

# DLNE: A Hybridization of Deep Learning and Neuroevolution for Visual Control

Andreas Precht Poulsen, Mark Thorhauge, Mikkel Hvilshj Funch, Sebastian Risi  
Center for Computer Games Research  
IT University of Copenhagen, Denmark  
{appo, marth, mhvf, sebr}@itu.dk

**Abstract**—This paper investigates the potential of combining deep learning and neuroevolution to create a bot for a simple first person shooter (FPS) game capable of aiming and shooting based on high-dimensional raw pixel input. The deep learning component is responsible for visual recognition and translating raw pixels to compact feature representations, while the evolving network takes those features as inputs to infer actions. Two types of feature representations are evaluated in terms of (1) how precise they allow the deep network to recognize the position of the enemy, (2) their effect on evolution, and (3) how well they allow the deep network and evolved network to interface with each other. Overall, the results suggest that combining deep learning and neuroevolution in a hybrid approach is a promising research direction that could make complex visual domains directly accessible to networks trained through evolution.

## I. INTRODUCTION

The field of artificial intelligence for games has seen significant advancements in the last few years, using new combinations of existing algorithms. Deep neural networks have been a central component in these advancements, and have proven to be a powerful representation, capable of solving a wide range of problems. Google DeepMind’s AlphaGo [21] used deep neural networks and Monte Carlo tree search to achieve remarkable results, beating a professional Go player for the first time in history.

While the fundamental algorithms behind these solutions have existed for decades, the driving factor has primarily been combining and adjusting them to fit the problem domain, as well as having large amounts of data and computational resources available. This naturally raises the question of what the potential of these algorithms is, and which problems they can solve. Inspired by the work of both Koutník et al. [10] and Chen et al. [2], who experimented with feature detection through convolutional networks for autonomous driving, and neuroevolution in visual decision making in a car racing simulation game, this paper investigates the combination of neuroevolution and deep convolutional neural networks to learn directly from raw visual input in a first person shooter (FPS) setting.

The main idea of the approach is to separate the visual recognition (VRC) and action inferring component (AIC) for visual control (Figure 1). A deep convolutional network is trained in a *supervised* fashion through gradient descent to determine the position of an enemy bot based on high-dimensional raw pixel input. This positional information is

then used as input to another network that is evolved to aim and shoot at a given target. Importantly, the evolving network is trained in a *non-supervised* way, i.e. it only relies on a fitness function and not on a large number of labeled examples. While some work exists in combining evolution and deep learning for the generation of 2D artefacts [13, 15] and 3D objects [12], the hybrid combination of these two techniques is an unexplored area in visual control. In fact, this combination could combine the advantages of both methods. Neuroevolution has difficulties scaling to problems with a large number of inputs [18], such as 3D shooting games [16], which could be solved with a deep learning-based visual recognition component. On the other hand, evolutionary-based approaches do not rely on differentiable architectures, and work well in domains with sparse rewards, a challenge for most deep reinforcement learning approaches [6, 14].

The purely neuroevolutionary-based approach by Koutník et al. [10] has comparable advantages. It includes two networks, which are both trained through evolution: a convolutional network that learns to transform raw visual input into compact features, and a recurrent neural controller. While Koutník et al.’s approach was able to evolve a controller for the TORCS racing simulator using only visual input, the training time is a concern; it takes almost 40 hours on an 8-core machine to evolve a controller that is capable of driving smoothly. A convolutional network trained with supervised learning could provide a better foundation for neuroevolution in terms of evolutionary speed, as the feature representation of the visual state is explicitly designed for neuroevolution.

The FPS game DOOM [9, 11] and Atari 2600 games [14] have seen several implementations of deep reinforcement learning. While yielding impressive results (and it is likely that our approach requires significantly more development to achieve a similar performance), training a visual recognition component and action inferring component separately could have advantages. For example, we imagine it could potentially lead to several application opportunities in the field of real-world robotics, as reinforcement learning directly from visual perception to action is very time consuming [14] and challenging due to the difficulty of simulating the real world with realistic visual inputs.

The goal of this paper is an initial proof-of-concept of how deep learning and neuroevolution could be combined to create a neural controller for a FPS agent capable of aiming and

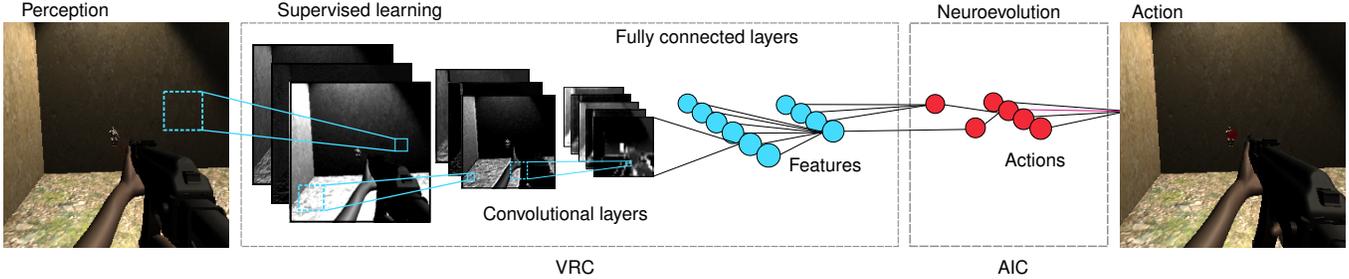


Fig. 1: **The DLNE Hybrid Approach.** The combination of deep learning and neuroevolution translates the high-dimensional visual state to actions. A deep network visual recognition component (VRC) is trained in a supervised way to identify important game features (e.g. location of the enemy), which are then provided as input to an evolving network that infers actions (AIC).

shooting, only using the raw visual input of the game. An important question in this context is which feature representation of a visual partially observable state makes the combination of neuroevolution and supervised deep learning perform well. We aim to answer these research questions implementing two deep learning-based recognition components with different feature representations, as well as two matching representations for the neuroevolution approach. The results in this paper demonstrate that a particular representation based on partitioning the visual input works well together with neuroevolution and hints at the promise of scaling neuroevolution to more complex domains by exploiting the advances of recent deep learning approaches.

## II. BACKGROUND

This section explains the theoretical framework of the approach presented here and related work on agents learning from raw visual input in game environments.

### A. Deep Learning

Although the idea of training multi-layer neural networks through back-propagation of error is not new, advances in computational power, in the availability of data, and in the understanding of many-layered ANNs, have culminated in a high-interest field called *deep learning* (DL) [6]. The basic idea is to train many-layered (deep) neural networks on big data through stochastic gradient descent (SGD).

Deep learning approaches now achieve cutting-edge performance in diverse benchmarks, including image, speech, and video recognition; natural language processing; and machine translation [6]. Such techniques are generally most effective when the task is *supervised*, i.e. the objective is to learn a mapping between given inputs and outputs, and when training data is ample. Importantly, the deep neural network (DNN) must be composed only from differentiable operations, which limits the type of neural architectures and problems it can directly be applied to.

One focus of DL is object recognition, for which the main benchmark is the ImageNet dataset [3]. ImageNet is composed of millions of images, labeled from 1,000 categories spanning diverse real-world objects, structures, and animals. DNNs trained on ImageNet are beginning to exceed human levels of performance [7], and the learned feature representations of

such DNNs have proved useful when applied to other image comprehension tasks [17].

The stereotypical deep learning approach is to construct a network from alternating layers of convolution and pooling, which facilitate learning increasingly higher-level translation-invariant features. These convolutional layers often feed into fully-connected layers and ultimately into a classification layer. The approach in this paper builds on such deep convolutional neural networks (DCNN) to detect the location of an enemy in a 3D environment that are provided as compact input features for a network evolved by NEAT, which is explained next.

### B. NEAT

NeuroEvolution of Augmenting Topologies (NEAT) [19] is an evolutionary algorithm for evolving both the topology and weights of an artificial neural networks. NEAT starts with a population of simple neural networks and then adds complexity over generations by adding new nodes and connections through mutations. By evolving networks in this way, the topology of the network does not need to be known a priori; NEAT searches through increasingly complex networks to find a suitable level of complexity. Because it starts simply and gradually adds complexity, it tends to find a solution network close to the minimal necessary size. For a complete overview of NEAT see Stanley et al. [19].

### C. Related Work

This section presents some related work aimed at allowing agents to learn directly from raw visual input. In 2014, a visual agent for playing TORCS was developed using reinforcement learning to evolve a network with a CNN component [10]. The fitness functions used to train the top layer network were different than the fitness function for the CNN component. The top layer network adapted the output of the CNN component and successfully used it, despite the output of the CNN component being humanly incomprehensible, and based solely on the image.

Chen et al. [2] trained a DCNN to detect features of road images in TORCS, retextured to simulate real driving. The features included angle of the road tangent and distance to land markings. The features are similar to the angular representation we employ (Section III-B). Chen et al. trained their network

with 484,815 training examples for 140,000 iterations, which is quite resource intensive. The features were translated to driving actions by a handwritten controller, which is the main difference to the work presented here.

The FPS shooter DOOM has seen several successful AI implementations using convolutional neural networks with reinforcement learning in the Visual DOOM AI competition [9, 11]. The game domain is similar to the one presented in this paper, but the training methodology differs. Many of the top performing entries for the competition used deep reinforcement learning to estimate the Q-value of states. This way of controlling the agent is quite different from our approach, as it is based on a state representation of the game and does not utilize neuroevolution.

Deep reinforcement learning has been applied to several different games, using high dimensional sensory input, such as images. Mnih et al. [14] used deep reinforcement learning to play Atari 2600 games, surpassing human level performance in many of them. One of the advantages of this approach is that it is independent of the feature representation, which can be a challenge to design manually. However, most deep reinforcement learning approaches still struggle on games requiring long term strategic planning or games that only provide sparse rewards, and in general still take a relatively long time to train.

Numerous examples of applying supervised learning or evolutionary algorithms to FPS games exist [1, 4] and neuroevolution is frequently applied to many different types of games [18]. Some of the examples employ the indirect HyperNEAT encoding [20] to deal with relatively high dimensional input ( $16 \times 21$  for Atari and  $7 \times 7$  for Go, compared to  $256 \times 256$  images used in this paper), as HyperNEAT can evolve repeating connection and topological patterns.

### III. APPROACH: INTERFACING DEEP LEARNING AND NEUROEVOLUTION

The basic idea behind the *Deep Learning Neuroevolution* (DLNE) approach presented in this paper is shown in Figure 1. A DCNN preprocesses the scene and extracts relevant features (e.g. distance to enemies, etc.), trained through a supervised gradient descent approach. Once this network is trained it can be used in a pipeline, in which the extracted features are inputs for a network evolved through NEAT. The NEAT network can either be trained independently beforehand with the ground truth position of the enemy extracted from the engine (the approach we follow in this paper), or directly with the already trained DCNN. The idea is that by separating the visual recognition part from the training of the agent’s policy, we can combine some of the advantages of neuroevolution and deep learning-based approaches. However, how to best combine such different approaches is an open question that the experiments in this paper are trying to elucidate.

#### A. Experiments

In the proof-of-concept experiments in this paper the goal of the agents is to aim and shoot a stationary enemy. The position

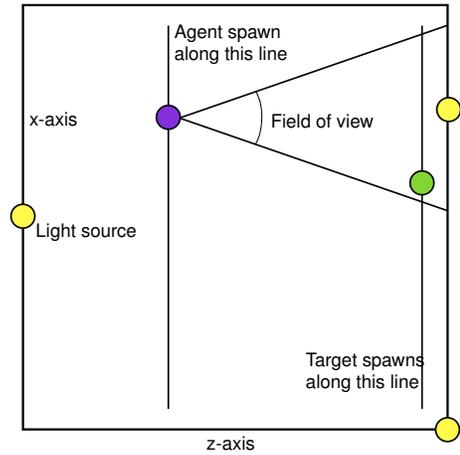


Fig. 2: The arena as seen from above. This illustration omits the  $y$ -axis. The lines represent planes, as the agent and the target spawn randomly along both the  $y$ -axis and the  $x$ -axis.

of the agent is fixed but it can turn, shoot and reload. The evolving network has four turning outputs (*up*, *down*, *left*, *right*), which are scaled to allow a maximum turn speed of 100 degrees/second. Two additional outputs allow the agent to decide when to *shoot* and *reload*, which are binary actions that are triggered when the corresponding network outputs are above a certain threshold.

The game’s arena is quadratic, with the agent spawning on one side and the targets spawning on the other, as seen in Figure 2. Both the agent and the target spawn in a random  $x$  and  $y$ , while the  $z$  coordinate is constant. The first target in the arena always spawns within sight of the agent in order to facilitate early learning. The arena is outfitted with three lights to create different visual variations of the target and weapon overlay. One light is in the middle of the arena, behind the agent. The other two lights are placed on the side of the target, such that they produce varied lighting conditions depending on the distance of the target. This setup ensures that the visual recognition component is tested under a variety of different visual conditions.

The agent starts with an automatic weapon that has 30 bullets in a single magazine and has six extra magazines at its disposal. The weapon does not automatically reload when the magazine is empty, forcing the agent to learn to reload at the right time. If the agent reloads a non-empty magazine, the remaining bullets are lost. Since no information is given regarding bullets and magazines left, the agent should ideally learn to count how much ammo was used. This is important in order to not waste time attempting to shoot or reload when there are no more bullets or the weapon is fully loaded, it is however possible to get very reasonable behaviour from the agent without perfect reload times. The target spawns with 100 health points, and a single bullet hit causes 10 damage. Therefore, 10 hits are required to kill the target.

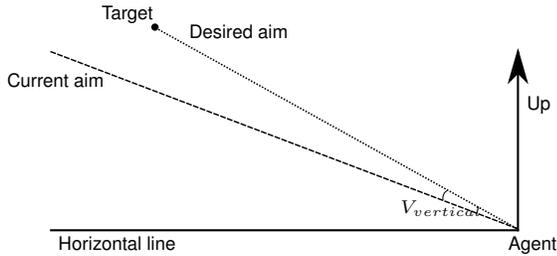


Fig. 3: The vertical angle of the Angular Representation (AR).

### B. Feature Representations

Recall that the feature representation is the integration point between the DCNN and the evolved network. The feature representation of the visual state should allow the evolving network to locate an enemy, aim and shoot it, while having as few dimensions as possible, as the evolutionary speed is a central concern. Additionally, the DCNN has to be able to learn the features from training examples through supervised learning. We compare two different visual representations: the angular and the visual partitioning representation.

**Angular Representation (AR).** The angular feature representation unambiguously defines the position of the target on the screen. This representation has a total of four outputs: two angles, a distance and a binary output indicating whether the target is within sight. The vertical angle (Figure 3) is defined by the angle between the current facing of the agent and a projection of the vector from the agent to the target. The projection is onto a plane determined by the vector of the current facing and the up-vector. The horizontal angle is calculated in the same manner, except that the angle is calculated based on the projection of both vectors onto the horizontal plane. These two angles allow the NEAT network to infer the aiming direction. To allow for changing shooting behaviour based on distance to the enemy, this information is also provided. For example, if the agent shoots a full automatic rifle at a long distance, it might be better to fire separate shots than using automatic fire. The binary indication of a target within sight indicates to the agent whether the given angles reflect the position of a target, or just assume default values.

**Visual Partitioning Representation (VPR).** The visual partitioning representation defines the position of the target as a classification task, where each point on screen belongs to a class bounded by a square (Figure 6).

As the target can be visually present in multiple squares, the correct square is defined as the square that contains its center of mass. The partitioning is finer in the center of the agent’s view field, allowing for more accurate aim adjustments the closer the target is to the line-of-fire. The feature representation is a sparse vector with an entry for each partition. The entry of the partition that contains the target is one, while all others are zero. If no partition contains the target, an additional term indicates the absence of a target. The feature representation of the VPR has a significantly larger number of dimensions than the AR (25 compared to 4). The VPR is less detailed and

bound by the width of the partitions. Additionally, the AR allows the agent to aim precisely, which many FPS shooter games reward by increasing the damage inflicted for hitting vulnerable areas on the target. However, we speculate the VPR representation might be easier for the DCNN to learn.

### C. Neuroevolutionary Training

The training of the agent was performed in the Unity 5 game engine<sup>1</sup> with the UnityNEAT framework<sup>2</sup>, which is a port of the C# NEAT framework SharpNEAT<sup>3</sup>.

Instead of evolving the network with the DCNN as input, which is computationally expensive, here the evolving network is trained with the ground truth information from the game engine itself (e.g. exact position of the enemy). A fitness evaluation lasts for 15 seconds, and as the evaluation includes some randomness, each evaluation is repeated 20 times and the results are averaged. The random spawning locations of the target and agent ensure that the agent does not learn a specific pattern, but instead learns a general policy.

The agent is rewarded for hitting the target and aiming close to it. The fitness can be formulated as:  $f = damage + aim\_bonus$ , where  $damage$  is the total damage dealt, and  $aim\_bonus$  is calculated as:  $aim\_bonus = \frac{1}{N} \sum_n \frac{c}{(1+v(n))^2}$ , where  $N$  is the number of frames and  $v(n)$  is the angle measured in radians between the forward pointing vector of the agent and the line from the agent to the target in frame  $n$ . Hence  $v(n) = 0$  if the agent aims directly at the target. The constant  $c$  was set to 75, which was found to work well through prior experimentation; because a killed agent does generate  $aim\_bonus$ , lower  $c$  values sometimes lead to agents that refrain from dealing damage to the target.

The size of each population is 100 with 20% elitism and a termination criterion of 260 generations. Sexual offspring (20%) does not undergo mutation. Asexual offspring (80%) has 0.8 probability of weight mutation, 0.05 chance of link addition, and 0.03 chance of node addition. Connections are deleted with a 0.01 probability.

**Data generation for vision module training.** In order to generate the data necessary for the supervised training of the DCNN (explained in the next section), we record gameplay (images taken from random positions and camera rotations, and enemy locations) from random NEAT networks that use the angular game state as input. The idea is to try to sample from states the agent might find itself in during actual play. In the future it will be interesting to perform this training interleaved with training of the DCNN or using a network evolved on the ground truth. To increase training speed, the number of images without an enemy are reduced from approx. 50% to 15%, resulting in a total of 130,000 training images.

### D. DCNN Architecture and Training

Both the VPR classification and AR regression networks are tested with a 12 and a 6 layered architecture, referred to as

<sup>1</sup><https://unity3d.com/>

<sup>2</sup><https://github.com/lordjesus/UnityNEAT>

<sup>3</sup><http://sharpneat.sourceforge.net/>

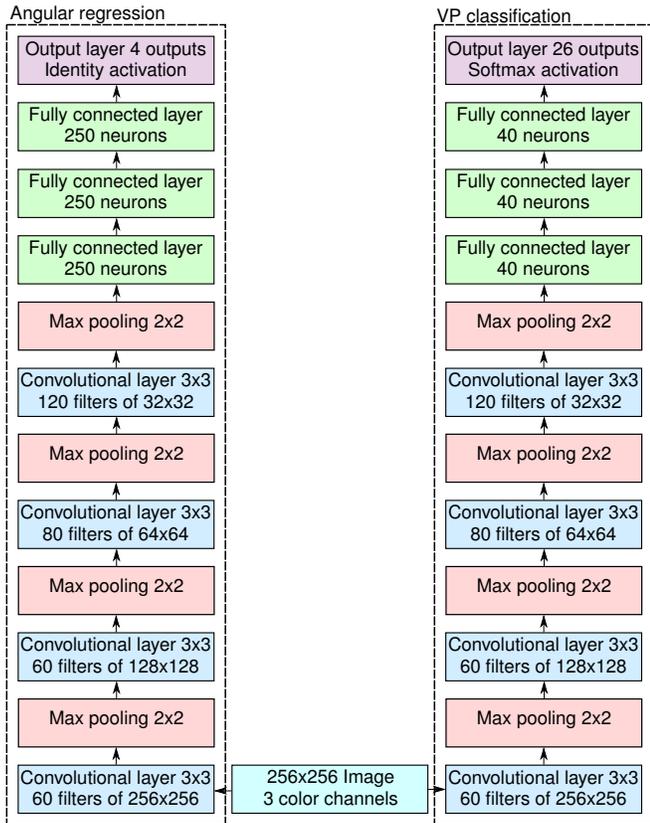


Fig. 4: The topology of the 12-layer DCNN using the AR representation (left) and VPR representation (right).

deep and shallow respectively. The difference between the two networks is in the number of neurons in the fully connected layers and in the activation functions used for the output (Figure 4). All activation functions employ the rectified linear unit (ReLU), except in the output layers; here a softmax is used for classification and identity is used for regression. The regression network outputs the angular representation, scaled into the range  $[-1, 1]$ . The classification network outputs 26 probabilities summing to 1. The networks takes as input an RGB image with a size of  $256 \times 256 \times 3$ .

Of the deep network’s 12 layers, the first 8 alternate between convolutional and pooling layers, while the next 3 layers are fully connected followed by the output layer. The stride and zero padding of all the convolutional layers are 1, while the stride of all the pooling layers is 2. Both networks have a total of 163,940 parameters (weights and biases) in the convolutional layers. The classification network has 1,228,800 weights from the last convolutional layer to the first fully connected layer and 4,386 parameters in the remaining layers.

The shallow convolutional network has a similar architecture but only 6 layers, the first 4 being alternating convolutional and pooling layers, followed by a fully connected layer and an output layer. The stride and zero padding of all the convolutional layers are 1, while the stride of all the pooling layers is 4. Both networks have a total of 246,200 parameters



Fig. 5: The two different visual settings V1 (left) and V2 (right). In contrast to V1, V2 has a weapon overlay that can occlude the view of the enemy bot and includes more detailed textures and dynamic lighting.

in the convolutional layers. The classification network has 3,686,400 weights from the last convolutional layer to the first fully connected layer and 3,266 parameters in the remaining layers. The regression network has 23,040,000 weights from the last convolutional layer to the first fully connected layer and 3,754 parameters in the remaining layers.

The parameters are updated from a mini-batch of size 32, with a learning rate of  $10^{-3}$ . The process does not include dropout, but L2-regularisation with a coefficient of  $5 \cdot 10^{-4}$ . Nesterov’s accelerated gradient is used with a momentum coefficient of 0.9. The networks are initialised with Xavier initialisation. The framework used for supervised learning is DL4J, a deep learning framework for Java, accessible at <https://deeplearning4j.org>. In order to allow communication between the trained DCNN and evolved network, a simple socket bridge was implemented. The overhead introduced by this bridge is minimal as it is possible to send the required data back and forth 10,000 times in approximately three seconds.

**Visual Settings.** We created two different data sets that test the network under different difficulty levels (Figure 5). In the more complex setup V2, the player and weapon overlay can partially or fully cover the target, making the recognition task harder, or even impossible. Additionally, the textures in V2 are more detailed and the lighting is dynamic.

#### IV. RESULTS

**Deep Network Accuracy.** The accuracy of the VPR (Figure 7) is measured as the percentage of correct predictions. The topologies of the networks do not seem to have a significant impact on the accuracy. The training examples that the networks failed to predict correctly were either when the target was in between partitions, or behind the weapon overlay. The networks were able to classify the partition of the enemy correctly, even when it was difficult to locate, as seen in Figure 6. Varying lighting did not seem to cause incorrect predictions. These results suggest that the VPR is well suited to detect the location of enemies in this experiment.

The performance measure of the AR representation is the absolute error on the predicted target distances. The distance error of the deep network is 0.0893 in V1 and 0.1614 in



Fig. 6: The Visual Partitioning Representation (VPR) is more fine grained in the direction the gun is pointing. The trained deep CNN with VPR representation correctly predicts the class (green square) with a confidence of 55.7%, with only four pixels of the target being visible.

V2. The shallow network reaches a distance error of 0.153 in V1 and 0.1658 in V2. The distance error is almost twice as high with the more complex V2 setup and, while more definite conclusions will require further testing, increasing network depth does seem to have a greater effect on the AR representation than on the VPR representation.

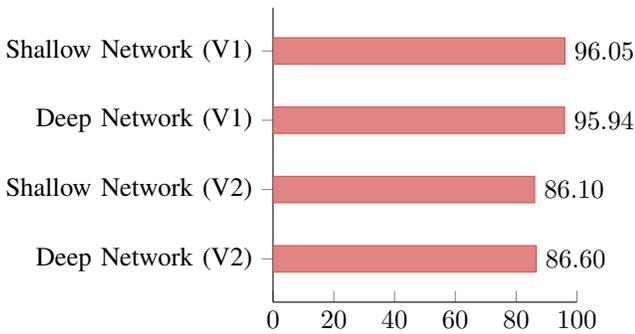


Fig. 7: Accuracy of the best deep and shallow DCNN with VPR representation on the two different visual settings V1 and V2.

**DCNN Feature Map Example.** The feature maps shown in Figure 8 are visualised by scaling the output range  $[0, 1]$  of every neuron in the convolutional layers linearly to the grey scale range  $[0, 255]$ . The feature maps from the two different representations are different in both the magnitude and the variance of the output. The network estimating the AR has often fewer “useful” feature maps in the last max pooling layer and a tendency to highlight features that are potentially less relevant for the task at hand, such as the weapon overlay.

**Evolutionary Training.** A total of ten independent evolutionary runs were performed for each of the two neuroevolution setups: networks with VPR and networks with AR as input

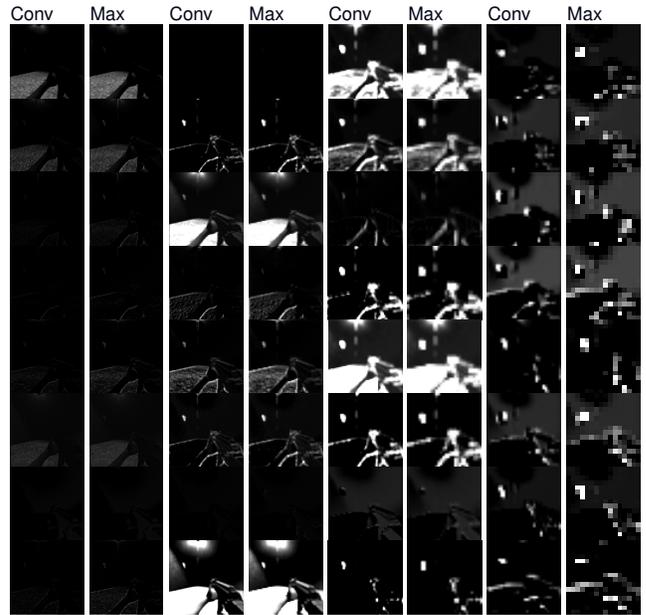


Fig. 8: A small subset of the feature map activations of a DCNN with VPR representation. The feature maps highlight the position of the target. The leftmost column of feature maps comes from the first conv layer, and the rightmost max pooling layer is the input to the fully connected layers.

representation trained using ground truth enemy locations. The average final fitness for the VPR setup is 318,4 ( $sd = 58,36$ ), while it is 436,9 ( $sd = 30,12$ ) for AR. This difference is significant ( $p < 0.01$ ; Mann Whitney U-test). Each run took approximately 10 hours to complete. We observe an overall tendency of the AR representation to perform better than the experiments with the VPR, both learning faster and reaching a higher fitness.

**Hybrid Approach.** Here we report results on taking the best network found during evolution on the ground truth enemy locations combined with the best 12-layer DCNN we found during supervised training (for both VPR and AR representations). In other words, the two networks used for the hybrid approach are trained separately and no further training is performed once they are combined.

While the game is running with 10 Hz during evolutionary training, this frequency is halved during testing of the combined approach due to the higher computational costs of the DCNN in the pipeline. We decided to use the deeper 12-layer networks for the combined approach, since this setup performed as good on the VPR representation but had a lower distance error on AR. We applied the hybrid approach to the more complex V2 setup that includes more detailed textures and dynamic lighting, and an additional setup that resembles V2 but without the weapon overlay (V2-no-overlay). Figure 9 shows the performance of these treatments averaged over 200 trials. The evolved network with AR representation as input reaches a significantly higher score using the ground truth than any other method ( $p < 0.05$ ; all tests are two-tailed Mann

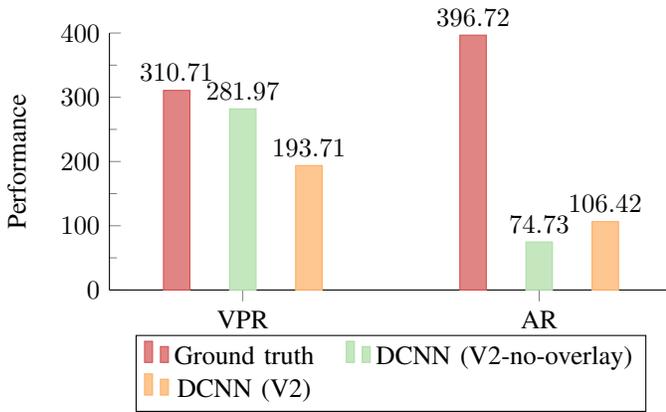


Fig. 9: Comparing DCNN and ground truth as features for the evolved network for visual settings V2 and V2 without the weapon overlay (V2-no-overlay). Testing performance is calculated in terms of fitness (see Section III-C).

Whitney U-tests) but the distances and angles returned by the DCNN component are not accurate enough for the network to perform well. In the case of the VPR, performance decreases slightly (though not significantly) from ground truth to visual setting V2-no-overlay, but significantly when compared to the setting that includes the overlay ( $p < 0.05$ ).

The main conclusion is that, while the combination of the two techniques is not perfect (see the example video of the VPR hybrid approach at: <https://youtu.be/daFvJa90f8Y>), the hybrid approach performs reasonably well and is in fact able to aim&shoot based on the raw pixel representation of  $256 \times 256 \times 3$  images. With 22.87/31.31 hits averaged over the 200 trials, the best hybrid approach has a shooting accuracy of 42%.

## V. DISCUSSION AND FUTURE WORK

### A. The DCNN Component

The incorrect VPR predictions of images where the target is present are generally cases where the target could be classified as being in either partition, depending on its exact location of the center of mass. Consequently, such slightly incorrect predictions do not have a significant impact on the NEAT network that relies on these features as inputs. These inaccuracies should therefore be viewed as a natural consequence of the vague VPR classification definition, and not a failure of the optimisation of the model.

The AR results show varying degrees of success in regards to estimating the target’s location to an adequate level. The topology of the network has a significant impact on the error of the horizontal angle, vertical angle and distance, especially when trained without visual distortion. In the future, the testing error could likely be reduced further by tweaking the network’s topology or increasing the amount of training data.

The error of the deep network to estimate the AR in V2 is more than twice as large as in the simpler setting V1. Recall that in contrast to V1, V2 includes dynamic lighting,

detailed textures and weapon overlay. These results raise the question of how much each of these factors contribute to the error of the network. In fact, the network has a large error on some examples where the weapon overlay does not even partially cover the target, which suggests that dynamic lighting and textures are more difficult to deal with for the AR representation than for VPR.

The performance increase between the shallow and deep network for the AR setup (and preliminary experiments with deeper architectures) suggest that the AR could be improved by increasing the number of layers or tuning the learning process. The network trained by Chen et al. [2] is estimating features similar to the AR, but uses far more training data and learns for more iterations. The difference in quality of feature maps also indicates that the convolutional layers of the network estimating the AR are not as functional as the convolutional layers of the network estimating the VPR, which solidifies this assumption. From a theoretical point of view, the difference in the quality of feature maps is not surprising, as the negative log likelihood cost function optimises better than the euclidean loss cost function used for regression, as described and visualised by Glorot et al. [5].

### B. The Neuroevolution Component

The results of the neuroevolutionary training show that NEAT is capable of producing networks that can aim and shoot in a FPS game, using both the AR and the VPR input representation. However, the applicability of these networks to other FPS games is challenged by their tendency to learn from the particularities of the training setup. The distance to the target does not vary much, which the evolved networks exploit to implement reloading and shooting behaviour based on a fixed distance to the target; the relationship between degrees rotated and the number of pixels that the target shifts on the screen is almost proportional. Networks based on AR can exploit the fact that a new target spawns immediately after an elimination by reloading when the target is near the edge of the screen. This behaviour tends to result in a single reload as the time it takes to reload is greater than the time it takes to aim directly at the target. The VPR-based evolved network associates specific partitions with reloading; consequently, none of network learns generalisable reloading behaviour. Proper reloading behaviour requires either information about the number of shots left or some form of memory through recurrent connections, which the networks currently do not seem to evolve. Running evolution for a longer timespan could alleviate some of these challenges. In the future, it will be important to investigate the potential of this approach in FPS game environments that are more varied and more complex.

### C. The Combined Approach

The results show a significant difference between the performance of the AR and the VPR representation when combined with an evolved network. The hybrid approach with AR clearly performs worse than VPR; the estimation of the AR is far too inaccurate for the evolved network to successfully infer an

adequate action, and has its fitness reduced by 73.2% when using the DCNN as input to the evolving network.

The combined approach proved performance intensive, and the overall performance of the agent is certainly reduced by running with only 5 Hz instead of the original 10. While the combined approach could be improved, and studying these hybrids in more depth is an important future research direction, the results in this paper do demonstrate that a DLNE hybrid can learn to aim and shoot from high-dimensional raw input.

The modularity of the combined approach makes it applicable to more complex games but also domains outside of games, such as real-world robotics. It is often challenging to apply neuroevolution to physical robotics directly, as the evaluations take considerable time. Consequently, the neural networks controlling the robots are often evolved in a simulator and only afterwards transferred to the real robot. However, one major issue with this transfer is the *reality gap* [8] and especially robot sensors are notoriously difficult to model. The abstraction provided by the hybrid approach could allow neuroevolution to be applied to physical robots in a novel way: A DCNN can be trained on labeled real-world images or other robot sensory data, while the neuroevolutionary component can be trained without a visually realistic environment and without the need for labeled data in a simulator.

## VI. CONCLUSIONS

This paper investigated the potential of combining neuroevolution and deep learning in the context of a FPS game and experimented with two different feature representations: a visual partitioning and an angular representation. Trained on ground truth data, it was easier for NEAT to evolve a policy based on the angular input than on the visual partitioning input. However, the VPR representation was easier to train with deep learning and therefore the combination of an evolved network that takes the predictions of the correct visual partition of the enemy as input worked best overall. The presented hybrid approach opens up a new research direction in neuroevolution by combining some of the advantages of gradient and evolutionary-based approaches. Extending DLNE to more complex games and applications in the future could be a fruitful application area for such a hybrid approach.

## REFERENCES

- [1] S. Bonacina, P. Luca Lanzi, and D. Loiacono. Evolving dodging behavior for openarena using neuroevolution of augmenting topologies. 2008.
- [2] C. Chen, A. Seff, A. L. Kornhauser, and J. Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. *CoRR*, abs/1505.00256, 2015. URL <http://arxiv.org/abs/1505.00256>.
- [3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.
- [4] B. Geisler. An empirical study of machine learning algorithms applied to modeling player behavior in a first person shooter video game. 2002.
- [5] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2010, Chia Laguna Resort, Sardinia, Italy, May 13-15, 2010*, pages 249–256, 2010. URL <http://www.jmlr.org/proceedings/papers/v9/glorot10a.html>.
- [6] I. Goodfellow, Y. Bengio, and A. Courville. Deep learning. Book in preparation for MIT Press, 2016. URL <http://goodfeli.github.io/dlbook/>.
- [7] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.
- [8] N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. *Advances in artificial life*, pages 704–720, 1995.
- [9] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski. Vizdoom: A doom-based AI research platform for visual reinforcement learning. *CoRR*, abs/1605.02097, 2016. URL <http://arxiv.org/abs/1605.02097>.
- [10] J. Koutník, J. Schmidhuber, and F. Gomez. Evolving deep unsupervised convolutional networks for vision-based reinforcement learning. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, GECCO '14*, pages 541–548, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2662-9. doi: 10.1145/2576768.2598358. URL <http://doi.acm.org/10.1145/2576768.2598358>.
- [11] G. Lample and D. S. Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521, 2016. URL <http://arxiv.org/abs/1609.05521>.
- [12] J. Lehman, S. Risi, and J. Clune. Creative generation of 3d objects with deep learning and innovation engines. In *Proceedings of the 7th International Conference on Computational Creativity*, 2016.
- [13] A. Liapis, H. P. Martínez, J. Togelius, and G. N. Yannakakis. Transforming exploratory creativity with DeLeNoX. In *Proceedings of the Fourth International Conference on Computational Creativity*, pages 56–63, 2013.
- [14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.
- [15] A. Nguyen, J. Yosinski, and J. Clune. Innovation engines: Automated creativity and improved stochastic optimization via deep learning. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2015.
- [16] M. Parker and B. D. Bryant. Neuro-visual control in the quake ii game engine. In *Neural Networks, 2008. IJCNN 2008.(IEEE World Congress on Computational Intelligence). IEEE International Joint Conference on*, pages 3828–3833. IEEE, 2008.
- [17] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2014 IEEE Conference on*, pages 512–519. IEEE, 2014.
- [18] S. Risi and J. Togelius. Neuroevolution in games: State of the art and open challenges. *IEEE Transactions on Computational Intelligence and AI in Games*, 9(1):25–41, March 2017. ISSN 1943-068X. doi: 10.1109/TCIAIG.2015.2494596.
- [19] K. O. Stanley and R. Miikkulainen. Evolving neural network through augmenting topologies. *Evolutionary Computation*, 10(2):99–127, 2002. doi: 10.1162/106365602320169811. URL <http://dx.doi.org/10.1162/106365602320169811>.
- [20] K. O. Stanley, D. B. D’Ambrosio, and J. Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial Life*, 15(2):185–212, 2009. doi: 10.1162/artl.2009.15.2.15202. URL <http://dx.doi.org/10.1162/artl.2009.15.2.15202>.
- [21] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014. URL <http://arxiv.org/abs/1409.4842>.