Playing MarioKart in Real-Time

Magalie Andorfer-Hirsch
CentraleSupélec
Gif-sur-Yvette
magalie.andorfer-hirsch@student-cs.fr

Rian Touchent
CentraleSupélec
Gif-sur-Yvette
rian.touchent@student-cs.fr

Abstract

Mario Kart is a good environment to train self-driving agents in a smaller setting than real-life simulations. We proposed two new datasets structure to train agents playing Mario Kart in real-time. We also provide trained models on this new datasets, however results are disappointing. We identified multiple improvement points. We think that this is due to how we generate the dataset. We used PPO as an expert agent, yet we think that recording a human playing would be at the same time easier and more efficient to obtain a more diverse dataset. We also tried 2 classical Computer Vision approaches, Lane Detection and Image Segmentation, which we think could be some interesting feature extraction methods that we could give to a learning algorithm.

1. Introduction

Mario Kart is a popular racing game. Its arcade game style makes it an interesting environment for self-driving agents. Tracks are simple and the player has to deal with limited information compared to real-life self-driving. We can also simplify actions to a very limited number. However, it introduces some interesting challenges like jump or holes in the tracks. They are very different from one another, with different road colors, textures and shapes so we cannot just hard-code feature extraction or any other presets.

We tried to develop some agents based on deep learning that could complete a track. We used two different settings : reinforcement learning and supervised learning. For the supervised part we only used the screen as inputs for our models. For the Deep RL part, we used a gym [3] environment. More precisely we used retro [11] which is a platform to turn classic retro games from famous game consoles using emulation into a gym environment. It is compatible with megadrive, SuperNintendo and GameBoy for example. We choosed the SuperNintendo version of MarioKart.

2. Related work

Numerous attempts of developing an agent for Mario Kart have been done before. TensorKart by Kevin Hughes is one of the first example. It is based on an end-to-end approach [2] made by NVIDIA which, using a simple CNN, tries to predict a steering angle, which will then be applied to an emulated controller. He trained it on 3 different tracks and was able to successfully generalize on a new track.

Another attempt was NeuralKart [9] made by Stanford students. The method is composed of a "search AI" which has total control of the emulation. It can generate a dataset composed of frames and corresponding steering angles. A simple CNN will then train on this dataset. Finally they randomly sample some states to augment the dataset so the agent learn how to deal with bad situations. Their bot successfully learn and complete some tracks.

A supervised approach was made by SethBling who tried to train a Neural Network to copy its game style. He recorded himself playing the game and then trained an agent on this data. The model was an LSTM who tries to predict an action between left, straight or right. The input is a transformed image of the screen. He cut half of the screen to remove the map and then turned the image to black and white and reduced its resolution. Using this transformation, he gives to the LSTM the last 8 frames to produce a prediction. He also recorded himself get out of bad situations like being stuck on the wall to show to the agent how to get out of it.

Finally, inspired by Sethbling, some students of the Instituto Superior Técnico de Lisbonne [5] made a retro [11] environment out of Super Nintendo Mario kart, which we will be using for the whole project. The reward function is basically the episode length and the episode ends when we achieve 4 laps or when we go too far outside the road. They successfully train an agent using PPO. However they could only train an agent for a specific map, they were not able to generalize.

Our idea was then to use the Sethbling approach mixed with the one from students from Instituto Superior Técnico. Training an agent to copy the game style of PPO agents

Player	PPO	Human
MarioCircuit	**1:17:15**	1:28:20
KoopaBeach	**1:25:15**	1:36:18
GhostValley	DNF	**1:32:16**
RainbowRoad	DNF	**1:28:20**
BowserCastle	2:30:15	**1:51:58**

Table 1. Times for each player on some tracks. *DNF: Did not finish*

trained in the retro environment.

3. Deep reinforcement learning

Inspired by the project from Instituto Superior Técnico's students [5], we trained a deep reinforcement learning agents to generate a dataset to later train a supervised agent using knowledge from every agents on each track.

We used PPO [12] as our algorithm and trained an agent for each tracks. We trained for 5 000 000 steps using stable baselines 3 [8] and open AI gym [3]. We used the reward function of their environment, which is the episode length, unchanged. However we did changed the observation. We applied some wrappers as we can see on fig.2 over the gym environment to transform the observation. We were inspired by a deep mind paper on atari games [10]. In this paper they warp the observation into a 84 by 84 grayscale image, they normalize the reward between -1 and 1 and they apply a stochastic frame skip. For the last one we used a value of 4 which means that the agent has to do a choice for an action only every 4 frames. Researcher from DeepMind showed that it drastically improve performance and training speed.

On some roads the training was successful and the trained agent could complete a race. However it fails completely on some tracks like rainbow road, which is also one of the most difficult for humans but should still be feasible. In this track the road is a grid of colors. We can imagine that because of the warping to 84x84 grayscale, we loose to much information and its too hard for the agent to distinguish the road anymore.

When the training is successful, like for the track MarioCircuit, it achieves good performance. It can complete the track with good times, sometimes even better than a human could do. It is still way below the Speedrun world record however, which is at 0:55:97. As we only give reward for episode length, the agent is not incentivized to complete the track as fast as possible. We can observe some interesting things. On Mario Circuit, the agent choose to go over grass, which is a slower path than following the road. Probably this is a way to maximize the reward without going too far outside the road which would end the episode. We should probably change the reward function after some training steps to incentivize the agent to complete tracks faster.

4. MarioKartFrameAction dataset

We propose a new dataset called MarioKartFrameAction. As we can see on fig.3b, it is composed of frames and corresponding actions. The action space is simplified to 4 inputs which are going straight, turning left, turning right and jumping.

It was generated by running each trained PPO agent on its respective track, we then saved the observation as an image, which is transformed by our wrappers, with its associated action choosed by the agent. We choosed 4 tracks to build this dataset which are MarioCircuit, BowserCastle, Koopabeach and ChocoIsland. This were the tracks were PPO had the best results. We arbitrarily decided that the last 3 would be used for training and MarioCircuit for testing. This way we will be able to see if an agent is able to generalize correctly by playing in an unseen track. During generation we ran each track for 10 000 steps. Which result in 10k images per track.

4.1. Limitation

We can note some limitation to our dataset. First, as PPO achieves good results, it is only composed of good situations. It means that even if a neural network successfully train on this dataset, if it fails a single time, it will be in a situation it has never seen without knowing how to get out of it. We should probably add in some way data about bad situations, either by playing ourselves, or by spawning PPO on random states to begin from.

The game runs at 60 frames per second, which means PPO makes a choice very often. Sometimes, for two very close frames it will choose something different, which is not an issue when playing because driving is a difficult task, you need to correct your trajectories every time. If in a span of 2 seconds you press turn left 30 times and turn right 2 times, you are still turning left. But for the neural network which will try to learn from it, this is very confusing. We will try to address this issues later.

4.2. Models

We tried three different models on this dataset. A MLP, a CNN and ResNet [7].

For the MLP, we designed a very simple network which takes as input 84x84 pixels, it goes through a hidden layer of 256 and then outputs 4 logits for each class corresponding to each action.

The CNN as shown in fig4 is composed of 2 convolutional layers which alternates with max-pooling layers. We used the ReLu activation function :

$$Relu(z) = max(0, z)$$

Figure 1. PPO training in parallel on 7 different tracks

Figure 2. Example of an observation seen by PPO

Finally ResNet [7] was borrowed from torch.hub. We modified its first layer to take only one channel has our images is in grayscal. We also modified its last layer so it can outputs only 4 logits.

All this models were trained using the cross-entropy function which goes as below :

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

with M the number of classes.

4.3. Evaluation

For evaluation we used two metrics which are Accuracy and F1 score. Accuracy is simply the number of correct predictions over all predictions. F1 score is more nuanced. It is based on Precision and Recall :

Models	MLP	CNN	RESNET
Train accuracy	0.67	0.74	**0.94**
Train F1	0.66	**0.69**	0.60
Test accuracy	0.32	**0.313**	0.288
Test F1	032	**0.313**	0.288

Table 2. F1 and accuracy scores measured for each model using MKFrameAction dataset

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

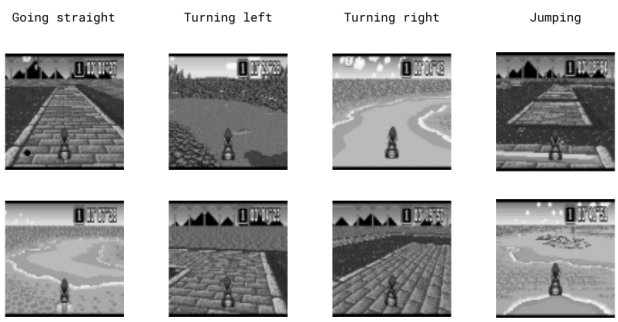
The formula of F1 is then :

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

4.4. Results

We can see that results are globally disappointing. ResNet achieve the best performance on training, however it is the worst on the test set. The CNN outperforms the MLP every time, which is expected as the CNN has the ability to extract spatial features in the image which is intuitively crucial for self-driving.

Resnet with an accuracy of 0.28 on test is just a little bit better than pure random which is not satisfying. The CNN is a little bit better but this is not very significant. We think



(a) Some examples of the MarioKartFrameAction dataset

List of actions :

```
going straight  : 4
turning left    : 4 + 1-left
turning right   : 4 + 1-right
jumping         : 4 + 8
```

(b) Illustration of our simplified action space

Figure 3. MarioKartFrameAction illustrated

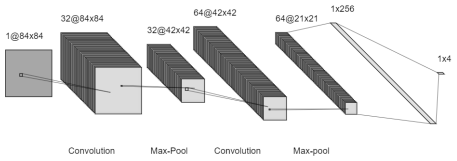


Figure 4. Design of our CNN

that this is due to the limitations of our dataset. That is why we decided to create another dataset.

5. MarioKartFrameSequenceDataset

As illustrated in fig.5, we designed two new datasets, this time based on sequence to palliate limitations of the first one.

First we designed it as a sequence because we made the hypothesis that it would be easier for the model to make a decision using previous steps. To go with this new dataset we will then design a recurrent neural network.

Then to solve the issue of multiple labels for very similar images, we designed a voting system. While PPO is running to generate the dataset, we stack previous frames. Then each time, the last x frames are saved as one sequence. The corresponding label is then assigned. We used two variants. We first tried to make every frame eligible to vote, then we made only the last four frames eligible. The action with most votes will be the label of the sequence. We think that this voting system can solve this issue because while PPO is turning left, the majority of actions will be turning left, so we won't confuse the model with very similar frames having different label, it will only see turn left.

We now have two new datasets, MarioKartFrameSequence8_4 with sequences of 8 images with the last 4 eligible to vote. And MarioKartFrameSequence16 with sequences of 16 images and all of them eligible to vote.

5.1. Model

For this new datasets, we needed a new model. We designed an architecture based on RNNs. However, in the last experiement we found that the CNN was better because it was using spatial feature extracted in the image. So we decided to Combine CNNs with LSTMs. Our network is composed of two convolutional layers that extract visual features that feeds in an LSTM layer. The output then goes through a dense layer to outputs 4 logits. This model takes as input a sequence of images and outputs a prediction only after it has seen all the images.

5.2. Limitation

We see some limitation with this approach too. First this model is heavier that the first ones. It has 13M parameters, which makes it more complicated to run in real time. However the hyperparameters could be more tuned than what we have done to reduce its size.

Then, it does not solve the issue of situations never seen, worse, we think it is even more vulnerable, because now it could be on a situation that he knows but coming from an unusual path.

5.3. Results

We achieve the best performance on train set for the MKFrameSequence8_4 however we generalized better on MKFrameSequence16. We can note that we achieve a way better f1 score than previously. We think this is due to the voting system than makes the label more smooth in the dataset.

We find it interesting that MKFrameSequence16 has better results on test set. This was not expected. This is probably due to sequences being longer. We are now curious to

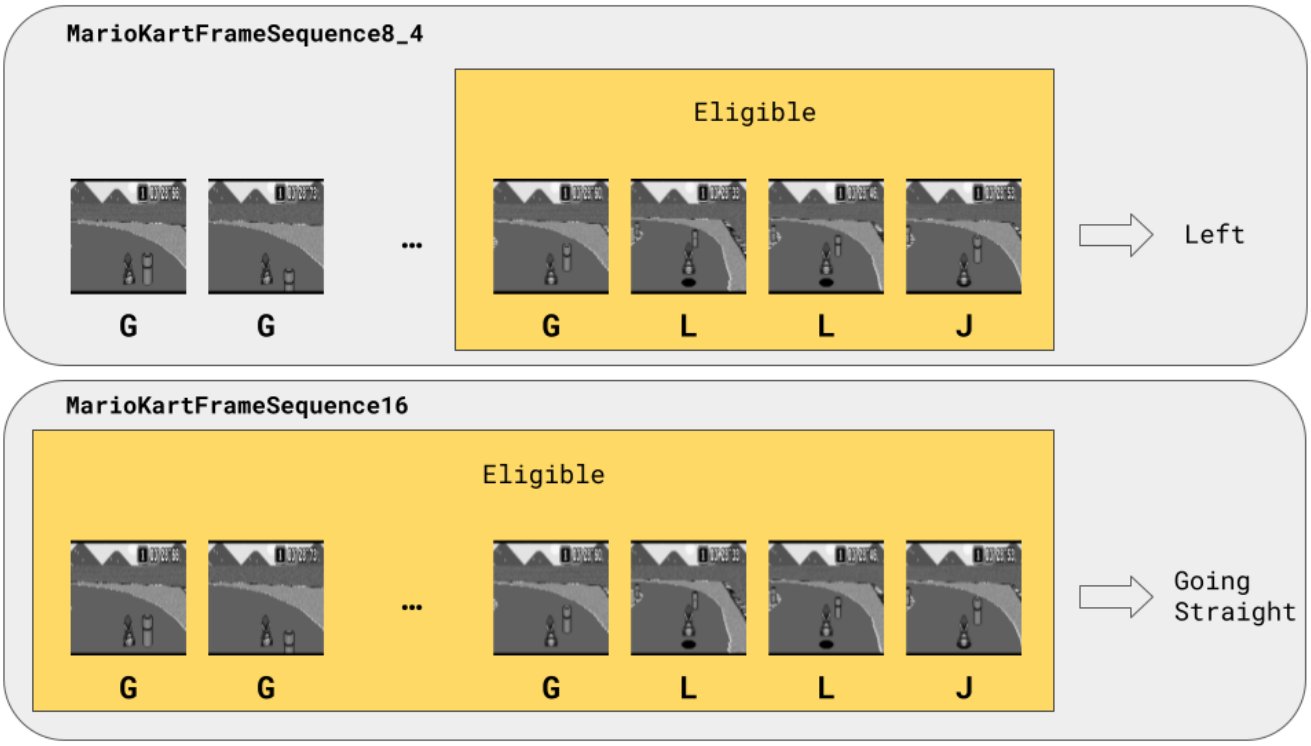


Figure 5. Illustration of our two variants of MarioKartFrameSequence

Dataset	MKFrameSequence8_4	MKFrameSequence16
Train accuracy	**0.89**	0.85
Train F1	**0.89**	0.83
Test accuracy	0.30	**0.38**
Test F1	0.30	**0.38**

Table 3. F1 and accuracy scores measures on both dataset using the CNN-LSTM architecture

try a MKFrameSequence16_4 to see if its better or worse to make eligible only the last frames.

6. Testing real-time

We then tried to test our models in real-time on the test track. For that we used the gym environment. When using the CNN-LSTM, we also used a new wrapper on our environment. Because this models needs a sequence of images, we used a wrapper called FrameStack, that will stack frames so the observation contains now 16 frames.

None of the model performed correctly on the game. Some of them were good on some parts of the tracks. CNN-LSTM was globally very good, however it inevitably falls in a bad situation which it cannot exit like in fig6. As our models does not know how to behave in this unseen situation they are stuck and they cannot complete the track.

Figure 6. Example of bad situation

The fact that they fails at some points is expected, even for human Mario Kart can be tricky and we fall in a hole. We think that the major failing point of our project is the inability of our agent of getting out of unusual situations.

We can also see that the CNN-LSTM model is too big to run in real-time. We should reduce its size. We could also convert it to a framework more relevant for production and inference like ONNX which should greatly improve inference time.

Figure 7. Detected edges using Canny algorithm

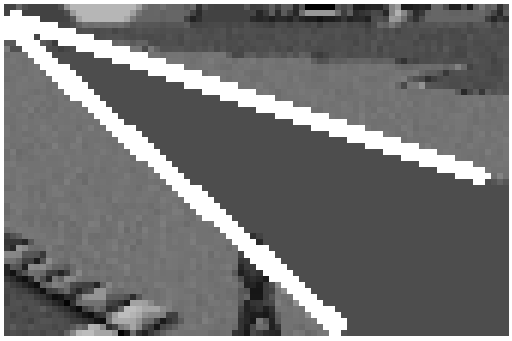


Figure 8. Detected lanes using Hough transform

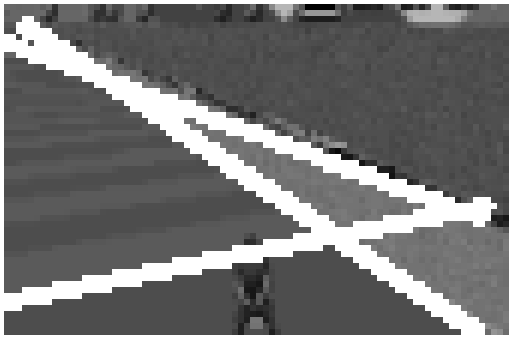


Figure 9. Unexpected lane detection using Hough transform

Figure 10. Observation segmented using K-means

7. Classical computer vision

We also tried some classical computer vision approaches on our datasets. We tried Lane Detection using Hough transform and segmentation using kmeans on colors.

7.1. Lane detection

Lane detection is a task in which we try to identify and find lanes in a road. It is useful in self-driving to compute a path for a car for example. There exists numerous techniques to detect lanes. We choosed the Hough Transform because it is easy to implement and quick enough to run in real-time.

The Hough transform is used to isolate an object of particular shape in an image through a voting procedure. In our case we are trying to detect lines corresponding to the lanes of the road. We will express lines in the polar system :

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right)$$

If there is the minimum number of votes, which we set at 50, and if it will create a line of more than 50 pixels, we consider it as a lane detected. We applied this lane detection algorithm to each frame.

To run the Hough transform we first need to detect edges in the image on which will apply the transformation, for that we used the canny algorithm. We first filtered only white pixels because we know that the lanes are white, so we applied Canny only on those pixels.

We can clearly see on fig.7 that the lanes of the road are detected as edges and that the road itself is flat. This is the ideal situation for lane detection. We probably doesn't even need to mask to choose more precisely where to apply the Hough transform.

On fig.8 we applied the Hough transform on the detected edges and we successfully detected lanes of the road. However sometimes the system completely fails.

On fig.9, the right side of the road is detected, however the border of the map is also detected along with a white pattern drawn on the road. We should try to detect only vertical lines to remove the pattern being detected. And we should more precisely tune the parameters, like filtered colors, so that border is not detected as a lane. We should also mask the middle of the screen because this is our Region of Interest (ROI)

7.2. Segmentation

Finally we tried to segment the observation. The idea was to segment using colors because the road in Mario Kart is often of a different color than everything else. We used the k-means algorithm for that with $k = 3$

On fig.10 we can see the result. The road is clearly distinguishable. We can also see Mario at the center. We could

try to fill the edges to that it is even more clear. In this figure, while we have way less information that with the normal observation, it is obvious that it is a left turn. It could be interesting using k-means to extract this feature and give it to a CNN instead of giving it the full observation.

We could also try a semantic segmentation algorithm. Using a network like U-Net, eventually pre-trained on CityScape for example, we could train it to segment the road. However this technique would require to collect and annotate a dataset which can be a slow and hard process.

8. Future ideas

We see multiple points of improvements for our projects. Generating a dataset with PPO is difficult and has multiple challenges. First the resulting agent is often either too strong or too weak. In the first case, it will maximize the reward as much as it can. Our reward was based on episode length so the agent was taking long cut so it can have more reward, which leads to strange and unusual path like driving outside the road to slow down the car. Then it will often converge on a single path that it will repeat on each lap. This makes dataset very redundant and not very diverse. To palliate this, we could take inspiration from [9] and make our agent spawn at random state of the game so it has to learn new paths and it will also make the dataset more diverse. When it is too weak, it just fails to learn how to complete a track. It happened for us on RainbowRoad. Surely there should be a way to make PPO work on this track but that would need more exploration and trials with wrappers.

Another idea and probably the easiest would be to generate a dataset by hand. By recording a human playing, we will have a more diverse dataset without having to wait hours to train an agent. We could simply ask the human to make himself stuck or in bad situation and record how he get out of that. Another advantage is that a Human is probably smoother in the inputs. While PPO can output a different action between 2 frames, a human will probably not do that. This was one of our major issue, some very similar frames had different label which confused our models.

Another improvement we could think of is to move from classification to regression. Outputting a steering angle seems a more reasonable task for self-driving. This should solve the issue mentioned above. It seems expected during a left turn to have some right turns inputs to control the path. But this confuses a lot the model because its nearly the same image. However a steering angle could have more nuance and not be treated as two independent classes. The neural network could learn what values of steering angle are needed to perform a turn.

Finally a LSTM coupled with a CNN seems a pretty reasonable choice because we need to extract visual features and treat them as a sequence. We could also try a model like ResNet3D [6] which achieves good results on action recognition. Or even a transformer like in this paper [4] where they use a model similar to GPT but in a reinforcement learning setting. It achieves SOTA on atari games using the screen buffer. It could probably achieves some great performances on our task.

9. Implementation details

We made all of our experiments using open AI gym [3], retro [11] and gym-snes [5]. We used stable-baselines3 [8] for PPO and PyTorch Lightning for the other models. The MLP, CNN and CNN-LSTM are implemented from scratch using PyTorch while ResNet was borrowed from torch.hub.

We keep track of our experiments using wandb [1]. We trained all of our agents using an Nvidia RTX 2070 SUPER running on a windows machine. We trained the 7 PPO agents in parallel because it didn't took that much VRAM. Training PPO took approximately 20 hours while training the CNN-LSTM, the largest model, took only few minutes. Our code is publicly available at this repo.

10. Conclusion

We proposed two new datasets for evaluating self-driving agents in a easier setting than real-life simulations. First MarioKartFrameAction which is a Frame/action pair dataset for classification algorithms. Then MarioKartFrameSequence which is composed of sequences of frames and their corresponding label which has been elected between actions of all frames of the sequence.

We also trained multiple models on those datasets. However while achieving some good results on the tracks we trained for, we fail to generalize to new tracks. Our best model, which has an architecture composed of a CNN feeding an LSTM, while achieve some good results, fails to play in real-time. First because it is a too big network and inference time is too slow. Then because it inevitably falls in an unusual or bad situations from which he doesn't know how to get off.

We think that this is due to a lack of diversity in our dataset and a lack of examples of bad situations. We proposed some improvements for generating a dataset using PPO, however we do think that generating a dataset by recording humans playing is at the same time easier and more efficient.

References

[1] Lukas Biewald. Experiment tracking with weights and biases, 2020. Software available from wandb.com. 7

[2] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. 1

[3] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *CoRR*, abs/1606.01540, 2016. 1, 2, 7

[4] Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Michael Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. *CoRR*, abs/2106.01345, 2021. 7

[5] Bernardo Esteves. Super mario kart snes - openai retro integration. 2019. 1, 2, 7

[6] Kensho Hara, Hirokatsu Kataoka, and Yutaka Satoh. Learning spatio-temporal features with 3d residual networks for action recognition. *CoRR*, abs/1708.07632, 2017. 7

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. 2, 3

[8] Ashley Hill, Antonin Raffin, Maximilian Ernestus, Adam Gleave, Anssi Kanervisto, Rene Traore, Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, Alex Nichol, Matthias Plappert, Alec Radford, John Schulman, Szymon Sidor, and Yuhuai Wu. Stable baselines. https://github.com/hill-a/stable-baselines, 2018. 2, 7

[9] Harrison Ho. Neuralkart: A real-time mario kart 64 ai. 2017. 1, 7

[10] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. 2

[11] Alex Nichol, Vicki Pfau, Christopher Hesse, Oleg Klimov, and John Schulman. Gotta learn fast: A new benchmark for generalization in rl. *arXiv preprint arXiv:1804.03720*, 2018. 1, 7

[12] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017. 2