**Construction of an AI player for historic arcade games using object recognition and reinforcement learning**

A project proposal submitted in partial fulfilment of the requirements for the MSc in Advanced Computing Technologies

by Henrik Bilar

Department of Computer Science and Information Systems

Birkbeck, University of London

April 2019

Supervisor: Prof Stephen J. Maybank

**Academic declaration:**

This report is substantially the result of my own work except where explicitly indicated in the text. I give my permission for it to be submitted to the TURNITIN Plagiarism Detection Service. I have read and understood the sections on plagiarism in the Programme Handbook and the College website.

The report may be freely copied and distributed provided the source is explicitly acknowledged.

Table of Contents

1. Introduction 2

2. Background research of related work 2

2.1 DeepMind playing Atari games 2

2.2 Gym toolkit from OpenAI 2

2.3 TensorFlow for Object Detection 3

3. Problem description 4

3.1 Problem description - the emulator 5

3.2 Problem description – the training set 5

3.3 Problem description – Training the TensorFlow object detection model 5

3.4 Problem Description – The AI Player 6

3.4.1 AI player to emulator network communication 6

3.4.2 Game play network 7

3.4.3 Training of the GPN 7

3.4.4 Other functions of the AI player software 8

4. Fall back position 8

5. Work-plan and schedule 9

5.1 Tools and programming languages 9

5.1.1 NES Emulator 9

5.1.2 Python PyGame graphical library 9

5.1.3 TensorFlow object detection model – SSD Mobilenet v2 10

5.2 Indicative schedule 10

6. References 12

# 1. Introduction

In 1983 Nintendo released its “Nintendo Entertainment System” (NES) games console (Wikipedia, 2019). A large variety of games were produced for the NES, and there were over 61 million NES consoles sold worldwide.

The console is connected to a TV to display images. The player controls the game using a handheld controller known as a “joypad”. The joypad has directional left/right/up/down type button (where two adjacent directions buttons can be pressed at once), and also an A and a B button. There is also a “start” and a “select” button, but these are typically not used in game play.

Even though the NES console itself has not been on sale for many years, it is still possible to play NES games today using one of a variety of software emulators. A NES software emulator accurately simulates the original NES console, with the original TV display rendered on a computer screen, allowing a player to play the games.

In the 36 years since the NES console was released, Artificial Intelligence (AI) has come a long way, particularly in the field of object recognition and deep neural networks. This project aims to build a stand-alone AI program to act as a player for the well-known game Super Mario Brothers, by “seeing” (detecting) objects in the video stream produced by the emulator, and then responding by issuing commands to the emulator. The commands will be produced by a neural network that will take as input the current state of the world in the game.

# 2. Background research of related work

## 2.1 DeepMind playing Atari games

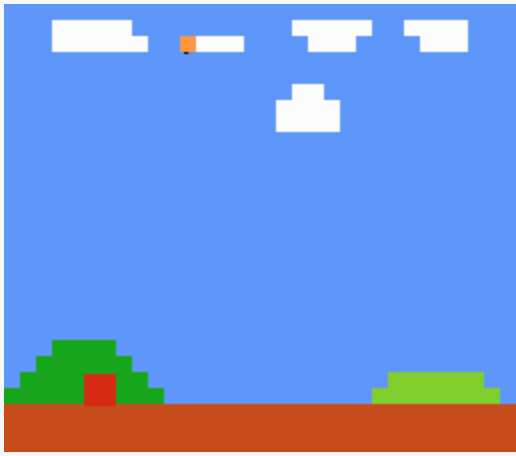
DeepMind have shown that it is possible to play Atari games with Deep Reinforcement Learning (Volodymyr Mnih, 2013). DeepMind’s solution feeds the raw video stream (as pixels) from an Atari emulator into a convolutional neural network, and trains the network using reinforcement learning. Only the reward function and action space are hand coded for each game. Note, this is different to the approach of this project, as DeepMind did not explicitly attempt to detect objects, and then feed them into a separate network.

## 2.2 Gym toolkit from OpenAI

The Gym toolkit (OpenAI) is a framework for developing Reinforcement Learning (RL) algorithms. It can be used for any type of RL training, and it supports plugins for extending the functionality. Several plugins exist for training a Super Mario Brothers player, for example gym-super-mario (Paquette, 2018) and Super Mario Bros for OpenAI Gym (Kauten, 2018).

(Kauten, 2018) shows down-sampling of game play images, to reduce the number of discrete pixels considered by the model. This could potentially lead to great speed-ups in detecting objects in games, at the cost of positional accuracy. Figure 2.1 shows an original image, and the down-sampled equivalent.

Figure 2.1. Left image shows original game frame, and right image shows a down sampled image (Kauten, 2018). Information has been lost from the right image, but the information loss also means a computer can easily identify Mario in the frame, even by simple iteration over the image.

## 2.3 TensorFlow for Object Detection

TensorFlow (TF) (Google/Tensorflow, 2019) is a popular Open Source framework used in many areas of machine learning (ML). TF is used by both researchers and companies building ML enabled products. Example products include language translation, classifying photographs, and financial fraud detection (TensorFlow).

TF can also be used for object detection in images (Google, 2019). Several pre-trained models of varying speed/accuracy are available to download from an “Object detection Zoo” (Google, 2019). Given an input image, the models generate a set of suggested bounding boxes, object labels, and a confidence scores. The confidence score describes how certain the model is that the label and bounding box is correct. Figure 2.2 shows the bounding boxes overlaid on a sample image.

TF has also been used to detect and classify objects in video streams, even on low powered mobile devices. An example application is a real time dog breed detector (Sara Robinson, 2018). The authors claim to be able to train their object detection model in less than 30 minutes on Google’s cloud platform.



Figure 2.2. Labelled objects detected in sample photograph (Google, 2019)

# 3. Problem description

In order to produce a full, standalone, AI player setup, several pieces of software will have to be built, training/test data generated, and neural networks trained.

The following high-level tasks have been identified:

* An existing open source NES emulator will be modified to send a video stream over a network connection, and also accept inputs over a network connection
* The TensorFlow (Google/Tensorflow, 2019) libraries will be leveraged for enemy object detection. This requires a training set to be generated for the object detection, The training set will be created by a program that will be developed as part of this project
* Training of an existing TensorFlow object detection model
* The AI player software itself, which will use TensorFlow’s object detection to build a view of the world. This view will be fed into a model that will produce the appropriate key combination to press to play the game. The AI player software will also need to deal with the interaction with the modified emulator and displaying images to the screen, so that a human can see how the AI player plays.

The emulator and player will be completely separate, and they will share no state. All communication will happen over a network channel, in order to make the AI player appear as much as possible as a real human.

## 3.1 Problem description - the emulator

The job of the NES emulator is to run NES games on hardware that is different to the original NES console. This is achieved by emulating the different hardware components that made up the original NES console in software. There are a number of different Open Source NES emulators available, written in a range of programming languages.

The main requirements for the emulator used in this project are:

* Ease of modification, and written in a language that the author of this project feels comfortable using.
* Accurate emulation to ensure that the game runs as well as possible.

As the goal of this project is to create a software player, an interface for interaction has to be defined. The AI player cannot directly press buttons on the keyboard, nor “see” the screen, so the emulator will be modified to accept input over a network connection. The emulator will also send the screen pixel values to the AI player over the same network connection.

## 3.2 Problem description – the training set

The pre-trained TensorFlow object detection models cannot distinguish objects in NES games, as they have not been trained on them. The training set for object detection models typically consist of a large number of images, with ground truth boxes defined in XML, typically created by hand (Vladimirov, 2019).

As there are only a limited number of enemies and objects to detect in Super Mario Brothers, a stand-alone Python program to generate training data will be developed, thus avoiding time-intensive manual training set creation.

## 3.3 Problem description – Training the TensorFlow object detection model

There are a variety of object detection models for TensorFlow already developed (Google, 2019). The models implemented in TensorFlow broadly fall into Regional Convolutional neural networks (RCNN), or Single Shot Detection networks (SSD).

Pre-trained models, trained on common data sets, are available for download from the TensorFlow Object Detection Zoo (Google, 2019). TensorFlow provide average inference times for the models in the zoo, and it appears that the SSD based models are mostly quicker for detection. This project will likely utilize an SSD based model, as that should allow the full setup to run on a low spec computer without a GPU.

Training a TensorFlow model can take several days of GPU time, however as the project aims to only detect objects within a game frame, it is expected that the required training time will be somewhat shorter. Should training using one computer be too slow, it is possible to train the model on cloud-based GPUs (Google Cloud). This will speed the training up tremendously.

## 3.4 Problem Description – The AI Player

The AI player software will likely be the biggest piece of software for this project. In addition to running the models for object detection and game play, it will need to act as a bridge between the emulator and the models, and also to display images to an actual screen so that a human can see how well the player plays.

An overview of the expected data flows in the system is shown in Figure 3.1 below. To simplify development, the data flows and steps will be kept synchronous, even though the several of the peripheral components could run in parallel.

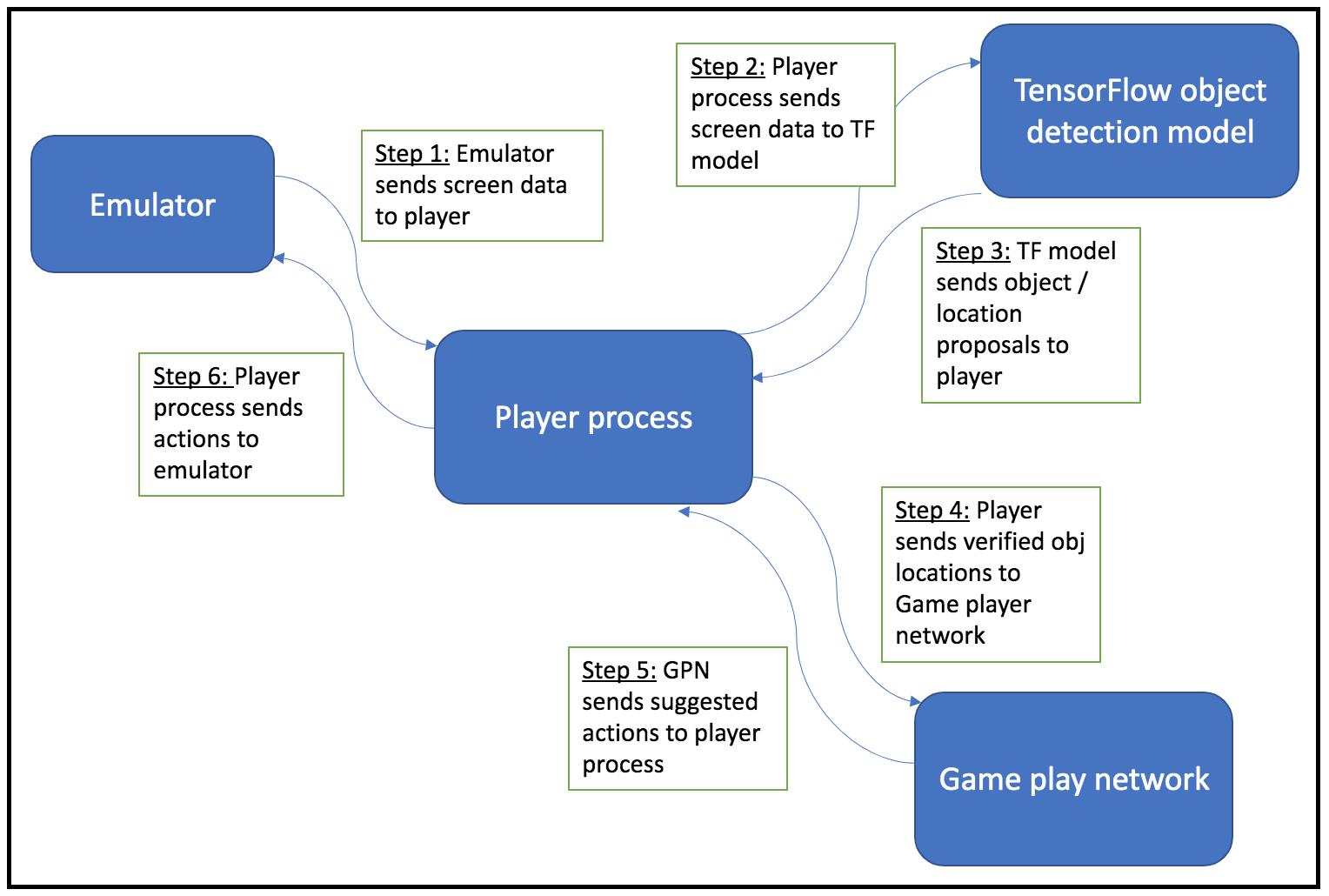


Figure 3.1 – Data flows between the various components.

### 3.4.1 AI player to emulator network communication

The communication between the player software and the emulator will most likely be over a simple bi-directional TCP/IP based protocol. The following functionality will need to be offered by the system:

* Player to request screen update from emulator, and emulator responding with the current pixel values for the internal frame buffer used to represent the screen. The player will need to store the pixels in its own frame-buffer, so that the data can be fed to the object detection model.
* Player to send joypad input to the emulator. That is, simulate key presses, which will make Mario perform actions in the game.
* Player to request a reset the console and restart from the beginning. This particular functionality is not strictly required, however it is likely to speed up training of the RL model, as the time to restart the full setup could be quite large. This would be particularly true if the model was trained using a cloud, where stopping and starting an instance is potentially not as fast as on a local system.

There is a risk that the AI player will be too slow to keep up with a real-time video feed. To overcome this, all the network operations will probably have to be initiated by the AI player, as it will allow the player to throttle the amount of data, by being able to request data from the emulator when it is ready to process it.

### 3.4.2 Game play network

The actual game playing decisions will be taken by a Game Playing Network (GPN). It will consist of a neural network with several layers. The network will have one output neuron per possible operation of the joypad. There are four directional buttons, of which two adjacent buttons can be pressed at once (for example, up and left), and also two relevant buttons (‘a’ and ‘b’), giving a total of 9 possible directional outputs (including ‘none’), and 4 different states of the buttons (for a total of 9 \* 4 = 36 states).

The inputs to the network will need to include at least the following:

* Distance to next enemy relative to Mario (both left and right)
* Distance to next obstacle (left and right)
* Distance to next hole (where Mario can fall off the screen and hence die).

It is possible that as the solution gets implemented, more inputs will be required, but these are not known yet. For example, the game levels in Super Mario Brothers are time limited, and one potential input would be the time left until time is up (Perhaps to get Mario to run (as opposed to walking) as time runs out).

### 3.4.3 Training of the GPN

RL has applications in robotics, where robots are taught to find good solutions to various problems by means of trial-and-error (Jens Kober, 2013). This sounds similar to what the AI player has to achieve, except the AI player acts in a software universe. This project will attempt to devise and train a GPN using RL, perhaps combined with an evolutionary approach.

To clear a level in Super Mario Brothers, the player has to move as far as possible to the right, where Mario reaches a flagpole. A reward function to reflect this will need to be devised. There are additional constraints for the reward function, such as a time element – a level must be cleared before time runs out.

### 3.4.4 Other functions of the AI player software

In addition to the functions mentioned previously, the AI player software will also need to be able to detect when the game has finished. That is, when Mario runs out of time, is killed by a monster, or falls down a hole off the screen.

The AI player will also need to be able to decode the textual information at the top of the screen to determine the amount of time left for the level. As can be seen from Figure 3.2, the information needed to identify the time is just the numbers 0-9, and with the numbers always being uniform and in the exact same locations, the software could use a simple byte pattern match to determine the number values.



Figure 3.2 - typical game play scene with textual information seen in the top 1/5th of the screen

The AI player software will also be responsible for calculating the reward function as described in section 3.4.3.

Super Mario Brothers involves elements of collecting coins for extra rewards, and the concept of in-game points that can be collected by killing enemies, or completing a level quickly. However, as these factors do not directly affect the objective of completing a level, they will be ignored.

# 4. Fall back position

The stated aim of this project is to create a piece of software that can play Super Mario Brothers, and at least clear level one. As this is quite an ambitious undertaking, a suggested fall back position, should the main goal prove too difficult to implement in the time available, is:

* Modify the emulator to send/receive data over the network
* Create a training set for object recognition using a Python program
* Train a TensorFlow model using the generated dataset
* Create the beginning of the AI player software, such that it can at least connect to the emulator, display the relevant graphics on screen, and use the object detection model to correctly identify objects on the screen (and draw bounding boxes around them)

# 5. Work-plan and schedule

## 5.1 Tools and programming languages

This section details the main Open Source components likely to be used in the final system.

Python will be used for all programming, except where other constraints force a different language. For example, the NES Emulator chosen is written in C/C++.

### 5.1.1 NES Emulator

This project will utilize the LaiNES open source NES emulator. It is a multi-platform emulator, written in roughly 1000 lines of C/C++, and it can be extended to provide the networking functions required by the rest of the project (Orru, 2017).

As implemented on GitHub, the emulator utilizes the SDL graphics library. As a consequence the emulator will not run without a screen to draw images onto (also known as “headless mode”). The lack of headless mode complicates training, as most cloud-based do not have a native display, which means that the emulator cannot currently run in the cloud. It may be possible to provide enough of a fake screen by running the emulator in for example VNC (TigerVNC developers, 2019), or with the X virtual frame buffer (Wiggins). Alternatively, the graphics routines could be removed from the emulator, however this could prove time consuming.

### 5.1.2 Python PyGame graphical library

The AI player needs to draw images onto the screen, so that a human may observe current game-play. The AI player will utilize the open source PyGame library for graphical functionality (Shinners, 2000). It was first released in 2000, and is still in active development, and a large number of applications and games use the library.

### 5.1.3 TensorFlow object detection model – SSD Mobilenet v2

SSD style object detection models are already implemented in TensorFlow. This project will use the SSD Mobilenet v2 model as it performs fast, and still has reasonable accuracy.

The Mobilenet model uses grey-scale images as inputs, and there is a risk that the proposed objects are of the wrong type, because their shape is similar.

For example, the background shrubbery and the “Goomba” enemy have a similar shape, as seen in Figure 5.1



Figure 5.1 The shrubbery and background mountain have the same shape as the enemy seen on this screenshot.

To reach an acceptable accuracy for the object detection, it may be necessary to perform extra analysis on the model suggestions where collisions are likely. In this particular example, looking at the colour histogram might help determine if the object is a real enemy or not.

## 5.2 Indicative schedule

**Note on the schedule:**

The project was commenced before the proposal was completed, shortly after a project supervisor was found. As a consequence, some tasks have already been completed. As of the beginning of April 2019, the emulator has been modified, a rudimentary Python program exists (without a Neural Net for playing), a TensorFlow object detection model has been trained (with poor accuracy), and the Python program is able to use the TensorFlow model to identify objects and draw bounding boxes around them.

**January:**

Modify the LaiNES emulator (Orru, 2017), such that an external process can communicate with it to play a game. The key requirement here is to be able to connect to the emulator using TCP/IP.

Produce a skeleton Python program that can connect to the emulator over TCP/IP and display the game play images, and send commands (joypad instructions) to the emulator.

**February:**

Generate a training set for TensorFlow using a Python program and screenshots taken from Super Mario Brothers, and train a model using the training set.

**March:**

Develop the neural network part, with the aim of having a model that can start training towards the end of the month, at least locally on a desktop/laptop.

Develop a reward function that produces a reward from the game state, for example checking if Mario is still alive, the number of seconds left on the level countdown timer, and how far along the course Mario is.

Finish up the project proposal to a state such that it can be submitted.

**April:**

Submit project proposal.

Continue tweaking the neural net/reward function until satisfactory results are achieved.

**May:**

The reward function should be complete by now, and the player should take at least some action when presented with different scenarios in the game.

**June - July:**

Write project report.

**August:**

Contingency.

# 6. References

Google Cloud. (n.d.). *Cloud ML Training Overview*. Retrieved 04 06, 2019, from https://cloud.google.com/ml-engine/docs/tensorflow/training-overview

Google. (2019, 03 7). *Tensorflow Detection Zoo*. Retrieved 03 25, 2019, from https://github.com/tensorflow/models/blob/master/research/object\_detection/g3doc/detection\_model\_zoo.md

Google. (2019, 03 16). *TensorFlow Object Detection*. Retrieved 03 16, 2019, from https://github.com/tensorflow/models/tree/master/research/object\_detection

Google. (2018, 07 13). *TensorFlow Object Detection Sample Configuration mobilenet v2 coco*. Retrieved 03 23, 2019, from https://github.com/tensorflow/models/blob/8d5d36e0a0e7ffb9b8746d6fb1d88b3cc7566b40/research/object\_detection/samples/configs/ssd\_mobilenet\_v2\_coco.config

Google/Tensorflow. (2019, 03 16). *TensorFlow*. Retrieved 03 16, 2019, from http://www.tensorflow.org

Jens Kober, J. A. (2013). Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research* *, 32* (11), 1238-1274.

Kauten, C. (2018). *Super Mario Bros for OpenAI Gym*. Retrieved from GitHub: https://github.com/Kautenja/gym-super-mario-bros

OpenAI. (n.d.). *Gym*. Retrieved 03 25, 2019, from OpenAI Gym: https://gym.openai.com/

Orru, A. (2017, 02 06). *LaiNES emulator on GitHub*. Retrieved 03 25, 2019, from https://github.com/AndreaOrru/LaiNES

Paquette, P. (2018, 07 3). *gym-super-mario*. Retrieved 03 25, 2019, from https://github.com/ppaquette/gym-super-mario

Sara Robinson, A. C. (2018, 07 13). *Training and serving a realtime mobile object detector in 30 minutes with Cloud TPUs*. Retrieved 03 23, 2019, from Medium Corporation US: https://medium.com/tensorflow/training-and-serving-a-realtime-mobile-object-detector-in-30-minutes-with-cloud-tpus-b78971cf1193

Shinners, P. (2000, 10 28). Retrieved 03 25, 2019, from www.pygame.org: http://www.pygame.org

TensorFlow. (n.d.). *TensorFlow case studies*. Retrieved 04 07, 2019, from TensorFlow.org: https://www.tensorflow.org/about/case-studies/

TigerVNC developers. (2019, 03 01). *GitHub - TigerVNC*. Retrieved 04 12, 2019, from https://github.com/TigerVNC/tigervnc

Vladimirov, L. (2019, 02 28). *Training Custom Object Detector*. Retrieved 03 25, 2019, from https://github.com/sglvladi/TensorFlowObjectDetectionTutorial/blob/master/docs/source/training.rst

Volodymyr Mnih, K. K. (2013, 12 19). *Playing Atari with Deep Reinforcement Learning.* Retrieved 03 25, 2019, from arXiv.org: https://arxiv.org/pdf/1312.5602

Wiggins, D. P. (n.d.). *XVFB*. Retrieved 04 06, 2019, from X.org Foundation: https://www.x.org/archive/X11R7.6/doc/man/man1/Xvfb.1.xhtml

Wikipedia. (2019, 03 01). *Nintendo Entertainment System*. Retrieved 03 16, 2019, from Wikipedia: https://en.wikipedia.org/wiki/Nintendo\_Entertainment\_System