Match-3 Levels Similarity Measurement using AutoEncoder

Salil Deshpande and Simon Cheng Liu
salil@cmu.edu, liucheng@levelup.ai

***Abstract* – Companies developing Match-3 Games have always striven to develop new, interesting and playable levels for their games. Having developed a new level either manually or algorithmically, it is important for these companies to understand if the new level generated is similar or different to previously generated levels. An algorithm for measuring the similarity between any given levels of a Match-3 game could enable the companies to develop and publish even more diverse and interesting levels which could lead to a better gaming experience for the players. Our research aimed to develop an unsupervised learning approach for generating a similarity score between an arbitrary pair of levels from a dataset of levels. The progress of research in AI has enabled the development of deep learning networks which can perform dimensionality reduction on unlabeled data through unsupervised learning. In this paper, we present a Simple AutoEncoder network that has been developed for performing unsupervised dimensionality reduction of the Match-3 Game level data into a compact latent space vector for cosine distance calculation. We have also researched and developed an alternate rule-based model for comparing the the 2 models on various aspects.**

Index Terms – AutoEncoder, Match-3 Level Similarity, Neural Networks, Unsupervised Learning.

INTRODUCTION

Match-3 games have always been at the forefront of casual gaming, irrespective of the platform. The tremendous growth in availability and usage of mobile devices across the world lead to conditions which were conducive to the growth of casual games, and in particular, Match-3 games. Because of this, Match-3 games have always topped the worldwide lists of top grossing and top downloaded games, especially for mobile platforms. For instance, Figure 1 shows the 10 top grossing iPhone games in USA. Of these, 3 are Match-3 games: Candy Crush Saga, Homescapes and Matchington Mansion.

The greatest strength of Match-3 games is that the underlying rules remain constant, which make them playable universally without any kind of barriers for worldwide gamers. But this strength can also turn into a huge weakness – in the sense that as the games have the same basic core mechanism, they can get boring quite quickly, and unless the developers introduce new and interesting features, they can easily lose out on their user-base. At the same time, they cannot divulge from the very core of the game, which is the reason such games are successful.

Also, Match-3 games are supposed to have a huge number of levels, which form the core content of the game. Moreover, the game should have an accurate mixture of diverse levels in order to have a high user engagement.

Considering this, the developers have a very limited scope to make changes without straying much further from the original concept of Match-3 games, but at the same time, they face a challenging task of making their games interesting. Thus, for ensuring the success of their games, the development of interesting levels is necessary. A good gaming experience can be ensured through a good mix of similar and diverse levels. It is thus quite important for the developers to understand whether the new levels generated are similar to any previously generated levels, or whether they are completely different. This is especially important when the levels are being generated algorithmically, using technologies like Procedural Content Generation.

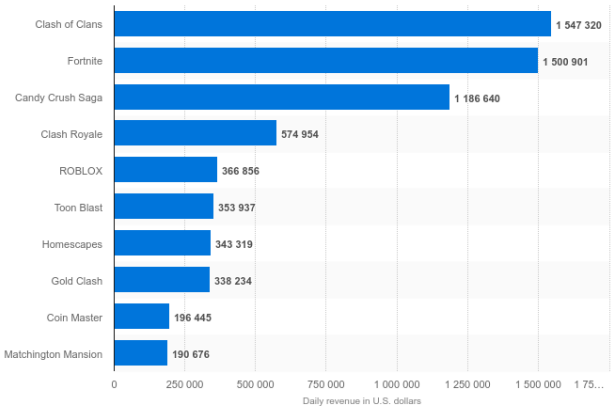

FIGURE 1
TOP GROSSING IPHONE GAMES IN USA, AS OF MAY 2019, RANKED BY DAILY REVENUE IN USD

The lack of availability of labels for similarity between pairs of levels and the need for an "automatic" solution. In addition to that, the lack of manually labeled data meant that it was not possible to use supervised learning algorithms, and we had to develop either an unsupervised learning solution or a rule-based mathematical solution. This directed us towards developing 2 types of solution: a rule-based algorithm and an unsupervised learning algorithm. Developing an unsupervised learning solution was desirable as the game board data consists of a large number of attributes and

dimensionality reduction was a requirement for generating any plausible solutions.

This paper presents a deep, fully-connected network modeled as an AutoEncoder and applies it for the purpose of Match-3 level similarity score generation. This model can effectively reduce the high-dimensional input data into a condensed low-dimensional vector for an easier and more accurate comparison of level data for generation of a similarity score. We have also discussed a rule-based algorithm as a solution for the same problem in order to compare and contrast the feasible solutions.

In the subsequent sections we have discussed in detail about the existing research in this domain related to this and similar such problems, our attempt at collection of human data on similarity of levels, the autoencoder model and architecture, generation of the results using this model, comparison of this model with a rule-based approach and the advantages and disadvantages of both these methodologies.

Literature Survey

As mentioned earlier, there is very little available research on levels similarity in Match-3 Games, and so the development of the proposed solution was done by dividing the problem into smaller ones. On their own, some of these smaller problems had certain existing research, and so we developed the solution on the basis of studies and evaluation of already developed solutions for similar problems in the field of machine learning and game development. Although this problem has not been addressed before, certain components of the problem do have existing research.

Having identified the need for an unsupervised learning dimensionality reduction algorithm, we decided that an AutoEncoder would be the most suitable solution for the problem. The concept of AutoEncoders was first introduced in [1] in 1986. The paper discussed about learning internal representation of data using a feed-forward network and back propagation. Research on AutoEncoder models was accelerated due to the success of back propagation and the continued research resulted in development of various AutoEncoder models, including Simple [1], Convolutional [3], Denoising [14], Transforming [4], Contractive [5], Variational [7] and Adversarial [13].

To develop a Deep Neural Network as the solution, it is important to model the data in such a way that the network is able to accept, process and output results based on that data. Level representation in a way that a Deep Neural Network will be able to handle it has been discussed in [8], and a similar approach has been utilised by us.

Manual Data Collection

Human player's perception of similarity of levels in a Match-3 games plays an important role in understanding their expectations of an interesting level within the game. In order to fully understand the nature of this problem and the nature of results that would be generated, a local website was developed wherein the employees at our workplace could view a random pair and select a score which they believed to be the most relevant. Later on, the users could also see the score assigned by the algorithms for a particular pair.

A screenshot of the website is shown in Figure 2. A random sequence of pairs of levels was generated and was displayed on the website. The users could browse through the pairs, and for each pair, a screenshot of both the levels was displayed. 4 choices were presented below the images, numbered from 1 to 4, of which the users had to select the most relevant score. The scores and their relevance have been listed below:

- 1 – Very dissimilar
- 2 – Somewhat dissimilar
- 3 – Somewhat similar
- 4 – Very similar

Underneath the options, the scores generated by different algorithms including the ones mentioned in this paper as well as the ones developed by other colleagues have been displayed, to help analyze the results.

FIGURE 2
A Screenshot of the Website for Level Similarity Rating

During data collection through this website, the results were collected for 235 pairs of levels from 7 different users and the statistics of scores for each pair of levels was generated to better understand the data. According to these statistics, the users seem to highly agree on the similarity of certain pairs of levels, whereas some pairs of levels show high level of variance and disagreement between human assigned labels. According to the distribution of scores as shown in Figure 3, it is evident that most of the pairs have been rated as "Very dissimilar", and it is logical as the game is supposed to have diverse levels for an engaging gameplay. In line with this logic, a very low number of pairs of levels have been rated as "Very similar".

A careful and detailed analysis of the collected ratings helped us to accurately define the problem and our approach for solving this problem. The ratings seem to be heavily influenced by the visually appealing elements of the game level. The topmost layers or the most "vibrantly" colored pieces affected the decisions of users while assigning the ratings. For instance, pairs of levels wherein

both the levels have a high amount of pink jelly are rated to be highly similar by most of the users, irrespective of the location or shape dissimilarities in the 2 levels.

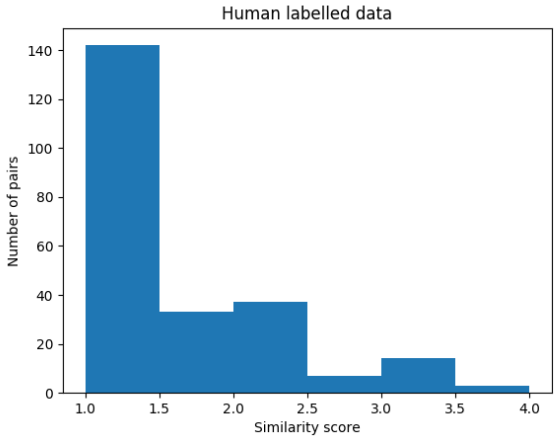

FIGURE 3
SCORE DISTRIBUTION OF HUMAN-ASSIGNED LABELS

THE AUTOENCODER METHODOLOGY

We decided to use an AutoEncoder Network because many of its qualities fulfilled the requirements of our problem. The network itself is trained like an unsupervised network [1], meaning the only requirement is the data itself, and it does not require any additional labels. Furthermore, the latent space representation developed at the **bottle-neck** between the encoder and decoder acts as a compact representation of the original input. Thus, this network can be used as a dimensionality reduction mechanism [9], which is exactly what we needed for the game level data.

Now we will discuss this methodology further by breaking it down to its individual steps and discussing each step in detail. Figure 4 shows an overview of the entire process of training the model and using it for dimensionality reduction. As shown, the solution is mainly divided into 2 phases: Training and Deployment. The Training phase consists of Data Pre-processing and training the AutoEncoder using that data. The Deployment phase is concerned with using only the Encoder part of the network to generate the dense vector, which is further used for the purpose of distance calculation.

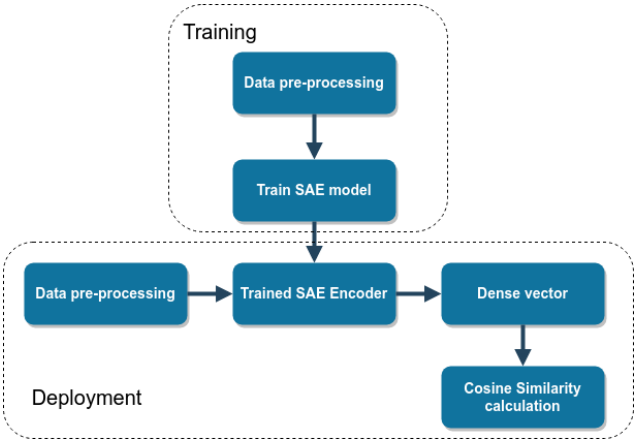

FIGURE 4
FLOWCHART OF THE AUTOENCODER SOLUTION

I. Game Levels Dataset Pre-processing

The original level data was in JSON format, as shown in Figure 5, and contained a large amount of hierarchical and categorical attributes. In its original form, it was not suitable for developing a Machine Learning (ML) model. Because of this, it was necessary to pre-process and convert the original dataset into a format which would be much more suitable for an ML model to train on.

Similar to the technique mentioned in [8], the vectorization of the data was done by reformatting the level data into a 3D binary tensor. The game board was represented as a 12×12 grid with 35 binary feature planes. Each of these 35 channels represented a unique component of the game board. For a given channel if the tile of that particular component was present on the level board, it would be marked as a 1 in the new data representation or else it would be a 0. On 0-padding and stacking these planes together, the feature planes form a 35 channel 2D input to the network, similar to Figure 6. As the data was completely numeric now, it was much easier to process for the model.

A complete list of all the 35 layers is showcased in Table I.

```
data = {
    'level_index': 69,
    'move_count': 30,
    'board_info': {
        (0, 0): {
            'bg_number': 41,
            'fall_point': (0, -1),
            'next': (0, 1),
            'prev': (0, -1)
        },
        (0, 1): {
            'bg_number': 41,
            'next': (0, 1),
            'prev': (0, -1)
        },
        ...
    },
    'trans_info': {
        (0, 0): {
            41: 50,
            54: 40
        }
    },
    'apple_box_info': [
        (0, 0),
        (0, 3),
        (8, 0),
        (8, 3),
        (4, 6)
    ]
}
```

FIGURE 5
ORIGINAL LEVEL DATA JSON STRUCTURE

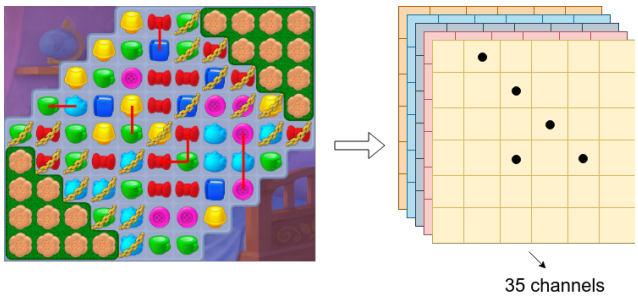

FIGURE 6
LEVEL DATA SEPARATION INTO CHANNELS

TABLE I
CHANNEL BREAKDOWN

Channel No.	Channel Name	Channel No.	Channel Name
1	Cell present	19	Box Level 1
2	Bow	20	Box Level 2
3	Cup	21	Box Level 3
4	Pot	22	Ice Level 1
5	Book	23	Ice Level 2
6	Lamp	24	Chain Level 1
7	Button	25	Chain Level 2
8	Special Ball	26	Chain Level 3
9	Bomb	27	Jelly Level 1
10	Rocket (Up-Down)	28	Jelly Level 2
11	Rocket (Left-Right)	29	Jelly Level 3
12	Plane	30	Jelly Level 4
13	Background Oil	31	Jelly Level 5
14	Background Rug	32	Cookie Level 1
15	Cherry	33	Cookie Level 2
16	Donut	34	Cookie Level 3
17	Foam	35	Wall
18	Apple Box		

II. AutoEncoder Network

The network architecture described in this paper is the **Simple AutoEncoder (SAE)**. Figure 7 shows the basic functioning and architecture of an SAE, and its core parts have been highlighted and labeled for clarity.

The SAE is a feed-forward network and the connections between the layers are fully-connected i.e. every neuron in the previous layer is connected to every neuron in the next layer. The dimensions of input and output layers are the same, which is a 1D vector of the size 12×12×35, which is the size of the flattened level data vector. It is important to note that the input and output dimensions need to exactly equal, as we will need to compare the input and the output produced by the network in order to train it.

Conforming to the core concept of AutoEncoders, the network itself was divided into 2 distinct parts: the Encoder and the Decoder. The size of every subsequent layer in the Encoder was smaller than that of the previous layer i.e. the layers get narrower as we proceed through the Encoder. The exact opposite holds true in the case of the Decoder, wherein the size of every subsequent layer was larger than that of the previous layer.

The training of this model was done by passing some input data (a game level) to it, and then trying to replicate the input as closely as possible at the output. While the input data was being passed through the network, it went through smaller and smaller sized layers until it reached the bottleneck. After that, the latent space vector was passed through increasingly larger layers until it was output in the same dimensions as that of the input. Thus, the network was made to extract the important features from original input and learn its dimension reduced representation. The network updated its weight such that it learnt to identify the important features of the input and compress the input to give out the dense representation at the end of the Encoder part of the network.

III. Network Architecture Description

As we were using an SAE, the architecture consisted purely of Fully-Connected layers, but each of varying dimensions. The exact architecture of the SAE used in this paper is described in Table II.

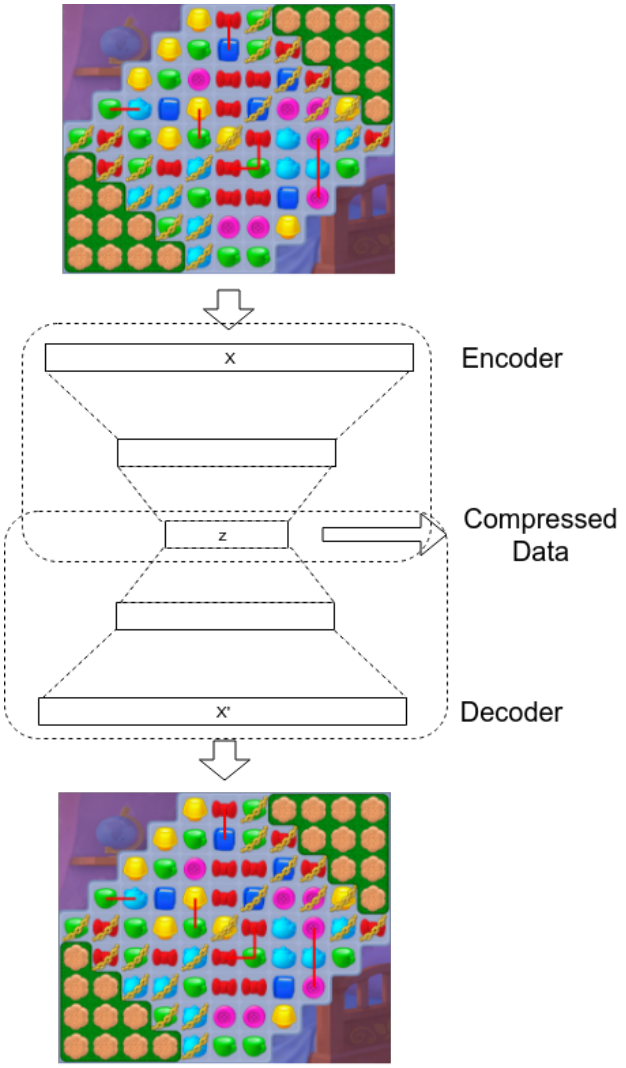

FIGURE 7
SAE BASIC ARCHITECTURE

As shown in the Figure 7 and as mentioned earlier, the network consists of 2 distinct parts: the Encoder and the Decoder, both placed sequentially. The 12×12×35 input tensor was flattened into a 1D vector of size 5040 and fed to the Encoder at its first Fully-Connected layer, and it forwarded this input to the subsequent hidden layers. At the end of the Encoder, a **Latent Space Vector** was obtained, which was essentially a low-dimensional representation of the original input. This Latent Space Vector was then fed to the Decoder, which passed it through a series of Fully-Connected Layers to "up-sample" the data. Finally, an output was produced by the

Decoder and it was compared with the original input. The aim of training this network was to update the weights of all the layers such that the network could reproduce the input with as much accuracy as possible.

TABLE II
MATCH-3 AUTOENCODER FULLY-CONNECTED LAYERS

Layer No.	Part of	Layer size	Activation function
1	Encoder	5040	ReLU
2		2500	ReLU
3		1000	ReLU
4		500	ReLU
5		10	Linear
6	Decoder	500	ReLU
7		1000	ReLU
8		2500	ReLU
9		5040	Linear

The choice of library for development of this network was PyTorch [10]. In an initial attempt, we had made use of Convolutional layers in the AutoEncoder, but it failed to produce any valid results, due to the possibility that we had a very limited amount of training data. Thus, we decided to implement the network using fully-connected layers. Also, due to a significant number of layers, we decided to use the ReLU activation function, as it offers good performance as well as accurate results as proved in [11] and [12] as compared to the Sigmoid activation function. Furthermore, ReLU solves the problem of vanishing/exploding gradients, thus making it a favorable choice for deep networks. The activation function at the end of Encoder and Decoder was selected to be the Linear function itself as the expected output was a regression output.

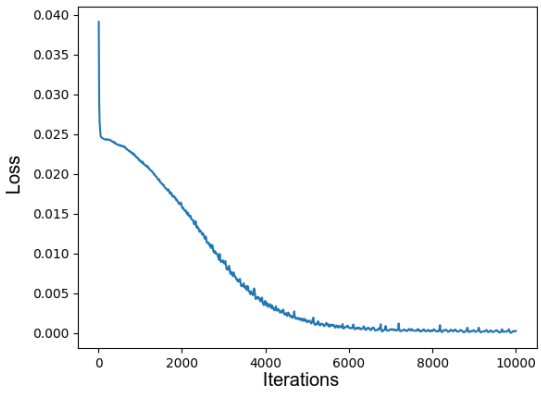

FIGURE 8
SAE TRAINING LOSS

In practice, once the network is fully trained, only the Encoder part is required. The Encoder of the trained SAE acts as a standalone dimensionality reduction network, as evident from previous explanation and Figure 7.

An interesting extended use of this model is to re-generate levels, or rather generate new levels which are a combination of the components of the levels the model has been trained on. Given the right input to the Decoder, it can produce the 35 channel binary feature planes representation of a valid level. Some changes in the architecture can in fact convert it into a Generative AutoEncoder, which will be capable of generating the new levels as demonstrated in [13].

IV. Similarity Score Generation

Once the SAE was trained to produce accurate replicas of input, the Encoder was the part which was required. On its own, the Encoder part of the network now acted as a dimensionality reduction network.

On passing the original level data to the Encoder, we got back the condensed representation of the original level. The input given was of the size 12×12×35 and the output of the Encoder was a 10 attributes vector. This low dimensional vector was useful for comparison and further calculations. In this paper, we have made use of the cosine distance (1) for the calculation of similarity between a pair of vectors. Formula (1) can be rewritten as (2) for a better understanding of the underlying operations.

$$similarity(A,B)=\cos(\theta)=\frac{A.B}{\|A\|\|B\|} \tag{1}$$

$$similarity(A,B)=\frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2}\times\sqrt{\sum_{i=1}^{n}(B_i)^2}} \tag{2}$$

The advantage of cosine distance for measuring similarity is that it calculates the angle between the vectors, and not their absolute distance, thus providing a better idea of the actual variations in the attributes. Euclidean distance would not have been useful in this case as it would have calculated distance between 2 overlapping vectors as well

V. Results and Analysis

After training the network, it was able to reproduce the original level data accurately as shown in Figure 9.

FIGURE 9
ORIGINAL AND REGENERATED GAME LEVELS COMPARISON

As for the accuracy of the generated similarity scores, since there was no baseline or absolute reference, we compared the scores to the limited human generated ratings that we had. Of all the assigned ratings, only

43.64% of the ratings generated by this algorithm lied within a range of ±1 score of the mean of human assigned ratings.

The cosine similarity scores generated by this algorithm by default lied in the range (-1, 1). To make these scores compatible with the scores displayed on the website, we scaled the scores to a range of (1, 4) using (3). In (3), the max and min refer to the boundaries of the original scale of data, and the max' and min' refer to the boundaries of the new scale of the data.

$$x' = \frac{(x - min) \times (max' - min')}{max - min} + min' \quad (3)$$

The distribution of scores generated by this algorithm as displayed in Figure 10 and Figure 11 is distinctively in the shape of the "Bell-curve" or the "Normal Distribution". This finding was much different to the distribution of human ratings as shown in Figure 3, which was heavily skewed towards 1 score. Ideally, the normal distribution of the scores can seem logical, as we can expect that most of the levels lie along the border of being termed similar or dissimilar due to the reuse of limited number of components in each level, and much fewer levels are highly similar or highly dissimilar.

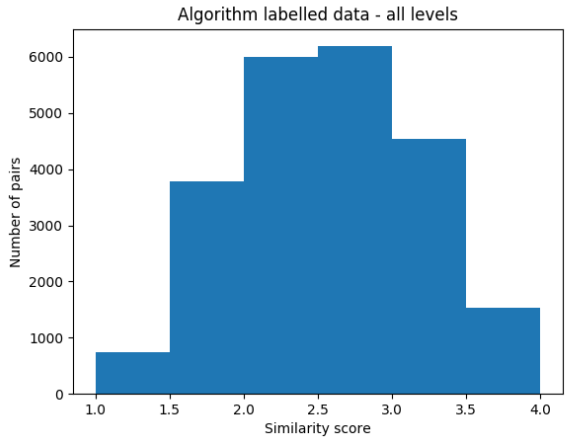

FIGURE 10
SCORE DISTRIBUTION OF ALL LEVELS FOR AUTOENCODER-BASED ALGORITHM ASSIGNED LABELS

Figure 12 discusses the difference of the scores generated by the algorithm and the human generated scores against the 0 line. Ideally, the expected graph would be a straight line overlapping the 0 line. In this scenario, the difference in the scores can be seen to be positive for maximum number of pairs. The points where the plot is close to the 0 line is where the both the scores by the algorithm and humans highly agree, and similarly the points where the plot is the furthest from the 0 line expresses high disagreement between the 2 scores.

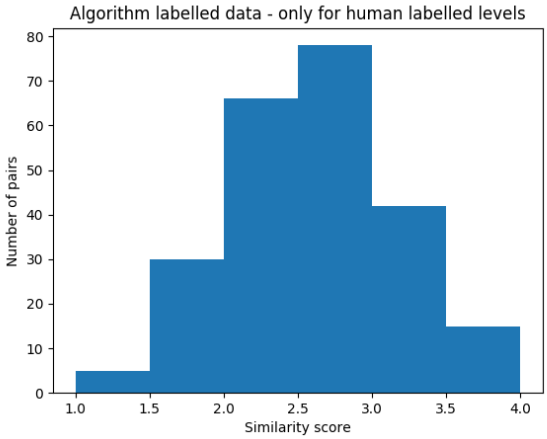

FIGURE 11
SCORE DISTRIBUTION OF HUMAN-RATED LEVELS FOR AUTOENCODER-BASED ALGORITHM ASSIGNED LABELS

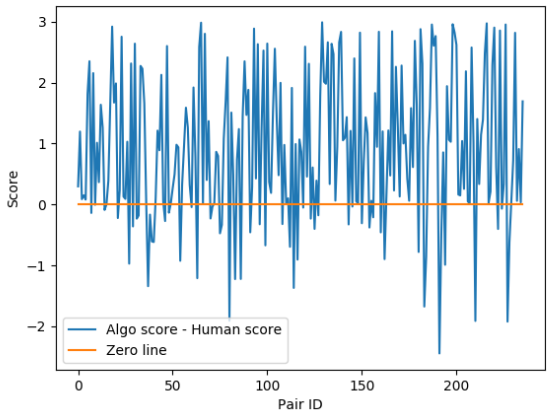

FIGURE 12
PLOT OF SAE RATINGS VS. HUMAN RATINGS

RULE-BASED ALGORITHM

Although we had successfully implemented the SAE for this problem, we were also interested in developing a different technique for analysis of this problem. Particularly, we were interested in a different vectorization technique and so we developed the Rule-based algorithm as well. An additional advantage of developing this technique was that we could compare various aspects of the 2 algorithms to decide which one was better.

The Rule-based solution developed was based on aggregation of data to formulate vectors of each level. In this technique, we aggregated the quantities of various pieces on the game board including base pieces, covers, special powerups and so on.

Figure 13 shows an overview of the entire process of training the model and using it for dimensionality reduction. As shown, the solution is mainly divided into 2 phases: Vectorization and Score Calculation. The Vectorization phase consists of Data Pre-processing and formulation of level vectors using the existing data. The Score Calculation phase utilizes the vectors generated in the last phase and calculates the cosine similarity between all the pairs of levels.

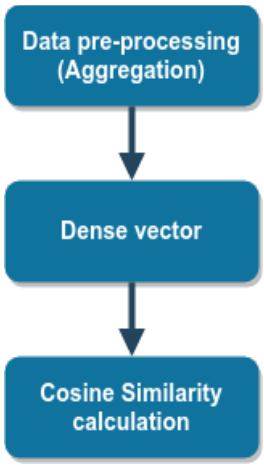

FIGURE 13
FLOWCHART OF THE RULE-BASED SOLUTION

I. Game Levels Dataset Pre-processing

In order to explore the effectiveness of either techniques, a different method of vectorization of levels was developed. Unique pieces in the game were identified, and a vector was developed by aggregating the number of pieces of each type for a particular level. In all, 13 unique pieces were recorded, and so every level was converted into a vector containing 13 attributes. Each of these attributes have been described in Table III.

TABLE III
ATTRIBUTES FOR AGGREGATED VECTOR

Attribute No.	**Attribute Name**
1	Normal pieces
2	Powerups
3	Cherries
4	Donuts
5	Foam
6	Cookies
7	Boxes
8	Ice
9	Chain
10	Jelly
11	Background
12	Walls
13	Apple Boxes

The important part to note about this method was that as it was aggregating the number of elements on the game board, it did not consider the location of the elements i.e. the location data of the pieces such as the absolute position of pieces on board and the relative position of pieces with respect to one another, and the data related to the formation of the level were lost in the vectorization process.

II. Similarity Score Generation

As the vectorization of original dataset results in each level having a vector of 13 attributes or components, the cosine similarity formula could be directly applied for these vectors. As done earlier, cosine similarity score was generated for all the pairs of levels using (2) and then normalized using (3).

III. Results and Analysis

The distribution plot of similarity scores generated has been shown in Figure 14 and Figure 15. A comparison of the scores distributions in Figure 14 Figure 15 and Figure 3 shows that the algorithm, similar to humans assigns scores much closer to 1 most of the times. On comparing the generated scores against the pairs of levels on the local website, it is evident that the algorithm is heavily reliant on the visual aspect of the levels, and is thus providing fair results. Without any additional weights or improvements in the generated results, **68.3%** of the scores generated were within a range of ±1 score as compared to the human assigned labels.

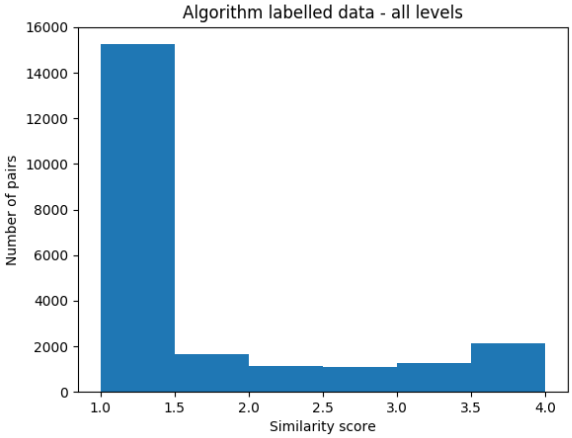

FIGURE 14
SCORE DISTRIBUTION OF ALL LEVELS FOR RULE-BASED ALGORITHM ASSIGNED LABELS

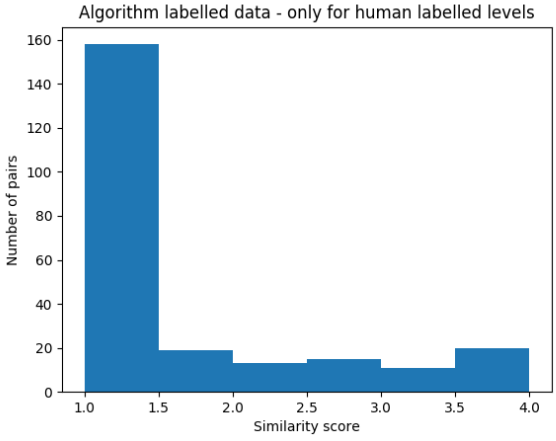

FIGURE 15
SCORE DISTRIBUTION OF HUMAN-RATED LEVELS FOR RULE-BASED ALGORITHM ASSIGNED LABELS

IMPROVING THE ACCURACY

The results generated by both the algorithms were compared with the results of the manual data collection. The initial results generated were not highly accurate, and

thus there was a need to improve the accuracy of both the algorithms. As both the algorithms consisted of attributes which gave unique properties to the vectors, there was a possibility that some attributes might have a higher correlation to the factors that determined similarity of the levels, whereas some other attributes might have a lower correlation. This property could be confirmed by assigning weights to each of the vector attributes.

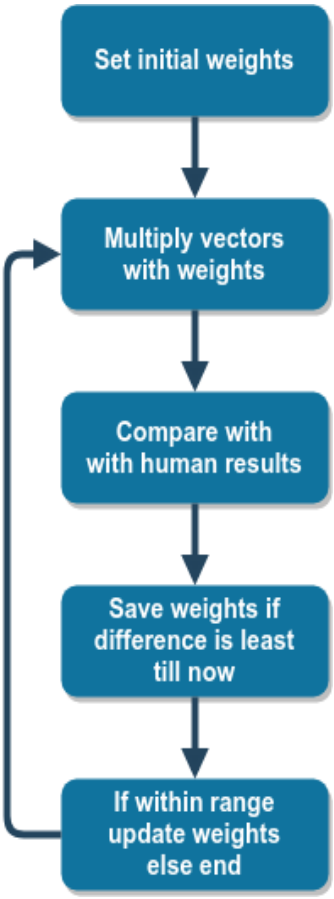

FIGURE 16
FLOWCHART OF THE WEIGHTS-SEARCH ALGORITHM

In order to maximize the accuracy of the algorithms, we needed to find the best weights for the vector. This was done by using a greedy search methodology. To start with, all the attributes were assigned a weight of 1. The similarity scores generated after assigning the weights were then compared with the mean of manually assigned scores for each pair. If the new scores generated were closer to the manual scores, the weights were saved. Moving on, the weight for a particular attribute was increased or decreased by a set amount called "step" and the entire process was repeated until the entire range of values for a particular attribute was exhausted. This entire algorithm was repeated for the weights of all the attributes, thus helping us derive the best weights in a greedy way.

The advantage of this algorithm was that it could be applied to both the score generation algorithms by simply setting the size of the vectors it needs to work on. Also, being greedy search, it could find the optimized weights quite quickly as compared to other algorithms.

The disadvantage of this algorithm was that being a greedy approach, it did not ensure that the results were always the global optimal. This algorithm has been described as a flowchart in Figure 16.

COMPARISON OF THE ALGORITHMS

In this section, we have compared the results of both the algorithms using weights and without using the weights to achieve a better understanding of the results.

I. Without Weights

The accuracy of AutoEncoder model was **43.64%**, and the accuracy of the Rule-based algorithm was **68.3%**. Thus, the Rule-based algorithm outperformed the AutoEncoder, on comparison with the human-assigned ratings.

II. With Weights

The accuracy of AutoEncoder model was **59.74%**, and the accuracy of the Rule-based algorithm was **90.7%**. The Rule-based algorithm outperformed the AutoEncoder heavily, on comparison with the human-assigned ratings.

III. Analysis

The weight search algorithm could only improve the accuracy of the AutoEncoder's results by approximately 16%. On the other hand, the accuracy of the Rule-based algorithm improved more than 20% with the use of weights, and thus it can be said that it has been optimized quite a lot.

A possible explanation for this result would be that for the Rule-based methodology, each of the vector attributes represents a unique and individual game level component. As opposed to this, the attributes of the dense vector produced by the SAE are a combination of the correlated original features of the game level, and this representation is already optimized with every epoch of the model training phase, and thus cannot be optimized further.

CONCLUSION

From all the results that have been generated in this paper, it is quite clear that the Rule-based algorithm is in much more agreement with the human ratings than the AutoEncoder solution. This is interesting in the sense that the Rule-based algorithm does not consider the location data or the shapes and formations within the levels, whereas the SAE method considers it.

One possibility exists, that the SAE algorithm is in fact accurate in its ratings considering all the aspects of a game level, but the way humans perceive similarity between 2 levels does not highly agree with the SAE's logic, thus leading to a difference in the ratings. As noted earlier, the ratings assigned by humans tend to be heavily focused on the top layer details and the vivid colours present on the game board, whereas the SAE has its own way of determining the important features of the levels during dimensionality reduction.

REFERENCES

[1] D.E. Rumelhart, G.E. Hinton, R.J. Williams, "Learning internal representations by error propagation.", *Parallel Distributed Processing. Vol 1: Foundations. MIT Press*, 1986.

[2] G.E. Hinton, R.R. Salakhutdinov, "Reducing the dimensionality of data with neural networks", *Science*, 313, 2006, 504-507.

[3] V. Turchenko., A. Luczak, "Creation of a deep convolutional auto-encoder in Caffe", *arXiv*, 1512.01596, 2015.

[4] G.E. Hinton, A. Krizhevsky, S.D. Wang, "Transforming auto-encoders", *Lecture Notes in Computer Sci.,* 6791, 2011, 44-51.

[5] S. Rifai, P. Vincent, X. Muller, X. Glorot, Y. Bengio, "Contractive auto-encoders: Explicit invariance during feature extraction", *International Machine Learning Society*, 2011.

[6] J. Masci, U. Meier, D. Ciresan, J. Schmidhuber, "Stacked convolutional auto-encoders for hierarchical feature extraction", *Lecture Notes in Computer Sci.,* 6791, 2011, 52-59.

[7] D.P. Kingma, M. Welling, "Auto-encoding variational bayes", *arXiv,* 1312.6114, 2014.

[8] Gudmundsson, Stefan & Eisen, Philipp & Poromaa, Erik & Nodet, Alex & Purmonen, Sami & Kozakowski, Bartlomiej & Meurling, Richard & Cao, Lele, "Human-Like Playtesting with Deep Learning", 1-8. 10.1109/CIG.2018.8490442, 2018, 4.

[9] Y. Wang, H. Yao, S. Zhao, "Auto-encoder based dimensionality reduction", *Elsevier Neruocomputing*, 10.1016, 2015.08.104, 2016, 232-242

[10] "PyTorch - Tensors and Dynamic neural networks in Python with strong GPU acceleration", https://github.com/pytorch/pytorch/, accessed 30.05.2019.

[11] AFM. Agarap, "Deep Learning using Rectified Linear Units", *arXiv*, 1803.08375, 2019.

[12] V. Nair and G.E. Hinton, "Rectified linear units improve restricted boltzmann machines", *In Proc. 27th International Conference on Machine Learning (ICML)*, 2010.

[13] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, B. Frey, "Adversarial Autoencoders", *arXiv,* 1511.05644, 2016.

[14] P. Vincent, H. Larochelle, Y. Bengio, P.A. Manzagol, "Extracting and composing robust features with denoising autoencoders", *25th International Conference on Machine Learning (ICML)*, 2008, 1096-1103.

AUTHOR INFORMATION

Salil Deshpande, Student of Master of Information Systems Management, Carnegie Mellon University and PCG Research Intern at Levelup AI.

Simon Cheng Liu, Levelup AI, Beijing, China