UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Master in Deep Learning for
Audio and Video Signal Processing

MASTER THESIS

SEMANTIC SEGMENTATION IN 2D VIDEOGAMES

Javier Montalvo Rodrigo
Advisor: Álvaro García Martín
Lecturer: Jose María Martínez Sánchez

JUNE 2021

SEMANTIC SEGMENTATION IN 2D VIDEOGAMES

Javier Montalvo Rodrigo
Advisor: Álvaro García Martín
Lecturer: Jose María Martínez Sánchez

Dpto. Tecnología Electrónica y de las Comunicaciones
Escuela Politécnica Superior
Universidad Autónoma de Madrid
JUNE 2021

This work has been partially supported by the Ministerio de Economía, Industria y Competividad of the Spanish Government under the project TEC2017-88169-R MobiNetVideo (2018-2020): Visual Analysis for Practical Deployment of Cooperative Mobile Camera Networks.

Resumen

Este Trabajo Fin de Master se centra en utilizar segmentación semántica, una técnica de visión por ordenador, para mejorar el rendimiento de modelos entrenados mediante aprendizaje profundo por refuerzo, y en concreto, el rendimiento sobre el juego Super Mario Bros original.

Aunque una persona es capaz de jugar una partida a un juego como Super Mario Bros, y detectar rápidamente cual es el personaje con el que juega, qué es un enemigo o qué es un obstáculo, este no es el caso para una red neuronal, puesto que requieren de un cierto entrenamiento para poder entender lo que se muestra en la pantalla. Mediante segmentación semántica, podemos simplificar los fotogramas del juego a la información verdaderamente relevante para jugarlo: la posición de las cosas en la pantalla, y la clase a la que pertenecen.

En este trabajo se ha desarrollado un generador de datasets sintéticos para segmentación semántica que simula fotogramas del videojuego Super Mario Bros. Este dataset ha sido utilizado para entrenar modelos de segmentación semántica basados en aprendizaje profundo, que posteriormente han sido incorporados a un algoritmo de aprendizaje profundo por refuerzo de distintas formas para intentar mejorar el rendimiento del mismo.

Se ha comprobado que en efecto, aplicar segmentación semántica como un método de procesado de las imágenes que utiliza como entrada un algoritmo de aprendizaje por refuerzo puede ayudar a entrenar de forma más eficiente y general a este algoritmo. Estos resultados sugieren que puede haber otras técnicas de visión por ordenador, como detección o seguimiento de objetos, que también puedan resultar eficaces, y podría ser interesante estudiarlas en un futuro.

Palabras clave

Segmentación Semántica, Datos sintéticos, Aprendizaje por refuerzo, Visión por Ordenador

Abstract

This Master Thesis focuses on applying semantic segmentation, a computer vision technique, with the objective of improving the performance of deep-learning reinforcement models, and in particular, the performance over the original Super Mario Bros videogame.

While humans can play a stage from a videogame like Super Mario Bros, and quickly identify from the elements in the screen what object is the character they are playing with, what are enemies and what elements are obstacles, this is not the case for neural networks, as they require a certain training to understand what is displayed in the screen. Using semantic segmentation, we can heavily simplify the frames from the videogame, and reduce visual information of elements in the screen to class and location, which is the most relevant information required to complete the game.

In this work, a synthetic dataset generator that simulates frames from the Super Mario Bros videogame has been developed. This dataset has been used to train semantic segmentation deep-learning models which have been incorporated to a deep reinforcement learning algorithm with the objective of improving the performance of it.

We have found that applying semantic segmentation as a frame processing method can actually help reinforcement learning models to train more efficiently and with better generalization. These results also suggest that there could be other computer vision techniques, like object detection or tracking, that could be found useful to help with the training of reinforcement learning algorithms, and they could be an interesting topic for future research.

Keywords

Semantic Segmentation, Synthetic Data, Reinforcement Learning, Computer Vision

Acknowledgements

I would like to thank my tutor, Álvaro for all the provided help and advice, and for allowing me to explore this theme as my master thesis. I would also like to thank my partner Jess and my parents Inma and Luis for all the support given through the course.

Thank you also to the Universidad Autónoma de Madrid for funding this master thesis.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The motivation for this work is the interest I have taken in reinforcement learning during this master course, and I would like to explore ways of combining some of the computer vision knowledge acquired during the master with reinforcement learning both to acquire more expertise with the computer vision techniques and with reinforcement learning algorithms.

1.2 Objectives

The objectives of this master thesis are improving the learning capabilities of reinforcement learning models using semantic segmentation as a post processing method for the images the reinforcement learning model will be feed with. In this work, many relevant topics have been worked on, like synthetic data generation, semantic segmentation and reinforcement learning algorithms.

Three different main objectives were set, with each focusing on a specific topic:

1. Dataset generation, where a synthetic dataset was created to emulate real videogame frames.

2. Semantic Segmentation module development, different models were tested in order to choose the best one for our purpose.

3. Reinforcement Learning model, different approaches were tested to incorporate the segmentation module in the reinforcement learning pipeline.

1.3 Report structure

This report has the following chapters:

- chapter 1 Introduction.
- chapter 2 Related work.
- chapter 3 Design and development.

- chapter 4 Evaluation.
- chapter 5 Conclusions and future work.

Chapter 2

Related work

On this chapter, an introduction on different concepts used in this work is presented. These concepts are from various fields in computer vision and AI, and include a brief introduction on classical approaches to solving these problems as well as information about more recent, state-of-the-art deep learning methods used in these fields.

2.1 Semantic segmentation

Semantic segmentation is a classification process in which instead of classifying an image at a per image level, the classification is performed at a pixel level, so each pixel has a class label. Figure 2.1 represents an example of semantic segmentation.

Figure 2.1: Semantic Segmentation Example.

This process has multiple applications, from autonomous drive, to video surveillance, aerial imaging, scene understanding or medical purpose (among others), and has evolved considerably in later years.

Before deep-learning, semantic segmentation was performed using hand crafted algorithms, like the Split and Merge algorithm [8] or using Conditional Random Fields (CRFs) [9].

With deep learning, the development pipeline for semantic segmentation was greatly simplified and also deep learning models obtained better segmentation quality than classic engineered approaches.

2.1.1 Semantic Segmentation with Deep Learning

One of the first and simplest deep learning architectures for semantic segmentation was using Fully Convolutional Networks (FCN) [1], where the authors proposed using a first set of convolutions to downsample the image input (encoder) and then upsample the encoded output either using transpose convolutions or bilinear interpolation.

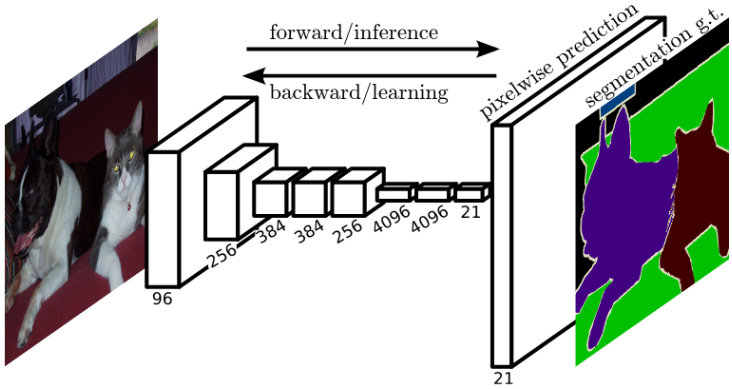


Figure 2.2: Fully convectional networks for semantic segmentation [1].

This method also had some drawbacks, like checkerboard artifacts generated at the upsampling by the transpose convolutions and poor resolution at image borders due to the encoding process.

To solve these drawbacks, different models were proposed and proved to be effective, with the main trends being the addition of skip connections, like U-Net [10], which provides multi-scale information to deeper layers of the model, or models using multi-scale blocks like PSPNet, or Pyramid Scene Parsing Network [2], where a *pyramid parsing module* (which has convolutions with different sizes and hence different receptive fields) is used to obtain different sub-region representations, and then an upsampling and layer concatenation while also using skip connections is used to form the final feature representation.

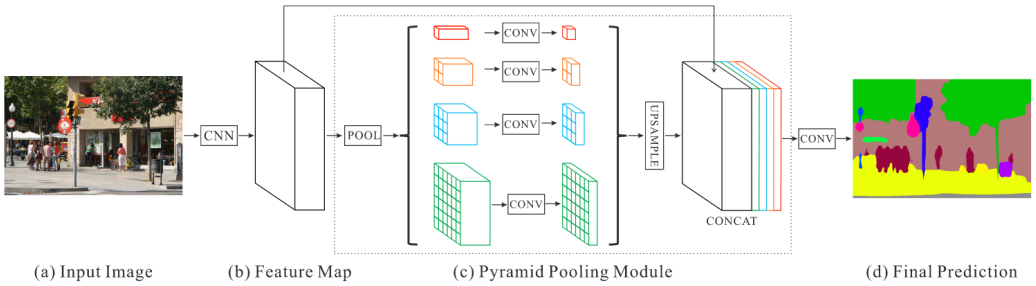


Figure 2.3: Pyramid Scene Parsing Network architecture[2].

The trend with these models is using as much multi-scale information so the models have indeed a bigger receptive field and are able to retain better the contextual information from the original frame.

Another approach to retaining as many contextual information as possible is using atrous or dilated convolutions [11]. These convolutions have the particularity that can control the resolution at witch responses are computed at the CNN without requiring extra trainable parameters.

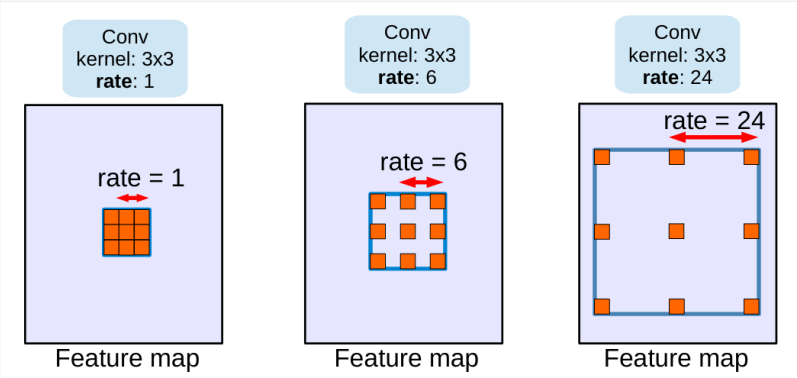


Figure 2.4: Dilated convolution example for different dilatation rates [3].

DeepLab [12] used atrous convolution in the spatial pyramid pooling module, which they named Atrous Spatial Pyramid Pooling (ASPP), although the upsampling was then performed using fully connected Conditional Random Fields for boundary refinement.

In this work, the core segmentation algorithm used was DeepLabV3. DeepLabV3 [3] is an evolution from the original DeepLab which makes use of atrous convolution in both feature extraction and spatial pyramid pooling, additionally, and due to the number of valid filter weights in the ASPP being reduced as the sampling rate increases, DeepLabV3 also applies global average pooling at the last feature map and feed the resulting image-level features to a 1x1 convolution to the ASPP to increase the amount of valid filters. Then, all the layer outputs are concatenated and logit outputs are obtained. These logits are then upsampled to the desired resolution.

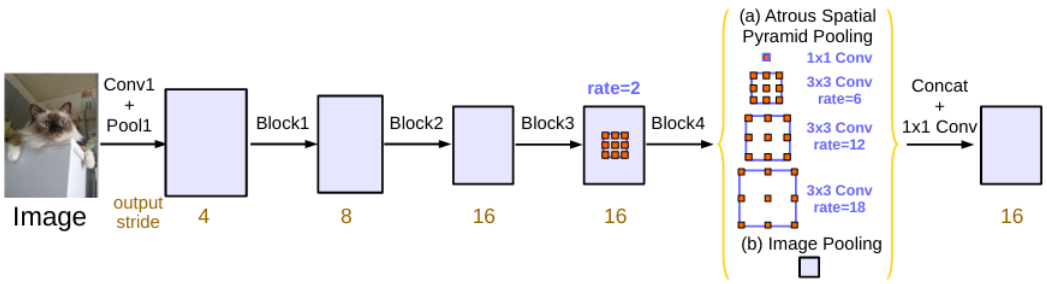


Figure 2.5: Parallel modules with atrous convolution (ASPP), augmented with image-level features[3].

More recent state of the art semantic segmentation algorithms also rely on multi-scale, but with some additional modifications. For example, [13] adds a hierarchy to attention modules to improve inference performance, or [14] which improves the DeepLabV3 module by using spatial pyramid pooling for context information and an

encoder decoder-module with atrous convolutions is used for precise boundary segmentation.

2.1.2 Using synthetic data to train semantic segmentation models

One of the main drawbacks of semantic segmentation is that the ground truth annotation process is labor intensive and tedious, as it requires labeling the images at a pixel level, and therefore the generation of a dataset can have a high cost, both in human resources and time. If instead of manually labeling images, we can generate these images synthetically, we can also generate the semantic segmentation ground truth at the same time, with perfect accuracy, faster speed than manual annotation and also can generate much larger datasets as you could generate as many images as you require.

There are different works that take this approach, with the most common synthetic data for image segmentation usually resembles data from the Cityscapes [15] dataset, which contains images from different german cities.

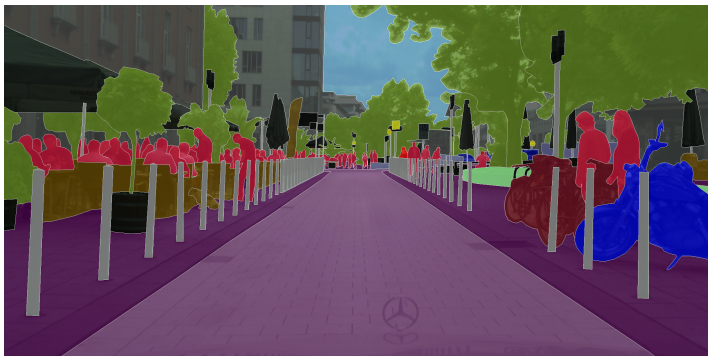


Figure 2.6: Cityscapes image example.

Some examples of synthetic datasets are Synscapes [4], Synthia [16] or GTA V [17], which was generated using a videogame of the same name. It is also common to process images from the dataset to make them look more realistic, as done in [18], where a network is used to make synthetic frames look more realistic.

The main advantage is that it allows to train a model using both real images datasets and virtual images datasets, and have been proven to improve semantic segmentation accuracy in some instances when used in conjunction in comparison to just using real images [16]

Figure 2.7: Synthetic frames from different datasets. From left to right: Synscapes, GTA and Synthia [4].

In this work, a synthetic dataset for semantic segmentation is generated containing frames similar to those that can be seen while playing the game Super Mario Bros.

2.2 Reinforcement Learning

Reinforcement Learning is one of the three machine learning paradigms (the other ones being Supervised Learning and Unsupervised Learning). The main difference with the other two paradigms is that in reinforcement learning, instead of learning from data, usually we have an *agent* that can perform different actions to interacts with an *environment* and whose objective is to maximize the *reward* obtained after performing these actions. Using trial and error, the agent learns how to maximize that reward by performing the best action on each situation.

We can think of this reward as a *score* the agent receives after it performs an action, and this reward function can be adapted to what we expect from the agent. For example, if in Super Mario we want Mario to get all coins in the level, we can set it so when it gets a coin he receives a big reward. If by contrast, we want Mario not to stand still, we can give him a negative reward every second, so the more time passes without completion of the level, the lower it's reward will be, this incentivises the agent to be constantly moving.

The agent has to know which actions can be performed inside the environment, and has to learn how to maximize the reward using these actions. This makes reinforcement learning limited in it's current application, as the agent requires a well defined environment, a finite amount of possible actions, and a way to calculate the reward after each action. For example, if we wanted to train a *car* agent in the real world to maximize a *traveled distance* reward, the results could be catastrophic. As the agent will learn by trial and error, it will probably end up crashed against an obstacle, and this would be a risk and not be cost effective.

It is usual therefore to use simulations in order to train agents and test algorithms, as you can easily define all possibilities of the environment, how the agent can interact with the environment and define reward functions based on the environment and agent status after each action.

Many games generally fit all these requirements, they are often used for test and evaluation of reinforcement learning algorithms, like for example the Backgammon, was used in the early 1990 to train some algorithms like TD-Gammon [19], which attained a level of play better than experts of the game by self playing against himself, and it also proved some unorthodox plays that had been previously ruled out by the backgammon community (as they were thought to be erroneous) to be successful. As a curiosity, this algorithm used a multilayer perceptron network to learn how to play the game.

More recently, reinforcement learning models have bested professional top players in other different games, like Go [20], or Dota 2 videogame [21].

In this work, the videogame Super Mario Bros from the Nintendo Entertainment System was used, with the Double Deep Q-Learning algorithm and the OpenAI gym framework.

2.2.1 Q-Learning

Q-Learning is a model-free reinforcement learning algorithm whose purpose is learning the value of a given action at a particular state. For Markov decision processes, Q-

learning can find an optimal policy in terms of maximization of the total reward on each step.

The algorithm can therefore be considered a function Q that calculates the quality of any given state-action combination:

$$Q : S \times A \to \mathbb{R}$$

First, the Q function is initialized to an arbitrary value, and then at each time step t the agent chooses an action a_t, gets a reward r_t, enters a new state s_{t+1} and updates the Q values.

In reality, the algorithm focuses on optimization using value iteration update and following the Bellman Equation, for this purpose, weights are applied to the previous information and the new one, so the Q-learning core equation is as follows:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \Big(r_t + \gamma \cdot \max Q(s_{t+1}, a)_a - Q(s_t, a_t)\Big) \tag{2.1}$$

In this equation we have different terms:

- $Q(s_t, a_t)$ Represents the output of the Q function for a given state (s_t) action pair(a_t) at instant t.
- α Is the learning rate $(0 < \alpha \leq 1)$
- γ Is called the discount factor, and weights how relevant we want the expected future information to be in our decision making.
- r_t Is the reward received when moving from the state s_t to s_{t+1}
- $\max Q(s_{t+1}, a)_a$ Is the maximum reward that can be obtained from the state s_{t+1}

In practice, this Q function is represented using a table which stores all values for each state-action pair. This table is also one of the factors that limits the algorithm, as the more states required, the biggest this table has to be and it could end up being unbearably heavy to process.

Every training epoch in Q-Learning is called an episode, and each episode ends when the current state the agent is at is *final state*, like for example, death or level completion (in a videogame environment).

2.2.2 Deep Q-Learning

Deep Q-Learning is a variation of the Q-Learning algorithm on which the Q function parameters are substituted by a neural network that estimates the values for each state-action pair. This network is called Q-network, and if we denote its parameters as θ, we can rewrite the equation 2.1 as:

$$Q(s_t, a_t; \theta_t) = r_t + \gamma \cdot \max Q(s_{t+1}, a_{t+1}; \theta_{t+1})_{a_{t+1}} \tag{2.2}$$

This network is then trained by minimising the gradient of the loss function $L_i(\theta_i)$ that changes at each iteration i.

$$L_i(\theta_i) = \mathbb{E}\big[(r_t + \gamma \cdot \max Q(s_{t+1}, a_{t+1}; \theta_{t+1})_{a_{t+1}} - Q(s, a; \theta_i))^2\big] \tag{2.3}$$

For the training, a procedure called *experience replay* [22] is used. With this method, the agent "*experiences*" at each time step, e_t are stored into a fixed size dataset pooled over many episodes, which is called *replay memory*, and it's also updated during the training process. Each *experience* stores the information required in the Equation 2.1, $et = (st, at, rt, st + 1)$.

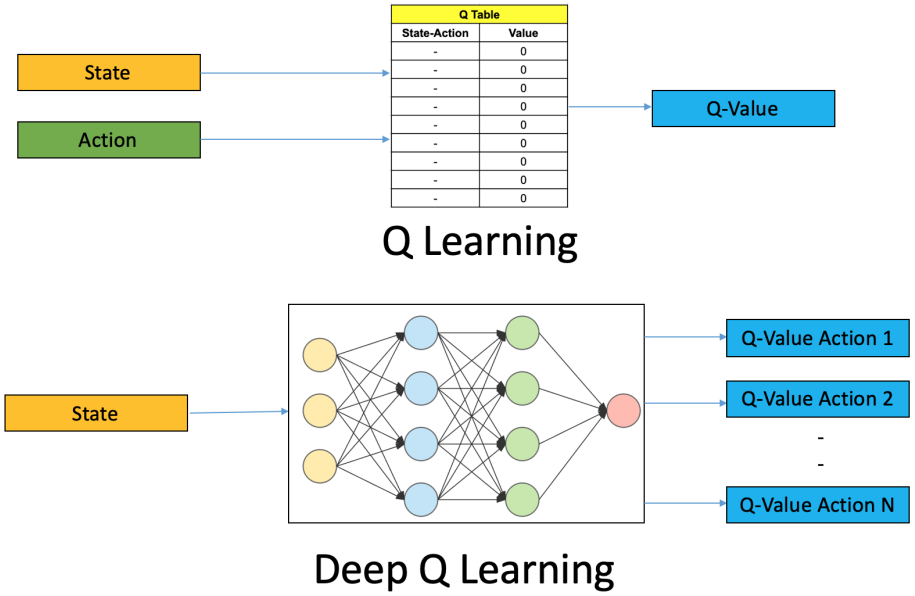


Figure 2.8: Conceptual comparison between Q and Deep Q Learning [5].

In a simplified manner we could say that with Deep Q-Learning, the table from Q-Learning is substituted with a network that acts as the Q-function, as shown in Figure 2.8.

2.2.3 Double Q-Learning and Double Deep Q-Learning

One of the main drawbacks of Q-Learning is that it often overestimates the value of some actions at given states. In 2010, Double Q-Learning [23] was proposed to solve this overestimation issue.

It was proven that the estimator used in Q-Learning was biased, and the proposed solution was to utilize two sets of Q-functions instead of one, with one Q-function used for action selection and the other one for action evaluation, and periodically update the parameters of the evaluation function to match the selection ones.

In 2015, an alternative version with neural networks, Double Deep Q-Learning [6] was proposed, and once again these two Q-functions were substituted by neural networks, and instead of periodically updating parameters, the weights of the selection network are copied to the evaluation network after a given number of episodes.

Double Deep Q-Learning has been proven to be substantially better than Deep Q-Learning in applications relevant to the work here presented, like playing different Atari videogames. Figure 2.9 represents a comparison between action estimation value evolution and score evolution during training for Deep Q-Learning and Double Deep Q-Learning on two Atari games, where we can clearly see how Deep Q-Learning tends to over estimate action values, and this makes the algorithm stall or even decrease

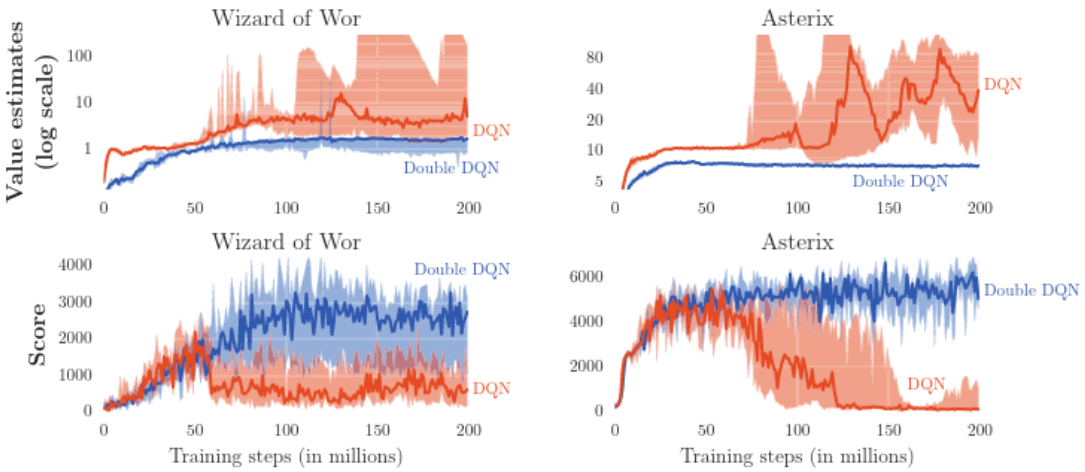


Figure 2.9: Value estimates and Scores comparison between Deep Q and Double Deep Q-Learning in Atari videogames [6].

performance, while Double Deep Q-Learning doesn't over-estimate actions and is able to keep improving for longer time. The figure represents with the darker lines the median of six different executions of the algorithm with different seeds, and the shade represent the percentile 90 and 10 of the extreme values across these runs. More information about these graphs and results on other Atari videogame can be found on the original paper [6].

In fact, when performing similar tests over the Super Mario Bros NES videogame, we can see higher average reward at the same number of epochs with Double Deep Q-Learning, than with just Deep Q-Learning, as shown in figure 2.10. In this figure, reward averages were calculated using rolling windows of 250 samples. Rolling window averages calculate the average on the last N samples, in this case 250, to obtain a trend for the results (as they can have high inter-episode variability).

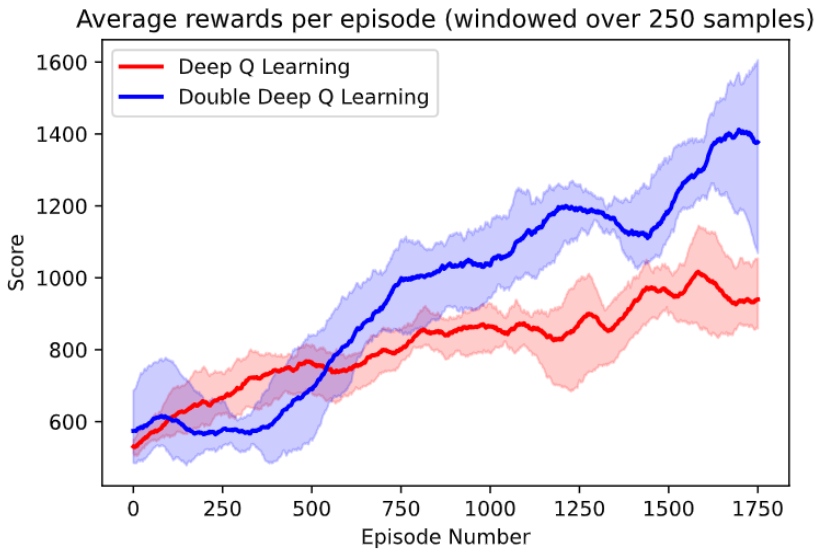


Figure 2.10: Average rewards training in Super Mario Bros using Deep and Double Deep Q-Learning.

In Figure 2.10 the darker line represents the mean of three executions after applying a rolling mean with a window of 250 (the darker line), and the shaded region is bounded by the max values and the min values of these rolling means at each episode.

We can see how, while Deep Q-Learning might perform better in early epochs, Double Deep Q-Learning is able to obtain higher rewards soon after epoch 500, which is when the evaluation network weights are first updated.

2.2.4 OpenAI Gym

OpenAI Gym [24] is an open source toolkit for reinforcement learning research which includes a growing collection of benchmarks and environments, as well as a website where people can compare algorithms and scores.

Provides abstraction for different environments, but the agent has to be implemented by the user. Has an emphasis on sample complexity, not just final performance. *Sample complexity* measures the amount of time the agent required to learn how to play the game.

At the time it was published, it included many different environments by default, like Atari games, some board games, and other small videogames, but it is also prepared for 2D and 3D robot simulation. Currently there are different extra packages available, either officially or implemented by other users, which allow to use other emulators like NES, Sega Megadrive and others.

Chapter 3

Design and development

The objective of this work is to apply computer vision techniques to help a reinforcement algorithm to learn, in this case, how to play the Super Mario Bros videogame.

For this, a synthetic dataset for semantic generation that simulates Super Mario Bros frames has been created, and semantic segmentation models have been trained using this dataset. This semantic segmentation module is then used during the training process of a reinforcement learning agent that learns how to play the first level of the videogame.

In order to check the impact of this semantic segmentation, four approaches were tested: A baseline (for comparison) with stock RGB frames, an approach using only semantic segmentation as input of the reinforcement learning, another using a blend of the RGB frame and the semantic segmentation, and at last, by depth-stacking RGB with segmentation and feeding it to the network.

This chapter presents a description of each of the parts previously mentioned: the design and implementation of the dataset generator, semantic segmentation and the reinforcement learning agent.

3.1 Dataset

In order to train a network to semantically segmentate videogame frames, a dataset with frames and it's segmentation ground-truth is required. For this master thesis the videogame chosen was Super Mario Bros from the Nintendo Entertainment System, which, as is a game that has been largely modded, one of the first approaches to generate the dataset was editing in game files to replace the sprites with solid colors representing the semantic segmentation for each object.

This approach has two issues. First, the way the sprites are represented in Super Mario is using an image for the details of the image, and then a color filter is applied over the sprites to change their look based on the level. The second challenge is that you require to modify the game to obtain a segmentation, which is more intrusive than just processing the visual output of the game with a semantic segmentation model.

Therefore the alternative chosen was to create synthetic frames that look like Super Mario frames using what are known as tileset images. These images contain all the different appearances for different elements, like world objects, enemies, and Mario itself.

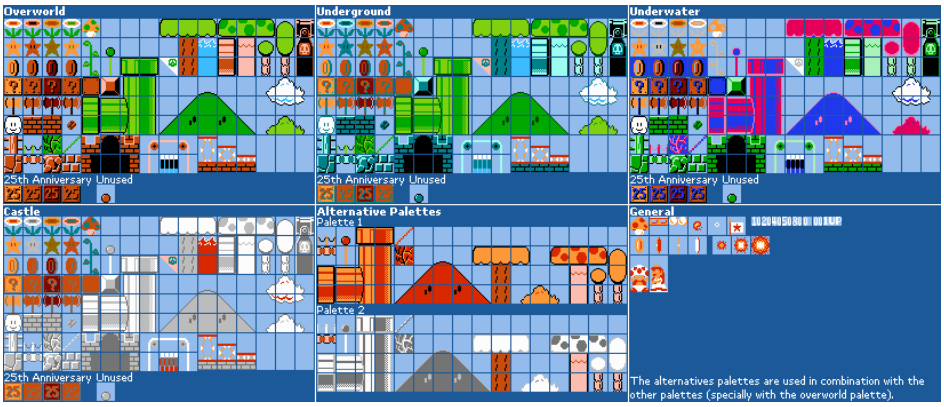


Figure 3.1: Tileset for Super Mario Bros videogame [7].

Using 16 by 16 crops from these tilesets, extracted using a script, the sprites for all elements for the different levels of the game. The main advantage is that as the elements are placed in the same positions across all four levels, using adequate numeration the generator can change the look of the level easily.

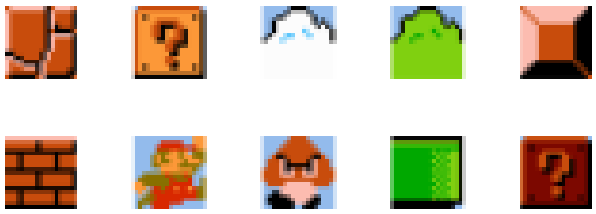


Figure 3.2: Examples of 16 by 16 sprites.

Then, another script was created to generate the synthetic frames. In general, there are common elements between different Super Mario levels, like for example, the floor being two blocks in height, question mark blocks and bricks always at the same given heights, clouds and bushes are always at the same height too. The amount of each of these elements that appear in a synthetic frame is decided using random number generation with sets probabilities, with these probabilities being set based on appearances in the first whole level of the videogame, although some *classes*, like the holes in the floor, whose probability of appearing was under a 10% per frame were given at least a fixed 10% probability in order to keep them representative in the dataset.

Also, as every sprite extracted from the tileset is 16 by 16, and the real frames have a 256x240 size, the frames will be generated using an array of "blocks" of 16 by 16 in which each block has a class label that defines what sprite will be contained at each level.

This allows for easier frame generation. In order to generate these frames, the depth at which sprites is also relevant. Mario moves in front of the bushes for example, but can't be in front of a pipe. Therefore, the frames are generated keeping a hierarchical level in order to place sprites in front of other sprites.

Then, using this grid of labels, the frames are generated based using the tiles from the level type we want to use, see Figure 3.3.

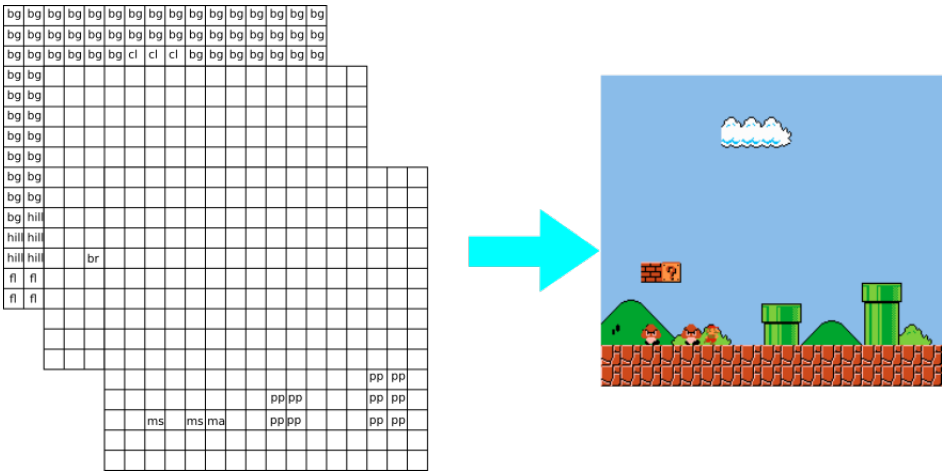
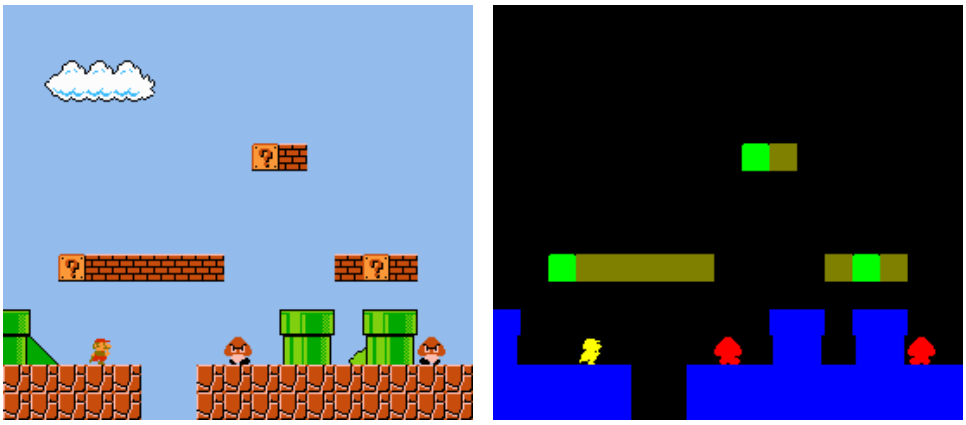


Figure 3.3: Example of labels and their generated frame.

There are other problems, for example, if we just created frames using a grid with cells of size 16 by 16, these frames would have a size of 256x240 pixels, but the separation between blocks would always lay on the same pixels on every frame. This is not the case for the real frames as the image moves laterally on the video-game world, and therefore these block separations are also moving. To fix this, bigger images of 272 by 240 are generated, and then a random cropping is applied in order to ensure these block separations are not always on the same positions.

The main advantage of these synthetic frames is that it generates the frame and its segmentation at the same time using the label information, so no extra computations are required.



(a) Synthetic Frame (b) Segmentation

Figure 3.4: Generation Example.

By changing some generation parameters, images for other levels can easily be generated too. For example, world 8-1 has trees instead of hills and bushes, and world 6-2 has a black background and some different enemies.

While the generated frames look quite convincing, they are not 100% equal to those from the real videogame. For example, in the game there is also some text in the upper side of the frame, where the time left, score, lives left and stage information is displayed. This is not present in the synthetic frames as they are not necessary for

the completion of the videogame (it is actually cropped when fed to the reinforcement learning network).

At last, in order to improve the generation speed of the dataset, some modifications were done in order to be able to generate images using multiprocessing, with significant time savings, and high scalability to multi-core CPUs. In the next table, time comparisons for the generation of a dataset with 5000 frames with different core counts are presented.

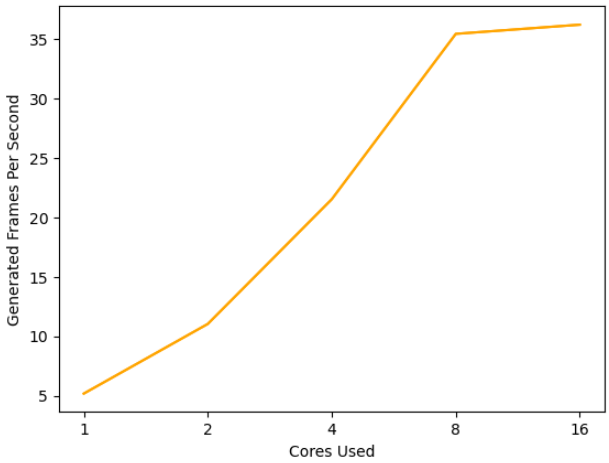


Figure 3.5: Generation throughput depending on core count.

In Figure 3.5 it can be observed that the difference between 8 and 16 cores is substantially lower than compared to previous steps. This happens probably due to the used CPU having only 8 physical cores and 16 threads, and not 16 physical cores.

The main advantage of this parallelization is that, as it can generate close to 7 times the amount of images per second than without parallelization, a much larger dataset can be generated in the same amount of time.

3.2 Semantic Segmentation

3.2.1 Model Selection

For the implementation of the semantic segmentation, the Pytorch [25] tensor library was used, due to the availability of pretrained model for different tasks.

Inside Pytorch library, there is a module called *torchvision* which includes different pretrained models for different tasks. The version 0.8.1 of torchvision was used, which included two different segmentation models, FCN [1] and DeepLabV3 [3], with two ResNet [26] backbones available. The newest version of the torchvision library, 0.9.0, also includes a MobileNetV3 [27] backbone for DeepLabV3 and LR-ASPP[27] model.

For this work, the DeepLabV3 model was used, and test were performed using all available backbones in order to select the one that better fits our purpose. The three available backbones for DeepLabV3 in torchvision are:

- MobileNetV3

- Resnet50
- Resnet101

It is required to perform different tests on them in order to find the one that better suits our purpose. The model requires a low inference time and good capacity of segmentation for different classes. Also, must be able to perform adequately on real frames and not only on the synthetic dataset.

3.2.2 Training methodology

In order to train the model, a fine tuning approach was taken. First the model with ImageNet [28] pretrained weights is downloaded, and the classifier layer is substituted by a new layer of the class DeepLabHead from the torchvision library. This class is a sequential neural network composed by a 3x3 Convolution of 256 input channels and 256 output channels and using padding 1, which is then followed by a 2d batch normalization and a ReLU activation layer, and then another 1x1 convolution with 256 inputs, as many outputs as the number of classes to segment.

No pretrained weights were frozen during training.

Models were trained using a version of the dataset comprised of 20000 images, with 16000 images for training and 4000 images for validation.

Simple data augmentation was used to crop the image to 256 by 240 pixels from the 272 by 240 pixel size from the dataset, in order to minimize the impact of the *grid* used to generate the frames.

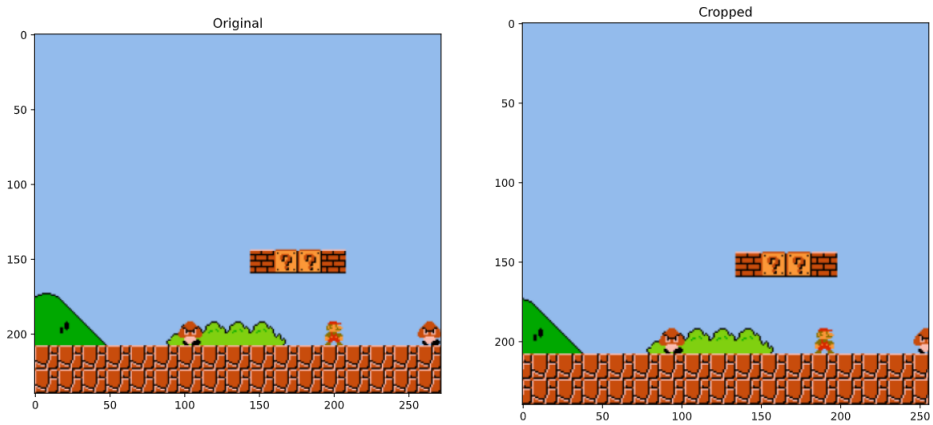


Figure 3.6: Hiding the grid with data augmentation.

In Figure 3.6 we can see how after cropping, the grid has no longer the same position and some objects appear partially occluded, as they would if this was a real frame and the level was scrolling to the right.

All models were trained with and without class weighting. The weights for each class were calculated using their respective proportions in the generated synthetic dataset.

3.2.3 Model Evaluation

One of the main drawbacks of this synthetic approach for the training data is that, while we can measure the performance of the semantic segmentation model over the

validation dataset, this is not the case for real videogame frames as there is no ground truth for them. Therefore, it has to be tested using some real frames and personally check that in fact these frames are segmented properly.

It is also a requirement for this system that the amount of time required for the semantic segmentation processing must be reduced, otherwise training of the reinforcement learning agent would take much more time due to frame processing times.

The backbone for the DeepLabV3 model that performed the best overall in all three tests was ResNet50, and this is the model that is used in the reinforcement learning step of this work.

In the results section, a detailed comparison in real frames performance, validation dataset scores and inference time is included.

3.3 Reinforcement Learning

First, the environment for the Super Mario Bros training was prepared. For this purpose, the python package gym_super_mario_bros, available on *pip*, was used.

It already contains the Super Mario Bros videogame *Rom*, with some additional Roms which include graphical modifications, and allows to easily load levels by changing one parameter. Only the stock Rom was used in this work, as the other Roms alter the videogame which defeats the purpose of using a semantic segmentation model.

The reinforcement learning agent was limited in the amount of actions it could perform to *move right*,*jump* or don't do anything.

Frames were cropped to remove the upper part of the frame (which contains the text with level information and score), and were then downsampled to 84x84 in all approaches (after segmentation, except for the baseline).

All the agents were trained on the first stage of the game (World 1-1).

3.3.1 Baseline Agent

The baseline agent was trained using Double Deep Q-Learning (as in 2.10 I found it to perform better on this game).

In the baseline model, the agent receives RGB frames as inputs.

Figure 3.7: Network Input for the baseline model.

3.3.2 Only Semantic Segmentation

In theory, when segmenting the frames the image is simplified enormously, textures are removed and only shapes and uniform colors remain. A priori, this gives the intuition that the network should then more easily learn how to play the game, as background disappears and only relevant information remains on screen. Also, this *domain shift* from textured frames to a more abstract and simple representation of the level could in theory help with generalization.

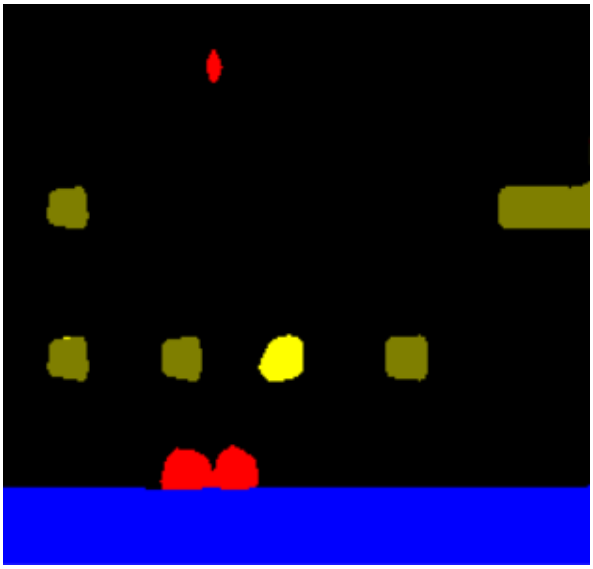


Figure 3.8: Network Input only using segmentation model.

3.3.3 Semantic Segmentation and RGB Frame blend

Another approach taken was to blend the segmented frame and the original one, in order to try to exploit the additional information given by the semantic segmentation while also retaining some original frame information, as it could also help in situations

where semantic segmentation might under perform (for example, variations of enemies, decorations not in the dataset, or particularly challenging frames). The proportion used for the blend was 50% of the original frame and 50% of the segmented frame.

Figure 3.9: Network Input using segmentation and real frame blend.

3.3.4 Depth Stacking Semantic Segmentation and Real Frames

Another approach tested was stacking, in the depth dimension, the semantically segmented frames and the original frames and feed them to the network at the same time, so the network is able to see both semantically segmented frames and real frames at the same time, and therefore has access to all the available information.

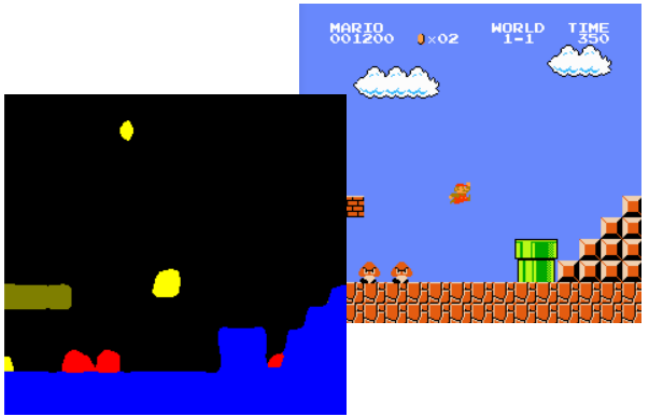


Figure 3.10: Network Input stacking in depth. 6-channel image, displayed as two pictures for visualization.

3.4 System and Workflow

When combining all these previously mentioned models, we obtain an offline trained system where each module can be improved separately.

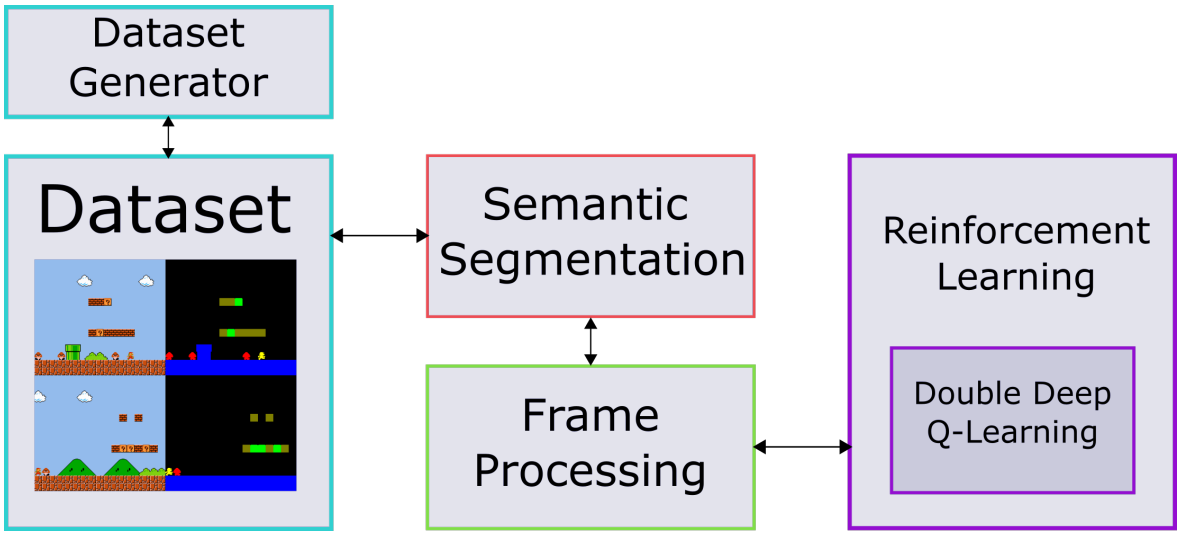


Figure 3.11: System Diagram.

The development procedure took an iterative design: after every modification or addition to the dataset generator, a new dataset was generated and the semantic segmentation model was retrained and tested to ensure adequate performance. After the semantic segmentation model was working successfully, the development of the reinforcement learning module was then started, including the method which encapsulates the different approaches to incorporate the semantic segmentation into the reinforcement learning training.

Chapter 4

Evaluation

In this chapter, the results of the different tests performed during the master thesis are presented. All results have also information about the relevant parameters used for repeatability purposes.

4.1 Semantic segmentation

Different backbones for the DeepLabV3 model were tested to decide which one to use in the reinforcement learning step. In this evaluation, three different factors were measured: inference times, accuracy per class over validation set, and performance over non synthetic frames from the videogame.

All models were trained using a synthetic dataset with 20000 images.

4.1.1 Inference time for each backbone

First, inference times were evaluated, as a large inference time would immediately discard a given backbone due to unsuitability for the task (otherwise, agent training would take too long). The next table summarizes the inference time of each DeeplabV3 backbone available in torchvision.models.segmentation, averaged after 2000 executions. These tests were performed using a Ryzen 3700x CPU with a RTX 2070 Super GPU.

Backbone	Inference Time (ms)
MobileNetV3	0.015
Resnet50	0.018
Resnet101	0.029

Table 4.1: Inference times for different semantic segmentation models.

In table 4.1 we can see how, in terms of inference time, the backbones MobileNetV3 and Resnet50 are the clear winners, as using Resnet101 backbone takes almost twice as much to process a frame than using a MobileNetV3 one. There was no noticeable difference in execution time between models trained with and without class weights (which is to be expected as the network topology and parameter number are the same, only the weights change).

4.1.2 IoU per class

Intersection over Union (IoU) is a metric used in computer vision to quantify the overlap percentage between a target mask and a predicted output. It measures the amount of real pixels correctly labeled (intersection between prediction and ground truth) over the combined total of pixels in the ground truth and the prediction (union).

It is useful to apply this metric at a per class basis, as it gives us information about which class are better segmented and which ones the model has more trouble with.

In order to decide which model to use, their performance on the synthetic dataset was evaluated using IoU. The following results are obtained over the validation dataset, which contains 4000 synthetic frames, after training for 30 epochs.

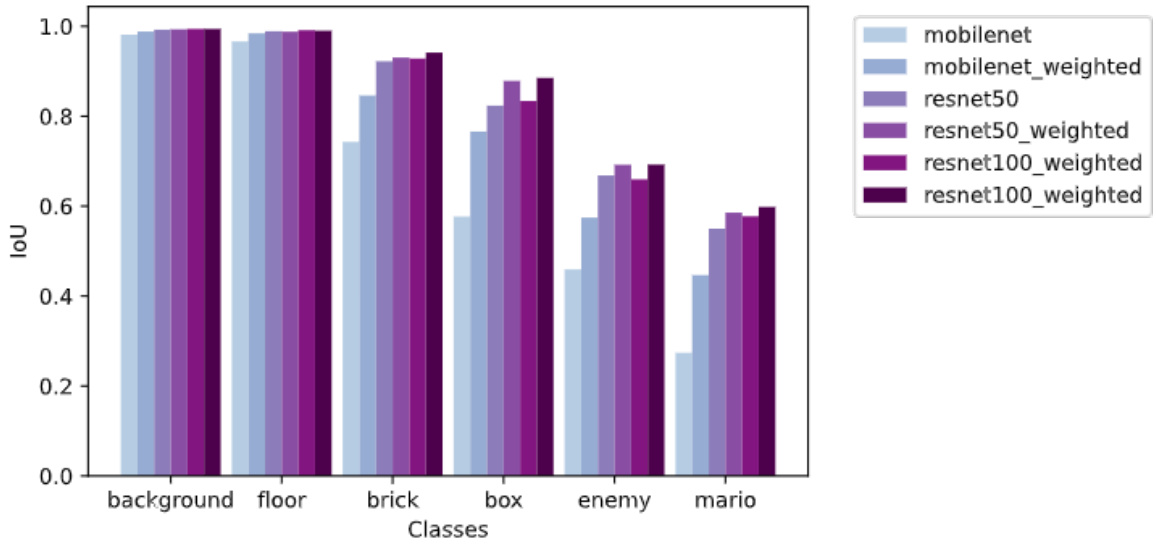


Figure 4.1: IoU for each class per backbone.

We can also see the performance in terms of mean Intersection over Union (mIoU), which gives us an idea of how the segmentation model performs overall, rather than on a per class basis:

Backbone	mIoU
Mobilenet	0.667
Mobilenet Weighted	0.767
ResNet50	0.824
ResNet50 Weighted	0.844
ResNet101	0.831
ResNet101 Weighted	0.850

Table 4.2: Mean IoU for each Backbone.

From Figure 4.1 and 4.2, we can conclude that any model trained with a MobileNetV3 backbone performed worse than the models with ResNet backbones. Using a ResNet-101 backbone we obtain better validation performance than with a ResNet50 (when training on the same conditions) but by a small margin. It is also noticeable that using class weights at training doesn't provide a large performance boost, with the

exception of the MobileNetV3 model. That said, during training, it was noted that the class weights made the model learn more easily smaller classes like Mario or enemies in early epochs.

4.1.3 Performance over real frames

As both training and validation datasets are synthetic, we need to check suitability by evaluating over real frames. From previous tests, we came to the conclusion that the best suited backbones would be either MobileNetV3 and ResNet50, and the training should be performed with class weighting. In the next figure, segmentation results on 4 frames with different complexity for both MobileNetV3 trained models and the model with a ResNet50 backbone trained with class are displayed.

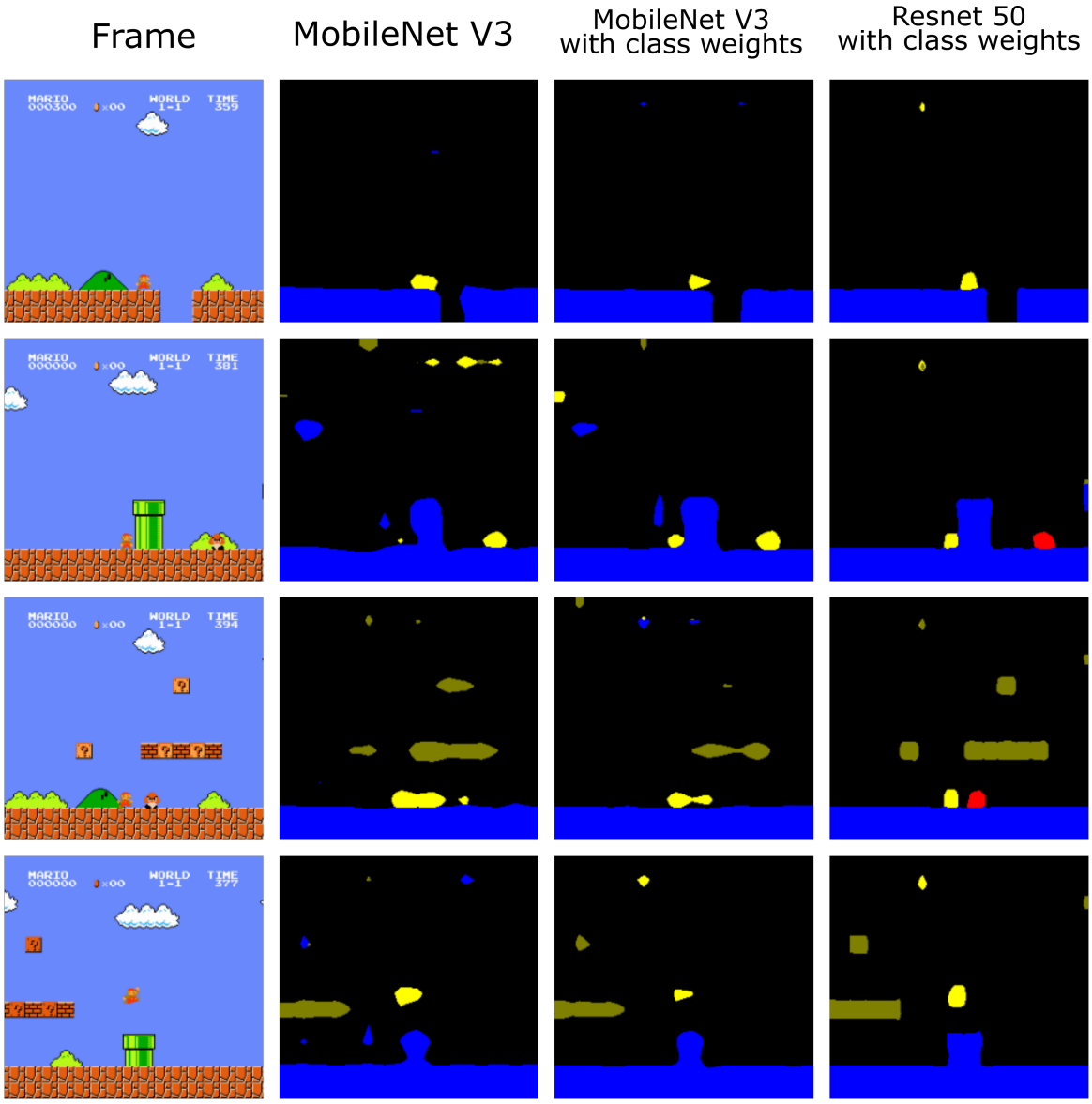


Figure 4.2: Comparison between backbones on different real frames.

In 4.4 we can see the output for three of the tested backbones on four different

frames. Each frame proposes a different challenge: the first one has a hole on the floor, while only has two classes. The second one has three classes but is a simple image, the third one has four classes and in the fourth one Mario is mid air and has a different appearance.

We can observe how both models with the MobileNet backbone hallucinate objects mid air, and both are incapable of correctly classifying the goomba as an enemy, as it is classified as Mario, although in evaluation these models had about 46% and 57% class IoU for enemy.

Combining results from real test images and inference times for the different backbones, the best results for our task are obtained with a ResNet-50 backbone, which has high mIoU, performs well on real frames and is almost as fast as the MobileNet ones.

4.2 Reinforcement Learning Results

As the reinforcement learning algorithm used, Double Deep Q-Learning, has some random exploration factored in, all algorithms were trained multiple times in order to minimize the impact of this randomization on the results.

This random exploration factor is subject to a decay every given number of steps, so the agents were trained for 3000 episodes in order to ensure the final agent performance was not heavily dependent of random exploration.

In this testing, the decay for the exploration rate (the probability the agent has of doing something random instead of what the net suggests) was set to 0.99 every 16 epochs, so the random exploration factor final value was about 0.15 at epoch 3000. Three runs per agent settings were performed per method to ensure repeatability.

4.2.1 Average Rewards

The average reward gives us a measurement of how well the agent is learning on each episode. The next graph illustrates the average rewards for each agent, calculated using rolling windows of 500 elements.

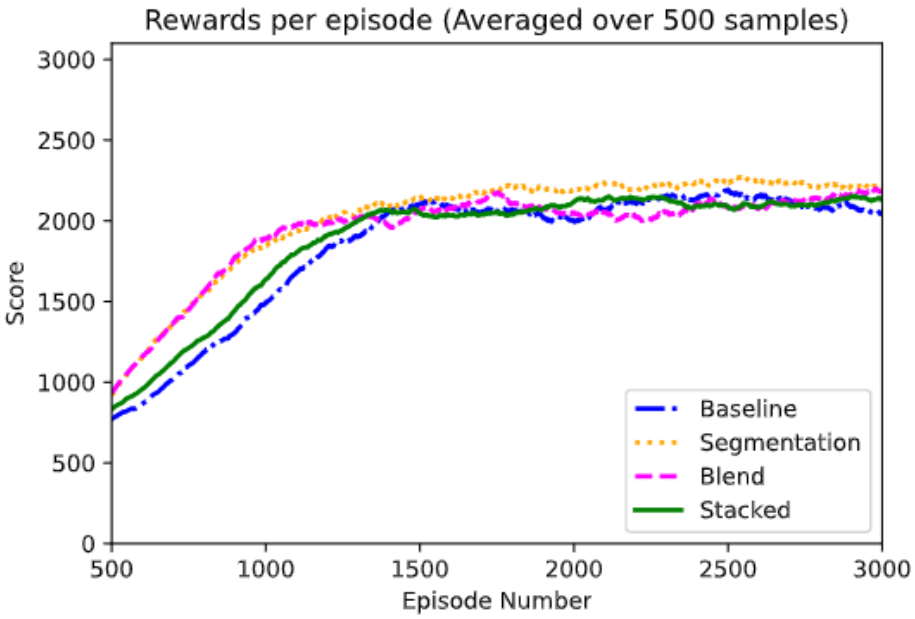


Figure 4.3: Average rewards for different approaches.

As we can see if Figure 4.3, agents trained with semantic segmentation information are able to obtain the baseline level of performance but in less epochs. In the case of the depth-stacked frames the improvement is not as noticeable but for the only segmentation and blend they are able to reach the baseline performance 400 epochs before the baseline.

4.2.2 Win count

That said, mean reward alone doesn't tell us much on it's own. We might have an agent that is not able to finish the level but gets a higher average reward than a agent that completes the level every few runs, but dies very quickly on the other runs.

Therefore, it is interesting to check the win probability at a given window for each trained agent, as it gives us a better notion of the proportion of epochs that each agent can complete the level at each step of the training.

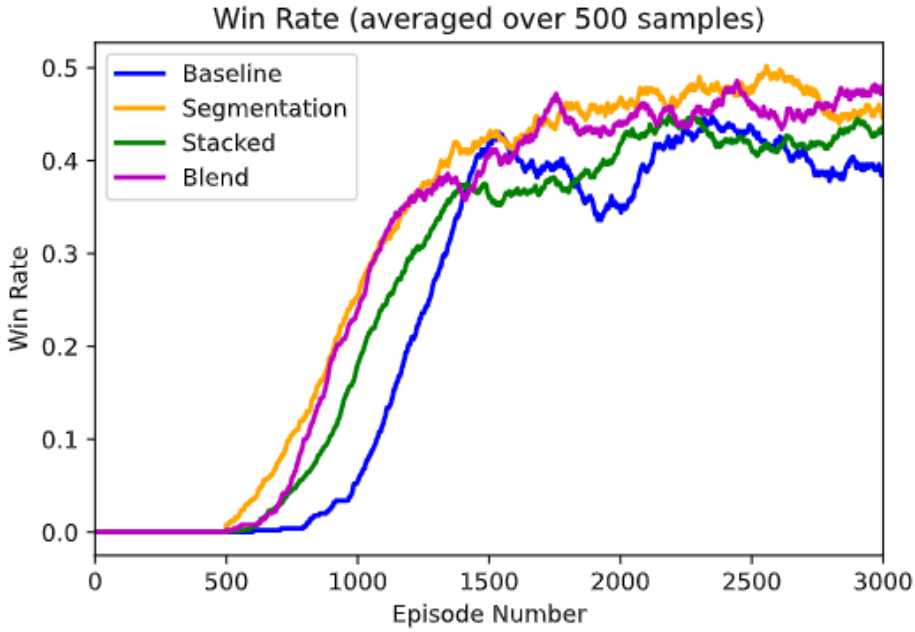


Figure 4.4: Win probability at each time window.

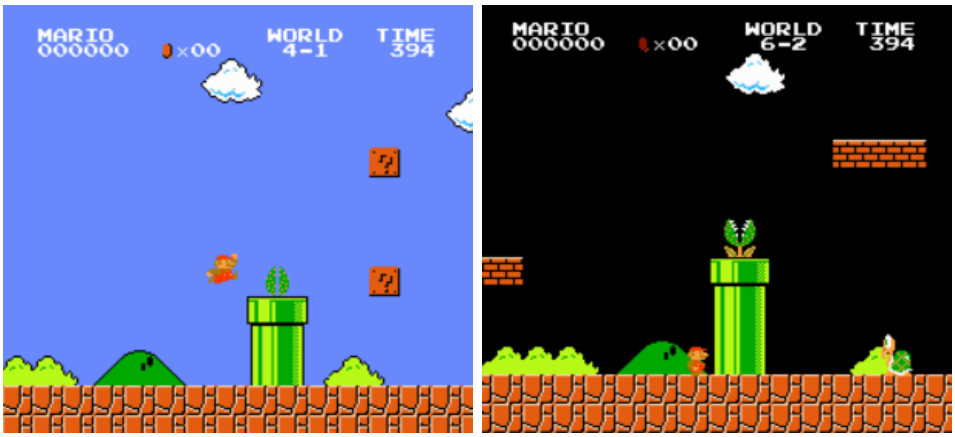
In this case, it is clear how agents that exploit semantic segmentation information are able perform equal or better than baseline, and in less training episodes.

We can therefore conclude that semantic segmentation does actually help training reinforcement learning agents, obtaining the baseline level of performance while requiring less training episodes.

4.2.3 Generalization Testing

The results previously mentioned were obtained over the first stage of the game, where agent trained. One of the biggest issues with reinforcement learning is that often agents over-adjust their behaviour to the specific training environment, and when they play a different level they don't know how to behave. For this purpose, agents trained on world 1-1 were used to try the completion of other similar levels to check performance.

The levels used for this test were 4-1 and 6-2. Level 4-1 has the same visual style as level 1-1 where the agent was trained on, but has some additional enemies the agent has not seen (carnivorous plants) and therefore doesn't know how to react to them, and level 6-2 has also the same visual elements, new enemies but also has a different background (black instead of blue). In figure 4.5 we can see how these levels look and the mentioned carnivorous plant enemy.



(a) Level 4-1 (b) Level 6-1

Figure 4.5: Levels used in the generalization test.

In this test, trained agents were executed for 50 episodes each, in order to see if they were able to progress in levels they had not seen previously, or if they could only progress in the level they were trained at.

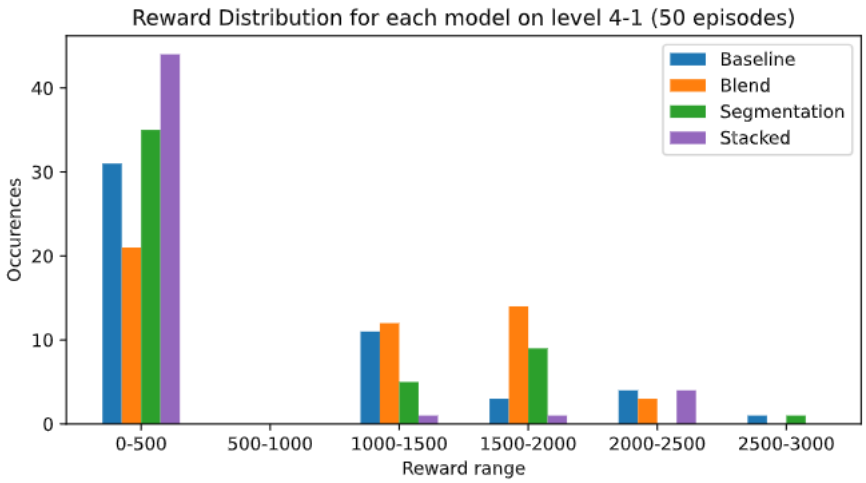


Figure 4.6: Reward distribution for level 4-1.

For the level 4-1, the reward distribution is represented on 4.6. We can see how the agent that is able to consistently get a higher reward the best is agent that gets blended images as input, while the other agent perform worse. We can also see how both the baseline and segmentation outperform the stacked input agent in this test too.

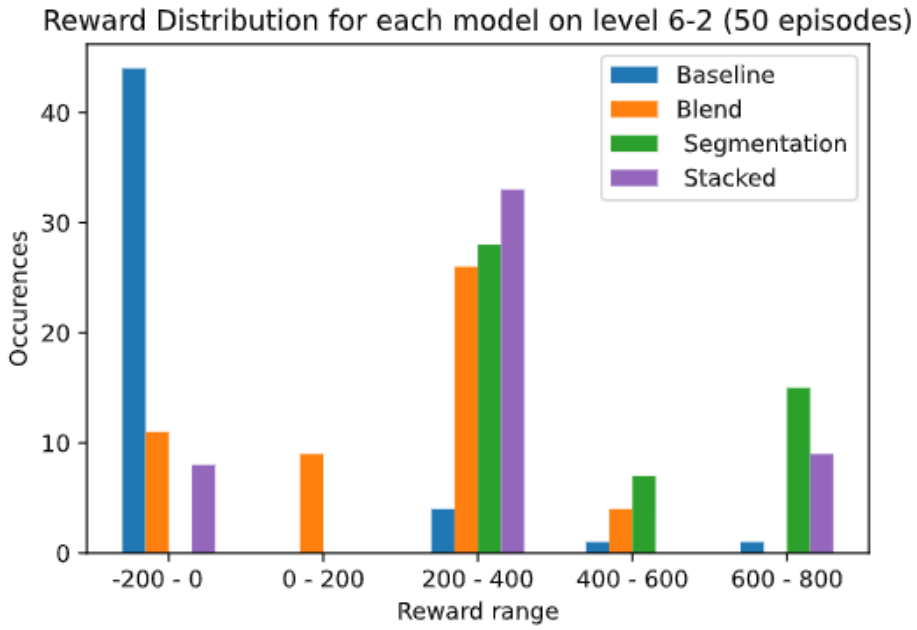


Figure 4.7: Reward distribution for level 6-2.

In Figure 4.7, we can clearly see how the baseline model consistently obtains a negative reward. This happens because he's not able to evade the first obstacle, and loses by running out of time, incurring in a negative reward. Meanwhile, the agents that were trained also using semantic segmentation are able to consistently obtain higher rewards, with the semantic segmentation model being the best performer as is able to avoid this first obstacle on every test run performed, and consistently obtains higher rewards than the other agents.

From these two graphs, we can conclude that in fact, semantic segmentation can help our agents to learn in a more generic way, making them more robust to environment appearance and layout changes, although performance in these other environments is not as good as on the environment they were trained at.

4.2.4 Other interesting data to look at

By generating a scatter plot with all the final rewards from all executions, we can clearly see how there are values where rewards tend to accumulate. These values are, in general, challenging points of the level. In the next figure, this event is represented and some lines have been labeled with the part of the level that where Mario dies.

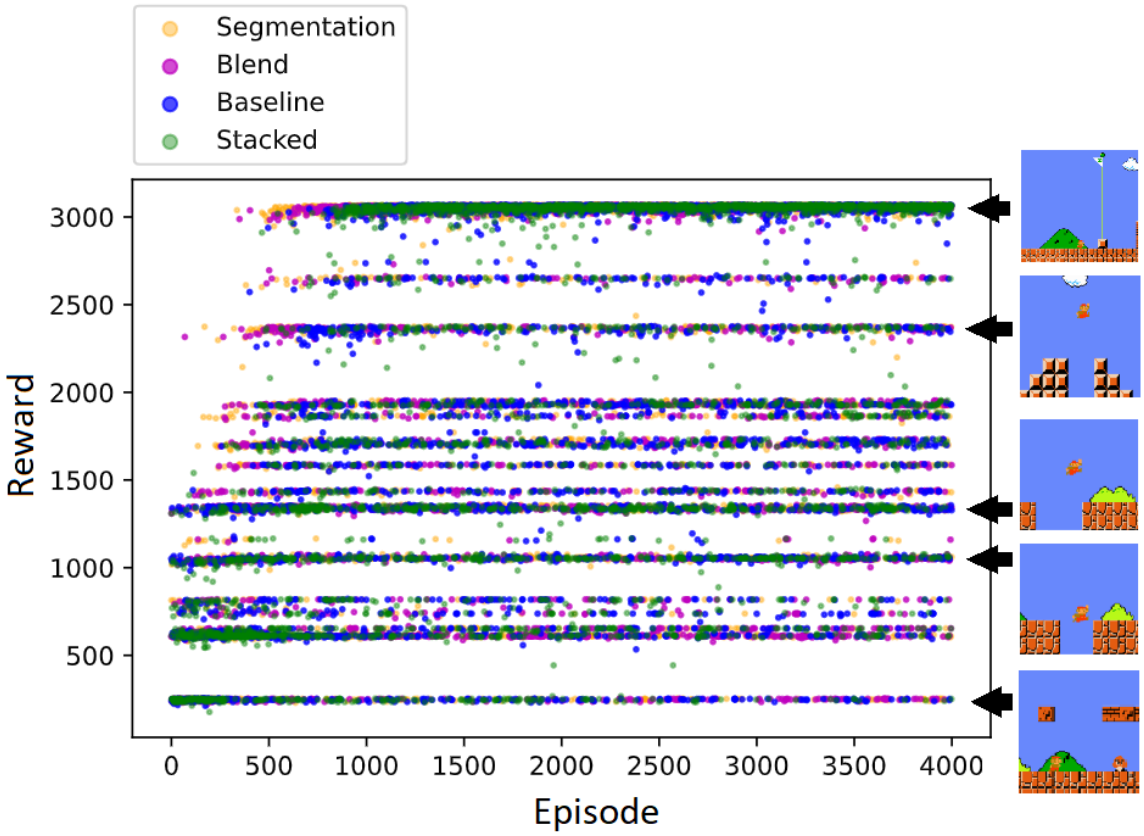
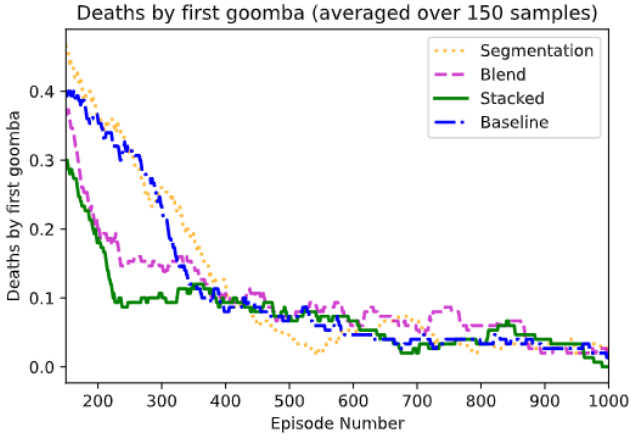


Figure 4.8: All obtained rewards. Most populated horizontal lines are due to challenging parts of the level.

For example, as shown in Figure 4.8, there are many episodes that ended with a reward lower than 500. By observation of the execution, we can find this is the reward obtained when Mario runs into the first enemy of the level and dies.

We can therefore, evaluate agents using this rate of failure with a reward lower than 500, or, with a fancy name, the *Rate of Death by the first Goomba*.



(a) Rate of death by dying to the first enemy.

(b) First goomba of the level.

Figure 4.9: Rate of death by the first *goomba* of the level.

This also gives us significant results. As we can see in Figure 4.9 (a), methods that combine information learn how to evade the first enemy quicker than methods that only use RGB or segmentation information, which suggest that it is easier for the network to understand what action it should do at a given state when more information about the environment is available.

Chapter 5

Conclusions and future work

5.1 Conclusions

We can therefore conclude that using semantic segmentation as a pre-processing for the input of some reinforcement learning algorithms can help, at least on certain environments, to reach the same level of performance in less episodes, while also improving the performance of the agent on situations different to what he has been trained on, making the agent more versatile and robust.

5.2 Future work

In the future there are many possible things to study in order to extend this work:

1. Extend the dataset generation to generate much different levels, like the mushroom worlds or castle words.
2. Add more worlds to the training algorithm so in every episode it trains over different worlds and re-evaluate different agents to see if with this training approach better results can be obtained in generalization testings.
3. Create a new dataset for other different videogames and check if these results also apply for other games.
4. Test this approach with other reinforcement learning algorithms, like Proximal Policy Optimization (PPO) [29].
5. It could be interesting to explore ways of generating or improving the semantic segmentation model using the agent experience while training, instead of improving the synthetic frame generator.
6. Test other computer vision techniques like detection or tracking in conjunction with semantic segmentation and try to further improve our results.

Bibliography

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015. ix, 4, 16

[2] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2881–2890, 2017. ix, 4

[3] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," 2017. ix, 5, 16

[4] M. Wrenninge and J. Unger, "Synscapes: A photorealistic synthetic dataset for street scene parsing," 2018. ix, 6

[5] A. Choudhary, "A hands-on introduction to deep q-learning using openai gym in python." ix, 9

[6] H. van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, Mar. 2016. ix, 9, 10

[7] Spriters-Resource, "Super mario bros level tileset." ix, 14

[8] A. Faruquzzaman, N. R. Paiker, J. Arafat, Z. Karim, and M. A. Ali, "Object segmentation based on split and merge algorithm," in *TENCON 2008-2008 IEEE Region 10 Conference*, pp. 1–4, IEEE, 2008. 3

[9] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001. 3

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015. 4

[11] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 2016. 5

[12] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017. 5

[13] A. Tao, K. Sapra, and B. Catanzaro, "Hierarchical multi-scale attention for semantic segmentation," 2020. 5

[14] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," 2018. 5

[15] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, "The cityscapes dataset for semantic urban scene understanding," 2016. 6

[16] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243, 2016. 6

[17] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*, pp. 102–118, Springer, 2016. 6

[18] Y. Chen, W. Li, X. Chen, and L. V. Gool, "Learning semantic segmentation from synthetic data: A geometrically guided input-output adaptation approach," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019. 6

[19] G. Tesauro, "Temporal difference learning and td-gammon," *Communications of the ACM*, vol. 38, no. 3, pp. 58–68, 1995. 7

[20] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017. 7

[21] OpenAI, :, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray, C. Olsson, J. Pachocki, M. Petrov, H. P. d. O. Pinto, J. Raiman, T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang, F. Wolski, and S. Zhang, "Dota 2 with large scale deep reinforcement learning," 2019. 7

[22] L.-J. Lin, *Reinforcement learning for robots using neural networks*. Carnegie Mellon University, 1992. 9

[23] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, pp. 2613–2621, 2010. 9

[24] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016. 11

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 8024–8035, Curran Associates, Inc., 2019. 16

[26] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. 16

[27] A. Howard, M. Sandler, G. Chu, L.-C. Chen, B. Chen, M. Tan, W. Wang, Y. Zhu, R. Pang, V. Vasudevan, Q. V. Le, and H. Adam, "Searching for mobilenetv3," 2019. 16

[28] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, Ieee, 2009. 17

[29] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017. 33

Appendix

Appendix A

Glossary

- CNN - Convolutional Neural Network.
- CRF - Conditional Random Fields.
- FCN - Fully Convolutional Networks.
- PSP - Pyramid Scene Parsing.
- ASPP - Atrous Spatial Pyramid Pooling.
- NES - Nintendo Entertainment System.
- CPU - Central Processing Unit.
- RGB - Red Green Blue colorspace.