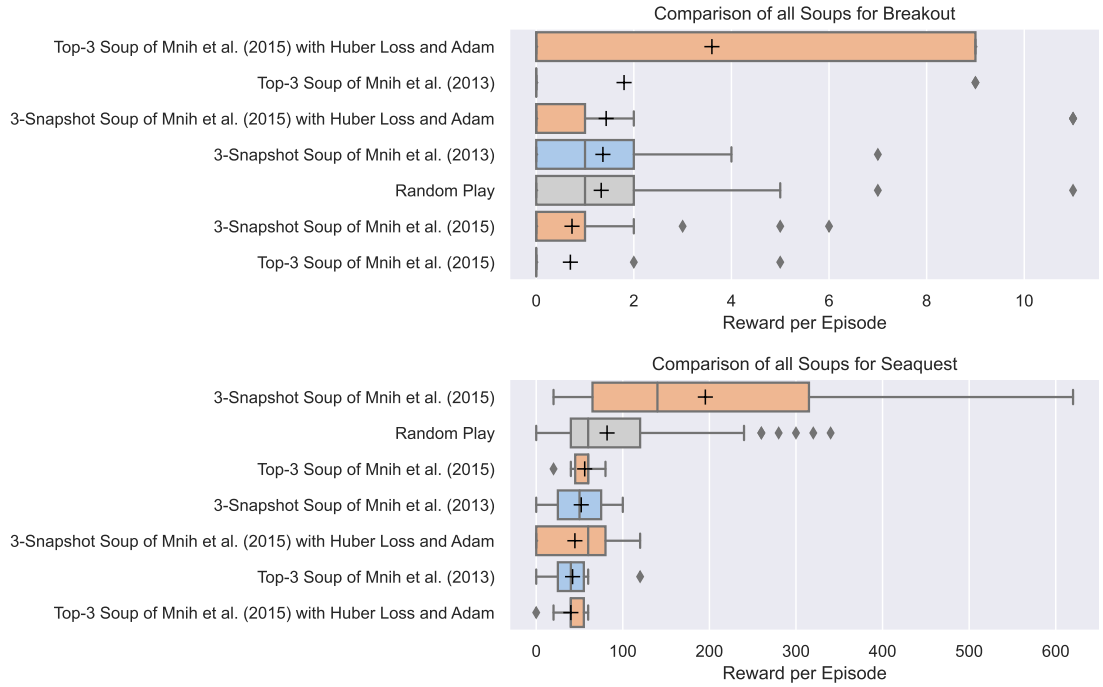

Figure 5.6.: Performance of Soup models on Breakout, Enduro, and Seaquest. (None of the Soups achieved a score higher than 0 on Enduro, hence the plot for Enduro is omitted.)

5.4. Soups

None of the Soup models achieved a score higher than 0 on Enduro, hence the plot for Enduro is omitted in Figure 5.6. There seems to be no difference in performance in the Soups created from three different models of the same architecture and the Soups created from three snapshots of the same model. This is surprising: Soups take the average of the weights of the ingredient models, hence I expected the new weigths to have higher quality if they stem from the same learning procedure (snapshot Soups) and are already more similar before the averaging. The Wilcoxon pairwise significance test (α = 0.05) resulted in no significant difference between the different soups.

Figure 5.7.: Best performing individual models, Ensembles, and Soups. The best performing models of each domain for each model architecture were selected.

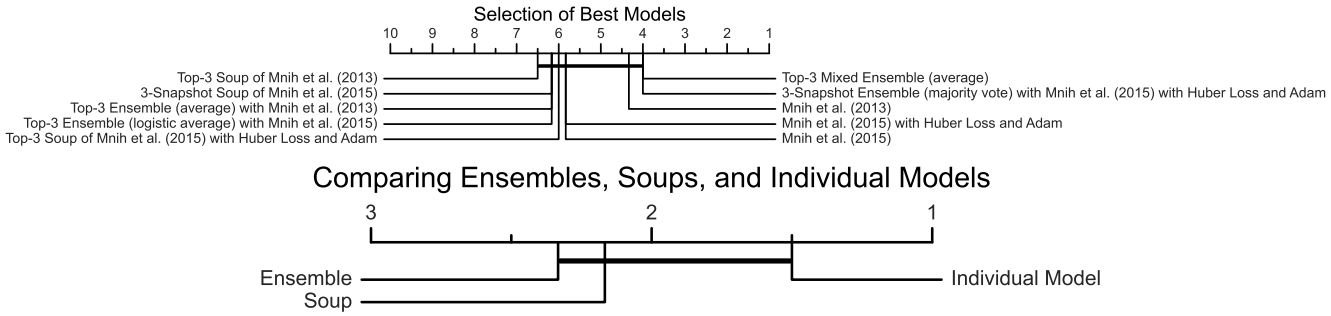

Figure 5.8.: *Top*: Critical Difference Diagram comparing a selection of the best performing models. No significant differences were found. *Bottom*: Comparing the overall performance of Ensembles, Soups, and individual models. No significant differences were found.

5.5. Overview

This section aims to answer the question if Ensembles and Soups generelly achieve better results than single models in the context of DRL on Atari 2600 games. As Figure 5.7 shows, Ensembles and Soups occupy the majority of the top ranks, but also the majority of the bottom ranks. Only the mixed Ensembles containing models with different architectures achieve top performance consistently.

The Critical Difference Diagram in Figure 5.8 shows that there is no significant difference between the best performing models of each domain. It also shows, that the comparison of all Ensembles with all Soups and all individual models does not reveal a significant difference.

6. Conclusion

The results of this project do not support the hypothesis that Ensembles and Soups achieve better results than single models in the context of DRL on Atari 2600 games. Nonetheless, the results of this project are not conclusive, as the number of models

trained is small, the models were only trained for 3 million frames, and only three games were used.

- Does not give Information about frameworks like SUNRISE

- Fix problem with sparse rewards by integrating more time steps in future reward calculation

- Select more "easier" games

A. All Results

Table A.1 shows the exact results for all models, ensembles, and soups.

	Breakout	Enduro	Seaquest
Interpretable CNN	1.000 ± 1.000	40.375 ± 43.694	40.375 ± 43.694
Mnih et al. [20]	2.500 ± 1.871	88.438 ± 95.957	88.438 ± 95.957
Mnih et al. [20] with Huber Loss and Adam	3.000 ± 3.536	31.500 ± 36.776	31.500 ± 36.776
Mnih et al. [21]	3.167 ± 2.927	38.000 ± 42.715	38.000 ± 42.715
3-Snapshot Ensemble (average) with Mnih et al. [20]	1.500 ± 1.291	82.167 ± 96.577	82.167 ± 96.577
3-Snapshot Ensemble (average) with Mnih et al. [20] with Huber Loss and Adam	3.833 ± 3.971	26.300 ± 35.979	26.300 ± 35.979
3-Snapshot Ensemble (average) with Mnih et al. [21]	2.000 ± 1.581	26.833 ± 28.103	26.833 ± 28.103
3-Snapshot Ensemble (logistic average) with Mnih et al. [20]	1.500 ± 1.291	65.500 ± 66.148	65.500 ± 66.148
3-Snapshot Ensemble (logistic average) with Mnih et al. [20] with Huber Loss and Adam	4.167 ± 4.355	24.500 ± 32.992	24.500 ± 32.992
3-Snapshot Ensemble (logistic average) with Mnih et al. [21]	2.200 ± 1.924	42.818 ± 50.864	42.818 ± 50.864
3-Snapshot Ensemble (majority vote) with Mnih et al. [20]	3.750 ± 5.560	59.545 ± 67.784	59.545 ± 67.784
3-Snapshot Ensemble (majority vote) with Mnih et al. [20] with Huber Loss and Adam	3.333 ± 3.266	29.895 ± 35.774	29.895 ± 35.774
3-Snapshot Ensemble (majority vote) with Mnih et al. [21]	2.000 ± 1.581	28.111 ± 28.367	28.111 ± 28.367
Top-3 Ensemble (average) with Mnih et al. [20]	1.667 ± 1.528	78.500 ± 79.156	78.500 ± 79.156
Top-3 Ensemble (average) with Mnih et al. [20] with Huber Loss and Adam	1.000 ± 1.000	40.429 ± 41.158	40.429 ± 41.158
Top-3 Ensemble (average) with Mnih et al. [21]	2.400 ± 2.074	23.778 ± 21.515	23.778 ± 21.515
Top-3 Ensemble (logistic average) with Mnih et al. [20]	1.500 ± 1.291	104.182 ± 97.274	104.182 ± 97.274
Top-3 Ensemble (logistic average) with Mnih et al. [20] with Huber Loss and Adam	1.000 ± 1.000	29.000 ± 31.932	29.000 ± 31.932
Top-3 Ensemble (logistic average) with Mnih et al. [21]	1.500 ± 1.291	18.000 ± 23.615	18.000 ± 23.615
Top-3 Ensemble (majority vote) with Mnih et al. [20]	0.500 ± 0.707	7.000 ± 11.269	7.000 ± 11.269
Top-3 Ensemble (majority vote) with Mnih et al. [20] with Huber Loss and Adam	4.500 ± 6.364	41.500 ± 40.367	41.500 ± 40.367
Top-3 Ensemble (majority vote) with Mnih et al. [21]	2.000 ± 1.581	32.846 ± 43.353	32.846 ± 43.353
Top-3 Mixed Ensemble (average)	1.500 ± 1.291	48.600 ± 57.233	48.600 ± 57.233
Top-3 Mixed Ensemble (logistic average)	3.500 ± 3.271	47.550 ± 60.250	47.550 ± 60.250
Top-3 Mixed Ensemble (majority vote)	1.500 ± 1.291	69.238 ± 90.704	69.238 ± 90.704
3-Snapshot Soup of Mnih et al. [20]	2.833 ± 2.317	185.708 ± 187.752	185.708 ± 187.752
3-Snapshot Soup of Mnih et al. [20] with Huber Loss and Adam	3.500 ± 5.066	43.400 ± 44.335	43.400 ± 44.335
3-Snapshot Soup of Mnih et al. [21]	2.833 ± 2.483	28.818 ± 35.977	28.818 ± 35.977
Top-3 Soup of Mnih et al. [20]	2.333 ± 2.517	29.571 ± 31.379	29.571 ± 31.379
Top-3 Soup of Mnih et al. [20] with Huber Loss and Adam	4.500 ± 6.364	25.800 ± 24.253	25.800 ± 24.253
Top-3 Soup of Mnih et al. [21]	4.500 ± 6.364	41.500 ± 44.153	41.500 ± 44.153
Random Play	4.125 ± 3.563	123.720 ± 117.537	123.720 ± 117.537
Humans (via Mnih et al. [21])	31	368	28010
Mnih et al. [21]	168	470	1705
Mnih et al. [20]	401 ± 26.9	301.8 ± 24.6	5286 ± 1310

Table A.1.: Table with results for Breakout, Enduro, and Seaquest. The number after ± indicates the standard deviation

B. A Note on Keras and PyTorch

I implemented the algorithm using Tensorflow and Keras, as I am more familiar with Keras and used it for more projects. It worked well with the *CartPole*-environment and achieved good results in reasonable time. When I went on to test it on the Atari Games for 10,000 or 50,000 frames, it also seemed to work, but did not learn a lot. To achieve good performance, a lot more experience is necessary; [21] trained for 10 Million frames and [9] trained for around 200 Million frames.

While running it for a few thousand frames on my desktop computer worked fine, it is not feasible to run the algorithm for 10 million frames on it. Instead, I tried to run it on the *BWUniCluster* for 1 million frames with one GPU and 100GB of RAM allocated. Unfortunately, the program execution failed because the program needed more than the available 100GB.

It was obvious now, that I there was some problem with the memory use, as 1 Million frames should only need around 28GB of memory space. The same problems occurred in the official code examples of Keras[1] where they only managed to run 1 million frames with a buffer of 100,000 frames.

The memory footprint still didn't change after optimizing the buffer to delete each tensor separately when a tuple is removed from the buffer. It became clear that most of the memory was used in the background by Keras itself, hence I figured, that Keras must be calculating the gradients for everything that was happening. After isolating most calls to Keras functions into separate functions (with disabled automatic gradient calculation) the amount of memory was reduced by two-thirds and the program execution speed up by three. Still, it was not possible to isolate all calls to Keras and some gradients are still calculated unnecessarily.

In retrospec, it would have been better to use PyTorch from the beginning, as PyTorch allows more control over which gradients are calculated. Maybe that is one of the reasons why other RL libraries like Stable-Baseline3[2] use PyTorch.

[1] `https://keras.io/examples/rl/deep_q_network_breakout/`
[2] `https://stable-baselines3.readthedocs.io/en/master/`

Bibliography

[1] Relja Arandjelović and Andrew Zisserman. “Objects that Sound”. en. In: *arXiv:1712.06651 [cs, eess]* (July 2018). arXiv: 1712.06651. URL: http://arxiv.org/abs/1712.06651 (visited on 12/02/2020).

[2] Kai Arulkumaran et al. “Deep Reinforcement Learning: A Brief Survey”. In: *IEEE Signal Processing Magazine* 34.6 (Nov. 2017). Conference Name: IEEE Signal Processing Magazine, pp. 26–38. ISSN: 1558-0792. DOI: 10.1109/MSP.2017.2743240. URL: https://ieeexplore.ieee.org/abstract/document/8103164 (visited on 09/28/2023).

[3] Leo Breiman. “Bagging predictors”. en. In: *Machine Learning* 24.2 (Aug. 1996), pp. 123–140. ISSN: 1573-0565. DOI: 10.1007/BF00058655. URL: https://doi.org/10.1007/BF00058655 (visited on 11/23/2022).

[4] Anna Choromanska et al. “The Loss Surfaces of Multilayer Networks”. en. In: *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*. ISSN: 1938-7228. PMLR, Feb. 2015, pp. 192–204. URL: https://proceedings.mlr.press/v38/choromanska15.html (visited on 11/23/2022).

[5] Charles Dansereau et al. *Model soups to increase inference without increasing compute time*. arXiv:2301.10092 [cs]. Jan. 2023. DOI: 10.48550/arXiv.2301.10092. URL: http://arxiv.org/abs/2301.10092 (visited on 09/28/2023).

[6] Manik Goyal and Jagath C. Rajapakse. *Deep neural network ensemble by data augmentation and bagging for skin lesion classification*. arXiv:1807.05496 [cs]. July 2018. DOI: 10.48550/arXiv.1807.05496. URL: http://arxiv.org/abs/1807.05496 (visited on 11/21/2022).

[7] Richard H. R. Hahnloser et al. “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit”. en. In: *Nature* 405.6789 (June 2000). Number: 6789 Publisher: Nature Publishing Group, pp. 947–951. ISSN: 1476-4687. DOI: 10.1038/35016072. URL: https://www.nature.com/articles/35016072 (visited on 09/26/2023).

[8] L.K. Hansen and P. Salamon. “Neural network ensembles”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12.10 (Oct. 1990). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 993–1001. ISSN: 1939-3539. DOI: 10.1109/34.58871.

[9] Hado van Hasselt, Arthur Guez, and David Silver. *Deep Reinforcement Learning with Double Q-learning*. en. arXiv:1509.06461 [cs]. Dec. 2015. URL: http://arxiv.org/abs/1509.06461 (visited on 09/02/2023).

[10] Gao Huang et al. *Snapshot Ensembles: Train 1, get M for free*. arXiv:1704.00109 [cs]. Mar. 2017. DOI: 10.48550/arXiv.1704.00109. URL: http://arxiv.org/abs/1704.00109 (visited on 08/16/2023).

[11] Peter J. Huber. “Robust Estimation of a Location Parameter”. In: *The Annals of Mathematical Statistics* 35.1 (Mar. 1964). Publisher: Institute of Mathematical Statistics, pp. 73–101. ISSN: 0003-4851, 2168-8990. DOI: 10.1214/aoms/1177703732. URL: https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-35/issue-1/Robust-Estimation-of-a-Location-Parameter/10.1214/aoms/1177703732.full (visited on 09/26/2023).

[12] Hassan Ismail Fawaz et al. “Deep Neural Network Ensembles for Time Series Classification”. en. In: *2019 International Joint Conference on Neural Networks (IJCNN)*. Budapest, Hungary: IEEE, July 2019, pp. 1–6. ISBN: 978-1-72811-985-4. DOI: 10.1109/IJCNN.2019.8852316. URL: https://ieeexplore.ieee.org/document/8852316/ (visited on 11/06/2022).

[13] Monisha Kanakaraj and Ram Mohana Reddy Guddeti. “Performance analysis of Ensemble methods on Twitter sentiment analysis using NLP techniques”. In: *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*. Feb. 2015, pp. 169–170. DOI: 10.1109/ICOSC.

2015.7050801. URL: https://ieeexplore.ieee.org/document/7050801 (visited on 09/28/2023).

[14] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. arXiv:1412.6980 [cs]. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 03/19/2023).

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012. URL: https://proceedings.neurips.cc/paper/2012/hash/c399862d3b9d6b76c8436e924a68c45b-Abstract.html (visited on 09/23/2023).

[16] Suk Wah Kwok and Chris Carter. "Multiple decision trees". en. In: *Machine Intelligence and Pattern Recognition*. Ed. by Ross D. Shachter et al. Vol. 9. Uncertainty in Artificial Intelligence. North-Holland, Jan. 1990, pp. 327–335. DOI: 10.1016/B978-0-444-88650-7.50030-5. URL: https://www.sciencedirect.com/science/article/pii/B9780444886507500305 (visited on 11/23/2022).

[17] Kimin Lee et al. "SUNRISE: A Simple Unified Framework for Ensemble Learning in Deep Reinforcement Learning". en. In: *Proceedings of the 38th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, July 2021, pp. 6131–6141. URL: https://proceedings.mlr.press/v139/lee21g.html (visited on 09/23/2023).

[18] N. Littlestone and M. K. Warmuth. "The Weighted Majority Algorithm". In: *Information and Computation* 108.2 (Feb. 1994), pp. 212–261. ISSN: 0890-5401. DOI: 10.1006/inco.1994.1009. URL: https://www.sciencedirect.com/science/article/pii/S0890540184710091 (visited on 09/26/2023).

[19] Roman C. Maron et al. "Model soups improve performance of dermoscopic skin cancer classifiers". In: *European Journal of Cancer* 173 (Sept. 2022), pp. 307–316. ISSN: 0959-8049. DOI: 10.1016/j.ejca.2022.07.002. URL: https://www.sciencedirect.com/science/article/pii/S0959804922004129 (visited on 09/28/2023).

[20] Volodymyr Mnih et al. “Human-level control through deep reinforcement learning”. en. In: *Nature* 518.7540 (Feb. 2015), pp. 529–533. ISSN: 0028-0836, 1476-4687. DOI: 10.1038/nature14236. URL: http://www.nature.com/articles/nature14236 (visited on 05/08/2023).

[21] Volodymyr Mnih et al. *Playing Atari with Deep Reinforcement Learning*. en. arXiv:1312.5602 [cs]. Dec. 2013. URL: http://arxiv.org/abs/1312.5602 (visited on 05/08/2023).

[22] Kun Shao et al. *A Survey of Deep Reinforcement Learning in Video Games*. en. arXiv:1912.10944 [cs]. Dec. 2019. URL: http://arxiv.org/abs/1912.10944 (visited on 09/28/2023).

[23] Guihua Wen et al. “Ensemble of Deep Neural Networks with Probability-Based Fusion for Facial Expression Recognition”. en. In: *Cognitive Computation* 9.5 (Oct. 2017), pp. 597–610. ISSN: 1866-9964. DOI: 10.1007/s12559-017-9472-6. URL: https://doi.org/10.1007/s12559-017-9472-6 (visited on 11/21/2022).

[24] Mitchell Wortsman et al. *Model soups: averaging weights of multiple fine-tuned models improves accuracy without increasing inference time*. arXiv:2203.05482 [cs] version: 3. July 2022. DOI: 10.48550/arXiv.2203.05482. URL: http://arxiv.org/abs/2203.05482 (visited on 08/16/2023).