Efficient Imitation Learning for Game AI

Chao Huang, Like Zhang, Yanqing Jing and Dajun Zhou
Turing Lab, Tencent Inc.
ChengDu, China
andrehuang, likezhang, frogjing, dijinzhou@tencent.com

***Abstract*—Automation testing is an important approach for bug detection and analysis of software applications, especially for computer games. This paper proposes an efficient imitation method to learn game strategy from a small set of manually recorded game samples, which only takes one hour to complete training. This work can be divided into four steps. Firstly, we collect a set of manually recorded data composed of image frames and user actions. The discriminative region is extracted from image to eliminate noise. Then, data alignment is applied to solve the problem of action delay. Due to the large variation of image samples from different classes, data resampling is performed to avoid bias. Finally, these samples are fed into a fast and lightweight network with LSTM structure, which is designed to boost speed of prediction. Our work discards the dependence of game internal interface and performs well in real time with CPU, which has been verified in a variety of commercial games.**

***Index Terms*—imitation learning, game AI, fast and lightweight network, class balance**

I. Introduction

Reinforcement learning and imitation learning are two widely-used methods to learn game AI [1]–[4]. For the former, AI model is required to interact with game environment to get current state, which is fed into neural network. Then, the network outputs the corresponding action and gets the reward from environment. The goal is to maximize the expected rewards. Comparing with reinforce learning which has no prior knowledge, imitation learning requires a set of manually recorded data. The goal of imitation learning is to output action similar with human behaviors.

There exist a variety of methods focusing on reinforcement learning [5]–[7]. For instance, Deep Q-Learning (DQN) [6] is proposed to construct replay memory composed of state, action and reward, which are fed into a network to predict score for each action based on input state. The action with maximum score is chosen as the optimal move. Actor-Critic method [8] uses actor network to output action and applies critic network to calculate the advantage of action. Proximal Policy Optimization (PPO) [9] uses importance sampling method to add weight for each sample. The advantage of reinforcement learning is that AI can handle abnormal situations due to the large-scale exploration. However, it is extremely time consuming to apply reinforcement learning, which is required to interact with the environment for large number of times. This problem gets worse when the back-end API that bypasses the user interface is not provided, which is common for most commercial games.

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

Imitation learning [10] [11] aims to complete a task based on expert demonstrations. There are mainly three kinds of methods: behavior clone [12], inverse reinforcement learning [13] and generative adversarial imitation learning (GAIL) [2]. For behavior clone, the training data is composed of game image and the corresponding expert action. The image is fed to deep neural network, which outputs action similar to expert. Since the trained model is not completely consistent with the manual strategy, there exists a gap between the sample distribution of the training and test set. It is easy to make a wrong decision when game AI enters a scene which has not appeared in training data. For inverse reinforcement learning, the reward function is estimated based on expert demonstrations. Then, game AI is trained based on reinforcement learning. The principle of reward function is that the expert demonstrations always get the largest reward compared with game AI. GAIL introduces generator and discriminator to learn strategy of expert. The goal of generator is to generate samples similar to expert demonstrations. While discriminator aims to distinguish between expert samples and the generated samples, which evaluates the performance of game AI. It is noted that both inverse reinforcement learning and GAIL are required to interact with the environment, which meet similar problem with reinforcement learning.

This paper proposes an efficient imitation learning method using game image to learn game strategy for specific task. Data alignment and class balance are applied to improve performance of AI in this work. There are mainly four steps to train model. Firstly, a set of expert demonstrations for specific task are collected, which are composed of images and the corresponding actions. Data alignment is employed to solve action delay, followed by preprocessing to balance sample number of different classes. Finally, we construct a deep and lightweight network with LSTM [14]–[18] to extract discriminant feature from image and output action similar to expert. During test period, the network outputs probability for each action based on current game image, we choose action with maximum probability or randomly select action based on probability distribution, which is determined by the type of game. Our work discards dependence on game interface and takes only one hour to obtain an AI model, which achieves good performance in real time with CPU. We evaluate our methods on three popular commercial games imposed of racing game, cool running game and shootout game. The good performance indicates the efficiency of the proposed method.

This paper is organized as follows. The second section

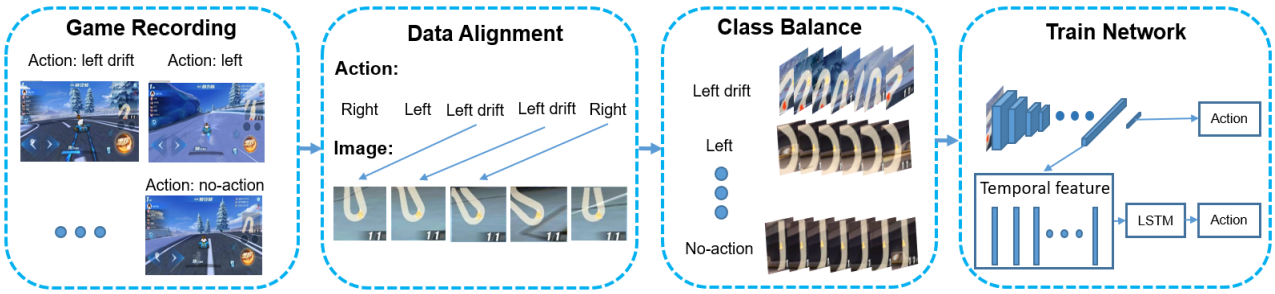

Fig. 1. Training framework of the proposed method.

describes details of the proposed method. In the third section, we introduce our dataset composed of images from three popular commercial games. The experimental setup and results are reported in the fourth section, followed by the conclusion of our work.

II. PROPOSED METHOD

Both inverse reinforcement learning and GAIL are required to interact with the game environment. while internal interface to speed up the game is not provided by most commercial games. Interacting with the game environment is time consuming. To train game AI in faster speed, we apply behavior clone method and add a series of strategies to compensate the defects of behavior clone. The training process is shown in Figure 1.

Firstly, we play game for about half an hour and record image frames and the corresponding actions. The discriminative regions are extracted to reduce the difficulty of training model. Then, data alignment is applied to solve the problem of action delay. Since the number of samples corresponding to different actions varies greatly during the recording process, data resampling is performed to ensure that the sample number of each action exceeds a threshold, which is empirically set. Finally, the adjusted samples are fed into deep network for training.

A. Game Recording

We play game in specific scene for nearly half an hour. The frequency of collecting samples is set as 10 for one second. The game buttons are set according to the game. For instance, for QQ speed[1], we use left, right and drift. For Cool Running Every Day (CRED) [2], squat and jump are used. Two examples are shown in Figure 2, where the corresponding game buttons are shown in red boxes.

Different buttons can be combined to define one action. For instance, during the recording of the QQ speed game, if the player presses the left and the drift buttons at the same time, this behavior is defined as the left drift. If the right and the drift buttons are simultaneously pressed, it is defined as the right-drift. If no game button is pressed, we define it as no-action.

[1]https://speedm.qq.com
[2]https://pao.qq.com

Fig. 2. Example of game button.

Since the region around game character is usually more important for determining action than other regions, we extract these regions as the input of network to reduce the difficulty of training. In QQ speed, CrossFire Mobile (CFM) [3] and other First-Person Shooter Game (FPS) games, radar map is a very important region, which has a high degree of relevance to the performed action. For such games, radar map is used as the input of deep network. It is noted that the agent is required to follow the fixed path in CFM for performance testing, which can be done based on radar. For games without radar maps, we select a rectangular region that contains the game character. An example of region extraction is shown in Figure 3. The region with blue box is used as the input of network.

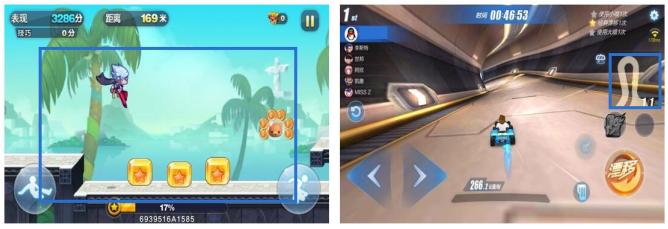

Fig. 3. Example of region extraction

B. Data Alignment

The game is still running when the action for previous frame is not provided by AI model, which induces time delay imposed of image transmission and forward operation of AI model. As shown in Figure 4, the mobile client intercepts the game image at time $t1$. Then, the image is sent to the deep network, which outputs action probability at time $t2$.

[3]https://cfm.qq.com

The action is delayed by $t2 - t1$, during which the game is still running. If the AI model is unable to predict the action of subsequent scenes in advance, it will fail to achieve good AI results.

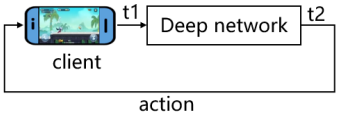

Fig. 4. Example of action delay.

To solve the problem of action delay, we align label and image based on time delay and the sampling time. For example, the delay time is 0.2 seconds and the interval between recorded adjacent frames is 0.1 second, the action of t-th frame image needs to be moved ahead two frames, as shown in Figure 5.

Fig. 5. Example of data alignment.

C. Class Balance

There exists a large variation of sample number for different actions during game recording. For instance, we define five actions in QQ speed: left, right, left drift, right drift, and no-action. Among them, the sample number of no-action is much higher than the number of samples corresponding to other actions. If these samples are directly applied to train network, the model may fail to capture feature for class with few samples. Motivated by this problem, data resampling is proposed to ensure that the number of samples for each action exceeds a pre-defined threshold. The sampling times for each class is computed as follows:

$$count_i = max(\frac{\sum_i^C N_i \times ratio_t}{N_i}, 1) \quad (1)$$

Here, $count_i$ indicates the sampling times for the i-th class, N_i is the corresponding sample number, C is the class number, $ratio_t$ is the min ratio to calculate the threshold. The default value of $ratio_t$ is set as $\frac{1}{C}$.

In behavior clone, each instance is given the same weight, which discards the advantage of the chosen action compared with other actions. To focus on more important action, we only do specific actions in necessary situation, which increases importance of these actions. For instance, player only jumps when there exists obstacle in front during recording. The sample number for specific action is usually much smaller than other actions. By performing class balance, the weights for actions with larger advantage are increased, which helps model focus on more important actions.

D. Training Deep Network

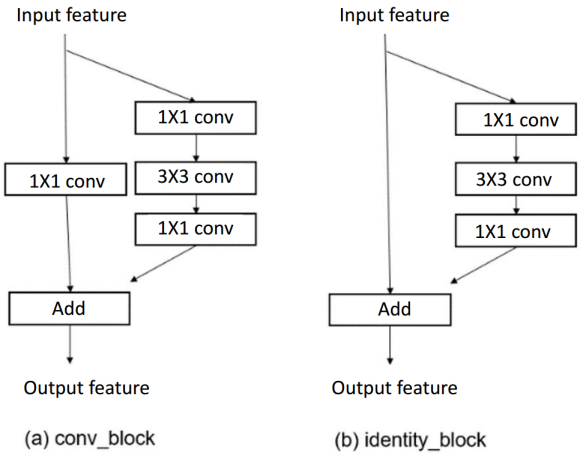

Fig. 6. The structure of two modules.

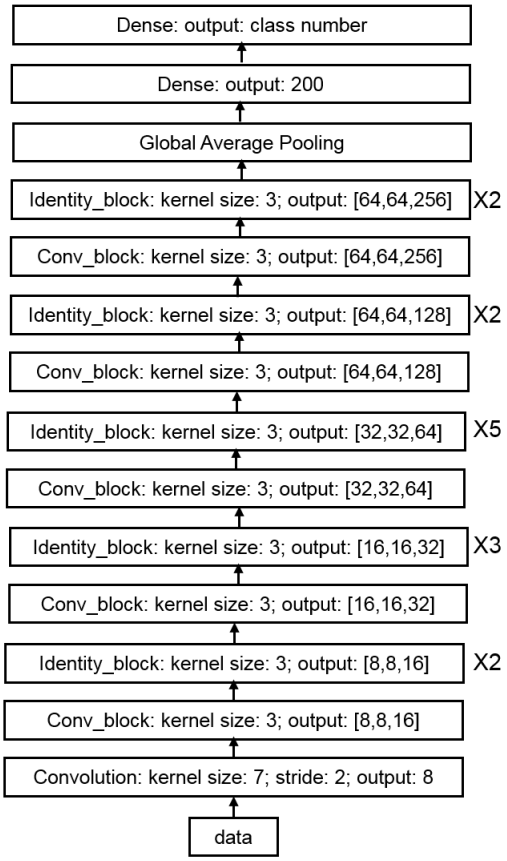

Fig. 7. Lightweight network structure.

In order to reduce the computational complexity, the extracted region is resized to 150×150 pixels. Since there exists a large variation of appearance, illumination and angle in the

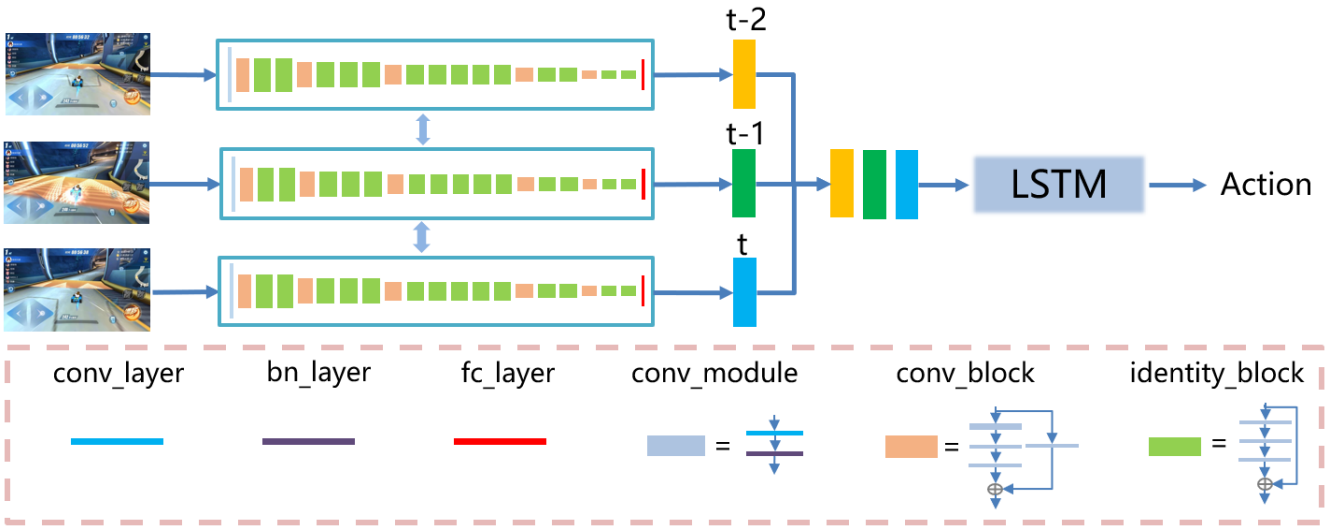

Fig. 8. LSTM network structure. Here, the block with different color represents a type of layer. We concatenate feature from the second last fully-connected layer of the lightweight network based on t-2, t-1 and t-th frames.

game, it is difficult to extract discriminative features through a simple deep network. We design a lightweight residual network that can perform about 80 frames per second using CPU. By combining with the previous features, the network can prevent gradient vanishing and boost training speed. The network is mainly composed of two modules, which are shown in Figure 6. The proposed lightweight network is built by repeating the modules. The structure of the network is shown in Figure 7, where $\times 2$ means to repeat the module twice. During training period, the cross-entropy loss [14] is applied as the objective function of the model.

To extract the temporal feature of the game, a network based on LSTM [19] is proposed. The input feature is set as the concatenated feature from the second last fully-connected layer of the lightweight network based on 5 continuous frames. The output dimension of the LSTM is set as 100, followed by a fully connected layer to predict probability of different actions. The network structure is shown in Figure 8. After 20 epochs of training, an optimized deep network with LSTM can be obtained.

During test period, we first extract important region from current game image. Then, the image region is fed to the light-weight network to obtain deep feature. Finally, the LSTM network outputs probability of each action based on the concatenated features from 5 continuous frames. To improve performance of game AI, we sum up several tricks listed as follows:

- It is recommended to set rules when manually recording games. For example, FPS games are played according to a fixed path. In QQ speed, players should avoid unnecessary action. This reduces the difficulty of training, which helps network converge quickly.
- Game AI should add reset strategy. Once the AI enters a scene that has not appeared during the recording or AI is stuck for a long time, reset method should be applied. In the QQ speed, the reset operation is to click the reset button, which can return AI to the center of the track. In FPS game, a set of the predefined actions are executed to get rid of the stuck situation.

III. DATASET CONSTRUCTION

This work has been evaluated in three commercial games: QQ speed, CRED and CFM. For each game, we record game for around half an hour to get samples composed of image and action. In QQ speed, we define five actions: left, left drift, right, right drift and no action. After resampling data to make sample number of each class higher than threshold, there are 1064 images for left, 1686 images for left drift, 1362 images for right, 1910 images for right drift and 7047 images for no-action. To decrease the difficulty of learning model, the radar map is extracted from image to train network. The examples of QQ speed are shown in Figure 9. Each row shows the examples of radar maps for one action.

For CRED, we define three actions: jump, squat and no action. After resampling, there are 1812 images for jump, 1776 images for squat, 4855 images for no-action. The region around game character is extracted to decrease noise of background. The examples are shown in Figure 10. It shows that game image is more complex than radar map.

For CFM, we record game using the fixed path for around half an hour. The example of recording is shown in Figure 11.

The moving angle is divided into 8 parts and the press action on shooting button is recorded. Similar to QQ speed, the radar map is applied as the input of network. There are 4841 images related to moving, 959 images for shooting and 2374 images for no-action. The examples for CFM are shown in Figure 12.

For each game, we randomly pick 80% samples from database as training and the rest samples are applied for testing.

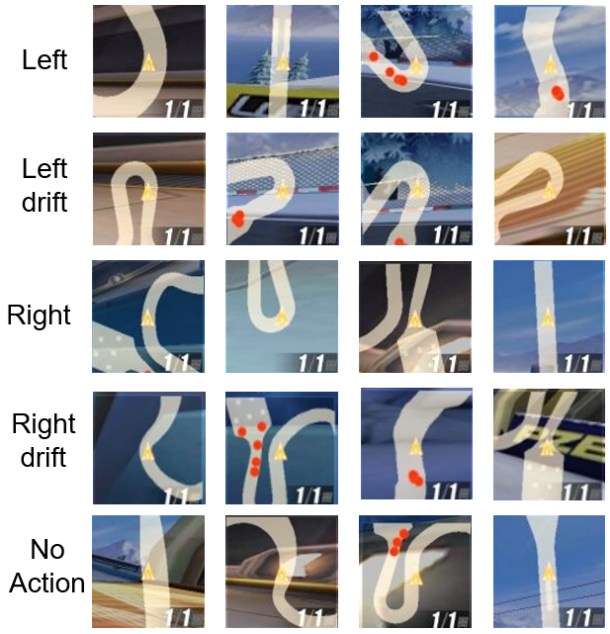

Fig. 9. Examples for QQ speed.

Fig. 10. Examples for CRED.

Fig. 11. Examples of recording for CFM.

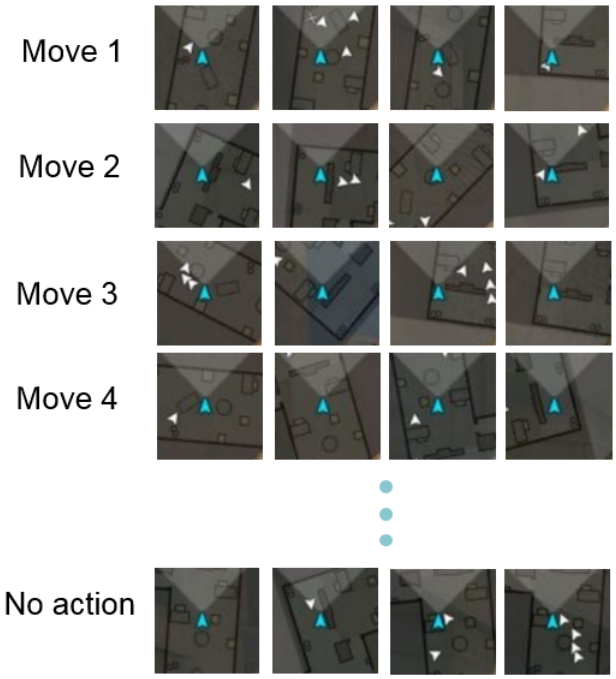

Fig. 12. Examples of CFM.

IV. EXPERIMENT

Our work is evaluated based on tensorflow and keras [20]. The learning rate is set to 0.001 and the model is trained based on Adam optimizer. During test period, the important region is extracted from current image of game. Then, the lightweight network is applied to obtain deep feature of the region. The feature from the second last fully-connected layer is used to represent image. To utilize temporal feature, we concatenate features from 5 continuous frames, which are fed to network with LSTM to output probability of each action.

In our experiment, we firstly show the classification accuracy of different network. Then, the game performance of our method and recent state-of-art reinforcement-based methods are compared to verify the efficiency of the proposed method.

A. Evaluation on Classification Accuracy

Our work is firstly evaluated in three datasets to test accuracy of behavior clone, which can be treated as a specific task of supervised learning. The proposed network is compared with VGG16 [21], which is fine-tuned using the pre-trained model based on imageNet [22]. The results are shown in Table I. It shows that the proposed methods obtain higher accuracy compared with VGG16. The reason is perhaps that the appearance of game image is significantly different from the real image. VGG16 learns the limited feature from the pre-trained model. When the recorded samples are employed to train VGG16 model with a large amount of parameters, it is easy to make model overfit.

TABLE I
CLASSIFICATION ACCURACY IN THREE DATASETS

	VGG16 [21]	lightweight network	LSTM model
QQ speed	70.3%	72.5%	73.6%
CRED	84.6%	89.9%	91.7%
CFM	66.4%	67.3%	67.9%

The experiments show that the classification accuracy for

QQ speed is 72.5% based on the proposed lightweight model. When network with LSTM is applied, the classification accuracy is improved by 1.1%. For CRED, the accuracy is 89.9% and 91.7% for lightweight network and LSTM network respectively. For CFM, the corresponding classification accuracy is 67.3% and 67.9%. This dataset is more difficult for classification due to the low action advantage in CFM, which means that multiple game strategies are suitable in the same situation.

B. Evaluation on Game Performance

To investigate the efficiency of the proposed light-weight network, we build a baseline network composed of 8 convolution layers and 2 fully-connected layers. The structure of baseline is shown in Figure 13. We evaluate this network in CRED. The accuracy of baseline is 86.1%, which is 3.8% lower than our proposed network. The average distance using baseline is 461, which is 97 less than the light-weight network. The reason is perhaps that our work extracts more discriminative feature due to the deep structure.

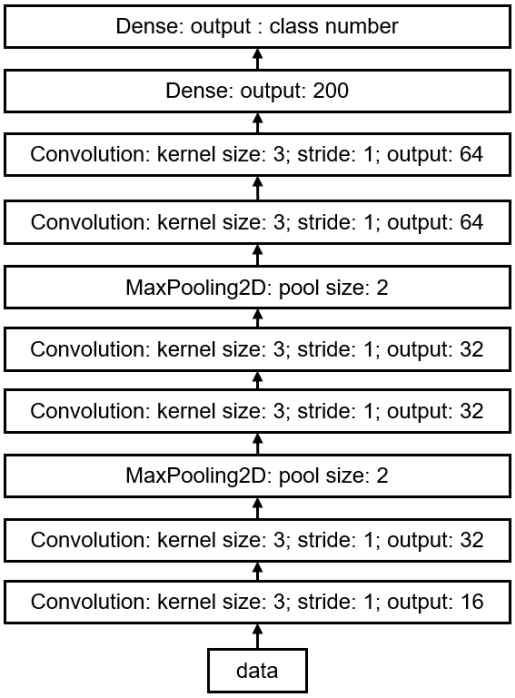

Fig. 13. Network structure of baseline.

We continue to analyze the performance in game environment for different AI model. For QQ speed and CRED, AI performs the action with maximum probability to get higher score. The frequency of action is set as 10 actions per second for these games. For CFM, the action is executed randomly based on probability distribution from AI model to explore game and avoid stuck. Since high frequency of action is not required in CFM, we set it as 2 actions per second.

To illustrate the advantage of imitation learning for commercial game without game interface, our work is compared with recent state-of-art reinforcement learning methods. Since we control character to follow the pre-designed path in CFM, it is hard to define the corresponding reward function for reinforcement method. We apply the classical DQN [6] and PPO algorithm [9] to train AI model based on radar map and game image for QQ speed and CRED respectively.

The network structure for DQN is shown in Figure 14. We take the gray images of 4 consecutive frames as the input of model. Then, value scores for all actions are calculated based on three convolution layers and two fully-connected layers. The network is constructed based on work [6], which trains model to play Atari games. While compared with commercial games, the game interface of Atari game is provided to boost training speed. Meanwhile, the state space of most Atari games is much smaller, which benefits fast reinforcement training.

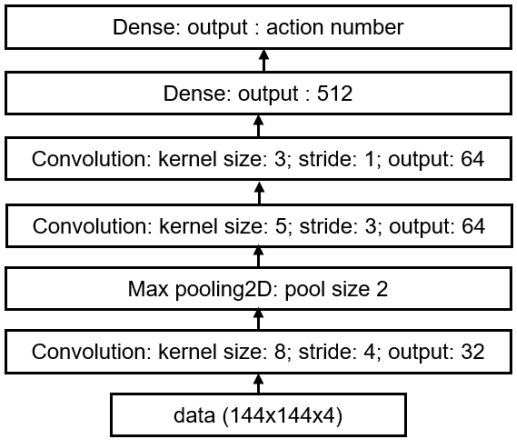

Fig. 14. Network structure for DQN.

For PPO algorithm, the radar map is fed into the pre-trained alexNet [16] to extract feature, followed by Actor and critic module to get the probability for each action and the corresponding advantage score. The action is randomly performed based on probability distribution and the network is optimized by getting larger expected reward. The structure of PPO algorithm is shown in Figure 15.

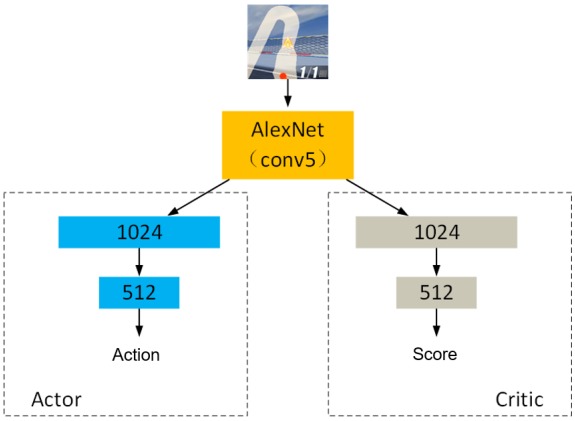

Fig. 15. Network structure for PPO.

TABLE II
PERFORMANCE COMPARISON WITH STATE-OF-ART METHODS

	Player	Random method	DQN [6]	PPO [9]	VGG16	lightweight network	LSTM model
QQ speed (second)	80	∞	94	93	96	92	88
CRED (meter)	825	185	560	-	452	558	634

TABLE III
PERFORMANCE COMPARISON (HUMAN NORMALIZED PERFORMANCE)

	Lightweight network	+ data alignment	+ class balance	+ LSTM
QQ speed	80.0%	84.7%	86.9%	90.9%
CRED	51.0%	55.6%	67.6%	76.8%

In our experiment, we find that it is difficult for AI to learn game strategy with drift action using reinforcement learning, the reason is perhaps that drift action is sensitive to action delay, which significantly increases the exploration space. Actions for reinforcement learning for QQ speed are set as left, right and no action.

To train reinforcement learning for QQ speed, reward is calculated based on the speed of game character. We collect a set of image patches for each number and apply template matching to recognize current speed. For CRED, the distance moved by the character is used as the reward. Positive reward is obtained with the increasing of distance. if the character dies or is stuck, AI model gets a negative feedback. The number of distance is recognized through template matching.

Model parameters are optimized through a large amount of interaction between AI and environment. For QQ speed, it takes 48 and 18 hours to obtain trained model for DQN and PPO, which means that PPO algorithm learns game strategy around 3 times faster than DQN. For CRED, the model is trained based on the whole game image, which is more complex than the radar map in QQ speed. It takes around one week to finish training using DQN method. Experiment results are shown in Table II, which records the average performance of 10 rounds. Due to action delay and lack of state exploration, the performance using reinforcement learning is similar to the results with lightweight network. This indicates the advantage of imitation learning, which learns game strategy from human recorded samples and supports off-line training.

The average performance of LSTM network is 4 seconds faster than lightweight network. It takes about 88 seconds to finish game, which is 8 more seconds than time cost of player who records the game. For CRED, we record running distance in a specific scene for 10 times. The average distance is 185, 558, 634 and 825 meters for random method, lightweight network, LSTM model and player. For CFM, we record game based on the fixed path. Using network with LSTM, AI can follow the recorded path. While the agent performs much worse with lightweight network, which lacks memory of previous states. In general, using temporal feature can boost performance of game AI by using relationship among adjacent actions.

C. Ablation Study

The importance of data alignment and class balance is also investigated. We record the human normalized performance using lightweight network, data alignment, class balance and LSTM in QQ speed and CRED. Experiment results are shown in Table III. For QQ speed, using light-weighted network achieves 80% performance of human, which is calculated based on time cost of finishing race. Using data alignment improves around 4% performance compared with only using lightweight network. The reason is that action delay is partially solved by data alignment. Using class balance further improves 2.2% and 12.0% for QQ speed and CRED. It shows the importance of class balance, which avoids the bias to specific action. By adding LSTM, the human normalised performance achieves peak value, which shows the efficiency of temporal feature.

Furthermore, we test the speed of our network using CPU. It takes 13 milliseconds to output action probability based on input image, which is resized to 150×150 pixels. This indicates the efficiency of the proposed lightweight network.

V. CONCLUSION

In this paper, we propose a novel framework for game AI, which only takes around one hour to get the trained model. A lightweight network is proposed to extract deep feature for input image, followed by LSTM structure to utilize temporal feature. It takes only 13 milliseconds to output action probability using CPU, which saves time and memory cost. This work is evaluated in a variety of popular commercial games. In our experiments, the proposed work is compared with recent state-of-art reinforcement learning methods. With the optimized imitation learning, our work achieves higher performance than DQN and PPO methods, which take around two days to complete training. Good performance in experiments demonstrates the efficiency of the proposed method.

In future, we can add weight of each sample in imitation learning, which indicates the advantage of the chosen action compared with other actions. Based on this method, model can focus on learning strategy with important samples, which will boost performance of AI.

REFERENCES

[1] J. Oh, Y. Guo, S. Singh, and H. Lee, "Self-imitation learning," *arXiv preprint arXiv:1806.05635*, 2018.

[2] J. Ho and S. Ermon, "Generative adversarial imitation learning," in *Advances in neural information processing systems*, 2016, pp. 4565–4573.

[3] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[4] L. Zhang, H. Pan, Q. Fan, C. Ai, and Y. Jing, "Gbdt, lr & deep learning for turn-based strategy game ai," in *2019 IEEE Conference on Games (CoG)*. IEEE, 2019, pp. 1–8.

[5] J. Peters and S. Schaal, "Reinforcement learning of motor skills with policy gradients," *Neural networks*, vol. 21, no. 4, pp. 682–697, 2008.

[6] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[7] Y. Duan, X. Chen, R. Houthooft, J. Schulman, and P. Abbeel, "Benchmarking deep reinforcement learning for continuous control," in *International Conference on Machine Learning*, 2016, pp. 1329–1338.

[8] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.

[9] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[10] Y. Duan, M. Andrychowicz, B. Stadie, O. J. Ho, J. Schneider, I. Sutskever, P. Abbeel, and W. Zaremba, "One-shot imitation learning," in *Advances in neural information processing systems*, 2017, pp. 1087–1098.

[11] S. Ross and D. Bagnell, "Efficient reductions for imitation learning," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 661–668.

[12] S. Shalev-Shwartz and S. Ben-David, *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[13] P. Sermanet, K. Xu, and S. Levine, "Unsupervised perceptual rewards for imitation learning," *arXiv preprint arXiv:1612.06699*, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[15] M. Zheng, S.-h. Zhong, S. Wu, and J. Jiang, "Steganographer detection via deep residual network," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2017, pp. 235–240.

[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[17] X. Li, M. Ye, Y. Liu, and C. Zhu, "Memory-based pedestrian detection through sequence learning," in *2017 IEEE International Conference on Multimedia and Expo (ICME)*. IEEE, 2017, pp. 1129–1134.

[18] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[19] G. Varol, I. Laptev, and C. Schmid, "Long-term temporal convolutions for action recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 6, pp. 1510–1517, 2017.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

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