Evaluation of Situations in RoboCup 2D Simulations Using Soccer Field Images

Tanguy Pomas and Tomoharu Nakashima(✉)

Department of Computer Science and Intelligent Systems,
Osaka Prefecture University, Sakai, Japan
mb104079@edu.osakafu-u.ac.jp, tomoharu.nakashima@kis.osakafu-u.ac.jp

Abstract. This paper proposes a convolutional neural network (CNN) that assesses the situation at one point of a RoboCup 2D soccer game, predicting which team will score next and when, by only taking soccer field images as input. To train this model, we define a metric, called *SituationScore* that estimates, for a frame, the remaining number of frames before next goal. A dataset containing more than one million RoboCup 2D soccer field images labeled with their SituationScore, from more than 5,000 games has been built to train our CNN. Our CNN-based model manages to predict the *SituationScore* of a frame with an average error lower than the other methods tested in this paper that use raw numerical data from log files.

1 Introduction

Since games in RoboCup 2D Soccer Simulation League, or simply RoboCup 2D, are simulated on computer, every piece of information regarding them is stored, and can be easily extracted for analysis purposes. Such logs contain, for example, the coordinates of every player, for every cycle of a game, as well as their velocity and their orientation. Therefore, it is not surprising that log files play an important role when analyzing games and designing strategies.

RoboCup 2D games can be visualized on computers thanks to tools such as soccerwindow2 that displays the position of all players and the ball for every cycle of the game, with one computational cycle corresponding to one frame on it. This tool not only allows audience to visualize games, but also researchers to replay the same game, the same actions, in order to easily analyze them. However, even when working with soccerwindow2, field images are only considered as a representation of numerical data stored in log files, not as data itself.

This paper proposes a model that uses such images as input data in order to assess how good or bad is the situation for both teams playing, without considering numerical data available in log files. A metric, called *SituationScore*, is proposed and used to evaluate the state of the field, estimating the number of remaining frames before the next goal. We decided to build our model by using a Convolutional Neural Network (CNN), given their efficiency for a large range of tasks related to image analysis. To allow comparison with our CNN-based

D. Holz et al. (Eds.): RoboCup 2018, LNAI 11374, pp. 275–286, 2019.
https://doi.org/10.1007/978-3-030-27544-0_23

model, other models using raw numerical data from log files have been built and evaluated, such as fully-connected neural networks and decision trees.

In order to evaluate these models, we built datasets containing more than one million soccer field images and their corresponding raw numerical data from more than 5,000 games. Our CNN leads to slightly better results than the other tested methods. This may indicate that spatial features extracted by our CNN can provide better information than raw numerical data.

The remainder of this paper is organized as follows. Section 2 presents research that shares common points with this paper. In Sect. 3, we detail what is the main objective of our work. Section 4 describes the procedure used to build our datasets and provides the reasons why several datasets are required. In Sect. 5, experiments relying on raw numerical data, as well as our model only training with soccer field images and their respective results are described. Finally, Sect. 6 summarizes our work, its results and limits, and provides an idea to improve it.

2 Related Work

CNNs are known to be really efficient for a wide variety of tasks, especially tasks related to computer vision, such as object classification, segmentation, or face recognition. We can mention the work of Krizhevsky et al. [1] who first used CNN in 2012 on the Imagenet dataset [2] and significantly outperformed every other method employed at that time to perform image classification. Their work is particularly appropriate to illustrate the learning power of CNNs, as Imagenet dataset contains millions of images corresponding to 1000 different classes.

More closely related to our work, Stanescu et al. [3] used CNNs to evaluate the state of a Real-Time Strategy (RTS) game, μRTS. Their CNN has been trained to predict which of the two players is the most likely to win. To do this, it analyzes the state of the game at three different moments. Each of these moments is represented by a $8 \times 8 \times 25$ image stack corresponding to the 25 features that can be found on the 8×8 map, such as resources, units and buildings of each player. Despite accurately predicting the winner of an RTS game, this prediction only concerns μRTS, which is a very simple RTS game designed for testing AI techniques.

Other more complex and popular RTS games have also been subjects of research. That is the case of StarCraft on which some researchers applied machine learning methods to estimate the global state of a game. For example, Erickson et al. [4] proposed a logistic regression method to predict which player will win a game by taking the global state of the game into account. To do so, many aspects of the game have been identified and converted into features such as number of units, map coverage and skill of each player. These game states have been taken every 10 s for each of their 400 replays, providing them enough data to evaluate their model on states from particular time intervals. Rivari et al. [5] further improve their results by computing new features and applying gradient boosting regression trees and random forests to estimate the winner of the game.

Another completely different way to predict the winner of a StarCraft game is proposed by Sánchez-Ruiz et al. [6] who use influence maps in order evaluate the state of the game. Influence maps are matrices representing the situation for each player on the StarCraft map. For each unit owned by a player, a numerical influence value is added to its corresponding position on the influence map and its surroundings. In their work, influence maps are then reduced from 128×128 to 4×4 matrices. Therefore, the global state of a game is represented by 16 numerical values per player. Several machine learning methods are then used to predict the winner of a game based on these influence maps that are computed every 30 s.

While [4] and [5] extracted numerical values to evaluate the state of a game, other work transform their initial raw numerical data into images. That is the case of Souza et al. [7], who convert time-series into Recurrence Plots (RP), considered as gray images. Several features are then extracted from these images and used as input of a Support Vector Machine (SVM) algorithm that will use this data to classify the time-series. Hatami et al. [8] propose a very similar method that converts time-series into RP images, which are then used as input images of a CNN that will perform a classification of the initial time-series. The main difference between these last two papers is that the latter uses a CNN directly working with RP, without extracting hand-crafted features first.

The work presented in this paper shares a few similarities with some of these researches, as it involves the use of a CNN training on soccer field images that are visual representation of available raw numerical data. However, it is still one of a kind as it aims to evaluate the immediate state of a RoboCup 2D game with only one image.

3 Task Definition

In RoboCup 2D soccer games, many different metrics could be defined to tell which of the two playing teams currently has the upper hand, for example considering how many players of each team are on which part of the field, as well as where is and who possesses the ball. However, such analysis, while easily conducted using numerical data from log files, would be much more complicated to conduct working only with field images. Therefore, another metric, independent from players' coordinates, has to be defined to assess the situation of a game.

To this end, we introduce the *SituationScore* of a frame f that is defined by

$$SituationScore(f) = \pm(100 - n), \tag{1}$$

where n is the number of frames between f and the frame corresponding to the next goal. It is assumed in this paper that the considered frames are at most 100 cycles away from the next goal. Therefore, in this formula, n is necessary lower than 100. The *SituationScore*'s sign is determined by the team that will score this goal. We chose to consider a positive score when left team will score next goal and vice versa. An example of a soccer field image and its corresponding score is provided in Fig. 1.

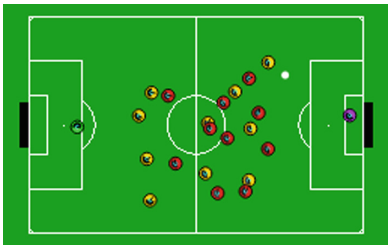


Fig. 1. Image taken 35 frames before the left team scores. The corresponding *SituationScore* is +65.

Such definition presents two main assets, the first one being the easiness to assign the correct score to newly produced images, as it does not require any complex computation. The second asset is inherent to our objective to work only with images. This metric does not take into account the state of the field itself at all, only the remaining time, the number of frames, before next goal. Therefore, many situations or formations can automatically be covered by this model.

On the other hand, this score is only defined for the last 100 frames before a goal, meaning that, for most frames of a game, this score is not defined. However, our model estimating such score is built only on valid frames, within 100 frames before a goal. Therefore this problem does not affect the training phase. Then, a solution to bypass this problem is to think of evaluated situations that are more than 100 frames away from a goal as similar to situations with identical scores.

Having defined the *SituationScore*, our goal is to build a model that estimates its value for any given frame. To this end, a dataset containing images associated with their score has first to be created. In this paper, we built a dataset containing more than one million images from more than 5,200 games, as well as other datasets containing their corresponding raw numerical data. Using these datasets, we built several models to estimate the *SituationScore*, in particular a CNN only working on the image dataset.

4 Datasets Construction

4.1 Procedure

In order for our dataset to cover as many situations and formations as possible, games between several 16 different teams have been simulated.

Dataset creation has been decomposed into several steps:

1. Games between each possible pair of teams are simulated and their log files saved
2. Log files of games where no goal have been scored are deleted
3. For each game, every frame is saved using soccerwindow2
4. A python script is used to analyze their corresponding log files to determine at which cycles goals have been scored

5. Every 100 frames before each of these goals are kept, and their *SituationScores* are computed. These frames are renamed to include their *SituationScore* while the other are deleted.

When using soccerwindow2 in this procedure, some graphic options have been precised, mainly to enhance the frames' quality, removing superfluous information and enlarging the players and the ball. These options include hiding score board, player numbers, view area, stamina and potential yellow card, as well as setting player size to 2, and ball size to 1.3. Size of saved images has also been precised in these options, but minimum size from soccerwindow2 being 280×167, kept frames have then been cropped to be of size 256×160. Soccer field images visible in this paper, such as in Fig. 1 offers a good insight of frames contained in our dataset.

A dataset containing about 1.02 million soccer field images taken from 5215 games has been constructed. This dataset has then been split into three parts: a training set containing ∼720,000 images, a validation set containing ∼156,000 images and a test set containing ∼135,000 images. Images from a specific game are all included in only one of these sets. In other words, each of these sets contains images from different games.

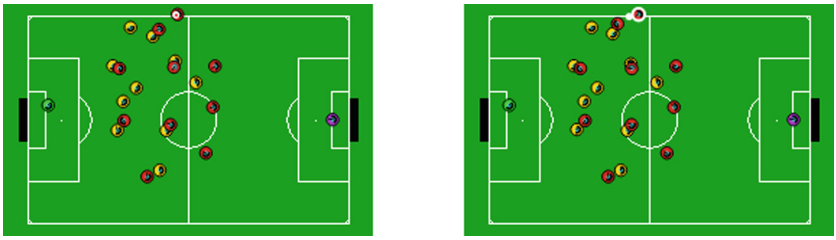
4.2 Play-On Only Dataset

During a soccer game, there are several phases during which players have some time to replace themselves on the field, while a player is about to make a kick. These phases are typically kick-in, free kicks or corner kicks, and they are implemented in RoboCup 2D Soccer Simulation League. We will refer to them as "Non Play-On" (NPO) phases or events.

These events are quite common and also happen regularly within the last 100 frames before a goal. Therefore, NPO events concern a significant proportion of images of the previously built dataset that systematically gathered the last 100 frames before a goal. However, it is not uncommon that players barely move during these NPO phases, leading to almost identical successive frames, with different *SituationScores*. Figure 2 illustrates this phenomenon, by showing two images that are taken 28 frames apart. Players have barely moved during these 28 frames, thus images are almost identical, but the *SituationScore* is completely different.

Regarding this issue, we decided to build another dataset using the same procedure as for the first one. We decided to call this second dataset "Play-On Only" (POO) dataset, in reference to the play-on phase, which is simply the standard game phase. The only difference with the first dataset that we may refer to as "All Frames" (AF) dataset, is that NPO phases frames, like kick-in and corner kicks frames, are not counted within the last 100 frames before a goal. Basically, this dataset contains the last 100 play-on frames before each goal, which often include frames that are more than 100 frames before a goal.

The POO dataset has been built using the same log files as the all frames dataset, it has also been split in the exact same way into training, validation



(a) *SituationScore* -1 (b) *SituationScore* -29

Fig. 2. Lack of players' movements during a kick-in.

and test sets. Therefore, each of these contains approximately the same number of images as its all frames counterpart.

4.3 Raw Numerical Dataset

To allow comparison with our CNN-based model, models using raw numerical data, such as coordinates, also have to evaluated. Consequently, a dataset containing numerical data from the same log files has first to be built.

In order to allow fair comparison with our CNN model, two different numerical datasets have been built. The first one that we will refer to as "coordinates dataset" only contains numerical data that could be retrieved by analyzing just one image. In other words, it contains for one frame, the ball and the players' coordinates along with their body angle, which can be obtained by paying attention to the orientation of their back, the black part of their circle visible on every frame, such as in Fig. 1. The second one that we may call "all numerical dataset" contains the same data as the first one, on top of ball and players' velocities, their absolute neck angle and their view angle range.

The problem of non play-on cycles being exactly the same whether field images or their corresponding coordinates are considered, numerical datasets have also been split into Play-On Only datasets and All Frames datasets. Therefore, four numerical datasets have been built: All Frames Coordinates (AFC), All Frames All Numerical (AFN), POO Coordinates (POOC) and POO All Numerical datasets (POON).

These numerical datasets have been built using the same log files as for the previous images datasets. Splitting into training, validation and test sets has also been done the same way, leading to sets of the same size, corresponding to the same games.

5 Experiments and Results

5.1 Experiments on Raw Numerical Data

While the main aspect of our work is to use CNN to build a model that accurately estimates the *SituationScore*, most, if not all, work on RoboCup 2D

exploits numerical data from log files. Therefore, to allow comparison with our CNN model and to determine if working directly with soccer field images has an interest, models using numerical data have first to be built and evaluated.

Two kinds of experiments using numerical data have been conducted. The first one was experiments using Fully-Connected Neural Networks (FCNN), build with the TensorFlow library. These experiments are similar to our CNN experiments, but with much simpler neural networks, containing only fully-connected layers, up to thirty of them. The only hyper-parameters that were tested during these tests were batch size, learning rate, number of fully-connected layers and number of units within them. These experiments were run on the four numerical datasets presented earlier.

Table 1 includes the best results of these experiments for each numerical dataset, based on the Mean Absolute Error (MAE) between their *SituationScore* predictions and true scores. As expected, using all numerical data available leads to better results than only using data visible on one frame. Moreover, these first results tend to prove the importance of distinguishing NPO phases from standard game phases, as results on POO datasets are significantly better than those on AF datasets.

For the second series of experiments, several models built on these datasets have been tested using various machine learning methods. Experiments were conducted using Scikit-learn Python machine learning library [9]. Tested methods were bagging of decision trees, random forest, extra trees regressor, linear regression, linear SVR and k-Nearest Neighbors. The latter three gave results much worse than the other methods, not being really efficient with huge datasets. In particular, without transformation of the training set, the kNN method need to compute a distance between a situation and every other 720,000 training situations to estimate the score of this situation, making it completely unusable for our task. Random forest, bagging of decision trees and extra trees regressor methods gave similarly good results.

Table 1 presents the best results obtained with all methods implemented and tested with Scikit-learn Python library. The extra trees regressor model gives slightly better results than the other on all datasets, except on the All Frames, all Numerical (AFN) dataset, for which best results are obtained with a Bagging of Decision Trees model. Interestingly enough, this time better results are achieved on AF rather than on POO datasets.

5.2 Experiments Using Images as Input Data

Regarding our CNN implementation, we decided to use the TensorFlow library in Python. A CNN architecture similar to the VGG architecture [10], along with appropriate hyper-parameter values leading to satisfactory results have been determined by preliminary experiments. Our architecture is illustrated in Fig. 3. Our CNN was trained with an initial learning rate of 0.0001, decreasing by 5% every 1500 steps, corresponding to batches of size 16. It should also be noted that our CNN takes 160×256 images as input and contains a 15% dropout term at the end of each convolutional block, as well as after each fully-connected layer.

Table 1. Lowest MAE obtained during numerical experiments on every dataset

Method	AFC	POOC	AFN	POON
Fully-connected neural network	14.91	14.39	14.07	13.79
Linear SVR	19.89	18.79	18.52	17.59
Linear regression	20.30	19.18	18.81	17.95
Decision tree bagging	14.66	14.69	13.96	14.23
Random forest	14.65	14.69	13.97	14.25
Extra trees regressor	14.60	14.58	14.14	14.21
k nearest neighbors	18.56	-	-	-

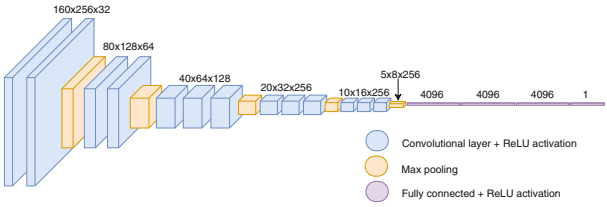


Fig. 3. Final architecture of our CNN, composed of the same four convolutional blocks as VGG, but with one more fully-connected layer

From lack of being truly optimal, our final architecture, along with presented hyper-parameter values quickly and consistently leads to a MAE usually between 13.3 and 13.6 that do not rise afterwards.

Table 2 shows the lowest MAE obtained using the presented architecture with its hyper-parameters values along with. Further hyper-parameters adjusting may lead to slightly better results, such as a MAE consistently around 13.3, but systematically getting below this value may be extremely difficult without changing this architecture.

A remark that has to be done regarding these results is that bagging of decision trees, extra trees and random forest can provide lower MAE if trained longer, with more trees. However, their MAE has only decreased by about 0.02 when going from 100 to 200 trees. Therefore an improvement of more than 0.1 is probably not to be expected. Extensive hyper-parameters adjusting, on the other hand, could possibly lead to better results, but that is also the case for FCNN and our CNN-based model.

Besides this issue that may have a slight impact on results, we can also consider the prediction time issue. Indeed, if we want to be able to compute the *SituationScore* in real time, our models have to be fast enough to do it before the next frame is displayed, which is within 100 ms. Even more, if we want the *SituationScore* to be a parameter that a team considers when making choices, prediction time has to be much shorter than 100 ms so that strategic choices can be made in the remaining time. Table 2 includes prediction time of

Table 2. Best MAE obtained using our CNN and most efficient methods

Method	AF	POO	Prediction time
Proposed CNN	13.31	13.27	∼4.1 ms
Fully-connected neural network	14.07	13.79	∼0.8 ms
Bagging of decision trees	13.93	14.23	∼6.0 ms
Random forest	13.97	14.25	∼5.1 ms
Extra trees	14.14	14.21	∼9.1 ms

the most efficient methods tested. All of them compute a *SituationScore* in less than 10 ms, which shows that their predictions are fast enough to be used in a team strategy. However, it should be precised that our CNN and FCNN models required a GeForce GTX 1080 GPU to be that fast, while the other methods used a simple Intel(R) Core(TM) i7-7700 CPU @ 3.60 GHz.

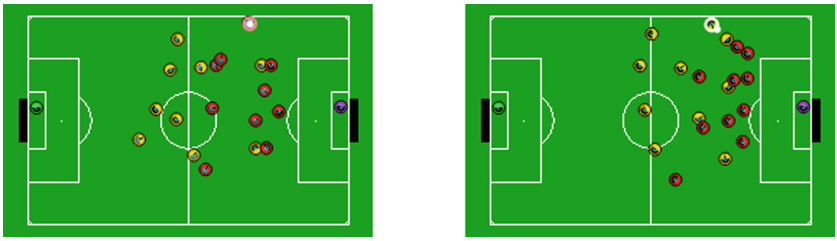
5.3 Additional Remarks

AF and POO Datasets. In some experiments, models built on AF datasets perform better than models built on POO datasets. In particular that is the case in decision tree bagging and random forest experiments. However, the contrary is observed in FCNN experiments. Our CNN-based model, as for itself, gives approximately the same results whether trained on AF or POO dataset. This proves that, contrary to what we could have thought, the POO datasets are not "better" datasets than AF datasets. In fact, it may be possible that ignoring movements of players during NPO phases when some of them move a lot compensates the improvement got by ignoring NPO frames when players barely move.

Indeed, in some NPO phases players barely move leading to many similar frames with different scores. However, in other NPO phases some players move a lot, meaning that ignoring them will bring a discontinuity in the *SituationScore* distribution. There will be frames corresponding to the same goal, with scores different by 1, but some players will seem to jump from one position to a completely another one between those frames, as shown in Fig. 4.

Therefore, both AF and POO datasets have a defect related to these NPO phases. As a consequence, it may be better to only focus on the AF dataset, as it represents everything that happens in the game.

Distribution of Prediction Error. Intuitively, the *SituationScore* is more easily estimated on some frames than other. For example, the closer the ball is to the goal, the sooner the goal is likely to be scored and the higher should be the *SituationScore*. Therefore, high *SituationScore* situations, regardless of their sign, are relatively easy to identify. On the other hand, when the ball is far from the goal, position of every player on the field has to be considered to assess the



(a) *SituationScore* +32 (b) *SituationScore* +33

Fig. 4. Players' position jumping between two consecutive POO images.

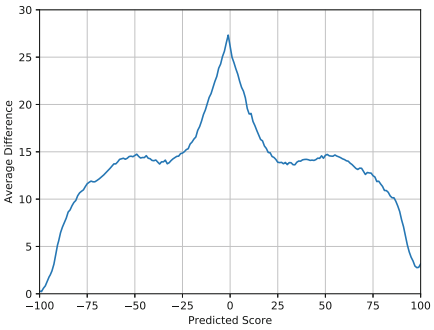


Fig. 5. Average error for each computed *SituationScore*.

situation, which makes it considerably more difficult to accurately estimate the *SituationScore*.

By computing the average difference between estimations and true scores for each possible value, this intuition can be confirmed. In fact, when true score is between −10 and +10, average error of our model is above 20 points while it drops below 5 points when true score is either above +90 or below −90, as visible in Fig. 5.

Situation Score Predictions on Other Frames. Another problem remains as the *SituationScore* is only defined for the last 100 frames before a goal but games used in our experiments contains more than 3,000 frames for only two goals on average. That means the *SituationScore* is not really defined for more than 90% of frames. This may not be a problem to train our model, as datasets have been built for this purpose, but make our model's predictions harder to interpret for frames that have no true *SituationScore*. A simple solution, when using our CNN on every frame of a game, would be to consider that the situation in the frame is similar to a situation with predicted score, whether or not a goal will be scored soon. However, it is still interesting to have an idea of the probability that a goal will be scored knowing a *SituationScore* estimation.

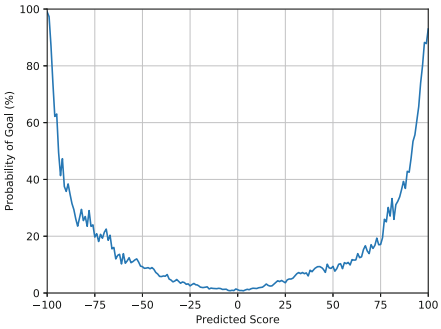


Fig. 6. Probability that a goal is scored within next 100 frames for each computed *SituationScore*.

In order to have the beginning of an answer to this question, another dataset, containing all frames from 120 games, totalling 360,612 images where *SituationScore* is not defined and 23,098 images where it is has been built. These first figures confirm that *SituationScore* is properly defined in only about 6% of all frames. After training on the All Frames dataset, our CNN estimated the score of all these frames. Without surprise, the higher the predicted *SituationScore* regardless of its sign, the more likely a goal will be scored soon, and vice-versa. While predicting a score between -10 and $+10$ has less than 2% chance to corresponds to an actual frame within 100 frames before a goal, chances that a goal will be scored soon increase with *SituationScore* prediction value. For example, this probability gets higher than 50% if predicted score is below -94 or above 92, as shown in Fig. 6.

Considering a new model, estimating whether or not a goal will be scored in the next X frames could be an interesting extension of this work.

6 Conclusion

In this paper, we have introduced the *SituationScore*, a metric that assesses the field situation at one point of a RoboCup 2D game by estimating the remaining number of frames before next goal. Datasets containing frames or numerical data from more than 5,000 games, along with their corresponding *SituationScore* have been built in order to train models estimating this score.

Several models that predict this score have been built. Most of them were trained using raw numerical data and, among them, the decision tree bagging, random forest, extra trees and FCNN performed the best. However, we also focused on the construction of a CNN-based model that outperformed these models by training only on soccer field frames, disregarding numerical data. However, most tested methods could provide slightly better results if trained longer, with extensive hyper-parameter adjusting.

If results using our CNN are satisfactory and able to predict the number of remaining frames before next goal with an average error around 13.5,

another problem inherent to the *SituationScore* definition subsists. In fact, the *SituationScore* definition assumes that a goal will be scored within 100 frames. Therefore, our model is not trained to estimate if a goal will be scored soon, but instead assumes that it will. In other words, when considering truly any frame of a RoboCup 2D game, our CNN estimates a possible number of frames before next goal is scored regardless of the probability that it happens.

Thus, this work could be extended by considering a new model that predicts whether or not a goal will be scored in the next X frames. Combining this new model with the model presented in this paper would be a way to complete it, to make it fully usable. Another way to do this would be to update the *SituationScore* definition so that it is defined on all frames of a RoboCup 2D game, for example considering a score of 0 when no goal is scored within the next 100 frames. It would be also useful to investigate the performance of various CNNs that are trained by using games from different years. This would lead some insight into the trend of teams by years. This is also left as a future task.

References

1. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, vol. 25, pp. 1097–1105 (2012)
2. Russakovsky, O., et al.: Imagenet large scale visual recognition challenge. Int. J. Comput. Vis. **115**(3), 211–252 (2015)
3. Stanescu, M., Barriga, N.A., Hess, A., Buro, M.: Evaluating real-time strategy game states using convolutional neural networks. In: Proceedings of the IEEE Conference on Computational Intelligence and Games, pp. 1–7 (2016)
4. Erickson, G.K.S., Buro, M.: Global state evaluation in StarCraft. In: Proceedings of the Tenth Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 112–118 (2014)
5. Ravari, Y.N., Sander, B., Spronck, P.: StarCraft winner prediction. In: Proceedings of the Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference, pp. 2–8 (2016)
6. Sánchez-Ruiz, A.A., Miranda, M.: A machine learning approach to predict the winner in StarCraft based on influence maps. Entertain. Comput. **19**, 29–41 (2017)
7. Souza, V.M.A., Silva, D.F., Batista, G.E.A.P.A.: Extracting texture features for time series classification. In: Proceedings of the Twenty-Second International Conference on Pattern Recognition, pp. 1425–1430 (2014)
8. Hatami, N., Gavet, Y., Debayle, J.: Classification of time-series images using deep convolutional neural networks. In: Tenth International Conference on Machine Vision (ICMV): Image Analysis and Imaging System, Vienna (2017)
9. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., et al.: Scikit-learn: machine learning in python. J. Mach. Learn. Res. **12**(Oct), 2825–2830 (2011)
10. Simonyan, K., Andrew, Z.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2015)