Computer Games Development
SE607

Technical Design Document

Year IV

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.

Table of Contents

1. 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.

2. Introduction

2.1. Intended audience and Reading suggestions

This document is intended for individuals seeking a deeper understanding of the code utilised in conjunction with the research paper tilted "*What advantages does using a Vision Transformer model offer over Convolutional Neural Network in playing video games?*" (Dudon 2023). The project aims to compare two neural network architectures in the context of Reinforcement Learning and determine the effectiveness and benefits of one over the other. For a more detailed description of the project, please refer to the project report.

2.2. Purposes

The purpose of this software is to implement the Double Deep Q Network algorithm (Hasselt et al. 2016) using PyTorch. The objective is to train an agent to play various Atari games environments, with a focus on image recognition, using two different neural network architectures: Convolutional Neural Networks and Swin Transformer. The software aims to support the comparison and assumptions made in the research paper by providing various performance metrics, such as data and charts, and by training a Deep Reinforcement Learning agent to evaluate the effectiveness and benefits of each neural network architecture.

3. Technical Design

3.1. Technology stack

3.1.1. Programming language

Python was chosen as the primary programming language for this project due to its widespread use in the machine learning community and the wealth of technical resources available in Python. Specifically, two versions of Python, 3.8 and 3.9, will be utilized in their respective Conda environments to ensure compatibility with all necessary libraries and packages.

3.1.2. Frameworks and Libraries

In this project, I exclusively utilize open-source libraries and frameworks. Although not exhaustive, the following list comprises the principal and most critical libraries used in my work.

Technology	**Version**	**Link**	**Description**
TensorFlow	2.12	https://www.tensorflow.org/	Machine Learning library for deep neural networks developed by Google and using Keras as backend.
PyTorch	2.0	https://pytorch.org/	Machine Learning framework for deep neural networks developed by the Linux Foundation umbrella.
Gymnasium	0.28.1	https://gymnasium.farama.org/	Standard API for reinforcement learning, and a diverse collection of reference environments.
Stable Baselines3	1.8.0	https://stable-baselines3.readthedocs.io/en/master/	Set of reliable implementations of reinforcement learning algorithms in PyTorch.
Jupyter Notebook	6.5.4	https://jupyter.org/	Interactive web-based Python code interface.
Transformers	4.28.1	https://huggingface.co/docs/transformers/	State-of-the-art collection of Transformers neural network architectures developed by Hugging Face.
Swin Transformer (Official Microsoft Implementation)	N/A	https://github.com/microsoft/Swin-Transformer	Official Microsoft implementation of Swin Transformer in PyTorch.

3.1.3. Data storage

The data gathered during training will be stored in TensorBoard log files and Excel/CSV files. Videos will be saved in the .mp4 format. Trained models and checkpoints will be saved in the .pth format for PyTorch and the .h5 format for TensorFlow.

3.1.4. Development tools

Software packages and tools used to develop the project.

- **Linux –** Ubuntu-based Operating System is recommended (i.e. Pop!_OS or Ubuntu) version 22.04
- **Miniconda:** small bootstrapped version of Anaconda. Anaconda is an open-source package and environment management system that improve the default Python environment management and pip package manager.
- **JetBrains PyCharm Professional**: multipurpose Python IDE
- **Git**: version control system to manage code
- **GitHub**: to store the code remotely and share it easily

3.1.5. System Dependencies

- NVIDIA GPU drivers (version 450.80.02 or higher)
- CUDA Toolkit 11.8 (Only needed for TensorFlow)
- cuDNN SDK 8.6.0 (Only needed for TensorFlow)

3.1.6. Others

- **LibreOffice**: to write the project documentations and generate the chart from the gathered data
- **Zotero**: manage bibliography of references for research
- **Notion**: create guides and resources, project management, note-taking

3.2. Architecture

The architecture diagram presented in *Figure 1* provides a schematic representation of the component interactions. While the codebase does not comprise a high number of classes, the experimental nature of the project necessitated extensive code rewrites. As a result, a considerable amount of previously developed code, notably the TensorFlow implementation of the Deep Q Network, had to be discarded.

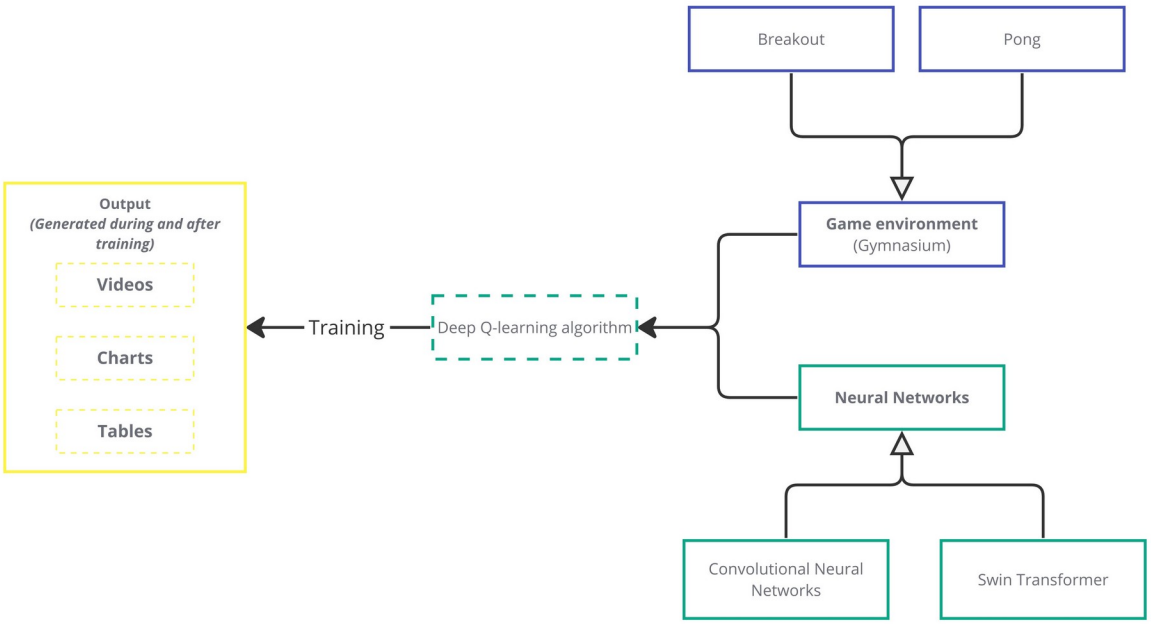

Figure 1: Rough representation of the code architecture

3.3. Class Responsibility Collaborator (CRC) Cards

DQNAgent	
• The class implement the Double DQN algorithm • Contains utility method to save and load a checkpoint of the training • Have to be network and environment agnostic • Hyperparameters are passed with a dictionary.	ReplayBuffer

ReplayBuffer	
• Store the experience replay of the DQN Agent • The buffer must be implemented as a deque. • It should allow the storage of: state, next state, reward, episode done and action taken to go from current state to next state.	

4. Features Breakdown

4.1. Deep Q Network

The Double Deep Q Network algorithm is the core component of the program and is used to perform the experiments. This algorithm is described in more detail in the project report document. To ensure reusability, the DQN algorithm should be implemented as a class, which takes the environment and the neural network architecture as input.

4.1.1. Algorithm

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)

4.1.2. Hyperparameters

The exact hyperparameters used in the algorithm are described in *Table 1*.

Input	4x84x84
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

Table 1: Double Deep Q Network hyperparameters

4.2. Neural Networks

4.2.1. Convolutional Neural Network

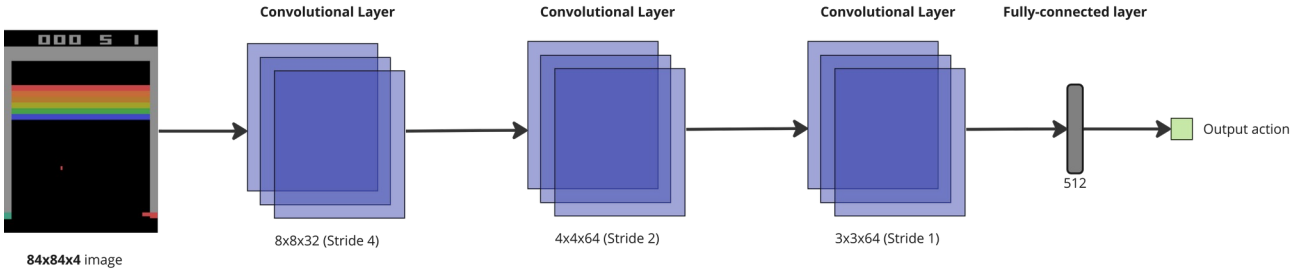

Figure 2: Convolutional Neural Network architecture of the Double DQN

Figure 2 shows the schematic representation of the Convolutional Neural Network architecture. In *table 2*, describes the exact parameters that are used.

Table 2: 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

4.2.2. Swin Transformer

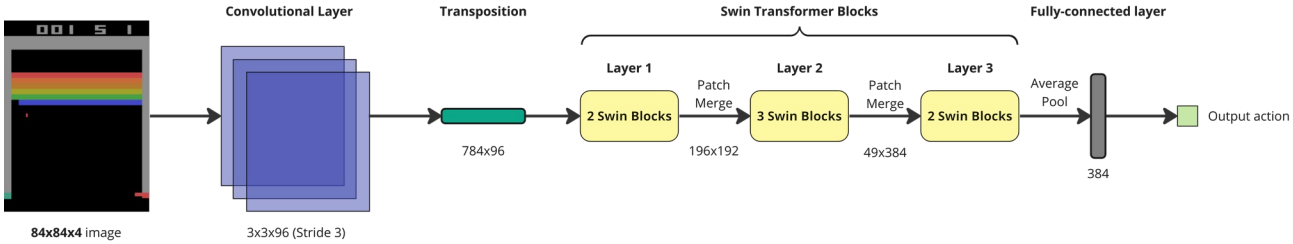

Figure 3: Swin Transformer architecture of the Double DQN

Figure 3 shows the schematic representation of the Swin Transformer architecture. In *table 3*, describes the exact parameters that are used.

Table 3: 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

4.3. Environment preprocessing

The environment used in the project is provided by the Gymnasium library, which is a collection of reinforcement learning environments. The specific environment used is part of the Atari group of environments, which are simulations of classic Atari games. The environments are simulated using the Arcade Learning Environment (ALE) and the Stella emulator.

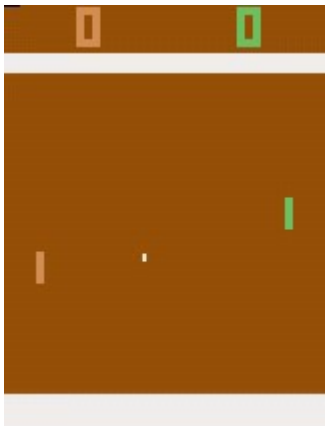

Image 1: Rendering of the Pong environment

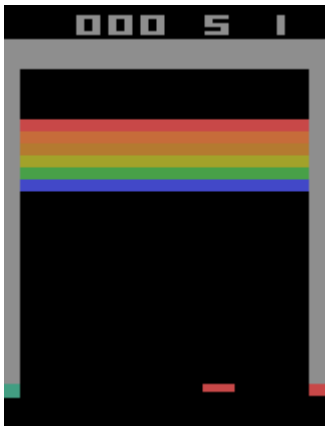

Image 2: Rendering of the Breakout environment

Prior to being used, the environment provided by Gymnasium necessitates a specific preprocessing phase in order to adhere to the requirements of the Deep Q-Network algorithm. As elucidated by Mnih et al. (2015), the raw image produced by the environment cannot be utilized as input for the model. The preprocessing specifications are as follows:

1. Every step in the environment comprises four frames; thus, one timestep is equal to four frames.
2. The default frameskip applied to the environment by the Atari emulator must be disabled as it is not consistent; instead, the parameter frameskip=1 should be utilized.
3. The image provided by the environment must be reduced to a size of 84×84.
4. The image's colors must be converted to grayscale.
5. The environment must return a stacked image consisting of the four most recent frames. Consequently, the image channel is four.

Following the aforementioned preprocessing steps, the image format is finalized as 84x84x4 (HxWxC), where H denotes height, W denotes width, and C denotes the number of channels.

4.4. Logs

During the training process, logging of training information is essential to track the performance of the model. To this end, TensorBoard is used to write and host training logs on TensorBoard.dev. At the end of each episode, five scalar values must be stored and displayed through TensorBoard. The first value is the current timestep value of the training loop. The other values include episodic return, which denotes the score reached by the agent for each episode, episodic length, which indicates the length of each episode, episode reward mean, which represents the mean reward over the last 100 episodes, and episode length mean, which shows the mean length over the last 100 episodes. In addition to these, the exploration rate, that is, the value of the exploration rate (epsilon) for each episode must be recorded.

Apart from the above values, data related to the gradient step backpropagation should be stored after each gradient step optimization. This includes the loss value, which is calculated between the Q values and the target Q values, and the Q value, which depicts the evolution of the q values over time.

4.5. Checkpoint

4.5.1. Saving checkpoint

During the training phase, which can take several hours to days, it is essential to regularly save checkpoints of the current training state to prevent potential crashes or issues. A checkpoint should retain various information regarding the current state of the training. Specifically, the checkpoint must include the environment ID, the current timestep, the exploration rate (epsilon), the policy network weights, the target network weights, and the optimizer weights. In addition, the replay buffer must be saved, which can be achieved using the Cloud Pickle feature from Python, as its size can be substantial, ranging down to the GB.

To save the above information, the PyTorch saving mechanism (torch.save()) can be utilized. However, the replay buffer must be stored separately from the PyTorch checkpoint file using the Cloud Pickle feature. It is important to note that the PyTorch checkpoint file must use the .tar extension, while the replay buffer checkpoint file must use the .pkl extension.

4.5.2. Loading checkpoint

Loading a previously saved checkpoint should be a seamless process. Before resuming the training loop, the following data must be set accordingly:

- Checkpoint timestep
- Epsilon value
- Policy network weights
- Target network weights
- Optimizer weights
- Replay buffer

All of these components must be restored to their state at the time the checkpoint was saved. The PyTorch load method (e.g. torch.load()) can be used to load the model and optimizer weights, while the Cloud Pickle load method (e.g. cloudpickle.load()) can be used to load the replay buffer.