Im*pool*a: The Power of Average Pooling for Image-Based Deep Reinforcement Learning

Raphael Trumpp, Ansgar Schäfftlein, Mirco Theile, Marco Caccamo

Keywords: Network architecture, network scaling, image encoder, Procgen Benchmark

Summary

As image-based deep reinforcement learning tackles more challenging tasks, increasing model size has become an important factor in improving performance. Recent studies achieved this by focusing on the parameter efficiency of scaled networks, typically using Impala-CNN, a 15-layer ResNet-inspired network, as the image encoder. However, while Impala-CNN evidently outperforms older CNN architectures, potential advancements in network design for deep reinforcement learning-specific image encoders remain largely unexplored. We find that replacing the flattening of output feature maps in Impala-CNN with global average pooling leads to a notable performance improvement. This approach outperforms larger and more complex models in the Procgen Benchmark, particularly in terms of generalization. We call our proposed encoder model Im*pool*a-CNN. A decrease in the network's translation sensitivity may be central to this improvement, as we observe the most significant gains in games without agent-centered observations. Our results demonstrate that network scaling is not just about increasing model size—efficient network design is also an essential factor.

Contribution(s)

1. This work proposes the Impoola-CNN as image encoder for image-based deep reinforcement learning (DRL). Impoola-CNN is built upon the widely used Impala-CNN and enhances its network architecture by leveraging global average pooling (GAP).
   Context: The state-of-the-art Impala-CNN image encoder does not utilize GAP. In contrast, GAP is used in many popular network architectures in computer vision (He et al., 2016; Xie et al., 2017; Huang et al., 2017; Hu et al., 2018; Liu et al., 2022).
2. Our analysis for the full Procgen Benchmark demonstrates that Impoola-CNN excels at generalization, especially in environments without agent-centered observations. We show that Impoola-CNN outperforms other works on scaled networks in DRL.
   Context: The Procgen Benchmark (Cobbe et al., 2020) allows for the evaluation of generalization capabilities of image-based DRL agents, which is hard to assess in Atari games.
3. We identify reduced translation sensitivity in Impoola-CNN as a key distinction from Impala-CNN. Moreover, we find that Impoola-CNN is affected by fewer dormant neurons.
   Context: GAP reduces translation sensitivity (Lin, 2013) and is considered a strong inductive bias in computer vision. Sokar et al. (2023) identified the dormant neuron phenomenon, i.e., a large fraction of neurons yielding near-zero output during, as a cause of wide-ranging performance decrease in scaled networks in DRL.

Im*pool*a: The Power of Average Pooling for Image-Based Deep Reinforcement Learning

Raphael Trumpp[1], **Ansgar Schäfftlein**[1], **Mirco Theile**[1], **Marco Caccamo**[1]
{raphael.trumpp,ansgar.schaefftlein,mirco.theile,mcaccamo}@tum.de

[1]**TUM School of Engineering and Design, Technical University of Munich, Germany**

Abstract

As image-based deep reinforcement learning tackles more challenging tasks, increasing model size has become an important factor in improving performance. Recent studies achieved this by focusing on the parameter efficiency of scaled networks, typically using Impala-CNN, a 15-layer ResNet-inspired network, as the image encoder. However, while Impala-CNN evidently outperforms older CNN architectures, potential advancements in network design for deep reinforcement learning-specific image encoders remain largely unexplored. We find that replacing the flattening of output feature maps in Impala-CNN with global average pooling leads to a notable performance improvement. This approach outperforms larger and more complex models in the Procgen Benchmark, particularly in terms of generalization. We call our proposed encoder model Im*pool*a-CNN. A decrease in the network's translation sensitivity may be central to this improvement, as we observe the most significant gains in games without agent-centered observations. Our results demonstrate that network scaling is not just about increasing model size—efficient network design is also an essential factor. We make our code available at https://github.com/raphajaner/impoola.

1 Introduction

Recent works on deep reinforcement learning (DRL) have revealed that apart from algorithmic improvements, considerable performance increases can come from the network architecture and training approach of the used deep neural networks (DNNs). Notably, the Impala-CNN model (Espeholt et al., 2018), a 15-layer convolutional neural network (CNN) with residual connections, outperforms the previously widely used Nature-CNN[1] as image encoder for image-based DRL substantially (Cobbe et al., 2020; Schwarzer et al., 2023; Obando-Ceron et al., 2024a). However, raising the parameter count of DNNs in DRL does not obey scaling laws for better performance as found in other areas in deep learning (DL) (Kaplan et al., 2020; Zhai et al., 2022), e.g., scaling ResNets to up to 152 layers improves performance for image classification (He et al., 2016).

There is high interest in finding methods for network scaling in image-based DRL in recent studies (Nikishin et al., 2022; Schwarzer et al., 2023; Sokar et al., 2023; Obando-Ceron et al., 2024a;b). Most of these works use the aforementioned Impala-CNN as the image encoder, typically scaling the network's width by increasing the output channels per Conv2d layer by a factor τ. Another line of work (Sinha et al., 2020; Lee et al., 2024) has emphasized the impact of improved network design in particular to scale fully connected networks in regular DRL. Similar design improvements for image-based DRL are of high practical appeal, primarily due to the prominence of end-to-end learning in robotic applications (Yang et al., 2021; Funk et al., 2022; Trumpp et al., 2023).

While experimenting with gradual magnitude pruning (Obando-Ceron et al., 2024a) for DRL, we accidentally reduced only the Linear layer after the flattened feature maps in the Impala-CNN to

[1]The CNN model used by Mnih et al. (2015), which consists of three Conv2d layers with $\{32, 64, 64\}$ filters.

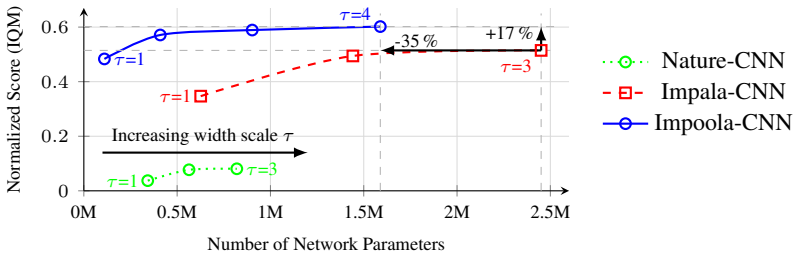


Figure 1: Impoola-CNN shows higher generalization over the full Procgen Benchmark than the Nature-CNN (Mnih et al., 2015) and Impala-CNN (Espeholt et al., 2018) image encoders when scaling the network width τ. By reducing the encoder's output dimension through GAP in Impoola-CNN, the number of weights in its Linear layers is reduced and, subsequently, the total parameters. The networks use a base configuration of $\{16, 32, 32\}$ filters per block. The presented results are obtained by evaluating the PPO agent on testing levels after 25M training steps. Normalized scores are aggregated as IQM across environments with 5 independent runs each, using different seeds.

a tiny fraction—the agent still performed well. This finding motivated us to analyze the effect of the Flatten layer in more detail, analyzing the dormant neuron distribution in scaled Impala-CNN encoders. Sokar et al. (2023) identified the dormant neuron phenomenon, i.e., a large fraction of neurons yielding near-zero output during DRL training, as a potential cause hindering wide-ranging performance gains through network scaling. After training a proximal policy optimization (PPO) (Schulman et al., 2017) agent in the Procgen Benchmark (Cobbe et al., 2020) using the Impala-CNN, we found that the total amount of dormant neurons is especially prominent in the Linear layer *after* the flattened features of the CNN-based encoder. Notably, this layer also has a very high fraction of the network's overall weights placed due to the high-dimensional embedding created by the Flatten layer. For image input of 64×64 pixels, the Impala-CNN has a total of 626,256 parameters, of which 83.76 % are in this Linear layer.

The Power of Average Pooling: We hypothesize that flattening the output feature maps in the CNN-based encoder is a root for training instabilities in image-based DRL as it creates a high-dimensional embedding. This hypothesis is in parallel to the results of Sokar et al. (2024) on soft mixture-of-experts (SoftMoE), discovering that tokenizing the feature maps, which replace the Flatten layer, is key for the performance gain. In reference to this, we discover a significant architectural difference between the Impala-CNN and standard ResNet models (He et al., 2016): There is a global average pooling (GAP) layer placed *before* the block of Linear layers in standard ResNets. This GAP reduces the feature maps to single values, ensuring input size independence. Moreover, this aggregates spatial information for reduced translation sensitivity while leading to a low-dimensional representation, potentially also enhancing gradient flow to earlier layers. Motivated by these benefits, we propose extending the Impala-CNN with a GAP layer, naming the resulting model Im*pool*a-CNN. While this modification may seem *subtle*, we show in Figure 1 that the Impoola-CNN outperforms the Impala-CNN for the Procgen Benchmark substantially while making efficient use of increased network widths. Our results emphasize the value of GAP in image-based DRL with scaled networks.

Our main contributions are the following:

- We identify architectural constraints in the Impala-CNN and propose the improved Im*pool*a-CNN image encoder, which unlocks performance gains through efficient network scaling.
- We provide extensive experiments for the full Procgen Benchmark with PPO and deep Q-network (DQN) agents. Our results show that our largest tested Impoola-CNN improves generalization in Procgen by 17 % while using 35 % fewer parameters than Impala-CNN.
- Our analysis investigates the effect of the GAP layer on the network dynamics, identifying its decreased translation sensitivity as a characteristic quality of Impoola-CNN.
- The used code can be accessed at `https://github.com/raphajaner/impoola`.

2 Related Work

Network Scaling in Deep RL: For many control applications, the typically used fully connected networks do not or only marginally improve performance when scaling their parameter count (Henderson et al., 2018). However, recent works (Sinha et al., 2020; Bjorck et al., 2021; Nauman et al., 2024) demonstrate that gains can be unlocked by improving the network architecture itself first before scaling, e.g., by introducing a residual block (Lee et al., 2024). Similarly in image-based DRL, updating the standard Nature-CNN (Mnih et al., 2015) encoder to the modern Impala-CNN model (Espeholt et al., 2018) yielded significant performance improvements for Atari (Schwarzer et al., 2023; Song et al., 2020) and Procgen games (Cobbe et al., 2019). Song et al. (2020) compare different image encoder models in DRL, highlighting the performance of Impala-CNN as models perform very differently in DRL than supervised learning. Further performance gains for the Impala-CNN model by scaling the network width are the subject of recent studies. Nikishin et al. (2022); D'Oro et al. (2022); Schwarzer et al. (2023); Sokar et al. (2023) stabilize training through periodic reinitialization of the full network or neurons. Obando-Ceron et al. (2024a) show that performance for value-based DRL is improved by unstructured gradual magnitude pruning during training. Obando-Ceron et al. (2024b) propose replacing the encoder's Linear layer with a SoftMoE layer. Further analysis by Sokar et al. (2024) identifies the tokenization of the feature maps for SoftMoEs, rather than the use of multiple experts, to drive the performance gain found in Obando-Ceron et al. (2024b).

Translation Invariance: A strong inductive bias in computer vision is to incorporate *translation invariance*, i.e., invariance to the shifts of an object in the input image. Intuitively, GAP (Lin, 2013) induces translation invariance, as the average operation is invariant to position (Mouton et al., 2020). However, Conv2d layers are not fully equivariant due to subsampling and boundary effects (Mouton et al., 2020), e.g., they can exploit zero-padding and image orders to learn absolute positions (Islam* et al., 2020; Kayhan & Gemert, 2020). Thus, CNNs are not fully translation invariant even if GAP is used. However, networks with GAP are typically less sensitive to spatial translations of the input (Lin, 2013). Translation sensitivity maps are a measure to quantify this property (Kauderer-Abrams, 2017). GAP is still effective in practice and used in many popular network architectures (He et al., 2016; Xie et al., 2017; Huang et al., 2017; Hu et al., 2018; Liu et al., 2022).

Generalization in Deep Reinforcement Learning: Using the same environment for both training and testing results in high overfitting of DRL agents (Zhang et al., 2018; Cobbe et al., 2019). Overfitting in DRL may be associated with a loss of network plasticity (Nikishin et al., 2022; Sokar et al., 2023) and generalization is theoretically closely related to invariances (Lyle et al., 2019). The Procgen Benchmark (Cobbe et al., 2020) introduces various procedurally generated environments to quantify generalization. A number of invariance-based methods have been shown to facilitate generalization for Procgen environments, ranging from auxiliary losses (Raileanu & Fergus, 2021) to data augmentation (Lee et al., 2020; Kostrikov et al., 2020; Raileanu et al., 2021).

3 Background

3.1 Deep Reinforcement Learning

The iterative optimization in model-free DRL is formalized by a Markov decision process (MDP) with tuple $(\mathcal{S}, \mathcal{A}, \mathrm{T}, \mathrm{R}, \gamma)$. Here, $\mathcal{S}$ and $\mathcal{A}$ represent the state and action spaces, respectively, while the transition function $\mathrm{T} : \mathcal{S} \times \mathcal{A} \to \mathcal{P}(\mathcal{S})$ defines the probability distribution over the next state given the current state and action. The reward function is defined as $\mathrm{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and γ is a discount factor. The mapping $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ is called a stochastic action policy. A DNN with weights θ parameterizes the policy π_θ in DRL. The optimal policy π^*_θ maximizes the expected return $V_{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{R}(s_t, a_t) \mid s_0 = s\right]$.

Q-Network Methods: These DRL methods are typically based on an estimate of the Q-value function $Q_{\pi_\theta}(s, a) := \mathbb{E}_{\pi_\theta}\left[\sum_{t=0}^{\infty} \gamma^t \mathrm{R}(s_t, a_t) \mid s_0 = s, a_0 = a\right]$. This function can be learned iteratively by temporal difference learning (Sutton, 1988) and bootstrapping the current Q-value esti-

mate. DQN (Mnih et al., 2015) implements this by training a DNN with loss function $L(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(r + \gamma \max_{a'} Q(s', a'; \bar{\theta}^-) - Q(s, a; \bar{\theta})\right)^2\right]$ where transitions $(s, a, r, s') \sim \mathcal{D}$ are sampled from the experience replay buffer $\mathcal{D}$ and by using a target network with θ^- as delayed copies of θ. Actions are obtained greedily by $a = \arg\max_a Q(s, a; \theta)$.

Actor-Critic Methods: In addition to a critic network, e.g., $V(s; \phi)$ that estimates the state value, the action policy is defined as a dedicated actor network that can be directly optimized towards an optimization goal. PPO (Schulman et al., 2017) is an *on*-policy DRL method, where the weights θ are updated with respect to the advantage function $A(s, a) = Q(s, a) - V(s)$. The generalized advantage estimate (GAE) (Schulman et al., 2015) is the common choice to estimate $A(s, a)$. The loss (clip version) of the PPO actor for a transition tuple $e = (s, a, r, s')$ of a trajectory $\tau = \{e, e', ...\}$ is given by $L(\theta) = \mathbb{E}_\tau\left[\min\left\{r(\theta)A, \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)A\right\}\right]$. Here, $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is the probability ratio between the old and new policy, where the hyperparameter ϵ limits their deviation.

3.2 Convolutional Neural Networks

Impala-CNN: The Impala-CNN was introduced by Espeholt et al. (2018) as a 15-layer model with residual connections specifically for encoding image inputs in DRL. The architecture combines two building blocks. As visualized in Figure 2, the ConvSequence S_j blocks consist first of a Conv2d layer with MaxPooling and ReLU activation and then 2 subsequent ResBlock blocks as $S_j : \{C_j \to P \to R_{0,j} \to R_{1,j}\}$; the ResBlock blocks $R_{i,j}$ are based on two Conv2d layers with ReLU activation and a residual connection. The vanilla Impala-CNN stacks three ConvSequences $\{S_0, S_1, S_2\}$ with each block having the same amount of output channels $\{16, 32, 32\}$; scaled network versions multiply this configuration by a width scaling factor τ. The original implementation by Espeholt et al. (2018) adds a Linear layer with 256 neurons to project the flattened feature map encodings e to a fixed-dimension encoder output z as part of the model.

Pooling Layers: This fundamental operation in CNNs reduces the spatial dimensions of feature maps $\mathbf{x} \in \mathbb{R}^{C\times H\times W}$ where C, H, and W represent channels, height, and width, respectively. Average pooling computes the mean within a $k \times l$ window and stride s as

$$\mathbf{y}(c, i, j) = \frac{1}{k \cdot l} \sum_{p=0}^{k-1} \sum_{q=0}^{l-1} \mathbf{x}\left(c, s \cdot i + p, s \cdot j + q\right),$$

where $\mathbf{y}$ is the pooled output. Typically, the window is square $k = l$, and the stride equals the window size. *Global* average pooling (GAP) (Lin, 2013) reduces the spatial dimensions of feature maps $\mathbf{x}$ to a single value per map $\mathbf{y} \in \mathbb{R}^C$ by setting $k = H, l = W$. GAP reduces a network's translation sensitivity (Lin, 2013) and simplifies its architecture by inducing no additional learnable parameters, facilitating scalable and efficient architectures. Common machine learning frameworks provide adaptive implementations of Pooling layers where only the required output map dimension must be defined. We refer to this by writing `XPool(n,m)`, which calculates the necessary s, k, l, and padding such that the output feature maps are of dimension $\mathbf{y} \in \mathbb{R}^{C\times n\times m}$.

4 Impoola-CNN

The Impoola-CNN is a convolutional neural network with an intended use as image encoder for image-based DRL. Typically, the encoder output z is fed into Linear layer prediction heads, e.g., an actor and critic head for PPO. We discuss the essential design choices of this architecture below.

Network Design: The Impoola-CNN builds upon the Impala-CNN from Espeholt et al. (2018) and adds a GAP layer after the last Conv2d layer as shown in Figure 2. This change has a vast influence on the model architecture, as it reduces feature maps to single entries. As listed in Table C.7, the scaled Impala-CNN ($\tau = 2$) consists of 1,441,680 learnable parameters for an input image of 64x64, of which 72.75 % are located in the encoder's last Linear layer. In contrast, for the same width

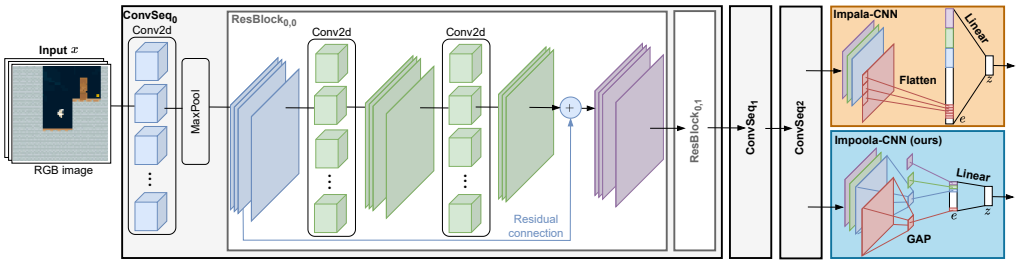


Figure 2: The Impala/Impoola-CNN models encode input images x through a ResNet design, consisting of stacked ConvSeq blocks. ConvSeq blocks are built upon a first Conv2D layer, MaxPooling with stride 2, and two consecutive ResBlocks. ResBlocks are based on two Conv2d layers with a residual connection. The final feature maps are reduced to single values by using a GAP layer in Impoola-CNN, while the Impala-CNN flattens all features directly, resulting in a high-dimensional encoding e. This encoding is then projected to the encoder's output variable z by a Linear layer.

scale τ, Impoola-CNN contains 409,488 parameters which are equally distributed over the network, with 4.06 % in the Linear layer. We speculate that this balanced distribution, specifically reducing the number of Linear layer weights, contributes to the performance increase of Impoola-CNN.

Implementation Details: The Impala/Impoola-CNN encoders are deployed with an output feature dimension of $z \in \mathbb{R}^{256}$ in all experiments (Espeholt et al., 2018; Huang et al., 2022). Setting the correct learning rate η is crucial when comparing networks of different parameter counts. We searched a parameter space $\{7.5, 5.0, 3.5, 2.5, 1.0\}_{\times 10^{-4}}$ for Impala/Impoola-CNN models, using a scaled version with $\tau = 2$ as the base configuration. We found that for PPO, both models work best with a learning rate of $\eta|_{\tau=2} = 3.5_{\times 10^{-4}}$, while setting $\eta|_{\tau=2} = 1.0_{\times 10^{-4}}$ is favorable for DQN. We also tested this learning rate for scaled versions and concluded that we can obtain consistent results by adjusting the learning rate η according to the following scaling rule $\eta|_{\tau} = \eta|_{\tau=2} \cdot \frac{\tau}{2}$, which was also shown to work well in (Obando-Ceron et al., 2024a).

5 Experiments

Experiment Design: We base our analysis on the Procgen Benchmark (Cobbe et al., 2020). Our evaluation focuses on measuring the generalization of DRL agents, for which Atari games are unsuitable. Unless otherwise stated, the presented results are based on the *full* benchmark with all 16 environments. A qualitative description of the environment characteristics is given in Appendix A. The Procgen Benchmark allows for two tracks: *efficiency*, where each level is sampled from the full level distribution, while the *generalization* track restricts the levels per environment to a fixed set of 200 or 1000 levels for easy and hard settings, respectively. We follow the training recommendation of Cobbe et al. (2020) and train for 25M timesteps for the *easy* and 100M for the *hard* setting.

Evaluation Metrics: We run periodic evaluations during training, gathering the episodic returns of 2,500 episodes. We normalize episodic returns and report normalized scores S using the normalization constants from Cobbe et al. (2020) so that 1.0 corresponds to an optimal policy and 0.0 is equivalent to a random one. For statistical relevance with a reasonable computational cost, we run all experiments for each environment with 5 independent runs using different seeds. We presented results when aggregated across environments as interquantile mean (IQM) (Agarwal et al., 2021) scores and corresponding 95-% stratified bootstrap confidence interval as shaded areas.

Deep Reinforcement Learning Agents: This work uses PPO and DQN agents. Our implementations are derived from CleanRL (Huang et al., 2022) for PyTorch (Paszke et al., 2017). The actor and critic for PPO share the image encoder. Our DQN agent is extended by double Q-learning (Van Hasselt et al., 2016), multi-step rewards (Sutton, 1988), and a simplified prioritized experience replay (PER) (Schaul et al., 2015). Hyperparameters are listed in Appendix B.1. Implementation details of other methods for benchmarking are given in Appendix B.2.

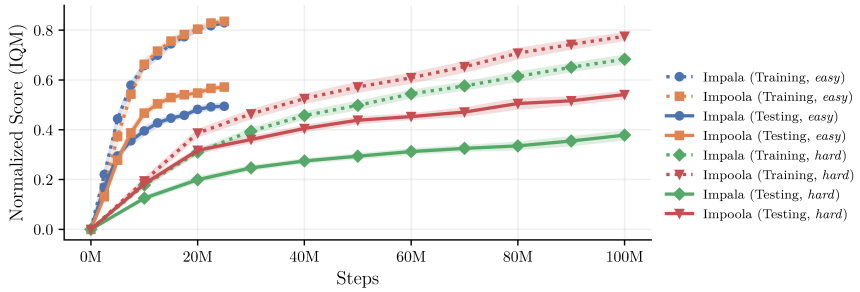
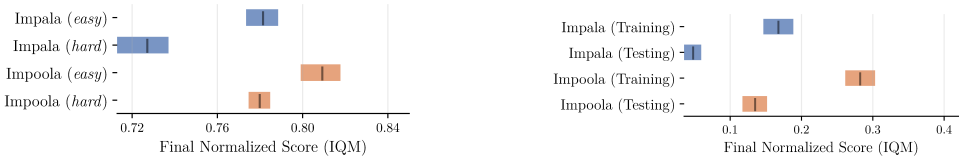


Figure 3: Generalization track for the *easy* (blue and orange) and *hard* setting (green and red) using PPO and scaled networks of $\tau = 2$. Results for the levels used during training are depicted as dotted lines; test performance on unseen levels are solid lines. Agents in easy games are evaluated every 2.5M steps; the hard game setting requires longer training, so evaluations run every 10M steps. We utilize linear learning rate annealing in the hard setting to stabilize the long training duration.



(a) Efficiency tracks for PPO.

(b) Generalization track (*easy* setting) for DQN.

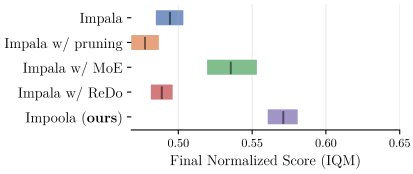
Figure 4: Further evaluation of Impala-CNN and Impoola-CNN encoders ($\tau = 2$). We test PPO additionally for the efficiency track (**left**) and show results for DQN-based agents (**right**).

5.1 Evaluation

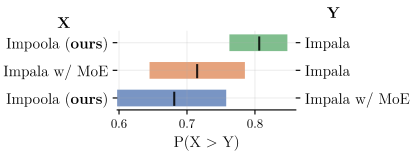
Our initial experiment evaluates the effect of scaling the width of Impala-CNN and our proposed Impoola-CNN from $\tau = 1$ to 3 and 4, respectively. The results are shown in Figure 1 for PPO for the full Procgen Benchmark. First, it can be seen that Impoola-CNN has substantially fewer total parameters at the same width levels τ. Overall, the largest tested Impoola-CNN achieves a 17 % higher IQM score for *generalization* with 35 % less parameters than the Impala-CNN. These results demonstrate the efficacy of the proportional weight distribution in the Impoola-CNN, as a high parameter count in the Linear layer does not directly translate into higher performance.

Figure 3 presents the results for the *generalization* track with PPO in detail for $\tau = 2$, comparing the scores for the *easy* and *hard* settings. In addition to the usually only evaluated testing performance on the full distribution of levels, we display results for the restricted set of levels used in training. In the *easy* setting, the performance of Impala-CNN on *training* levels trails the proposed Impoola-CNN only by a limited margin. However, Impala-CNN's learned action policies generalize worse to unseen *testing* levels than Impoola-CNN. This trend persists for the *hard* setting as Impoola-CNN improves performance on unseen *testing* levels but additionally outperforms Impala-CNN on the restricted *training* levels. We posit that in the *easy* setting, Impala-CNN is able to capture the fundamental game mechanics for high *training* performance. However, in contrast to Impoola-CNN, it fails to scale to the increased task complexity in *hard*. We hypothesize that its GAP layer encourages the agent to learn more universal feature representations, which enable better adaptation to new levels and facilitate learning under challenging conditions.

We conduct additional experiments, and first show results for PPO also in the *efficiency* track, i.e., training on the full level distribution, in Figure 4a. Impala-CNN's performance drops from the *easy* to *hard* setting meaningfully, while Impoola-CNN's remains more consistent. This result is interesting since the *efficiency* setting may favor larger networks due to their larger hypothesis space. Nevertheless, the Impoola-CNN agent, consisting of 409,488 parameters, outperforms the Impala-CNN, which has approximately a 3x higher parameter count of 1,441,680. Finally, Figure 4b shows another



(a) Benchmarking with other methods.



(b) Probability of improvement that Algorithm X (left) performs better than Algorithm Y (right).

Figure 5: Benchmarking the Impoola-CNN ($\tau = 2$) against other methods for the generalization track (*easy*) across the full Procgen Benchmark. We extend the Impala-CNN ($\tau = 2$) by gradual magnitude pruning (Obando-Ceron et al., 2024a), SoftMoE (Obando-Ceron et al., 2024b), and ReDo (Sokar et al., 2023) for comparison. We present the final IQM scores (**left**) and the probability of improvement (**right**). The probability of improvement is a measure to estimate how likely it is that an algorithm outperforms another one in a single environment on average.

experiment with DQN for the *generalization* track (*easy*). While DQN achieves overall substantially lower performance than PPO in this track, the results affirm the previous trends. Impoola-CNN not only improves testing performance for DQN agents but also training results. An experiment with adding GAP to the classical Nature-CNN can be found in the Appendix D, but with inconclusive results due to the Nature-CNN's overall weak performance and potential underparametrization.

5.2 Benchmark

A comparison of Impoola-CNN to other recent methods related to network scaling is given in Figure 5a for the *generalization* track (*easy*). It is evident that the gradual magnitude pruning method (Obando-Ceron et al., 2024a) and ReDo (Sokar et al., 2023) do not translate to performance increases. We presume that this situation is due to the fact that these methods were initially developed in the context of value-based DRL, potentially magnified by our relatively short training period of 25M time steps in the easy setting. However, using SoftMoE with 10 experts (Obando-Ceron et al., 2024b) leads to a noteworthy increase compared to the vanilla Impala-CNN. This observation is particularly significant, as Sokar et al. (2024) attribute the performance improvement of the method primarily to the tokenization of the encoder's output feature maps. This approach aligns conceptually to the GAP layer in Impoola-CNN, as both mitigate the need for an excessively large Linear layer. Impoola-CNN achieves the highest performance gains among the discussed methods and demonstrates the greatest likelihood of improvement over Impala-CNN in Figure 5b.

5.3 Understanding the Power of Average Pooling

Introducing a GAP layer in Impoola-CNN has two clear implications. First, GAP is well-known to reduce translation sensitivity in CNNs (Lin, 2013). Second, the feature map encoding e is reduced to the number of output feature maps, thus decreasing the connections to the subsequent fully connected Linear layer. To understand these implications better, we evaluate related alternative approaches and extend the Impala-CNN instead of a GAP layer, i.e., AvgPool(1,1), by AvgPool(2,2), MaxPool(1,1), or add a fourth ConvSeq block. Moreover, we test a depthwise Conv2d layer, which creates 1×1 feature maps.

Translation Sensitivity: We quantify the translation sensitivity of the actor network, following a similar approach to Kauderer-Abrams (2017), by measuring the change in the action probability distribution when translating the input image by (x,y) pixels.[2] The corresponding translation maps are displayed in Figure 6. Impala-CNN exhibits substantial variations in action distribution when

[2]We measure translation sensitivity as the L1 distance between action probabilities from a Categorical distribution, computed from the actor's output logits. This metric quantifies changes in action probabilities due to image translation and ensures comparability across networks via Softmax normalization. Only the agent and non-player characters are shifted against a stationary background to prevent artifacts. See Appendix B.3 for details.

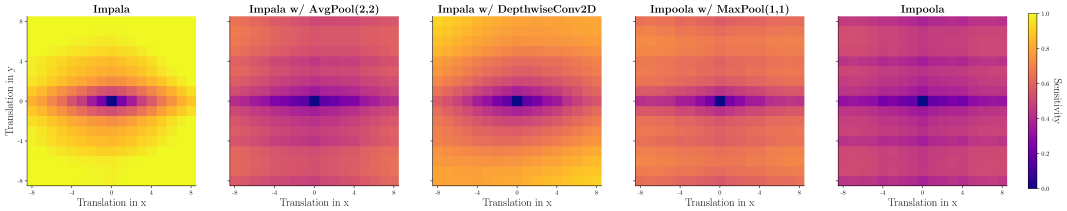


Figure 6: Translation sensitivity maps for the actor network of PPO in Bigfish (non-agent-centered game). The maps depict at each pixel (x,y) the corresponding sensitivity score that measures how the action probability distribution deviates when translating the input image by (x,y) pixels compared to the untranslated image. As the x and y axes are centered around 0, the center pixel's sensitivity score is always 0 as it references the untranslated image. Bright yellow colors indicate high translation sensitivity, i.e., the action probabilities differ substantially when translating the input by (x,y).

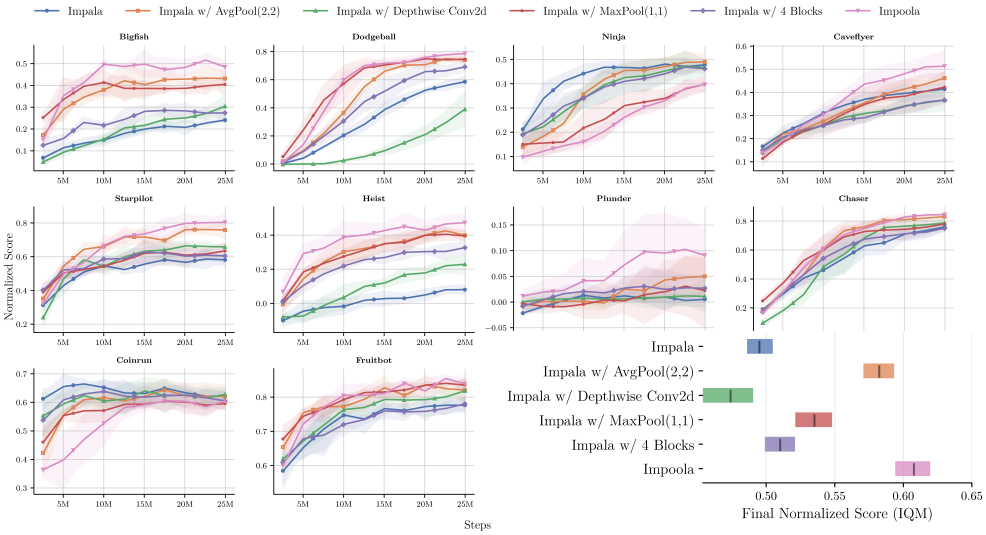


Figure 7: Ablation study of Impoola-CNN ($\tau = 2$) with results for generalization (*easy*) using a subset of 10 Procgen games. Note that Caveflyer, Coinrun, and Ninja are environments with agent-centered observations. We show the final IQM scores (**bottom right**) and training curves with mean and standard deviation (**rest**).

the input image is shifted, leading to inconsistent actions. In contrast, Impoola-CNN-based agents demonstrate reduced sensitivity to positional shifts, maintaining a stable action profile regardless of the absolute positions of entities in the image observation. While MaxPooling(1,1) reduces the output feature maps to single entries like GAP, the $\max$-operation appears to reduce the translation sensitivity less meaningfully. We also see that AvgPool(2,2) is more sensitive than Impoola-CNN; the average pooling to (2,2) feature maps retains some spatial information. Given that Depthwise Conv2D exhibits high translation sensitivity and performs substantially worse than models incorporating AvgPool(), despite identical parameter counts in the Linear layers, our hypothesis is further supported that translation insensitivity is a primary contributor to the observed performance gains.

Agent-Centered Observations: We discuss the influence of translation sensitivity in relation to the characteristics of the Procgen Benchmark games. The game-specific reward curves are presented in Figure 7. First, it can be seen that Impala-CNN only exhibits favorable performance in the Coinrun and Ninja environments, which have agent-centered observations, especially early in training. However, we find no advantage for Impala-CNN in Caveflyer, despite being an environment with agent-centered observations, and the training curve in Coinrun indicates overfitting. As shown in Figure 8, the advantage of Impala-CNN in environments with agent-centered observations diminishes in the

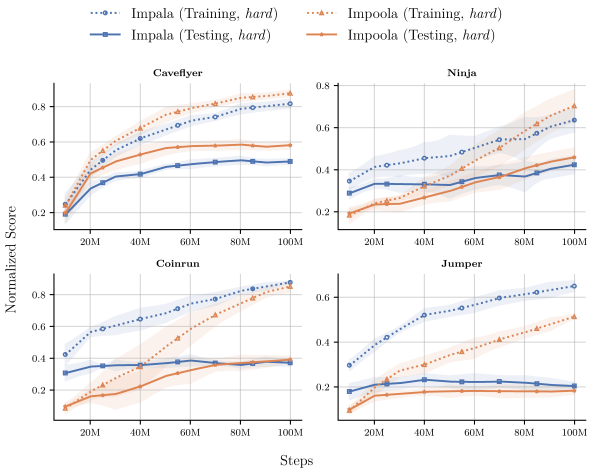


Figure 8: Results for games with agent-centered observations in the *generalization* track (*hard*) for PPO ($\tau = 2$).

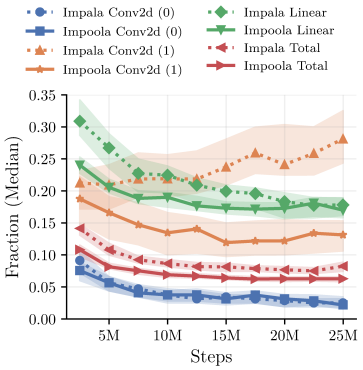


Figure 9: Fraction of dormant neurons per layer throughout training for the first two Conv2d layers of the encoder ($\tau = 2$), its output Linear layer, and for the total network.

hard setting. Impoola-CNN achieves comparable final performance for these environments as the generalization performance of Impala-CNN saturates early. We reason that positional information may only be a helpful inductive bias early in training for agent-centered games. However, it is also prone to overfitting, leading to networks that show weaker generalization than networks that pose low translation sensitivity. The results for AvgPool(2,2) highlight an interesting trade-off as seen in Figure 7. While falling short of Impoola-CNN in non-agent-centered games, it outperforms Impala-CNN for them and meets Impala-CNN's performance in agent-centered games.

Dormant Neurons: When monitoring the fraction of dormant neurons (Sokar et al., 2023) during PPO training in the *generalization* track (*easy*), we see in Figure 9 that the total number of dormant neurons decreases during training for both encoder models. While the Linear layer in Impala-CNN has a higher initial dormant fraction, it decreases during training to the same fraction. However, we find a distinct difference for the *second* Conv2d layer (1), which has no residual connection such as the first Conv2d layer (0): while Impoola-CNN's count decreases, Impala-CNN's dormant neuron count increases here during training. We reason that this may be attributed to better training stability and gradients to this early network layer in Impoola-CNN, as the Impala-CNN unbalanced distribution of network weights along the network depths might reduce effective gradient flow.

6 Conclusion and Future Work

This work introduces the Impoola-CNN model, an improved image encoder for DRL that is based on the widely used Impala-CNN architecture. Our advancement is based on the introduction of a GAP layer to the Impala-CNN, which has a two-fold implication. First, it leads to a reduction of required weights in the encoder's Linear layer and creates a balanced weight distribution along the network's depth. Second, we find that this change effectively reduces the network's translation sensitivity. Our experiments for the full Procgen Benchmark show that Impoola-CNN leads to a significant performance increase, most prominent in environments without agent-centered observations. We also find that the stronger dependence on absolute positions of Impala-CNN may become detrimental during the longer training for the hard setting. We hypothesize that its GAP layer encourages the agent to learn more universal feature representations, which enable better adaptation to new levels and facilitate learning under challenging conditions.

For future work, we plan to conduct further experiments outside the Procgen Benchmark. While results for other game-inspired environments, e.g., Atari games, would double-down our results, we see a stronger need for evaluation in real-world image-based DRL applications, e.g., automated driving (Trumpp et al., 2023) or vision-guided quadrupedal locomotion (Yang et al., 2021).

Acknowledgments

Marco Caccamo was supported by an Alexander von Humboldt Professorship endowed by the German Federal Ministry of Education and Research.

References

Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep reinforcement learning at the edge of the statistical precipice. *Advances in neural information processing systems*, 34:29304–29320, 2021.

Nils Bjorck, Carla P Gomes, and Kilian Q Weinberger. Towards deeper deep reinforcement learning with spectral normalization. *Advances in neural information processing systems*, 34:8242–8255, 2021.

Karl Cobbe, Oleg Klimov, Chris Hesse, Taehoon Kim, and John Schulman. Quantifying generalization in reinforcement learning. In *International conference on machine learning*, pp. 1282–1289. PMLR, 2019.

Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*, pp. 2048–2056. PMLR, 2020.

Pierluca D'Oro, Max Schwarzer, Evgenii Nikishin, Pierre-Luc Bacon, Marc G Bellemare, and Aaron Courville. Sample-efficient reinforcement learning by breaking the replay ratio barrier. In *Deep Reinforcement Learning Workshop NeurIPS 2022*, 2022.

Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pp. 1407–1416. PMLR, 2018.

Niklas Funk, Georgia Chalvatzaki, Boris Belousov, and Jan Peters. Learn2assemble with structured representations and search for robotic architectural construction. In *Conference on Robot Learning*, pp. 1401–1411. PMLR, 2022.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022.

Md Amirul Islam*, Sen Jia*, and Neil D. B. Bruce. How much position information do convolutional neural networks encode? In *International Conference on Learning Representations*, 2020.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361v1*, 2020.

Eric Kauderer-Abrams. Quantifying translation-invariance in convolutional neural networks. *arXiv preprint arXiv:1801.01450v1*, 2017.

Osman Semih Kayhan and Jan C van Gemert. On translation invariance in cnns: Convolutional layers can exploit absolute spatial location. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14274–14285, 2020.

Ilya Kostrikov, Denis Yarats, and Rob Fergus. Image augmentation is all you need: Regularizing deep reinforcement learning from pixels. *arXiv preprint arXiv:2004.13649v4*, 2020.

Hojoon Lee, Dongyoon Hwang, Donghu Kim, Hyunseung Kim, Jun Jet Tai, Kaushik Subramanian, Peter R Wurman, Jaegul Choo, Peter Stone, and Takuma Seno. Simba: Simplicity bias for scaling up parameters in deep reinforcement learning. *arXiv preprint arXiv:2410.09754v1*, 2024.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

M Lin. Network in network. *arXiv preprint arXiv:1312.4400v3*, 2013.

Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11976–11986, 2022.

Clare Lyle, Marta Kwiatkowksa, and Yarin Gal. An analysis of the effect of invariance on generalization in neural networks. In *International conference on machine learning Workshop on Understanding and Improving Generalization in Deep Learning*, volume 1, 2019.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

Coenraad Mouton, Johannes C Myburgh, and Marelie H Davel. Stride and translation invariance in cnns. In *Southern African Conference for Artificial Intelligence Research*, pp. 267–281. Springer, 2020.

Michal Nauman, Mateusz Ostaszewski, Krzysztof Jankowski, Piotr Miłoś, and Marek Cygan. Bigger, regularized, optimistic: scaling for compute and sample-efficient continuous control. In *ICML 2024 Workshop: Aligning Reinforcement Learning Experimentalists and Theorists*, 2024.

Evgenii Nikishin, Max Schwarzer, Pierluca D'Oro, Pierre-Luc Bacon, and Aaron Courville. The primacy bias in deep reinforcement learning. In *International conference on machine learning*, pp. 16828–16847. PMLR, 2022.

Johan Samir Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. In value-based deep reinforcement learning, a pruned network is a good network. In *Forty-first International Conference on Machine Learning*, 2024a.

Johan Samir Obando-Ceron, Ghada Sokar, Timon Willi, Clare Lyle, Jesse Farebrother, Jakob Nicolaus Foerster, Gintare Karolina Dziugaite, Doina Precup, and Pablo Samuel Castro. Mixtures of experts unlock parameter scaling for deep RL. In *Forty-first International Conference on Machine Learning*, 2024b.

Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

Roberta Raileanu and Rob Fergus. Decoupling value and policy for generalization in reinforcement learning. In *International Conference on Machine Learning*, pp. 8787–8798. PMLR, 2021.

Roberta Raileanu, Maxwell Goldstein, Denis Yarats, Ilya Kostrikov, and Rob Fergus. Automatic data augmentation for generalization in reinforcement learning. *Advances in Neural Information Processing Systems*, 34:5402–5415, 2021.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *CoRR*, abs/1511.05952, 2015.

John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347v2*, 2017.

Max Schwarzer, Johan Samir Obando Ceron, Aaron Courville, Marc G Bellemare, Rishabh Agarwal, and Pablo Samuel Castro. Bigger, better, faster: Human-level atari with human-level efficiency. In *International Conference on Machine Learning*, pp. 30365–30380. PMLR, 2023.

Samarth Sinha, Homanga Bharadhwaj, Aravind Srinivas, and Animesh Garg. D2rl: Deep dense architectures in reinforcement learning. *arXiv preprint arXiv:2010.09163v2*, 2020.

Ghada Sokar, Rishabh Agarwal, Pablo Samuel Castro, and Utku Evci. The dormant neuron phenomenon in deep reinforcement learning. In *International Conference on Machine Learning*, pp. 32145–32168. PMLR, 2023.

Ghada Sokar, Johan Obando-Ceron, Aaron Courville, Hugo Larochelle, and Pablo Samuel Castro. Don't flatten, tokenize! unlocking the key to softmoe's efficacy in deep RL. *arXiv preprint arXiv:2410.01930v1*, 2024.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*, 2020.

Richard S Sutton. Learning to predict by the methods of temporal differences. *Machine learning*, 3:9–44, 1988.

Raphael Trumpp, Martin Büchner, Abhinav Valada, and Marco Caccamo. Efficient learning of urban driving policies using bird's eye-view state representations. In *2023 IEEE 26th International Conference on Intelligent Transportation Systems (ITSC)*, pp. 4181–4186, 2023.

Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30, 2016.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

Ruihan Yang, Minghao Zhang, Nicklas Hansen, Huazhe Xu, and Xiaolong Wang. Learning vision-guided quadrupedal locomotion end-to-end with cross-modal transformers. In *Deep RL Workshop NeurIPS*, 2021.

Xiaohua Zhai, Alexander Kolesnikov, Neil Houlsby, and Lucas Beyer. Scaling vision transformers. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12104–12113, 2022.

Chiyuan Zhang, Oriol Vinyals, Remi Munos, and Samy Bengio. A study on overfitting in deep reinforcement learning. *arXiv preprint arXiv:1804.06893v2*, 2018.

Supplementary Materials

The following content was not necessarily subject to peer review.

A Procgen Benchmark

Description: The Procgen Benchmark was developed by Cobbe et al. (2020) to test generalization and sample efficiency of DRL agents. The Benchmark consists of 16 games and allows for *hard* and *easy* game settings to balance computation demand accordingly. For the *generalization* track, a restricted fixed set of 200 levels is used for training in the *easy* setting, while all possible procedurally generated levels are used for evaluation. Cobbe et al. (2020) recommend training for 25M steps in this setting. When the *hard* setting is used, 1000 training levels are used with 100M training steps. The *efficiency* tracks do not restrict the set of training levels but use the full distribution of levels. The action space of the Procgen environments consists of 15 discrete actions. Observations are RGB images with $3 \times 64 \times 64$ pixels. No stacking of images is required, as we utilize the environments without the setting that requires memory. We show example image observations for the games in Figure A.1 and list specific game characteristics in Table A.1.

Figure A.1: All ProcGen environments depicted with a single image observation (64x64 pixels): Bigfish, Bossfight, Caveflyer, Chaser, Climber, Coinrun, Dodgeball, Fruitbot, Heist, Jumper, Leaper, Maze, Miner, Ninja, Plunder, and Starpilot (left to right).

Normalized Score: As suggested by Cobbe et al. (2020), we report normalized scores S by

$$S = \frac{R - R_{\text{min}}}{R_{\text{max}} - R_{\text{min}}},$$

where R is the raw return collected by the agent, R_{min} is the score for the environment by a random agent, R_{max} is the maximum possible score. The normalization constants are shown in Table A.2 and Table A.3 for *easy* and *hard* game settings, respectively.

Table A.1: Game characteristics of the Procgen Benchmark environments. Fixed translation in the x and y directions means the image is centered on the agent, i.e., there is no relative movement of the agent in the image. Agent-centered images convey that the map observation is not fixed but moves relatively to the agent.

Game	**X Translation**	**Y Translation**	**Rotation**	**Map**
Bigfish	Free	Free	Left/right	Fixed
Bossfight	Free	Limited	No	Fixed
Caveflyer	Fixed	Fixed	Free	Free
Chaser	Free	Free	No	Fixed
Climber	Free	Fixed	Left/right	Fixed
Coinrun	Fixed	Fixed	No	Free
Dodgeball	Free	Free	Free	Fixed
Fruitbot	Free	Fixed	No	Free
Heist	Free	Free	No	Fixed
Jumper	Fixed	Fixed	Left/right	Free
Leaper	Free	Free	Free	Fixed
Maze	Free	Free	Free	Fixed
Miner	Free	Free	Fixed	Fixed
Ninja	Fixed	Fixed	Left/right	Free
Plunder	Free	Fixed	No	Fixed
Starpilot	Free	Free	Free	Free

Table A.2: Normalization constants from Cobbe et al. (2020) for all Procgen Benchmark environments in the *easy* setting.

Game	R_{min}	R_{max}	**Game**	R_{min}	R_{max}
Bigfish	1	40	Jumper	3	10
Bossfight	0.5	13	Leaper	3	10
Caveflyer	3.5	12	Maze	5	10
Chaser	0.5	13	Miner	1.5	13
Climber	2	12.6	Ninja	3.5	10
Coinrun	5	10	Plunder	4.5	30
Dodgeball	1.5	19	Starpilot	2.5	64
Fruitbot	-1.5	32.4	Heist	3.5	10

Table A.3: Normalization constants from Cobbe et al. (2020) for all Procgen Benchmark environments in the *hard* setting.

Game	R_{min}	R_{max}	**Game**	R_{min}	R_{max}
Bigfish	0	40	Jumper	1	10
Bossfight	0.5	13	Leaper	1.5	10
Caveflyer	2	13.4	Maze	4	10
Chaser	0.5	14.2	Miner	1.5	20
Climber	1	12.6	Ninja	2	10
Coinrun	5	10	Plunder	3	30
Dodgeball	1.5	19	Starpilot	1.5	35
Fruitbot	-0.5	27.2	Heist	2	10

B Experiment Details

B.1 Hyperparameters List

Table B.4: Hyperparameters for Proximal Policy Optimization (PPO).

Hyperparameter	**Values**
Number Parallel Environments	64
Environment Steps	256
Learning Rate ($\tau = 2$)	3.5×10^{-4}
Batch Size	2048
Epochs	3
Discount Factor γ	0.99
GAE Lambda (λ)	0.95
Clip Range	0.2
Value Function Coefficient	0.5
Entropy Coefficient	0.01
Max Gradient Norm	0.5
Optimizer	Adam
Shared Policy and Value Network	Yes

Table B.5: Hyperparameters for Deep Q-Network (DQN).

Hyperparameter	**Values**
Number Parallel Environments	128
Learning Rate ($\tau = 2$)	1×10^{-4}
Batch Size	512
Discount Factor γ	0.99
Target Network Update Frequency	64,000 steps
Learning Starts	250,000 steps
Train Frequency	1
Replay Buffer Size	1×10^{6}
Exploration Initial ϵ	1.0
Exploration Final ϵ	0.025
Exploration Decay Fractions	0.1
Max Gradient Norm	10.0
Optimizer	Adam

B.2 Benchmark Methods

- Unstructured gradual magnitude pruning (Obando-Ceron et al., 2024a): PyTorch provides tools for unstructured pruning. We use a target sparsity of 0.9 and follow the proposed schedule that starts pruning 20 % into training and stops at 80 %.
- SoftMoE (Obando-Ceron et al., 2024b): PyTorch reimplementation on base of the official code release `https://github.com/google/dopamine/tree/master/dopamine/labs/moes`. We deploy 10 experts.
- ReDo (Sokar et al., 2023): PyTorch reimplementation from `https://github.com/timoklein/redo` that was based on the official code release. We initialize neurons every 100 iterations and set $\tau = 0.025$.

B.3 Translation Sensitivity Maps

Adapting Translation Sensitivity Maps for Procgen: Originally, translation sensitivity maps were generated by shifting MNIST digits, which feature unicolor backgrounds. In this case, the backgrounds fuse naturally and no artifacts are introduced. This approach cannot be transferred to Procgen as the backgrounds are not unicolor. Moreover, it is more meaningful to simply measure the agent's sensitivity to translations of the entities in the foreground, while keeping the original 64×64 pixel background. Translation sensitivity maps can then be created as a visualization where each pixel (x,y) of a heatmap corresponds to the translation sensitivity score s given an image x_{trans} that was translated by (x,y) pixels compared to an original image x_{orig}. As shown in Figure B.2, the x and y axes in these maps are centered around 0 which means the center pixel's sensitivity score is always 0 as it references the untranslated image.

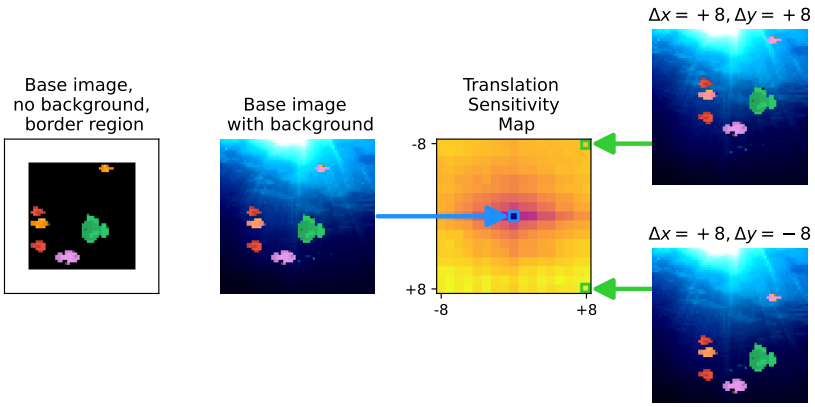


Figure B.2: Generation of translation sensitivity maps. For the arrangement of fish, we require a free border region (white) with a width of 10 pixels, which ensures that translations by ± 8 pixels do not remove any fish from the frame (**left**). We then add the background to obtain the base image, which provides the base actor output (**middle left**). Each pixel of the translation sensitivity map (**middle right**) corresponds to a translation of the entities in the foreground relative to the base image. For example, the pixel in the right upper corner corresponds to the maximal translation to the right and upward (**upper right**).

Data Generation: We select the Bigfish game for evaluation and data generation, as its Procgen game engine allows us to create images of the background and fish entities independently. We originate data from different episodes to ensure data diversity and constrain the agent to be in the center 22×22 pixels square of the frame to mitigate the effect of proximity to the image border. We also ensure that the agent is not alone in the frame, so only interactions with other fish are investigated. As shown in Figure B.2, the collected fish images can then be translated independently over different background scenarios to create translated images x_{trans} given the original image x_{orig}.

Measuring Translation Sensitivity The actor network Actor_θ in PPO with discrete actions outputs the logits of a Categorical distribution, from which actions $a \in \{1, \ldots, 15\}$ are sampled. We define the translation sensitivity score s as follows

$$s = \left\| \text{SoftMax}(\text{Actor}_\theta(x_{\text{orig}})) - \text{SoftMax}(\text{Actor}_\theta(x_{\text{trans}})) \right\|_1,$$

quantifying how much the action probabilities change when translating the input image x_{orig} to x_{trans} while not altering the relative positioning of the entities. Unlike the approach of Kauderer-Abrams (2017) that measures translation sensitivity by computing distances of the network's output vector and then normalizes the score, our method directly operates in the action probability space. By computing the L1 distance between probability distributions, our method ensures a common probability space that directly allows for comparisons between different networks, in contrast to the normalization of the output vector, where finding a meaningful normalization is not obvious.

C Network Architecture

Table C.6: Model summary of the Impala-CNN ($\tau = 2$) for PPO with 64 x 64 input images. The overall parameter count is 1,441,680, with a total of 118.26M multi-adds.

Layer (type:depth-idx)	Input	Output	Param #	Kernel	Param %	Multi-Adds
ImpalaPPOActorCritic	[3, 64, 64]	[15]	–	–	–	–
Impala-CNN	[3, 64, 64]	[256]	–	–	–	–
ConvSequence	[3, 64, 64]	[32, 32, 32]	–	–	–	–
Conv2d	[3, 64, 64]	[32, 64, 64]	896	[3, 3]	0.06%	3,670,016
ResidualBlock	[32, 32, 32]	[32, 32, 32]	–	–	–	–
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	0.64%	9,469,952
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	0.64%	9,469,952
ResidualBlock	[32, 32, 32]	[32, 32, 32]	–	–	–	–
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	0.64%	9,469,952
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	0.64%	9,469,952
ConvSequence	[32, 32, 32]	[64, 16, 16]	–	–	–	–
Conv2d	[32, 32, 32]	[64, 32, 32]	18,496	[3, 3]	1.28%	18,939,904
ResidualBlock	[64, 16, 16]	[64, 16, 16]	–	–	–	–
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	2.56%	9,453,568
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	2.56%	9,453,568
ResidualBlock	[64, 16, 16]	[64, 16, 16]	–	–	–	–
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	2.56%	9,453,568
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	2.56%	9,453,568
ConvSequence	[64, 16, 16]	[64, 8, 8]	–	–	–	–
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	2.56%	9,453,568
ResidualBlock	[64, 8, 8]	[64, 8, 8]	–	–	–	–
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	2.56%	2,363,392
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	2.56%	2,363,392
ResidualBlock	[64, 8, 8]	[64, 8, 8]	–	–	–	–
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	2.56%	2,363,392
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	2.56%	2,363,392
Flatten	[64, 8, 8]	[4096]	–	–	–	–
Linear	[4096]	[256]	1,048,832	–	72.75%	1,048,832
Actor	[256]	[15]	3,855	–	0.27%	3,855
Critic	[256]	[1]	257	–	0.02%	257

Table C.7: Model summary of the Impoola-CNN ($\tau = 2$) for PPO with 64 x 64 input images. The overall parameter count is 409,488, with a total of 117.23M multi-adds.

Layer (type:depth-idx)	Input	Output	Param #	Kernel	Param %	Multi-Adds
ImpoolaPPOActorCritic	[3, 64, 64]	[15]	–	–	–	–
Impoola-CNN	[3, 64, 64]	[256]	–	–	–	–
ConvSequence	[3, 64, 64]	[32, 32, 32]	–	–	–	–
Conv2d	[3, 64, 64]	[32, 64, 64]	896	[3, 3]	0.22%	3,670,016
ResidualBlock	[32, 32, 32]	[32, 32, 32]	–	–	–	–
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	2.26%	9,469,952
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	2.26%	9,469,952
ResidualBlock	[32, 32, 32]	[32, 32, 32]	–	–	–	–
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	2.26%	9,469,952
Conv2d	[32, 32, 32]	[32, 32, 32]	9,248	[3, 3]	2.26%	9,469,952
ConvSequence	[32, 32, 32]	[64, 16, 16]	–	–	–	–
Conv2d	[32, 32, 32]	[64, 32, 32]	18,496	[3, 3]	4.52%	18,939,904
ResidualBlock	[64, 16, 16]	[64, 16, 16]	–	–	–	–
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	9.02%	9,453,568
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	9.02%	9,453,568
ResidualBlock	[64, 16, 16]	[64, 16, 16]	–	–	–	–
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	9.02%	9,453,568
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	9.02%	9,453,568
ConvSequence	[64, 16, 16]	[64, 8, 8]	–	–	–	–
Conv2d	[64, 16, 16]	[64, 16, 16]	36,928	[3, 3]	9.02%	9,453,568
ResidualBlock	[64, 8, 8]	[64, 8, 8]	–	–	–	–
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	9.02%	2,363,392
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	9.02%	2,363,392
ResidualBlock	[64, 8, 8]	[64, 8, 8]	–	–	–	–
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	9.02%	2,363,392
Conv2d	[64, 8, 8]	[64, 8, 8]	36,928	[3, 3]	9.02%	2,363,392
AdaptiveAvgPool2d	[64, 8, 8]	[64, 1, 1]	–	–	–	–
Linear	[64]	[256]	16,640	–	4.06%	16,640
Actor	[256]	[15]	3,855	–	0.94%	3,855
Critic	[256]	[1]	257	–	0.06%	257

D Nature-CNN with GAP

As suggested during the reviewing process, we also evaluate the effect of adding GAP to the Nature-CNN (Mnih et al., 2015). This experiment aims to improve the understanding of the effect of adding a GAP layer. However, as the overall performance of the Nature-CNN is weak and there are limited scaling gains, the results are not fully conclusive.

It can be seen in Figure D.3 that for Nature-CNN, adding the GAP layer is not beneficial. We hypothesize that the reason for this finding is that the Nature-CNN, even when increasing the network width, has too little depth and is still underparameterized. The Nature-CNN with τ=2 has 342,448 parameters in total, of which 262,400 are located in the Linear layer after flattening. In contrast, when adding the GAP, this Linear layer is reduced to 16,640 parameters, resulting in a total of merely 96,688 network parameters. As such, the parameter reduction, which was beneficial for the overparametrized Impala-CNN, consisting of 15 layers and 1,441,680 parameters for τ=2, may have a degrading effect, even if other network characteristics may be improved.

We plan further experiments with an altered, deeper network version of the Nature-CNN in future work, as a more complex experiment design is required to investigate this finding meaningfully.

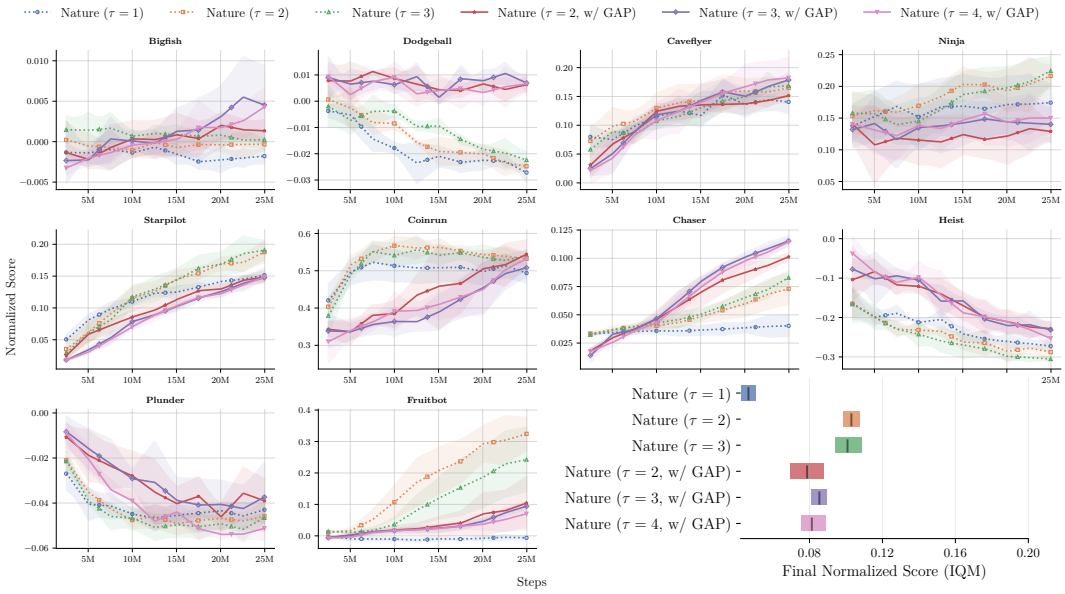


Figure D.3: Scaled Nature-CNN with results for generalization (*easy*) using a subset of 10 Procgen games. The original Nature-CNN is shown as dotted lines; results with GAP are solid. We show the final IQM scores (**bottom right**) and training curves with mean and standard deviation (**rest**).

E Additional Material for Experiments

Learning curves based on the evaluation runs which run every 2.5M and 10M, respectively, time steps. The results show the mean and standard deviation values of the normalized scores S per Procgen environment, i.e., 1.0 corresponds to an optimal policy.

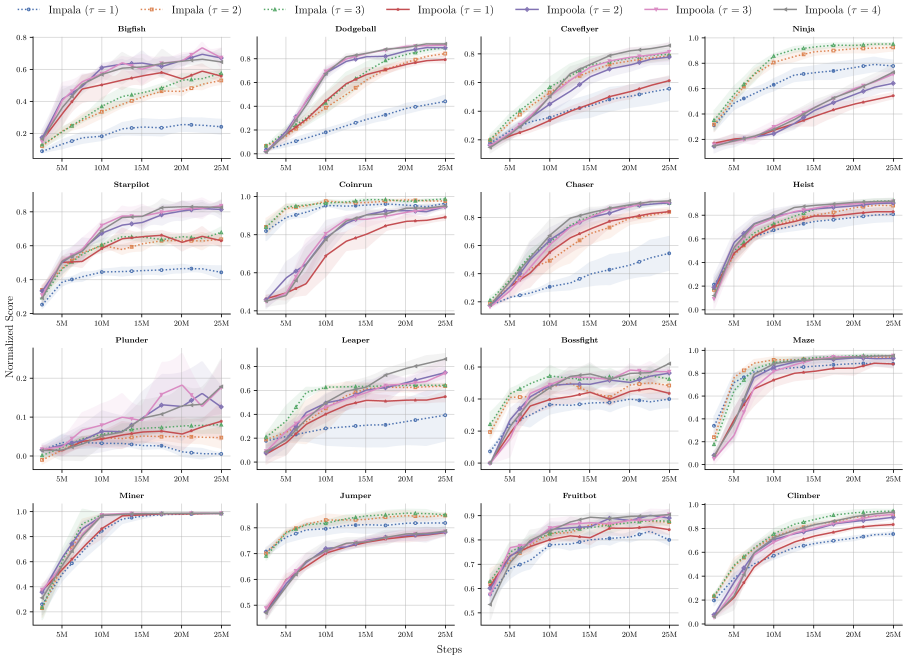


Figure E.4: Detailed results for PPO with different width scalings τ in the *easy* generalization track for *training* levels. Impala-CNN is depicted as dotted lines.

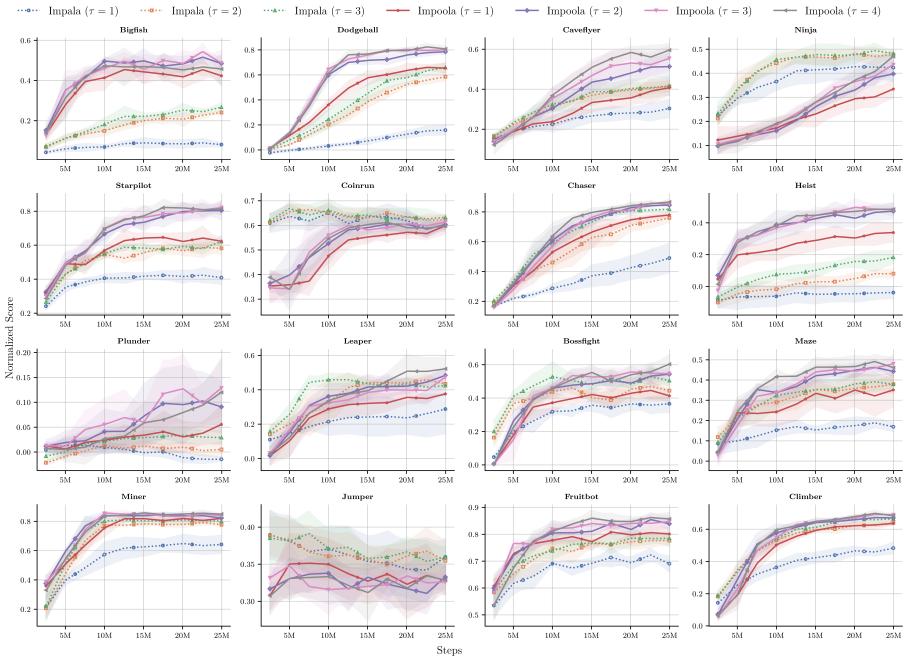


Figure E.5: Detailed results for PPO with different width scalings τ in the *easy* generalization track for *testing* levels. Impala-CNN is depicted as dotted lines.

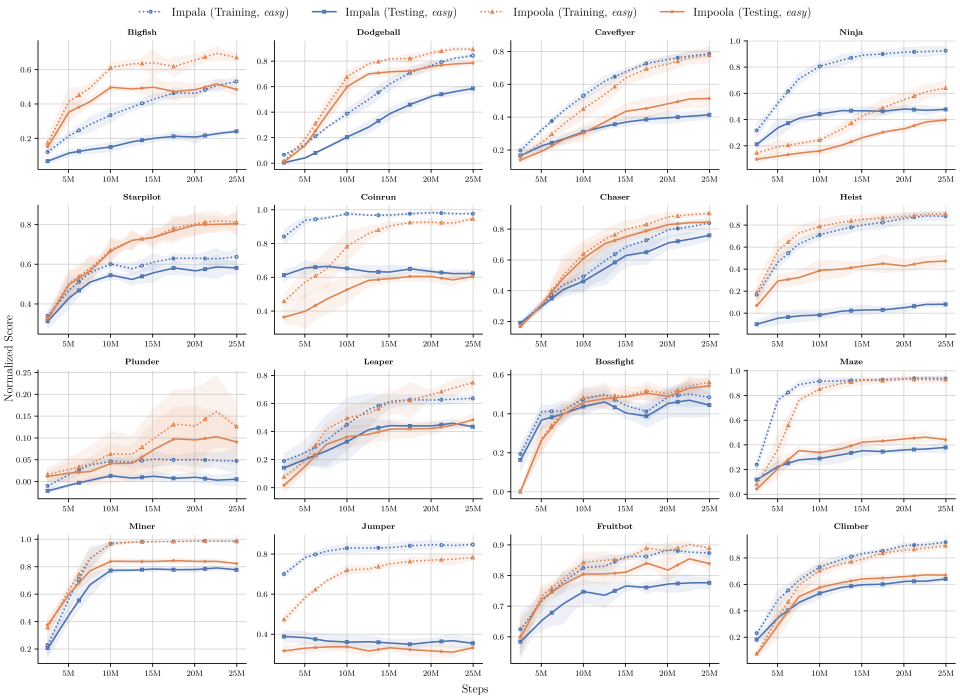


Figure E.6: Detailed results for PPO ($\tau = 2$) in the *easy* generalization track for *training* and *testing* levels. Training levels are depicted as dotted lines.

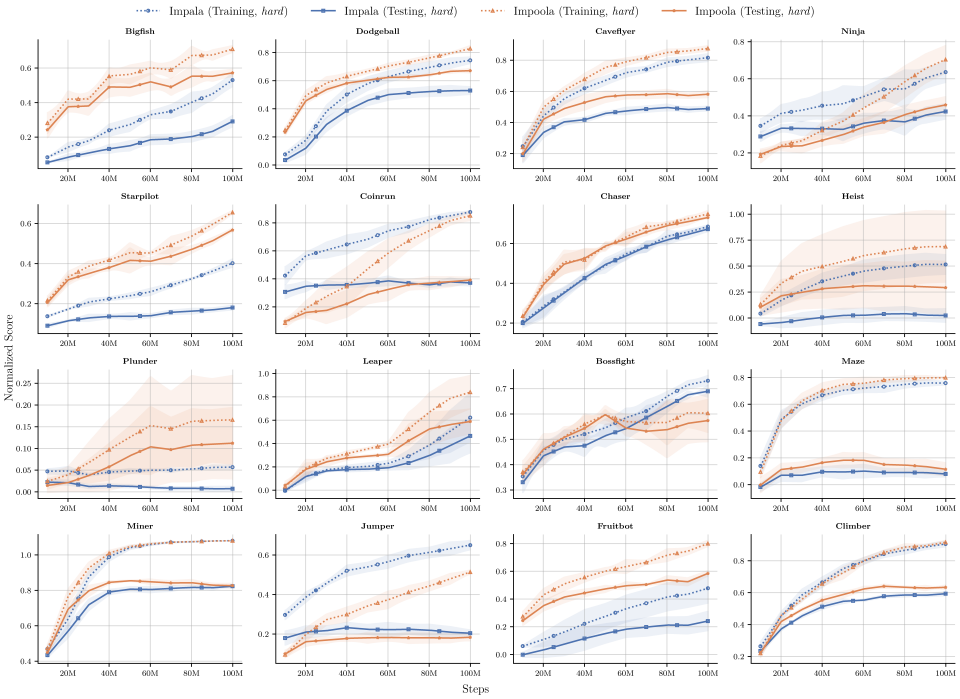


Figure E.7: Detailed results for PPO ($\tau = 2$) in the *hard* generalization track for *training* and *testing* levels. Training levels are depicted as dotted lines.

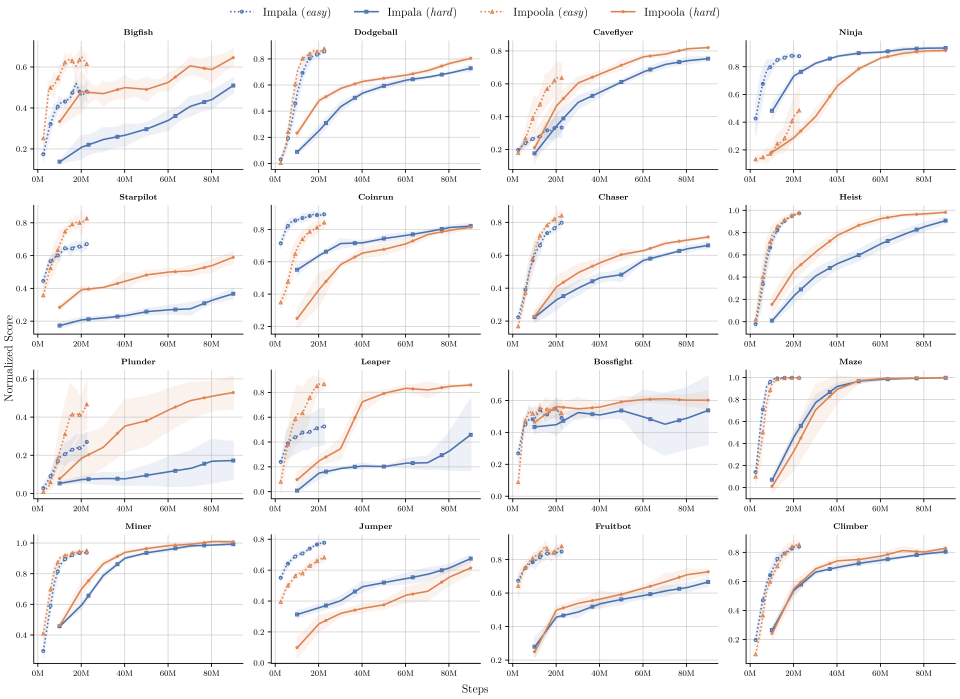


Figure E.8: Detailed results for PPO ($\tau = 2$) in the *efficiency* track (*easy* and *hard*). Easy levels are depicted as dotted lines.

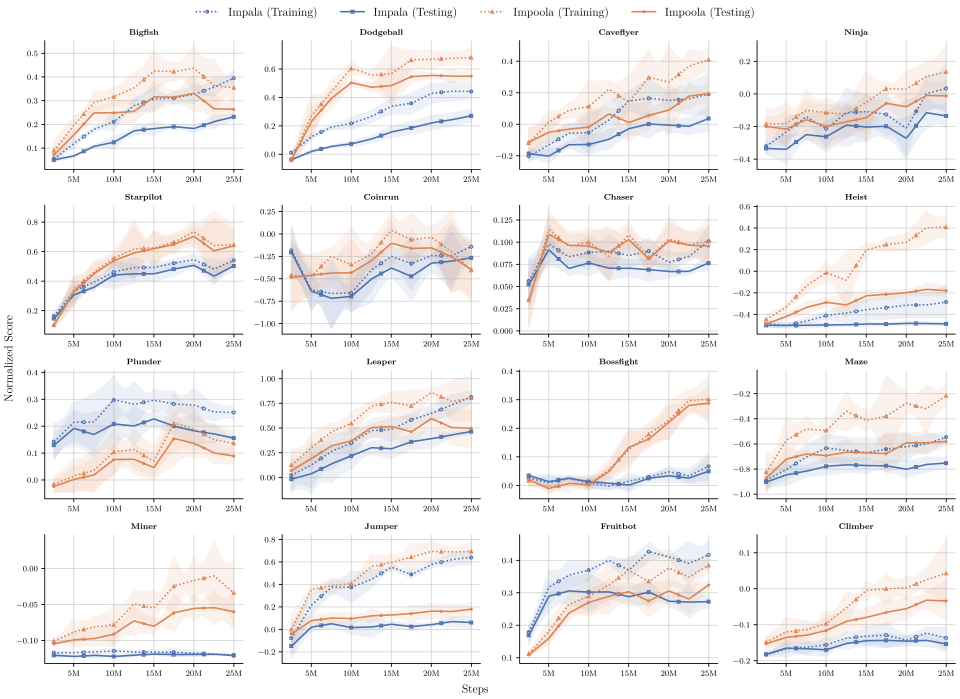


Figure E.9: Detailed results for DQN ($\tau = 2$) in the generalization track (*easy*) for *training* and *testing* levels. Training levels are depicted as dotted lines.

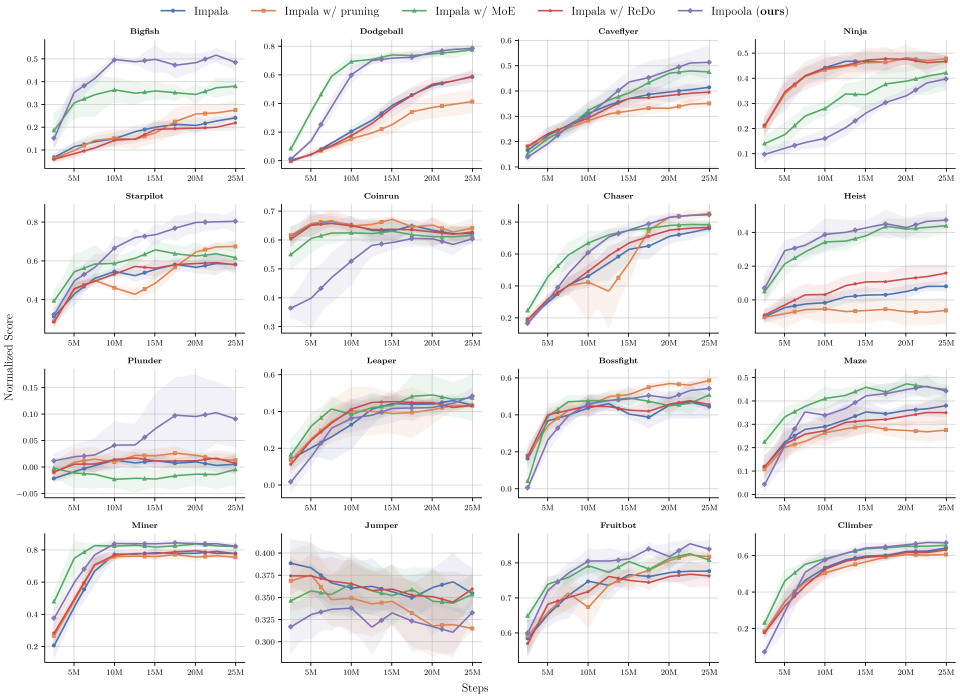


Figure E.10: Detailed results for the benchmark of Impoola-CNN ($\tau = 2$) against other methods in the *generalization* track (*easy*) for *testing* levels.