Towards Understanding Single-Task vs Multi-Task agents in Atari games

Sven Kellenberger, Daniel Klingmann, Andreas Lüscher, Simon Wengeler

I. Introduction

With the advent of deep neural networks, the field of reinforcement learning (RL) has made tremendous progress over the last couple of years and is currently most widely known for its breakthroughs in playing games at super-human performance. The interest in this field was ignited by Mnih *et al.* [1], which presented the first algorithm that could be trained to play a variety of Atari 2600 games at a competitive level. Very rapidly, optimised algorithms have been put forward and agents nowadays achieve scores orders of magnitude higher than the ones obtained initially. In contrast to a controlled video game environment, real world situations are much more diverse and demand for an agent that can react to a variety of situations in a meaningful manner. One way of simulating more complex environments, while still staying in the controlled video game setup, is to consider a so-called multi-task agent which can play several different games. As in the real world, the Atari environments are very distinct from one another and as such, serve as an ideal playground for studying such multi-task agents. One of the challenges of multi-task learning is finding the right balance between the different tasks and thus preventing the agent from focusing on environments with particularly high or frequent rewards. In this report, we study two multi-task agents on a subset of Atari games and compare their behaviour to the corresponding single-task models. Inspired by previous work on trying to understand RL agent behaviour [2], we use saliency maps for a qualitative analysis and propose two methods of quantitatively assessing the differences between models.

In RL, an agent observes a state s_t at time t and interacts with its environment by taking action a_t, upon which it receives a reward $r_t = r(s_t, a_t)$ [3], [4]. The sequence of states, actions and rewards observed over a whole episode (from the beginning until the end of the game) is called a trajectory or rollout of the policy or decision-making rule $\pi(s, a)$. Every rollout is characterised by the cumulative discounted reward $R_t = r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots$, with the discount factor $\gamma \in (0, 1]$. The goal of RL is to find the policy that maximises the state value function $V^\pi(s) = \mathbb{E}_\pi[R_t \mid S_t = s]$.

II. Models and Methods

While there are over 50 Atari 2600 games in Arcade Learning Environment (ALE) [5], we limit our analysis to the following six environments: AirRaid, Carnival, DemonAttack, NameThisGame, Pong, and SpaceInvaders. Note that AirRaid and Carnival are not part of the widely used Atari-57 suite and have thus not been included in the multi-task approach presented in [6]. All our results are based on the NoFrameskip-v4 variant, in which the input to the network is comprised of four contiguous greyscale frames of size 84 by 84 pixels, as illustrated in Fig. 1. As a starting point, we use the

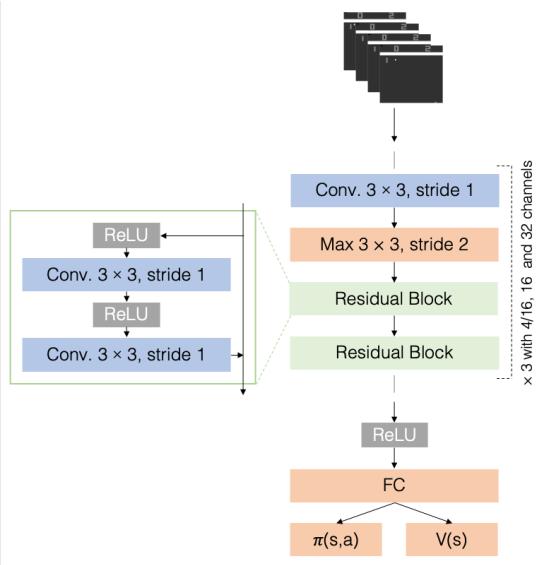


Fig. 1: Deep neural network used in the experiments[7]: The network takes four consecutive greyscale images of size 84 by 84 pixels as inputs and returns the policy $\pi(s, a)$ and the state value function $V(s)$. The model has 15 convolutional layers and approximately one million parameters. The corresponding computational graph of the model is shown in Fig. 7.

large model proposed in [7] without the LSTM-layer, as illustrated in Fig. 1. The model has 15 convolutional layers and $1'089'232 + 258$ ($\#\,\text{actions} + \#\,\text{tasks}$) parameters, where in the single- or naïve multi-task approach (see below), the number of tasks is 1. While the size of the action space does not substantially alter the total number of parameters, we believe that restricting ourselves to an identical and reduced space allows for a more controlled comparison of results and is also beneficial with regards to training times. Note that 90% of the parameters are contained in the fully connected layer at the end of the network. The network produces two outputs, the policy $\pi(s, a)$ and the state value function $V^\pi(s)$, respectively.

Two years ago, Espeholt *et al.* [7] proposed an efficient scalable architecture for reinforcement learning in simulated environments (named Importance Weighted Actor-Learner Architecture or IMPALA), comprised of one or more central learners that consume environment rollouts from many indi-

vidual actors to learn a policy. One difficulty of this distributed environment is the inherent asynchronicity between actor and learner policies: because the learner policy evolves during trajectory generation, it is already several steps ahead when it consumes the rollout obtained by the then already outdated actor policy. To circumvent this problem, IMPALA uses a V-trace off-policy actor-critic algorithm.

Very recently, the IMPALA architecture, originally based on TensorFlow, was ported to PyTorch [8] and made available under the name TorchBeast. We decided to use this framework as a basis for our extensions and experiments and added multi-task training functionality and an implementation of PopArt normalisation [6], [9].

A. Naïve approach

Generally, multi-task training can be achieved in two ways: Either the model is exposed to all environments simultaneously or training evolves sequentially, exposing it to one environment after another. Clearly, the first approach seems more promising, since in sequential training, one would expect the adaptations to previous environments to be washed out by the most recent one. This intuitive choice is corroborated by experiments, see Sec. III-B. We thus adapted the existing code to the parallel multi-task setting, using a fixed number of actors per environment. In this way, even a single mini-batch processed by the learner could (and in most cases did in fact) contain trajectories from different environments.

B. PopArt

One of the difficulties of multi-task learning lies in the fact that even for similar action and observation spaces, training a single agent to learn multiple tasks requires the agent to make comparisons between rewards obtained in different environments. In the naïve approach described above, this was achieved by limiting the rewards to the interval $[-1, 1]$. Clearly, clipping the rewards at some arbitrary value is not desirable, because it makes it impossible to rank actions with rewards above this cut-off value. This problem can be eliminated very elegantly by introducing task-dependent normalised state value functions, see [6], [9]. With this extension, the output layer of the network produces a normalised state value function for every environment or task, while at the same time keeping track of the mean and the variance of the unnormalised values. We would like to emphasise that this task-dependence is only used during training to guide the policy updates, the action predictions are always task-independent.

C. Saliency maps

To analyse and compare the models in more detail, we use the perturbation-based saliency method described in [10]. The basic idea behind perturbation approaches is to alter the input data fed to the network and extract the modifications arising from this perturbation. In such a way, one can identify input regions which are particularly important for the decision making. Perturbations can range from blurring localised image patches to introducing whole new elements to the input scenery. Here, we follow [10] and apply a Gaussian blur to localised regions of the input image. In this way, we construct a saliency image for the policy (always shown in red) and the value estimate (shown in green). In contrast to the input used in [10], which uses single greyscale images, our models are trained with four stacked consecutive frames and we introduce the perturbation at the same locations in all four images. While such an adaptation is trivial from a technical point of view, it is not a priori clear that it yields equally meaningful results, since an object blurred in the first frame might already have moved outside the perturbed area on a subsequent frame. However, our results suggest that such a fixed-location blurring also works with a small number of stacked images. In most of our saliency maps, the activity values were widely dispersed, such that after linear transformation to RGB images, only the maximum was clearly visible while all the other features vanished in the background. To prevent this feature disappearance, we resorted to a piece-wise linear transformation of the features above the 90% quantile, using 10 segments.

D. CNN filter comparisons

Furthermore, we compare the model parameters directly, in particular the weights learned by the convolutional layers of the network. From work on the visualisation of Convolutional Neural Networks it is known that CNN filters early in the network respond to more low-level features of the input images (e.g. edges) and later layers learn higher-level features [11]. We propose to quantitatively measure the difference between models by computing the mean over the Sum of Squared Differences (SSD) of all corresponding filters in a convolutional layer. In addition, we also minimise that value by finding the optimal matching between the filters of the compared networks (separately for each layer). This is done because the initialisation of the models' weights may cause specific filters to learn different features in different models. We mainly want to study the difference between the two multi-task models and how each of them differ from the single-task models, but also compare pairs of single-task models. For this we use 10 checkpoints for each model recorded at equal intervals until the end of training.

E. Action Distributions

Another metric we compare is the distribution of the actions the different models choose for different environments. We quantify this by evaluating a given model on a given environment for 50 complete games. We then compute the mean and standard deviation for each action. Because some actions can have multiple effects for some environments, we compute the distributions once for every distinct action and once for every distinct effect the actions can have.

III. Results and Discussion

Using a subset of Atari games, we trained three different agents and compared their behaviour: a multi-task agent with

reward clipping to the interval $[-1, 1]$, an unclipped multi-task agent with the additional PopArt layer and the corresponding single-task agents. Note that we also use reward clipping for the single-task agents to allow for a proper comparison between single-and multi-task agents and also to validate our experiments with published results that almost exclusively use reward clipping. While it would have been desirable to also include the results from single-task models with the PopArt extension, the restricted time frame did not allow us to complete these experiments.

A. Model performance

Fig. 2 shows the performance of the three different agents as a function of the number of training steps. The corresponding videos are available online. As performance measure, we always use the mean episode returns over 100 episodes together with corresponding standard deviation. While absolute scores are not important for the comparison of different agents, it is nevertheless worthwhile pointing out that two models clearly outperform a professional human game tester [1]. A more detailed analysis of the absolute scores of individual games can be found in Sec. V-C. Interestingly, despite averaging, the performance varies tremendously even after millions of training steps. Measuring performance in terms of only the mean value can thus be somewhat misleading. While using the median instead of the mean leads to a slight reduction of fluctuations and might thus be a better measure, we think that for game playing, the mean value over a certain number of episodes is a more meaningful performance indicator. Single-task models were trained for 50 million steps, while for the multi-task models, we used a total of 300 million training steps equally divided among the six environments. Taking into account frame stacking, this translates to 200 million environment steps per environment.

Focusing on the relative performance of the selected models, we observe that:

- In all six environments, the PopArt extension has a clear positive effect on performance.
- In five environments, the PopArt extension even results in a multi-task model that outperforms the clipped single-task agent.
- In all cases, both multi-task implementations exhibit a faster initial learning period compared to the single-task variant.
- Even the naïve multi-task agent performs surprisingly well compared to the expert single-task models.

We would like to emphasise that these statements only hold when fluctuations are neglected. Taking the observed confidence intervals into account more strictly, they would only hold for a single environment.

Clearly, the use of a normalised task specific value function during learning allows the model to better capture the dynamics of the individual environments within a single model, see also Sec. V-D for detailed results. Importantly, as pointed out in [7], this is not only due to the enlarged representational power of the network - recall that with the PopArt extension, the model also has a few additional parameters, but is only enabled by the proper normalisation, see Fig. 1 in [6].

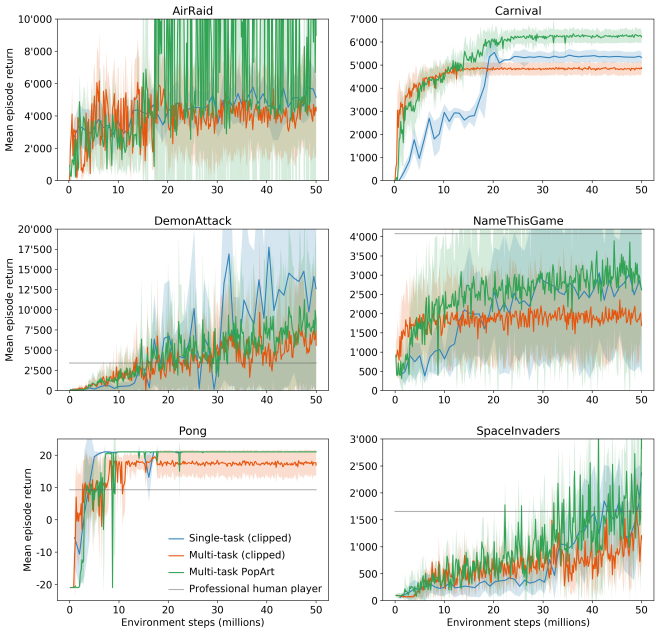


Fig. 2: Mean episode returns and standard deviation (shaded area above and below the mean) over 100 episodes as a function of the number of training steps for a three different agents: a multi-task agent with reward clipping, an unclipped multi-task agent with PopArt extension and the corresponding clipped single-task agents. Multi-task agents are trained for a total of 300 million steps, equally divided among the six environments. The corresponding videos are available online.

B. Pre- and detraining

The observation of faster initial training regimes in multi-task agents suggests that some sort of transfer learning must naturally take place in multi-task models. Exploiting this phenomenon could not only be beneficial to multi-task agents, but also lead to better or at least more efficiently trainable single-task agents. To elucidate this assumption further, we have trained several single-task agents starting from a pre-trained multi-task model, see top row of Fig. 3. We used a naïve three-task model for AirRaid, Carnival and DemonAttack as basis, monitored the subsequent training of a fourth single-task environment and compared its evolution to the same single-task agent starting from a randomly initialised model. Clearly, starting from a pre-trained multi-task model has only a very small positive or in one case even a detrimental effect on the subsequently trained model. This influence could be expected for Pong, which has a very different game plan compared to the other five games, but is rather surprising for the two other environments.

The setup of pre-trained networks also allows one to study the effects of de-training. To that end, we again use the naïve three-task network described above, but this time report the performance of the three initially trained models during the

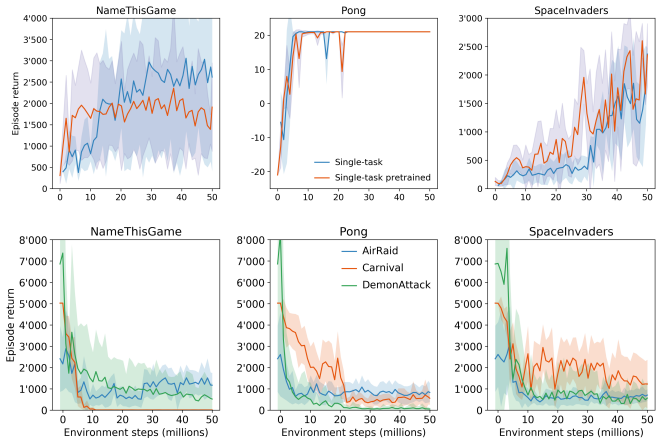


Fig. 3: Visualisation of pre- (top row) and de-training (bottom row). As in Fig. 2, we report the mean episode returns over 100 episodes as a function of the number of training steps. The three-task network used as a starting point was comprised of AirRaid, Carnival and DemonAttack. In contrast to earlier results, we trained the model for a total of 100 million steps, with comparable contributions from each environment. The top shows the training evolution of a subsequently trained single-task model whereas the bottom row shows the degrading performance of the model exposed to the same three networks while training the fourth one.

learning cycle of the new fourth environment, see bottom row of Fig. 3. In other words, the results for top- and bottom row of this figure are based on the same network, but its performance is assessed in different environments. These results indicate that the model realigns its weights to the new environment within only a few million iterations and that any remnants of the old environments are completely washed out at the end of the training cycle.

C. Saliency maps

To analyse the behaviour of the three models in more detail, we have computed the saliency maps for whole episodes. Recall that red indicates the policy saliency amd green represents the saliency for the value estimate. The videos are also available online. Fig. 8 shows selected images from these videos. To compare the three models, we have used the trajectories obtained from the best performing model and replayed this policy with the other two models. While the game plans for the three models show quite different behaviour, leading for instance to the different overall performance shown in Fig. 2, the saliency maps are surprisingly similar.

Despite their similarities, one can identify several differences: In NameThisGame, the single-task agent constantly follows the boat on the surface of the water, while the PopArt network is focused on the moving tentacles. Both behaviours are absent in the naïve multi-task model. In SpaceInvaders, the PopArt agent shows a very distinct focus on all the invaders above the shooter, while in the other two models, this is less pronounced, and attention is much more dispersed. Despite producing nice videos and images, the practical impact of this technique is in our opinion rather limited, because it seems impossible to translate these finding into actual model improvements. To illustrate this point further, we have

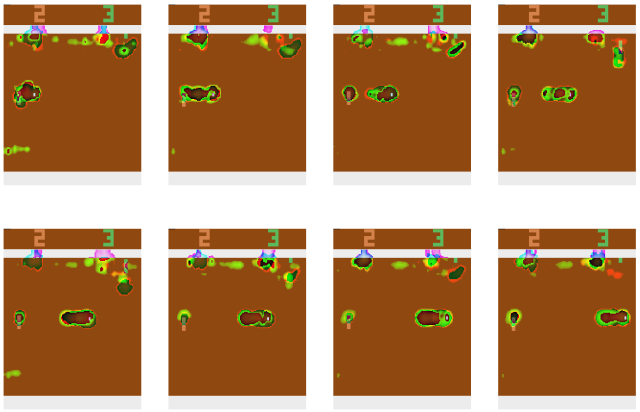


Fig. 4: Saliency map for the odd behaviour of the naïve multi-task agent playing Pong. Despite the fact that the network follows the trajectory of the ball, the agent does not make any attempt to move the paddle downwards to catch the ball.

analysed the most intriguing observation encountered in Fig. 2, namely the poor performance of the naïve multi-task agent in the Pong environment. Despite the short learning period indicating that Pong is a rather simple environment which is also the least harmed by reward clipping, the simple multi-task model cannot match the performance of the single-task agent. This difference becomes even more apparent when watching the agent play the game: In certain situations, the agent keeps the paddle fixed at the top and does not make any attempt to catch the ball. Fig. 4 shows the corresponding saliency maps, which clearly show that the network follows the trajectory of the ball but does not give any indication as to why the paddle is stuck at the top.

D. CNN filter comparisons

One commonality of all comparisons between different models is that the differences between filters increase with more training but flatten out towards the end of training. This is likely due to models converging on a set of parameters that changes less when the model has been trained for a long time. Furthermore, the mean of SSDs between filters in each layer is generally quite small, with all recorded values below 0.45. For a change in a single element of a filter this would only constitute an absolute difference of about 0.67. While we have not compared the weights of the fully connected layer in the networks (which is harder to interpret), the clear differences between models detailed below still suggest that these small changes have an impact.

Comparing each single-task model with the others reveals that the differences between each pair are very similar. Thus Fig. 9a in the appendix shows only the comparison between the model trained on the Carnival environment and the other

single-task models. It also shows that the differences are smaller in later layers (darker blue) and larger in earlier layers (brighter blue). In fact, the first layer (the lightest blue) clearly has the largest differences and is the only one that shows bigger differences between specific pairs of models. Since this pattern is even more pronounced for optimal matching between the filters (Fig. 9b) this is unlikely to be only due to different permutations of the same kind of learned filters. Instead it may indicate that different low-level features are learned for different games (especially in the first layer), which may be more important for that specific game. Comparing the optimal with the default matching between filters gives a fairly consistent ratio between the values for all layers and model pairs of about 4.7. However, the ratios increase with the depth of the network, with the first layers only having a ratio of about 2.8.

The differences between the two multi-task models are larger than between individual single-task models, except for the first layer (see Fig.10). Here there is also less of clear separation between earlier and later layers. The mean ratio between values for the optimal and default matching is similar (except for the first layer). The differences between some of the early layers of the networks seem to be especially large and it would be interesting to investigate the features learned in these layers more in the future (e.g. by generating images activating these filters strongly). Additionally, the "ordering" of the layers in terms of their relative differences changes much more between optimal and default matching than for single-task comparisons.

The comparison between the vanilla multi-task model and the single-task models gives similar results. The differences are larger than between single-task models (although still smaller than between the two multi-task models, except for the first layer), the ratio between optimal and default matching is smaller (about 3.6), and the relative ordering of the layer differences changes (see Fig. 11). Similar patterns can be observed when comparing the multi-task model with PopArt with the single-task models (see Fig. 12). However, the differences are generally larger for all layers and the curves flatten out less quickly over training. Additionally, a clearer separation of earlier and later layers can be observed again, with earlier layers showing more of a difference. In addition, while differences for the other layers are generally similar between the games, for the first layer they vary more widely (also compared to their smaller variation in the vanilla multi-task model).

These results show that PopArt normalisation has a clearly observable impact on the convolutional layers of the trained network beyond the changes that can be expected from a vanilla multi-task agent. Our results indicate that this may have more of an impact on the earlier layers in the network which may be of interest for further analysis. The structure of the differences between pairs of single-task models appears to show some similarities, particularly the separation of earlier and later layers and the similarity between all of the differences between all pairs. This is also a result which may be interesting to investigate further.

E. Action Distributions

An overview of the results can be found in Sec. V-H. Especially when looking at the distribution of every distinct action, we can see that the different agents can vary quite a lot in their chosen strategies to play the game. This phenomenon occurs because the Atari environments allow for multiple actions to have the same or similar effects and therefore can be played in multiple different ways. However, we can also see that every agent specialises in one strategy or another, i.e. the agents usually limit their action set further down than the minimal action set of the environments.

By looking at the effects of the actions rather than the actions themselves we can see a lot more similarities between the agents. However, one thing that becomes clear very quickly is that the single-task agents usually are more patient than the multi-task agents. The single-task agents fire less and in the case of Pong they spend more time than the other agents doing nothing (*NOOP*). This can be explained by the multi-task agents generally preferring to perform actions which move the player and fire simultaneously. This means they just fire as soon as possible because there is a cool-down on the fire functionality in each of the environments. The single-task agent on the other hand generally uses the *FIRE* action explicitly when it wants to fire and not in conjunction with another effect. This suggests that the single-task agents can better distinguish between the effects their actions have and use them more deliberately.

IV. Conclusion

To conclude, we have implemented and analysed two different multi-task agents - a simple and a more sophisticated one - on a subset of Atari games and compared its behaviour to the corresponding single-task agents. To the best of our knowledge, this is the first freely available PopArt extension on top of a scalable architecture. Assessing the raw performance of these agents, we found that addressing the problem of different reward scales with a sophisticated normalisation approach is clearly superior to the naïve reward-clipping method but were surprised to see that a simple multi-task agent still performs very decently. To further analyse the behaviour of these three agents, we compared their saliency maps - we are not aware that this approach has been carried out before for multi-task environments - and were again surprised to see that despite exhibiting different game plans and raw performances, the saliency information was remarkably similar and thus did not shed much light on the differences present in the deep neural network. While further research needs to be done to better understand the meaning/effect of these results, our proposed method of comparing CNN filters shows clearer differences between the vanilla and PopArt multi-task models as well as surprising similarities between the single-task models.

REFERENCES

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015. [Online]. Available: http://dx.doi.org/10.1038/nature14236

[2] A. Mott, D. Zoran, M. Chrzanowski, D. Wierstra, and D. J. Rezende, "Towards interpretable reinforcement learning using attention augmented agents," *ArXiv*, 2019. [Online]. Available: http://arxiv.org/abs/1906.02500

[3] R. Bellman, "A markovian decision process," *Indiana Univ. Math. J.*, vol. 6, pp. 679–684, 1957.

[4] R. Sutton and A. Barto, *Reinforcement learning: An introduction*. Cambridge Univ Press, 1998, vol. 116.

[5] M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling, "The arcade learning environment: An evaluation platform for general agents," *Journal of Artificial Intelligence Research*, vol. 47, pp. 253–279, 2013.

[6] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, "Multi-task deep reinforcement learning with popart," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3796–3803.

[7] L. Espeholt, H. Soyer, R. Munos, K. Simonyan, V. Mnih, T. Ward, Y. Doron, V. Firoiu, T. Harley, I. Dunning *et al.*, "Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures," *arXiv*, 2018, https://github.com/deepmind/scalable_agent. [Online]. Available: http://arxiv.org/abs/1802.01561

[8] H. Küttler, N. Nardelli, T. Lavril, M. Selvatici, V. Sivakumar, T. Rocktäschel, and E. Grefenstette, "TorchBeast: A PyTorch Platform for Distributed RL," *arXiv*, 2019, https://github.com/facebookresearch/torchbeast. [Online]. Available: http://arxiv.org/abs/1910.03552

[9] H. P. van Hasselt, A. Guez, M. Hessel, V. Mnih, and D. Silver, "Learning values across many orders of magnitude," in *Advances in Neural Information Processing Systems*, 2016, pp. 4287–4295.

[10] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv*, 2017. [Online]. Available: http://arxiv.org/abs/1711.00138

[11] Z. Qin, F. Yu, C. Liu, and X. Chen, "How convolutional neural network see the world-a survey of convolutional neural network visualization methods," *arXiv preprint arXiv:1804.11191*, 2018.

V. Appendix

A. Source code and trained models

The source code and the final trained models are available on the github repository. Our extensions are based on [8], [10].

B. Hyperparameters

The hyperparameters used for training are summarised in Tab. I and correspond to the default values of the TorchBeast implementation, slightly differing from the ones used in [7], [8]. The models were trained on Google Cloud Platform on a standard VM instance with 16 CPUs and two NVIDIA Tesla P100 GPUs. Compiling and running PolyBeast on the ETH cluster turned out to be more cumbersome than initially planned.

Parameter	Our Value	IMPALA and TorchBeast [7], [8]
alpha	0.99	0.99
baseline_cost	0.5	0.5
batch_size	8	32
discounting	0.99	0.99
entropy_cost	0.0006	0.01
epsilon	0.01	0.01
grad_norm_clipping	40	40
learning_rate	0.00048	0.0006
momentum	0	0
reward_clipping	abs_one / none	abs_one
unroll_length	80	20

TABLE I: Hyperparameters used for training.

C. Comparison with TorchBeast results

In [8], the authors evaluated the model performance using the results obtained during training. The reported performance is thus the mean of only a small number of episodes. In contrast, the mean episode returns in Fig. 2 are the average over 100 episodes. In order to compare the two results more directly, we have followed the same data extraction procedure as in [8], although only for a single training run. The comparisons are shown in Fig. 5. In five of the six environments, we find good agreement between the results, but for the SpaceInvaders environment, our returns are a factor of magnitude smaller than the reported ones. After carefully reviewing the optimisation settings, we found out that in contrast to [8], we have used slightly different hyperparameters, see Tab. I. However, even after using the same parameters as in the paper, we still found results very similar to our initial experiments. We are thus quite puzzled by this huge discrepancy observed in only a single environment. We would like to thank Heinrich Küttler for providing us with the raw data of Fig. 3 in [8].

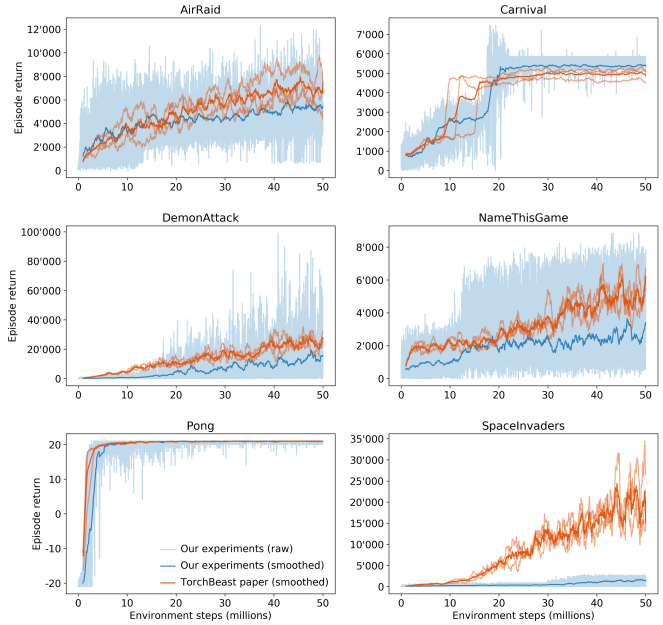


Fig. 5: Comparing the mean episode returns with [8], which in contrast to Fig. 2 are directly obtained during training, we find a good agreement in five of the six environments. The discrepancy for SpaceInvaders might be due to different ATARI ALE settings. Note that the red curve (Torchbeast paper) is obtained from three different training runs, whereas we only have a single run available.

D. PopArt statistics

To visualise the effect of the PopArt normalisation, we have monitored the game-specific statistics during training, see Fig. 6, which nicely illustrates that reward structures can differ by orders of magnitude from one environment to the other. As explained in [9], they follow the total episodes returns shown in Fig. 2, the difference in magnitude being due to discounting.

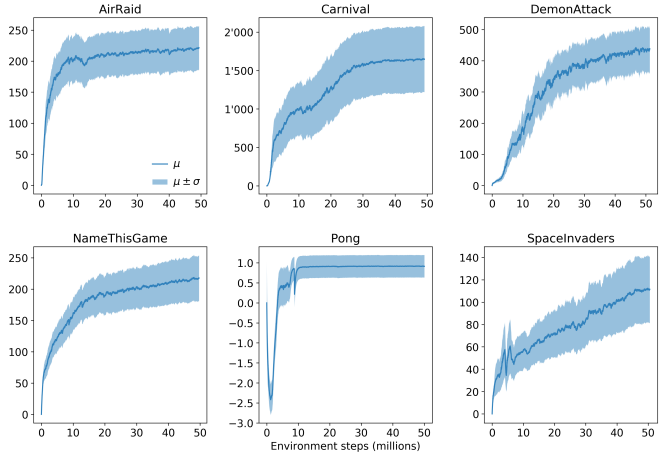


Fig. 6: The evolution of the normalisation statistics during training show that the reward structures differ significantly from one environment to the other and that this variability cannot be captured adequately by a single variable for all environments. As explained in [9], they follow the total episodes returns shown in Fig. 2, the difference in magnitude being due to discounting.

E. Computational graph

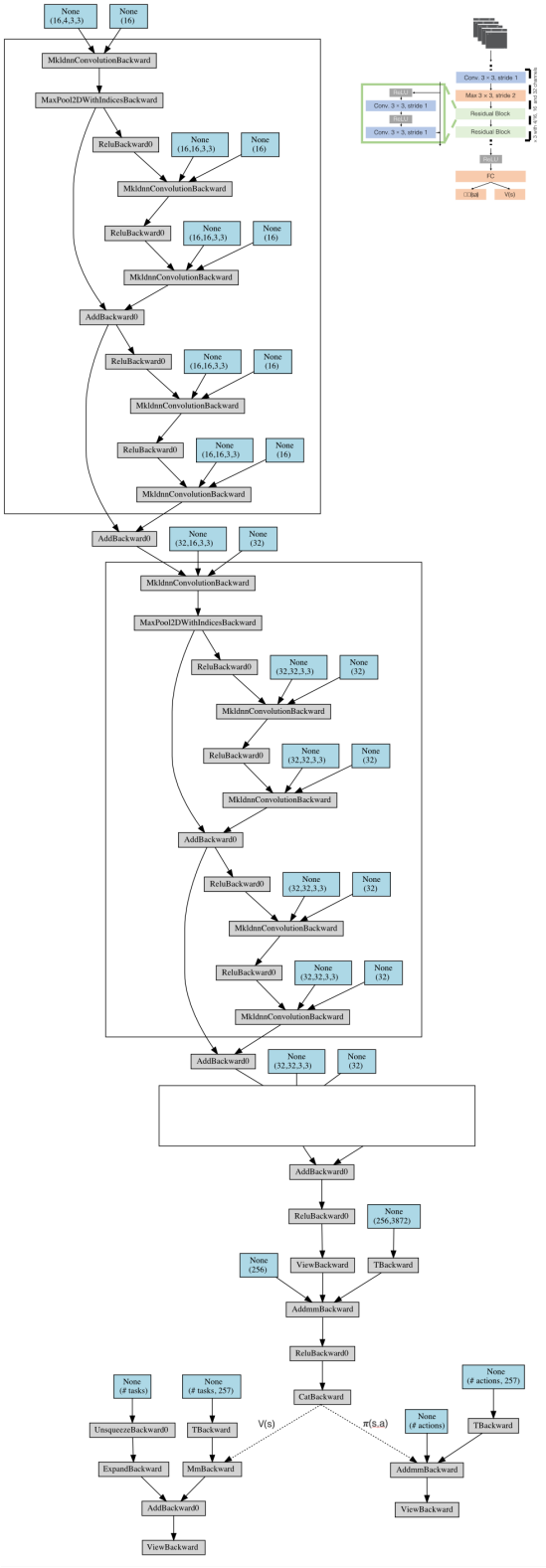


Fig. 7: Computational graph of the studied model.

F. Saliency maps

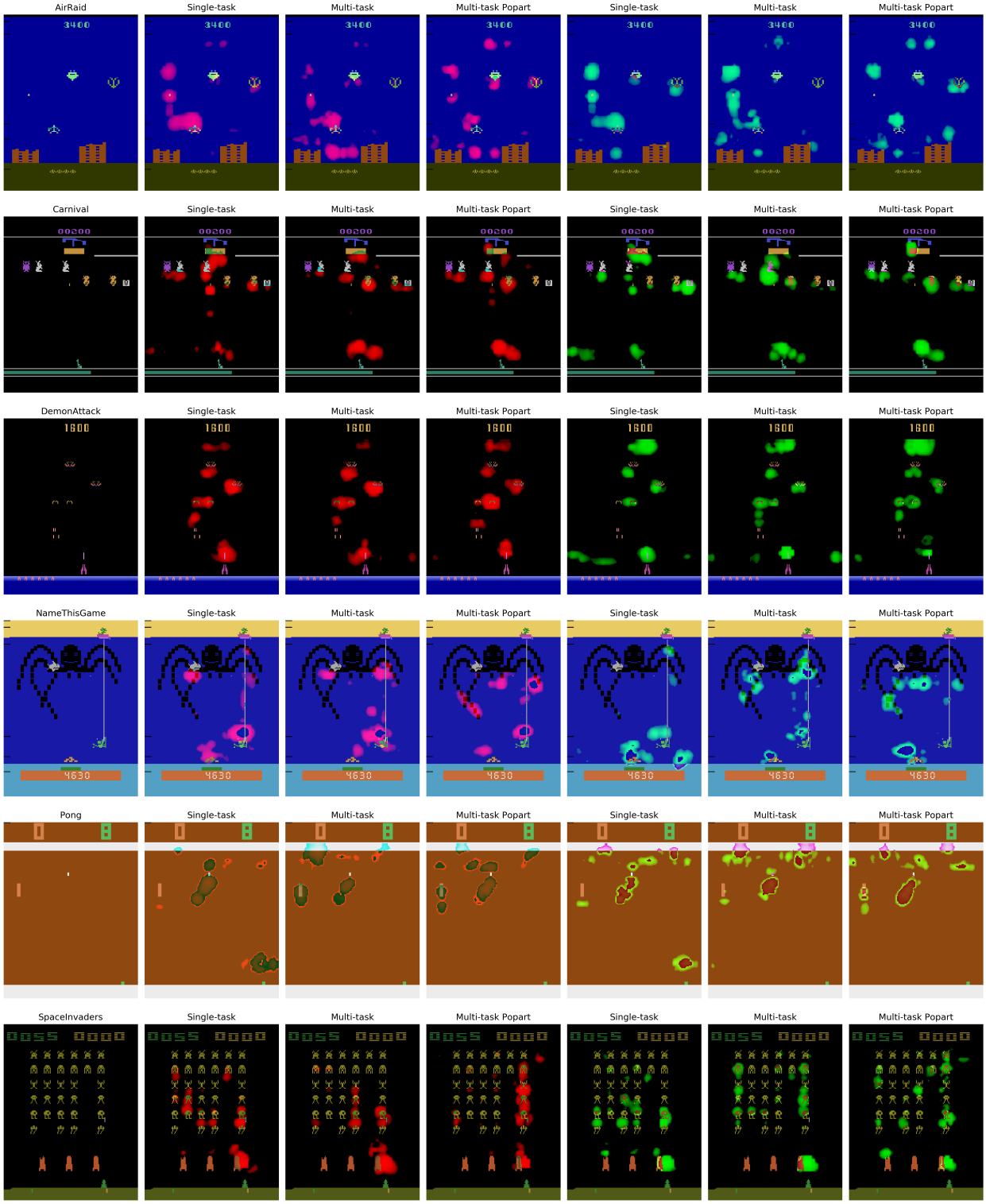
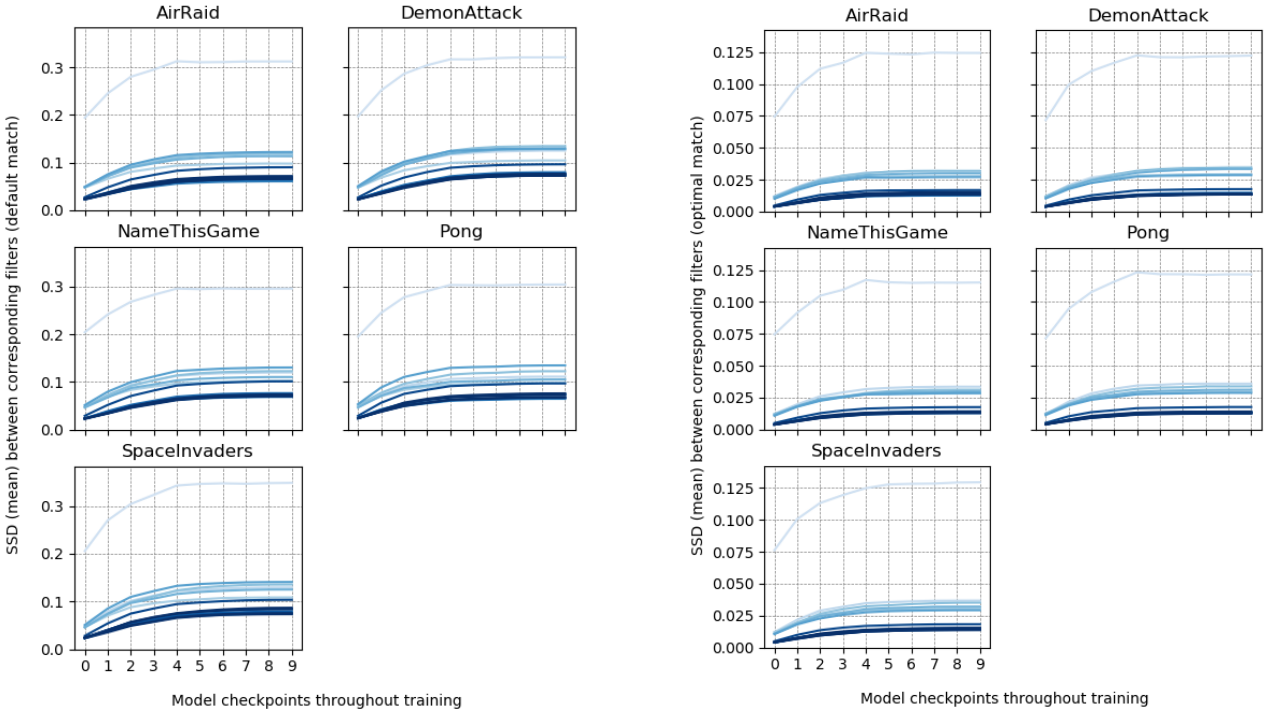


Fig. 8: Still images from the saliency videos. The policy saliency is drawn in red and the baseline saliency is shown in green. For the PopArt results, the task-specific baseline is shown. Despite the fact that the three agents exhibit very different game plans leading to measurably different overall performance, the saliency maps are surprisingly similar, with the differences being very local in nature. Note that the normalised and denormalised baselines obtained within the PopArt approach lead to the same saliency image.

G. CNN filter comparison

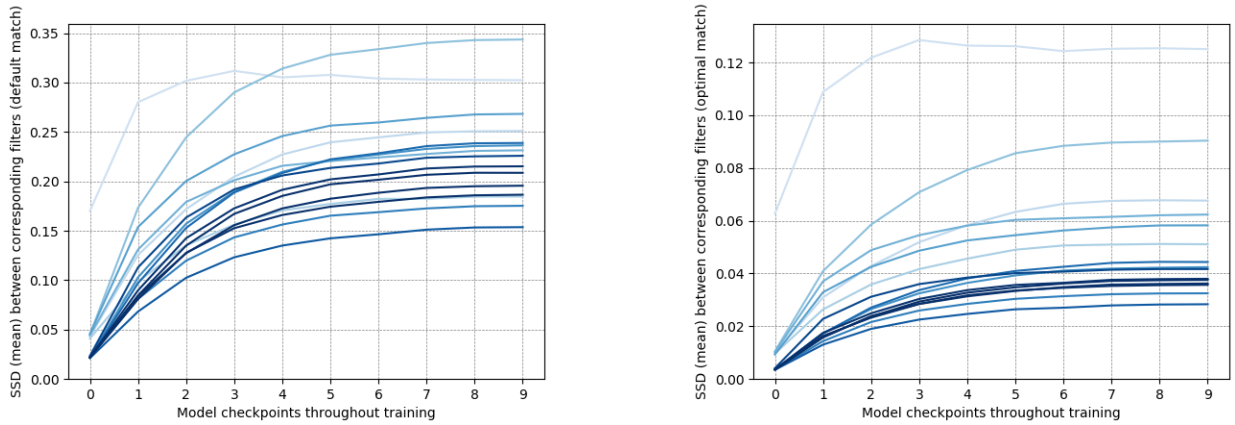
Figures 9 through 12 provide plots for the CNN filter comparison analysis done in Section III-D.



(a) Default filter matching (SSD computed between corresponding filters of both networks).

(b) Optimal filter matching (SSD computed between filter pairs selected s.t. the sum over all SSDs in the layer is minimised).

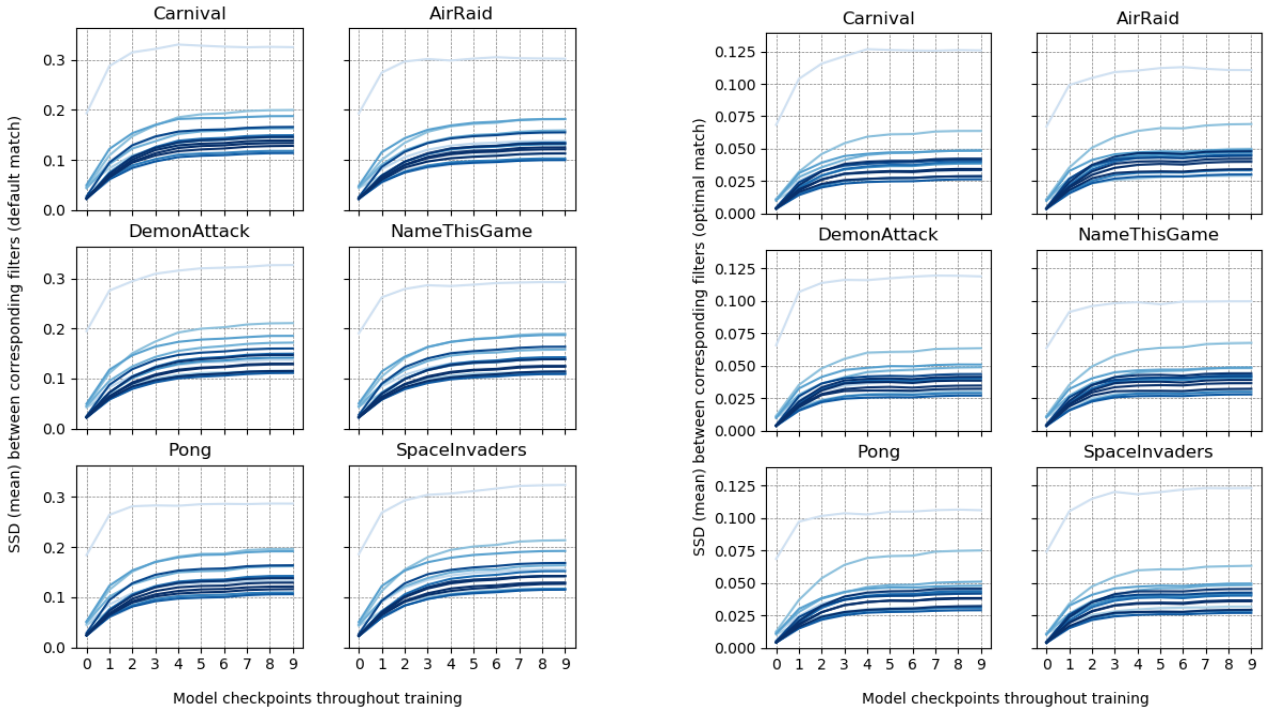
Fig. 9: The plots show the differences between the 15 convolutional layers of the model trained only on the Carnival environment with those trained only on the other specified environments throughout training. Each line shows the mean SSD between the filters of a layer, where lighter blue corresponds to earlier layers in the network. One can see that the first layer (lightest blue) varies much more between networks in both cases and (especially for optimal matching) that earlier layers generally have larger mean SSD.



(a) Default filter matching.

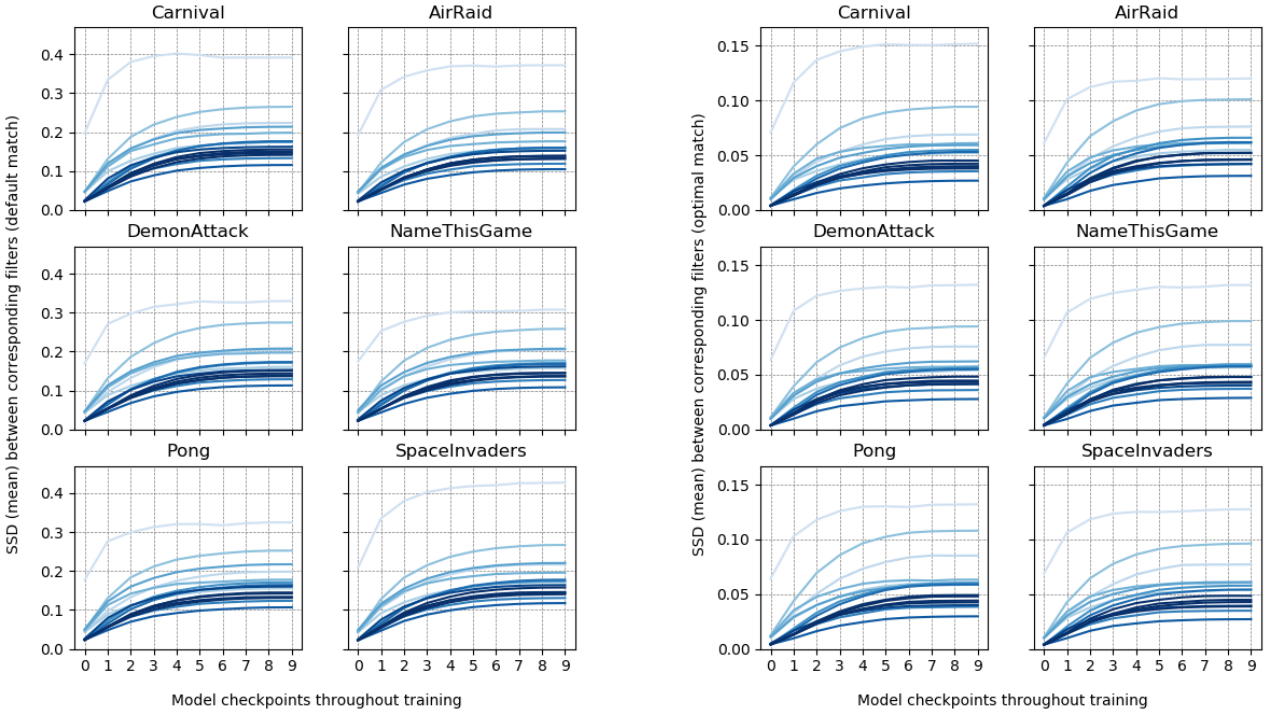
(b) Optimal filter matching.

Fig. 10: Here the differences between the vanilla multi-task model and the multi-task model with PopArt are shown. Unlike with the single-task comparison in Fig. 9 layers are not separated clearly by mean SSD. Additionally, the values for the first layer are not as clearly higher when using default matching.



(a) Default filter matching. *(b) Optimal filter matching.*

Fig. 11: The plots show the differences between the vanilla multi-task model and all single-task models. Here the first layer also has a significantly higher mean SSD for both types of matching.



(a) Default filter matching. *(b) Optimal filter matching.*

Fig. 12: The plots show the differences between the multi-task model with PopArt and all single-task models. The first layer again has higher mean SSD than the other layers, but the values vary more between the different models. In general the values are also higher compared to Fig. 11.

H. Action Distributions

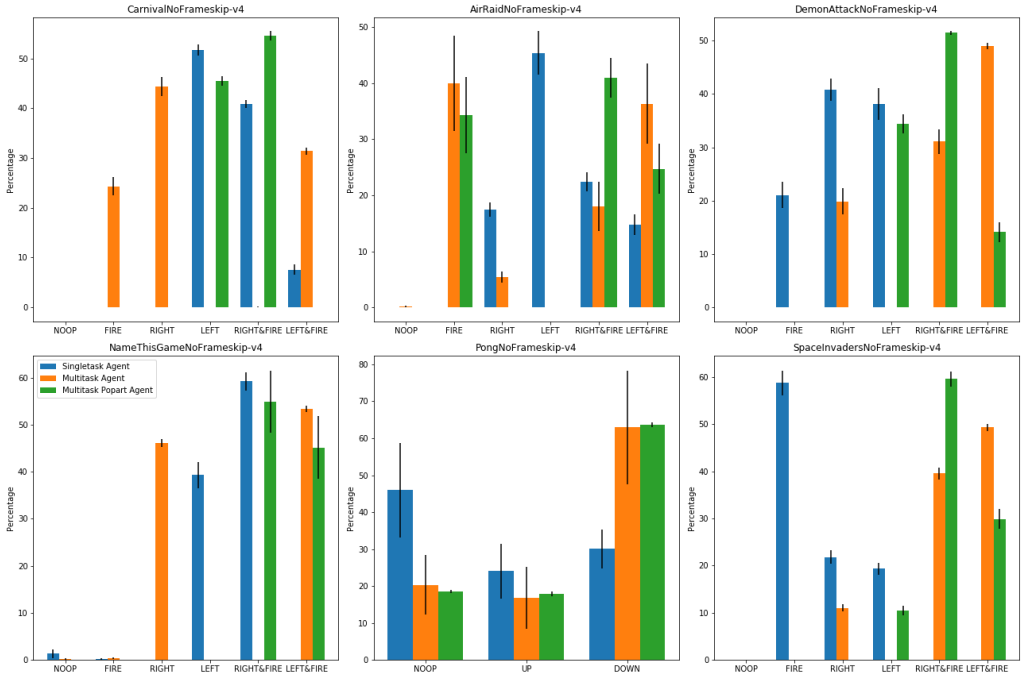


Fig. 13: The action distributions for the agents on the different game environments. Each agent played 50 games on every environment and the mean and standard deviation were computed over all games played. This figure shows all distinct actions an agent could take separately.

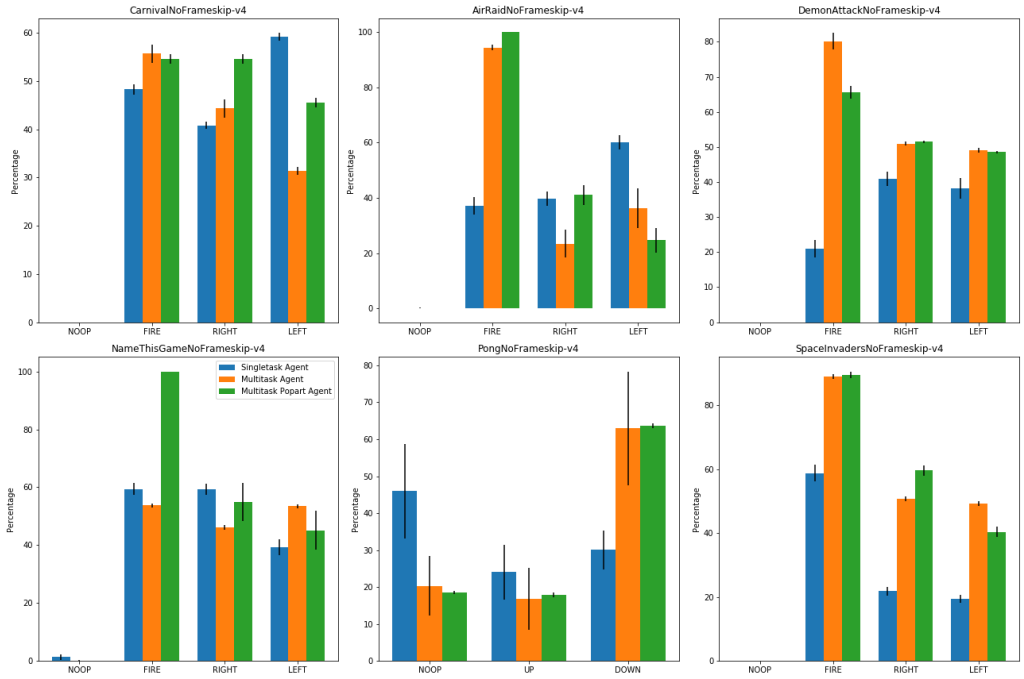


Fig. 14: This figure shows the same results as Fig. 13 however actions which have multiple effects such as RIGHT&FIRE now count both towards RIGHT and FIRE and are no longer shown separately. For this reasons the sum of the percentages over all actions taken per agent is no longer necessarily 100.