Article

Image Classification Method Based on Multi-Agent Reinforcement Learning for Defects Detection for Casting

Chaoyue Liu, Yulai Zhang * and Sijia Mao

School of Information Technology and Electronics Engineering, Zhejiang University of Science and Technology, Hangzhou 310023, China; 222008855017@zust.edu.cn (C.L.); 222008855019@zust.edu.cn (S.M.)
* Correspondence: zhangyulai@zust.edu.cn

Abstract: A casting image classification method based on multi-agent reinforcement learning is proposed in this paper to solve the problem of casting defects detection. To reduce the detection time, each agent observes only a small part of the image and can move freely on the image to judge the result together. In the proposed method, the convolutional neural network is used to extract the local observation features, and the hidden state of the gated recurrent unit is used for message transmission between different agents. Each agent acts in a decentralized manner based on its own observations. All agents work together to determine the image type and update the parameters of the models by the stochastic gradient descent method. The new method maintains high accuracy. Meanwhile, the computational time can be significantly reduced to only one fifth of that of the GhostNet.

Keywords: multi-agent reinforcement learning; casting image classification; casting defects detection

check for updates

Citation: Liu, C.; Zhang, Y.; Mao, S. Image Classification Method Based on Multi-Agent Reinforcement Learning for Defects Detection for Casting. *Sensors* **2022**, *22*, 5143. https://doi.org/10.3390/s22145143

Academic Editors: Jiachen Yang and Dezong Zhao

Received: 17 June 2022
Accepted: 7 July 2022
Published: 8 July 2022

1. Introduction

Casting defects detection is an essential problem in the machinery industry. High quality castings are very important for automobiles, engineering equipments and other products. It is necessary to find and deal with these defective castings in time. Otherwise, it will harm the quality of downstream industrial products [1]. With the development of computer vision technology, defects detection methods based on casting image processing have been proposed and applied in the recent years [2–5]. These progresses are made upon various excellent network structures, such as AlexNet [6], VGG [7], and ResNet [8]. In the past decade, in order to improve the classification accuracy, the models are designed to be more and more complex, which leads to an increasingly large number of parameters and an exponential increase in the computational burdens, especially for large images [9,10]. The image size of the castings is also large, but usually the defects are only located in a small number of the pixels, which means large numbers of the pixels in the image are useless for the defects detection task, but they still consume equivalent computational resources. So, the reinforcement learning method with multiple agents are used in this paper to find the most informative sub-images.

Image classification based on reinforcement learning has been a research hot spot in recent years. In the work of Ref. [11], the policy gradient method [12], which is a very basic reinforcement learning algorithm, is used to optimize the convolutional neural network model for image classification. In the work of Ref. [13], the Faster R-CNN model [14] is used to implement anomaly detection while the Deep Q-learning algorithm [15] is used to classify anomalies in the images. In the above works, the reinforcement learning algorithms are used as the optimizer on all the pixels of an image or a sub-image derived by some other methods, so single agent algorithms are natural choices in such a situation. On the contrary, reinforcement learning algorithms can also be used as an effective tool to find the key region in the image. In Ref. [16], Deep Q-learning algorithm is used to select a vital area of the image in a vehicle classification task. In Ref. [17], a multi-agent algorithm

PixelRL, which is based on the A3C [18] algorithm, is proposed. In PixelRL, each pixel has been assigned an agent so the number of agents is equivalent to the number of pixels, and it achieves excellent results compared to traditional methods [19,20]. At the same time, another multi-agent algorithm is proposed in Ref. [11], where each agent controls a sliding square window. This method is more robust and has better generalization based on the results of the experiments on the CIFAR-10 dataset and the MNIST dataset [21].

The idea of this work is to use multiple isomorphic agents with sliding square window to find the defect regions. Meanwhile, GRU (Gated Recurrent Unit) [22] models are used to aggregate beliefs on the trajectories of the agents. Compared with the original image, the size of the observation image of each agent has been significantly reduced, and the image category can be judged more quickly. In the following part of this paper, the multi-agent problem is constructed in Section 2. The proposed method and the corresponding models are described in Section 3. In Section 4, experiments are performed on the casting image dataset and compared with several mainstream convolutional neural network based models. The conclusions are given in the last section.

2. Multi-Agent Reinforcement Learning

For the problem of casting image classification, a decentralized partially observable Markov decision process model can be used [23–27], which can be expressed as an eight-tuple $\langle N, S, A, T, R, O, Z, \gamma \rangle$, where N represents the number of agents, usually greater than or equal to 2 in the case of multi-agent reinforcement learning. S represents the state space. $A = A_1 \times A_2 \times \cdots \times A_N$ represents the joint action space of all agents, where A_i represents the action space of the i-th agent. $T : S \times A \times S \rightarrow [0, 1]$ is the state transition function, which is used to represent the probability of the agent transitioning to state $s' \in S$ after performing a joint action $a \in A$ in state $s \in S$. $R : S \times A \times S \rightarrow \mathbb{R}$ is the reward function, which represents the reward obtained by the agent after the state transition occurs. Under fully cooperative tasks, the reward value of all agents can be the same. $O = O_1 \times O_2 \times \cdots \times O_n$ represents the joint local observation space of all agents, where O_i represents the local observation space of the i-th agent. $Z : S \times A \rightarrow O$ is the observation function. The i-th agent can only obtain local observations O_i under the condition of $s \in S$. $\gamma \in [0, 1]$ is the discount factor.

According to the different training schemes, multi-agent reinforcement learning methods can be divided into three categories, namely CTCE (Centralized Training Centralized Execution), DTDE (Distributed Training Decentralized Execution), and CTDE (Centralized Training Decentralized Execution) [28–30]. In the CTCE scheme, there is a centralized controller. All agents submit their observations and rewards to the centralized controller, and then the centralized controller decides the action for each agent. In the DTDE scheme, each agent interacts with the environment independently and determines actions by itself. In the CTDE scheme, the centralized controller knows each agent's observations, actions, and rewards during the training phase. After training, the centralized controller is not needed [31,32]. This paper adopts the DTDE scheme, does not require a centralized controller, and adds a communication mechanism, and each agent can obtain information from other agents, as shown in Figure 1.

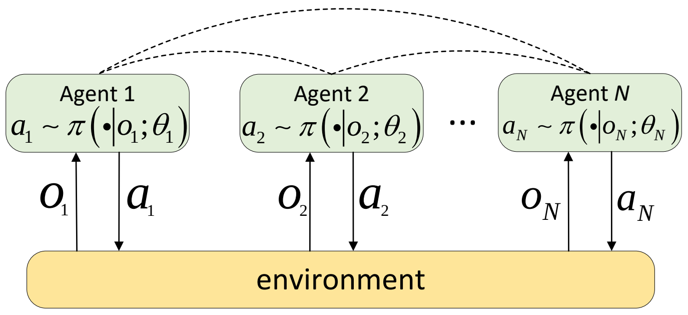

Agent 1
$a_1 \sim \pi(\bullet|o_1;\theta_1)$
Agent 2
$a_2 \sim \pi(\bullet|o_2;\theta_2)$
···
Agent N
$a_N \sim \pi(\bullet|o_N;\theta_N)$
o_1
a_1
o_2
a_2
o_N
a_N
environment

Figure 1. Schematic diagram of distributed training decentralized execution.

According to the different types of tasks, multi-agent reinforcement learning methods are divided into three categories, namely the fully competitive, fully cooperative, and mixed task [33–35]. In the fully competitive task, assuming there are two agents, the reward value R_1 of the first agent and the reward value R_2 of the second agent have the relationship $R_1 = -R_2$. The competitive task emphasizes the performance of the single agent. However in this paper, agents are fully cooperative. In the fully cooperative tasks, the rewards can be jointly maximized since all of the agents' goals are the same. The joint performance, instead of the performances of the single agents, is emphasized, which makes for a shorter computational time. The common goal of the agents in this paper is to reduce the prediction errors, which will be described in the next section.

3. Network Structure and Training Method

The overall network includes five modules, namely the feature extraction module, position encoding module, prediction module, resolution module, and communication module. Taking two agents as an example, the overall network structure is shown in Figure 2. The specific structure of each module will be introduced in the following few sections.

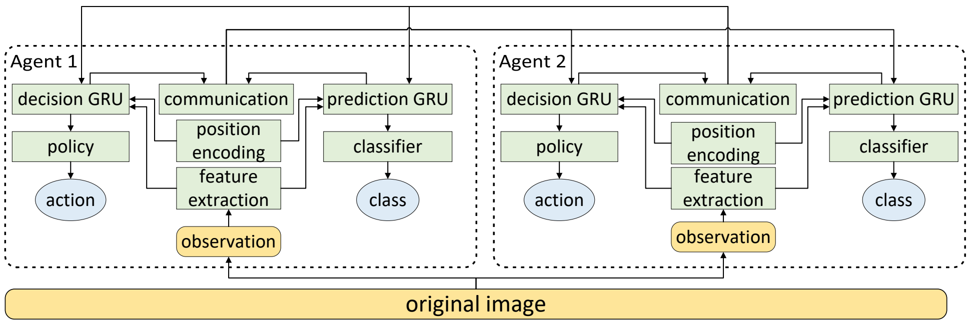


Figure 2. The overall network structure.

3.1. Feature Extraction Module

First, local observations are generated from the original image and the coordinates of the i-th agent, which can be expressed as follows:

$$o_i(t) = \mathrm{O}(I, p_i(t), w) \tag{1}$$

where I is the original image, $p_i(t)$ is the position coordinate of the i-th agent at time t, and w is the window size. The initial position coordinate of the i-th agent $p_i(0)$ is randomly generated with uniform probability. By default, the window is set to 20×20 pixels because many experiments have shown that a window of 20×20 pixels can already achieve good results. Taking two agents as an example, the schematic diagram of local observation is shown in Figure 3. As can be seen from Figure 3, the i-th agent move one step to the right and the $i+1$-th agent move one step down from time t to $t+1$. The step length here is equivalent to the window size.

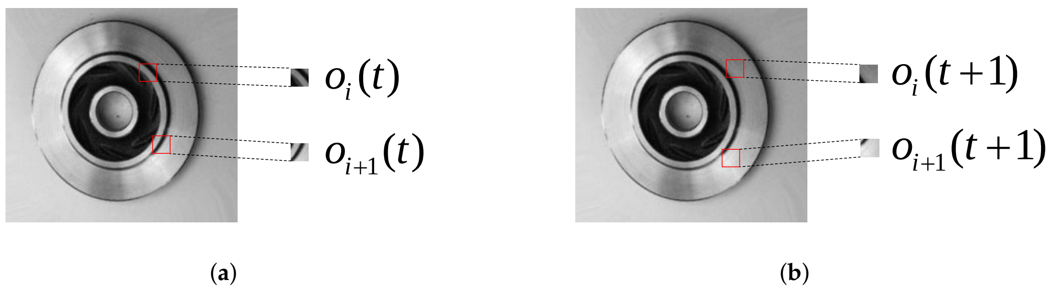


Figure 3. Schematic diagram of local observations. (**a**) The local observations of the i-th agent and the $i+1$-th agent at time t; (**b**) The local observations of the i-th agent and the $i+1$-th agent at time $t+1$.

After the local observation is generated, the feature of the local observation is extracted by the convolutional neural network. Taking the local observation of 20×20 pixels as an example, the structure of the convolutional neural network is shown in Figure 4. It includes three convolutional layers and two max-pooling layers and finally passes through the flatten layer. Using θ_1 to represent the parameters in the convolutional neural network, the feature extraction process can be expressed as follows:

$$b_i(t) = \mathrm{f}_1(o_i(t); \theta_1) \tag{2}$$

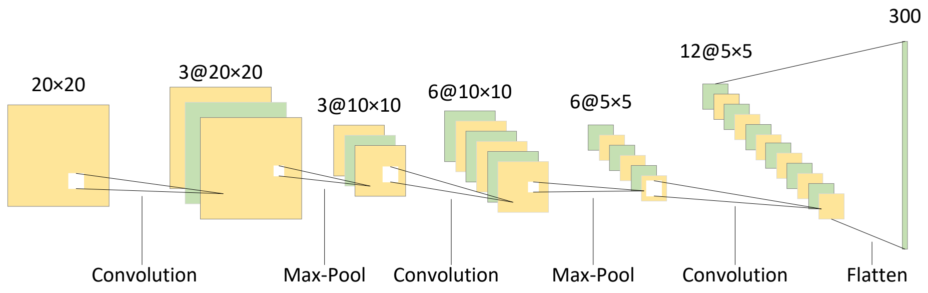


Figure 4. The structure of the convolutional neural network.

3.2. Position Encoding Module

The position encoding module is the encoding of the agent's position, which consists of a fully connected layer, and the CeLU (Continuously Differentiable Exponential Linear Units) [36] function is used as the activation function, as shown in Figure 5. The CeLU function is differentiable at all points, and gradient vanishing or gradient exploding problems can be avoided by using CeLU as the activation function. The positions of the agents are quite critical in this method, since it directly determines the performance of the prediction. If the agent is right at the location of the defect, the accuracy of the output can be much higher. The position encoding process can be expressed as follows:

$$\lambda_i(t) = \mathrm{f}_2(p_i(t); \theta_2) \tag{3}$$

where θ_2 represents the parameters of the position encoding module. The CeLU function is shown as follows:

$$\mathrm{CeLU}(x) = \max(0, x) + \min(0, \exp(x) - 1) \tag{4}$$

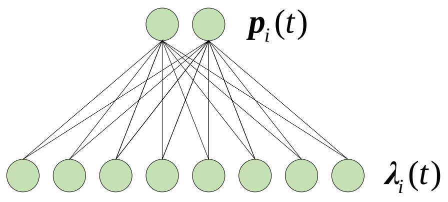


Figure 5. The position encoding module.

3.3. Prediction Module

The prediction module consists of the prediction GRU and the classifier. Each agent is assigned a prediction GRU to enable the agent to learn long-term dependencies. The function of this GRU is to aggregate the posterior information throughout the task. The structure of the GRU is shown in Figure 6. The update method of GRU is given by the following four formulas:

$$z(t) = \sigma(W_z \cdot [h(t), x(t)]) \tag{5}$$

$$r(t) = \sigma(W_r \cdot [h(t), x(t)]) \tag{6}$$

$$\tilde{h}(t) = \tanh(W \cdot [r(t) * h(t), x(t)]) \tag{7}$$

$$h(t+1) = (1 - z(t)) * h(t) + z(t) * \tilde{h}(t) \tag{8}$$

where σ represents the Sigmoid function, W_z, W_r, and W are all parameters that need to be trained.

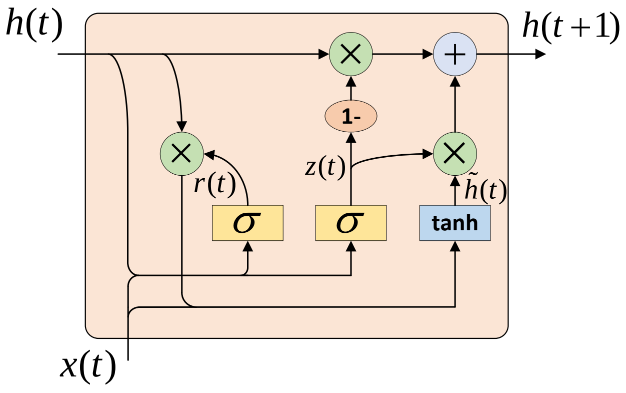


Figure 6. The structure of GRU.

Let $h_i(t)$ denote the hidden state of the i-th agent at time t. The update method of prediction GRU is shown as follows:

$$h_i(t+1) = \mathrm{f}_3(h_i(t), x_i(t); \theta_3) \tag{9}$$

where θ_3 represents the parameters in the prediction GRU, and $x_i(t)$ consists of three parts: the features of local observations of the i-th agent, the average message from other agents and the position encoding of the i-th agent. The classifier is composed of two fully connected layers, and CeLU is used as the activation function, which can be expressed as follows:

$$q_i = \mathrm{f}_4(h_i(T); \theta_4) \tag{10}$$

where T represents the time step of each episode, $h_i(T)$ represents the hidden state of the prediction GRU at time T, θ_4 represents the parameters of the classifier and q_i represents the prediction category. The final image prediction is made jointly by all agents, which can be expressed as follows:

$$q_c = \operatorname{argmax}\left(\text{Softmax}\,\frac{1}{N}\sum_{i=1}^{N} q_i\right) \tag{11}$$

3.4. Resolution Module

The resolution module consists of the decision GRU and the policy network. Note that the decisions of the agents not only depend on the current observations, but also on their past observations and states, so the GRU model is suitable for making decisions based on the current and the past information. Each agent is assigned a decision GRU whose structure is the same as the prediction GRU, but with slightly different inputs, which is updated as follows:

$$\hat{h}_i(t+1) = \mathrm{f}_5\left(\hat{h}_i(t), x_i(t); \theta_5\right) \tag{12}$$

where $\hat{h}_i(t)$ is the hidden state of the decision GRU of the i-th agent at time t, θ_5 is the parameter in the decision GRU, and then we input $\hat{h}_i(t+1)$ into the policy network to obtain the action at time $t+1$. The process can be expressed as follows:

$$a_i(t+1) = \mathrm{f}_6\left(\hat{h}_i(t+1); \theta_6\right) \tag{13}$$

where θ_6 represents the parameters of the policy network, which is constructed by two fully connected layers, using CeLU as the activation function, and finally passing through the softmax layer, then the position $p_i(t+1)$ at time $t+1$ is generated from the action $a_i(t+1)$ of the i-th agent at time $t+1$ and the position $p_i(t)$ at time t, which can be expressed as follows:

$$p_i(t+1) = \mathrm{g}(p_i(t), a_i(t+1)) \tag{14}$$

3.5. Communication Module

Each agent can obtain messages from other agents. The communication module is constructed using CeLU as the activation function by two fully connected layers. The communication process can be expressed as follows:

$$m_i(t) = \mathrm{f}_7\left(h_i(t), \hat{h}_i(t); \theta_7\right) \tag{15}$$

where $h_i(t)$ and $\hat{h}_i(t)$ come from the prediction GRU and decision GRU, respectively, and θ_7 represents the parameters of the communication module. So far, $x_i(t)$ can be expanded as follows:

$$x_i(t) = (b_i(t), \overline{m}_i(t), \lambda_i(t)) \tag{16}$$

where $\overline{m}_i(t)$ is the average message obtained by other agents, which can be given by the following equation:

$$\overline{m}_i(t) = \frac{1}{N-1}\sum_{j\neq i} m_j(t) \tag{17}$$

where $m_j(t)$ represents the message of other agents except the i-th agent.

3.6. Prediction Process and Training Method

The overall prediction process is shown in Algorithm 1.

In this paper, the stochastic gradient descent method is used to update the network parameters, and $\Theta = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6, \theta_7]$ represents the parameters defined by Algorithm 1. Let $\mathcal{T}$ denote all possible trajectories, and let τ indicate the trajectory sampled from $\mathcal{T}$. Taking the negative of the error between the actual label and the predicted label as the reward value of this trajectory, it can be expressed as follows:

$$r_\tau = -\mathrm{L}(q_\mathrm{c} - \hat{q}) \tag{18}$$

where $\hat{q}$ represents the actual label, and the mean squared error loss function is used to replace L. The goal is to maximize the expected reward value J, which can be expressed as follows:

$$J = \mathbb{E}(r_\tau) = \sum_{\tau\in\mathcal{T}} p_\tau r_\tau \tag{19}$$

where p_τ represents the probability of the occurrence of this trajectory, which is expanded as follows:

$$p_\tau = \prod_{t=0}^{T} \pi\left(a^t \mid s^t\right) = \prod_{t=0}^{T}\prod_{i=1}^{N} \pi_i\left(a_i^t \mid o_i^t\right) \tag{20}$$

To simplify the formula, denote $a_i(t)$ as a_i^t, and denote $o_i(t)$ as o_i^t. Then we need to calculate the gradient of J, which is expanded as follows:

$$\begin{aligned}\nabla_\Theta J &= \sum_{\tau\in\mathcal{T}} r_\tau \nabla_\Theta p_\tau + p_\tau \nabla_\Theta r_\tau \\ &= \sum_{\tau\in\mathcal{T}} p_\tau \nabla_\Theta (\ln p_\tau) r_\tau + p_\tau \nabla_\Theta r_\tau \\ &= \mathbb{E}(\nabla_\Theta(\ln p_\tau) r_\tau + \nabla_\Theta r_\tau)\end{aligned} \tag{21}$$

where r_τ is related to the parameter Θ, so the gradient must be calculated for r_τ. Because it is not possible to traverse all trajectories, a Monte Carlo method is used to estimate J by sampling multiple times. Suppose C trajectories are sampled to estimate J, then the gradient of the estimated value can be expressed as follows:

$$\nabla_\Theta \hat{J} = \frac{1}{C}\sum_{k=1}^{C} \nabla_\Theta(\ln p_k) r_k + \nabla_\Theta r_k \tag{22}$$

where $\nabla_\Theta \hat{J}$ is an unbiased estimate of $\nabla_\Theta J$, which is as follows:

$$\mathbb{E}\left(\nabla_\Theta \hat{J}\right) = \nabla_\Theta J \tag{23}$$

Algorithm 1: Multi-agent prediction of image classes

```
   Input: original image $I \in R^{n\times n}$
   Output: image class $q_c$
1  Initialize the position of each agent;
2  for $t \leftarrow 0$ to $T$ do
3      for $i \leftarrow 0$ to $N$ do
4          $o_i(t) = \mathrm{O}(I, p_i(t), w)$;
5          $b_i(t) = \mathrm{f}_1(o_i(t); \theta_1)$;
6          $\overline{m}_i(t) = \frac{1}{N-1}\sum_{j\neq i} m_j(t)$;
7          $\lambda_i(t) = \mathrm{f}_2(p_i(t); \theta_2)$;
8          $x_i(t) = (b_i(t), \overline{m}_i(t), \lambda_i(t))$;
9          $h_i(t+1) = \mathrm{f}_3(h_i(t), x_i(t); \theta_3)$;
10         $\hat{h}_i(t+1) = \mathrm{f}_5\left(\hat{h}_i(t), x_i(t); \theta_5\right)$;
11         $m_i(t+1) = \mathrm{f}_7\left(h_i(t+1), \hat{h}_i(t+1); \theta_7\right)$;
12         $a_i(t+1) = \mathrm{f}_6\left(\hat{h}_i(t+1); \theta_6\right)$;
13         $p_i(t+1) = \mathrm{g}(p_i(t), a_i(t+1))$;
14     end
15 end
16 for $i \leftarrow 0$ to $N$ do
17     $q_i = \mathrm{f}_4(h_i(T); \theta_4)$;
18 end
19 $q_c = \mathrm{argmax}\left(\mathrm{Softmax}\ \frac{1}{N}\sum_{i=1}^{N} q_i\right)$;
```

Therefore, the gradient $\nabla_\Theta \hat{J}$ obtained according to Algorithm 1 converges to the actual gradient $\nabla_\Theta J$ by probability, so the convergence of the Algorithm 1 is guaranteed.

4. Experiment Results and Discussion

4.1. Dataset and Setups

In terms of hardware, the machine memory is 16GB, the graphics card is NVIDIA TITAN Xp, the video memory is 12 GB, the CPU is Intel Xeon E5-2680 v4, and the clock speed is 2.4 GHz. In terms of software, the Pytorch 1.8.1 framework is used, the programming language is Python3.8 and the CUDA version is 11.1. The size of each image is 300×300 pixels. The training dataset contains 2875 non-defective casting images and 3758 defective casting images; the test dataset includes 262 non-defective casting images and 453 defective casting images.

In this data set, the casting defects can be mainly divided into two categories: open holes and casting fins. The defects of open holes can be caused by trapped air when the metal is poured into the mold. The defects of the casting fins show up when some extra materials are attached to the edges of the casting. Examples of both type of the defects are shown in Figure 7. However, this paper only studies the surface defects of the castings, which can be detected by visual signals.

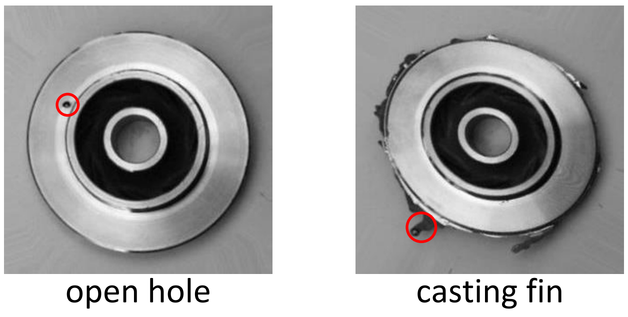


Figure 7. The examples of the open hole defect and the casting fin defect. The red circles are manually marked.

There are four choices for each agent's action, which are up, down, left, and right, each time moving a window size. The position of each agent is represented by two numbers, which represent the coordinates of the agent in the original image. If the agent's action is unavailable, the position will not change. For example, if the agent's local observation is already at the far right of the original image at time t, but the action at time $t+1$ is to move to the right, the agent's position remains unchanged. By default, 10 agents are constructed, the time step of each episode is 5, the local observation of each agent is 20×20 pixels, the number of trajectory sampling is three times and the learning rate is 0.002. The same preprocessing was performed on the dataset for each experiment. Compare our method with GhostNet [37], MobileNetV3 [38], Res2Net [39], and ShuffleNetV2 [40]. This paper uses the $F1$-score as the performance indicator. The $F1$-score is the harmonic mean of precision and recall, which can be described as follows:

$$F1 = 2\frac{precison * recall}{precison + recall} \tag{24}$$

4.2. Performance Analysis under Different Parameters

Under the default settings, the algorithm is run for 200 epochs, as shown in Figure 8. As can be seen from Table 1, the top-1 $F1$-score in the test dataset can reach 0.95551, and the prediction time of each epoch is only 1.84 s. To better evaluate the proposed algorithm, a comparative experiment is designed for four parameters: the time step T of each episode, the number of agents N, the number of trajectory sampling times C and the window size w. Changes are made each time based on the default settings.

To better show the moving process of agents, taking $T = 3$ as an example, the moving process of 10 agents is shown in Figure 9. It can be seen that the two agents on the far left did not move in the second step because, at this time, the agent's action was to move to the left, but it was unable to move to the left.

First, we verify the impact of the time step T of each episode on the performance of the algorithm, and modify the T value based on the default setting, as shown in Figure 10. It can be seen that as the T value increases, the algorithm converges faster. As can be seen from Table 1, with the T value increases, the top-1 $F1$-score also increases. When $T = 9$, the average training time per epoch increases by 147% compared to when $T = 3$, but the average test time per epoch only rises by 58%. The change in T value is more of an impact on the training time.

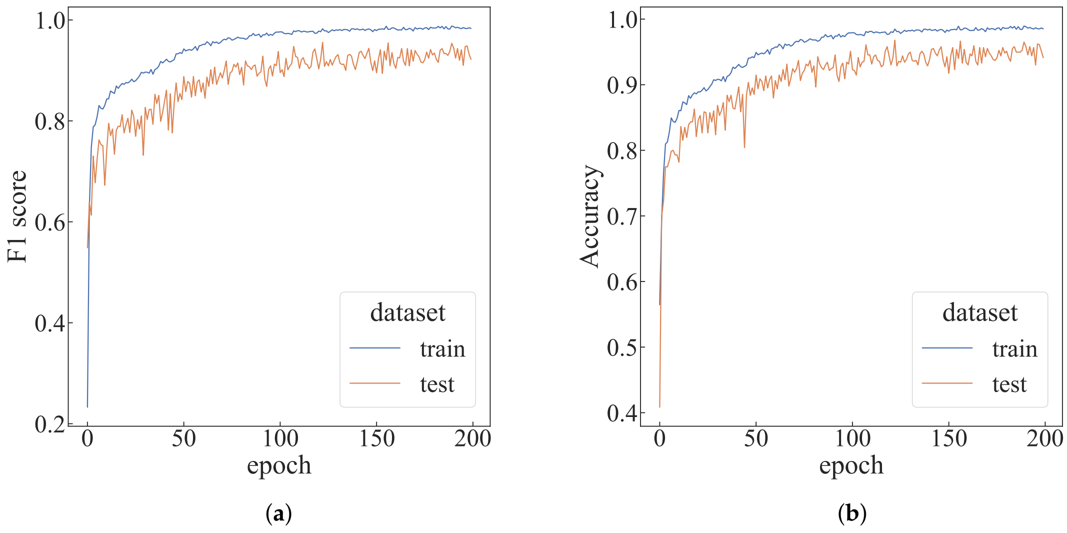


Figure 8. The performance of default settings. (**a**) $F1$-score of default settings; (**b**) accuracy of the default settings.

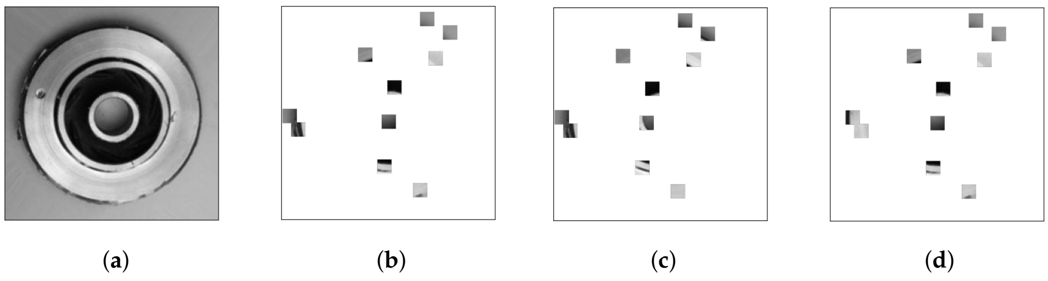


Figure 9. The moving process of agents. (**a**) original image; (**b**) step = 1; (**c**) step = 2; (**d**) step = 3.

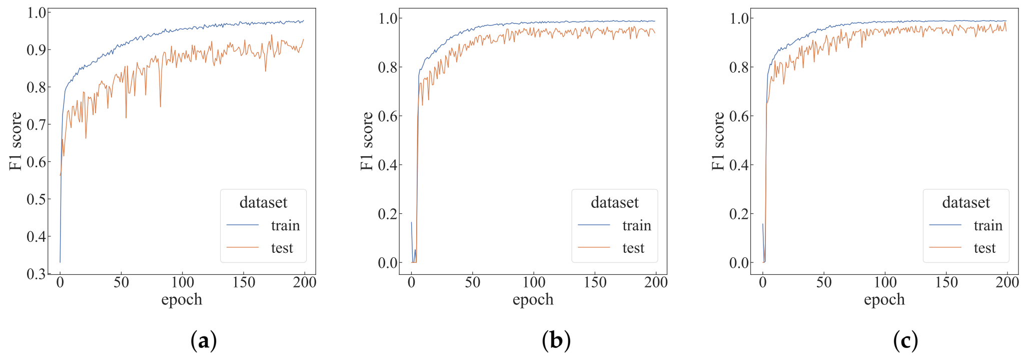
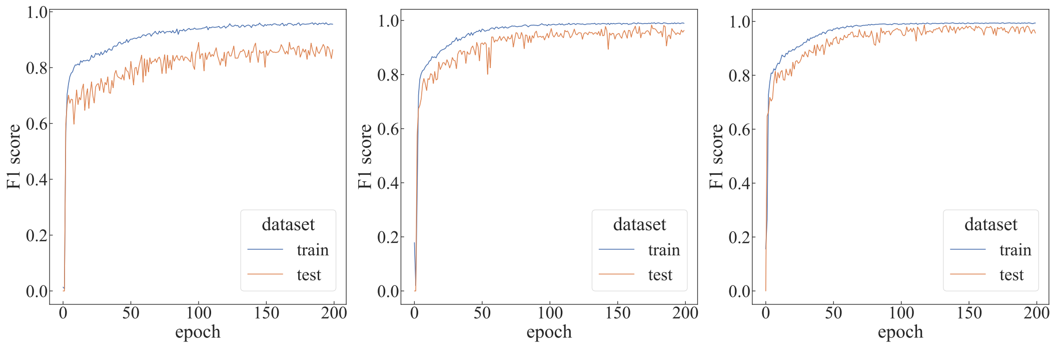


Figure 10. Performance comparison with different time steps. (**a**) $T = 3$; (**b**) $T = 7$; (**c**) $T = 9$.

By modifying the number N of agents based on the default settings, the change in the $F1$-score is shown in Figure 11. It can be seen that when the number of agents is increased to 15, the curve of the $F1$-score changes significantly and can converge faster. As can be seen from Table 1, for each additional five agents, the average training time per epoch will increase by about 15%, and the average test time per epoch will increase by about 10%. When the number of agents is 20, the top-1 $F1$-score in the test dataset can reach 0.98677. Compared with the case of $N = 5$, the number of agents has become four times the original, but the average training time per epoch only increases by 47.7%, and the average test time per epoch increased by 33%.

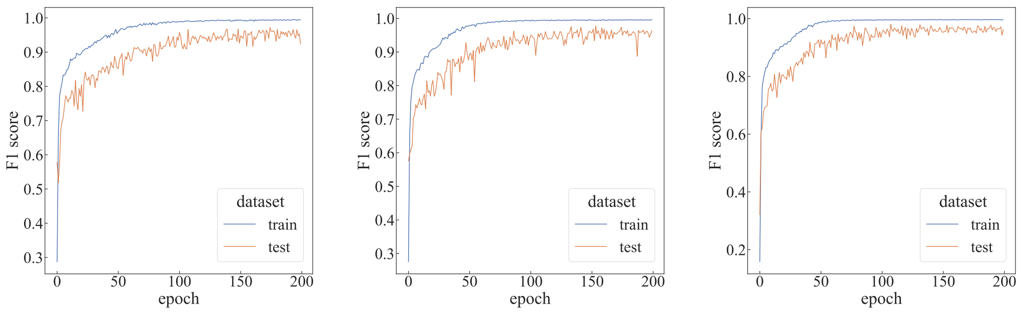


(**a**) (**b**) (**c**)

Figure 11. Performance comparison with different numbers of agents. (**a**) $N = 5$; (**b**) $N = 15$; (**c**) $N = 20$.

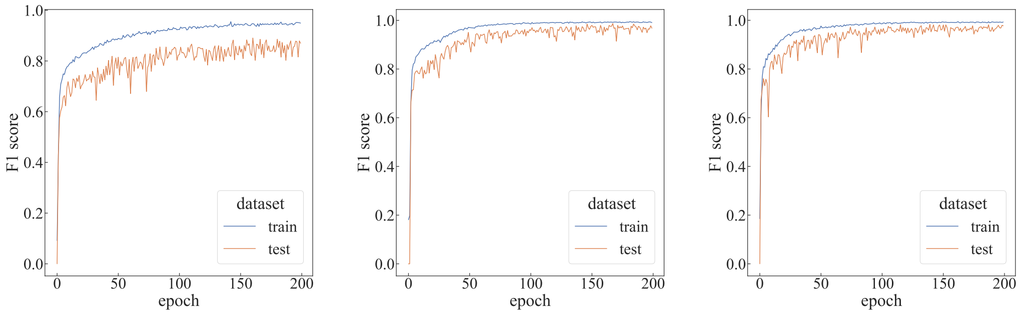
Modifying the number of trajectory samples C based on the default settings, the change in the $F1$-score is shown in Figure 12. As can be seen from the figure, when $C = 9$, the $F1$-score of the training dataset is relatively stable in the later stage. It can be seen from Table 1 that the change of the C value mainly affects the training time, and has little effect on the test time.

Modifying the window size w based on the default settings, the change in the $F1$-score is shown in Figure 13. As can be seen from Figure 13, when w = 10, the $F1$-score of the training dataset is lower, and the convergence speed is also accelerated as the window size increases. It can be seen from Table 1 that the change of window size has less impact on the training and test time, but when the window size becomes too small, the $F1$-score of the test dataset drops a lot.



(**a**) (**b**) (**c**)

Figure 12. Performance comparison with different trajectory sampling times. (**a**) $C = 5$; (**b**) $C = 7$; (**c**) $C = 9$.



(**a**) (**b**) (**c**)

Figure 13. Performance comparison with different window sizes. (**a**) $w = 10$; (**b**) $w = 30$; (**c**) $w = 40$.

In Figure 14, the result of a non-defective case and two defective cases are given. In the left sub-figure, no agent has detected any defects. In the middle sub-figure of Figure 14, a casting fin defect is detected by the agent, whose observation window is marked in green. In the right sub-figure of Figure 14, a open hole defect is also detected by the agent, whose observation window is marked in green. All the observation windows of the normal agents are marked in red.

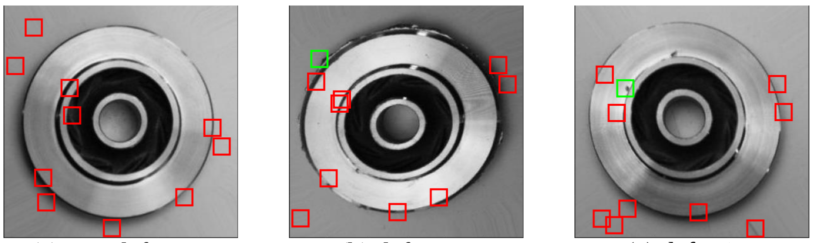

(a) non-defective (98.3%) (b) defective (99.7%) (c) defective (99.9%)

Figure 14. The prediction results of a defective casting and two non-defective castings. (**a**) A non-defective case, whose probability of being non-defective is 98.3%; (**b**) A defective casting case (with fins), whose probability of being defective is 99.7%; (**c**) A defective casting case (with open holes), whose probability of being defective is 99.9%. The observation windows of the normal agents are marked in red, and the observation windows of the defects detected agents are marked in green.

Table 1. Performance comparison with different parameters.

Parameters	**Top-1 Training $F1$-Score**	**Top-1 Test $F1$-Score**	**Average Training Time Per Epoch (s)**	**Average Test Time Per Epoch (s)**
$N = 5, T = 5, w = 20, C = 3$	0.96077	0.89051	39.58	1.68
$N = 10, T = 3, w = 20, C = 3$	0.97834	0.93985	30.62	1.67
$N = 10, T = 5, w = 10, C = 3$	0.95514	0.89051	44.80	1.84
$N = 10, T = 5, w = 20, C = 3$	0.98789	0.95551	44.85	1.84
$N = 10, T = 5, w = 20, C = 5$	0.99515	0.97164	67.28	1.85
$N = 10, T = 5, w = 20, C = 7$	0.99653	0.97692	94.18	1.83
$N = 10, T = 5, w = 20, C = 9$	0.99688	0.98099	117.51	1.86
$N = 10, T = 5, w = 30, C = 3$	0.99463	0.98667	44.89	1.85
$N = 10, T = 5, w = 40, C = 3$	0.99446	0.98305	46.25	1.88
$N = 10, T = 7, w = 20, C = 3$	0.99084	0.96629	59.79	2.41
$N = 10, T = 9, w = 20, C = 3$	0.99084	0.98299	75.76	2.64
$N = 15, T = 5, w = 20, C = 3$	0.99153	0.98292	52.58	2.05
$N = 20, T = 5, w = 20, C = 3$	0.99498	0.98677	58.46	2.24

4.3. Comparative Experiments

The GhostNet, MobileNetV3, Res2Net, and ShuffleNetV2 models are selected for comparative experiments. All four networks use the cross-entropy loss function, the learning rate is set to 0.0003, the batch size is set to 32, and each model is trained for 50 epochs. The $F1$-scores of each network are shown in Figure 15.

The top-1 $F1$-score of each network and the average time of per epoch are shown in Table 2. It can be seen from the table that the average test time of each epoch of the comparison model is at least 10.90 s, while the algorithm proposed in this paper only needs 1.84 s under the default setting. When the number of agents is 20, the top-1 $F1$-score of this algorithm in the test dataset reaches 0.98677, which is only 0.00754 lower than GhostNet, but the average test running time per epoch is only 20.55% of GhostNet.

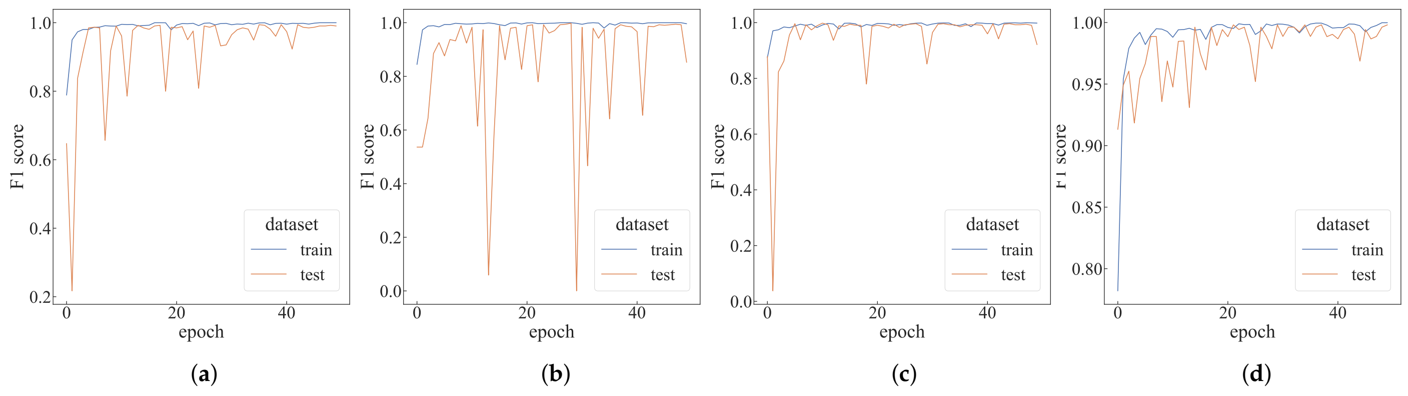


Figure 15. The *F*1-score of different models. (**a**) GhostNet 1.0×; (**b**) MobileNetV3 Small; (**c**) Res2Net-50; (**d**) ShuffleNetV2 1.0×.

Table 2. Performance comparison of the different models.

Models	Top-1 Training *F*1-Score	Top-1 Test *F*1-Score	Average Training Time Per Epoch (s)	Average Test Time Per Epoch (s)
GhostNet 1.0×	1	0.99431	110.93	10.90
MobileNetV3 Small	1	0.99809	129.54	13.59
Res2Net-50	0.99965	0.99809	185.67	12.81
ShuffleNetV2 1.0×	0.99983	0.99809	126.63	13.64

5. Conclusions and Future Works

This paper uses multiple agents to jointly predict the image category, significantly reducing the prediction time and the total calculation amount. This method can be applied to devices with weak computing power, since the input of each device is only a tiny part of the original image. The joint operation of multiple agents not only ensures accuracy but also reduces the computational burden of each device. With the increasing size of images today, a 2 K or even 4 K image requires considerable computing resources on traditional convolutional neural networks for classification, so the method in this paper has great potential for large-size image classification. For the casting dataset, compared with the traditional convolutional neural network GhostNet, the prediction time of the method in this paper is only one-fifth of the original, which has made significant progress in saving computing resources and reducing the defects detection time.

The proposed method in this work can be extended in several aspects in future works. First, the way of communication between the agents can be further investigated, by which the observation of each single agent can be better utilized. Second, instead of being a fixed value, the step size can be determined in an adaptive manner in the future methods. Third, the sizes or even the shapes of the observation windows can also be changed in the process with respect to the agents' states. All these possible future works can make further improvements on the prediction accuracy of the casting defects detection or the corresponding computational burdens.

Author Contributions: Conceptualization, Y.Z.; methodology, C.L. and Y.Z.; formal analysis, C.L. and Y.Z.; writing—original draft preparation, C.L. and Y.Z.; writing—review and editing, C.L. and S.M.; visualization, C.L. and S.M. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Young Scientists Fund of the National Natural Science Foundation of China (grant numbers: 61803337).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The author would like to thank the editors and anonymous reviewers for their suggestions, which helped improved the quality of the paper.

Conflicts of Interest: The authors declare that there is no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Jiang, X.; Feng Wang, X.; Chen, D. Research on Defect Detection of Castings Based on Deep Residual Network. In Proceedings of the 2018 IEEE 11th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), Beijing, China, 13–15 October 2018; pp. 1–6.
2. Du, W.; Shen, H.; Fu, J.; Zhang, G.; He, Q. Approaches for improvement of the X-ray image defect detection of automobile casting aluminum parts based on deep learning. *NDT Int.* **2019**, *107*, 102144. [CrossRef]
3. Duan, L.; Yang, K.; Ruan, L. Research on automatic recognition of casting defects based on deep learning. *IEEE Access* **2020**, *9*, 12209–12216. [CrossRef]
4. Ferguson, M.; Ak, R.; Lee, Y.T.T.; Law, K.H. Automatic localization of casting defects with convolutional neural networks. In Proceedings of the 2017 IEEE International Conference on Big Data (Big Data), Boston, MA, USA, 11–14 December 2017; pp. 1726–1735.
5. Lee, J.H.; Oh, H.M.; Kim, M.Y. Deep learning based 3D defect detection system using photometric stereo illumination. In Proceedings of the 2019 IEEE International Conference on Artificial Intelligence in Information and Communication (ICAIIC), Okinawa, Japan, 11–13 February 2019; pp. 484–487.
6. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2012; Volume 25.
7. Simonyan, K.; Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv* **2014**, arXiv:1409.1556.
8. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 770–778.
9. Chen, L.; Li, S.; Bai, Q.; Yang, J.; Jiang, S.; Miao, Y. Review of Image Classification Algorithms Based on Convolutional Neural Networks. *Remote Sens.* **2021**, *13*, 4712. [CrossRef]
10. Rawat, W.; Wang, Z. Deep convolutional neural networks for image classification: A comprehensive review. *Neural Comput.* **2017**, *29*, 2352–2449. [CrossRef] [PubMed]
11. Gupta, S.K. Reinforcement based learning on classification task could yield better generalization and adversarial accuracy. *arXiv* **2020**, arXiv:2012.04353.
12. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 1999; Volume 12.
13. Mansour, R.F.; Escorcia-Gutierrez, J.; Gamarra, M.; Villanueva, J.A.; Leal, N. Intelligent video anomaly detection and classification using faster RCNN with deep reinforcement learning model. *Image Vis. Comput.* **2021**, *112*, 104229. [CrossRef]
14. Ren, S.; He, K.; Girshick, R.; Sun, J. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2015; Volume 28.
15. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
16. Zhao, D.; Chen, Y.; Lv, L. Deep reinforcement learning with visual attention for vehicle classification. *IEEE Trans. Cogn. Dev. Syst.* **2016**, *9*, 356–367. [CrossRef]
17. Furuta, R.; Inoue, N.; Yamasaki, T. Pixelrl: Fully convolutional network with reinforcement learning for image processing. *IEEE Trans. Multimed.* **2019**, *22*, 1704–1719. [CrossRef]
18. Mnih, V.; Badia, A.P.; Mirza, M.; Graves, A.; Lillicrap, T.; Harley, T.; Silver, D.; Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, New York, NY, USA, 20–22 June 2016; pp. 1928–1937.
19. Fan, L.; Zhang, F.; Fan, H.; Zhang, C. Brief review of image denoising techniques. *Vis. Comput. Ind. Biomed. Art* **2019**, *2*, 1–12. [CrossRef] [PubMed]
20. Park, J.; Lee, J.Y.; Yoo, D.; Kweon, I.S. Distort-and-recover: Color enhancement using deep reinforcement learning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, USA, 18–22 June 2018; pp. 5928–5936.
21. Mousavi, H.K.; Nazari, M.; Takáč, M.; Motee, N. Multi-agent image classification via reinforcement learning. In Proceedings of the 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Macau, China, 3–8 November 2019; pp. 5020–5027.
22. Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; Bengio, Y. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv* **2014**, arXiv:1406.1078.
23. Littman, M.L. Markov games as a framework for multi-agent reinforcement learning. In *Machine Learning Proceedings 1994*; Morgan Kaufmann: Burlington, MA, USA, 1994; pp. 157–163.

24. Tian, Y.; Wang, Y.; Yu, T.; Sra, S. Online learning in unknown markov games. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual Event, 13–14 August 2021; pp. 10279–10288.
25. Pajarinen, J.; Peltonen, J. Periodic finite state controllers for efficient POMDP and DEC-POMDP planning. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2011; Volume 24.
26. Aşık, O.; Akın, H.L. Solving multi-agent decision problems modeled as dec-pomdp: A robot soccer case study. In *Robot Soccer World Cup*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 130–140.
27. Kumar, A.; Mostafa, H.; Zilberstein, S. Dual formulations for optimizing Dec-POMDP controllers. In Proceedings of the Twenty-Sixth International Conference on Automated Planning and Scheduling, London, UK, 12–17 June 2016.
28. Gronauer, S.; Diepold, K. Multi-agent deep reinforcement learning: A survey. *Artif. Intell. Rev.* **2022**, *55*, 895–943. [CrossRef]
29. Li, M.L.; Chen, S.; Chen, J. Adaptive learning: A new decentralized reinforcement learning approach for cooperative multiagent systems. *IEEE Access* **2020**, *8*, 99404–99421. [CrossRef]
30. Zimmer, M.; Glanois, C.; Siddique, U.; Weng, P. Learning fair policies in decentralized cooperative multi-agent reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Virtual Event, 18–24 July 2021; pp. 12967–12978.
31. Lowe, R.; Wu, Y.I.; Tamar, A.; Harb, J.; Pieter Abbeel, O.; Mordatch, I. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*; MIT Press: Cambridge, MA, USA, 2017; Volume 30.
32. Rashid, T.; Samvelyan, M.; Schroeder, C.; Farquhar, G.; Foerster, J.; Whiteson, S. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In Proceedings of the International Conference on Machine Learning, PMLR, Stockholm, Sweden, 10–15 July 2018; pp. 4295–4304.
33. Zhang, Q.; Zhao, D.; Zhu, Y. Data-driven adaptive dynamic programming for continuous-time fully cooperative games with partially constrained inputs. *Neurocomputing* **2017**, *238*, 377–386. [CrossRef]
34. Park, Y.J.; Cho, Y.S.; Kim, S.B. Multi-agent reinforcement learning with approximate model learning for competitive games. *PLoS ONE* **2019**, *14*, e0222215. [CrossRef] [PubMed]
35. He, J.; Li, Y.; Li, H.; Tong, H.; Yuan, Z.; Yang, X.; Huang, W. Application of game theory in integrated energy system systems: A review. *IEEE Access* **2020**, *8*, 93380–93397. [CrossRef]
36. Barron, J.T. Continuously differentiable exponential linear units. *arXiv* **2017**, arXiv:1704.07483.
37. Han, K.; Wang, Y.; Tian, Q.; Guo, J.; Xu, C.; Xu, C. Ghostnet: More features from cheap operations. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, Seattle, WA, USA, 14–19 June 2020; pp. 1580–1589.
38. Howard, A.; Sandler, M.; Chu, G.; Chen, L.C.; Chen, B.; Tan, M.; Wang, W.; Zhu, Y.; Pang, R.; Vasudevan, V.; et al. Searching for mobilenetv3. In Proceedings of the IEEE/CVF International Conference on Computer Vision, Seoul, Korea, 27–28 October 2019; pp. 1314–1324.
39. Gao, S.H.; Cheng, M.M.; Zhao, K.; Zhang, X.Y.; Yang, M.H.; Torr, P. Res2net: A new multi-scale backbone architecture. *IEEE Trans. Pattern Anal. Mach. Intell.* **2019**, *43*, 652–662. [CrossRef] [PubMed]
40. Ma, N.; Zhang, X.; Zheng, H.T.; Sun, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Proceedings of the European conference on computer vision (ECCV), Munich, Germany, 8–14 September 2018; pp. 116–131.