Automatically Generating Image Segmentation Datasets for Video Games

David Gregory LeBlanc and Greg Lee
Acadia University, Wolfville, Canada

Keywords: Image Segmentation, Computer Vision, Machine Learning, Deep Reinforcement Learning, Video Games.

Abstract: Image segmentation is applied to images fed as input to deep reinforcement learning agents as a way of highlighting key-features and removing non-key features. If a segmented image is of lower resolution than its source, the problem is further simplified. However, the process of creating a dataset for the training of an image segmenting network is long and costly if done manually. This paper proposes a methodology for automatically generating an arbitrarily large image segmentation dataset with a specifiable segmentation resolution. A convolutional neural network trained for image segmentation using this automatically generated dataset had higher accuracy than a network using a manually labelled training set. Furthermore, an image segmenting network trained on a dataset generated in this manner gave superior performance to an autoencoder in reducing dimensionality while preserving key features. The method proposed was tested on Super Mario Bros. for the Nintendo Entertainment System (NES), but the techniques could apply to any image segmentation problem where it is possible to simulate the placement of key objects.

1 INTRODUCTION

The motivation for this work originates from a desire to discover a method of simplifying and generalizing state inputs to deep reinforcement learning agents, especially in video game domains. There are many video games, typically within the same genre, which convey similar information with their graphics, such as how most platforming games have the concept of an enemy object, but utilize different sprites to display enemies. If the state fed to an agent uses a game-independent encoding, then the agent should be capable of producing a more general solution than it would with a game-dependent encoding, such as pixel data.

Super Mario Bros. (Nintendo, 1985) was chosen as a subject for the proposed dataset creation method because it is an NES game: it is one console generation ahead of where state of the art deep reinforcement learning agents can reliably outperform humans (e.g. Atari 2600) while using game-agnostic techniques. Furthermore, Super Mario Bros. is part of the platformer genre, and there are many other well known games that fall within the same category on the NES to which the same segmentation scheme could be applied such as Mega Man (Capcom, 1987), Castlevania (Konami, 1986), and Adventure Island (Hudson Soft, 1986).

The ability to specify a segmentation resolution was another key requirement in this work, as a lower segmentation resolution lowers the memory overhead for a deep reinforcement learning agent (deep-RL). NES games have twice the number of pixels per frame compared to any Atari 2600 game, such as Pong (Atari, 1972), and later console generations increase this gap further. Some problems are only possible to solve with deep reinforcement learning if large batches are used (Baker et al., 2020), and this may include these more sophisticated video game environments. Thus, it is important that a state encoding is capable of lowering the memory requirements so that these batches, as well as the experience replay mechanisms from which they are sampled (Schaul et al., 2016), are computationally affordable.

2 RELATED WORK

There are two branches in the state of the art for state representation in the deep-RL video game domain. Firstly, there are those which outperform humans in modern games such as OpenAI Five in Dota 2 (Berner et al., 2019) which utilize a great deal of game-specific information in their state representation. For instance, OpenAI Five utilizes roughly 16,000 inputs to their agent, many of which are meaningless outside of the game of Dota 2 (ex. “is Roshan definitely dead?”, “time since seen enemy courier...”).

LeBlanc, D. and Lee, G.
Automatically Generating Image Segmentation Datasets for Video Games.
DOI: 10.5220/0011693800003393
In *Proceedings of the 15th International Conference on Agents and Artificial Intelligence (ICAART 2023) - Volume 3*, pages 509-516
ISBN: 978-989-758-623-1; ISSN: 2184-433X

In contrast, Agent57 (Badia et al., 2020) utilizes very little game specific information in its state representation, and is capable of outperforming humans in 57 different Atari 2600 games (a much older domain compared to Dota 2). Its state representation consists only of pixel-data from the games being played with some game-agnostic image preprocessing (ex. conversion to greyscale).

Image segmentation has been utilized in the domain of autonomous vehicles to simplify information used to make driving decisions (Papadeas et al., 2021). Popular datasets, used to create these segmentation models, such as the Cityscapes Dataset, are manually annotated, and can take as long as 1.5 hours per sample to create (Cordts et al., 2016).

There are some techniques that have been explored for automating the creation of segmentation datasets, such as in the domain of hand segmentation (Bojja et al., 2018). Lasso-type tools, such as those found in Adobe Photoshop or GIMP, improve the speed at which a human may segment an image.

3 APPROACH

Algorithm 1 details the proposed automatic data generation algorithm. If the segmentation resolution (the size of the grid of segments) is higher than the source resolution, Algorithm 2 is used to create the segmentation grid. The state of each object is described by its position in the scene, and the sprite that it is using. Each sprite should have its own bounding box defined for the purposes of Algorithm 2.

Static objects are those whose states do not change past the initialisation of the scene. Semi-static objects are those with special rules that outline a small number of states they can be in. Dynamic objects are those which can appear anywhere in the scene, so long as they do not overlap with another object. The distinction between semi-static and dynamic objects is made because there are many objects in video games that follow rules that are simple to simulate (semi-static), and other objects that are difficult to simulate accurately (dynamic objects). An example of a dynamic object is a player character, where there are many possible states that the character can be in that depend on the states of other objects in the environment. By contrast, a semi-static object could be an animated but immobile piece of terrain.

Applying this to Super Mario Bros. the main labels are listed below in order of priority, player being the highest priority label and ground being the lowest priority label:

Algorithm 1: The data generation algorithm for the platformer autolabeller.

Labels: A list of possible labels ordered by priority of segmentation
S: The background of the environment being simulated
O: Objects, which are static, semi-static, or dynamic. Each object has an associated label from *Labels* and a number of sprites which it can display.
N_{needed}: Number of samples needed
$N_{current}$: Number of samples created so far
$Grid_{x,y}$: Label located at (x,y) in segmentation grid
C_x, C_y: Camera x and y positions in S
Res_h: Horizontal image resolution
Res_v: Vertical image resolution
Screen: $region(C_x, C_y, C_x + Res_h, C_y + Res_v)$
Initialise S, create all O_{static}, $O_{semistatic}$
Define all valid C_x, C_y for camera
while $N_{current} < N_{needed}$ **do**
    Assign random new valid C_x, C_y
    Destroy all $O_{dynamic}$
    Instantiate a random valid number of $O_{dynamic}$
    Place $O_{dynamic}$ instances randomly within *Screen*, do not allow overlapping
    Randomize sprite used for each $O_{dynamic}$
    Update state of all $O_{semistatic}$
    $Grid \leftarrow update\_grid(Grid)$ {See Algorithm 2}
    Save *Grid*, pixel data of *Screen* as a sample
    $N_{current} \leftarrow N_{current} + 1$
end while

Algorithm 2: The grid update function for the platformer autolabeller.

$CellW \leftarrow Res_h / width(Grid)$
$CellH \leftarrow Res_v / height(Grid)$
for $CellX \leftarrow C_x, CellX < Res_h, CellX \leftarrow CellX + CellW$ **do**
    for $CellY \leftarrow C_y$, $CellY < Resv$, $CellY \leftarrow CellY + CellW$ **do**
        $CellRegion \leftarrow region(CellX, CellY, CellX + CellW, CellY + CellH)$
        Find *Label*, s.t. *Label* is the label with highest priority associated with objects in *CellRegion* according to *Labels*
        $Grid_{CellX,CellY} \leftarrow Label$
    end for
end for

- Player: the object the player controls (Mario, Luigi). This is a dynamic object, and there is always exactly one per generated sample.
- Enemy: any hostile object that can be defeated normally (that is, by jumping on them, or with the use of a power-up). These are dynamic objects, and there are up to two of these in a given sample (There are typically fewer than 2 enemies at a given time in normal gameplay).
- Hazard: objects that will harm the player on contact that cannot be defeated normally (e.g. Bowser, Piranha Plants).
- Ground: objects that may be stood on. These are a mixture of semi-static objects (e.g. moving platforms that follow a set path) and static objects (e.g. bricks).

This labelling scheme was designed in such a way that it could apply to any platformer game, although some may require some additional labels (perhaps for power-ups, friendly projectiles, or interactable objects). Due to the automatic nature of the algorithm, it is relatively straightforward to edit objects' classification. The main source of work for implementing this algorithm lies in importing the relevant resources for an object or scene.

Any region of the screen which does not fall into one of the identified labels (e.g. score indicators or background objects) is given a none label to indicate that there are no key features in that region. The player's location is a component in all decision making in platfomer games, hence its position at the top of the priority list. Enemies and hazards both take priority over the ground labels, as it is generally more advantageous to avoid hazards or enemies than to be cognisant of whether there is terrain in that same region. This is especially relevant in Super Mario Bros. where enemies may be used as a form of terrain if they are jumped on. Note that in Algorithm 1, dynamic objects are placed randomly within a room with random sprites, and this this can produce screenshots that are improbable or impossible to reproduce within the game being simulated. Similarly, it is possible to mix and match resources from multiple games within a single dataset.

For example, the enemy sprites could be a mixture of Super Mario Bros. and Castlevania sprites. The reasoning behind keeping this behaviour is twofold: perfectly simulating the source game slows down the process of creating the simulation, and it may be that the unconventional placement of sprites leads to the generation of models that are more general. For instance, it is very rare in Super Mario Bros. for the player character to be at the very top of the screen, but in a game with more verticality, such as Mega Man, such scenarios are common.

For the experiments in this paper, the automatic labelling algorithm was implemented in GameMaker using assets from Super Mario Bros. The first world, consisting of four levels, and the first level of world 2 were simulated. The automatic labeller could generate samples of the implemented levels at a rate of 165 samples per second. The process of adding a new level to the simulation consisted of importing the level as a background to a new room, labelling all the terrain (ground labels), and adding in any special objects (e.g. moving platforms). This process would take roughly 1 hour per level added. Some elements of the game were not added to the simulation; the GUI, consisting of the white text and flashing coin sprite was only partially simulated. The coin would flash as it would in normal gameplay, but all text elements were left static in the automatically created sets.

In addition to the automatically created datasets, a manually created dataset was also generated. The manual labelling software used was created in Python using Tkinter (Lundh, 1999) specifically for the purposes of this research. The created software features hotkeys to switch between label types and images to increase the speed of labelling. After some practice using the software, an expert user could produce one sample every 15 seconds on average (or 0.07 samples per second). Note that this number is based on the segmentation resolution being 15x15; a higher segmentation resolution would result in a slower labelling speed. The 15x15 segmentation resolution was chosen for both automatically generated and manually generated datasets as when dividing the resolution of Super Mario Bros. this way, each cell of the segmentation grid corresponds to a roughly 16x16 region of the source image (16x14.9 after overscan), and many sprites in Super Mario Bros. are comprised of 16x16 tiles.

The manual dataset and the small automatically generated dataset consist of 3,734 samples each, while the large automatic dataset consists of 1 million samples. Furthermore, the large automatic dataset took approximately 2 hours to generate unsupervised in one session, the small automatic dataset took less than one minute, while the manual dataset took approximately 24 hours to generate by hand, spread across 5 sessions. In addition to the time spent labelling the manual dataset, an extra hour was spent playing the game to generate sufficiently diverse gameplay footage whose frames formed the images to be labelled. This is necessary to prevent state bias in a resultant model.

In theory, it is possible to generate the images for manual labelling at a rate matching the game's framerate (60 frames per second in the case of Super Mario Bros.), but in practice, many frames have to be discarded as the gameplay contains irrelevant images such as game-over screens and menus. In addition, normal gameplay produces a skewed dataset; a skilled player may get the player character into a powered up state, which has its own sprite, and never reach the other player states that may be seen in the game. Conversely, an unskilled player would produce gameplay with few frames in the powered up state.

In addition to the videos whose frames were utilized for the manually labelled dataset, other gameplay videos were recorded for testing purposes. That is, another set of videos were created with frames that did not appear in the datasets to act as a test set. Since the intention for the creation of the test videos is to benchmark models created from both the manually and automatically generated datasets, whose segmenting methodologies are different, no ground truth segmentation is given for frames in the test videos. Due to limitations in the recording environment, the test videos all suffer from some compression artifacts, meaning that there is noise present in the test videos that would not have been seen in the automatic dataset. Since the manual dataset was derived from similar videos, the manual dataset has the advantage of sharing noise characteristics with the test videos.

To test the viability of the automatically created dataset, three segmenting models were trained; one using the full automatic dataset (large automatic segmenting model), another with the manual dataset (manual segmenting model), and finally with one trained on the small automatic dataset (small automatic segmenting model). All used images from their respective datasets as input to predict the segmentation given by the grid in the dataset. They were evaluated using test sets generated from the same dataset they were trained on (although samples from the test set were not given to the agent during the training phase), as well as frames from the test video.

The small automatic and manual segmenting models were subject to 10-fold cross-validation to account for their small datasets. In those cases, 187 samples (5% of the total number of samples) of each dataset were removed prior to the cross-validation process, so that each model could be evaluated on a fixed test set. The models with the highest macro averaged F1-score on their respective test set were selected for further evaluation using the test videos.

In addition to the segmenting networks, an autoencoder was trained on just the image components of the automatic dataset. Unless otherwise stated in the tables, all non I/O layers of the neural networks made use of a ReLU activation function, and all networks were trained using a 1e-4 learning rate with an Adam optimizer. Additionally, early stopping was applied to all training sessions with a patience of 10 epochs and a minimum delta of 1e-4. Mean Squared Error, or MSE, was used as the loss function for all investigated models.

Each segmenting model had its own model architecture and hyperparameters optimized for validation accuracy on its respective datasets using a combination of a Hyperband tuner (Li et al., 2018) and hand-tuning. Table 1 lists the values that were altered during the tuning process. Values that do not appear in the table were not tuned. Table 2, Table 3, and Table 4 give the model architectures for the small automatic / manual segmenting models, automatic segmenting model, and autoencoder, respectively.

As mentioned above: there were some changes to the networks during the model designing process that were not determined by the tuner; the manual and small automatic segmenting models received max pooling and dropout layers to help in overcoming the small size of the source dataset and avoid overfitting. While tuning the automatic segmenting model, hyperparameters were tuned in batches using multiple datasets, with smaller ranges for larger datasets. Smaller datasets were created of sizes ranging from 100,000 samples to the final 1,000,000 sample dataset during this process. All networks utilized the same preprocessing pipeline:

- Crop 8 pixels from each side of the image (overscan) to simulate how Super Mario Bros. would be displayed in an emulator like in Gym Retro (Nichol et al., 2018).
- Convert the image to greyscale to reduce dimensionality; the RGB channels are combined into one greyscale channel.
- Normalize all pixel values to the range [0,1] to reduce absolute distance between similar colors.

When evaluating the segmenting models, each cell in the predicted segmentation grid has its value rounded to the nearest integer (label) and clipped to the range of possible labels, [0,4]. This way, per-class accuracy can be calculated. To evaluate the autoencoder, reconstructions of frames from the test video were reviewed by a human expert familiar with the rules and appearance of Super Mario Bros. All experiments were performed on a PC utilizing 32 GB of RAM, a 3.5 GHz 8 core processor, and an NVIDIA GeForce RTX 3070 GPU with 8GB of VRAM.

Table 1: The parameters that were altered during the tuning process for the automatic and manual segmenting models.

Parameter	Considered Values
# Convolutional layers	2, 3
Filters	32,64
Stride	16, 4, 2, 1
Kernel Size	16x16, 8x8, 3x3, 4x4, 2x2, 1x1
Max pooling	Yes, No
# Dense Layers	1, 2, 3
Dense Units	[32,512], steps of 16
Learning Rate	1e-3, 1e-4, 1e-5
Dropout value	0.0, 0.1

Table 2: Neural network architecture for the manual and small automating segmentation networks.

Layer Type	Details
Input	-
Conv2D	64 filters, stride 4, 16x16 kernel
Max pooling	-
Conv2D	64 filters, stride 2, 8x8 kernel
Max pooling	-
Flatten	-
Dense	512 units
Dropout	0.1 chance
Dense	512 units
Dropout	0.1 chance
Dense	No activation, $n_{outputs}$ units
Reshape	Reshape to 2D for output

Table 3: Neural network architecture for the segmenting model trained on the full automatically generated set.

Layer Type	Details
Input	-
Conv2D	32 filters, stride 4, 4x4 kernel
Conv2D	64 filters, stride 2, 2x2 kernel
Conv2D	64 filters, stride 1, 1x1 kernel
Flatten	-
Dense	128 units
Dense	No activation, $n_{outputs}$ units
Reshape	Reshape to 2D for output

Table 4: Neural network architecture for the autoencoder trained on the automatically generated set.

Layer Type	Details
Input	-
Conv2D	16 filters, stride 4, 16x16 kernel
Conv2D	16 filters, stride 2, 8x8 kernel
Dense	No activation, $n_{outputs}$ units
Conv2D-Transpose	16 filters, stride 2, 8x8 kernel
Conv2D-Transpose	16 filters, stride 4, 16x16 kernel
Conv2D	1 filter, stride 1, 3x3 kernel

4 EXPERIMENTS

4.1 Segmentation Comparison

The manual and the automatic segmenting models gave near perfect accuracy in labelling the ground and non-key features (none label), with 92-99% accuracy across the different models. Given that the two most common labels are ground and none, this result is expected. These two labels are also the two least variable; configurations of ground and empty space do not change for a given place in a level, simulated or not, with the exception of moving platforms, which comprise a small portion of total ground labels.

The largest performance difference between the segmenting models is that the large automatic segmenting model demonstrates much higher accuracy in correctly labelling player, enemy, and hazard objects. The smaller models tend to over-predict ground labels, likely due to the ground labels being the most common label, as well as the smallest non-zero label in the set.

With the information from the confusion matrices, it was calculated that the manual segmenting model achieved a macro averaged F1-score of 0.52 ± 0.00 on its test data (41850 predictions), the automatic segmenting model achieved a macro averaged F1-score of 0.88 on its test set (921600 predictions), and the small automatic segmenting model achieved a macro averaged F1-score of 0.48 ± 0.01 (41850 predictions). By all metrics, the small automatic segmenting model does not perform as well as the manual model, but only slightly in comparison to the difference of both with the large automatic model. In all probability, this is due to how the manually created data is much more tightly correlated than the automatically generated data, as the data is created through gameplay which has more strict rules than the automatic sample generation process. For example, in all samples in the manually generated set, Mario, labelled player, tends to be close to the ground due to the in-game gravity. However, the automatically generated sets had Mario in any position onscreen with equal probability.

The large automatic segmenting model achieves higher accuracy than the manual segmenting model on the dynamic and semi-static objects (hazards, enemies, and the player) even though those same elements were more variable in the automatically generated set than the manually generated set. This suggests that the higher variability may be overcome with a sufficient number of samples. This is of course very feasible given the multiple orders of magnitude in time advantage the automatic approach has over the manual approach.

Most misclassifications by the models are in nearby classes, which perhaps would be remedied by encoding the segmentation grids with a one-hot encoding instead of a unique integer for each label. Even on samples from the test set, like the comparison shown in Figure 2, the large automatic segmenting model produced more reasonable predictions of the encoding. This is despite the test images containing elements that were not simulated in the training data. For example, the numbers in the GUI at the top of the screen are different from what they are anywhere in the training set. It is the case that the GUI region in the test image is erroneously categorised as ground in several cells of the prediction in Figure 2, and this may account for some of the false positives predicted in the ground class.

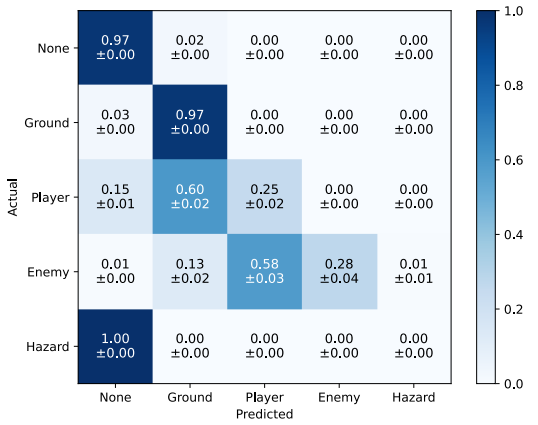
In Figure 2 the differences between the labelling schemes of the datasets can be seen. The manually labelled dataset featured closer-fitting segmentations to most objects, thus the thinner labels for the pipes and player. The predictions for the automatic segmenting model could be made closer fitting by adjusting the bounding boxes of the relevant sprites.

The results from the automatic segmenting model on the test video demonstrate that a model is capable of overcoming the noise introduced by the video compression artifacts. This could be useful in situations where it is not possible to capture noise-free footage.

In summary, the large automatic segmenting model outperformed the manual segmenting model in terms of per-class accuracy and summary F1-score, and neither the imperfect simulation performed to create the automatically generated set nor the test video's compression artifacts prevented the automatic segmenting model from generalizing to footage from the actual game. Much of the full automatic segmenting model's success may in part be due to the larger training set. However, the sample creation rate of the proposed algorithm is higher than that of the manual approach (165 samples per second, compared to 0.07 samples per second, a difference of 4 orders of magnitude), and these differences would be exacerbated by a higher segmentation resolution, as there would be less computation required for each image. That is, the large automatic model's performance is representative of the sample creation rate increase that the automatic approach offers over the manual approach.

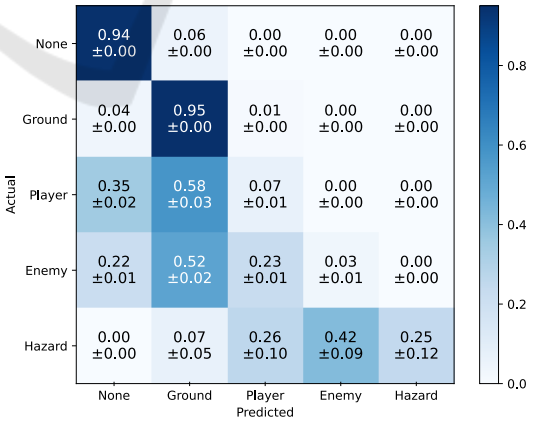
4.2 Dimensionality Reduction and Key Feature Preservation

Figure 3 shows the reconstruction of an image from the test set as created by the autoencoder. Comparing to predicted encodings created by the automatic seg-



(a) The confusion matrix manual model. 187 samples from the original dataset were removed from the training set to form the test set for this confusion matrix.

(b) The confusion matrix for the large automatic model. An additional 4096 samples were generated to form the test set.



(c) The confusion matrix for the small automatic model. Like the manual model, 187 samples from the set were removed to form the test set.

Figure 1: Confusion matrices for the segmenting models.

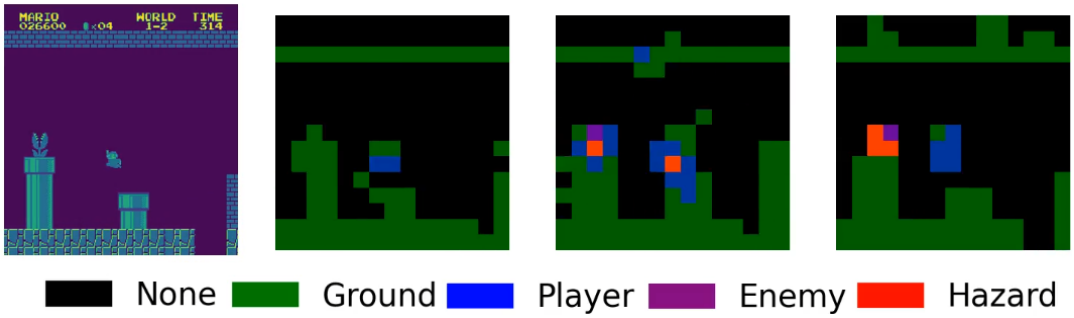


Figure 2: The predicted segmentation for the segmenting models on an image from a test video. From left to right: the image being used as input, the manual model's prediction, the small automatic model's prediction, and the large automatic model's prediction. The bottom source image depicts Mario between two pipes (ground) and a Piranha Plant (hazard).

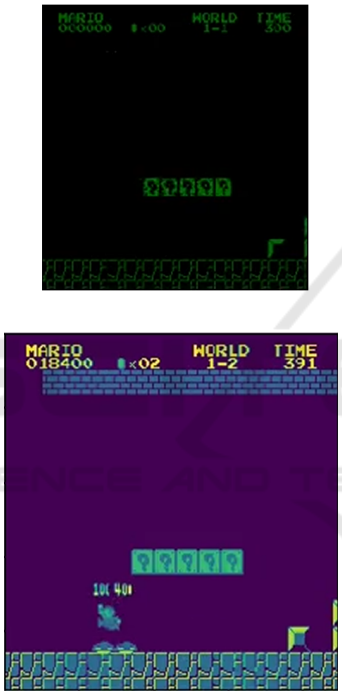


Figure 3: A reconstruction of a screenshot from Super Mario Bros. created by the autoencoder (above) and the source (below).

menting model (Figure 2), a decrease in key feature preservation is shown. For example, in all test frames reviewed (a 2 minute, 60fps test video showing reconstructions alongside the source footage), neither the player character nor any enemies that were present in the source frames could be distinguished in the reconstruction by a human proficient in playing the game. Furthermore, some of the reconstructions did not contain all of the terrain that was present in the source image. In the case of the reconstruction shown in Figure 3, the topmost bricks are completely missing in the reconstruction.

In contrast, when shown a video containing the source frames of the same video alongside predictions made by the automatic segmenting model (Figure 2), the human expert could identify most key labels and their associated objects in both images throughout the video. The automatic segmenting model utilized a 15x15 segmentation resolution, and the autoencoder utilized a 15x15 latent space. That is, given the same space to encode key features from the source frames, the automatic segmenting model preserved more of the key features than the autoencoder as judged by a human expert, to the extent that the autoencoder did not preserve any enemies or player characters.

5 CONCLUSION AND FUTURE WORK

The proposed algorithm for automatically generating an image segmenting dataset is capable of producing a dataset that, when used to train a segmenting network, leads to a more effective network compared to one that is trained on a manually labelled dataset created over a longer period of time.

It is understood that there are some additional costs to using this algorithm over the manual approach:

- Sufficient expertise in the environment is required to craft a sufficiently realistic simulation (e.g. objects need to be classified as static, semi-static, or dynamic with proper behaviours).
- Assets from the simulated environment must be available, or else close replicas need to be produced.

However, the automatic labelling approach offers a number of advantages over the manual method:

- Given the greater rate of sample creation compared to the manual approach, it is easier to re-

generate the dataset with altered parameters such as new labels or new objects.

- Automatic labelling leads to perfect consistency in the labelling process; human error is contained within the parameter setting process.
- One human expert may have control over the dataset's parameters, rather than having a human expert training a number of less experienced workers.
- The automatic labelling approach allows the combination of assets which otherwise would not be seen together, potentially leading to a dataset that could create a more general model.

From the experiment on the autoencoder, with the specified latent space size / segmentation resolution of 15x15, the automatic segmenting model outperformed the autoencoder at maintaining key features. From that result, it is assumed that being able to specify the segmentation resolution of the dataset is a useful tool in creating a model while seeking to optimize the amount of space used to summarize key features.

It may be possible to generate a dataset by treating all game objects as dynamic with the proposed algorithm, or in other words, placing all game assets randomly within a screen with no adherence to game rules. Such an approach was not tested for this paper, as it was assumed that a more realistic dataset should be used to create more accurate segmentation models. However, this may be worthy of additional experimentation.

One of the areas deemed most important in further evaluating the overarching segmentation approach is to create deep reinforcement learning agents that use a segmenting encoding as state input powered by a model trained on a dataset created with the proposed methods. This may reveal whether a segmented state input is a useful component in creating agents capable of exceeding human performance across a broader state space, perhaps one that even spans multiple environments. The utilization of a low segmentation resolution in the dataset could increase the number of samples that could be stored in an experience replay mechanism, as well as the number of samples that could be used in a batch in an environment.

Another potential future work of interest is applying a similar algorithm to a 3D environment. In particular, automatically creating a traffic dataset for the purposes of training an autonomous driving agent.

In conclusion, the automatic labelling approach is an effective way of lowering the time cost of dataset generation over manual methods. Generating data in this way enables rapid experimentation with image segmentation parameters, and as such it should be used to determine the effectiveness of segmentation as input to deep reinforcement learning agents.

REFERENCES

Atari (1972). Pong. Atari 2600.

Badia, A. P., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskyi, A., Guo, D., and Blundell, C. (2020). Agent57: Outperforming the atari human benchmark. number: arXiv:2003.13350 arXiv:2003.13350 [cs, stat].

Baker, B., Kanitscheider, I., Markov, T., Wu, Y., Powell, G., McGrew, B., and Mordatch, I. (2020). Emergent tool use from multi-agent autocurricula. arXiv: 1909.07528.

Berner, C., Brockman, G., Chan, B., Cheung, V., Dennison, C., Farhi, D., Fischer, Q., Hashme, S., Hesse, C., Józefowicz, R., Gray, S., Olsson, C., Pachocki, J., Petrov, M., Salimans, T., Schlatter, J., Schneider, J., Sidor, S., Sutskever, I., Tang, J., Wolski, F., and Zhang, S. (2019). Dota 2 with large scale deep reinforcement learning.

Bojja, A. K., Mueller, F., Malireddi, S. R., Oberweger, M., Lepetit, V., Theobalt, C., Yi, K. M., and Tagliasacchi, A. (2018). Handseg: An automatically labeled dataset for hand segmentation from depth images. arXiv:1711.05944 [cs].

Capcom (1987). Mega man. Nintendo Entertainment System.

Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., Franke, U., Roth, S., and Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding.

Hudson Soft (1986). Adventure island. Nintendo Entertainment System.

Konami (1986). Castlevania. Nintendo Entertainment System.

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., and Talwalkar, A. (2018). Hyperband: A novel bandit-based approach to hyperparameter optimization. arXiv:1603.06560 [cs, stat].

Lundh, F. (1999). tkinter.

Nichol, A., Pfau, V., Hesse, C., Klimov, O., and Schulman, J. (2018). Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*.

Nintendo (1985). Super mario bros. Nintendo Entertainment System.

Papadeas, I., Tsochatzidis, L., Amanatiadis, A., and Pratikakis, I. (2021). Real-time semantic image segmentation with deep learning for autonomous driving: A survey. *Applied Sciences*, 11(19):8802.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2016). Prioritized experience replay. arXiv:1511.05952 [cs].