Dealing with Adversarial Player Strategies in the Neural Network Game iNNk through Ensemble Learning

Mathias Löwe  
malw@itu.dk  
IT University of Copenhagen  
Copenhagen, Denmark

Jennifer Villareale  
jmv85@drexel.edu  
Drexel University  
Philadelphia, Pennsylvania, USA

Evan Freed  
emf67@drexel.edu  
Drexel University  
Philadelphia, Pennsylvania, USA

Aleksanteri Sladek  
am939@drexel.edu  
Drexel University  
Philadelphia, Pennsylvania, USA

Jichen Zhu  
jichen.zhu@gmail.com  
Drexel University  
Philadelphia, Pennsylvania, USA

Sebastian Risi  
sebr@itu.dk  
IT University of Copenhagen  
Copenhagen, Denmark

ABSTRACT

Applying neural network (NN) methods in games can lead to various new and exciting game dynamics not previously possible. However, they also lead to new challenges such as the lack of large, clean datasets, varying player skill levels, and changing gameplay strategies. In this paper, we focus on the adversarial player strategy aspect in the game iNNk, in which players try to communicate secret code words through drawings with the goal of not being deciphered by a NN. Some strategies exploit weaknesses in the NN that consistently trick it into making incorrect classifications, leading to unbalanced gameplay. We present a method that combines transfer learning and ensemble methods to obtain a data-efficient adaptation to these strategies. This combination significantly outperforms the baseline NN across all adversarial player strategies despite only being trained on a limited set of adversarial examples. We expect the methods developed in this paper to be useful for the rapidly growing field of NN-based games, which will require new approaches to deal with unforeseen player creativity.

CCS CONCEPTS

• Computing methodologies → Neural networks; Transfer learning; Ensemble methods; • Applied computing → Computer games.

1 INTRODUCTION

With the recent boom in neural network (NN) applications, game designers have been increasingly exploring a variety of NN approaches in computer games [65]. These include approaches where a NN is directly incorporated into the gameplay experience or as a method for dynamically generating content that would otherwise be created by a human artist [65]. This approach has been utilized in well-known games such as Black and White [36], Creatures [23], and Forza Motosport [17], which adapt game agent behavior in response to player input. In these cases, the NN makes gameplay more personalized and potentially more engaging.

With the recent advances, it is not surprising that more games utilize NNs to provide novel gameplay experiences. While these novel experiences would not be possible without NNs [9, 58], designers are faced with new challenges when applying this technology. For instance, players may learn gameplay strategies that exploit weaknesses in the NN, resulting in unintended interactions with the game. Adapting a NN to account for this can be a challenging task, as effectively utilizing supervised learning with NNs often requires a large, labeled dataset. This is due to the considerable number of free parameters in the NN architecture and the high dimensional input space. How well this dataset portrays the target behavior of the NN has significant influence on the final performance. Such large, clean datasets which accurately represent the task are rare and expensive to obtain.

In this paper, we unpack the technical challenges that developers face when using a NN in games and provide an approach to counter the player strategies that exploit the NN. We utilize the NN-based game, iNNk [56], as a case study to introduce a new NN training procedure that leverages transfer learning and ensemble machine learning (ML) techniques. iNNk [56] is a web-based multiplayer drawing game where two or more people play together against a NN (Figure 1). To win the game, the players need to successfully communicate a secret code word to each other through drawings without the NN deciphering the drawing. This game is particularly well suited for our purpose because the game mechanics encourage players to exploit the NN. Understanding and preparing for such exploitative strategies is essential to balance gameplay.

Our method trains an ensemble consisting of the original NN and a set of NN specialists. Each specialist is trained exclusively on a specific player strategy that we observed overexploit the NN. An example of such a player strategy is what we refer to as distraction.
Using this strategy, the player adds random strokes to the drawing canvas. These strokes work as noise and successfully stump the NN. We use transfer learning to clone the state of the original NN to each specialist in order to kick-start the training process. Our method differs from existing ML literature [6, 14, 42, 60] as each ensemble specialist is built upon the same baseline NN state with its individually distinct dataset to address a particular adversarial player strategy. Further, defense mechanisms for NNs against adversarial human-generated data is a less explored field, and we demonstrate the efficiency of this approach for such settings with sparse datasets.

Our approach allows iNNk’s NN to successfully classify the sketches that utilize these exploitative strategies that the game designer may not have foreseen. A key benefit of this approach is that it requires only a limited number of labeled examples. We test our approach against the set of examples generated for each player strategy. In addition, we evaluate the impact our modifications have on the performance of the original task of classifying regular, non-adversarial sketches. Importantly for games, our method is computationally and temporally efficient, only requiring a limited amount of data. Lastly, it does not significantly compromise the model’s performance on the original task, and the game is more prepared for game-breaking player strategies.

With the increase of NN-based games, how to address unforeseen player creativity and ensure a balanced gameplay is an exciting new research field. In this work, we present the steps in this direction with a NN ensemble learning-based approach that enables learning from relatively sparse data and is generally applicable to a wide range of games. Our experiments show consistent performance improvements on all observed exploitative player strategies, without suffering catastrophic forgetting on the original task. These results highlight the potential of the approach presented in this paper.

2 BACKGROUND

Numerous studies have examined a variety of NN approaches used to introduce experimental features in existing games [3, 13, 54, 57]. We focus on games built with a NN as an integrated part of the original game in this work. Particularly, we focus on how game developers have applied NNs to solve a variety of tasks in their game. For instance, NNs have been used to procedurally generate a variety of different game content [26, 46]. In the game GAR [26], the NN is responsible for generating new weapons for the player to use in battle. This content would otherwise be generated by a human artist, making the game more variable based on the player’s preferences. NNs are also used to vary the behavior in non-player characters (NPCs). For example, in Creatures [23], players raise alien creatures, known as Norns, whose behavior is controlled by a NN. When a new Norn is born, players direct the behaviors of the Norn through positive (petting), negative (striking), and text input to develop the actions and traits that will lead it to survive the longest.

Among NN–based games, there are two types of learning methods: offline and online learning. In offline learning games, they use a fixed NN that is not adaptive during gameplay. A notable example of offline learning is AI Dungeon [58], which is a text-based adventure game. Players are free to enter any text input to direct the story, and the NN can still respond in reasonable ways. In the latest version of the game, Dragon [52], the NN used is commonly known as GPT–3. GPT–3 is trained on a dataset with 570 GB of text using 175B parameters, and cost $12M to develop [8]. Rarely, game-developers have the time and resources to develop such massive NNs from scratch. Collecting such a large and clean dataset is unrealistic for most games. Further, due to the high dimensionality and lack of interpretability of modern NNs, making guarantees of their behavior is not feasible. Even well-performing NNs in offline learning games can easily be exploited by misleading them with adversarial input, as demonstrated in section 4.1.

For online learning games, the NN is continuously trained as the player interacts with it. Therefore, these games can adapt to individual players’ actions in real-time. However, designers need to account for the possible ways players may exploit the training procedure, as this can quickly lead to an unbalanced gameplay experience.

The general lack of guarantees on a NN’s behavior makes NNs hard to deploy in both online and offline settings. A sublime goal for using NNs in games is an automatic adaption to emerging
player strategies without unintended side-effects. An example of such unintended side-effects is introducing strong predictive bias, e.g. by always generating the same output regardless of the input. Another example is devoting substantial attribution to non-robust features, also known as overfitting. Therefore, to achieve automatic adaptation of emerging player strategies, numerous challenges need to be resolved that apply to online and offline learning games. These challenges include 1) sparsely collected and labeled datasets, 2) time-consuming training processes, and 3) the risk of overcompensating with catastrophic loss of performance on the original task solved by the NN.

In this paper, we aim to unpack the technical challenges and offer an approach to better account for and mitigate emerged player exploitations of the NN in the game iNNk. First, we will review related literature on misleading NNs and some related countermeasures. Then, we present our method with a case study on the game iNNk. Finally, we present our results and discuss future possible improvements to the method.

2.1 NNs Are Easily Fooled
Misleading a NNs is not an uncommon endeavor. Adversarial attacks against NNs are an established field of research in the ML community. Research has repeatedly shown how NNs are easily fooled, and has demonstrated how to trigger NNs to make incorrect predictions with high confidence [2, 21, 37, 39]. For instance, the concept of Generative Adversarial Networks (GANs) is founded on its ability to trick a NN to believe an artificially generated image is real [20]. This problem has led to the creation of many interesting artifacts, from inferring missing information [62] to the generation of completely new data [7, 44]. Despite these seemingly innocent applications, the technique can also be used as an attack with severe consequences. An example is the modification of traffic signs such that a NN would misclassify a ‘Stop’ sign for a ‘Speed Limit’ sign [12]. To fool a NN, a common approach is using gradient descent to generate adversarial examples [20, 21, 34]. Others use evolutionary algorithms (EAs) [30, 50] or human-generated data [12].

Recently, a new research area has emerged to counter these adversarial methods and provide alternative defensive solutions [24, 27, 35, 41, 47, 59]. This work has ignited an arms race in the field with continuous developments on both attack and defense mechanisms. Despite the numerous defense methods developed, they focus on computer-generated data to strengthen the robustness of the NN. However, what seems to have gained little attention thus far is developing defense mechanisms against adversarial attacks using human-generated data and make the NN adapt to such datasets.

2.2 Defending Against Adversarial Attacks with Sparse Datasets
One approach to defend against adversarial attacks is the use of ensemble methods. For instance, Abbasi and Gagné [1] used a confusion matrix to choose the training dataset for fitting an ensemble of NN specialists. In general, ensemble methods have been shown to yield more robust models [25]. The baseline approach to ensemble methods involves a set of ensemble members (weak learners), which, when combined, perform better than individual members alone [40]. This approach assumes the ensemble members are accurate and diverse [11, 25]. The ensemble approach can also leverage different types of weak learners and have proven successful with NNs [25].

Alternate approaches to ensemble methods have a considerable variation in recent literature [6, 14, 42, 60]. Some methods, like boosting [14], train each ensemble member sequentially. Each weights the importance of each entry in the training dataset by the previous model’s loss in that example — a higher loss gives a higher priority. Boosting decreases the ensemble’s bias by focusing on ill-performing parts of the dataset. However, this also increases the risk of overfitting. Other ensemble methods, like bagging, have proven less prone to overfitting as they decrease the variance of the predictions. They are also effective with unstable learners due to the smoothing effect of model averaging [11, 64]. To achieve a generalization improvement with bagging, diversity among the ensemble members is essential. This diversity results in a lower error correlation, as the ensemble members are less likely to make the same mistake on the same data [19].

Training robust, supervised, ML models from the ground up usually requires a large and labeled dataset. If one does not have access to such a dataset, one can utilize the ML technique called transfer learning. This technique leverages the finding that using a fitted model on one dataset might serve as a good starting point for training a new model on a similar dataset [61]. Transfer learning is of particular use in domains with sparsely labeled datasets to help prevent overfitting [51, 63]. Further, it allows for utilizing fewer resources to acquire a model for a new task by adjusting the capabilities of an existing model [55]. Transfer learning has proven successful across a wide range of ML problems, from natural language processing (NLP) [5, 29, 38] to image recognition [10, 16, 48].

3 Method
Our approach uses a NN training procedure that leverages the transfer learning and ensembling ML techniques. Our method trains an ensemble consisting of the original NN and a set of NN specialists. Each specialist is trained exclusively on a specific player strategy. We use transfer learning to clone the state of the original NN to each specialist to kick-start the training process. To demonstrate our approach, we utilize the NN-based game, iNNk [56]. In this game, some of the emerged drawing strategies have shown to overexploit the NN to an extent where players would always win. This outcome resulted in an unbalanced gameplay experience, as applying this strategy would consistently stump the NN regardless of the object to be drawn.

We aim to decrease the model’s bias on the adversarial strategies while maintaining the best possible performance on the original dataset. The most straightforward approach would be to simply append these new instances to the existing dataset and continue the training process with this augmented dataset. This approach, however, might not yield the best player adaptation as discussed next.

In case the model’s capacity is already fully used, reducing bias on the adversarial examples would result in reduced performance on
To test our method, we use the game iNNK. Therefore, it is unlikely for such an approach to result in a NN guesses correctly. The NN always plays the role of a Guesser, their guess of the code word based on the Sketcher’s drawing before accurately before the NN. The Guessers are tasked with entering the code word so that the human Guesser can interpret the code word assigned by the game. The goal is to draw the assigned one of the two roles during the game: the Sketcher and need to successfully communicate a secret code word to each other. People play together against a NN. To win the game, the players must recognize and account for the emerged player strategies.

4 CASE STUDY: iNNK
To test our method, we use the game iNNK as a case study. The game is a web-based multiplayer drawing game where two or more people play together against a NN. To win the game, the players need to successfully communicate a secret code word to each other through drawings without being deciphered by the NN. Players are assigned one of the two roles during the game: the Sketcher and the Guesser. The Sketcher is tasked with drawing something based on the code word assigned by the game. The goal is to draw the code word so that the human Guesser can interpret the code word accurately before the NN. The Guessers are tasked with entering their guess of the code word based on the Sketcher’s drawing before the NN guesses correctly. The NN always plays the role of a Guesser, and its goal is to decipher correctly first. Screenshots of the game are shown in Figure 1.

4.1 Adversarial Player Strategies
Investigating winning strategies employed by the players in iNNK [56] (i.e., observing about eight players play iNNK over four games each), we found three different adversarial strategies that were commonly developed (Figure 4). These strategies are not exhaustive, and a more thorough user study will likely reveal many more strategies. Our objective is to demonstrate the technical challenges when dealing with emerging player strategies and provide an approach to alleviating these explorations in a NN-based game for which these three initial strategies were deemed sufficient.

The first strategy includes the Sketcher drawing the code word in a sequential set of images (i.e., as a rebus). The Sketcher was given the code word ‘eyeglasses’, they sketched two separate images, an eye and a pair of drinking glasses, in an attempt to stump the NN. The second strategy includes adding visual noise or other shapes in addition to the drawing of the code word. In this case, the player would crosshatch or draw extra lines to mislead the NN. The players added these extra lines at the beginning of the round. Adding them later would allow the NN to classify the drawing correctly before adding the distraction lines. The third strategy includes drawing the code word more stylistic (i.e., using a dotted line instead of a solid line).

Based on these three observed adversarial player strategies, the game designers manually drew around one hundred sketches for each one (105 for Distraction, 93 for Dotted, and 66 for Rebus). This process was followed to create a clean dataset of each strategy with a sufficient size while minimizing labeling cost. For the Distraction strategy, this was as simple as appending arbitrarily placed lines on some part of the drawing canvas at the beginning of the sketch. It was similarly straightforward to generate data for the Dotted Line strategy by drawing while releasing the mouse periodically. The Rebus Puzzle strategy presented some issues as not all of the classes in our dataset lend themselves to this approach. For example, as seen in Figure 4, the class ‘keyboard’ can be broken into a key and a board of wood which a human player may understand while the NN cannot. However, if the class chosen by the game is ‘dog’, the word cannot be broken down using this strategy at all. This resulted in a sparser dataset with a less diverse set of classes for training a model on this strategy.

4.2 Neural Network Architecture and Training Details
The task of the NN is to classify a sketch correctly given a sequence of stroke data. As mentioned in Section 3, each NN in the ensemble shares the same structure and is similar to the one introduced in a tutorial by Tensorflow [53]. Each of the ensemble NNs is first pretrained on the dataset collected through Google’s ‘Quick, Draw!’ online game [22]. The dataset consists of approximately 40M sketches of 345 distinct classes. Our experiments utilized 10% of it as a validation dataset and 20% as a testing dataset. The sketches in this dataset are human-made digital drawings represented as temporally ordered sequences of strokes that make up the drawing. Each stroke in this sequence is a continuous curve. The curve starts when the...
sketcher’s drawing tool (such as a computer mouse) was activated and ends when the drawing tool was deactivated. This curve is represented as a sample of temporally ordered 2-D point coordinates along the curve. In addition to the recording of the sketch, the dataset also contains metadata for each sketch, the most relevant of which for training our NN was the label of what each sketch depicts. We utilized the ‘simplified’ version of this dataset in our training process, in which the sketches had been preprocessed by centering and scaling them into a $256 \times 256$ space. The strokes had also been simplified with the Ramer-Douglas-Peucker algorithm [45]. We also applied further preprocessing in the NN training process in the form of min-max normalization of the strokes. In addition to the temporally ordered 2-D coordinates, the NN also receives a binary signal, indicating when a stroke is ended. Due to the small datasets containing the user strategies, we used a testing and validation dataset of 10% for these.

Each NN contains convolutional, recurrent, and dense layers. First, a series of three 1-D convolutional layers map the input into $[48, 64, 96]$ feature maps using kernel sizes $[5, 5, 3]$. The first convolutional layer takes three input channels due to the nature of the data. After each layer, dropout [49] and batch normalization [31] are applied to regularize the network. The feature maps are then consumed by the recurrent layers. These layers are responsible for memorizing features across time. We use three bi-directional Long Short-Term Memory (LSTM) layers[28], each with a hidden state of size 256. After each LSTM layer, we also apply a dropout layer. Finally, a dense, fully connected layer computes a class-wise list of classification likelihoods. This layer takes an input vector with a cardinality of 512 due to the bi-directional LSTM layers. Because of the multinomial logistic regression, the output is a vector of logits. These are modified using the Softmax function to convert them into probabilities over the 345 distinct classes. During training, we compute the cross entropy loss between these predictions and the ground-true label.

The hyperparameters of the training process remain unchanged for all ensemble members. Namely, we apply early stopping [43] with a patience of 20 for the training of all ensemble members, including the baseline model. We use a learning rate of $3 \times 10^{-4}$, a batch size of 256, the adam optimizer for gradient descent [33], a dropout [49] rate of 0.3 for all dropout layers, and Xavier initialization [18] for the free parameters of the baseline model. To prevent exploding gradients of the recurrent model, we apply gradient clipping [19] with a maximum $\ell^2$-norm of 1. The experiments were
Figure 3: Making predictions with the ensemble. To make predictions, we employ all ensemble members and query each for a prediction based on the same observation. The set of model predictions needs to be combined into a single prediction to compute the final, class-wise, probabilities.

Figure 4: Examples of the three adversarial player strategy sketches used for retraining: Rebus Puzzle (left), Distraction (center), and Dotted Line (right). For the example of the Rebus Puzzle strategy, the code word is 'Keyboard'. The Sketcher decided to divide this into two separate sketches, one of a key and one of a wooden board. An example of the Distraction strategy is a drawing for the code word 'Piano'. Using this strategy, the Sketcher added straight lines to stump the NN. The superimposed sketch of a piano is otherwise unchanged. Finally, for the Dotted Line strategy, the example shown is for the code word 'Moustache'. Here, the Sketcher only modifies the linestyle.

executed on a machine with an Intel Core i7-5820K CPU, with 64GB of RAM and an NVIDIA GP102 TITAN X GPU.

5 RESULTS
To test the performance of our proposed ensemble model with transfer learning, we evaluate each ensemble member and the ensemble
Table 1: Performance on the various datasets for each model on a set of performance metrics.

| Neural Network | Top-1 Accuracy | Top-5 Accuracy | Cross entropy |
|----------------|----------------|----------------|---------------|
|                | Quick, Draw!  | Distraction    | Dotted        | Rebus         | Quick, Draw!  | Distraction    | Dotted        | Rebus         |
| Baseline Model | 83.52%        | 0.00%          | 0.00%         | 0.00%         | 96.20%        | 20.00%         | 0.00%         | 0.00%         | 0.62          | 5.22          | 7.52          | 6.97          |
| Distraction Model | 80.51%        | 70.00%         | 0.00%         | 16.67%        | 95.35%        | 86.67%         | 23.92%        | 16.67%        | 0.74          | 1.82          | 5.39          | 11.46         |
| Dotted Model   | 77.80%        | 10.00%         | 70.37%        | 0.00%         | 94.33%        | 30.00%         | 88.89%        | 16.67%        | 0.84          | 6.00          | 1.69          | 7.50          |
| Rebus Model    | 79.32%        | 13.33%         | 7.41%         | 11.11%        | 95.05%        | 50.00%         | 14.81%        | 44.44%        | 0.77          | 4.45          | 5.76          | 4.28          |
| Ensemble Model | 82.59%        | 50.00%         | 0.00%         | 16.67%        | 95.98%        | 70.00%         | 44.29%        | 33.33%        | 0.65          | 2.51          | 4.42          | 4.98          |

Table 1 shows the performance of different models on various datasets using Top-1 and Top-5 Accuracy, and Cross entropy loss. The Baseline Model shows the highest Top-1 Accuracy on the Quick, Draw! dataset, while the Dotted Model shows the lowest Top-1 Accuracy on the Dotted dataset. The ensemble model performs consistently across all presented datasets and is the ensemble model that is used in the final model. The discussion section further elaborates on the implications of these results and their significance for the game iNNk.

6 DISCUSSION

The Dotted model’s loss on the original dataset could indicate that the parameters of the model need substantial adjustments for improving performance on the Dotted dataset. One explanation could be the low-level feature maps in the first convolutional layers, which typically work as edge- and curve-detectors. These might transfer poorly from the ‘Quick, Draw!’ to the Dotted dataset, as an edge in the Dotted dataset no longer indicates separation but is still often part of a continuous line.

The motivation behind reporting the Top-5 Accuracy as a performance metric is due to the nature of the game iNNk. Here, an incorrect guess gets blacklisted, and neither players nor NN is allowed to make this prediction again. Technically, this happens by setting the NNs predicted logits to $-\infty$ for these blacklisted classes. For instance, when the class with the highest probability is blacklisted, the NN would predict its perceived second most likely class. This is to prevent the repeating of incorrect guesses. To prevent exhausting the list of whitelisted classes, the NN is only queried for a prediction every 2.5 seconds. For this reason, having the correct class among the Top-5 predictions is important for the gameplay, as the four incorrect guesses would be blacklisted within 10 seconds at worst. One could argue that not having a perfect Top-1 accuracy score even improves the gameplay experience, as an oracle-like NN might discourage players from playing the game. From the results in Table 1, it is clear that the proposed ensemble model gains significant improvements for all datasets on this Top-5 Accuracy.
An inherent limitation in our approach on how to prepare for adversarial player strategies is that to identify one, it must have a recognizable pattern that the NN can be trained on. Therefore, our approach will not work on strategies developed when players communicate outside of the game to achieve the correct answer. This information cannot be made available to the NN, meaning there is no way to compensate for this type of strategy. Other strategies may involve inside knowledge within the player group, which would also be challenging and most likely not particularly useful to train the NN on.

Additionally, the proposed process of identifying and collecting examples of player strategies is limited in terms of scalability. For instance, our work identified player strategies through human observations and analyses. However, identifying strategies on a much larger scale or using a more complex game with this process may not be as easily executed. To address these limitations, we plan on investigating the automation of these processes (see section 7).

Finally, each time a NN is added as a member of the ensemble, it prolongs the time it takes to make predictions. Making predictions with the ensemble can be run in parallel on separate GPUs. In practice, however, there is a limit to the scalability of the method and hence the number of addressable, distinct player strategies.

7 FUTURE WORK
Our planned next steps involve identifying further player strategies and extending iNNk’s NN to recognize them. Given that there could exist a large number of player strategies that dynamically emerge over the game’s life-cycle, it would be beneficial to automate the process of discovering new player strategies and the retraining of the NN. This would lead to a more dynamic experience of the game evolving over time and also reduce the amount of work in achieving this. Therefore, we plan to explore methods for automating the player strategy discovery process using clustering ML methods, for example. Given the difficulties with generating player strategy data by hand, we also plan to explore automated approaches for augmenting training data to produce new examples, such as via programmatic generation of player strategy drawings. Finally, we plan to perform further fine-tuning on our proposed ensemble via experimentation with different model prediction aggregation (voting) methods.

An additional point of investigation could be to assess the impact of batch normalization. Recent studies suggest the use of batch normalization can increase a NN’s vulnerability to adversarial attacks [4, 15]. While more analysis is needed on this subject, future work could explore whether the baseline model applied in this research could benefit from abandoning the use of batch normalization in the model architecture. While such modification potentially increases model robustness, it will also remove the benefits from applying batch normalization – faster model convergence being one of them [32]. This could lead to an unacceptable increased cost in model training, as the current model already takes >3 weeks to fit.

8 CONCLUSION
In this paper, we use the NN-based game iNNk as a case study to demonstrate an approach to adapt a NN for adversarial player strategies in order to maintain balanced gameplay. To counter those strategies, we propose an approach that combines transfer learning and ensemble methods to strengthen our classifier based on sparse datasets. We evaluate the performance of our approach using three different performance measures on the testing datasets. As a result, we found that each model dedicated to a strategy lost performance on the baseline dataset, while the baseline model was ineffective on all player strategies. We found our ensemble approach to provide the best, most consistent performance across all datasets, with <1% loss on Top-1 Accuracy on the original dataset compared to the baseline model. Further, we found our use of transfer learning crucial for the ensemble setup to alleviate overfitting. Since our method is effective and efficient to set up, despite our ensemble of models being trained on a very limited amount of data, it can be used by others to create more competent NNS in their own domains. Lastly, we present potential ways to develop this method further that include automating the process and building more sophisticated ensembles, which would result in more accessible and more interesting use cases for NNS in games.

ACKNOWLEDGMENTS
This project was partially supported by the Innovation Fund Denmark (Grant ID 8053-00073B), a DFF-Research Project1 grant (9131-0042B), and U.S. National Science Foundation under grants IIS-1917855 and IIS-1816470.

REFERENCES
[1] Mahdieh Abbasi and Christian Gagné. 2017. Robustness to adversarial examples through an ensemble of specialists. arXiv preprint arXiv:1702.06856 (2017).
[2] M. A. Alcorn, Q. Li, Z. Gong, C. Wang, L. Mai, W. Ku, and A. Nguyen. 2019. Strike (With) a Pose: Neural Networks Are Easily Fooled by Strange Foses of Familiar Objects. In 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR). 4840–4849. https://doi.org/10.1109/CVPR.2019.00498
[3] Maren AWSJRISX, Frederik Schubert, and Bodo Rosenhahn. 2020. TOAD-CAN: Coherent Style Level Generation from a Single Example. Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 16, 1 (Oct 2020), 10–16. https://ojs.aaai.org/index.php/AIIDE/article/view/7401
[4] Philipp Benz, Chaoming Zhang, and In So Kweon. 2020. Batch Normalization Increases Adversarial Vulnerability: Disentangling Usefulness and Robustness of Model Features. arXiv:2010.03315 [cs.LG]
[5] John Blitzer, Mark Dredze, and Fernando Pereira. 2007. Biographies,ollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In Proceedings of the 45th annual meeting of the association of computational linguistics. 440–447.
[6] Leo Breiman. 1996. Bagging predictors. Machine learning 24, 2 (1996), 123–140.
[7] Andrew Brock, Jeff Donahue, and Karen Simonyan. 2019. Large Scale GAN Training for High Fidelity Natural Image Synthesis. In 7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019. OpenReview.net. https://openreview.net/forum?id=B1xsqj09Fm
[8] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alex Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. arXiv:2005.14165 [cs.CL]
[9] Gabriele Cimolino, Sam Lee, Quentin Petruoiva, and TC Nicholas Graham. 2019. Oui, Chef!!: Supervised Learning for Novel Gameplay with Believable AI. In Extended Abstracts of the Annual Symposium on Computer-Human Interaction in Play Companion Extended Abstracts. 241–246.
[10] Wang Dawei, Deng Limiao, Ni Jiangong, Gao Juyue, Zhu Hongfei, and Han Zhongzhu. 2019. Recognition pest by image-based transfer learning. Journal of the Science of Food and Agriculture 99, 10 (2019), 4524–4531. https://doi.org/10.1002/jsfa.9689 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/jsfa.9689
[11] Thomas G Dietterich. 2000. Ensemble methods in machine learning. In International workshop on multiple classifier systems. Springer, 1–15.
[59] Katy Warr. 2019. Strengthening Deep Neural Networks: Making AI Less Susceptible to Adversarial Trickery (1 ed.). O’Reilly Media, Inc. 246 pages.

[60] David H Wolpert. 1992. Stacked generalization. Neural networks 5, 2 (1992), 241–259.

[61] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. 2014. How transferable are features in deep neural networks?. In Advances in Neural Information Processing Systems. Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (Eds.), Vol. 27. Curran Associates, Inc., 3320–3328. https://proceedings.neurips.cc/paper/2014/file/375c71349b295fbc2dcdca920620a0e-Paper.pdf

[62] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. 2018. Generative Image Inpainting with Contextual Attention. In 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. 5505–5514. https://doi.org/10.1109/CVPR.2018.00577

[63] Wei Zhao. 2017. Research on the deep learning of the small sample data based on transfer learning. AIP Conference Proceedings 1864, 1 (2017), 020018. https://doi.org/10.1063/1.4992835 arXiv:https://aip.scitation.org/doi/pdf/10.1063/1.4992835

[64] Zhi-Hua Zhou. 2012. Ensemble methods: foundations and algorithms. CRC press.

[65] Jichen Zhu, Jennifer Villareale, Nithesh Jarvaji, Sebastian Risi, Mathias Löwe, Rush Weigelt, and Casper Harteveld. 2021. Player-AI Interaction: What Neural Network Games Reveal About AI as Play. In Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems. 1–17.