Toolpath Planning through Reinforcement Learning in Additive Manufacturing

Master's Thesis Draft

Student: Yixiao Wang

Advisor: Professor Kornel F. Ehmann

Content

1 Introduction

Toolpath planning in additive manufacturing is a complex, tough but vital for a good quality product. From a superficial level, toolpath planning determines the path length, the number of lifting the nozzle and so on. These factors will influence the manufacturing efficiency. From a deep level, toolpath planning determines the path pattern (ring-shaped, zig zag, etc.), feed direction and other factors which will affect the final product quality including mechanical properties and surface finish.

1.1 Background

Additive Manufacturing (AM) produces physical objects layer by layer through a series of manufacturing process like extrusion, sintering, melting, light curing, spraying, etc. Compared with the traditional processing, for example, modeling, cutting and assembly, it is a "bottom-up" manufacturing method through the accumulation of materials, starting from nothing. This makes it possible to manufacture complex structural parts that were restricted by traditional manufacturing methods in the past.

Metal additive manufacturing is the focus of this thesis. From a superficial level, the idea path has the shortest length, minimal number of switching lasers and minimal number of turns. Path length will influence the manufacturing time. Frequently switching lasers will reduce the efficiency, damage the laser transmitter, and cost more energy. Too many turns will reduce the efficiency and may influence the product quality. For example, there may be holes in the turns. From a deep level, the idea path has some patterns or feed directions which will potentially influence the product quality. However, the inner mechanism remains unknown or requires very complex simulation and calculation.

1.2 Challenge

From the superficial level, path planning problem in additive manufacturing is a coverage path planning problem. Current algorithms are mainly depended on certain application and human designed patterns. From the deep level, geometry algorithms cannot handle the complex manufacturing process when manufacturing mechanism is introduced.

1.3 Contribution

This thesis makes following primary contributions

- From the superficial level, I propose a stable, fast, flexible, and universal 2D coverage path planning algorithm. This algorithm significantly reduces total path length and provides more diverse paths to fit different situation.
- From the superficial level, I cooperate with the senior to establish the Muzero reinforcement learning structure to solve the coverage path planning problem. I study how hyperparameters influence the performance, explore the effect of pretraining, distribute the calculation to accelerate the training, and modify the network structure to improve the performance.
- From the deep level, I explore impact factors on the performance of Muzero in the sparse reward structure.

1.4 Scope and Outline

Chapter 2 introduces 2D coverage path planning algorithms. Chapter 3 introduces Muzero reinforcement learning structure and related explorations.

2 2D Coverage Path Planning

Toolpath planning in additive manufacturing is a series of 2D coverage path planning (CPP) because AM should print the product layer by layer and in each layer, toolpath planning problem is a 2D CPP problem. In the current commercial software, toolpath planning algorithms are simple, stable but not efficient. Figure 1 shows two typic toolpath. The objective is to fill the grey area and the red lines are toolpaths. The toolpath consists of horizontal lines. This kind of paths pass through many white areas which are useless, and the machine should take the laser off.

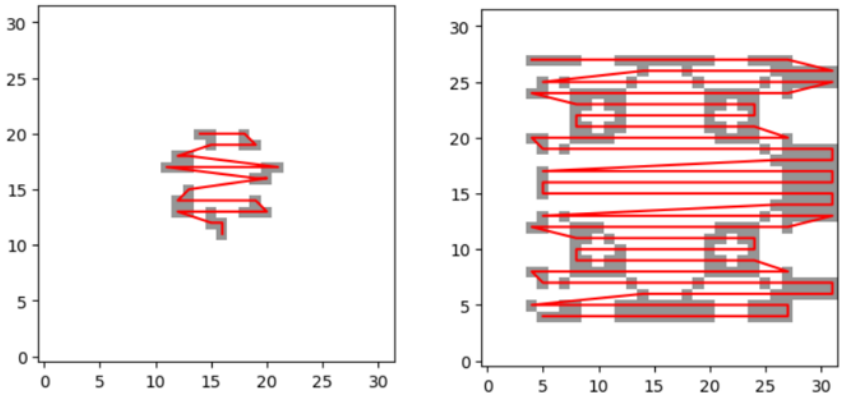


Figure 1: Two typic toolpath

2.1 Division of the Target Area

First, I divide the target area, the grey area, into small areas through connected graph. One pixel is connected to another pixel if it is at the left, right, up, or down of the other. Figure 2 shows the details. Arrows in (a) represent the connection direction. (b) shows pixel 1 is connected to pixel 0 but pixel 2 is not connected to pixel 0. However, if defining upper left is also a connection direction, pixel 2 is connected to pixel 0.

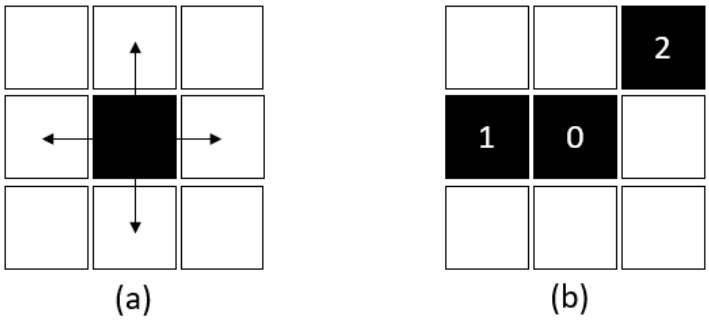


Figure 2: Connected graph

After dividing target grey area into connected graphs, local path planning can be done in each graph. Figure 3 shows two examples.

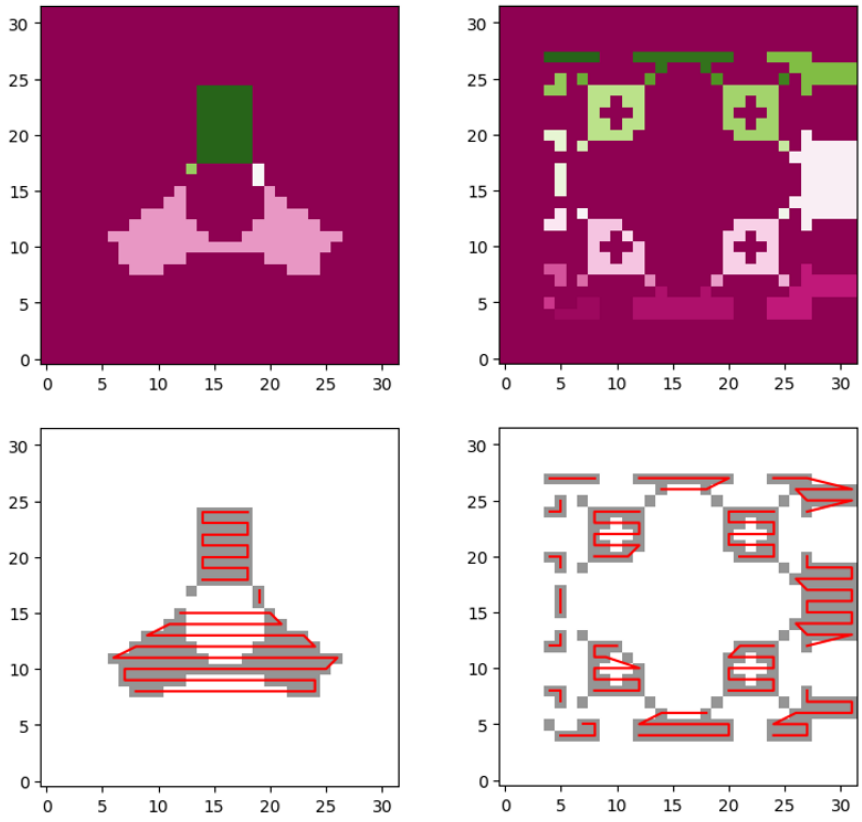


Figure 3: Examples of connected graphs and local path planning

After local path planning, total toolpath can be obtained if connecting local paths. Figure 4 shows an example. It is obvious that this kind of path pass less white areas.

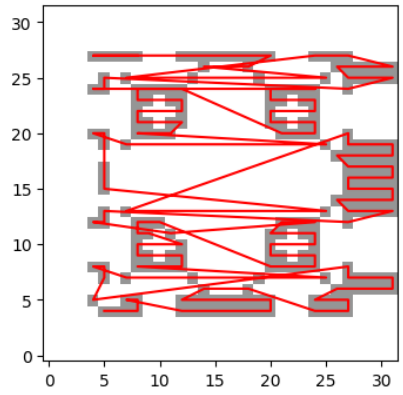


Figure 4: Total toolpath

2.2 Connection Order Planning

It is obvious that connection order of local paths will influence the total path length. In fact, connection order planning is a **Traveling Salesman Problem** (TSP).

Define the number of local paths as n. The path length of ith area is pl_i, the start point and final point are s_i and f_i, respectively. Define the virtual places as $vp_i, i\epsilon\{1,2,3,\dots,n\}$. For $i \in \{2,3,4,\dots,n\}$, the distance between vp_{i-1} and vp_i is $distance(f_{i-1}, s_i) + pl_{i-1}$.

Now, the problem is to plan a visit order from vp_0 to pass through all the $\{vp_i\}$ with the minimal length, which is a typical **Traveling Salesman Problem** (TSP). **Lin-Kernighan-Helsgaun Method** (LKH) is used to solve it.

Figure 5 shows three examples of total paths before and after connection order planning. Total path length of (c) after connection order planning decrease from **564.8** to **332.4**.

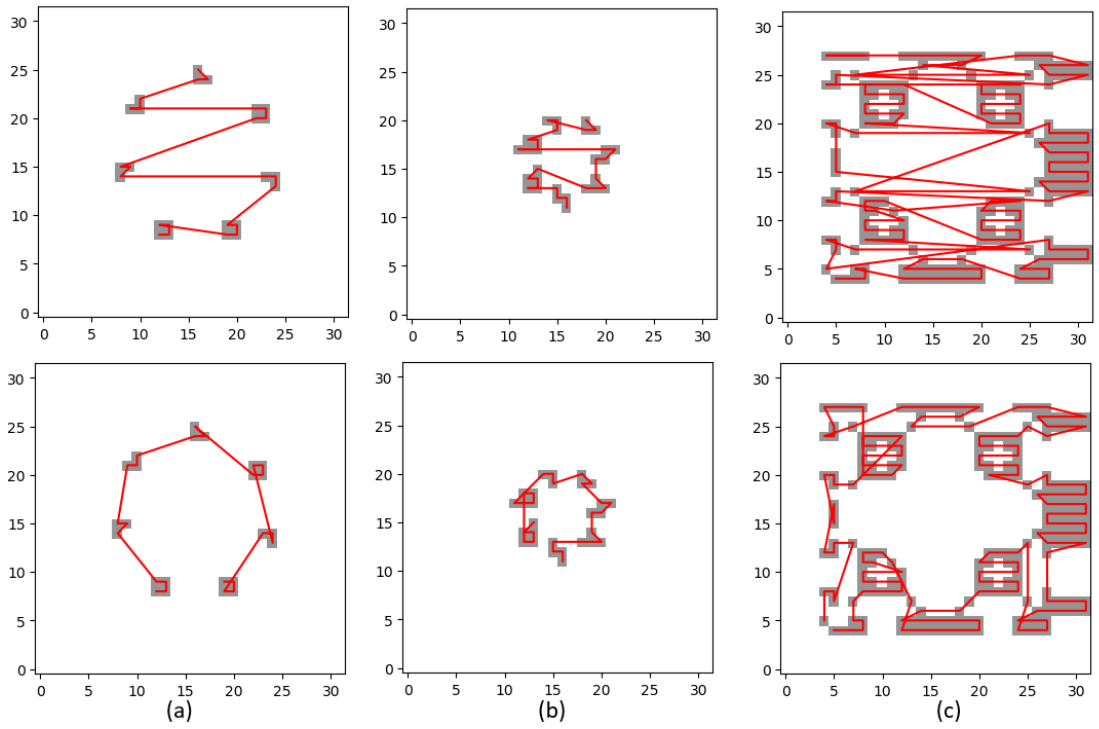


Figure 5: Total paths before and after connection order planning

2.3 Path Type Planning

Local path has several types. For example, path line can be vertical and horizontal. Figure 6 shows 8 typical path types.

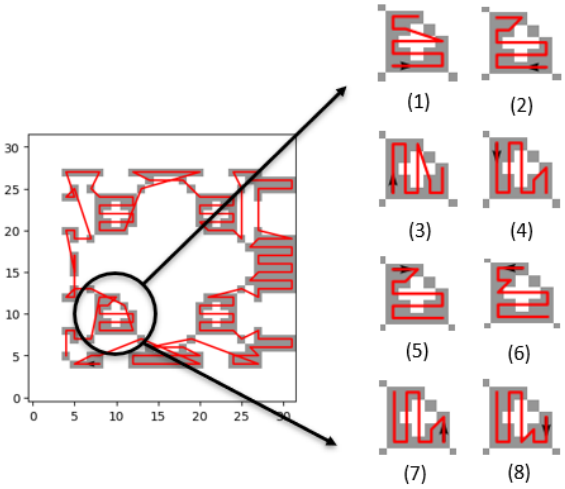


Figure 6: Typical eight path types

Define the type in ith local path as α_i. If $\alpha_i, i \in \{1,2,3,\dots,n\}$ are known, connection order can be calculated through LKH method so the total path length can be obtained. Define total path length as a function $TSP(\alpha_1, \alpha_2, \alpha_3, \dots, \alpha_n)$ and a nonlinear integer optimization problem is formed.

$$\min_{\alpha_i} TSP(\alpha_1, \alpha_2, \ldots, \alpha_n)$$
$$s.t. \alpha_i \in \{1, 2, 3, 4\}, \forall i$$

Genetic Algorithm is suitable to solve it. The reason is that crossover and mutation will reserve the good information. In this problem, in some connected parts, optimal path types can be unique and be affected little from the change of other parts parameters.

Figure 7 shows total path after path type order planning. The total path length decreases from **332.4** to **298.5**.

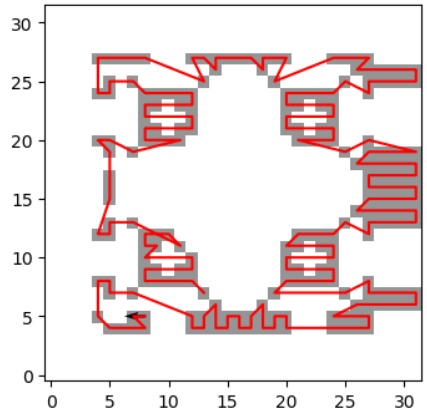


Figure 7: Total path

2.4 Experiments in Industrial Data Set

To be more convincing, I test the algorithm in the 32x32 section data set, which is established by my coworker Mojtaba. This dataset contains the planar projections of common industrial models, which is representative in metal additive manufacturing. Figure 5 shows the frequency distribution graphs. It is obvious that the number of long paths decreases, especially after path type planning. The average path lengths of (a), (b), (c) and (d) are **215.2**, **210.0**, **200.2** and **144.5**.

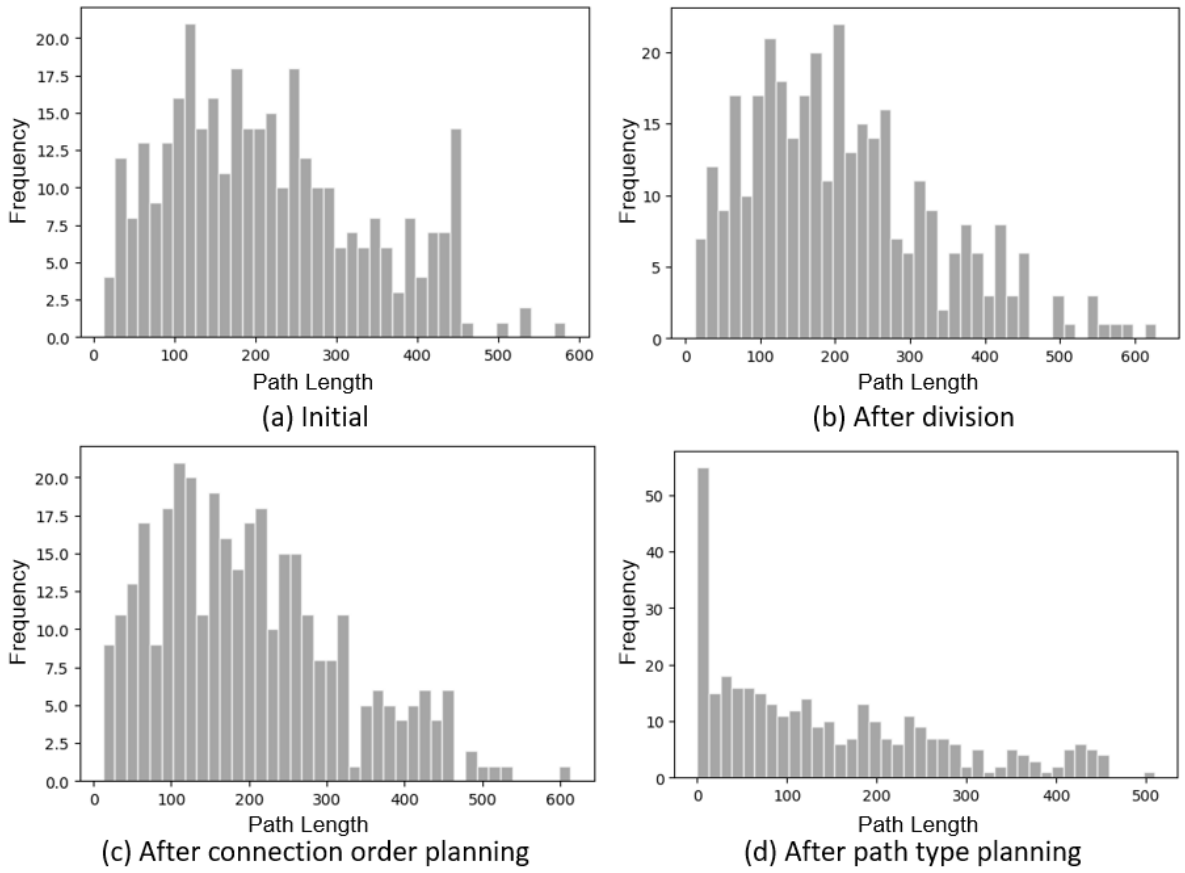


Figure 5: Path length frequency distribution graphs

3 Toolpath Planning through Reinforcement Learning

Traditional toolpath planning is based on human designing. According to certain objectives, such as minimal path length, less turns and properties from complex simulations, people design the path planning methods to approach these targets. For example, to reduce the number of turns, people design spiral path pattern and then calculate parameters to form a spiral path. This kind of path planning method is not universal, requires targeted design and in-depth understanding on the mechanism.

Learning method could overcome this problem. It does not need too much understanding about the mechanism so that prior path patterns or other rules can be obtained, and later, people can design the algorithm to achieve these priorities.

Reinforcement learning is utilized to achieve this kind of target. To be more specific, one of the most advanced reinforcement learning structure, Muzero, is applied in toolpath planning.

3.1 Muzero

Muzero is proposed by DeepMind in 2019. It does not tell the agent what kind of next step (next jet position in additive manufacturing) will lead to a good product. It makes the agent learn next step priorities itself.

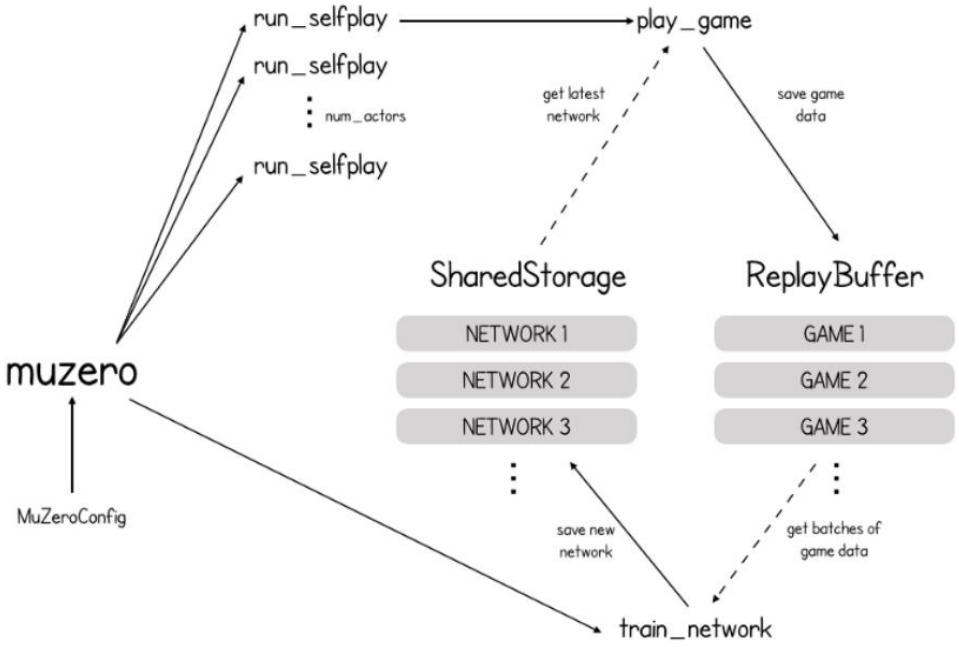


Figure 6: Overview of the MuZero [1]

Muzero will create several actors to play games simultaneously. When playing games, each actor will use the latest network in the SharedStorage to choose the next step. After game is over, the game history will be stored in ReplayBuffer. Using data in ReplayBuffer, networks will be trained and newest network will be stored in SharedStorage. After some loops, the program stops and Muzero can be utilized to play another different game and achieve great rewards. Figure 6 shows the overview structure of Muzero.

The most important part is how each agent plays the game, in other words, how each agent knows what the next step is. Muzero uses Monte Carlo Tree Search (MCTS) to tell the agent which next step will lead to a good future result. The main idea is to simulate what will happen in the next few steps and then to choose the next step which leads to the best future result.

There are three networks in the Muzero. First network h is to convert the input such as images, into hidden state. The function is similar to feature extraction. Second network f is to predict the current value and policy based on hidden state. Policy can be represented as the probability distribution of choosing an action. For example, going up will lead to a better product at the current state in AM. Current value can represent current score. The higher current score is, the better final product will be. Third network g is to predict the probability distribution of next hidden states if an action is applied and to predict the reward of this action if leading to a certain next hidden state. Value can be viewed as accumulated rewards. Figure 7 shows how these three networks work in the reinforcement learning. 0

means the current input, S_0 means current hidden state, S_1 means one of the possible next hidden states if action a is taken.

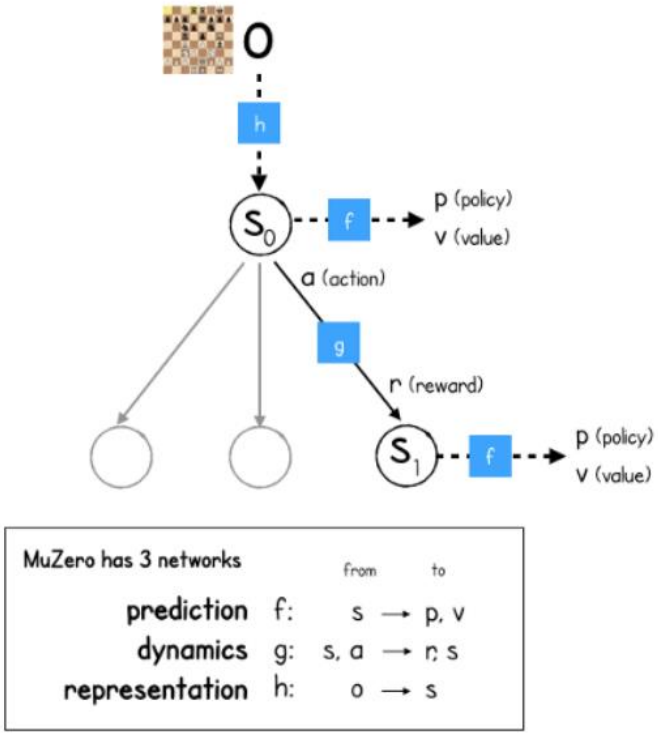


Figure 7: How networks work[1]

To apply Muzero into toolpath planning problem, we should define the toolpath planning environment. 32x32 section data set is used in this chapter, which has been mentioned in the last chapter. The aim is to plan a path passing through all the grey pixels as soon as possible. Figure 8 shows an example.

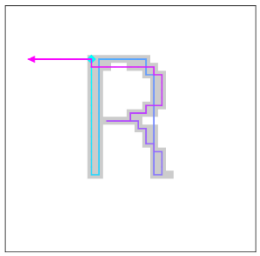


Figure 8: An example of toolpath from Muzero

3.2 Hyperparameter Tuning

3.2.1 The choice of actions

In the MCTS, the agent chooses the action a which will maximize the **Upper Confidence Bound** (UCB).

$$a^k = \arg\max_a \left[Q(s,a) + P(s,a) \cdot \frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)} \left(c_1 + \log \frac{\sum_b N(s,b) + c_2 + 1}{c_2} \right) \right]$$

where Q denotes current value, P denotes policy, N denotes visit counts. b in $N(s,b)$ denotes all the actions from node(state) s, and $N(s,b)$ denotes the visit count of the node after conducting action b from s. Each MCTS will backpropagate to the root node and all the node visited will add one visit count. Thus, $\frac{N(s,a)}{\sqrt{\sum_b N(s,b)}}$ shows the priority of the action a from all the actions and its reciprocal shows the exploration. Note that if prediction network works well, which shows that $P(s,a)$ is like MCTS, $P(s,a)$ will reduce the influence of the $\frac{\sqrt{\sum_b N(s,b)}}{1+N(s,a)}$. So, we can conclude that the first term means greedy strategy and the second term means exploration.

Increasing c_1 will increase exploration obviously because current value Q means greedy strategy, which is to always choose the action leading to greatest current value. Reducing c_2 will increase exploration but the exploration is affected by the maximum visit count in MCTS according to

$$\log \frac{\sum_b N(s,b) + c_2 + 1}{c_2} = \log \left(\frac{\sum_b N(s,b) + 1}{c_2} + 1 \right)$$

Note the maximum of $\sum_b N(s,b)$ is num_simulations (=50) if the laser does not pass the same pixel repeatedly. In the go game [2], $c_1 = 1.25$ and $c_2 = 19652$. $c_2 = 19652$ is not suitable for our case, because if $c_2 = 19652$, logarithmic term will always be zero. So I choose $c_2 = 1000$ (new_section_baseline), $c_2 = 500$, $c_2 = 100$ and conduct the experiments. Figure 9 shows the results and we should choose $c_2 = 500$ because the total reward increases faster and maximum total reward is higher.

Figure 9: Total reward

3.2.2 Temperature

Final step of MCTS is to calculate the probability of choosing action a, which is

$$p_a = \frac{N(s,a)^{1/T}}{\sum_b N(s,b)^{1/T}}$$

$T = 1$ means probability is proportional to the visit count. $T < 1$ means the bigger visit count means bigger probability compared with the proportion. $T = 0$ means choosing the maximum one. Small T leads to more randomness which means more exploration.

$$T_1(x) = 1, \forall \in x(0,1)$$

$$T_2(x) = \begin{cases} 1.0, x \in (0,0.5) \\ 0.5, x \in (0.5,0.75) \\ 0.25, x \in (0.75,1) \end{cases}$$

where x is the ratio of current training step to total training steps. Figure 10 shows the results. Blue line uses $T_1(x)$ as the temperature strategy and orange one uses $T_2(x)$ instead. The results show temperature strategy does not affect the total reward significantly.

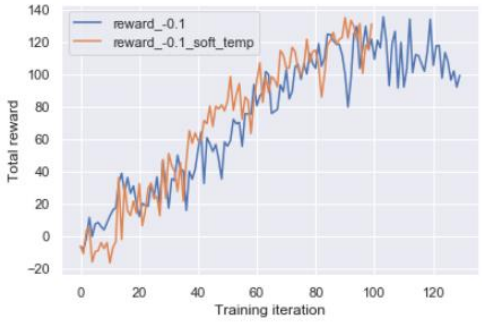


Figure 10: Total reward

In [2], for the Atari game, actions are selected from visit count distribution as just mentioned throughout the duration of the game. For Go, actions are selected from the visit count distribution before T_s moves. And when the number of moves reach T_s, T will be set to 0. From Figure 11, our experiments show this hyperparameter does not affect the total reward significantly. Temperature threshold T_s of baseline is 300.

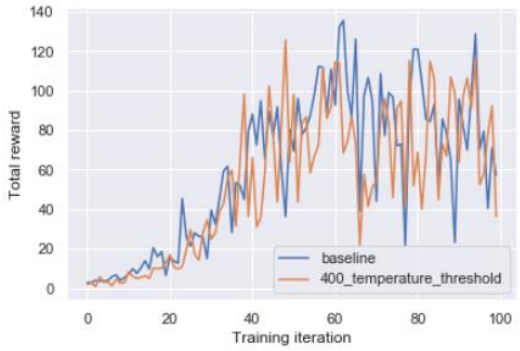


Figure 11: Total reward

3.2.3 Dirichlet Distribution

When we have the action distribution p_a from MCTS, we should add some randomness p to it with certain ratio r in the training procedure so that the agent could explore more. The final action distribution is

$$P(a) = (1 - r)p_a + rp$$

In Muzero, randomness p is Dirichlet distribution whose pdf (probability density function) is

$$f(x_1, \ldots, x_k; a_1, \ldots, a_k) = \frac{1}{B(a)} \prod_{i=1}^{k} x_i^{a_i - 1}$$

where $\sum_{i=1}^{k} x_i$, and $\forall x_i > 0$.

Note that we do not need to care about $B(a)$.

And it is obvious that when $\forall a_i < 0$, $x = (x_1, x_2, \ldots, x_k)$ will be more likely at the boundary which means the probability of choosing one certain action can be much higher than the others. If $\forall a_i = 1$, the probability of this point x appearing at any position within the k-dimensional cube is equal, which means the probability of choosing any action is the same.

Keep r as a constant. x near the boundary may change the order of action probabilities because some probabilities of certain actions may greatly increase. And if $\forall a_i = 1$, the order of action probabilities will not change but the gap between them will shorten.

3.2.4 Td Step

Td_steps means the number of steps used to calculate the target value and target reward.

3.2.5 Unroll_Steps

When getting batches from game_history, firstly sample the games and then sample the game position. After that, get *unroll_steps* number of histories for certain game position in certain game. This term is used to train the dynamic network g.

3.2.6 Reward Assignment

Reward assignment is to define what reward the agent can get if certain action is taken at the current state. For example, when laser goes out of target region (grey pixels), the machine cannot print anything so the reward can be zero. When laser goes to grey pixel and this pixel has not been printed, the reward can be one.

Figure 12 shows the results from the first set of experiments. In this set of experiments, there are four kinds of actions: left, right, up and down. There are three kinds of situation for additive manufacturing. The laser goes to target region and it has not been printed, defined as r_1; the laser goes to target region and it has been printed, defined as r_2; The laser goes out of target region, defined as r_3. Figure 12(a) is the total reward when $r_1 = 1, r_2 = r_3 = 0$. Figure 12(b) is the total reward when $r_1 = 1, r_2 = 0.1, r_3 = 0$. Results show that latter reward assignment is better and more stable than the other. The reason is that latter assignment will tell the agent to go inside the target region rather than go out of target region. If r_2 is more than 0.2, the total reward will decrease significantly. The reason is that the agent gets stuck.

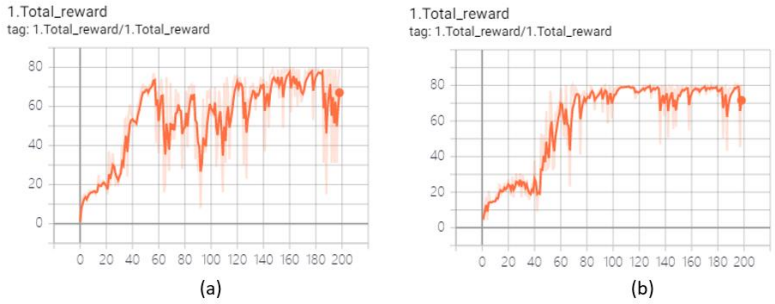


Figure 12: Total reward

Figure 13 shows the result from another set of experiments. In this case, the kinds of actions are eight: laser-on and left, laser-on and right, laser-on and up, laser-on and down, laser-off and left, laser-off and right, laser-off and up, laser-off and down. There are three kinds of reward. The machine prints the target region which has not been printed, defined as r_1; the machine prints at the wrong place (out of target region or where has been printed), defined as r_2; the machine turns the laser off, defined as r_3. r_1 and r_3 can be always set as 1 and 0. In Figure 13, legend "baseline_8action" means $r_2 = 0$; legend "reward_-1" means $r_2 = -1$; legend "reward_-0.3" means $r_2 = -0.3$; legend "reward_-0.1" means $r_2 = -0.1$. Results show too small r_2 is bad and r_2 in a proper range does not have much effect on the total reward.

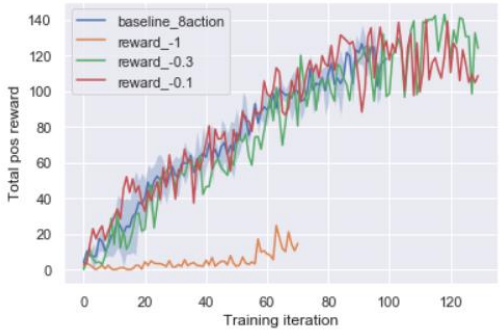


Figure 13: Total reward

From these experiments, we can conclude that reward assignment should be in the proper range, otherwise the total reward will be very bad. But in this proper range, reward assignment will not affect the total reward much.

3.2.7 Loss Weight

The loss function in Muzero is the weighted sum of policy loss, value loss and reward loss.

$$l = w_p l_p + w_v l_v + w_r l_r$$

The weights of three losses may influence the total reward. Set $w_p = 1$. Figure 14 shows the result of experiments. Legend "reward_-0.3" means $w_v = 0.25$, $w_r = 1$; legend "weight_1_4" means $w_v = 1$, $w_r = 4$. After several experiments of former weight assignment, we find that policy loss is always around 60, value loss is around 15, reward loss is around 15. So we quadruple the weights of value loss and reward loss and want the agent to train each loss equally. The result shows latter reward

assignment will make the total reward increase much faster at the beginning. The reason is that at the beginning, reward and value predictions have large errors. And the policy is obtained from reward and value prediction. Training the policy with wrong reward and value predictions leads to bad total reward. If increasing the weights of them, reward and value loss can be trained more at the beginning so they will be more accurate. In the end, it will benefit the total reward.

Figure 14: Total reward

3.2.8 Window Size

The input of reinforcement learning is single window centered at the laser position. The section used is 32x32. Intuitively, the window size is also 32x32. But there may loss some information about the environment. Figure 15 shows an example in which this window losses the information of target region. At this case, the agent cannot know what the next step is because in its sight, there is no target region to be printed. So 64x64 window size could solve this problem. Figure 16 shows the result, which conflicts with our expectation. The reason may be if we set 64x64 as the window size, the window contains too much useless information (3/4 of the information is useless). And there are many convolutional layers in the network which will dilute useful information. As a consequence, the agent cannot extract important information so that the performance is worse.

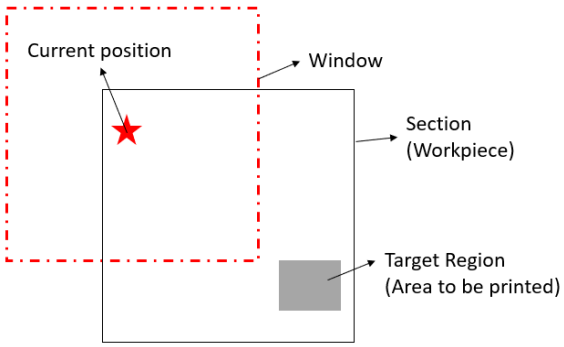


Figure 15: Window and Section

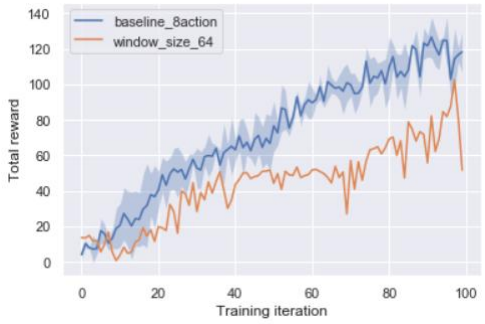
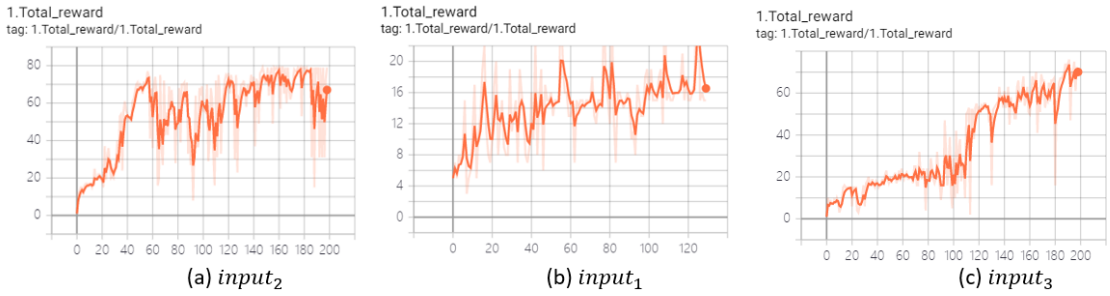


Figure 16: Total reward

3.2.9 Input Design

In our case, the remaining grey pixels and current position of laser are two necessary information to the agent (when one grey pixel has been printed, it will turn white afterwards). How to represent necessary information and express them to the agent is essential in reinforcement learning.

Traditional input in reinforcement learning is sequential images. In our case, two sequential images of the sections can calculate the current position of laser, defined as $input_1$. Another kind of input can be single window centered at the laser position, defined as $input_2$. Also, we could design the input as two images of section and laser position, defined as $input_2$. Results in Figure 17 show $input_2$ is better for our case. The reason is that $input_2$ is more clear and the agent can easily extract necessary information.



(a) $input_2$ (b) $input_1$ (c) $input_3$

Figure 17: Total reward

3.3 Pretraining to Accelerate the Training

Reinforcement learning is time consuming because it contains playing, interacting with the environment and training. In our 32x32 coverage path planning case, it always takes more than one day to finish one experiment. Generally, reinforcement learning policy (RL policy) requires 100,000s of games histories to learn effectively. As mentioned before, inaccurate reward and value prediction is bad for policy convergency. Inspired by [3], we intuitively think pretraining the representation network h may help the reinforcement learning training. The strategy is to establish the dataset of remaining grey pixels, actions, rewards and values. With the same network of Muzero, we can predict the rewards and values through remaining grey pixels and actions. Here comes to a supervised learning problem. We believe that after pretraining procedure, representation network can extract useful information and predict the value and reward accurately so that RL policy can fast go to convergency.

Dataset is established by 32x32 section data set, the same as what we use in toolpath planning problem. When training the network of Muzero, real inputs are the windows of sections where the machine has conducted some operations. So we set several kinds of toolpath like lines and rectangles, and add them randomly into the section in the dataset. Then we choose the current position of laser randomly and finally get the window as network input. Reward can be determined by the laser position, window and randomly chosen action. Value can be predicted as

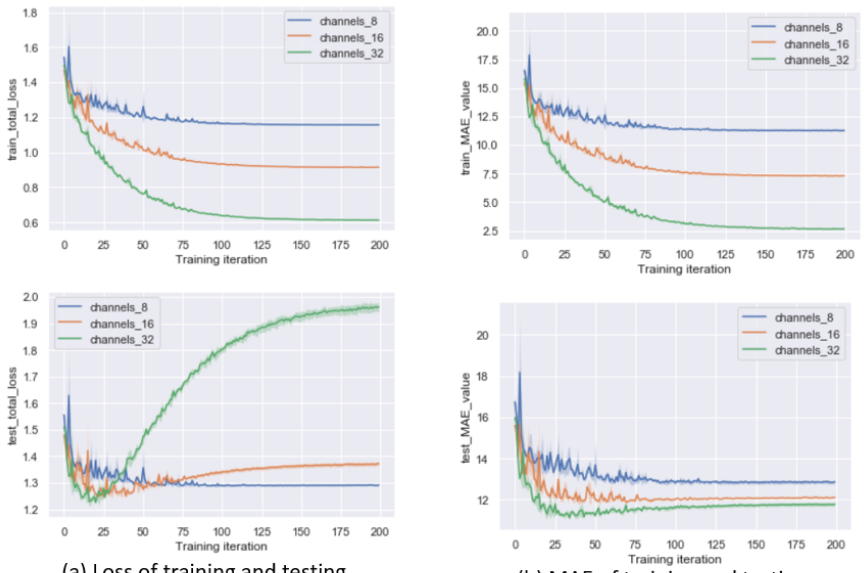
$$V = r(\gamma^0 + \gamma^1 + \cdots + \gamma^{n-1})$$

where r is the reward when filling a target pixel, n is the number of remaining target pixels, γ is the discount rate.

3.3.1 Neural Network Structure

When we train this supervised network, we find the total loss of value and reward cannot converge to a very low number as expected. As we all know, neural network can fit any function so supervised learning under idea settings can reduce the loss into a small number. We begin to suspect whether our neural network is capable to predict the value and reward. And the capability of the neural network is essential in reinforcement learning.

First, we change the number of channels for each layer. Figure 18 shows the result. We can conclude that with the increase of number of channels, total loss of training can be reduced but results from testing say that overfitting will occur if we choose too many channels. Because the loss here is calculated from support vector [2] of the scale, which does not represent the scale value directly. To be more intuitive, Figure 18(b) shows mean absolute error (MAE) of value (not reward) dialog. Maximum value is less than 32x32=1024. Assume average value is 300. So the MAE rate is around 0.8%~4%. This is acceptable in our case because the target value is not accurate. And the MAE of reward is always less than 0.01, which means MAE rate is less than 1%. So we can conclude our neural network with 32 channels has the capability to predict the value and reward accurately. MAE rates of value and reward are both less than 1%. To increase the capability, we can choose 32 channels and stop the training when test loss increases. But 32 channels will cost much more time than 8 channels.

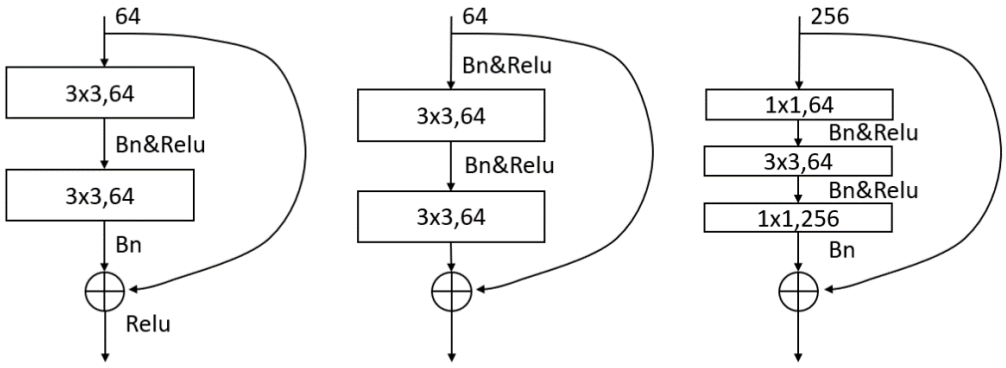


(a) Loss of training and testing

(b) MAE of training and testing

Figure 18: Loss

The effects of expanding the neural network in depth and breadth are similar. So second, we turn to another factor the structure of residual blocks. Traditional residual block structure is shown in Figure 19(a), which is used in the initial Muzero. According to [4] , it would be better if we put BN layer and ReLu layer before Conv layer, which is Residual block 2 as shown in Figure 19(b). To enhance the adaptability of the residual block, we make it work when the number of input channels differs with that of output channels based on [5], which is shown in Figure 19(c). Figure 20 shows experiment results. From Figure 20(a), we can conclude residual block 1&2 have similar and better performance. From Figure 20(b), residual block 2 has a good calculation efficiency. All in all, we should utilize the residual block 2 structure.

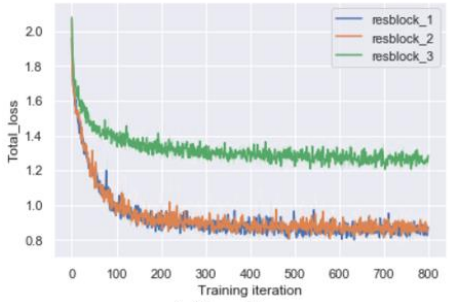


(a) Residual block 1

(b) Residual block 2

(c) Residual block 3

Figure 19: Residual block structure



(a) Total loss

resblock	experiment	time
resblock_1	1	1h2m32s
resblock_1	2	1h1m59s
resblock_1	3	1h1m19s
resblock_2	1	37m42s
resblock_2	2	37m37s
resblock_2	3	37m22s

(b) Cost time

Figure 20: Results

3.3.2 Pretrain Results

After pretraining, we can fix the weight of representation network, which means using the hidden state calculated by pretrained network. We can also use the weight of pretrained network as the initial network of Muzero. This is similar to transfer learning. From experiments, the result of the latter procedure is similar to that with no pretraining. So we fix the weight in the later experiments.

We can pretrain the reward and value at the same time. But the result is bad. Total reward cannot increase an obvious number. The reason we suppose is that the target value in the dataset is not accurate. So we pretrain the reward only. Figure 21shows the result which is not good as expected. The total reward increases faster than that with no pretraining but later the total reward cannot reach the expected number. The reason we suppose is that hidden state extracted is only useful to reward prediction.

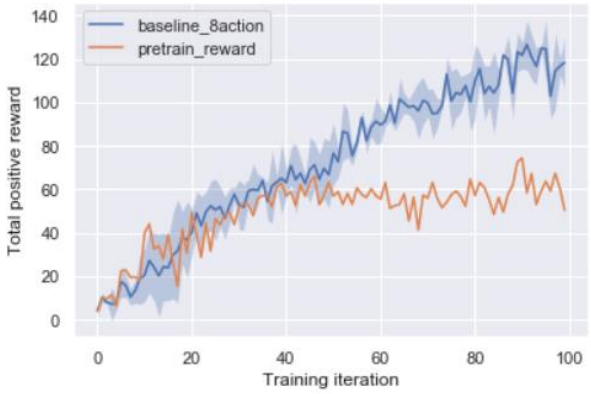


Figure 21: Total reward

From above analysis, we can conclude that pretraining methods we have tries are useless in our case.

3.4 Distributed Calculation

Training reinforcement learning network is time consuming. In our case, it takes several days to finish one experiment. We should take some methods to accelerate the training.

In the initial version of the Muzero, we adopt a linear calculation method. First, we make agents play games one by one and then train the network. We can use different CPU cores to play games simultaneously and when agents are playing, we make GPU to training the network. We realize this kind of distributed calculation through python package ray.

3.5 Sparse Reward Structure

This part is not yet complete and needs more experiments.

The discussion in chapter for now is around dense reward structure, which means the agent will get a reward after taking one step. However, from a deep level, it is more meaningful to give the agent a reward after all the steps are finished. The reason is that the inner mechanism in AM which influences the final product quality is unknown or too complex to simulate. We can only get the final product quality when we finish the manufacturing process. This kind of reward structure is called sparse reward structure.

First, we have tested different types of inputs to see which is more suitable for sparse reward. From experiments, we can conclude that inputs must contain necessary information to predict the reward. For example, if we use the number of occurrences of certain path pattern in the end like appearing “up, up, left, down, down” in order, it is not suitable to use single window as the input. Because single window cannot contain the trajectory information. Second, we have added “budget” as another kind of input. Budget means the remaining steps the agent can take in the future. The result in the dense reward structure is promising. The result in sparse reward structure needs more tuning. Third, we have tested influence of weight of reward loss. Rewards in the sparse reward are always zero except the last one. It may be bad to train the reward loss. But the results seem similar.

4 Conclusion

This part is not yet complete.

5 Acknowledgment

Thanks to Professor Ehamnn. He gave me a lot of suggestions and freedom in topic selection. The advice on how to arrange my master’s study life is very helpful and always guided me when I was important nodes in life during these years.

Thanks to Mojtaba Mozaffar. He is my important coworker in this thesis and has given me a lot of academic advice and life suggestions. We spent a period of meaningful and fulfilling scientific research life together. I benefited a lot from our weekly meetings.

Thanks to AMPL. Lab meetings in AMPL are enthusiastic, open, friendly, and of highly academic level. I could always learn a lot from them.

Thanks to Samantha Webster. She led me to the laboratory and introduced many master’s topics I could choose. These topics are interesting and meaningful. Though I didn’t choose one of these topics out of consideration for the future, thanks a lot to Samantha!

Thanks to Shuheng Liao! We are good friends and have spent a lot of great time together. I will never forget the time I spent with you in America!

6 Reference

[1] https://medium.com/applied-data-science/how-to-build-your-own-muzero-in-python-f77d5718061a.

[2] Schrittwieser, Julian, et al. "Mastering atari, go, chess and shogi by planning with a learned model." arXiv preprint arXiv:1911.08265 (2019).

[3] Mirhoseini, Azalia, et al. "Chip Placement with Deep Reinforcement Learning." arXiv preprint arXiv:2004.10746 (2020).

[4] He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

[5] He K, Zhang X, Ren S, et al. Identity mappings in deep residual networks[C]//European conference on computer vision. Springer, Cham, 2016: 630-645.