Investigating the Performance of Optimization Techniques on Deep Learning Models to Identify Dota2 Game Events

Matheus Prado Prandini Faria[a], Etienne Silva Julia[b], Henrique Coelho Fernandes[c], Marcelo Zanchetta do Nascimento[d] and Rita Maria Silva Julia[e]

Computer Science Department, Federal University of Uberlândia, Uberlândia, Minas Gerais, Brazil

Keywords: Deep Learning, Convolutional Neural Network, Genetic Algorithm, Bayesian Optimization, Video Games, Game Events, Classification, Dota2.

Abstract: Game logs are an important part of the player experience analysis in literature. They describe the major actions and events (related to the players or other elements) that affect the progress of a game. In most existing games (especially popular commercial games like FIFA, Dota2 and Valorant), their access is typically restricted to the game's developers. Deep Learning (DL) approaches have been proposed to perform game event classification from videos. However, retrieving relevant information about these game events (normally associated with actions performed by players) in real-time is still a challenge. Existing approaches require high computational power that serves as an additional issue. In this sense, the present paper investigates a set of approaches that aim to reduce the computational cost of DL-based models - more specifically, Convolutional Neural Networks (CNN) based on Residual Nets architectures - through Genetic Algorithm and Bayesian Optimization. This investigation is carried out in the context of Dota2 game event classification. The comparative analysis showed that the models obtained herein achieved a classification performance as good as the models of the state-of-the-art considering the Dota2 dataset, but with significantly fewer parameters. Thus, this work can help in the generation of optimized CNNs for real-time applications.

1 INTRODUCTION

Video games have become a critical part of the entertainment industry, impacting the lives of billions of people every day (Wijman, 2019). They exert wide influence among the younger generation, whether for the appealing side of their content or for the messages and ideas they disseminate (Moosa et al., 2020). In addition to being designed for fun and entertainment, games can also be used in education, health and many other areas. For example, to understand underlying psychological and behavioral implications of the interaction of players with this medium. This is because video games provide extremely appropriate domains for the occurrence of rich affective experiences measurable through in-game behavior (Azadvar, 2021).

For this purpose, it is essential that developers count on information that efficiently represents the players' experiences. This is generally achieved through game logs, which register what the players are doing and what is happening in the scenario. Accessing these logs traditionally requires access to the game engine, that is, to the software-development environment used to make the game. This poses a challenge because game engines are usually inaccessible due to the companies' privacy concerns (Luo et al., 2019a) and, even considering game platforms which make available the game logs, it is not possible to count on game information in situations in which it must be retrieved from recorded games.

State-of-art works to derive approximated game logs from gameplay videos rely on Deep Learning (DL) approaches - more specifically, on Convolutional Neural Network (CNN). These networks are widely applied to video action recognition because they present an excellent level of performance in tasks involving visual data (He et al., 2016). For example, (Xu et al., 2019) proposes an approach based on CNNs and Recurrent Neural Networks (RNN) capable of encapsulating a lot of relevant functionalities for online action detection into a single framework (Temporal Recurrent Network), which confirmed the

[a] https://orcid.org/0000-0003-1468-9243
[b] https://orcid.org/0000-0003-3750-4264
[c] https://orcid.org/0000-0002-7078-9620
[d] https://orcid.org/0000-0003-3537-0178
[e] https://orcid.org/0000-0001-5181-5451

Faria, M., Julia, E., Fernandes, H., Zanchetta do Nascimento, M. and Julia, R.
Investigating the Performance of Optimization Techniques on Deep Learning Models to Identify Dota2 Game Events.
DOI: 10.5220/0011691800003417
In *Proceedings of the 18th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2023) - Volume 4: VISAPP*, pages 881-888
ISBN: 978-989-758-634-7; ISSN: 2184-4321

success of CNNs and RNNs in the task of extracting actions from videos.

However, CNNs require a significant amount of computational resources considering the context of real-time game analysis, such as online modeling the behavior of players and identifying actions performed in e-sport tournaments livestream (Luo et al., 2018). In this sense, (Luo et al., 2019a) proposed some approaches based on transfer learning and network pruning to build accessible CNN models - in terms of hardware - with the goal of extracting sequence of game events from gameplay videos in real-time with little computational resources. These models showed a considerable reduction in training time, inference time and memory usage of the Graphics Processing Unit in CNNs based on the ResNet152 architecture (He et al., 2016), while maintaining an excellent event classification performance on the Dota2 game.

These approaches proposed by (Luo et al., 2019a) rely on a model pre-trained on a large and high-quality real-world dataset. Considering the domain of games, this limits their application to the more realistic ones. As highlighted in (Luo et al., 2018), real-world datasets - as ImageNet (Deng et al., 2009) and UCF (Soomro et al., 2012) - do not transfer their characteristics well to games with a pixelated aesthetic (that is, games with simplistic graphics) like *Super Mario* and *Mega man*. Thus, an existing high-quality dataset with pixelated aspects would be required to apply such approaches to these games, which is not always an easy task to be performed.

Motivated by such facts, in this paper the authors extend the approaches proposed in (Luo et al., 2019a) by investigating the design of optimized CNN architectures for game event classification without the use of transfer learning techniques. This work explores two distinct methods - a Genetic Algorithm (GA) based one and Bayesian Optimization (BO) - in a DL optimization framework with the aim to minimize the number of parameters (size of weights vector) and the classification error of the CNN architectures.

The next sections are structured as following: section 2 resumes the background; section 3 describes the CNN architecture optimization; section 4 presents the experiments and results; finally, section 5 shows the conclusions and the future works.

2 BACKGROUND

Among the existing approaches to derive approximated game logs from video games, those based on deep learning stand out. In (Luo et al., 2018), the authors proposed two possible solutions for the problem of classifying events in video games (using Super Mario Bros as case study). They did so by presenting two possible approaches, the first one consisting on training a popular CNN detection model with a manually labeled game dataset. This approach requires a large labeled dataset, and a lot of processing power in order to train the network. The second approach utilizes a *Student-Teacher* technique (Wong and Gales, 2016) to lower the amount of training data, and processing, needed. The results obtained with it, by training and testing a network on different games in the same style and genre, also were not satisfactory for the author‘s goal. However, they managed to obtain solid results when applying the *Student-Teacher* method with a network pre-trained on the UCF dataset to the game Skyrim, which, in addition to having realistic graphics, also was tested with the same actions depicted in the real-world data.

In neural networks projects (covering CNNs), one of the most important tasks is the definition of their architectures (for example, hyperparameters such as the number of layers and the number of neurons in each one). Thus, due to the huge number of possible combinations (large search space), the task of defining a good architecture manually is very costly. There are architectures studied exhaustively for visual data classification tasks, among them MobileNet (Howard et al., 2017) and ResNet (He et al., 2016) stand out (each one of them has its peculiarities). However, as most of them are very deep in terms of the number of layers (and consequently with a very large number of parameters), high-performance machines are needed to guarantee success in real-time tasks.

In this sense, there exists some approaches to reduce training time, execution time and memory. Pruning is one of the most popular methods to reduce network complexity (in terms of number of layers and, consequently, number of parameters) (Han et al., 2016). Among the main algorithms proposed in this context, *ThiNet* (meaning ”Thin Net”) stands out (Luo et al., 2017). The goal of this algorithm is to prune the unimportant filters (without changing the original network structure) to simultaneously accelerate and compress CNN models in both training and test stages with minor performance (in this case, in classification problems) degradation. It allows transfer tasks such as classification of game events to run much faster (in both training and inference time), especially on small devices (low-performance machines).

In (Luo et al., 2019a), the *ThiNet* framework is used to compress the pre-trained ResNet152 (decreasing the inference time and the memory requirements) through the *ThiNet30* (it keeps 30% of its filters), *ThiNet50* (it keeps 50% of its filters) and *ThiNet70* (it keeps 70% of its filters) versions of ResNet152, which prunes 70%, 50% and 30% of the filters respectively. The best model generated by the mentioned work was *ThiNet30*. Its goal was to perform CNN optimization for the classification of Dota2 game events. Beside this, recent works aim to build optimized architectures for visual task using evolutionary techniques, highlighting (Faria et al., 2020) for image classification in FIFA (soccer simulation game). The mentioned work proposes the Minimum Convolutional Neural Network obtained by Genetic Algorithm (MCNN-GA) method that evolves a population of CNN architectures through genetic algorithm aiming to find the one with the best trade-off between performance classification and network accuracy.

As shown in Table 1, the present paper investigates new approaches to the Dota2 game event classification/identification task (*MCNN-GA* and *BO*) and compares with the state-of-art work (Luo et al., 2019a), which generated *ThiNet30* as the best model. The new methods explored here do not require pre-trained models to perform parameter optimization (unlike *ThiNet* which is run on a pre-trained model). Furthermore, the present work adds *BO* to the analyzed methods, which has a different nature of evolution of the *MCNN-GA* evaluated solutions.

3 DL OPTIMIZATION FRAMEWORK

This section presents the framework investigated herein to optimize DL models which identify game events in gameplay footage. Such architecture is composed of the following modules: *Preprocessing Module* and *Architecture Compression Methods*, as shown in Figure 1 and described in the sequence. The framework receives as input a dataset of game events from gameplay videos and aims to generate an optimized model (in terms of classification performance and neural network size) for such a dataset.

3.1 Capturing Frames from the Gameplay Footage

The required training data samples for this paper are extracted from Dota2 gameplay videos sourced from the main related work (Luo et al., 2019b). The samples that make up the dataset consists of individual frames containing a single active game event (that is, frames devoid of events or otherwise containing multiple events are discarded). 10 three second gameplay clips were collected of each event at 30 frames per second (FPS), which lead to a total of 9000 frames. These data are then pre-processed (subsection 3.2) in order to generate the dataset necessary to the execution of the experiments.

3.2 Frame Pre-Processing

The individual frames are submitted to a pre-processing that consists on the following two phases (following (Luo et al., 2019b)): 1) every frame is resized from its original size to 224×224 pixels, maintaining the RGB color channels; 2) every resized frame is submitted to a pixel-wise normalization (where the values range in the interval [0,1]).

3.3 Game Events

The dataset is composed of examples related to a set of the following 10 game events: **using Black King Bar**, **using Eul's Scepter of Divinity**, **using a Glyph**, **Ending the game**, **Roshan fight**, **using Shiva's Guard**, **activating a Shrine**, **team fight**, **teleport**, and **tower destruction**. They will be not described in detail here, but the authors of (Luo et al., 2019a) ensure ”they are a mix of important events involving multiple characters and individual characters employing powerful items or abilities”. In addition, the visual effects difference between such selected events is highlighted, which is an important factor in the success of CNN images classification/recognition approaches. It is important to note that this dataset it completely balanced (that is, there are the same number of examples of each class/game event), which facilitates the learning process of CNN models.

3.4 Architecture Compression Approaches

This section presents the implementation of both optimization approaches (*MCNN-GA* and *BO*) used in the *Architecture Compression* module. Basically, it is responsible for choosing an optimized CNN based on the classification performance over the input dataset evaluating a population of individuals, which correspond to solutions. The individuals and their evaluation (fitness) are described in the sequence. Next, the details of *MCNN-GA* and *BO* are presented.

3.4.1 Individuals

The individuals correspond to CNN architectures. In this work, such architectures are composed of a maximum of ten blocks (as justified in (Faria et al., 2020)) of the following types: Skip and Pooling. The Skip

Table 1: Parallel between the main related works and the present approach.

Approach	(Luo et al., 2019a)	(Faria et al., 2020)	Present paper
CNN Architecture	ResNet	ResNet	ResNet
Pre-trained Weights	Yes	No	No
Optimization Method	*ThiNet*	*MCNN-GA*	*BO* and *MCNN-GA*
Experiments Application	Dota2	FIFA	Dota2

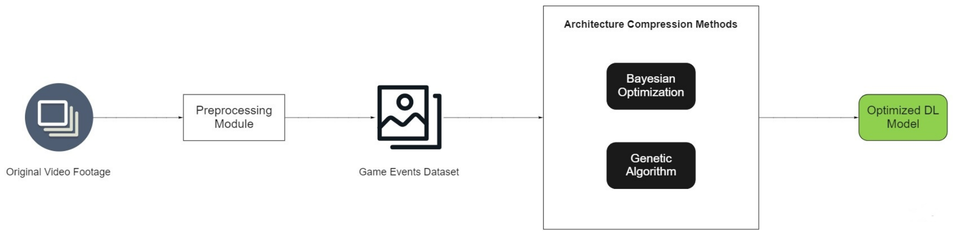

Figure 1: General architecture of the optimization framework.

block, inspired by residual blocks, is composed of two convolutional layers and a skip connection. This connection is responsible for connecting the input X of the first convolutional layer (referred to as *conv1*) to the output of the second convolutional layer (*conv2*). If the spatial sizes of the input of *conv1* and the output of *conv2* are different, a third convolutional layer (*conv3*) is applied to the input X in order to obtain the same spatial size as the output of *conv2*.

Table 2 presents the configuration used in the implementation of the Skip block. Note that the *Filter Size*, *Stride* and *Padding* hyperparameters are represented by constant values, while *Number of Filters* is the only one taken into account in the optimization process (F1 and F2). F1 represents the number of filters of the first convolutional layer, while F2 corresponds to the number of filters of the second and third (when needed) convolutional layers. In the case of the present work, they can assume the following values: 8, 16, 32 or 64.

Table 2: Skip block hyperparameters configuration.

	Number of Filters	**Filter Size**	***Stride***	***Padding***
conv1	F1	3x3	1x1	*same*
conv2	F2	3x3	1x1	*same*
conv3	F2	1x1	1x1	*same*

The *Pooling* block is composed of one pooling layer. It has the *Filter Size* and *Stride* hyperparameters values equal to 2x2. Thus, the only hyperparameter taken into account in the optimization process is the *pooling* operation type, which can be *max pooling* or *average pooling*.

All the constant values of the aforementioned hyperparameters were retrieved from (Faria et al., 2020).

3.4.2 Fitness Function

The fitness function indicates how well an individual (representing a solution candidate) is able to solve the optimization problem. In the case of this work, the evaluation of this function is based on the combination of the classification performance (in terms of the model error) and the architecture size (in terms of the number of parameters or vector of weights).

In this way, the CNN architecture of an individual is generated from its respective list of blocks (skip or pooling). Throughout this generating process, the batch normalization layer followed by the *ReLu* activation function is added to the output of each convolutional layer in the skip blocks. After all blocks are decoded, a fully connected - representing the output layer implemented with the *softmax* classifier - is added to the end of such an architecture. The number of neurons in the output layer is determined by the number of game events of the given dataset.

With the CNN model generated, the data is divided into training and testing sets using the holdout method in a way that the examples from the input dataset are distributed in the same proportion in each of these sets. It is initialized with random parameters (weight vectors), trained with the training data for ten epochs (considering the *Categorical Cross Entropy* loss function), and evaluated with the test data. The evaluation of an individual corresponds to a linear combination between two terms defined by Equation 1, FT_1 and FT_2. The former represents the fitness relative to the number of parameters of the CNN model while the latter corresponds to its classification performance (concerning the test examples). Thus, F (fitness function) is a minimization problem with respect to the aforementioned terms.

$$F = \min(FT_1 + FT_2) \tag{1}$$

Fitness Term (FT_1): it represents the evaluation of the CNN architecture in terms of its size (i.e. the number of parameters or size of the weight vector). The largest model that can be generated in the case of this work is composed of a total of 3,474,500 parameters, since in the worst case the architecture will have 10 skip blocks with values of F1 and F2 equal to 64 (the largest value among those defined for the number of filters). Thus, the smaller the size of the model, the better the evaluation of FT_1. Importantly, this value is normalized in the range [0,1].

Fitness Term (FT_2): it corresponds to the performance of the CNN architecture in terms of its classification error. After the CNN model is trained with the training data, its error based on the *Categorical Cross Entropy* loss function over the test data is calculated, FT_2. Thus, it can be stated that this term will have a good value when the test error is low, that is, when the architecture is robust in handling data from the given dataset. The value of FT_2 is also normalized in the range [0,1] just like FT_1.

3.4.3 Minimum CNN-GA (MCNN-GA)

Basically, the *MCNN-GA* tries to find the best CNN architecture to classify an image classification data set through an evolutionary process consisting of the following steps: 1) Initialization of an arbitrary population; 2) Evaluation of individuals' fitness; 3) Generation of the offspring; 4) Environmental selection.

The first two steps were previously explained in subsections 3.4.1 and 3.4.2 respectively. The third step is performed using binary tournament (Miller et al., 1995) to select parent individuals and applying the one-point crossover operation (Srinivas and Patnaik, 1994) over each pair of selected parents to generate the offspring. Then, these children can be mutated through the following operations: 1) Adding a *Skip* block randomly; 2) Adding a *Pooling* block randomly; 3) Removing a block randomly; 4) Modifying a block configuration (hyperparameters) randomly. The children are also limited to the number of ten blocks (excluding the output layer) in order to maintain the algorithm consistency. Finally, the fourth step is carried out by combining the binary tournament and an elitist strategy. The best individual is automatically placed in the next population and the others are selected through binary tournament.

3.4.4 Bayesian Optimization (BO)

BO tries to find the best CNN architecture to classify an image classification data set through a method known as surrogate optimization, that is, it models an approximation of the objective function (which is usually completely unknown) based on the following steps: 1) Sampling of random individuals; 2) Training a gaussian process; 3) Computing an acquisition function; 4) Generating a new solution (individual).

The first step was previously explained in subsections 3.4.1 and 3.4.2. It generates 10 random individuals (with arbitrary architectures) and performs their evaluation creating some observations of the objective function (in this case, this function maps the parameter set x fitness relationship). From this, the second step involves building and training a probabilistic surrogate model of the objective function (typically a Gaussian Process). The idea behind it is that the more points (individuals) that are generated, the better the approximation of the surrogate model with the objective function. The third step is performed computing the acquisition function, which drives the proposition of new potential points to test, in an exploration and exploitation trade-off. This work used the Expected Improvement based method, which is one of the most widely used acquisition functions for *BO* (Mockus and Mockus, 1991). Finally, the fourth step corresponds to generating and evaluating the best potential point found from the acquisition function. This entire process is carried out until *N* individuals are evaluated (it should be noted that at some point this method can reach convergence, that is, it always generates the same point/individual to be evaluated).

4 EXPERIMENTS AND RESULTS

The experiments performed here are primarily aimed at validating the use of *MCNN-GA* and *BO* methods (*Architecture Compression Methods*) in the process of automatically building optimized CNN architectures and comparing, in the context of Dota2 game events, with the *ThiNet* technique used by (Luo et al., 2019a).

Thus, in a first step, an evaluation and comparison is performed between the two methods explored here (Experiment 1). Then, another comparative analysis is performed on the method with the best results obtained in the first experiment with respect to the best state-of-the-art method, *ThiNet* (Experiment 2). The parameters used to validate such comparisons are the classification performance, memory and time. The *independent-samples t-test* (*t-test*) (Coleman, 2009) is used to verify if there is a superiority in terms of performance between the different methods compared.

In this sense, subsection 4.1 describes all the experimental configurations used in this work, subsection 4.2 briefly comments the evaluative parameters.

Finally, subsections 4.3 and 4.4 present the results of the Experiment 1 and Experiment 2 respectively.

4.1 General Experimental Configurations

The experimental configurations used in this paper (and also retrieved from (Luo et al., 2019b)), are described below:

- **Size of Input Images:** 224x224x3.
- **Input Data Normalization:** pixel-wise pixel normalization (that is, the pixel values are in the range between zero and one).
- **Number of Training Epochs:** 10 (the best epoch in terms of validation loss represents the final model parameters/weights).
- **Optimizer:** Adam.
- **Learning Rate:** 0.001.
- **Loss Function:** Cross-Entropy.
- **Output Activation:** Softmax.
- **Split Method:** Holdout (80% of training data and 20% of test data). It is noteworthy here that the game events are evenly distributed between the two sets (that is, 80% of examples of each class is related to training data and 20% to test data).
- **Number of Examples - Dota2 Dataset:** 9.000 (7.200 training data examples and 1.800 test data examples).

Since the technique for splitting the data is simple holdout, 10 runs were performed for each method to allow evaluation of statistical tests on the results in all experiments. In each run of each method, exactly the same examples were used in the training and test sets (i.e., using the same random seed value for splitting the data), which means the performance comparison is independent of the potential issue of data distribution, as done in (Luo et al., 2019a). Finally, the experiments were executed in an architecture composed by a machine with a Tesla K80 GPU and 16 GB RAM[1].

4.2 Evaluative Parameters

The parameters used to evaluate and compare all the methods investigated in this work are presented in the sequence.

- **Accuracy:** classification performance parameter that represents the rate of correct inferences of the model (CNN architecture) over the total number of examples in the test set.
- **Loss:** classification performance parameter that corresponds to the model error based on the value of the loss function (in this case, *Categorical Cross Entropy*) in the test set.
- **Size:** memory parameter that measures the size of the model in terms of its number of parameters (weight vector dimension).
- **Inference Time:** parameter that represents the time (in seconds) it takes for the trained model to make predictions for all examples in the test set.
- **Execution Time:** parameter that represents the time (in minutes) that the method takes to generate the CNN model.
- **Fitness:** corresponds to the evaluation value of the best individual (CNN model) found (used only in the *MCNN-GA* and *BO* methods).

4.3 Experiment 1 - Comparing *MCNN-GA* and *BO* Methods Performance

The intention of this experiment was to validate and compare the performance of *MCNN-GA* and *BO* on the construction of optimized CNN models. The execution parameters for the *MCNN-GA* are shown in Table 3: population size equal to 10; number of generations equal to 10; probability of an individual performing the *crossover* operation equal to 0.8; and, probability of an individual performing the mutation operation equal to 0.2. This means that a maximum of 220 individuals will be evaluated (20 initial individuals plus 20 offspring individuals in each of 10 generations). Then, to maintain the fairness of this comparative analysis, the *BO* method was also set up to evaluate a maximum of 220 individuals.

Table 3: MCNN-GA Execution Parameters.

Population Size	10
Number of Generations	10
Individual *Crossover* Rate	*0,8*
Individual Mutation Rate	0,2

Table 4 summarizes the mean value of the evaluative parameters for best individuals found through these two approaches over 10 runs. They indicate superior performance of *MCNN-GA* over *BO* overall. In terms of classification, the former obtained a mean accuracy rate and a mean loss value, respectively, of 0.3% higher and 0.07 lower than the latter. Regarding the mean CNN network size, the evolution-

[1]https://github.com/matheusprandini/dota2-cnn-optimization

ary method generated lighter architectures by approximately 57%. Hence, this was reflected in the difference in the fitness value. There was no difference in the inference time, but the *MCNN-GA* execution time was, on average, 2.5 hours longer than the *BO*. A *t-test* with a *significance level* ($\alpha = 0.05$) was conducted to validate the performance comparison.

Table 4: *BO* and *MCNN-GA* comparison results (mean value).

	Accuracy	Loss	Size	Inference Time	Execution Time	Fitness
BO	99.05	0.169	80994.0	5.4	481.0	0.01445
MCNN-GA	99.35	0.099	35026.0	5.4	637.0	0.00864

The *t-test* computes the following main values: *t-statistic value* (*t-value*) and *p-value*. The former indicates the actual *t-test* result and the direction of the difference (if any). The latter represents the significance value of the test. In this manner, the following H_0 was created: *BO* holds the same level of performance as *MCNN-GA*.

As Table 5 states, the *BO* and *MCNN-GA* methods performed equally well only for the inference time parameter. In terms of the classification quality parameters (accuracy and loss) and size of the generated model, a superiority of *MCNN-GA* can be noted. While *BO* had a significantly lower execution time. The authors believe that the faster convergence of the solutions generated by *BO* is because it evaluates a smaller number of individuals to find an optimized CNN for the input dataset than *MCNN-GA*. However, since *BO* generates a single solution at a time, it analyzes a smaller diversity than *MCNN-GA* which keeps several points scattered in the search space. This indicates that although the *MCNN-GA* method takes longer (which is its main limitation) to return an optimized solution, it generates lighter neural networks with higher classification efficiency than *BO* with a 95% confidence interval for the mean difference. Therefore, it is concluded that *MCNN-GA* outperformed *BO* in Dota2 dataset.

4.4 Experiment 2 - Comparing *MCNN-GA* and *ThiNet30* Methods Performance

The first experiment proved the performance superiority of the models generated from *MCNN-GA* against *BO*. Then, in this second experiment, the best method investigated in the latter experiment (*MCNN-GA*) is confronted with the best method (*ThiNet30*) obtained in the state-of-art considering the Dota2 dataset. It is important to highlight that *ThiNet30* is implemented an trained as described in (Luo et al., 2019a).

Table 6 shows the mean results of these methods through 10 runs (the results of *MCNN-GA* were transposed from Table 4). They indicate a slight classification performance superiority of *MCNN-GA* over *ThiNet30* (0.25% for accuracy and 0.014 for loss). Regarding the mean CNN network size, the evolutionary method generated lighter architectures by approximately 95%. In terms of inference time, *MCNN-GA* also had a 1.1 seconds advantage over *ThiNet30*. The execution time and fitness parameters were not computed since the *ThiNet* method does not involve generating different architectures to obtain the optimal model (it only applies the reduction of the number of filters in the original convolutional layers of ResNet152 as previously explained in section 2).

An *independent-samples t-test* with a *significance level* ($\alpha = 0.05$) was conducted to validate this comparative analysis as shown in Table 7. The null hypothesis (H_0) stated that *MCNN-GA* holds the same level of performance as *ThiNet30*.

The positive accuracy *t-value* (1.983) and negative loss *t-value* (-1.199) suggest a slight superiority of *MCNN-GA* over *ThiNet30* in terms of classification performance. However, since the *p-value* of accuracy and loss are 0.062 and 0.245, respectively (values greater than the $\alpha = 0.05$ designated for the statistical test), the null hypothesis cannot be rejected. Thus, both methods are considered to have the same quality in the task of classification/identification of game events (with respect to the Dota2 dataset).

Regarding the network size and inference time, *MCNN-GA* generated significantly better results than *ThiNet30* (since the *p-value* for both parameters are smaller than α with a 95% confidence interval). So, it is concluded that despite presenting the same classification performance, the former brings the advantage of generating much lighter networks with faster data inference time than the latter. These results indicate that the best models generated by *MCNN-GA* are more suitable considering a real-time setting. This is because the depth of the models (number of layers) generated by this method is smaller than that of *ThiNet30* (and consequently *ResNet152*), which speeds up the prediction phase.

5 CONCLUSION AND FUTURE WORKS

This work investigated two different approaches (one based on genetic algorithm and the other based on bayesian optimization) in order to generate CNNs optimized in terms of classification performance and network size without the need to rely on pre-trained

Table 5: *T-test* applied to the results between methods *BO* and *MCNN-GA*.

Accuracy		**Loss**		**Size**		**Inference Time**		**Execution Time**		**Fitness**	
t-value	**p-value**	**t-value**	**p-value**	**t-value**	**p-value**	**t-value**	**p-value**	**t-value**	**p-value**	**t-value**	**p-value**
-3.125	**0.006**	**3.048**	**0.007**	**4.455**	**0.0003**	0.0	1.0	**-3.966**	**0.001**	**5.596**	**0.00003**

Table 6: *MCNN-GA* and *ThiNet30* comparison results (mean value).

Method	Accuracy	Loss	Size	Inference Time
MCNN-GA	99.35	0.099	35026.0	5.4
ThiNet30	99.10	0.113	6700576	6.5

Table 7: *T-test* applied to the results between methods *MCNN-GA* and *ThiNet30*.

Accuracy		**Loss**		**Size**		**Inference Time**	
t-value	**p-value**	**t-value**	**p-value**	**t-value**	**p-value**	**t-value**	**p-value**
1.983	0.062	-1.199	0.245	**-9968.12**	**0**	**-5.312**	**0.00004**

networks. The case study used to validate the analysis was the Dota2 game event dataset. The results showed that *MCNN-GA* generated CNNs which achieved a classification performance as good as the best model produced in (Luo et al., 2019a) (ThiNet30), but with significantly fewer parameters (resulting in models with less memory usage). It means that *MCNN-GA* have a great potential to generate highly efficient and suitable models for real-time applications (which can be transferable to domains beyond games), which helps in hardware accessibility for complex tasks.

As future works, the authors intend: to investigate the performance of the approaches studied here for different types of games (simpler 2D games like Super Mario Bros and more realistic 3D games like Skyrim); to publish the framework investigated here for the community's use; and, finally, to use such approaches to build real-time mechanisms that can help people with cognitive difficulties.

REFERENCES

Azadvar, A. (2021). Predictive psychological player profiling.

Coleman, A. M. (2009). *A dictionary of psychology / Andrew M. Colman.* Oxford University Press Oxford ; New York, 3rd ed. edition.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255.

Faria, M. P. P., Julia, R. M. S., and Tomaz, L. B. P. (2020). Improving fifa player agents decision-making architectures based on convolutional neural networks through evolutionary techniques. In Cerri, R. and Prati, R. C., editors, *Intelligent Systems*, pages 371–386, Cham. Springer International Publishing.

Han, S., Mao, H., and Dally, W. J. (2016). Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In *4th International Conference on Learning Representations*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications.

Luo, J.-H., Wu, J., and Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5068–5076.

Luo, Z., Guzdial, M., Liao, N., and Riedl, M. (2018). Player experience extraction from gameplay video. *CoRR*, abs/1809.06201.

Luo, Z., Guzdial, M., and Riedl, M. (2019a). Making cnns for video parsing accessible. *CoRR*, abs/1906.11877.

Luo, Z., Guzdial, M., and Riedl, M. (2019b). Making cnns for video parsing accessible: Event extraction from dota2 gameplay video using transfer, zero-shot, and network pruning. In *Proceedings of the 14th International Conference on the Foundations of Digital Games*, FDG '19, New York, NY, USA. Association for Computing Machinery.

Miller, B. L., Miller, B. L., Goldberg, D. E., and Goldberg, D. E. (1995). Genetic algorithms, tournament selection, and the effects of noise. *Complex Systems*, 9:193–212.

Mockus, J. B. and Mockus, L. J. (1991). Bayesian approach to global optimization and application to multiobjective and constrained problems. *J. Optim. Theory Appl.*, 70(1):157–172.

Moosa, A. M., Al-Maadeed, N., Saleh, M., Al-Maadeed, S. A., and Aljaam, J. M. (2020). Designing a mobile serious game for raising awareness of diabetic children. *IEEE Access*, 8:222876–222889.

Soomro, K., Zamir, A., and Shah, M. (2012). Ucf101: A dataset of 101 human actions classes from videos in the wild. *CoRR*.

Srinivas, M. and Patnaik, L. M. (1994). Genetic algorithms: a survey. *Computer*, 27:17–26.

Wijman, T. (2019). The global games market will generate $152.1 billion in 2019 as the u.s. overtakes china as the biggest market.

Wong, J. and Gales, M. (2016). Sequence student-teacher training of deep neural networks. pages 2761–2765.

Xu, M., Gao, M., Chen, Y.-T., Davis, L. S., and Crandall, D. J. (2019). Temporal recurrent networks for online action detection.