Computer Games Development Project Report Year IV

What advantages does using a Vision Transformer model offer over Convolutional Neural Network in playing video games?

Adrien Dudon
C00278154
26 April 2023

DECLARATION

Work submitted for assessment which does not include this declaration will not be assessed.

- I declare that all material in this submission e.g. thesis/essay/project/assignment is entirely my/our own work except where duly acknowledged.
- I have cited the sources of all quotations, paraphrases, summaries of information, tables, diagrams or other material; including software and other electronic media in which intellectual property rights may reside.
- I have provided a complete bibliography of all works and sources used in the preparation of this submission.
- I understand that failure to comply with the Institute's regulations governing plagiarism constitutes a serious offence.

Student Name: (Printed) Adrien Dudon

Student Number(s): C00278154

Signature(s): _____

Date: 26 April 2023

Please note:

a) *Individual declaration is required by each student for joint projects.
b) Where projects are submitted electronically, students are required to submit their student number.
c) The Institute regulations on plagiarism are set out in Section 10 of Examination and Assessment Regulations published each year in the Student Handbook.

I. Contents

II. Acknowledgements

The completion of this project would not have been possible without the invaluable assistance of several individuals whom I would like to acknowledge. Firstly, I would like to express my sincere gratitude to my supervisor, Dr Oisin Cawley, for providing me with expert guidance, constructive feedback, and unwavering support throughout the project. His insights and recommendations were invaluable in shaping my research and completing this project.

I would also like to extend my thanks to Dr Lei Shi, a lecturer at SETU Carlow, who generously provided me with the necessary hardware to produce the data required to obtain significant results. Their kind support and assistance are greatly appreciated.

III. Project Abstract

In recent years, images have been used to solve Deep Reinforcement Learning (DRL) problems with promising results. Convolutional Neural Network (CNN) architecture has been particularly successful in solving complex game environments, outperforming standard fully-connected neural network layers (Mnih et al. 2015; Hasselt et al. 2016). However, Vision Transformer (ViT) architecture has been shown to provide even better results in various tasks that were previously dominated by CNN (Han et al. 2023). ViT is a variant of the traditional Transformer architecture, which uses the self-attention mechanism to extract input features.

Although ViT and Transformer-based architectures are relatively new compared to CNN, ViT has not yet fully demonstrated its potential to replace CNN. This research aims to implement one of the recently introduced ViT models, the Swin Transformer (Liu et al. 2021), as the main network backbone in the Double Deep Q Network (Double DQN), a well-known DRL algorithm that typically uses CNN as its main backbone. Double DQN has been shown to perform well in Atari game environments. The aim of this research is to determine if a Vision Transformer can achieve good or even better results.

In this paper, standard techniques and technologies will be used to implement the Double DQN algorithm on Atari-based environments, using Gymnasium and PyTorch. The results will be compared with those of a recent paper that successfully used Swin Transformer as a replacement for CNN in the Double DQN algorithm (Meng et al. 2022).

The findings of this study show that although the Swin Transformer architecture performs well, the CNN architecture outperforms it, especially for a small number of training steps. CNN also requires less computing power and can work on older hardware while using a reasonable amount of memory, whereas Swin Transformer requires more data to be trained properly and a relatively recent GPU with sufficient VRAM to be trained in a reasonable amount of time. For Swin Transformer to outperforms CNN, the number of training steps should be pretty high.

IV. Project Introduction and Research Question

Artificial Neural Networks (ANNs) have emerged as popular and effective models for classification, clustering, pattern recognition, and prediction in various disciplines (Abiodun et al. 2018), including video games (Skinner and Walmsley 2019). The present research focuses on the application of ANNs in computer games.

ANNs belong to the broader discipline of Machine Learning (Skinner and Walmsley 2019), which is a component of the Artificial Intelligence (AI) field. AI plays a pivotal role in the development of computer games, contributing to the realism and intelligence of the game world. The AI domain encompasses various fields, including Machine Learning and its subdiscipline, Reinforcement Learning.

There exist many types of artificial neural networks, which can be categorized into larger groups such as feedforward, regulatory feedback, recurrent neural network, modular, physical, dynamic, memory networks, hybrids, and others. This research paper mainly focuses on image recognition neural networks, which are part of feedforward and recurrent neural network models.

Applying neural network concepts to a video game opens up numerous possibilities compared to conventional AI algorithms, owing to their ability to mimic the biological neural networks of human brains (Skinner and Walmsley 2019). However, game studios generally avoid using neural networks for their game, particularly for enemies, except in certain specific areas, as explained by Skarupke (2020), because of their uncertain behaviours. Nevertheless, the use of neural networks for video games is increasing, but determining which neural network model to use for specific needs can be challenging. It is difficult to find literature that compares different types of neural networks with a focus on video game AIs, as game studios tend to keep their work on neural networks confidential.

Standard Fully-connected Neural Networks have been utilized for various problems related to video games (Mänttäri and Larsson 2011). Although fully-connected models are helpful for solving complex problems, they are less capable of easily performing image recognition (O'Shea and Nash 2015). For the specific task of recognizing images, other neural network models should be used. In computer games, image recognition can serve numerous purposes that regular densely-connected layers cannot address.

This paper focuses on the specific problems related to computer game image recognition that can be more easily solved by certain types of neural networks. Convolutional Neural Networks (CNNs) are one such model that focuses on pattern recognition in images. CNNs are analogous to traditional fully-connected networks and belong to the same category of feedforward model. CNNs are a well-established neural network that has been used extensively in games. One of the most famous examples is the use of CNN to play the game Go, which has remained a difficult challenge in the field of AI (Clark and Storkey 2015).

Although CNNs are well-used neural networks for image recognition, Transformers is also a good candidate. Originally designed for text-based tasks, the Vision Transformers model (ViT) can now be used to perform image-recognition tasks. Transformers is a novel neural network model that was initially introduced in 2017 and then adapted for tasks in Computer

Vision with the Vision Transformers model in 2020 by the Google Research Center (Houlsby and Weissenborn 2020).

The present research aims to provide a fresh perspective on the usage of Convolutional Neural Networks and its modern alternative, Vision Transformers, in video game environments. The central research question that guides this investigation is: "**What advantages does using a Vision Transformer model offer over CNN in playing video games?"** To delve deeper into this research question, three sub-questions have been formulated, namely:

- How effective are CNNs in addressing computer vision challenges in video games?
- Which Vision Transformer architecture would be most effective in addressing computer vision problems in video games?
- What challenges are associated with training a neural network agent using Vision Transformers compared to CNNs?

To ensure a fair comparison of the neural networks, precise criteria must be used consistently throughout the experiments. This paper will also include tables and graphics displaying the data collected through the research.

V. Literature Review

Artificial Intelligence

Artificial Intelligence (AI) refers to the ability of computers to execute human and animal-like cognitive processes. This technology is primarily utilised to tackle complex problems that are beyond the capabilities of traditional computer algorithms, such as recognizing familiar faces, generating creative outputs, and conversing in natural languages (Millington 2019). The implmentation of AI has revolutionised the field of gaming, with Atari being the first game studio to integrate AI technology into their games Pong and Space Invaders. Altough the intial systems were basic, this marked a significant milestone in the evolution of AI in gaming (Skinner and Walmsley 2019).

Neural Network

An Artificial Neural Network (ANN) is a computing system that attempts to replicate the learning process of biological animal brains. ANNs have become increasingly popular in recent years and have proven to be helpful models for classification, clustering, pattern recognition, and prediction in various disciplines.

Neural Networks (NNs) are a type of computational model that draws inspiration from biological neural networks. As a part of the Artificial Intelligence field, NNs are used to model decision-making systems and provide automated knowledge extraction with high inference accuracy. They were designed to address different aspects or elements of learning such as how to learn, induce, and deduce, allowing them to draw conclusions from case observations (Yang and Yang 2014).

The basic building blocks of a Neural Network are interconnected units or nodes called artificial neurons. These neurons mirror the neurons in a biological brain and are connected by edges, which are similar to synapses in the brain. Each neuron and edge has a weight that adjusts as learning progresses. Neurons transmit signals to other neurons using these edges, and the output of each neuron is calculated by a non-linear function of the sum of its input (Artificial neural network 2023).

Neural Networks are the core components used in Deep Learning, which is a subfield of Artificial Intelligence.

Deep Learning

Deep Learning is a sophisticated branch of machine learning that allows computers to solve intricate computational problems. Deep Learning models consist of several neural networks, or processing layers, that learn data representations and extract features from the data with multiple levels of abstraction. To uncover the complex structure in data, Deep Learning employs the backpropagation algorithm to guide the machine on how to modify its internal parameters that are utilized to compute the representation in each layer based on the representation in the preceding layer (LeCun et al. 2015).

As discussed in a blog post from Seldon (Supervised vs Unsupervised Learning Explained 2022), Artificial Intelligence and Machine Learning can be classified into two categories: supervised and unsupervised learning. Supervised learning is when both the input and output

data are labeled and known. In other words, the labeled dataset is given as input and output to the machine learning algorithm during the training phase. Once the machine learning model has learned the relationship between the input and output data, it can be employed to classify new datasets and predict outcomes. On the other hand, unsupervised learning involves training models on unlabelled and raw data. In unsupervised learning, the human does not need to classify the dataset beforehand. This technique is used to identify patterns and trends in raw datasets or group similar data together. Unsupervised learning requires additional consideration to explain its output. Reinforcement Learning, a third category for Machine Learning, will be discussed in more detail in a subsequent paragraph.

To summarize, supervised machine learning discovers the relationship between input and output through labeled training data, which can then be utilized to classify new data by leveraging learned patterns or in predicting outputs. Conversely, unsupervised learning is advantageous in identifying underlying patterns and relationships within unlabelled and raw datasets.

Reinforcement Learning

Reinforcement Learning (RL) differs from Supervised and Unsupervised Learning, as it involves an AI agent learning by trial-and-error, using the "law-of-effect" tradition from psychology and other mechanisms from various fields of biology and psychology (Gallistel 1999). The fundamental idea behind RL is that agents learn to take actions that maximize a reward signal provided by the environment by exploring and exploiting the available options. The learner, or Agent, is not explicitly told which actions to take and has no prior knowledge of the environment. Instead, it must discover which actions give the most reward through trial-and-error. The challenge in RL arises from the fact that actions not only affect the immediate reward received but also subsequent actions and rewards (Sutton and Barto 2018). RL models the problem it attempts to solve as a Markov Decision Process (Puterman 1990) and commonly uses the Bellman equation in its algorithms to determine whether the goal has been achieved. The Bellman equation approximates a value function, estimating how good it is for the agent to be in a given state or perform a given action in a given state (O'Donoghue et al. n.d.).

RL has been the subject of extensive research due to its generality and has been applied in various fields, such as Game Theory, Multi-agent Reinforcement Learning (Nowé et al. 2012, chap.14), Control Theory (Gullapalli 1992), Economics and Finance (Charpentier et al. 2021), among others. This paper focuses on the use of RL in video games.

A range of methods and algorithms have been developed to solve RL problems, which can be broadly classified into two categories: model-based and model-free (Sutton and Barto 2018). Model-based algorithms learn a model of the environment dynamics and use it to plan actions, while model-free algorithms directly learn the optimal policy by interacting with the environment. Popular algorithms in this category include Q-learning, SARSA, and policy gradient methods (Sutton and Barto 2018).

Deep Reinforcement Learning (DRL) is a recent development in RL that combines Deep Learning with RL, enabling agents to learn from high-dimensional sensory inputs, such as images (Mnih et al. 2015), and generalize well on any types of data.

RL has gained popularity in recent years, with numerous applications in real-world scenarios. However, it faces challenges such as sample inefficiency, instability, and exploration-exploitation trade-off. Ongoing research in RL aims to address these challenges and develop more robust and efficient RL algorithms.

Q-Learning

Q-Learning is a reinforcement learning algorithm that facilitates an AI agent's ability to determine the optimal action in a Markov Decision Process environment (Watkins and Dayan 1992). It is a dynamic programming approach that improves its assessment of the best course of action in a given state through successive iterations. The Q function is central to the Q-Learning algorithm, as it estimates the expected cumulative reward achievable by an agent through a specific policy in a given state, taking a particular action. In the traditional Q-Learning algorithm, this Q function is stored as a table referred to as the Q-table.

$$Q_*(s,a) = max_\pi E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \vee s_t = s, a_t = a, \pi\right]$$

Formula 1: The Q-function in traditional Q-learning

Q-learning has several advantages, such as its ability to learn from experience without requiring a model of the environment. This makes Q-learning applicable to a wide range of problems where the dynamics of the environment are unknown or too complex to model accurately. Furthermore, Q-learning is flexible and can adapt to different problem settings, including online and offline learning, single-agent and multi-agent settings, and continuous and discrete action spaces (Brafman and Tennenholtz 2002; Sutton and Barto 2018). However, Q-learning also has limitations, such as the challenge of balancing exploration versus exploitation. Finding the right balance between exploration and exploitation is crucial for Q-learning, as exploration is necessary for the agent to learn the optimal policy, while exploitation is necessary to maximize rewards.

Deep Q-learning Network (DQN) is a variant of Q-learning that uses a deep neural network to approximate the Q-function instead of using a table. The combination of Q-learning and deep neural networks has led to significant breakthroughs in various domains, such as game playing, robotics, and autonomous driving (Bojarski et al. 2016; Mnih et al. 2015; Silver et al. 2016). The DQN algorithm addresses the instability issues of the standard algorithm by using an experience replay to store the previous experience of the agent's actions.

Double Deep Q-learning Network (DDQN) is an improvement on top of DQN that aims to overcome the overestimation issue of Q-learning. Q-learning suffers from overestimation because the max operator used in standard Q-learning and DQN uses the same value to select and evaluate an action. This instability can lead to suboptimal performance (Hasselt et al. 2016). DDQN addresses this issue by using two Q-functions or two Q-networks in the case of deep Q-learning. The two functions are learned by assigning experiences randomly to update one of the two Q-functions. One Q-function determines the greedy policy, and the other Q-function determines its value. In Deep Q Network, the standard way of achieving this is by keeping a copy of the policy network called the target network. The weights of the target network are updated regularly with those of the policy network.

In this project, Double DQN was specifically used due to its relatively simple algorithm and ease of use.

Deep Learning and video games

Zahedi (1991) posits that Artificial Intelligence and neural networks share a common goal of simulating convincing human intelligence. However, Zahedi's claim that AI and neural networks are distinct is erroneous, neural networks are a subpart AI. Rule-based AI, including expert systems, assumes the brain is a black box and attempts to replicate human reasoning as a series of explicit algorithms. In contrast, neural networks treat the brain as a white box and model its internal structure and function by representing knowledge implicitly through nodes and edges. Neural networks use inductive reasoning to process knowledge, and learning occurs within the system, similar to the way an animal brain learns. Both rule-based AI and neural networks can be used to address AI problems in video games, but they employ different approaches. Rule-based AI relies on external knowledge stored explicitly in the algorithm, whereas neural networks attempt to mimic the human brain to learn autonomously, with knowledge hidden in its *hidden layers*. Because neural networks have a greater capacity to replicate the workings of the human brain (Skinner and Walmsley 2019) they are particularly suitable for video games where rule-based AI algorithms would be inadequate. The choice of approach depends on the specific application and desired fidelity to reality and performance.

Neural networks are increasingly utilized in video games for controlling game agents, which may include non-player characters or represent the game environment itself (Qualls and David 2009). The scope of neural network application in video games is broad, as they can control a single agent or multiple agents, and represent various facets of the game. One commonly applied function of neural networks in video games is path navigation, where a neural-based agent adapts to the ever-changing game environment. Neural networks can be used to create a dynamic gameplay experience, where the game becomes progressively more challenging over time. Additionally, neural networks can be used to control the game ecology, to populate and mutate the environment based on player actions, for instance, by making animals more friendly or hostile. Another application is for creating realistic game animations. As computer games become more powerful, animations need to be increasingly lifelike. A neural network can teach an animal character how to walk on various surfaces and environments, allowing the character to adapt its movements to different terrains, such as mountains or flat plains.

A study by the laboratory of Professor of Psychology John O'Doherty at Caltech found that artificial neural networks exhibit similar behavior to the human brain while playing video games (Dajose 2021). This discovery is significant because it could aid our understanding of how the brain solves difficult problems, and in turn, help guide the development of more intelligent and human-like AI algorithms for video games.

To provide a concrete example of neural networks in video games, we can look at “Colin McRae's Rally 2”, a racing game developed by Codemasters that simulates rally races. To control the opponent cars in the game, Codemasters developers used neural networks trained using various techniques. Jeff Hannan, a developer at Codemasters, explained in an interview the techniques employed to develop the neural network used in the game (Hannan n.d.). To train the neural network, developers played the game and recorded their laps around the racetracks, capturing data on drivers' reactions to other cars and their average driving line.

This data was then used in supervised learning to train the neural network, which was subsequently integrated into the final game.

Types of neural network

The present research project aims to utilize images for training an agent to play games effectively. In this context, two prominent models of neural networks have been identified, namely the Convolutional Neural Network and the Vision Transformer (O'Shea and Nash 2015; Dosovitskiy et al. 2021).

Convolutional Neural Network

Convolutional Neural Network (CNN) is a type of neural network that is primarily used in image recognition (O'Shea and Nash 2015, p.2). Like traditional NNs, they are comprised of neurons that self-optimise through learning. Neurons receive input and perform operations (such as a scalar product followed by a non-linear function) – the basis of every ANNs. As stated by O'Shea and Nash, the only notable difference between CNNs and fully-connected NNs is that they are primarily used in the field of pattern recognition within images. In other words, to be able to know what the different "objects" are composing an image. CNNs can take image-specific features as input, which is why CNN is more suitable for image-focused tasks because fewer parameters are needed to set up the model.

Using traditional neural networks for image recognition poses a significant challenge due to the computational complexity required to compute image data, particularly as the number of pixels composing an image increases. For example, a 28×28 black-and-white image can be easily stored in a single neuron, containing 784 weights. However, for video games, rendering more pixels and accounting for colors increases the information required for recognition, making it impractical for fully-connected layers to perform this task (O'Shea and Nash 2015, p.3).

Transformer

The Transformer is a deep-learning model that employs the self-attention mechanism and has gained popularity for natural language processing tasks, including language translation and sentiment analysis.

Attention is a cognitive process that enables individuals to focus actively on specific information, while ignoring other details (James 1890). In machine learning, attention is a technique that seeks to emulate cognitive attention in humans. The Transformer model architecture relies solely on the attention mechanism to capture global dependencies between input and output (Vaswani et al. 2017).

The Transformer architecture adopts the encoder-decoder structure of neural sequence transduction models (Sutskever et al. 2014; Cho et al. 2014; Bahdanau et al. 2016), comprising self-attention and fully connected layers.

Vision Transformer

The Vision Transformer (ViT) is a neural network model inspired by the success of the Transformer architecture in natural language processing. Instead of processing a sequence of text, the ViT directly processes images by splitting them into fixed-size patches and

providing a sequence of linear embeddings of these patches as input to a Transformer. Similar to the tokenization of words in a standard NLP Transformer, patches are treated as tokens (Dosovitskiy et al. 2021).

While convolutional neural networks have traditionally been the most commonly used architecture for image recognition tasks, the trend is shifting towards the use of Vision Transformers as a replacement for standard CNNs. ViTs have shown superior performance in image classification tasks, given that the training dataset is sufficiently large (Zhao et al. 2021).

However, the use of ViT for Deep Reinforcement Learning remains rare due to the significant amount of data required for training. A recent study compared a pre-trained ViT, trained using self-supervised learning, with RAD, a leading CNN-based RL method. The study found that RAD still outperformed ViT (Tao et al. 2022). Training ViTs remains a difficult and costly task due to its quadratic complexity relative to the input image size.

Recent Transformer architectures have been adapted to directly solve Reinforcement Learning problems, such as Decision Transformer or Trajectory Transformer. These adaptations have shown promising results, but they require significant modifications to existing RL algorithms and may be more complex to use and understand. Additionally, these adaptations do not work as effectively with online RL (Chen et al. 2021; Janner et al. 2021).

Swin Transformer

The Swin Transformer is an advanced model designed to enhance the standard ViT architecture by addressing limitations of existing Transformer-based models used for vision applications. While standard Vision Transformer pose challenges and require changes in the underlying RL algorithm to replace CNN, the Swin Transformer can be a valuable solution. This model aims to build hierarchical feature maps by merging image patches in deeper layers, which ensures linear computation complexity to input image size. In contrast, previous ViT suffers from quadratic computation complexity due to computing self-attention globally, resulting in a single low-resolution feature map (Liu et al. 2021). Consequently, the Swin Transformer is emerging as a general-purpose backbone for computer vision, comparable to CNN.

Thus, the Swin Transformer is an excellent candidate to replace the CNN algorithm in Deep Reinforcement Learning algorithms, given that CNN is already considered a good general backbone for almost every task involving computer vision. Recently, the Swin Transformer was tested on solving Atari game environments with the Double DQN algorithm, where the Q network was replaced from the CNN to the Swin Transformer architecture (Meng et al. 2022). The authors compared their results with those of the 49 Atari games, and their findings were quite intriguing. As the paper is recent, no one has attempted to replicate this experiment yet (as far as I am aware). The results of this study show that the Swin Transformer can achieve significantly higher evaluation scores across most tested Atari games. I plan to conduct my own experiment to compare their results with mines.

VI. Methodology

Technologies

In order to ensure a fair and unbiased comparison, state-of-the-art software libraries and frameworks were used for this project. All tools utilized were open source and widely recognized in the research community.

Python was chosen as the programming language due to its popularity and the extensive range of machine learning libraries and frameworks available. Two of the most widely used deep learning frameworks, TensorFlow and PyTorch, were initially considered. TensorFlow was used at the beginning of the project, but issues encountered led to a switch to PyTorch towards the end of the project. Both frameworks provide a vast ecosystem and utilities for deep learning, with a large community of researchers utilizing them in their work (Martín Abadi et al. 2015; Paszke et al. 2019).

For the environment to train the agent, Gymnasium, a Python library was used. Formerly known as Gym and developed by OpenAI, Gymnasium is now managed by the Farama foundation (Farama Foundation 2022). Gymnasium offers a wide range of different environments representing general reinforcement learning problems. It features an API specifically designed for solving reinforcement learning problems using any kind of methods.

To ensure the reliability of the implementation of the Deep Q-Learning algorithm, the Stable Baselines3 library was utilized. This library features a set of reliable implementations of reinforcement learning algorithms in PyTorch (Raffin et al. 2021). Although the data generated by this library was not used to compare the neural networks, it was used as a reference point to validate the correctness of my algorithm implementation.

Method

As detailed in the literature review, the present study sought to replicate the approach of Meng et al. (2022), which entailed employing the Double DQN algorithm in conjunction with a Convolutional Neural Network, and subsequently replacing the CNN with a Swin Transformer. For each experiment, identical algorithmic and hyperparameter settings were employed.

Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\hat{\theta} = \theta$
For t = 1 to T **do**
  Reset the environment
  while game not done **do**
    t += 1
    if random(0,1) < ϵ greedy **do**
      Select a random action a
    else
      Select an action through policy network $a = argmax(Q(s, a; \theta))$
    end if

    Execute action a
    Store $(s, a, s', r, done)$ into replay buffer D
    Sample random minibatch of $(s, a, s', r, done)$ from D

    if t % train_frequency = 0:
      $target = r + (s', a; \theta) * (1 - done)$
      Compute loss $L(Q(s, a; \theta), target)$
      Update $Q(s, a; \theta)$ by loss L with gradient descent
    end if
  end while

  if t % target_update_interval = 0:
    Update $\hat{Q}$ with $\hat{\theta} = \theta$
  end if

end for

Algorithm 1: Double Deep Q Network (Pseudocode)

Table 1: *Algorithm parameters definition*

Symbol	Type	Definition
Q(s, a; θ) and $\hat{Q}$(s, a; $\hat{\theta}$)	Q Network	Policy and target networks with their weights
D and N	List, int	Store experience in a list of size N
train_frequency	int	The frequency to which perform a gradient descent step to update the policy Q-Network Q(s, a; θ).
target_update_interval	int	The interval to synchronise the target network with the policy network
t	int	The current timestep. *1 timestep = 4 frames*
s and s'	(84, 84, 4)	Current and next environement state stored as 4 stacked and grayscaled image frames.
a, r, done	int, int, bool	Action, reward and terminal state

Table 2: Parameters used in Double DQN

Input	4×84×84
Optimizer	Adam
Adam learning rate	0.0001
Loss function	Smooth L1
Max timesteps	10,000,000
Target update interval	1,000
Learning starts	100,000
Train frequency	4
Replay buffer size	10,000
Batch size	32
Discount rate γ	0.99
Exploration fraction	0.1
Final exploration rate ε	0.01

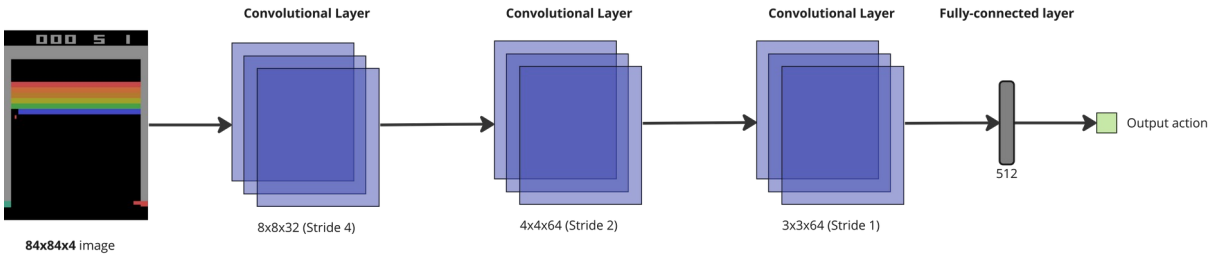


Figure 1: Structure of the CNN network

The input to both the CNN and Swin Transformer networks is a stack of four grayscale frames reduced to a size of 84×84. The CNN architecture is based on the one used by Mnih et al. (2015) in their Nature paper on DQN, which includes a 32-filter convolutional layer with 8×8 filters and stride 4, a 64-filter convolutional layer with 4×4 filters and stride 2, a 64-filter convolutional layer with 3×3 filters and stride 1, a fully-connected hidden layer with 512 units, and a fully-connected linear output layer with a single output for each valid action in the environment.

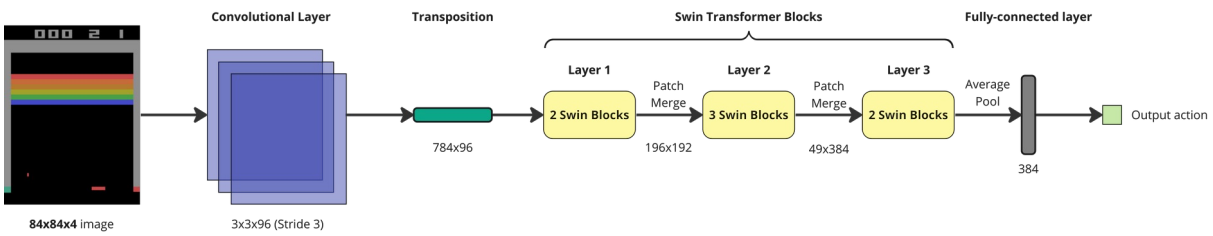


Figure 2: Structure of the Swin DQN

For the Swin Transformer architecture, the complexity of the model makes finding good parameters challenging. The architecture used in this project was adapted from Meng et al. (2022) and includes three layers of Swin Blocks, each containing 2, 3, and 2 blocks and 3, 3, and 6 attention heads, respectively. The patch size is set to 3×3, which yields 28×28 patches given the input size of 84×84 for the four channels. The embedding dimension for each patch is 96, resulting in a token size of 784×96 after patch embedding. The local window size is 7×7 and the windows are shifted by 3 patches for the first and third blocks. The MLP ratio is 4, indicating that the linear layers within Swin Blocks have 4 times the embedding dimension hidden units, i.e., 384. The drop path rate is set to 0.1, indicating a 10% chance that the input is kept as it is in skip connections (Meng et al. 2022).

Table 3 and 4 summarize the parameters for both networks.

Table 3: Parameters for the CNN network

Layers	3
Filters each layer	32, 64, 64
Strides each layer	4, 2, 1
Kernel size each layer	8, 4, 3
MLP units	64×7×7 = 512

Table 4: Parameters for the Swin network

Layers	3
Blocks each layer	2, 3, 2
Heads each layer	3, 3, 6
Patch size	3×3
Window size	7×7
Embedding dimension	96
MLP ratio	4
Drop path rate	0.1

VII. Evaluation and Discussion

Pong

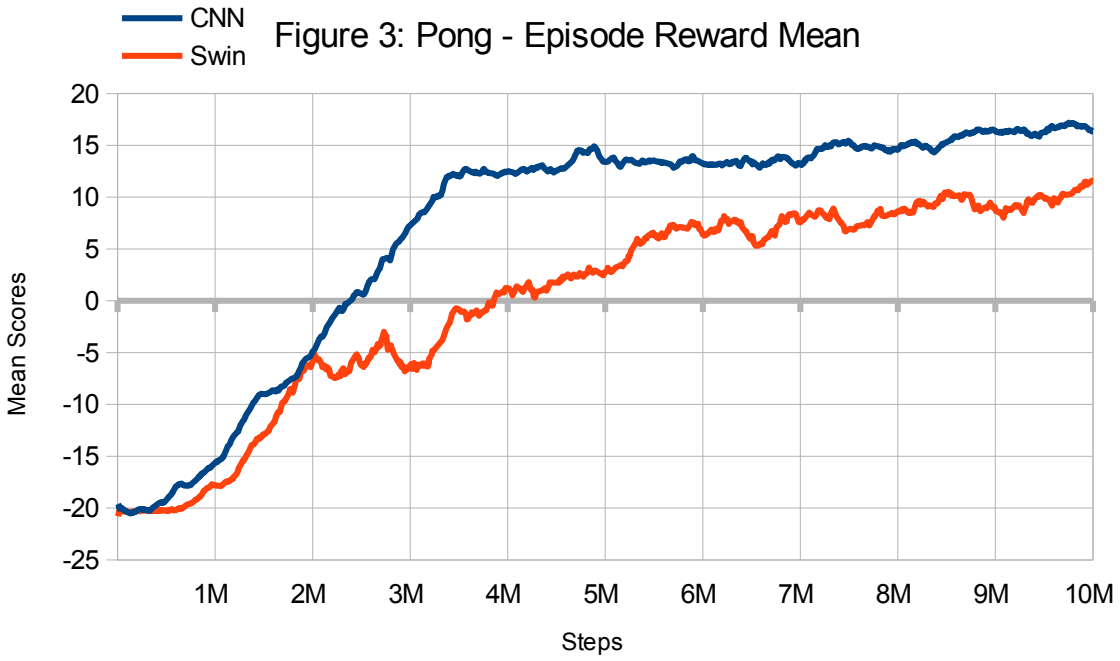


Figure 3: Pong - Episode Reward Mean

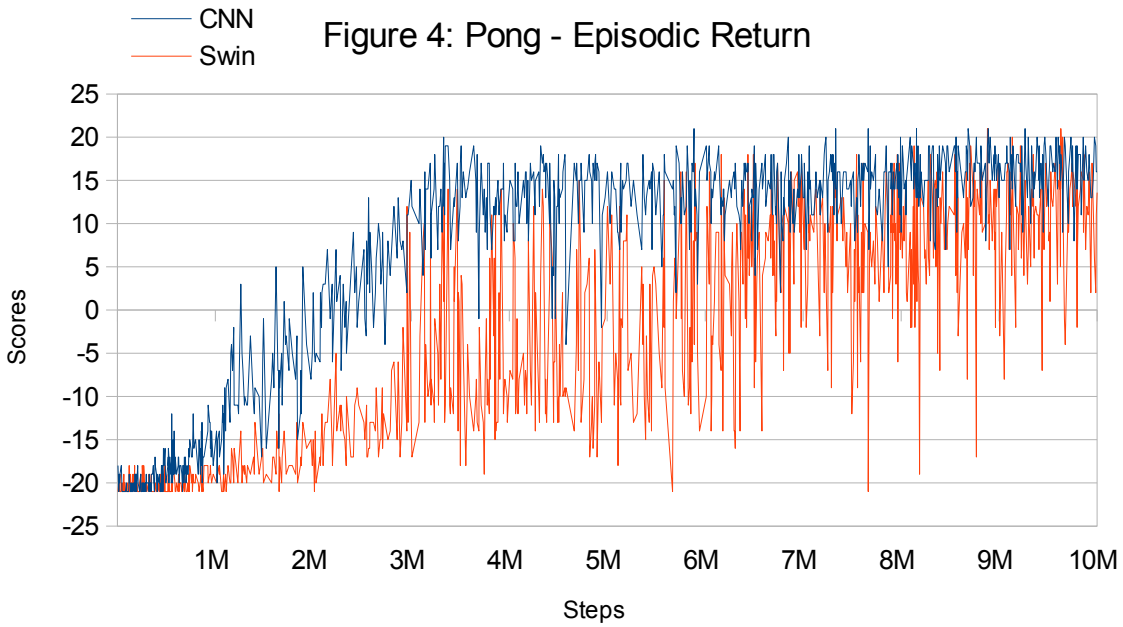
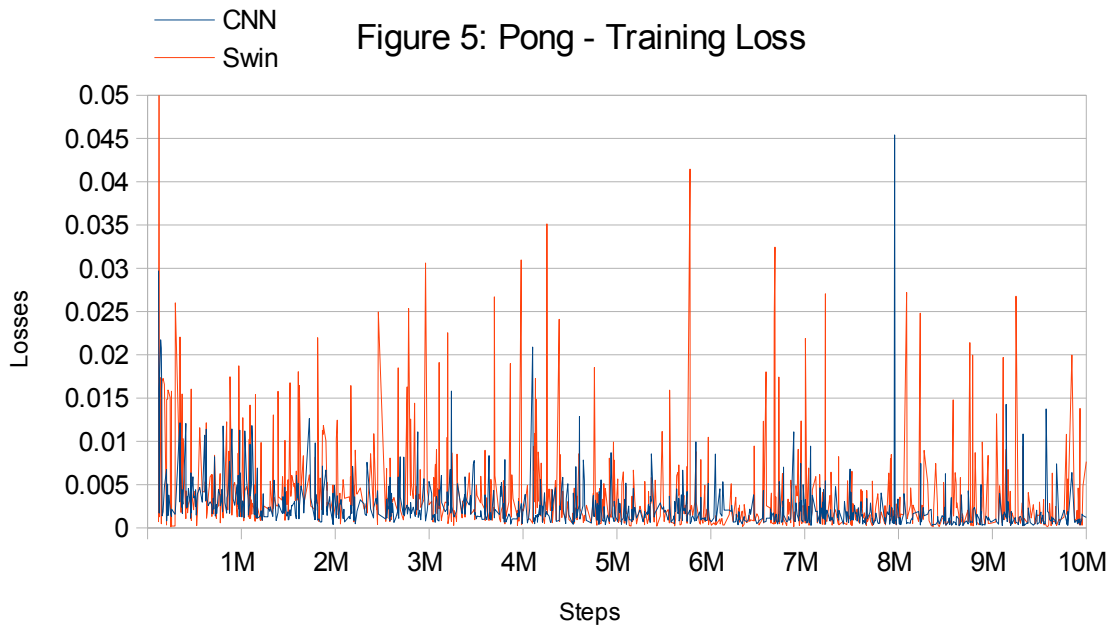


Figure 4: Pong - Episodic Return



As anticipated, the CNN architecture produced favorable results, with the agent achieving good performance. The CNN agent employed in this study required a relatively short training time of 7 hours and 36 minutes on a GTX 970 GPU, thereby demonstrating its efficacy. The computational resource constraints did not significantly impact the agent's ability to learn to play Pong. The agent began to converge around 500,000 steps, with a stable increase until it reached 3.5 million steps, where it stabilized. The training curve showed a linear increase with no significant dropouts, even after hours of training. However, the agent did not achieve a mean reward of +/- 20 (21 being the max score). This outcome might be attributed to the selection of the environment seed, which was 42 in this case.

In comparison to Swin, the CNN agent showed a lower amount of loss spikes as training progressed, indicating a quicker learning process with fewer mistakes. The agent's ability to learn from the environment and improve its performance is a key characteristic of the CNN architecture, and it has proven to be a powerful tool also in this case.

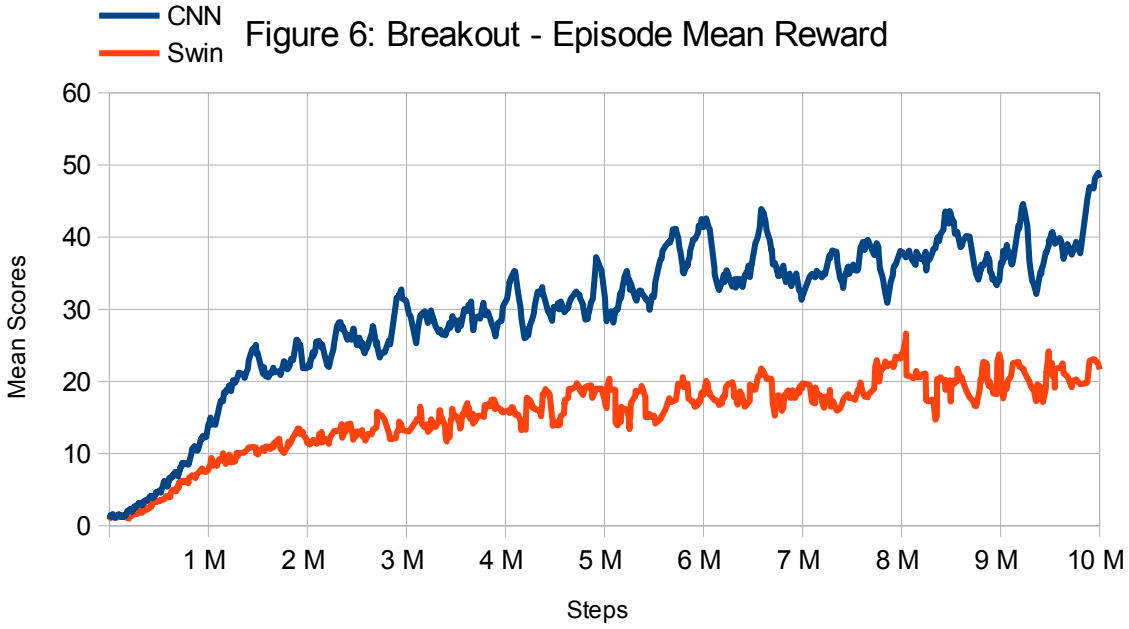
The Swin Transformer-based DQN agent required a substantially longer training period of 3 days and 7 hours on an RTX 3080 GPU. While the CNN model stabilized at 3.5 million steps with a mean reward of +/- 13, the Swin DQN model was still scoring in the negative at this point. Although the Swin DQN model only began to score positively at around 4 million steps, it showed a constant increase in performance compared to the CNN model, which stabilized for most of its training duration. However, Swin Transformer was not able to achieve the same mean reward as the CNN model, achieving only around ~12. Another interesting result, is that the CNN agent stayed stable, with no major dropout, while the Swin Agent results are very extremes, giving sometimes a few better result than DQN, and sometimes a completely worse result, with an episodic return of almost -21, even at the end of training. However, we can still see an evolution overhaul, with less big spikes at the end.

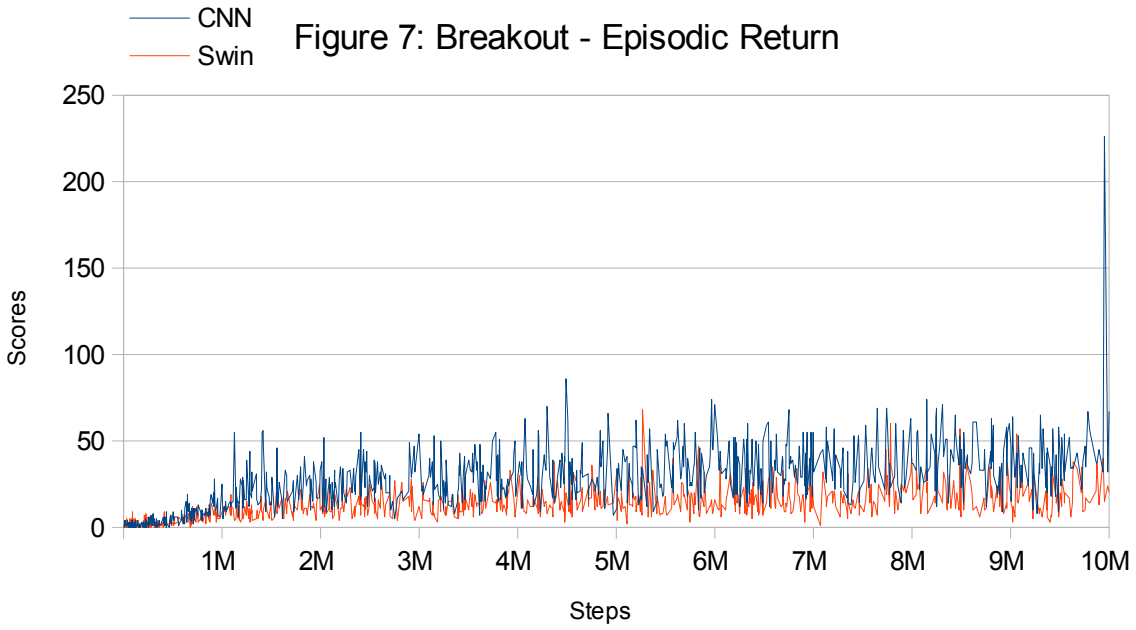
The performance curve for the Swin Transformer-based DQN model was less linear, exhibiting consistent drops to lower scores for a certain period before increasing again. The loss evolution during training was somewhat erratic, with significant spikes even near the end

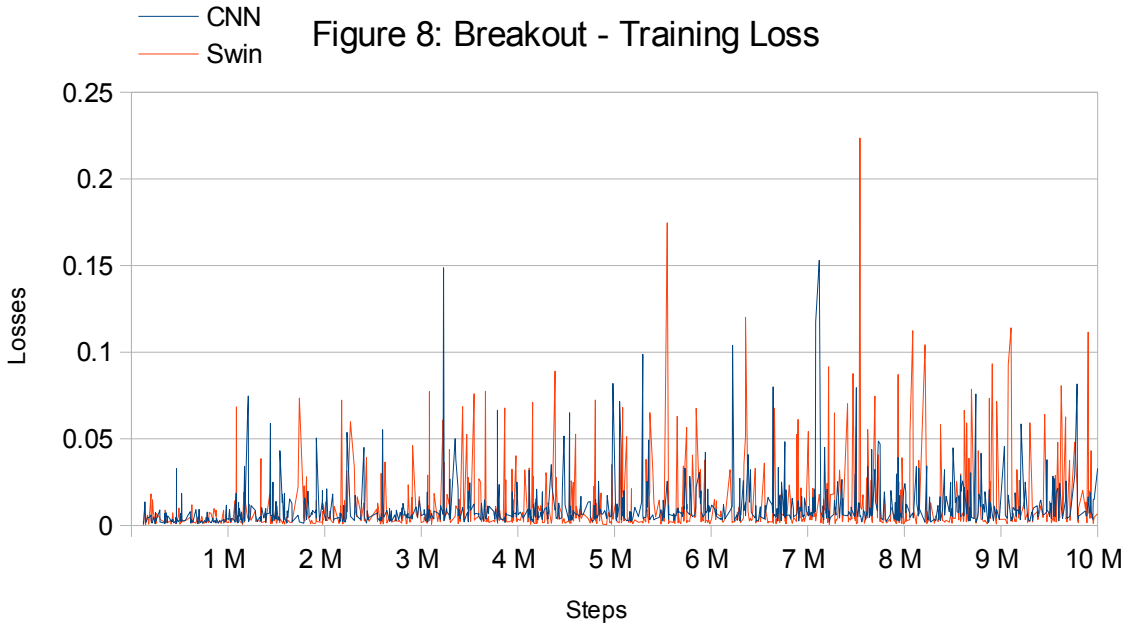
of training. This behavior was not unexpected, given that the Transformer architecture requires more data to be trained effectively.

An interesting comparison can be made between CNN and Swin, by observing the agents playing directly. The agent trained with CNN took actions in an erratic manner, whereas the Swin Transformer appeared to be smoother, despite its lower mean reward. This behavior is critical in video games since the smoother it is, the more appealing it will be to a human player. A compelling video comparing the two networks on Pong can be found at https://youtu.be/2WXqowueAFE. The video gives the impression that the Swin DQN agent thinks more before taking an action than the CNN agent.

Breakout







The CNN agent was trained for 1 day and 7 hours on an NVIDIA GTX 970 GPU, while the Swin agent required a significantly longer training period of 3 days and 6 hours on an NVIDIA RTX 3080 GPU. Interestingly, the Swin Breakout agent required less time to train than the Swin Pong agent, while the CNN Breakout agent required more time than the CNN Pong agent, despite being trained on the same hardware and under the same conditions. The reasons for these results are not immediately clear and may be due to random factors rather than the neural networks themselves.

In terms of performance, the CNN agent outperformed the Swin agent in Breakout, achieving a mean reward of almost 50 and a better episodic return overhaul, while the Swin DQN agent

occasionally performed better for a few episodes. Overall, the Swin agent's performance was worse than Pong, with a mean reward that was more than double that of the CNN agent. The convergence point for both agents started early in training, before 1 million steps. These results suggest that Breakout requires more than 10 million steps for proper training, although the CNN agent was able to achieve better results in less time.

Compared to Meng et al. (2022), my results on Breakout show lower scores for the first 10 million steps. This difference may be due to random factors or the choice of seed. In their paper, Swin DQN and CNN DQN achieved relatively similar results, which became sparser closer to 2 million steps. This suggests that the Swin agent requires more time and data to achieve optimal performance compared to the CNN agent.

VIII. Project Milestones

As a research-focused project, hard deadlines were not set for completing specific features, though general goals were established throughout the sprints and adjusted as needed based on progress and difficulties encountered. At the outset, the primary goal was to gain a comprehensive understanding of machine learning, given my limited knowledge in the field at the time. A significant amount of time was allocated to preparing a literature review by reviewing relevant papers to the field, which was necessary before delving into the practical aspects of the project.

After the initial research phase, a schedule was devised to set general goals for each sprint. However, it turned out to be a completely unrealistic schedule as the amount of work required to reach certain goals was underestimated. A considerable portion of the time was spent experimenting, with many unsuccessful attempts at making some experiments work.

The second sprint began at the end of November, during which time I researched the necessary technology stack required to develop the project. Simple examples were implemented to gain proficiency in the use of these technologies. During this sprint, I faced challenges in properly installing all the required tools, and I had to experiment with various methods before finally arriving at a working environment.

After the second sprint, I faced a significant roadblock when trying to understand and implement the algorithms necessary to complete the project. However, with perseverance, I successfully implemented my first experimentation of the DQN algorithm using TensorFlow.

During the last sprint, I was able to overcome most of the challenges encountered in the previous sprints and was finally able to perform the experiments I had planned. Looking back, I do not believe that focusing more on scheduling would have helped as the difficulties encountered would have made following the schedule challenging. However, I would have changed the approach taken to better learn the subject.

IX. Major Technical Achievements

At the beginning of this research project, I initially attempted to work on a conventional Windows OS machine. However, I quickly discovered that this was not the optimal choice due to the limited support for many of the libraries required for the project. Consequently, I exclusively employed a Linux operating system, Altough I already used Linux in the past, I had not used it as a primary work OS. This required me to adapt to a new environment and to employ novel, open-source, and less data-intrusive applications developed by the open-source community. Ultimately, I found a range of impressive alternatives to standard software used by large companies, and I will likely continue to use them and Linux as my primary work OS.

The field of AI and machine learning research is highly collaborative, with many experts sharing their work and creating useful libraries to facilitate further research without having to reinvent the wheel for every project. This has resulted in the creation of numerous open-source libraries such as TensorFlow, PyTorch, Gymnasium, and Stable Baselines3. Consequently, my project predominantly utilized open-source software, which I believe is the optimal approach for most AI development. This experience demonstrated a novel perspective on a more open-minded software development process, and I believe that further emphasis on open-source software development will greatly benefit research in this area.

For me, completing an open-source project and working with open-source pieces of software is a significant accomplishment, as it allows me to openly share my work and the software utilized in it.

The development of machine learning algorithms is more complex than developing conventional applications. In standard application development, one cancode, compile, test, and fix issues as needed. In contrast, with my project, assessing whether something is functioning appropriately could require waiting for several hours or even days. Therefore, implementing these algorithms is a significant achievement due to the time and difficulty required to achieve satisfactory results. I take pride in having been able to implement these machine learning algorithms in my project.

X. Project Review

Although there was potential for the project to explore more extensive experiments, such as testing other Vision Transformer architectures or implementing the Swin Transformer v2, the final outcomes were still promising. The results obtained were particularly intriguing in comparison to the research paper utilized as a reference to guide the implementation of the algorithms (Meng et al. 2022).

The greatest challenge was in developing an effective implementation of the DQN algorithm, especially given the limitations of my conventional gaming laptop with only a mid-range GPU. Initially, I had chosen to use TensorFlow as the machine learning framework, following the advice of my supervisor. However, a month before the final deadline, I switched to PyTorch. I encountered numerous issues with TensorFlow, primarily the library's high default VRAM usage even for simple model architectures. Additionally, writing and optimizing a TensorFlow-based application required cumbersome boilerplate code, which caused confusion. Moreover, finding pre-existing library and model architecture implementations in TensorFlow proved challenging, as many new architectures are implemented in PyTorch. For example, the official implementation of the Swin Transformer is in PyTorch, and popular Reinforcement Learning libraries like Stable Baselines3 are also in PyTorch. When attempting to use the Swin Transformer, I had to find a reliable, non-official implementation of it in TensorFlow, which I was unable to run on my laptop GPU (a GTX 1660 Ti).

Ultimately, I found it more straightforward to switch my scripts from TensorFlow to PyTorch. Prototyping algorithms became much more accessible with PyTorch, and the default VRAM usage was reasonable. I did not encounter any issues with resource depletion, even on my laptop. Additionally, I found that algorithms written in PyTorch are more readable than those in TensorFlow. While TensorFlow can be quicker than PyTorch, it requires more effort and time to tailor it to the project's needs.

In implementing the Double Deep Q Network algorithm, the option of utilizing a pre-existing implementation from a Reinforcement Learning framework was available. However, due to its inclusion in the scope of the project, the decision was made to develop my own implementation. However, as a non-expert in the field, it was important to ensure the validity of the implemented algorithm and avoid comparison inaccuracies caused by errors in the code. Consequently, the Stable Baselines3 DQN implementation was used to confirm the accuracy of my DDQN algorithm. Through this process, the confidence in the correctness of the implemented algorithm was established, ensuring the validity of subsequent comparisons.

Python was chosen as the programming language for its comprehensive libraries and user-friendly syntax in comparison to languages like C# or C++. It is noteworthy that almost all the algorithms utilized in this study are written in Python. The technologies employed in this project, except for TensorFlow, were suitable for the task and allowed the focus to remain on the comparison aspect. The choice of utilizing these technologies prevented the need to create new libraries like Gymnasium, which would have been challenging to implement and time-consuming.

For individuals pursuing a similar project, it is highly recommended that they carefully read and comprehend the research paper which first introduced the algorithm utilized. Additionally, it is advisable to attend high-quality, current practical courses on reinforcement learning to ensure adequate preparation for the project.

XI. Conclusions

The primary objective of this project was to compare the performance of a Vision Transformer variant with its CNN counterpart using the Deep Q Network (DQN) algorithm, a widely recognized Reinforcement Learning (RL) algorithm. However, during the research phase, a paper titled “Deep Reinforcement Learning with Swin Transformer” (Meng et al. 2022) was encountered, which had already replaced the CNN with the Swin Transformer model. Despite this, the idea of replicating their work and comparing my results with theirs remained intriguing, especially given my limited knowledge of hyperparameter tuning if I were to try a completely different architecture.

The conclusion of the Meng et al. paper showed that the Swin DQN outperformed its CNN counterpart. However, my results indicated that the CNN outperformed the Swin DQN in the two tested games, Pong and Breakout. Despite this, the results were still significant and interesting compared to the paper, indicating that the Swin Transformer's computational complexity and time requirements for training an agent are higher than those of the CNN. The higher results obtained by Meng et al. (2022) demonstrate that the Swin Transformer requires a significantly higher number of training steps.

Training an agent with Swin Transformer is computationally intensive and requires significant GPU memory, making it challenging for an average computer and not feasible for consumer video games due to the low number of individuals who own such hardware. Therefore, the CNN remains a viable option for high-performance algorithms, considering the hardware of the end-user machine.

Despite this, the Swin Transformer offers certain advantages over the CNN that are not related to raw performance. A fascinating comparison can be made between the CNN and Swin by observing the agents playing directly. The agent trained with CNN took actions in an erratic manner, whereas the Swin Transformer appeared to be smoother, despite its lower mean reward. This behavior is critical in video games since the smoother it is, the more appealing it will be to a human player.

XII. Future Work

This project has explored the comparison of a Vision Transformer variant to its CNN counterpart using the Deep Q Network algorithm. However, there are several avenues for future research that can build upon this work.

Firstly, while this project focused on the Swin Transformer model, there are several variations of the original ViT model that could yield interesting results. Thus, it would be valuable to compare and evaluate the performance of different Vision Transformer models.

Moreover, the recent introduction of the Swin Transformer v2 architecture presents a natural progression for this project. It would be worthwhile to explore this improved version and investigate its benefits compared to the original Swin Transformer model.

Additionally, this study could extend its investigation to other offline or online Deep Reinforcement Learning algorithms beyond DQN. Different algorithms such as A2C, DDPG, HER, PPO, SAC, TD3, have unique strengths and weaknesses and could benefit from a change in their network backbone (Mnih et al. 2016; Lillicrap et al. 2019; Andrychowicz et al. 2018; Schulman et al. 2017; Haarnoja et al. 2018; Fujimoto et al. 2018).

The findings of this project suggest that the CNN remains a viable option for high-performance algorithms in video games. However, exploring the benefits of newer and different architectures can contribute to the ongoing development of efficient and effective Deep Reinforcement Learning algorithms.

XIII. References

Abiodun, O.I., Jantan, A., Omolara, A.E., Dada, K.V., Mohamed, N.A. and Arshad, H. (2018). State-of-the-art in artificial neural network applications: A survey. *Heliyon*, 4(11), p.e00938.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., McGrew, B., Tobin, J., Abbeel, P. and Zaremba, W. (2018). Hindsight Experience Replay. , 23 February 2018. Available from: http://arxiv.org/abs/1707.01495 [accessed 26 April 2023].

Artificial neural network. (2023). *Wikipedia* [online], 18 April 2023. Available from: https://en.wikipedia.org/w/index.php?title=Artificial_neural_network&oldid=1150503192 [accessed 20 November 2022].

Bahdanau, D., Cho, K. and Bengio, Y. (2016). Neural Machine Translation by Jointly Learning to Align and Translate. , 19 May 2016. Available from: http://arxiv.org/abs/1409.0473 [accessed 22 April 2023].

Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L.D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J. and Zieba, K. (2016). End to End Learning for Self-Driving Cars. , 25 April 2016. Available from: http://arxiv.org/abs/1604.07316 [accessed 23 April 2023].

Brafman, R.I. and Tennenholtz, M. (2002). R-max – A General Polynomial Time Algorithm for Near-Optimal Reinforcement Learning. , 2002.

Charpentier, A., Élie, R. and Remlinger, C. (2021). Reinforcement Learning in Economics and Finance. *Computational Economics* [online], 23 April 2021. Available from: https://link.springer.com/10.1007/s10614-021-10119-4 [accessed 22 April 2023].

Chen, L., Lu, K., Rajeswaran, A., Lee, K., Grover, A., Laskin, M., Abbeel, P., Srinivas, A. and Mordatch, I. (2021). Decision Transformer: Reinforcement Learning via Sequence Modeling. , 24 June 2021. Available from: http://arxiv.org/abs/2106.01345 [accessed 22 April 2023].

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H. and Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. , 2 September 2014. Available from: http://arxiv.org/abs/1406.1078 [accessed 22 April 2023].

Clark, C. and Storkey, A. (2015). Training Deep Convolutional Neural Networks to Play Go. In: Bach, F. and Blei, D., eds. *Proceedings of the 32nd International Conference on Machine Learning*. Proceedings of Machine Learning Research. Lille, France: PMLR, pp.1766–1774. Available from: https://proceedings.mlr.press/v37/clark15.html [accessed 20 November 2022].

Dajose, L. (2021). *What Neural Networks Playing Video Games Teach Us About Our Own Brains* [online]. *California Institute of Technology* [online]. Available from: https://www.caltech.edu/about/news/neural-networks-playing-video-games-teach-us-about-our-own-brains [accessed 12 November 2022].

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J. and Houlsby, N. (2021). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. , 3 June 2021. Available from: http://arxiv.org/abs/2010.11929 [accessed 22 April 2023].

Farama Foundation. (2022). *The Farama Foundation* [online]. *The Farama Foundation* [online]. Available from: https://farama.org/ [accessed 25 April 2023].

Fujimoto, S., van Hoof, H. and Meger, D. (2018). Addressing Function Approximation Error in Actor-Critic Methods. , 22 October 2018. Available from: http://arxiv.org/abs/1802.09477 [accessed 26 April 2023].

Gallistel, C.R. (1999). Reinforcement Learning, reviewed in Journal of Cognitive Neuroscience. , 1999, pp.126–134.

Gullapalli, V. (1992). *Reinforcement Learning and its Application to Control* [unpublished]. University of Massachusetts.

Haarnoja, T., Zhou, A., Abbeel, P. and Levine, S. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. , 8 August 2018. Available from: http://arxiv.org/abs/1801.01290 [accessed 26 April 2023].

Han, K., Wang, Y., Chen, H., Chen, X., Guo, J., Liu, Z., Tang, Y., Xiao, A., Xu, C., Xu, Y., Yang, Z., Zhang, Y. and Tao, D. (2023). A Survey on Vision Transformer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 45(1), pp.87–110.

Hannan, J. *Colin McRae's Rally 2, interview with developer Jeff Hannan* [online]. *ai-junkie* [online]. Available from: http://www.ai-junkie.com/misc/hannan/hannan.html [accessed 13 November 2022].

Hasselt, H. van, Guez, A. and Silver, D. (2016). Deep Reinforcement Learning with Double Q-Learning. *Proceedings of the AAAI Conference on Artificial Intelligence* [online], 30(1). Available from: https://ojs.aaai.org/index.php/AAAI/article/view/10295 [accessed 23 April 2023].

Houlsby, N. and Weissenborn, D. (2020). Transformers for Image Recognition at Scale. *Google AI Blog* [online], 3 December 2020. Available from: https://ai.googleblog.com/2020/12/transformers-for-image-recognition-at.html [accessed 21 November 2022].

James, W. (1890). *The Principles of Psychology. Chapter XI: Attention* [online]. *Classics in the History of Psychology* [online]. Available from: http://psychclassics.yorku.ca/James/Principles/prin11.htm [accessed 13 November 2022].

Janner, M., Li, Q. and Levine, S. (2021). Offline Reinforcement Learning as One Big Sequence Modeling Problem. , 28 November 2021. Available from: http://arxiv.org/abs/2106.02039 [accessed 22 April 2023].

LeCun, Y., Bengio, Y. and Hinton, G. (2015). Deep learning. *Nature*, 521(7553), pp.436–444.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D. and Wierstra, D. (2019). Continuous control with deep reinforcement learning. , 5 July 2019. Available from: http://arxiv.org/abs/1509.02971 [accessed 26 April 2023].

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S. and Guo, B. (2021). Swin Transformer: Hierarchical Vision Transformer using Shifted Windows. , 17 August 2021. Available from: http://arxiv.org/abs/2103.14030 [accessed 22 April 2023].

Mänttäri, J. and Larsson, J. (2011). *Applications of Artificial Neural Networks in Games; An Overview*. Västerås, Sweden: Mälardalen University.

Martín Abadi et al. (2015). TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems. , 2015. Available from: https://www.tensorflow.org/.

Meng, L., Goodwin, M., Yazidi, A. and Engelstad, P. (2022). Deep Reinforcement Learning with Swin Transformer. , 30 June 2022. Available from: http://arxiv.org/abs/2206.15269 [accessed 23 April 2023].

Millington, I. (2019). *AI for games*. Third edition. Boca Raton: Taylor & Francis, a CRC title, part of the Taylor & Francis imprint, a member of the Taylor & Francis Group, the academic division of T&F Informa, plc.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T.P., Harley, T., Silver, D. and Kavukcuoglu, K. (2016). Asynchronous Methods for Deep Reinforcement Learning. , 16 June 2016. Available from: http://arxiv.org/abs/1602.01783 [accessed 26 April 2023].

Mnih, V. et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), pp.529–533.

Nowé, A., Vrancx, P. and De Hauwere, Y.-M. (2012). Game Theory and Multi-agent Reinforcement Learning. In: Wiering, M. and van Otterlo, M., eds. *Reinforcement Learning: State-of-the-Art*. Adaptation, Learning, and Optimization. Berlin, Heidelberg: Springer, pp.441–470. Available from: https://doi.org/10.1007/978-3-642-27645-3_14 [accessed 22 April 2023].

O'Donoghue, B., Osband, I., Munos, R. and Mnih, V. The Uncertainty Bellman Equation and Exploration.

O'Shea, K. and Nash, R. (2015). An Introduction to Convolutional Neural Networks. , 2 December 2015. Available from: http://arxiv.org/abs/1511.08458 [accessed 13 November 2022].

Paszke, A. et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: Wallach, H. et al., eds. *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., pp.8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Puterman, M.L. (1990). Chapter 8 Markov decision processes. In: *Handbooks in Operations Research and Management Science*. Elsevier, pp.331–434. Available from: https://www.sciencedirect.com/science/article/pii/S0927050705801720.

Qualls, J. and David, R.J. (2009). Applications of Neural-Based Agents in Computer Game Design. In: Dos Santos, W. P., ed. *Evolutionary Computation*. InTech. Available from: http://www.intechopen.com/books/evolutionary-computation/applications-of-neural-based-agents-in-computer-game-design [accessed 12 November 2022].

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M. and Dormann, N. (2021). Stable-Baselines3: Reliable Reinforcement Learning Implementations. *Journal of Machine Learning Research*, 22(268), pp.1–8.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. and Klimov, O. (2017). Proximal Policy Optimization Algorithms. , 28 August 2017. Available from: http://arxiv.org/abs/1707.06347 [accessed 26 April 2023].

Silver, D. et al. (2016). Mastering the game of Go with deep neural networks and tree search. *Nature*, 529(7587), pp.484–489.

Skinner, G. and Walmsley, T. (2019). Artificial Intelligence and Deep Learning in Video Games A Brief Review. In: 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS). Singapore: IEEE, pp.404–408. Available from: https://ieeexplore.ieee.org/document/8821783/ [accessed 13 November 2022].

Supervised vs Unsupervised Learning Explained. (2022). *Seldon* [online], 16 September 2022. Available from: https://www.seldon.io/supervised-vs-unsupervised-learning-explained [accessed 14 November 2022].

Sutskever, I., Vinyals, O. and Le, Q.V. (2014). Sequence to Sequence Learning with Neural Networks. , 14 December 2014. Available from: http://arxiv.org/abs/1409.3215 [accessed 24 April 2023].

Sutton, R.S. and Barto, A.G. (2018). *Reinforcement Learning, second edition: An Introduction*. 2nd ed. Adaptive computation and machine learning series. United States of America: Westchester Publishing Services.

Tao, T., Reda, D. and van de Panne, M. (2022). Evaluating Vision Transformer Methods for Deep Reinforcement Learning from Pixels. , 15 May 2022. Available from: http://arxiv.org/abs/2204.04905 [accessed 22 April 2023].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I. (2017). Attention Is All You Need. , 5 December 2017. Available from: http://arxiv.org/abs/1706.03762 [accessed 22 April 2023].

Watkins, C.J.C.H. and Dayan, P. (1992). Q-learning. *Machine Learning*, 8(3), pp.279–292.

Yang, Z.R. and Yang, Z. (2014). Artificial Neural Networks. In: Brahme, A., ed. *Comprehensive Biomedical Physics*. Oxford: Elsevier, pp.1–17. Available from: https://www.sciencedirect.com/science/article/pii/B9780444536327011011 [accessed 12 November 2022].

Zahedi, F. (1991). An Introduction to Neural Networks and a Comparison with Artificial Intelligence and Expert Systems. *Interfaces*, 21(2), pp.25–38.

Zhao, Y., Wang, G., Tang, C., Luo, C., Zeng, W. and Zha, Z.-J. (2021). A Battle of Network Structures: An Empirical Study of CNN, Transformer, and MLP. , 25 November 2021. Available from: http://arxiv.org/abs/2108.13002 [accessed 22 April 2023].