MASTER'S THESIS 2024

Cross-Domain Generalizability in Image Feature Extraction

Mamdollah Amini, Adi Creson

ISSN 1650-2884
LU-CS-EX: 2024-34

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datateknik

LU-CS-EX: 2024-34

Cross-Domain Generalizability in Image Feature Extraction

Mamdollah Amini, Adi Creson

Cross-Domain Generalizability in Image Feature Extraction

Mamdollah Amini
mamdollah@gmail.com

Adi Creson
adi@creson.se

June 20, 2024

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

Supervisors: Simon Kristoffersson Lind, simon.kristoffersson_lind@cs.lth.se
Alexander Dürr, alexander.durr@cs.lth.se

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

Abstract

This thesis examines the effectiveness of using ResNet-50, a pre-trained deep convolutional neural network, as a feature extractor in the reinforcement learning environment of the Atari game Breakout. The study evaluates the generalizability of features extracted from the last block of different stages of ResNet-50 in training a reinforcement learning agent and compares these across the stages. Through a multi-phase experimental setup, the research explores ResNet-50's ability to adapt to domains outside its original training, without fine-tuning the model. The findings reveal that all stages of ResNet-50 underperformed, particularly in comparison to a established benchmark. Notably, the last stage, stage 4, showed some potential for learning despite overall poor performance. The results suggest that ResNet-50 as a feature extractor has limited success in Breakout and depends heavily on careful integration and design of the reinforcement learning pipeline. This study contributes to the ongoing discussion about the practicality of leveraging large pre-trained models in new domains, underscoring both the challenges and opportunities of repurposing these models for diverse applications.

Keywords: ResNet, Feature Extraction, Reinforcement Learning, Atari Breakout, Generalizeability

Acknowledgements

We extend our sincere thanks to our supervisors, Alexander Dürr and Simon Kristofferson Lind, at the Faculty of Engineering at Lund University, for their invaluable guidance and expertise. Their mentorship was instrumental in our development from novices to knowledgeable practitioners throughout this project. We are particularly grateful for their availability and the insightful discussions during our weekly meetings.

Contents

Chapter 1
Introduction

In this chapter, we introduce the topic and discuss the problem statement along with our motivation for pursuing this thesis. We then define the purpose and objectives of our study. Following this, we outline our research questions and the corresponding hypotheses. We also describe the delimitations to clarify the scope of our research. Finally, we provide a brief overview of the structure of the paper.

1.1 Introduction

Machine learning (ML) has revolutionized the way we extract meaningful insights from vast amounts of data and tackle complex problems. Machine Learning is defined as the process of solving a practical problem by an algorithmically built statistical model based on extracted knowledge from gathered data [7]. Both the wealth of data and the adoption of Machine Learning and AI have increased in recent years, especially after the introduction of major applications like OpenAI's ChatGPT; and this accelerating rate of innovation in AI is driven by the investment, access to rich data, and the continuous researches directed by tech giants [1]. Before the adoption of machine learning and AI, intelligent systems operated based on manual human interventions and expert-designed decision rules to process data or handle user input [26]. These types of logic-based systems were feasible for deterministic and well-understood processes that could be modeled with hand-coded rules, but they were domain and task-specific, and a small change would require redesigning or rewriting the whole system by a human expert in that field [26]. Using ML, however, and presenting a large enough collection of training data enables an ML algorithm to learn and determine the underlying characteristics, patterns, and distribution in the data, which eliminates the need for a manual decision-making process [33].

The effectiveness of ML algorithms hinges significantly on the quality of the training data the ML algorithms are trained on [19]. This necessitates the preparation and transformation

of raw data since raw data collected from real-world settings often contains inconsistencies, missing values, or redundant information, which can impede the learning process of ML algorithms [26, 29]. Raw data like input images from cameras, for instance, need to be transformed into a suitable format before being used for training a desired model. Therefore, ML engineers and data scientists are often tasked with the responsibility of *data preparation*, among other tasks, a process that involves techniques such as cleaning, reshaping, and feature engineering the raw data [26]. Feature engineering is a cornerstone in data preparation and aims at enriching the dataset by deriving new informative features or transforming existing ones, a pivotal step in reducing the dimensionality of the data while preserving relevant information [26]. Feature extraction, a subset of feature engineering and a subject this study focuses on is a crucial step for many reasons, especially when working with large datasets or complex algorithms. Working with extracted features enables models to learn faster due to reduced computational cost, improves their performance due to reduced redundancy, prevents overfitting to training data, and provides overall insights into data [2]. There are different methods and techniques for feature extraction depending on the type of data or domain, i.e., depending on whether we work with image processing, text data, or audio processing.

Feature extraction is essential for enhancing the performance of machine learning algorithms and models by improving data processing capabilities. However, a model's performance is highly dependent on the quality and characteristics of its training data, including the features extracted during training. Consequently, a model trained on data from a specific source or environment often experiences performance degradation when applied to data from a different source with different statistical distributions.

For instance, a model trained on ImageNet-1k [9], which is a multi-category image dataset with 1000 distinct categories, is unlikely to maintain the same level of performance when evaluated with the COCO [24] dataset, which focuses on instance segmentation of 80 common objects in natural context [11]. This discrepancy arises because the feature extractors learned during training are typically tailored to the specific dataset, rather than incorporating *general features*. General features refer to characteristics that are not specific to a particular dataset. Instead, they encapsulate attributes common across various datasets and classes of objects. These features help in representing the overall distribution of diverse datasets.

In the context of Reinforcement Learning (RL), where agents must learn and adapt to different environments, using a general feature extractor can significantly enhance an agent's ability to operate in new, previously unencountered environments. By leveraging features that are universally applicable rather than dataset-specific, agents should be able to generalize better, thereby improving their performance in diverse and dynamic settings.

1.2 Problem Statement and Motivation

Machine learning models are often task-specific and trained on datasets that predominantly feature task-optimized features. This approach may yield high performance on the targeted task but models trained on task-specific features often lead to limited generalization capabilities, as expected, and struggle to adapt when confronted with new datasets or scenarios beyond the training scope.

To address the need for models that can generalize and adapt to diverse settings, this study investigates the use of large pre-trained models, such as ResNet-50, as feature extractors in environments and tasks different from those they were originally trained on. Although general features and the use of large pre-trained models as feature extractors hold significant promise, this area remains underexplored for reinforcement learning.

1.3 Purpose and Goal

The purpose of this thesis is to investigate how and to what extent large CNN-based pre-trained models like ResNet-50 can be used to extract general features, and how it could benefit model training in reinforcement learning environments. Hence, the primary objective of this thesis is to investigate whether pre-trained models can be used as general-purpose feature extractor in RL. Leveraging the inherent versatility of general features helps enhance model generalization capabilities and enables more robust and adaptable learning solutions. This would help minimize the need for building custom and specific feature extraction models for different tasks.

1.4 Research Questions

The following research questions are formulated to guide our study:

1. To what extent is it possible to train a reinforcement learning agent using a pre-trained ResNet-50 model as a feature extractor?

2. Is there a difference in performance among the different stages of ResNet-50 when used as feature extractors in training an RL agent?

1.5 Delimitations

In this research, we have decided to focus exclusively on ResNet-50 as our large pre-trained model of choice. Additionally, we have confined our analysis to the Breakout Atari game to assess the generalizability of ResNet-50.

1.6 Outline of the Paper

In this paper, we provide the theoretical foundation, covering the key components of ResNet-50, the reinforcement learning framework, the Proximal Policy Optimization (PPO) algorithm, and the specifics of Atari Breakout, the chosen training game. Our methodology involves a multi-phase study, including benchmark replication, an ablation study, a hyperparameter sweep, and the main experiment evaluating different stages of ResNet-50. We present the results using the mean episodic reward as the primary metric, discuss various observations and considerations for future research, and conclude with our findings.

Chapter 2
Theoretical Background and Preliminaries

This chapter presents the fundamental theoretical background of the thesis along with other relevant works. First, Section 2.1 presents essential concepts in Machine Learning, focusing on supervised and reinforcement learning (RL). Next, Section 2.2 presents the resurgence of Deep Learning, including deep neural networks (DNN) and convolutional neural networks (CNN). Section 2.3 provides an overview of backpropagation and Residual Networks, detailing the ResNet-50 architecture. Section 2.4 provides a more detailed explanation of RL, covering the RL framework, exploration versus exploitation, the credit assignment problem, reward shaping, the Bellman equation, and the Proximal Policy Optimization (PPO) algorithm used for training RL agents. Finally, Section 2.5 explores the application of ResNet-50 as a feature extractor in RL tasks, with a specific focus on the Atari Breakout environment.

2.1 Machine Learning

Machine learning employs techniques to identify patterns in data and uses these patterns to make predictions about new, unseen data without relying on explicitly programmed rules. This process can be executed using traditional algorithmic machine learning models or through deep neural networks that act as function approximators.[29]. There are several paradigms within ML, each with different for different types of data and tasks. Therefore, choosing the right learning algorithm requires an understanding of the fundamentals of various learning algorithms and how they apply to different tasks [29]. **Learning algorithms** are strategies or techniques used to learn from data. Examples of learning algorithms include, for instance, *k-means clustering* for grouping data points and *Q-learning* for training reinforcement learning agents. **Models**, on the other hand, are the concrete representations of the learned patterns and relationships from data, and the tangible outcomes of the learning processes [7]. Real-world data are of various forms and therefore the choice of the learning algorithm, hence the resulting model, depends on the specifications of the problem being solved, the nature of the data, and the desired outcome [7]. ML algorithms have different approaches when it comes

to practical implementation and most common algorithms are broadly categorized into *supervised learning*, *unsupervised learning*, and *reinforcement learning*.

2.1.1 Supervised Learning

Supervised learning is a setting in which the learning algorithm is provided with **labeled** training data, i.e., known pairs (input, output), and the model is expected to learn a function that maps inputs to outputs [29]. Supervised learning algorithms are applied in both *classification* and *regression* tasks. Input can be anything from email messages to sensor measurements and camera images, transformed into machine-readable feature vectors or attributes. Outputs can be real numbers, labels, sequences of labels, vectors, or some other structure[7]. These types of learning algorithms focus on the accurate prediction of new, previously unseen inputs or examples that have the same characteristics as the training set. These types of models are used in tasks such as classification in medical imaging and diagnoses, object detection, real estate valuation based on house features, and other similar tasks [26].

2.1.2 Reinforcement Learning

Reinforcement learning (RL) is another fundamental paradigm in ML and refers to the types of learning problems where the goal is to teach a learning agent to maximize some notion of cumulative reward through interactions with an environment. The agent observes the *state* of the environment, takes *actions* at discrete time steps based on the perceived state, receives *rewards* or penalties based on the executed actions, transitions into a new state and evaluates its decisions to adjusts its behavior to optimize its long-term performance [7]. An RL agent is expected to learn a *policy function* with a maximized *expected average reward* that can make the optimal action/decision for a given state [7]. Decision-making processes are modeled by frameworks based on Markov processes (MPs), discrete-time stochastic processes where the conditional probability distribution of the future states only relies on the present state [23]. The standard theory of RL is defined by a Markov Decision Process (MDP), an extension of the MP, and typically involves five elements as follows:

- $\boldsymbol{S}$: set of *states* or observation space from an environment.
- $\boldsymbol{A}$: set of *actions* the agent can choose from.
- $\boldsymbol{T}$: a *transition probability* function $T(x_{t+1}|s_t, a_t)$ that specifies the probability of transitioning to state $s_{t+1} \in S$ given the agent chooses action $a_t \in A$ in state $s_t \in S$.
- $\boldsymbol{R}$: a *reward* function $r_{t+1} = R(s_r, s_{t+1})$ which rewards/punishes the agent for transitioning from state s_t to state s_{t+1} by taking action a_t.
- $\gamma \in [0, 1]$: a discount factor that determines the relative importance of distant future rewards compared to the immediate future.

The agent's strategy for choosing an action a given a state s is called its *policy* $\pi(a|s)$. The tuple $(s_t, a_t, r_{t+1}, s_{t+1})$ is called a *transition*, and several sequential transitions are called *roll-out*. A finite *trajectory* or a sequence $(s_0, a_0, r_1, s_1, a_1, r_2, ...)$ with a finite length τ is called an episode. These terms become useful when analyzing the performance of RL agents by comparing their *episodic mean reward*, for instance. The ultimate goal of the agent is to learn an optimal policy π^* that maximizes the discounted expected reward. The discounted expected reward for an episode with length τ is defined as the weighted sum of immediate rewards:

$$\mathcal{G}(\pi) = \mathbb{E}_{\tau_\pi}\mathcal{R} = \mathbb{E}_{\tau_\pi}\sum_{t=0}^{\tau-1}\gamma^t r_{t+1}$$

Here, τ_π is a trajectory distribution, the probability of observing a trajectory $(s_0, a_0, r_1, s_1, a_1, r_2, ...)$, and defined as:

$$\tau_\pi = \prod_t \pi(a_t|s_t)T(s_{t+1}|s_t, a_t)$$

A short comparison helps clarify how RL differs from other ML methodologies like supervised learning. RL differs from supervised learning in learning objectives, data requirements, and feedback mechanisms. In a supervised setting, the model learns a mapping from inputs to outputs from a fixed dataset, but in RL the agent interacts with an environment and generates data through exploration. This is what makes the RL sample inefficient compared to supervised learning [32]. The model in a supervised setting receives explicit feedback in the form of correct labels while an RL agent receives feedback in a continuous loop where the agent's actions or decisions influence the environment and, consequently, future observations and rewards. The core components of an RL problem are:

- **Agent**: The learner or decision maker.
- **Environment**: The external system with which the agent interacts.
- **State (s):** A representation of the current situation of the agent.
- **Reward (r)**: The feedback from the environment based on the action taken.
- **Policy (π)**: The strategy that the agent employs to make decisions and determine actions based on the current state.

2.2 The resurgence of Deep Learning

Earlier algorithms in ML, often referred to as classic ML methods, had limitations and faced challenges such as the necessity of manual feature engineering, scalability issues with high-dimensional data or large-scale datasets, and the difficulty in capturing complex patterns as well as hierarchical features in data. The limitations imposed by these challenges were addressed when deeper neural networks emerged with the name of *deep learning* after 2010 [22]. The interest in neural networks was revived due to the increased and advanced computational power, particularly GPUs, the availability of large amounts of data, which deep learning models require for effective training, and more importantly development of successful applications capable of addressing actual, real-life problems. A key milestone in this

resurgence was the impact of AlexNet, a *convolutional neural network* (CNN) that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. This section will introduce the basics of neural networks and deep learning, and then explain the specialization of CNNs for visual tasks, and finally provide an overview of the ResNet-50 architecture which won the ILSVRC in 2015.

2.2.1 Neural Network

To clearly describe deep learning, we must first introduce the concepts of neural networks (also known as multi-layer perceptrons) and neurons. Neural networks are distinguished from other classical methods by their unique *structure*. A neural network comprises layers of nodes, or artificial neurons, a unique structure that distinguishes them from other classical methods [26]. Each neural network includes an input layer, an output layer, and one or more hidden layers with weighted connections to the neurons in the preceding and succeeding layers [7].

A **neuron** or a node is the most fundamental building block of a fully connected neural network that takes the weighted sum of inputs from every node in the preceding layer, adds a bias term, and distributes it to neurons in the succeeding layer through a non-linear *activation function*. Figure 2.1 shows a simple feed-forward neural network with $p = 5$ predictors or features $X_1, ..., X_5$ and $K = 5$ hidden units or neurons. It shows how a neuron in the hidden layer takes the weighted sum of features from the input layer and returns an output which is further used by the single neuron of the output layer. Activation functions are crucial for the neural network since they add non-linearity and enable the network to learn complex and non-linear patterns in the data [7]. It is called an activation function since the function compares the input value to a threshold value and decides whether a neuron should be active or not. The choice of activation function plays an important role in the performance of the model since they are involved in the *backpropagation* algorithm, a feed-backward process where the model parameters are updated. A more detailed explanation of backpropagation is provided in 2.3.

In general, a neural network model has the form

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k \mathrm{h}_k(X) = \beta_0 + \sum_{k=1}^{K} \beta_k \mathrm{g}(w_{k0} + \sum_{j=1}^{p} w_{kj} X_j)$$

Here, $w_{10}, ..., w_{Kp}$ and $\beta_0, ..., \beta_K$ represent the parameters. The K activations $a_k^2, k = 1, ..., K$, in the hidden layers, are computed as functions of the input features $X_1, ..., X_p$, and fed into the output layer, resulting in

$$f(X) = \beta_0 + \sum_{k=1}^{K} \beta_k a_k^2$$

The nonlinear activation function $g(z)$ is the Sigmoid function in Figure 2.1, however, the preferred choice in modern neural networks is the *ReLU* (*rectified linear unit*) activation function [19], which takes the form

$$g(z) = (z)_+ = \max\{0, z\}$$

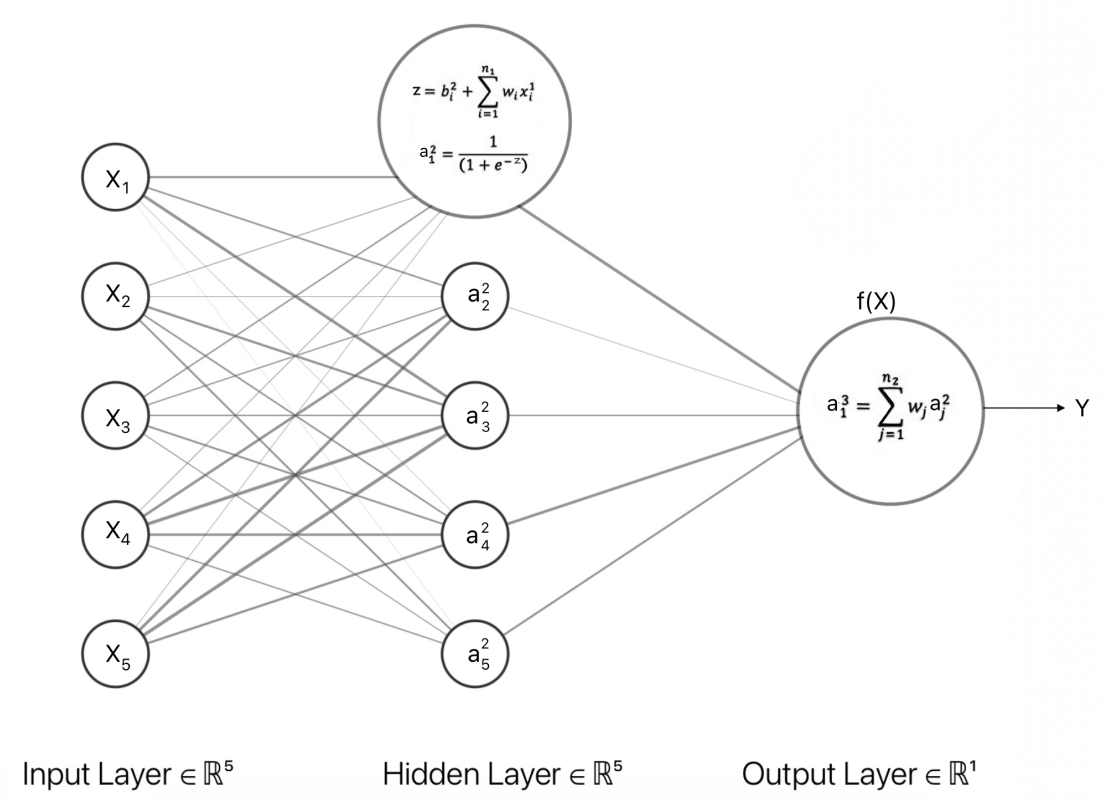


Figure 2.1: A simple feed-forward neural network with a single hidden layer. The hidden layer computes activations $a_j^{(2)} = \sigma(z_j) = \frac{1}{1+e^{-z_j}}$ that are nonlinear transformations of the weighted sum of inputs $X = (X_1, X_2, ..., X_5)$. The model has an input of five features, one hidden layer with five neurons and Sigmoid activation functions, and one output neuron [28]

2.2.2 Deep Neural Network

The "deep" in deep neural network (DNN) refers to the depth of neural networks, and networks with two or more hidden layers are considered deep neural networks [8]. Deep neural networks reduce the manual human intervention required by classic neural networks and enable DNNs to use large unstructured data, particularly since over 80% of an organization's data is estimated to be unstructured [8]. In general, and similar to the simple NN with one hidden layer described earlier, a DNN with p input units, L hidden layers each with K_l nodes, and m output nodes has the following form where $a_k^{(l)}$ is the activation of k:th node in l:th hidden layer, and each Z_m is a different linear model in the final output layer:

$$a_k^{(l)} = h_k^{(l)}(X) = g(w_{k0}^{(l)} + \sum_{j=1}^{p} w_{kj}^{(l)} X_j)$$

$$Z_m = \beta_{m0} + \sum_{j=1}^{K_L} \beta_{mj} h_j^{(L)}(X) = \beta_{m0} + \sum_{j=1}^{K_L} \beta_{mj} a_j^{(L)}$$

Here, the superscript notation indicates to which layer the parameters, activations, and activation functions belong. For instance, $h_k^{(l)}(\cdot)$, $a_k^{(l)}$, and $w_{kj}^{(l)}$ represent the activation functions, activations, and weights of the l:th layer, respectively. β_{mj} represent the weights connecting the j:th node in the last hidden layer to the m:th output node.

Despite being powerful for certain tasks, *plain* neural networks on itself have limitations when applied directly to visual and spatial data [19]. The *curse of dimensionality*, *parameter explosion*, feature engineering, and *translation invariance* are some of the limitations of plain neural networks when used with visual data. For instance, a 100x100 grayscale image has 10,000 dimensions as each pixel is considered a feature. The input layer in a neural network would require 10,000 parameters or weights for each neuron in the first hidden layer, $\mathbf{W} \in \mathbb{R}^{N \times 10000}$. Image data are usually high-dimensional and NNs struggle to efficiently process them due to the large number of parameters required in the network [29]. Consequently, the large number of connections among the hidden layers in a DNN increases both the computational complexity and memory requirements of the network. Furthermore, regular DNNs can not efficiently capture spatial features and translation invariance in visual data [29]. Convolutional neural networks address these limitations through convolutional layers with learnable kernels, which enable parameter sharing, sparse connectivity, and capture features regardless of spatial translations [3].

2.2.3 Convolutional Neural Network

As introduced in 2.2, DNNs resurged around 2010 when massive databases of labeled images of increasingly more classes were being accumulated, and a special family of NN called *convolutional neural networks* (CNNs) evolved and gained significant success in a wide range of problems related to visual tasks like image recognition, object detection, and segmentation [22]. This section will provide a more detailed explanation of the idea behind the unique architecture of CNNs and how they are inspired by the human visual system.

A CNN builds a hierarchy of representations from an input image by first identifying low-level features, such as edges and lines, and progressively combining them into higher-level features like parts objects. This process is achieved through a combination of *convolution* layers, which extract features, and *pooling* layers, which reduce the dimensionality of the feature maps [19]. Feature maps from the convolution backbone are then passed to fully connected layers for capturing global relationships, ending with a softmax activation to classify the image by supplying probability distributions over various classes [26].

A convolution layer consists of multiple small-sized *convolution kernels* or *filters*, such as 3x3 or 5x5 matrices of weights. Convolving the kernels with the input images produces feature maps, and enables parameter sharing across the entire image, which reduces the number of required parameters [22]. Formally, the convolved image $\mathbf{O} \in \mathbb{R}^{H' \times W'}$ from applying the discrete convolution between a kernel $\mathbf{K} \in \mathbb{R}^{k_h \times k_w \times 3}$ and an RGB input image $\mathbf{I} \in \mathbb{R}^{H \times W \times 3}$ is formulated as:

$$\mathbf{O}(i,j) = \sum_{c=1}^{3} \sum_{m=1}^{k_h} \sum_{n=1}^{k_w} \mathbf{K}(m,n,c) \cdot \mathbf{I}(i+m-1, j+n-1, c) + b$$

where $\mathbf{I}(i+m-1, j+n-1, c)$ is a local patch of the input image in the channel c, and $\mathbf{K}(m, n, c)$ is the filter weights. A non-linearity function, such as *ReLU*, is applied element-wise to the convolved image to get the feature map. The resulting feature map is then passed through a pooling layer to reduce its spatial dimensions. The most common pooling operation is *max pooling*, which takes the maximum value within a specified window, providing translation invariance and reducing computation[22]:

$$\text{MaxPool}(x) = \max_{(m,n) \in window} x(m, n)$$

Kernels in a CNN typically match or are smaller than the input image dimensions. The number of kernels in each convolution layer determines the number of channels for the subsequent layer. For instance, if the first convolution layer uses $\mathbf{K}$ kernels, the second layer's kernels should have $\mathbf{K}$ channels. The resulting feature maps are combined to form a two-dimensional output after applying an activation function and, consequently, no color information is passed to subsequent layers [22]. Figure 2.2 shows a deep CNN architecture with convolution layers, max-pooling layers, fully-connected layers, and softmax activation.

2.3 Backpropagation and Residual Networks

Convolutional neural networks have achieved remarkable success in various tasks, yet increasing their depth without compromising performance remains a significant challenge. Increased depth is crucial for the hierarchical representation of features. Empirical evidence suggests that the performance of a CNN is correlated with its depth; however, constructing deep neural networks involves more than merely adding new layers [29]. Neural networks are trained using the *backpropagation* algorithm, which relies on the c*hain rule* to update the network parameters effectively. During the final step of a forward pass, the network's predicted output is compared to the true output using a loss function. The backpropagation algorithm then performs a *backward pass* to minimize the cost function by adjusting the model's weights and biases. The adjustment to each parameter is determined by the gradients of the cost function for that parameter. Thus, backpropagation employs the chain rule to compute the gradients across different layers, making it a form of gradient descent tailored to the layered structure of neural networks.

Deeper networks benefit from the potential to learn more complex features and patterns, but training deep networks using backpropagation becomes increasingly challenging, especially with certain activation functions like the *Sigmoid* function [20]. As more layers are added, the gradients of the loss function may approach zero (vanishing gradients) or become excessively large (exploding gradients). This phenomenon occurs because the chain rule results in the multiplication of partial derivatives of each layer. For instance, in a network with n hidden layers, backpropagation updates the input layer's parameters by a fraction of the product of n small gradients multiplied by the learning rate, which typically results in a small product and, in rare cases, a large one. This issue, known as the *vanishing/exploding gradients* problem, causes gradients to decrease/increase exponentially as they propagate back to the first layer. Consequently, the degradation problem in deeper networks causes saturation in accuracy during training [3].

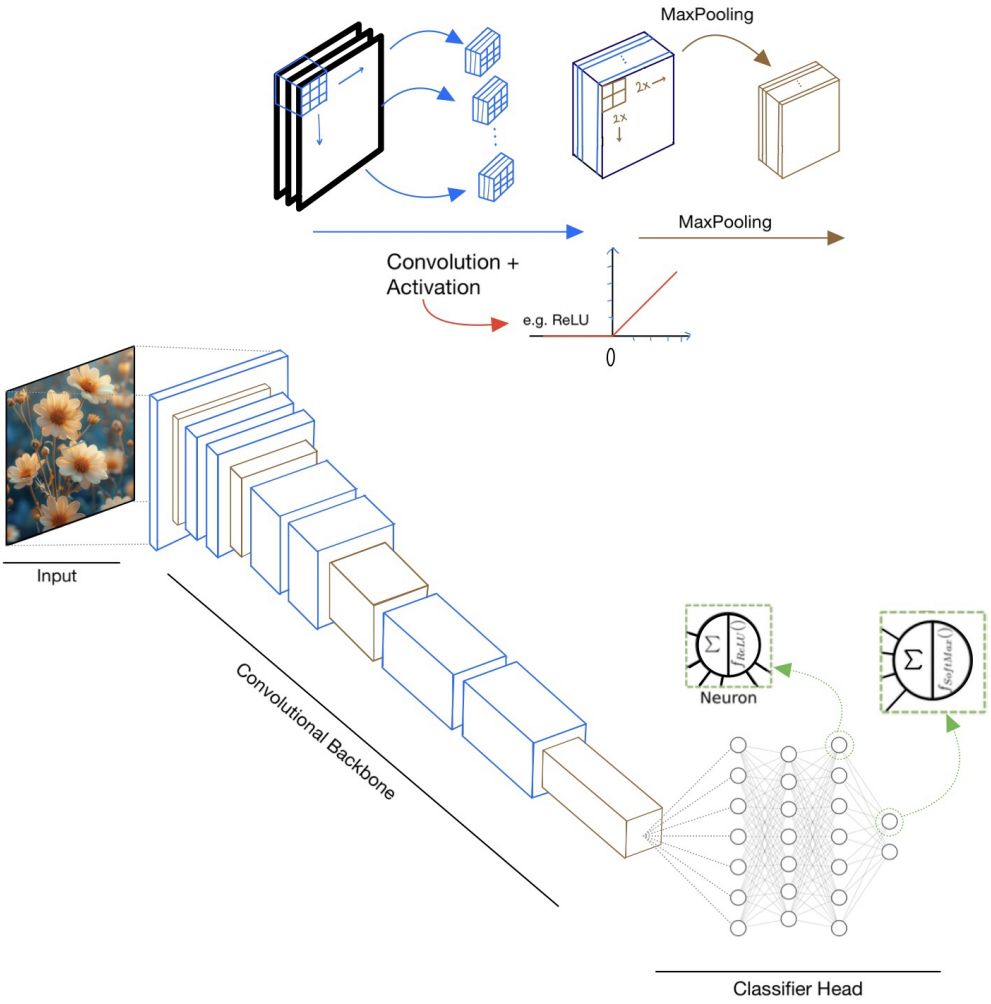


Figure 2.2: Overview of a Convolo neural network. A 2x2 max-pooling layer is applied after every two convolutional layer.

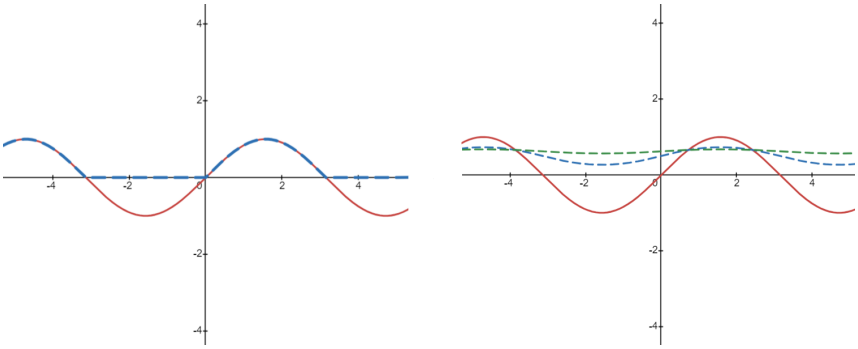


Figure 2.3: A sinusoidal input signal (red) propagates through a simple network with two different activation functions. The network has two layers, each with a single neuron. Left: ReLU activation function, $f(x) = \max\{0, \sin(x)\}$. Right: Sigmoid activation function, $f(x) = \frac{1}{1+e^{-\sin(x)}}$. The blue curves represent the output signals after passing through two ReLU functions. The green curves represent the output signals after passing through two Sigmoid functions.

This issue with the Sigmoid activation function can be illustrated using a simple sinusoidal input signal. Figure2.3 demonstrates how the same input signal propagates through a simple network of two layers, each with a single neuron, using two different activation functions. It can be observed that the signal diminishes as it propagates deeper into the network with Sigmoid activation functions, Figure 2.3 (right), whereas it undergoes a different transformation with Rectified Linear Unit (ReLU) functions, Figure 2.3 (left). As a result, the gradients in the network with the Sigmoid activation functions also shrink with the number of layers, as illustrated in Figure 2.3.

A simple solution to the vanishing gradient problem is to avoid using the Sigmoid activation function, thus preventing the squeezing of activation values and gradients. A more effective solution is to use *residual connections* or *shortcut connections* in neural networks [20]. This type of architecture, known as *residual neural network* or *ResNet*, does not require retaining the entire input signal and is designed to scale and support a large number of layers with minimal risk of degrading the network's accuracy and performance. The residual connections in ResNets allow the network to retain relatively more input information over stacks of layers, mitigating the problem of vanishing gradients while optimizing efficiency, accuracy, and complexity [20]. Residual connections, hence residual learning, are implemented in residual blocks which are based on the idea that if a stack of nonlinear layers can approximate a desired underlying mapping $H(x)$, then it can also approximate the residual function $F(x) = H(x) - x$, which can be reformulated to $H(x) = F(x) + x$.

2.3.1 Overview of ResNet-50 Architecture

This section provides an overview of the ResNet-50 architecture, a 50-layer variant of residual networks, and explains how this innovative architecture overcomes the vanishing/exploding gradients problem.

ResNet-50 is a 50-layer deep CNN architecture based on the residual convolution network introduced in the original paper. It consists of multiple building blocks called *bottleneck residual blocks*. These blocks consist of two paths: a main path with a stack of three convolutional layers, and a shortcut path that performs an identity mapping. The outputs of the two paths are added elementwise to form the output of the residual block. Figure 2.4 shows a 3-layer "bottleneck" residual block for ResNet-50. The first and last 1x1 convolutions in the block reduce and restore dimensions, resulting in smaller input/output dimensions for the middle 3x3 bottleneck layer [20]. Using a residual block allows the network to fit the residual mapping $F(X)$ which is easier to optimize than the underlying mapping $H(X)$ [20]. The skip connection in the residual block allows the unhindered flow of data along the feed-forward path, and the gradients along the backpropagation path. The identity shortcut connections do not add extra parameters, so adding additional layers does not degrade the network's performance since regularization will handle them if they provide no benefit [20]. On the other hand, these additional layers can improve the network's performance, because they have non-zero parameters even with regularization.

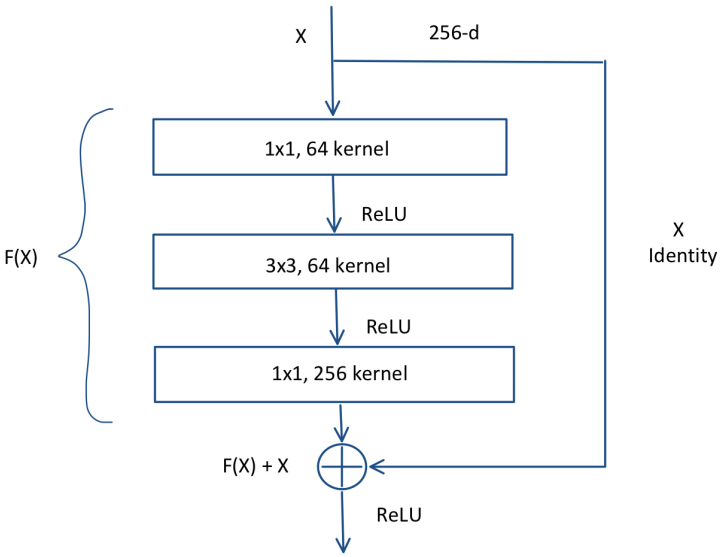


Figure 2.4: Overview of a bottleneck residual block for Resnet-50. To avoid a dimensional mismatch between stages, the identity path in the first residual blocks of stages two, three, and four also includes a convolutional layer.

Let X represent the input to the residual block, which can be the output of a previous layer. This input is fed to both the shortcut connection and the stack of convolution layers forming the main path. Let $F_i(X)$ denote the output of the i-th convolutional layer for $i \in \{1, 2, 3\}$, and $ReLU(X)$ is the activation function. The computation within the residual block and its output can be expressed as follows:

$$F(X) = F_3(\text{ReLU}(F_2(\text{ReLU}(F_1(X)))))$$

$$\text{Output} = \text{ReLU}(X + F(X))$$

Here, $F_i(X) = W_i * X + b_i$, where W_i are the convolutional filter weights, b_i are the bias terms and $*$ denotes the convolution operation. The ReLU activation function, defined as $\text{ReLU}(X) = \max(0, X)$, is applied after each convolutional layer.

The residual blocks are stacked on top of each other to create different ResNet architectures such as ResNet-18, ResNet-101, and ResNet-152. ResNet-50 consists of 16 residual blocks organized into 4 residual stages, with each stage containing a certain number of residual blocks. Specifically, the four stages have 3, 4, 6, and 3 residual blocks, respectively [20]. Additionally, ResNets include an initial convolutional layer, followed by a max pooling layer before entering the first residual block of the first stage. The network concludes the convolutional backbone with a global average pooling layer after the final residual block of the last stage, and a fully connected (dense) layer that produces the final output. This stacking of convolutional layers in residual blocks and organizing them in stages allows for both deep and efficient architectures, maintaining high performance and accuracy. Figure 2.5 illustrates an overview of the ResNet-50 architecture.

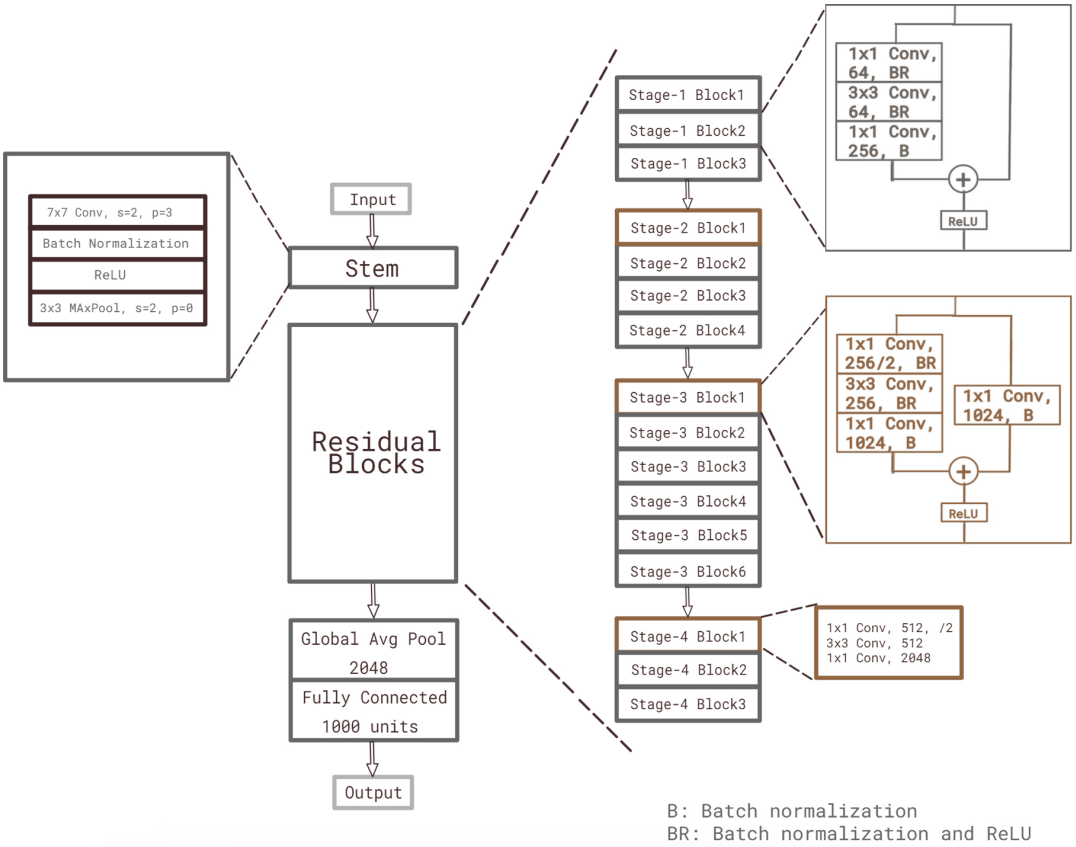


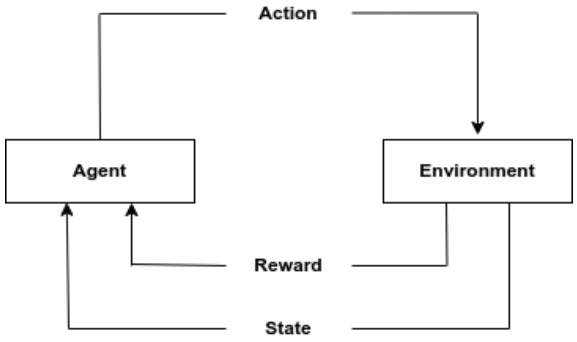
Figure 2.5: An overview of the ResNet-50 architecture. The stem or the initial stage consists of one convolutional layer and one max-pooling layer. Each bottle residual block in each stage consists of 3 convolutional layers: 1x1, 3x3, and 1x1. The final layers consist of an average pooling layer and a dense layer.

2.4 Reinforcement Learning

This section will focus on explaining core concepts of Reinforcement Learning which have been monumental in laying the foundations and have aided machines in surpassing expert human level in highly complex environments such as Go [16] [17], and the e-sport game, Dota [10].

2.4.1 Reinforcement Learning Framework

A good framework for understanding reinforcement learning is depicted in figure 2.6, below. It describes the core components of what comprises the reinforcement learning domain.



The

Figure 2.6: An overview of the reinforcement learning framework.

The agent, or learner, interacts with an environment through a set of actions. The environment, in turn, produces a new state for every action, with a corresponding reward. The reward may be zero (non-reward), positive (reward), or negative (penalty). A simple example of this interaction is illustrated through figure 2.7, below.

In this example, 2.7, the robot represents an RL **agent**. The robot has an **action space** consisting of four possible moves: {Move Left = ←, Move Right = →, Move Up = ↑, Move Down = ↓}. Whenever the robot chooses an **action**, the environment produces a new **state**, corresponding to a new grid with the robot's new position. With each transition, also called the **time step**, the environment also produces a reward, which as previously mentioned can be zero, positive, or negative. In this case, it is a penalty of minus one to prevent the agent from moving around aimlessly in the environment and encourage it to effectively find an optimal path to the target or the terminating state with a reward of 100 points. The long-term goal of a reinforcement learning algorithm would be to maximize the average accumulated reward of the robot, so that the robot reaches a certain goal, in this example, the target.

2.4.2 Exploration versus Exploitation

Let us say that the robot in figure 2.7 starts by exploring paths randomly. By chance, the robot may find a path to the treasure chests following the edge cells of the grid. This is of

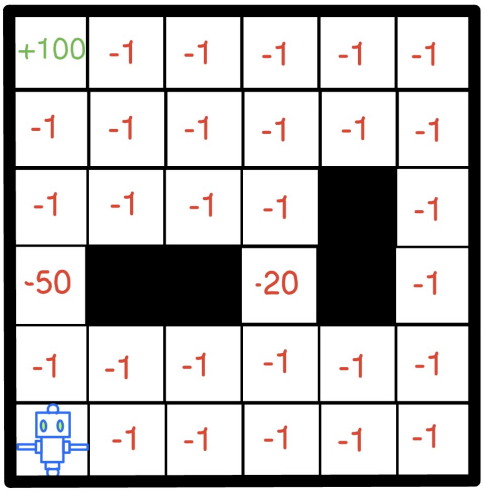


Figure 2.7: A reinforcement learning scenario, where the robot is the agent, and the grid world represents the environment.

course a suboptimal path, considering that the robot has other paths that would lead the robot to the treasure chest quicker. However, since the robot's goal is to maximize reward, the robot may choose to exploit its current knowledge and utilize the path every time. On the other hand, if the robot just explores a new path continuously, it will never leverage its learned knowledge. This trade-off between exploitation and exploration is a known as a crucial dilemma in reinforcement learning [32] and is important for finding good policies. One way to balance exploration vs exploitation would be to premiere exploration in the initial phases of the learning, to later exploit what is known.

2.4.3 The Credit Assignment Problem and Reward Shaping

The process in which you design the environment to produce rewards, e.g. in every time step or at the end of an episode, relates to an important challenge in the reinforcement domain, called the Credit-Assignment Problem, or as it was first called the Basic Credit-Assignment problem [25]. In the case of **sparse rewards** / **delayed rewards**, where the reward perhaps comes at the end of an episode consisting of a large sequence of actions, it is hard to assign credit to the actions that resulted in the reward. An example of this would be chess, where there are a wide array of actions taken by both players, which result in either a win, loss, or draw. Some of the actions may have been monumental towards the win, whereas others may have been insignificant. This problem can be alleviated by thorough **reward shaping**, for example, by placing rewards more closely in time to when they occur. However, this requires deep domain-specific knowledge and may result in unwanted behavior from the agent. If we take chess as an example again, one could reward the agent for capturing the queen, but this may result in the agent prioritizing the capture over a loss. Therefore, reward shaping must be done with utmost carefulness, and sometimes it is most beneficial to only provide a sparse reward, e.g. in the case of a win, which best aligns the agent with the goal of winning.

2.4.4 Bellman Equations

Some of the foundations for reinforcement learning were laid out as early as the 1950s, with Richard Bellman's work on dynamic programming and Markov Decision Processes [5] [4]. More specifically, he developed the principle of optimality and formulated what would later be known as the Bellman equation. These findings provided theoretical frameworks for how present decisions affect future outcomes. The principle of optimality states:

> "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy concerning the state resulting from the first decision".

The Bellman Optimality Equation for the state-value function $V^*(s)$:

$$V^*(s) = \max_a \sum_{s',r} P(s', r|s, a) \left[r + \gamma V^*(s') \right] \tag{2.1}$$

The Value Function, described in Equation 2.1, provides a method for finding the optimal policy. It states that the optimal value, $V^*(s)$, for a given state s, is obtained by choosing an action a that maximizes the expected accumulated reward. This includes not only the sum of immediate rewards r, but also the expected future rewards, discounted by a factor γ. This calculation considers the probabilistic nature of transitions, meaning that the outcome of any action a in a state s can lead to multiple possible next states s', each with a certain probability. The formula accounts for these probabilities, thus ensuring that the expected value of future rewards is properly weighted by the likelihood of each potential outcome. The process is accomplished recursively, enabling the algorithm to consider possible future states and their associated rewards.

The value function can be used to find the Optimal Policy π^* by:

$$\pi^*(s) = \arg\max_a \left\{ R(s, a) + \gamma \sum_{s'} P(s'|s, a) V^*(s') \right\} \tag{2.2}$$

The optimal Policy, found through the true value function, is nothing more than using equation 2.1 greedily. Using the equation greedily means using only locally available information [32]. This makes sense as equation 2.2 gives you the long-term value (accumulated reward) of a state for choosing a specific action which results in the next state s^*. Doing this greedily at each time step will result in the optimal policy π^*. defined in equation 2.2.

2.4.5 PPO and Deep Reinforcement Learning

In complex games like chess, where the true value functions are intricate and impossible to calculate due to the astronomical number of possible moves and states, approximating these functions is a common strategy. Neural networks are frequently employed due to their robust capability to approximate functions. This process involves parameterizing the value and policy functions, represented by theta, which encompasses the weights and biases of the

neural network. The parameterized value function $V(s;\theta)$ and policy function $\pi(a|s;\theta)$ can be defined as follows:

$$V(s;\theta) = f(s;\theta_V), \tag{2.3}$$

$$\pi(a|s;\theta) = \sigma(g(s,a;\theta_\pi)), \tag{2.4}$$

where f and g represent the neural network architectures for the value and policy functions, respectively. θ_V and θ_π are the parameters (weights and biases) specific to each function, and σ denotes the softmax function that outputs a probability distribution over actions.

PPO is an algorithm developed by OpenAI [30], particularly designed for updating neural networks in reinforcement learning contexts. It addresses some of the limitations observed in earlier methods like Deep Q-Networks (DQN) and Trust Region Policy Optimization (TRPO). PPO simplifies the approach to policy optimization by introducing a clipped surrogate objective function, which makes it computationally less demanding and easier to implement than TRPO. As an on-policy algorithm, PPO operates by learning from a batch of experiences gathered under the current policy. After these experiences are used for updating the neural networks, they are discarded, necessitating the collection of new experiences under the updated policy for further learning. Below follows the clipped surrogate object function (the loss function) from the OpenAi paper,

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

The advantage estimation at time step t, denoted by $\hat{A}_t$, is given by the following equation:

$$\hat{A}_t = -V(s_t) + r_t + \gamma r_{t+1} + \cdots + \gamma^{T-t+1} r_{T-1} + \gamma^{T-t} V(s_T) \tag{2.5}$$

Where:

- $V(s_t)$ represents the current value network estimation of being in S_t.
- $r_t, r_{t+1}, \ldots, r_{T-1}$ are the rewards received from time step t to $T-1$.
- γ is the discount factor. The idea is similar to the "time value of money" concept, where money received today is worth more than money received in the future.
- $\gamma^{T-t}V(s_T)$ applies the discount factor to the value function's estimation of being in the final state s_T, reflecting the value of ending in state s_T from time t.

Simply put, the advantage function quantifies the difference between the expected return as estimated by our value network and the actual return derived from the collected experiences.

This difference can be abstractly expressed by the following equation:

$$\text{Advantage} = \text{Actual Return} - \text{Expected Return} \tag{2.6}$$

Where:

- Expected Value represents the value function's estimation of the expected return from a particular state.
- Actual Return encapsulates the accumulated discounted rewards obtained from real experiences starting from that state.

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\theta_{\text{old}}}(a_t \mid s_t)} \tag{2.7}$$

The term $r_t(\theta)$ represents the likelihood of action under the new policy compared to the old policy. A value greater than 1 indicates that the action is more likely under the new policy, whereas a value between 0 and 1 means that it is less likely. When this ratio is multiplied by the advantage function, $r_t(\theta)\hat{A}_t$, the result quantifies how much better or worse an action is under the new policy relative to the baseline policy's average action.

The term $\textbf{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t$ modifies the probability ratio by clipping it within the range $[1-\epsilon, 1+\epsilon]$. This ensures that the policy update does not deviate too drastically from the old policy, specifically by the adjustments to be within the bounds set by ϵ, hence preventing drastic changes due to large advantage estimates or highly favorable/unfavorable actions under the old policy.

Using the `min` function, as shown in equation 2.8, enables the selection of the smaller of the two terms. This approach ensures a more cautious update, preventing the new policy from deviating significantly from the parameter space of the old policy.

$$\min\left(r_t(\theta)\hat{A}_t, \text{clip}\left(r_t(\theta), 1-\epsilon, 1+\epsilon\right)\hat{A}_t\right) \tag{2.8}$$

When having a shared network between the policy and the value network, a composite loss function at time t for policy parameters θ can be defined as:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t\left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)\right] \tag{2.9}$$

Where:

- $L_t^{CLIP}(\theta)$ represents the clipped policy loss.
- $L_t^{VF}(\theta)$ is the value function loss, scaled by the coefficient c_1.
- $S[\pi_\theta](s_t)$ denotes the entropy bonus for the policy π_θ at state s_t, scaled by c_2.

Equation 2.9 introduces two additional hyperparameters, c_1 and c_2. c_1 is the coefficient for the value function, which determines the priority given to updating the value function. A higher c_1 assigns greater importance to minimizing the value function loss. c_2 is the coefficient for the entropy bonus, which encourages exploration. Setting c_2 high promotes more exploratory behavior in the policy, and vice versa.

2.5 Feature Extraction

The combination and integration of deep learning and reinforcement learning (RL) has led to significant advancements in solving complex decision-making tasks. In RL, agents need to make optimal sequential decisions to maximize cumulative rewards. Leveraging deep neural network (DNN) architectures, such as convolutional neural networks (CNNs), to extract meaningful features from the environment's observations is crucial for achieving high performance.

Feature extraction from the input data is a crucial aspect of deep learning, particularly in tasks where visual perception is essential. It involves transforming raw input data into a

smaller set of features that can be effectively used by machine learning algorithms. Feature extraction is particularly important for RL-visual tasks that are difficult to solve using a predefined set of rules, like autonomous driving, robotics, healthcare, surveillance, and simulation such as mastering complex games like Go [29]. For instance, Google DeepMind's AlphaGo Zero was trained by RL algorithms, using neural networks consisting of many residual blocks of convolutional layers [31]. Although integrating RL in these tasks has offered advantages, working with visual data remains challenging due to several factors:

- **High Dimensionality**: *Large state* space in RL due to high-dimensional image data.
- **Sample Inefficiency**: RL agents require many samples for effective learning, which is both time-consuming and computationally expensive.
- **Robustness and Generalization**: Robustness to variation in the visual environment and generalization to unseen scenarios are crucial for RL models.

Reducing the dimensionality of raw input data through feature extraction helps address these challenges. Extracted features highlight the most relevant aspects of the environment for the agent, helping it to optimize its decision-making ability.

2.5.1 Atari Environment - Breakout

Atari games have become benchmarks in reinforcement learning (RL) research due to their well-defined rules, simplicity, and visual complexity. Breakout, a classic arcade game, requires players to control a paddle to hit a ball toward a wall of bricks while preventing the ball from falling off the screen. Training an agent in this environment involves understanding the spatial layout and making optimal decisions. The robot's action space includes four moves: {NOOP, FIRE, RIGHT, LEFT} [18]. "NOOP" means no operation, "FIRE" starts the game, and "RIGHT" and "LEFT" move the paddle accordingly. The environment generates a new state for each action. Each game, or episode, grants the agent five lives. The maximum score is 864 points, achieved by clearing two walls of bricks, each worth 432 points. The bricks are arranged in rows, with the first two rows (blue and green) worth 1 point each, the next two rows (yellow and brown) worth 4 points each, and the final two rows (orange and red) worth 7 points each, totaling 432 points per wall.

Figure 2.8: A state generated by the Atari Breakout environment.

Chapter 3
Methodology

In this chapter, we will explore our methodology for evaluating the potential of training a reinforcement learning (RL) agent to play the Atari game Breakout through the use of features derived from a convolutional neural network (CNN). We will start by discussing our experimental setup, followed by an overview of the study design, which includes benchmark replication, an ablation study, hyperparameter sweep optimization, and the main experiment: evaluating the integration of ResNet-50 as a feature extractor.

3.1 Experimental Setup

All the experiments were run on the same computer, with the following specifications:

- **Operating system**: Ubuntu 22.04.4 LTS x86_64
- **Processor**: 13th Gen Intel i7-13700K (24 cores)
- **RAM**: 64 GB
- **GPU**: NVIDIA GeForce RTX 4080

The main technologies used were:

- **Python**: The primary programming language used for implementing the models and running experiments.
- **Stable Baselines3**: A set of reliable implementations of reinforcement learning algorithms.
- **PyTorch**: An open-source machine learning library used for developing and training deep learning models.

- **Weights & Biases (Wandb)**: A tool for tracking experiments, visualizing metrics, and managing hyperparameters.
- **Gymnasium**: A toolkit for developing and comparing reinforcement learning algorithms through a standard API and a variety of environments.

The model of choice was a PyTorch implementation of the ResNet-50 architecture, which was trained on the ImageNet dataset for classification purposes, covering 1000 classes [14]. We choose Resnet50 because of its proven efficiency in the computer vision domain, achieving an accuracy of 80.858% for the top-1 classification and 95.435% for the top-5 classification. [14]

3.2 Study design

We conducted a comprehensive multiphase study that included the following stages:

1. **Replication of Benchmark**: We began by replicating an established benchmark to ensure the validity of our setup and approach.
2. **Ablation Study**: Next, we performed an ablation study to determine the contribution of each component to the overall performance.
3. **Hyperparameter Sweep**: Following this, we conducted a hyperparameter sweep to investigate whether there were better hyperparameters.
4. **Comparison of Feature Extraction Stages in ResNet-50**: Aside from determining how well Resnet-50 performs as a feature extractor for image-based reinforcement learning, we also wanted to evaluate if there was any difference in performance when using the extracted features from the last block of each of the four stages in Resnet-50, both using the benchmark hyperparameters and the hyperparameters found through the sweep study mentioned in the previous step.

3.3 Benchmark Replication

We replicated a benchmark from the RL Zoo3 repository [27] for our training. Identical preprocessing, initialization, frame skipping, and stacking methods were employed, along with the same hyperparameters. Our implementation utilized the Stable Baselines3 RL library [15] within the Atari Breakout Gymnasium environment, specifically BreakoutNoFrameskip-v4 [18]. Detailed explanations of these methodologies are provided in the sections that follow.

3.3.1 Preprocessing

The default image output from the Gymnasium environment is a 210×160 RGB image [18], which we converted to a grayscale image and resized to 84×84 using the cv2 Python library. This process is illustrated below in Figure 3.1.

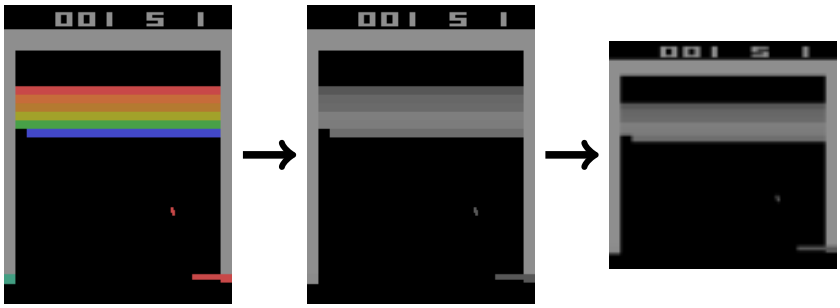


Figure 3.1: An overview of the preprocessing for the benchmark model. Left: original, center: after grayscale, right: final result.

3.3.2 Initialization

To introduce variability in the initial game state and prevent the agent from always starting from the same conditions, we employ a no-op (no operation) initialization. Upon resetting the environment, the agent performs a random number of no-op actions (between 1 and 30). This process allows the ball to start in different positions. By varying the starting conditions, the agent is exposed to a wider range of scenarios, which should help it develop a more generalized understanding of the game dynamics.

3.3.3 Frame Skipping with Action Repetition

To optimize computational efficiency, the frame-skipping technique with action repetition was utilized. The agent only selects an action every k-th frame. For the intermediate $k - 1$ frames, the same action is repeated. This approach was utilized in various studies, including [12], for Atari games. According to the authors, this technique allows the agent to play approximately k-times more games as the computational load of repeating an action that advances the environment, is less than letting the agent calculate a new action at every frame. We choose a k value of four, meaning that the agent chooses one action, and then that action

is repeated for the subsequent three frames. This process is illustrated in figure 3.2, below, where frames without a red cross illustrate the agent choosing an action and the subsequent three frames are repeated actions.

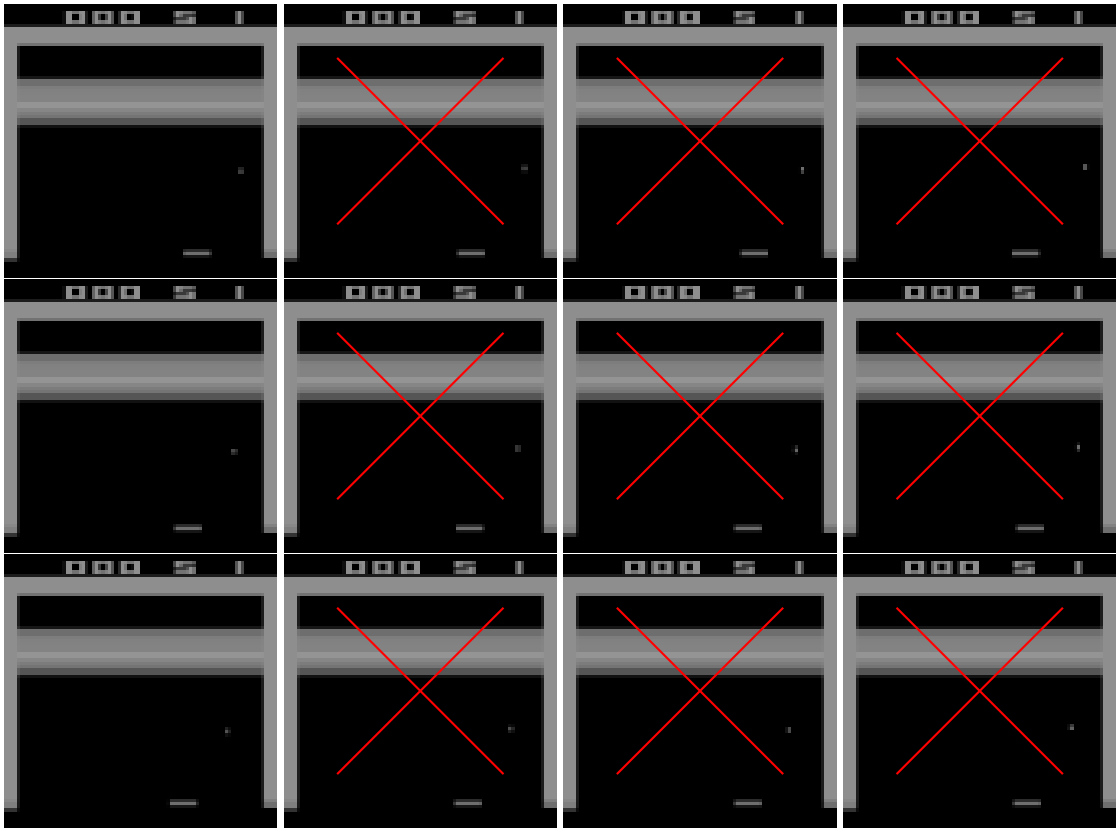


Figure 3.2: Frame skipping with action repetition. Original images represent chosen actions. Images with red crosses represent repeated actions.

3.3.4 Frame Stacking

Similar to what was done in to [12] and [13], we employed a frame stacking technique. This means that instead of inputting independent frames, stacks of frames were used. This is expected to help the agent make better decisions, giving it a sense of motion and temporal context. We used a stack of four frames, frames$(t, t+4, t+8, t+12)$, as was done in [12] and [13]. If we represent these frames as x_n, we get a state representation $S_1 = (x_1, x_2, x_3, x_4)$. This state representation can be described as a sliding window with a stride of one. Applying this method to the next state would give $S_2 = (x_2, x_3, x_4, x_5)$, and so forth. The frame stacking was only applied after each frame had gone through the preprocessing. These stacked frames, were used in both training and evaluation of the models, after the aforementioned preprocessing had been applied.

3.3.5 Hyperparameters

We used the hyperparameters listed in the table 3.1. All hyperparameters are exactly as in the benchmark taken from the rl-zoo3 repository [27].

Hyperparameter	Value
Algorithm	PPO
Number of Timesteps	10,000,000
Frame Stack	4
Learning Rate	0.00025
Learning Rate Schedule	Linear
Batch Size	256
Number of Steps	128
Number of Epochs	4
Gamma	0.99
GAE Lambda	0.95
Clip Range	0.1
Clip Range Schedule	Linear
Entropy Coefficient	0.01
Value Function Coefficient	0.5
Max Gradient Norm	0.5

Table 3.1: Hyperparameters for PPO algorithm for training the benchmark model.

3.3.6 Architecture

The network employed a convolutional neural network (CNN) architecture. It included three convolutional layers, each followed by a ReLU activation function, to enhance the non-linear properties of the decision function and model complex relationships in the data. After the convolutional layers, the output was flattened into a feature vector of size 3136. This vector was then passed to a fully connected layer that reduced its dimensionality to a 512-dimensional feature vector, serving as a consolidated representation of the input data. Subsequently, this feature vector is fed into two distinct branches of the network: the policy network (actor) and the value network (critic). The policy network outputs a four-dimensional vector, representing the probabilities of the four possible actions in Breakout, while the value network produces a single scalar value, estimating the current state's value. Both outputs are crucial for the reinforcement learning process, employing the Proximal Policy Optimization (PPO) algorithm to perform policy gradient updates effectively. An illustrative diagram of the network architecture is available in Figure A.1 of the appendices. In total, the architecture comprised approximately 1.686 million tunable parameters.

3.4 Ablation study

To comprehensively evaluate the factors affecting the performance and specifically assess the impact of integrating ResNet-50, including how adapting images to meet ResNet-50's input specifications influences outcomes, we conducted an ablation study. Please note that this is not an ablation study in the traditional sense, which involves removing components. Instead, this is more akin to a reverse ablation study, as we are adding components. However, for simplicity, it can be considered an ablation study. Throughout all experiments, we maintained consistent settings for initialization, frame skipping with action repetition, and frame stacking as per the benchmark protocol. The primary objective was to isolate and examine the effects of modifications made for ResNet-50 adaptation to pinpoint their individual contributions to overall performance. For this study, we specifically utilized the last block of stage four of ResNet-50 to explore its efficacy as a feature extractor. Later on, in the main experiment section, the last block of all the ResNet-50 stages will be evaluated. The preprocessing required for ResNet-50 included resizing images to 232×232 via bilinear interpolation, cropping to 224×224, and normalizing using the means $[0.485, 0.456, 0.406]$ and standard deviations $[0.229, 0.224, 0.225]$ as specified by PyTorch [14]. For this study, we streamlined the image processing by directly resizing images from 210×160 to 224×224, bypassing the intermediate resizing to 232×232 and subsequent cropping. We reasoned that the protocol of resizing to 232×232 before cropping to 224×224 was developed for images from the ImageNet database, which are of higher dimensionality and typically require downsizing before cropping. This method contrasts with our approach, where the frames needed to be upscaled from a lower dimensionality of 210×160 to 224×224 to meet the input specifications of ResNet-50.

The experiments were organized as follows:

- **Experiment 1**: Used the benchmark grayscale images at 84×84 (benchmark replication).
- **Experiment 2**: Skipped the grayscale-conversion of the 84×84 RGB images to assess the impact of adding color.
- **Experiment 3**: Resized the original RGB 210×160 to 224×224 to assess the impact of resizing the observation frame.
- **Experiment 4**: Normalized the resized RGB frame by the mean and standard deviation values provided by Pytorch, as previously mentioned, to assess the impact of normalizing the observation.
- **Experiment 5**: Integrated the ResNet-50 architecture, using features from the last block of stage four, to determine the impact of using extracted features instead of observation frames.

All of these steps are illustrated concisely in Table 3.2, below.

Experiment	Grayscale	RGB	Resizing	Processing	ResNet
1	X				
2		X			
3		X	X		
4		X	X	X	
5		X	X	X	X

Table 3.2: An overview of the ablation study. The X's represent the components that were used in that specific study.

3.5 Hyperparameter Sweep

Since our ablation study showed sub-par performance for the last block of the last stage in ResNet-50, we decided to conduct ten hyperparameter sweeps using Weights and Biases (W&B) [6], each consisting of one million time steps. Due to time constraints and computational demands, our focus was solely on optimizing the following hyperparameters using a Bayesian search. This decision was influenced by rl-benchmark, which modified only these parameters while keeping everything else at default settings. The goal of our Bayesian search was to maximize the mean episodic reward.

The hyperparameters and settings used for the sweep are listed below:

Parameter	Distribution	Min	Max	Description
batch_size	int_uniform	128	512	Batch size
clip_range	uniform	0.01	0.1	Clip range
ent_coef	uniform	0.001	0.01	Entropy coefficient
learning_rate	uniform	0.00125	0.005	Learning rate
n_epochs	int_uniform	1	10	Number of epochs
n_steps	int_uniform	64	256	Number of steps
normalize_advantage	values	false	true	Normalize advantage
vf_coef	uniform	0.25	1	Value function coefficient

Table 3.3: Sweep configuration parameters

3.6 Main experiment: Resnet-50 Evaluation

We conducted an evaluation of the last block in four different stages of ResNet-50 using two sets of hyperparameters: one set was the benchmark parameters detailed in table 3.1, and the other set was identified through a hyperparameter sweep, as discussed in the results chapter. It's important to note that the experiment for the end of stage 4 has already been conducted as part of the ablation study.

In all these experiments, we maintained consistency with the benchmark configurations in terms of initialization, frame skipping with action repetition. Additionally, we imple-

mented the preprocessing steps for ResNet-50 that were outlined in the ablation study. Details on the architectures used for each stage are provided in the appendices (see Figures A.2, A.3, A.4, A.5). Frame stacking was handled differently compared to the approach employed by the benchmark due to technical constraints imposed by ResNet-50. We chose to process the frames separately, and only concatenate the feature vectors at the end, forming the input to the RL model. This process is detailed further below, and illustrated in figure 3.3.

The feature maps generated by each stage of the ResNet-50 are averaged and stacked to form the input for the subsequent stage and layer. We define:

- C: the number of feature maps output by the last block of each stage,
- N: the number of feature vectors to be stacked (in this case, $N = 4$).

For a given block, the output consists of C feature maps. After global average pooling, the feature maps are reduced to a one-dimensional feature vector of size C. These vectors are then stacked to form the input to the next stage, as follows:

1. **Output after pooling**: A feature vector v of size C.
2. **Stacking**: Stacking N such vectors yields an input vector V for the next stage, calculated as:
$$V = [v_1, v_2, v_3, v_4]$$
where each v_i is of same dimension and size, C.
3. **Input Dimensionality**: The total size of the input vector V is $N \times C$. Example for stage1, where $C = 256$ and $N = 4$, the input dimension becomes:
$$|V| = 4 \times 256 = 1024$$

C varies for the end of stage1, stage2, stage3, and stage4, with values of **256**, **512**, **1024**, **2048** respectively.

This input structure is pivotal for the architecture, as it handles feature maps from the previous stage and prepares them for further processing in the subsequent stage. The whole process of the main experiment is illustrated in Figure 3.3, below.

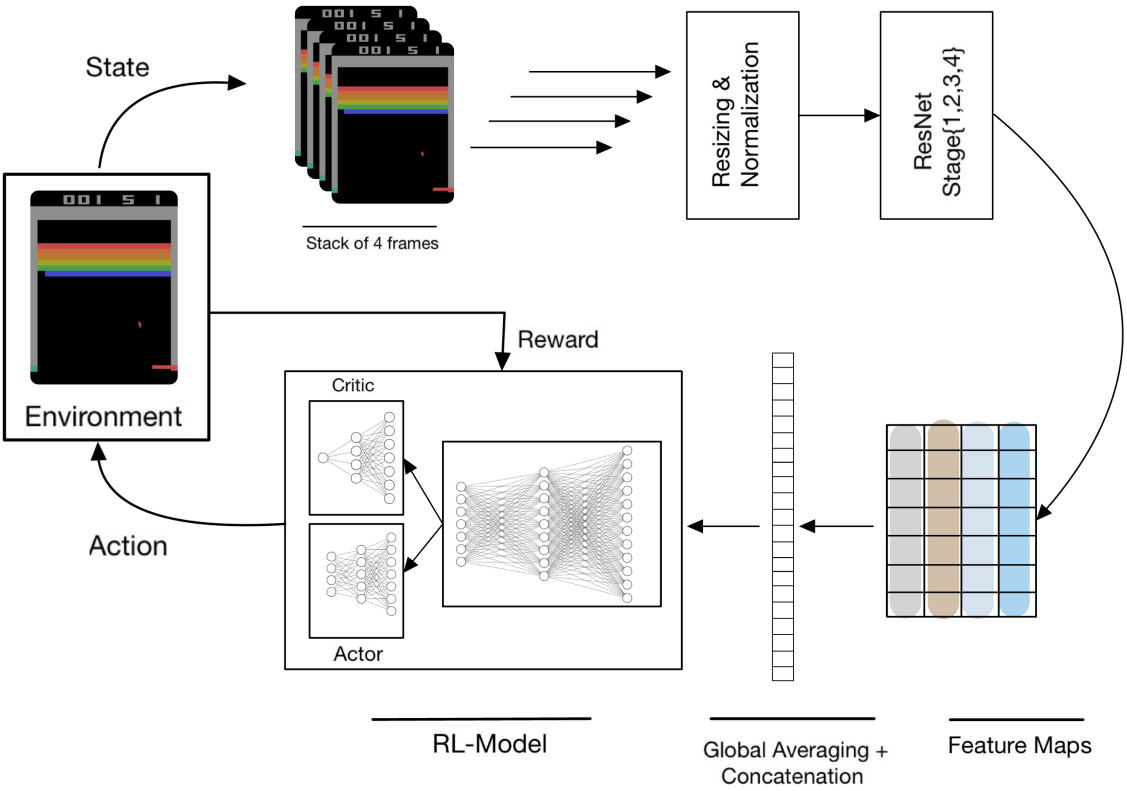


Figure 3.3: Main experiment pipeline. A stack of four RGB frames is resized, normalized, and fed to the ResNet-based feature extractor. Global averaging is applied to the resulting feature maps, reducing each feature map into a single value. Resulting feature vectors are concatenated together before being fed to the RL model.

Chapter 4

Results

This chapter presents the findings from our multiphase study. The graphs, except for those representing the best model, were created by running five evaluation episodes every 50,000 time steps, over a total of 10,000,000 time steps. We calculated the mean episodic reward (averaged accumulated reward) and the mean episodic length over five episodes. The best models were identified through these periodic evaluations during the training phase. The graphs for the best models were generated by first evaluating the models over 25 episodes and then averaging the rewards to ensure robust results for each model.

4.1 Benchmark

This section provides the mean episodic rewards and mean episodic length as key metrics of the replicated benchmark.

4.1.1 Mean Episodic Reward

Figure 4.1 illustrates the mean episodic reward averaged over five episodes, for the benchmark model. The slightly opaque trend lines in the background represent the raw, unsmoothed data. This visualization demonstrates how the model converged to a stable performance level of approximately 400 points. The benchmark experiments are referred to as experiment 1 in the ablation study, hence the naming in the legend.

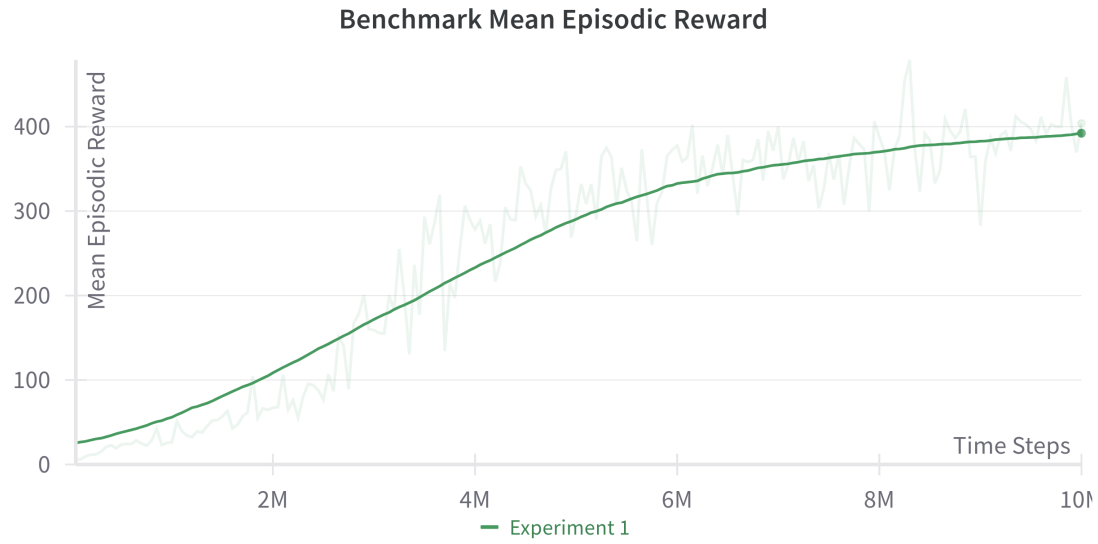


Figure 4.1: Benchmark mean episodic reward averaged over five games.

4.1.2 Mean Episodic Length

Figure 4.2 displays the mean episodic length, averaged over five episodes. This graph is unsmoothed to highlight the spikes in mean episodic length. The data reveals frequent and significant spikes, with levels occasionally exceeding 100,000 time steps.

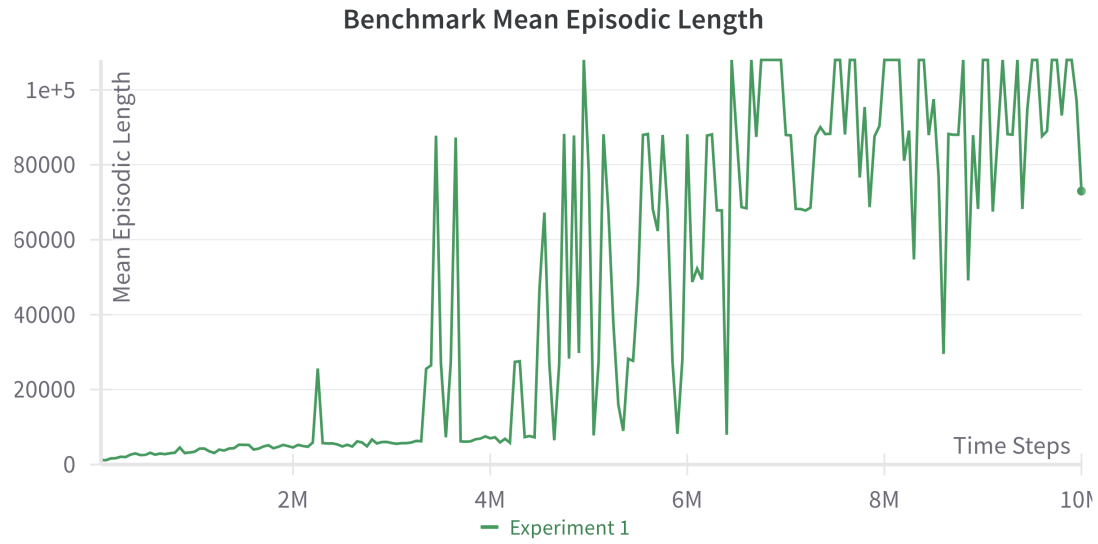


Figure 4.2: Benchmark mean episode length averaged over five games.

4.2 Ablation Study

This section provides the mean episodic rewards of the five experiments in the ablation study, together with mean episodic length of the first and last experiments in the ablation study representing the benchmark and integration of ResNet-50 as a feature extractor, respectively. Furthermore, it provides the metrics for the best model obtained for each experiment.

4.2.1 Mean Episodic Reward

Figures 4.3 illustrates the running average of mean episodic reward for the five experiments in the ablation study. The slightly opaque trend lines in the background represent the raw, unsmoothed data. Experiment 5, in which we included the ResNet-50 architecture, performed significantly worse, as can be noted from the graph.

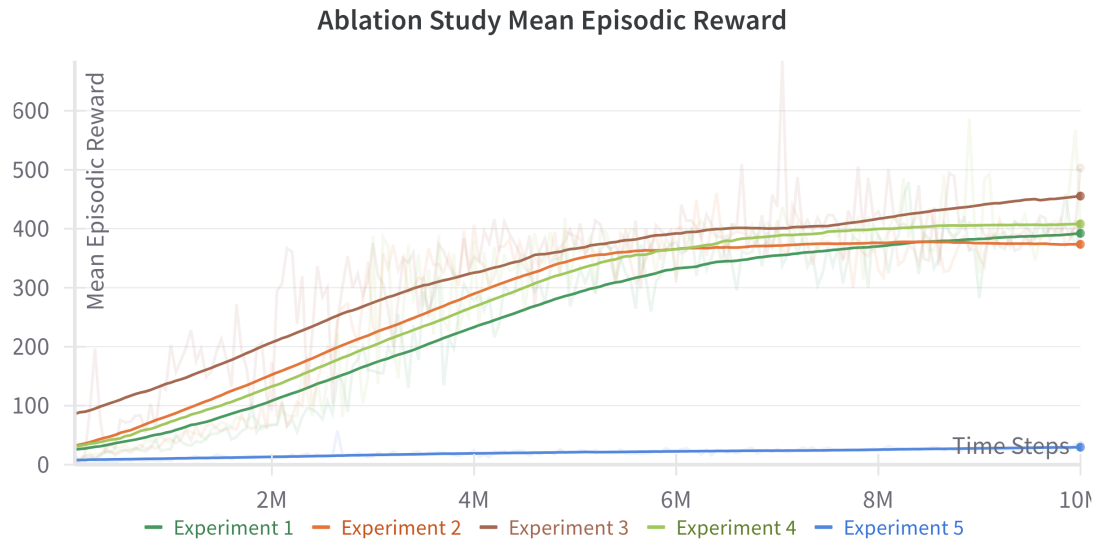


Figure 4.3: Mean episodic reward averaged over five games for the five experiments in the ablation study.

4.2.2 Mean Episodic Length

Figure 4.4 displays the mean episodic lengths for experiment 1 (the replicated benchmark) and experiment 5. Notably, experiment 5 also demonstrated spikes in mean episodic length as observed in the ablation study, with an average trend around 2700 time steps. To avoid cluttering the graph, experiments 2 through 4, which followed similar trends to experiment 1, were omitted.

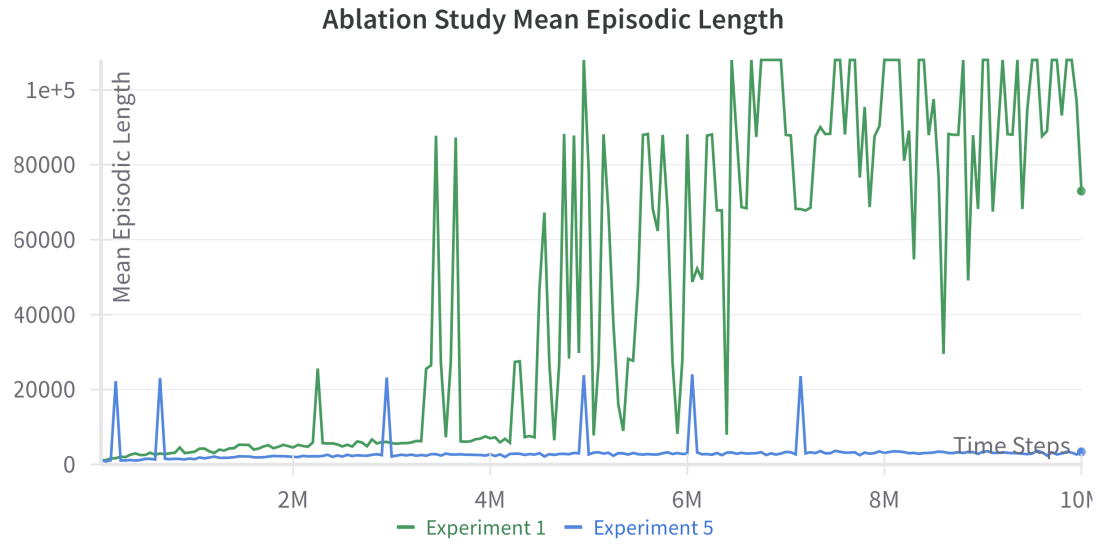


Figure 4.4: Mean Episodic Lengths for experiments 1 and 5 in the ablation study.

4.2.3 Best Model Mean Episodic Rewards

Once the best model for each of the experiments had been identified, they were evaluated again over 25 episodes to ensure robustness. Table 4.1 shows the Mean Episodic Reward. Most experiments in the ablation study converged around a mean episodic reward of 400 or above, whereas the experiment including the ResNet-50 architecture only achieved a mean episodic reward of around 30.

Experiment	Mean Episodic Reward	Standard Deviation
Experiment 1	410	19
Experiment 2	409	21
Experiment 3	409	21
Experiment 4	461	167
Experiment 5	30	5

Table 4.1: Summary of mean episodic rewards and their standard deviations for various experiments. The numbers have been rounded to the nearest integer.

4.3 Hyperparameter Sweeps

Figure 4.5 below, presents the results of ten runs conducted with varying hyperparameter values, each running for one million time steps. The yellow line represents the best-performing run. The specific hyperparameters used for this optimal performance are detailed in Table 4.2, below.

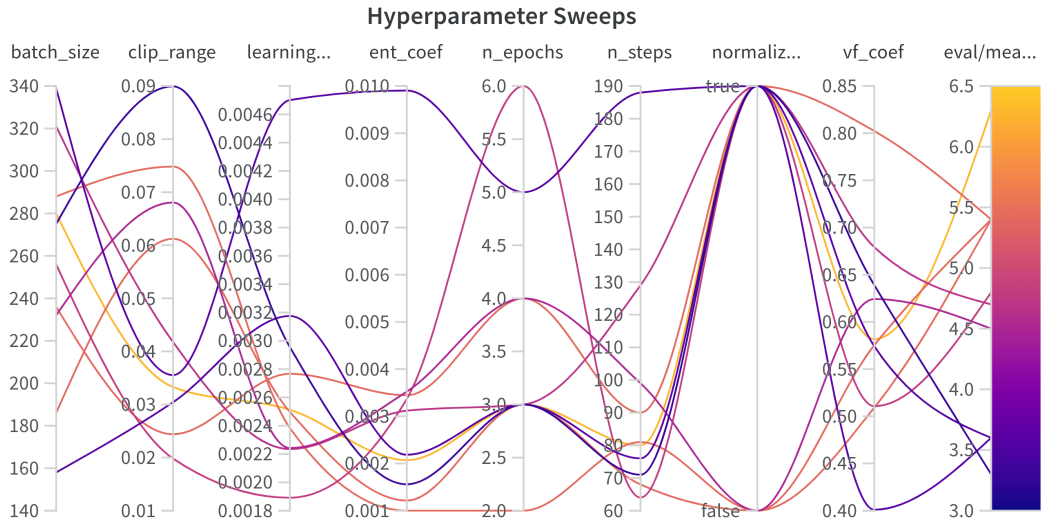


Figure 4.5: Overview of the configurations pertaining to the 10 hyperparameter sweeps.

Hyperparameter	Value
Batch Size	280
Clip Range	0.3329
Learning Rate	0.002513
Entropy Coefficient (ent coef)	0.00207
Number of Epochs (n_epochs)	3
Number of Steps (n_steps)	80
Normalize Advantage	True
Value Function Coefficient (vf_coef)	0.5818

Table 4.2: Hyperparameters for the best-performing run.

4.4 Main Experiment: Comparison of Feature Extraction Stages in ResNet-50

This chapter presents the results from the "Comparison of Feature Extraction Stages in ResNet-50 experiments. These results are derived from models trained using both the benchmark hyperparameters and the best-performing hyperparameters identified through hyperparameter sweep runs, as detailed in the methodology section. Note that "Benchmark Hyperparameters Stage 4", is equivalent to "Experiment 5", in the ablation study.

4.4.1 Mean Episodic Reward

Figure 4.6, shown below, displays the mean episodic reward for the different stages in ResNet-50, for the runs with both the benchmark hyperparameters and the hyperparameters of the best-performing sweep run. This figure shows that the only model displaying any form of learning is the one utilizing the features outputted by the end of stage 4. A comparison with the benchmark will be illustrated in the upcoming section "Comparisons".

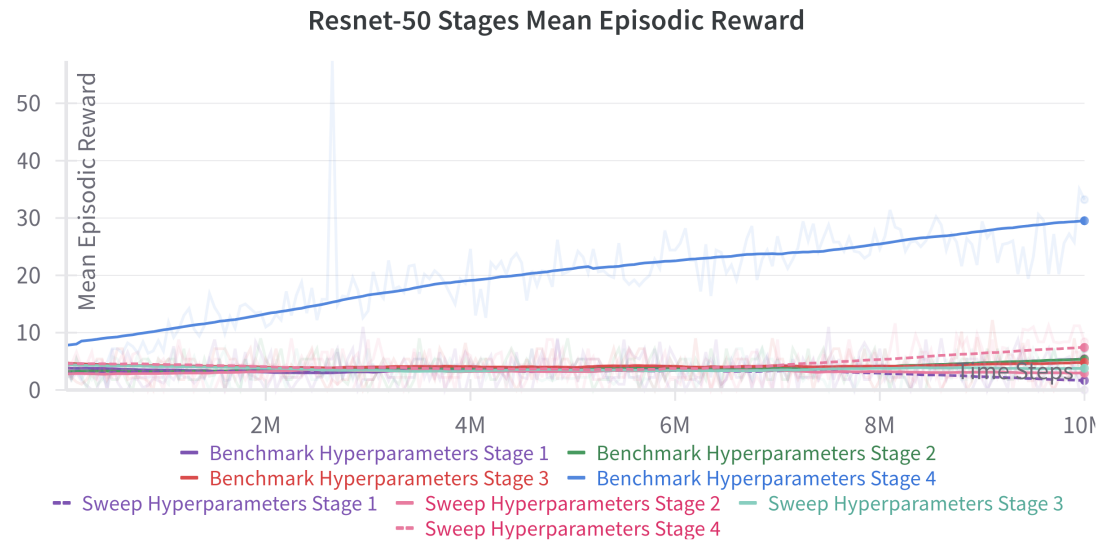


Figure 4.6: Mean episodic rewards for the four stages in ResNet-50 with the benchmark hyperparameters and the best performing hyperparameters obtained from the hyperparameter sweep.

4.4.2 Mean Episodic Length

Figure 4.7 illustrates the mean episodic lengths for stages 3 and 4. These stages were specifically selected to avoid overcrowding the graph. All other stages exhibited a similar trend to stage 3, including those using the hyperparameters from the most effective sweep. Typically, the values hover around 1500 mean episodic lengths but exhibit sudden spikes reaching up to 20,000 and even 40,000.

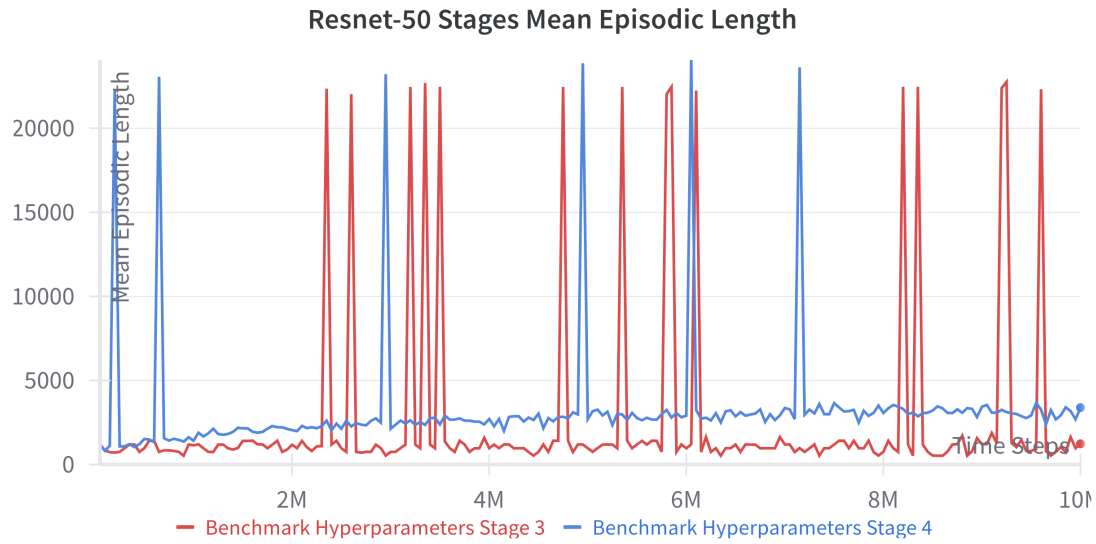


Figure 4.7: Mean episodic lengths across stages 3 & 4 in ResNet-50.

4.4.3 Feature Maps

For the convenience of the reader, we have included visualizations of one of the feature maps generated at the end of each residual stage. This helps to intuitively illustrate the differences between the stages. These visualizations can be viewed in Figure 4.8.

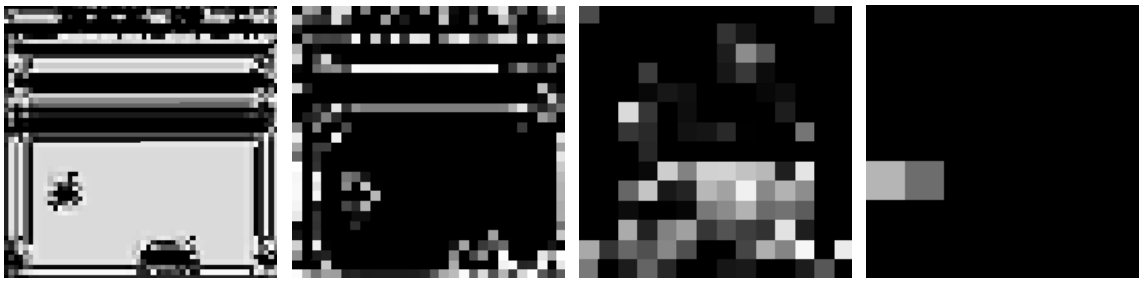


Figure 4.8: One feature map illustrated from each of the different stages of ResNet-50. Left: stage 1, left center: stage 2, right center: stage 3, right: stage 4

4.5 Comparisons

Table 4.3, below, displays a comparison of performance metrics for the benchmark and the four stages of ResNet-50, using both benchmark hyperparameters and the best-performing sweep hyperparameters. The table also includes the training times for each experiment, which includes evaluations performed at every 50,000 time steps. Thus, while the table primarily serves as an indicator of training duration, it also provides a comparison of the training efficiency across different stages.

Experiment	Mean Episodic Reward	Standard Deviation	Training Time
Benchmark (Experiment 1)	409.52	18.16	4h 15m
Benchmark Hyperparameters			
Stage 1	2.16	3.84	18h 11m
Stage 2	2.88	4.20	21h 5m
Stage 3	2.88	4.20	1d 6h 9m
Stage 4 (Experiment 5)	29.84	5.00	1d 9h 23m
Sweep Hyperparameters			
Stage 1	6.16	5.46	19h 24m
Stage 2	1.31	3.58	1d 4h
Stage 3	2.88	4.20	1d 8h 15m
Stage 4	6.16	6.95	1d 18h 19m

Table 4.3: Comparison of mean episodic rewards, standard deviations, and training times across different stages in ResNet-50 and hyperparameter settings.

Chapter 5
Discussion

The primary objective of this study was to investigate to what extent it was possible to use a large pre-trained CNN-based model as a versatile feature extractor in RL settings. Cross-domain generalizability enables pre-trained models to perform well on different data sets without significant performance degradation. In this study, we chose to investigate and identify the strengths and limitations of ResNet-50 as a feature extractor for training an RL agent in the Atari Breakout environment. This discussion will summarize the key findings and interpret the results. The discussion will critically evaluate the chosen and employed methodology, acknowledge the encountered strengths and limitations, and explore the challenges faced in achieving cross-domain generalizability. Also, this section will suggest directions for future research to address the challenges encountered. Through a comprehensive analysis of our findings, we strive to contribute to the ongoing efforts to develop robust image feature extraction methods.

5.1 Key Findings

Following are the key findings from our experiments using ResNet-50 as a feature extractor in the Atari Breakout environment:

First, we observed that integrating ResNet-based feature extractors generally exhibited poor performance in our study, despite having shown highly successful results in single-domain scenarios [21]. Specifically, the performance of the last stage of the ResNet-50 model was significantly different from the other three stages. This suggests that the representation level from different stages plays a crucial role in the effectiveness of extracted features.

Second, the results highlighted the impact of hyperparameter optimization on the performance of the feature extractor. Using benchmark hyperparameters gave better results compared to those with other settings. This underscores the significance of tailored hyperparameter configurations in optimizing model adaptability or generalizability to new domains.

Third, the results revealed that preprocessing techniques had significant effect on the

performance of the benchmark model. Methods such as grayscale transformation, normalization, and cropping had an impact on the learning speed, while the models converged to similar values. This suggests that a robust preprocessing pipeline is crucial for a versatile image feature extraction model.

In summary, our results suggest that there are multiple factors, besides the choice of pretrained model, that affect the cross-domain generalizability in image feature extraction. This emphasizes the need for careful consideration of model architecture, hyperparameter tuning, and preprocessing techniques to enhance the robustness of feature extraction methods.

5.2 Evaluation of Methodology

We employed a systematic approach in our evaluation of the feature extraction techniques and capabilities of ResNet-50 in this study. The goal of our experimental setup and design of the methodology was to ensure that the results were reliable and replicable.

First, we replicated the benchmark as a sanity check to ensure our experimental setup was functional and worked as expected. This is a necessary step that ensures that the performance of the subsequential experiments relies on factors other than the experimental setup. Then, we performed an ablation study to evaluate the impact of different processes, choices, and pipeline components on the overall performance of the study metrics. This step is crucial for identifying the importance and impact of different choices and components in the preprocessing pipeline, which in turn reduces uncertainties in the experiment. After finding an optimal configuration and design, we performed a hyperparameter optimization to evaluate how sensitive the experiment or pipeline was to hyperparameters. Optimizing hyperparameters is an important step in training ML models because hyperparameters govern the learning process and model structure, which in turn impact performance and efficiency. We train the RL agent with different stages as a feature extractor, with both the optimal set of hyperparameters identified through sweep, and the benchmark hyperparameters. Finally, we compare the results for both hyperparameters and stages.

5.2.1 Frame Stacking and Global Averaging

It is worth mentioning that the choice of frame stacking and global averaging methods impact the feature dimensions, and therefore might also play important roles in the effectiveness of the extracted features. Following the benchmark suggestion, we worked with stacks of four frames. ResNet-50 takes in 3-dimensional input data, and we used RGB frames, so our choice of giving ResNet-50 information about the dynamics of the environment was limited to concatenating four frames after running global averaging pooling on the extracted feature maps. This might have been a suboptimal choice or even a critical flaw in our approach when compared to the benchmark, which operates on stacks of four grayscale frames, enabling it to observe the stacked frames as a whole. ResNet-50 could also work with stacked frames if we had used a stack of three grayscale observations, since grayscale observations are 2-dimensional. It is however uncertain how this would have affected the performance, as ResNet-50 was not trained in this manner. Overall, a more fair comparison might have been to train a benchmark without stacked frames, and then compare the performances.

We could have also used an alternative method for the global average pooling layer that currently averages each feature map from the ResNet feature extractor to only a single value. Averaging a whole feature map to a single value can therefore be seen as information loss. An alternative method would have been to do pixel-wise averaging followed by a flattening layer to preserve more spatial information in feature maps, or perhaps look into other dimensional reduction techniques that would have resulted in less information loss. This would reduce the stack of four frames to only one frame embedded with temporal features and information on the environment dynamics. Consequently, it would reduce the computation time by 75% for the feature extraction part.

5.2.2 Reinforcement Learning Network Architectures

A possible explanation for ResNet-50's underperformance relative to the benchmark might be attributed to differences in network architectures. The network used for training the benchmark had 1,686,693 tunable parameters, including a CNN-based feature extractor which is trained to generate specific and optimal features for training the RL agent. In contrast, the architecture used with ResNet-50 to train the RL-agent incorporated a fully connected network with tunable parameters ranging from 139,845 for stage 1, to 1,057,349 for stage 4. This disparity in network complexity and parameter count may also account for the observed performance variations across different stages and the benchmark. However, it is highly unlikely that it accounts for all the difference in performance between the stages and the benchmark. For details regarding the architectures, see appendices.

5.3 Evaluation of Results

The following section discusses the results obtained from the different phases of the study.

5.3.1 Benchmark Replication

We replicated the *ppo-BreakoutNoFrameskip-v4* benchmark provided by RL-Zoo with the same settings and set of hyperparameters to establish a solid baseline for our experiments. Figure 4.1 shows that the mean episodic rewards converge stably towards 400 verifying that our implementation is consistent with the benchmark established and ensures that subsequent experiments and modifications are grounded in a validated starting point. Figure 4.2 shows the mean episodic length without running average smoothing. We can observe large spikes that stretch above 100,000 time steps. Visual investigation of the agent playing the game revealed that these spikes are caused by indefinite loops when the agent gets stuck with a particular break that it never successfully hits or some particular trajectory that results in the same trajectory again. The deterministic nature of the policy prevents the agent from getting out of the loop. The environment has a stop criterion of 108,000 time steps which resets the environment, enabling the agent to continue with a new episode. It is important to note, however, that this only happens in the evaluation, as it employs a deterministic policy. Fixing this issue, would've reduced the training time (the evaluations were run while training).

5.3.2 Ablation Study

The ablation study following the benchmark replication involved a series of controlled experiments where we systematically varied one component at a time. Table 3.2 provides the key factors examined.

Experiment 1 represents the benchmark with grayscale observations and acts a reference for the remaining experiments. It should be noted that the Atari Breakout environment returns RGB observations, and grayscale transformation is part of the preprocessing steps in the benchmark. Therefore, in experiment 2, we decided to investigate and evaluate the performance difference between grayscale and RGB observations by skipping the grayscale transformation in the preprocessing step. This was particularly important because ResNet-50 also works with 3-dimensional (RGB) input images. The performance difference was not significant, since color information gets lost when RGB images propagate through convolution layers.

Next, in experiment 3, we scaled the observation image from 210×160 to 224×224 to match the dimensionality required by the ResNet-50 architecture. The purpose was to evaluate the importance of scaling or resizing the environment observations. Resizing the original, 210×160 observation, resulted in both faster convergence and higher episodic mean rewards. It can be assumed that the higher resolution images provides more information for the network, enabling it to extract richer features.

Normalizing the input data is part of the ResNet-50 preprocessing steps, as outlined by PyTorch. Therefore, in experiment 4, we normalized the observations with the mean and standard deviations obtained by ResNet-50 from the ImageNet dataset to determine their influence. We expected normalization with ImageNet values to play an important role in extracting general features and speeding up the convergence rate. However, to our surprise, the overall performance did not experience significant changes but, on the contrary, the training time of the RL agent increased, probably due to the computational overhead.

Lastly, in experiment 5, we added a ResNet-based feature extractor after having added the scaling and normalizing steps in previous experiments. The increased computational time was expected and noticed immediately, however the resulting poor performance by adding ResNet-50 was not expected. This was not expected because of both the excellent empirical performance of ResNet-50 observed in other studies, and the theoretical advantages promised by the deep residual structure of ResNet-50 such as rich and hierarchical feature extraction capabilities.

5.3.3 Hyperparameter Optimization

After having performed the ablation study and gradually constructed a final setting, i.e., using the last residual block of stage 4 as a feature extractor, we utilized Weights & Biases (wandb) to conduct hyperparameter sweeps for fine-tuning our experimental setup. We conducted a Bayesian optimization over 10 configurations, each consisting of only 1,000,000 time steps due to time constraints. We retained the default hyperparameter settings from the RL-Zoo benchmark in our Bayesian optimization, opting not to modify them. While adjusting these parameters could potentially have led to better optimization, time constraints necessitated this approach. The set of optimal hyperparameters provided in table 4.2 only performed well during the optimization. Figure 4.2 also shows low mean episodic length for the benchmark

below $2 \cdot 10^6$ time steps. This is due to the exploratory nature of the PPO algorithm or the presence of high stochasticity at the beginning of the training which requires the models to run for a minimum of $2,5 \cdot 10^6$ time steps before focusing more on exploitation, considering that the benchmark performance improves after approximately time $2,5 \cdot 10^6$ steps. This suggests that more extensive hyperparameter sweeps should have been conducted. Additionally, our hyperparameter sweeps were confined to just the fourth stage. Conducting sweeps for each stage might have allowed us to tailor optimizations more precisely to each stage. Despite optimization, the optimized hyperparameters continued to underperform compared to those from the benchmark protocol, making it uncertain whether more comprehensive sweeps would have yielded significant improvements.

5.3.4 Main Experiment: Comparing ResNet-50 Stages

The main experiment involved a more comprehensive evaluation of the modified versions of the ResNet-50 model as feature extractors. This involved training the RL agent with the four different residual stages of ResNet-50 as feature extractors, using both the benchmark and optimized hyperparameters.

Through testing and analysis of the four different residual stages of the ResNet-50, we aimed to uncover insights into the effectiveness of different representation levels. We expected to see a notable difference in performance among the four residual stages. However, the only experiment that showed indication of being able to learn was the one utilizing stage 4 with the benchmark hyperparameters. The rest of the experiments had identical performance. Figure 4.6 illustrates this difference and suggests that only the features extracted by stage 4, using the benchmark hyperparameters, contain enough information for the RL agent to learn from. We know that kernels in different layers have different receptive fields. Kernels in earlier layers are activated by a small patch of the input image while kernels in deeper layers in the network include contributions from a larger area of the input image due to the cumulative effect of multiple convolutional and pooling layers. Therefore, it is reasonable to expect deeper layers to integrate information over larger portions of the input image, enabling the network to capture more complex dynamics from the environment observations. Stage four can, therefore, be expected to have better performance.

ResNet-50 is 50 layers deep and deep models are computationally intensive, both in terms of memory and more importantly processing power, which can be a drawback and cause limitations when quick iterations and real-time performance are required. We experienced a significant difference in training time when we modified the model to include up to a certain residual block or stage. Computationally, training the RL agent with the original ResNet-50 as a feature extractor took more than two days, modifying it to only include the first stage took approximately roughly 33 hours with the benchmark hyperparameters. Replicating the benchmark took approximately 4 hours of training time in comparison and performed significantly better, making it hard to argue for an effective application of ResNet-50 as a feature extractor.

5.3.5 Spikes in Mean Episodic Length

As noted earlier in sections 4.1.2 and4.4.2, both the benchmark and the four stages of ResNet-50 exhibited occasional spikes in performance. In the case of the benchmark, Figure 4.2, these

spikes could be attributed to high scores nearly completing the game, which sometimes led to infinite loops where the ball continuously bounced off walls. However, this explanation does not hold for the four stages of ResNet-50, as the observed spikes, Figure 4.7, occurred despite significantly lower mean episodic rewards, making infinite loops impossible. We attempted a visual inspection of these models to identify any recurring patterns but were unable to discern any clear reasons for these anomalies, leaving us without a definitive and reasonable explanation for their occurrence.

5.4 Future Work

ResNet-50 can be expected to perform well as a versatile feature extractor in tasks with various classes of images and objects since it is a powerful architecture for image classification. Using it in an Atari environment where the consecutive observations only differ in a few pixels from each other might be similar to asking ResNet-50 to classify them as different classes of images. It is difficult for ResNet-50 to accomplish this since the differences between observations become notable only when we compare observations with a large number of frames in difference. This is despite the fact that we use frame stacking. Therefore, further modifications in choices, methods, and the training pipeline as whole need to be considered in future studies.

After having mentioned the limitations and assumptions, and identified flaws in our approach, we provide the following aspects to be considered for future studies.

5.4.1 Frame Stacking

Frame stacking became a major concern towards the end of this study. Future work shall explore different and better strategies for frame stacking than simply concatenating the global averaged feature vectors obtained from four RGB observation frames, as we have done in this study which provided limited insights into the environment dynamics for the ResNet feature extractor. We have observed that the difference in performance between using grayscale or RGB observations is neglectable. Therefore, using stacks of three grayscale observations as inputs to the ResNet feature extractor shall not lead to significant performance degradation.

Alternatively, one could explore training a reinforcement learning agent with ResNet-50, without using stacked frames to determine if this impacts the performance. To ensure a fair and effective comparison, it would also be necessary to evaluate this approach against a benchmark that is similarly trained without frame stacking.

5.4.2 Global Average Pooling

Our current method reduces an entire feature map into a single point by taking the global average of the feature map. Spatial information embedded in feature maps gets lost with this reduction technique, which might significantly impact performance. Future work could investigate alternative pooling and dimensionality reduction techniques that preserve more spatial information, hence maintaining a more detailed feature representation.

5.4.3 Different RL Environments

Our study was limited to evaluating ResNet-50 in only one environment. To fully assess the cross-domain generalizability of ResNet-50 more comprehensively, future studies shall evaluate the robustness of ResNet-50 in multiple different RL environments with varying levels of complexity to see how well the features generalize.

5.4.4 Alternative Pre-Trained Models

Lastly, while ResNet-50 was chosen for its proven excellent performance in image classification tasks, we suggest that future studies explore other large pre-trained models. There are multiple models CNN-based models and transformer-based models like vision transformers (ViTs) which have shown promising results in various vision tasks that can be used for additional insights into cross-domain generalizability.

Chapter 6
Conclusions

Our study highlights that ResNet-50 underperforms relative to our benchmark in the tested configuration, with only the fourth stage showing some learning potential. This indicates a need for further investigation into this stage's capabilities. The computational inefficiencies observed when using ResNet-50 as a feature extractor for training reinforcement learning agents emphasize its limitations for such applications. However, given the complex and opaque nature of deep neural networks, continued research might reveal more about the generalizability of large pre-trained models. We also acknowledge that our methodology, particularly our approach to frame stacking and average pooling, may have influenced our outcomes. Therefore, these results should be interpreted as preliminary, specific to the use of ResNet-50 in Atari game environments, which vary widely in complexity. Additionally, our findings underscore the significant impact of hyperparameter selection on performance outcomes, as evidenced by the fact that benchmark hyperparameters facilitated learning in stage four, whereas the best-performing hyperparameters identified via sweep did not yield similar results.

References

[1] Artificial Intelligence Market Size, Share, Growth Report 2030 — grandviewresearch.com. https://www.grandviewresearch.com/industry-analysis/artificial-intelligence-ai-market. [Accessed 03-04-2024].

[2] Feature Extraction — deepai.org. https://deepai.org/machine-learning-glossary-and-terms/feature-extraction. [Accessed 05-03-2024].

[3] *Convolutional Neural Network (CNN) Fundamental Operational Survey*. Springer, Cham, 2020.

[4] Richard Bellman. A markovian decision process. *Indiana Univ. Math. J.*, 6:679–684, 1957.

[5] Richard E. Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, 1957.

[6] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com.

[7] A. Burkov. *The Hundred-Page Machine Learning Book*. Andriy Burkov, 2019.

[8] IBM Data and AI Team. Ai vs. machine learning vs. deep learning vs. neural networks | ibm. https://www.ibm.com/think/topics/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks. [Accessed 12-04-2024].

[9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[10] B. Christopher et al. Dota 2 with large scale deep reinforcement learning, 2019.

[11] K. Gauen et al. Comparison of visual datasets for machine learning. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 346–355. IEEE, 2017.

[12] M. Volodymyr et al. Playing atari with deep reinforcement learning, 2013.

[13] M. Volodymyr et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[14] P. Adam et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[15] R. Antonin et al. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

[16] S. David et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[17] S. David et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[18] T. Mark et al. Gymnasium, March 2023.

[19] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[21] Md Belal et al. Hossain. Transfer learning with fine-tuned deep cnn resnet50 model for classifying covid-19 from chest x-ray images - pmc. `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC8933872/`, 03 2022.

[22] G. James, D. Witten, T. Hastie, R. Tibshirani, and J. Taylor. *An Introduction to Statistical Learning: with Applications in Python*. Springer Texts in Statistics. Springer International Publishing, 2023.

[23] Ngan Le, Vidhiwar Singh Rathour, Kashu Yamazaki, Khoa Luu, and Marios Savvides. Deep reinforcement learning in computer vision: A comprehensive survey. *CoRR*, abs/2108.11510, 2021.

[24] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, Lubomir D. Bourdev, Ross B. Girshick, James Hays, Pietro Perona, Deva Ramanan, Piotr Doll'a r, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. *CoRR*, abs/1405.0312, 2014.

[25] Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[26] A.C. Müller and S. Guido. *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016.

[27] Antonin Raffin. Rl baselines3 zoo. `https://github.com/DLR-RM/rl-baselines3-zoo`, 2020.

[28] Bart Rogiers, Dirk Mallants, Okke Batelaan, Matej Gedeon, Marijke Huysmans, and Alain Dassargues. Estimation of hydraulic conductivity and its uncertainty from grain-size data using glue and artificial neural networks. *Mathematical Geosciences*, 44:739–763, 08 2012.

[29] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. *SN computer science*, 2(3):160, 2021.

[30] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[31] D. Silver, J. Schrittwieser, and K. et al. Simonyan. Mastering the game of go without human knowledge. *Nature*, 550, 10 2017.

[32] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, USA, 2 edition, 2018.

[33] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer; 2nd Edition, 2022.

Appendices

Appendix A
Network Architectures

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ActorCriticCnnPolicy                     [1]                       --
├─NatureCNN: 1-1                         [1, 512]                  --
│    └─Sequential: 2-1                   [1, 3136]                 --
│    │    └─Conv2d: 3-1                  [1, 32, 20, 20]           8,224
│    │    └─ReLU: 3-2                    [1, 32, 20, 20]           --
│    │    └─Conv2d: 3-3                  [1, 64, 9, 9]             32,832
│    │    └─ReLU: 3-4                    [1, 64, 9, 9]             --
│    │    └─Conv2d: 3-5                  [1, 64, 7, 7]             36,928
│    │    └─ReLU: 3-6                    [1, 64, 7, 7]             --
│    │    └─Flatten: 3-7                 [1, 3136]                 --
│    └─Sequential: 2-2                   [1, 512]                  --
│    │    └─Linear: 3-8                  [1, 512]                  1,606,144
│    │    └─ReLU: 3-9                    [1, 512]                  --
├─MlpExtractor: 1-2                      [1, 512]                  --
│    └─Sequential: 2-3                   [1, 512]                  --
│    └─Sequential: 2-4                   [1, 512]                  --
├─Linear: 1-3                            [1, 1]                    513
├─Linear: 1-4                            [1, 4]                    2,052
==========================================================================================
Total params: 1,686,693
Trainable params: 1,686,693
Non-trainable params: 0
Total mult-adds (M): 9.37
==========================================================================================
Input size (MB): 0.11
Forward/backward pass size (MB): 0.17
Params size (MB): 6.75
Estimated Total Size (MB): 7.03
==========================================================================================
```

Figure A.1: Benchmark Network Architecture

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ActorCriticPolicy                        [1]                       --
├─FlattenExtractor: 1-1                  [1, 1024]                 --
│    └─Flatten: 2-1                      [1, 1024]                 --
├─MlpExtractor: 1-2                      [1, 64]                   --
│    └─Sequential: 2-2                   [1, 64]                   --
│    │    └─Linear: 3-1                  [1, 64]                   65,600
│    │    └─Tanh: 3-2                    [1, 64]                   --
│    │    └─Linear: 3-3                  [1, 64]                   4,160
│    │    └─Tanh: 3-4                    [1, 64]                   --
│    └─Sequential: 2-3                   [1, 64]                   --
│    │    └─Linear: 3-5                  [1, 64]                   65,600
│    │    └─Tanh: 3-6                    [1, 64]                   --
│    │    └─Linear: 3-7                  [1, 64]                   4,160
│    │    └─Tanh: 3-8                    [1, 64]                   --
├─Linear: 1-3                            [1, 1]                    65
├─Linear: 1-4                            [1, 4]                    260
==========================================================================================
Total params: 139,845
Trainable params: 139,845
Non-trainable params: 0
Total mult-adds (M): 0.14
==========================================================================================
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.56
Estimated Total Size (MB): 0.57
==========================================================================================
```

Figure A.2: Reinforcement Learning Network Architecture Employed in Conjunction with ResNet-50 Stage 1

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ActorCriticPolicy                        [1]                       --
├─FlattenExtractor: 1-1                  [1, 2048]                 --
│    └─Flatten: 2-1                      [1, 2048]                 --
├─MlpExtractor: 1-2                      [1, 64]                   --
│    └─Sequential: 2-2                   [1, 64]                   --
│    │    └─Linear: 3-1                  [1, 64]                   131,136
│    │    └─Tanh: 3-2                    [1, 64]                   --
│    │    └─Linear: 3-3                  [1, 64]                   4,160
│    │    └─Tanh: 3-4                    [1, 64]                   --
│    └─Sequential: 2-3                   [1, 64]                   --
│    │    └─Linear: 3-5                  [1, 64]                   131,136
│    │    └─Tanh: 3-6                    [1, 64]                   --
│    │    └─Linear: 3-7                  [1, 64]                   4,160
│    │    └─Tanh: 3-8                    [1, 64]                   --
├─Linear: 1-3                            [1, 1]                    65
├─Linear: 1-4                            [1, 4]                    260
==========================================================================================
Total params: 270,917
Trainable params: 270,917
Non-trainable params: 0
Total mult-adds (M): 0.27
==========================================================================================
Input size (MB): 0.01
Forward/backward pass size (MB): 0.00
Params size (MB): 1.08
Estimated Total Size (MB): 1.09
==========================================================================================
```

Figure A.3: Reinforcement Learning Network Architecture Employed in Conjunction with ResNet-50 Stage 2

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ActorCriticPolicy                        [1]                       --
├─FlattenExtractor: 1-1                  [1, 4096]                 --
│    └─Flatten: 2-1                      [1, 4096]                 --
├─MlpExtractor: 1-2                      [1, 64]                   --
│    └─Sequential: 2-2                   [1, 64]                   --
│    │    └─Linear: 3-1                  [1, 64]                   262,208
│    │    └─Tanh: 3-2                    [1, 64]                   --
│    │    └─Linear: 3-3                  [1, 64]                   4,160
│    │    └─Tanh: 3-4                    [1, 64]                   --
│    └─Sequential: 2-3                   [1, 64]                   --
│    │    └─Linear: 3-5                  [1, 64]                   262,208
│    │    └─Tanh: 3-6                    [1, 64]                   --
│    │    └─Linear: 3-7                  [1, 64]                   4,160
│    │    └─Tanh: 3-8                    [1, 64]                   --
├─Linear: 1-3                            [1, 1]                    65
├─Linear: 1-4                            [1, 4]                    260
==========================================================================================
Total params: 533,061
Trainable params: 533,061
Non-trainable params: 0
Total mult-adds (M): 0.53
==========================================================================================
Input size (MB): 0.02
Forward/backward pass size (MB): 0.00
Params size (MB): 2.13
Estimated Total Size (MB): 2.15
==========================================================================================
```

Figure A.4: Reinforcement Learning Network Architecture Employed in Conjunction with ResNet-50 Stage 3

```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
ActorCriticPolicy                        [1]                       --
├─FlattenExtractor: 1-1                  [1, 8192]                 --
│    └─Flatten: 2-1                      [1, 8192]                 --
├─MlpExtractor: 1-2                      [1, 64]                   --
│    └─Sequential: 2-2                   [1, 64]                   --
│    │    └─Linear: 3-1                  [1, 64]                   524,352
│    │    └─Tanh: 3-2                    [1, 64]                   --
│    │    └─Linear: 3-3                  [1, 64]                   4,160
│    │    └─Tanh: 3-4                    [1, 64]                   --
│    └─Sequential: 2-3                   [1, 64]                   --
│    │    └─Linear: 3-5                  [1, 64]                   524,352
│    │    └─Tanh: 3-6                    [1, 64]                   --
│    │    └─Linear: 3-7                  [1, 64]                   4,160
│    │    └─Tanh: 3-8                    [1, 64]                   --
├─Linear: 1-3                            [1, 1]                    65
├─Linear: 1-4                            [1, 4]                    260
==========================================================================================
Total params: 1,057,349
Trainable params: 1,057,349
Non-trainable params: 0
Total mult-adds (M): 1.06
==========================================================================================
Input size (MB): 0.03
Forward/backward pass size (MB): 0.00
Params size (MB): 4.23
Estimated Total Size (MB): 4.26
==========================================================================================
```

Figure A.5: Reinforcement Learning Network Architecture Employed in Conjunction with ResNet-50 Stage 4

Appendix B
Popular Scientific Summary

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-06-14

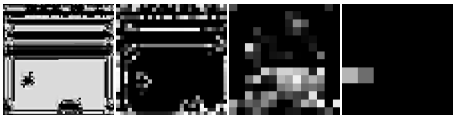
EXAMENSARBETE Cross-Domain Generalizability in Image Feature Extraction Utilizing ResNet-50 as a Feature Extractor in Reinforcement Learning
STUDENT Mamdollah Amini & Adi Creson
HANDLEDARE Alexander Dürr (LTH), Simon Kristoffersson Lind (LTH)
EXAMINATOR Volker Krüeger (LTH)

Cross-Domain Generalizability in Image Feature Extraction

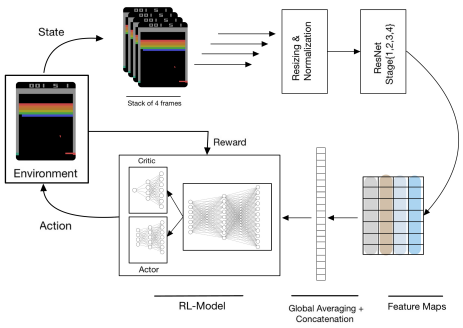
POPULÄRVETENSKAPLIG SAMMANFATTNING **Mamdollah Amini & Adi Creson**

Large pre-trained models are widely utilized for a variety of tasks, typically within environments specific to their training. How well do these models adapt and perform when applied to new domains?

To explore this further, we opted to evaluate a Pytorch implementation of ResNet-50, a deep convolutional neural network architecture originally trained on the ImageNet dataset for image classification tasks. We used it as a feature extractor to train a reinforcement learning agent in the Atari Breakout game environment. Additionally, we aimed to assess the effectiveness of various hierarchical features or representation levels by modifying the ResNet-50 model to include different numbers of residual blocks. These modifications allowed us to explore how different features outputted across the four stages of the model affected performance.



To promote certain behaviors in a reinforcement learning agent, we utilize a reward system. In our training with the Atari Breakout game, the agent receives rewards (points) for breaking bricks, which motivates it to hit as many bricks as possible before the round ends. We replicated our benchmark model from previous research. This model consistently performed well, achieving an average reward of around 400.



When we employed ResNet-50 as the feature extractor, performance significantly declined. Of the four stages in ResNet-50, only the fourth stage exhibited any learning capability, achieving a reward of around 30 and taking about 33 hours to train compared to roughly four hours for our benchmark model in our experimental setup. This performance discrepancy leads us to conclude that using ResNet-50 in this manner is suboptimal. However, further research in this area is warranted as it offers valuable insights into the opaque nature of deep learning and the generalizability of large pre-trained networks.