Rotterdam School of Management
Erasmus University

Individual Assignment 4

Develop a Convolutional Neural Networks (CNN)

Khoi Gia Pham - 523755kg

Course: Introduction to Deep Learning CMME139

MSc Business Information Management

Rotterdam School of Management, Erasmus University

RSM - a force for positive change

RSM Erasmus

Table of contents

1. Introduction **3**

2. Data Preparation **3**

2.1 Data Selection process *3*

2.2 Image Processing *3*

2.3 Partitioning *4*

3. Model Development **4**

3.1 Baseline Model *4*

3.2 Number of layers *6*

Model 2a 6

Model 2b 6

3.3 Number of filters *7*

Model 3a 7

Model 3b 7

3.4 Kernal sizes *8*

Model 4a 8

Model 4b 9

3.5 Pool sizes *9*

Model 5a 9

Model 5b 10

4. Experiment Results **10**

4.1 Baseline model *10*

4.2 Number of layers *11*

4.3 Number of filters *12*

4.4 Kernal sizes *14*

4.5 Pool sizes *15*

5. Conclusion **16**

Bibliography **17**

1. Introduction

This report delved into the potential of using Convolutional Neural Networks (CNNs) to predict user rating count based on game iconography, aiming to uncover whether game icons hold predictable markers for such a task. We employed a baseline CNN model and several variations, tuning different parameters including the number of layers, number of filters, kernel sizes, and pool sizes, in our endeavor to optimize performance. Despite achieving a certain level of accuracy, the results suggested that while the model could learn patterns from game iconography, the predictability for user rating count was moderate and didn't improve significantly over the baseline. This implies that the relationship between game iconography and user rating count is complex, likely influenced by other factors such as game quality and gameplay that extend beyond the visual design. Our exploration underscores the importance of careful hyperparameter tuning in CNNs and highlights the need to strike a balance between model complexity and its generalization capability. Furthermore, it prompts the consideration of incorporating other relevant data for more accurate predictions, showcasing the complexity of the gaming landscape and the intricate task of predicting user engagement based on visual elements alone.

2. Data Preparation

2.1 Data Selection process

In the original dataset, NAs in the 'Average User Rating' column is replaced with 0. The reason behind this step is that, according to the paper by (Kerim and Genç, 2022),t games with NAs in this column did not garner more than 5 rating counts. I then selected the top 200 games with the highest and lowest average user rating count and downloaded their corresponding images.

2.2 Image Processing

Initially, I created an empty array and objective vector. The array was designed to accommodate the quantity of images in the directory and the resized dimensions of each image (128 x 128 pixels), alongside three channels accounting for the Red, Green, and Blue color components. This array was purposed to store the processed pixel data for each image. The objective vector, on the other hand, was prepared to store the labels corresponding to each image.

Next, I executed several operations for each image:

1. Extraction of the unique ID from the image’s filename.
2. Identification of the corresponding label in the dataframe. The identified label was then stored in the objective vector at the index corresponding to the current image.
3. Reading and loading the image file for processing.
4. Resizing each image to standard dimensions (128x128 pixels), to ensure all images shared the same dimensions before being input into the model.
5. Performing a normalization step to scale pixel intensities between 0 and 1.
6. Storing the processed image in the array at the appropriate index.

In conclusion, the format of the image input for the neural network is a 128x128 pixel color image, with pixel intensities ranging between 0 and 1.

2.3 Partitioning

The dataset was divided into a training set and a testing set. The training set comprises 70% of the data, while the testing set comprises the remaining 30%. The data is shuffled to ensure randomness before dividing. After splitting, the labels in both sets are restructured into a binary matrix representation (one-hot encoding), suitable for a two-class classification problem.

3. Model Development

This section presents the architecture of the baseline CNN model. I experimented with four primary parameters: the number of layers, the number of filters, kernel sizes, and pool sizes. For each parameter, I created two variations of the baseline model, one with an increase and one with a decrease. It's important to note that only the structure of these variations differs from the baseline model. All models maintain the same compilation parameters and training parameters as in the baseline model.

3.1 Baseline Model

This model serves as the baseline model for further parameter comparison.

Structure:

1. **Input Layer**: The input layer is set to accept images of size 128x128 pixels with 3 color channels (Red, Green, Blue).

2. **Convolutional Layer 1**: The first convolutional layer applies 32 filters of size 3x3 using the ReLU (Rectified Linear Unit) activation function.
3. **Max Pooling Layer 1**: The first max pooling layer downsamples the image data extracted by the convolutional layers to reduce the dimensionality of the image size. It does this using a 2x2 pool size.
4. **Convolutional Layer 2**: The second convolutional layer applies 64 filters of size 3x3, again using the ReLU activation function.
5. **Max Pooling Layer 2**: The second max pooling layer further downsamples the image data with a 2x2 pool size.
6. **Dropout Layer 1**: The first dropout layer randomly sets 25% of the input units to 0 at each update during training, which helps to prevent overfitting.
7. **Flatten Layer**: The flatten layer converts the 2D matrix data to a vector. This layer does not affect the batch size.
8. **Dense Layer 1**: The first dense (fully connected) layer has 128 neurons and uses the ReLU activation function.
9. **Dropout Layer 2**: The second dropout layer randomly sets 50% of the input units to 0 at each update during training time, again to prevent overfitting.
10. **Output Layer (Dense Layer 2)**: The second dense layer (output layer) has 2 neurons (presumably corresponding to 2 classes of output) and uses the softmax activation function to output a probability distribution.

Compilation Parameters

The model is compiled with the RMSprop optimizer and uses categorical cross-entropy as its loss function, which is suitable for multi-class classification problems. The metric that will be used to measure model performance during training and testing is accuracy.

Training Parameters

The model is trained for 10 epochs. An epoch is one complete pass through the entire training dataset. A batch size of 32 is used, meaning the model weights are updated after each 32 samples. And 20% of the training data will be set aside and used as validation data.

3.2 Number of layers

Model 2a

In this model, a third convolutional layer and its corresponding max pooling layer is added to the baseline model. The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 32 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Dropout Layer 1: 25% rate
5. Convolutional Layer 2: 64 filters, 3x3 kernel, ReLU activation
6. Max Pooling Layer 2: 2x2 pool size
7. Dropout Layer 2: 25% rate
8. Convolutional Layer 3: 128 filters, 3x3 kernel, ReLU activation
9. Max Pooling Layer 3: 2x2 pool size
10. Dropout Layer 3: 25% rate
11. Flatten Layer
12. Dense Layer 1: 128 units, ReLU activation
13. Dropout Layer 4: 50% rate
14. Output Layer (Dense Layer 2): 2 units, softmax activation

Model 2b

In this model, the second convolutional layer and its corresponding max pooling layer is removed from to the baseline model. The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 32 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size

4. Dropout Layer: 25% rate
5. Flatten Layer
6. Dense Layer 1: 128 units, ReLU activation
7. Dropout Layer: 50% rate
8. Output Layer (Dense Layer 2): 2 units, softmax activation

3.3 Number of filters

Model 3a

In this model, the first convolutional layer now applies 64 filters (up from 32) and the second convolutional layer applies 128 filters (up from 64). The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 64 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 128 filters, 3x3 kernel, ReLU activation
5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

Model 3b

In this model, the first convolutional layer now applies 16 filters (down from 32) and the second convolutional layer applies 32 filters (down from 64). The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 16 filters, 3x3 kernel, ReLU activation

3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 32 filters, 3x3 kernel, ReLU activation
5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

3.4 Kernal sizes

Model 4a

In this model, the first and second convolutional layers now use a 5x5 kernel size instead of 3x3. The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 32 filters, 5x5 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 64 filters, 5x5 kernel, ReLU activation
5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

Model 4b

In this model, the first and second convolutional layers now use a 2x2 kernel size instead of 3x3. The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 32 filters, 2x2 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 64 filters, 2x2 kernel, ReLU activation
5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

3.5 Pool sizes

Model 5a

In this model, the pooling layers now use a 3x3 pool size instead of 2x2. The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 32 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 3x3 pool size
4. Convolutional Layer 2: 64 filters, 3x3 kernel, ReLU activation
5. Max Pooling Layer 2: 3x3 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer

8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

Model 5b

In this model, the pooling layers now use a 1x1 pool size instead of 2x2. Please note that a pool size of 1x1 effectively means that there's no pooling happening, as the max pooling layer is essentially taking the maximum of a 1x1 patch, which is just the single element itself. It's not common to use a 1x1 pool size, but it can be experimented in specific cases. The structure of this model is as follows:

1. Input Layer: 128x128x3
2. Convolutional Layer 1: 32 filters, 3x3 kernel, ReLU activation
3. Max Pooling Layer 1: 2x2 pool size
4. Convolutional Layer 2: 64 filters, 3x3 kernel, ReLU activation
5. Max Pooling Layer 2: 2x2 pool size
6. Dropout Layer: 25% rate
7. Flatten Layer
8. Dense Layer 1: 128 units, ReLU activation
9. Dropout Layer: 50% rate
10. Output Layer (Dense Layer 2): 2 units, softmax activation

4. Experiment Results

4.1 Baseline model

The base model achieved a loss of 0.8599 and an accuracy of 54.17%. Analyzing Figure 1, we can identify two trends indicating potential overfitting. Firstly**, the training loss consistently decreases, while the validation loss remains stagnant**. Secondly, **the training accuracy significantly surpasses the validation accuracy, with the former continuing to rise while the**

latter plateaus. These patterns are classic indicators of overfitting. The model is excessively focusing on the specific examples within the training data, hindering its ability to generalize and perform well on unseen data. (Carremans, 2020).

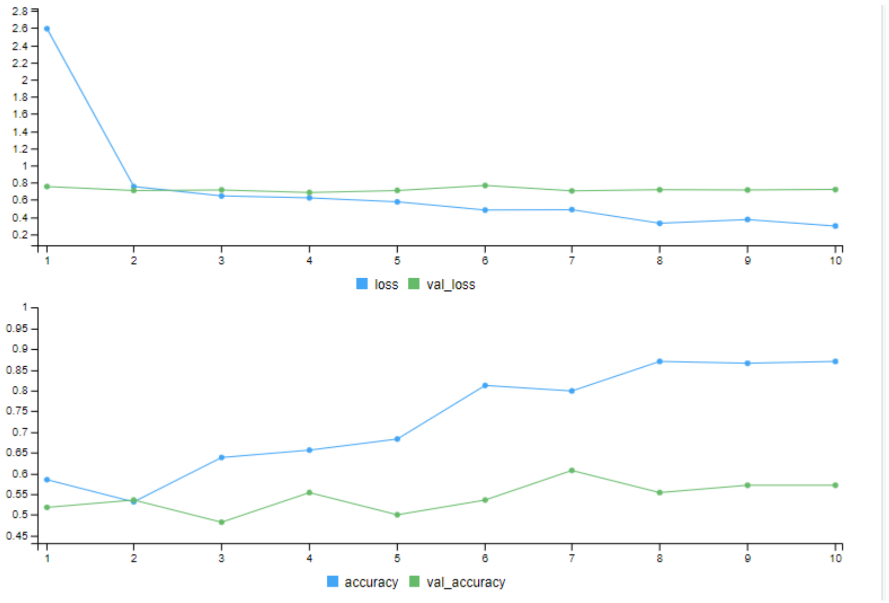

Figure 1. Training history of the Base line model

4.2 Number of layers

Table 1. Summary of model performance in changing number of layers

Model	**Loss**	**Accuracy**
Baseline	0.8599	54.17%
2a - Increase	0.7212	42.50%
2b - Decrease	1.4118	49.17%

2a - Increase: In this model, we add another layer of complexity (a third convolutional layer and its corresponding max pooling layer) to the baseline model. Theoretically, this is done to extract more intricate features from the data and potentially improve the model's accuracy. However, the accuracy here drops significantly to 42.50% and the loss decreases to 0.7212. This could suggest overfitting, where the model starts to learn the training data too well and performs poorly on unseen data. The decrease in loss but drop in accuracy is unusual, however, as typically when models overfit, the loss on the validation set or test set would typically increase. The unusual results could be due to noise in the dataset, an incorrect implementation, or perhaps the extra layer is capturing features that are not beneficial to improving the model's predictive performance.

2b - Decrease: In this model, a layer of complexity (the second convolutional layer and its corresponding max pooling layer) is removed from the baseline model. The accuracy decreases to 49.17% compared to the baseline, but it's higher than the model 2a. The loss however significantly increases to 1.4118. The increase in loss signifies that the model might be underfitting the data, meaning it's too simplistic to capture the relevant patterns in the training data. This is likely due to the reduction in the number of layers, leading to less expressive power for the model.

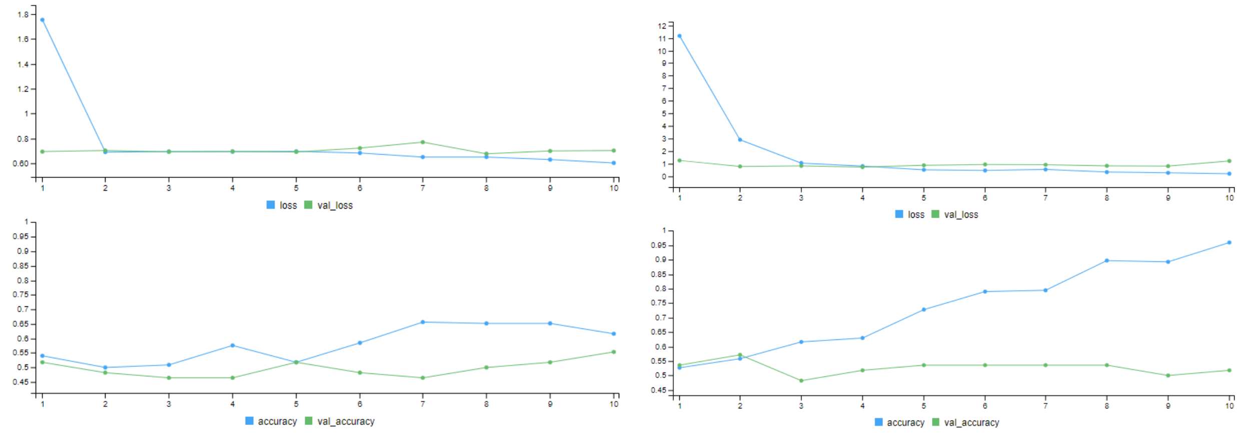

Figure 2. Training history of the model 2a (left) and model 2b(right)

In Figure 2, we also identify two trends indicating potential overfitting. Firstly**, the training loss consistently decreases, while the validation loss remains stagnant**. Secondly, **the training accuracy significantly surpasses the validation accuracy (model 2b)**. Interestingly, the accuracy curve of model 2a suggest underfitting as both training and validation accuracy remain low.

4.3 Number of filters

Table 2. Summary of model performance in changing number of filters

Model	**Loss**	**Accuracy**

Baseline	0.8599	54.17%
3a - Increase	0.8227	50.83%
3b - Decrease	1.1088	55.00%

3a - Increase: In this model, the number of filters in each convolutional layer is doubled, which is intended to allow the model to learn more complex representations from the data. However, the accuracy here decreases slightly to 50.83% and the loss also slightly decreases to 0.8227. The decrease in accuracy suggests that the additional filters might not be beneficial for this particular dataset, possibly leading to overfitting where the model learns the noise or outliers in the training data too well. However, the decrease in loss contradicts the decrease in accuracy, which might suggest that the model is improving in some areas (possibly on some classes in a multi-class problem), but overall the performance on the entire dataset is declining.

3b - Decrease: In this model, the number of filters in each convolutional layer is halved. Theoretically, this simplifies the model and might cause it to capture less complex features. Despite this, the model shows a slight improvement in accuracy to 55.00%, but the loss increases to 1.1088. The improvement in accuracy despite a higher loss suggests that the model might be performing better overall on correctly classifying instances, but when it makes errors, these errors are more severe (higher loss). The simplified model appears to generalize better than the more complex model (3a), potentially due to less overfitting.

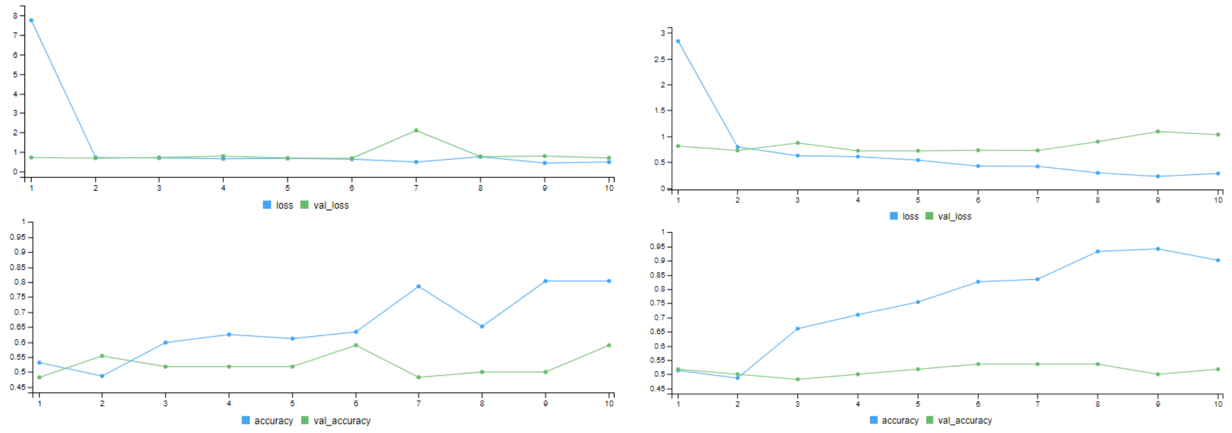

Figure 2. Training history of the model 3a (left) and model 3b(right)

Viewing Figure 3, both the loss and accuracy curves of the two models follows the similar trend of the baseline model, indicating potential overfitting.

4.4 Kernal sizes

Table 3. Summary of model performance in changing kernel sizes

Model	**Loss**	**Accuracy**
Baseline	0.8599	54.17%
4a - Increase	0.8810	46.67%
4b - Decrease	0.7573	47.50%

4a - Increase: In this model, the kernel size in the convolutional layers is increased from 3x3 to 5x5. Theoretically, a larger kernel size can capture more global or spatial information in the image, potentially extracting more diverse features. However, the model's accuracy decreases to 46.67%, and the loss increases to 0.8810. This could mean that the larger kernel size is not beneficial for this particular dataset and task, potentially because it might be too coarse and could be missing out on important local features that a smaller kernel size could capture.

4b - Decrease: In this model, the kernel size in the convolutional layers is reduced from 3x3 to 2x2. This smaller kernel size would focus more on local features in the image. The model's accuracy slightly decreases to 47.50% compared to the baseline, and the loss decreases to 0.7573. The reduction in loss might suggest that the model is improving in some areas (possibly on some classes in a multi-class problem), but overall, the accuracy on the entire dataset is declining.

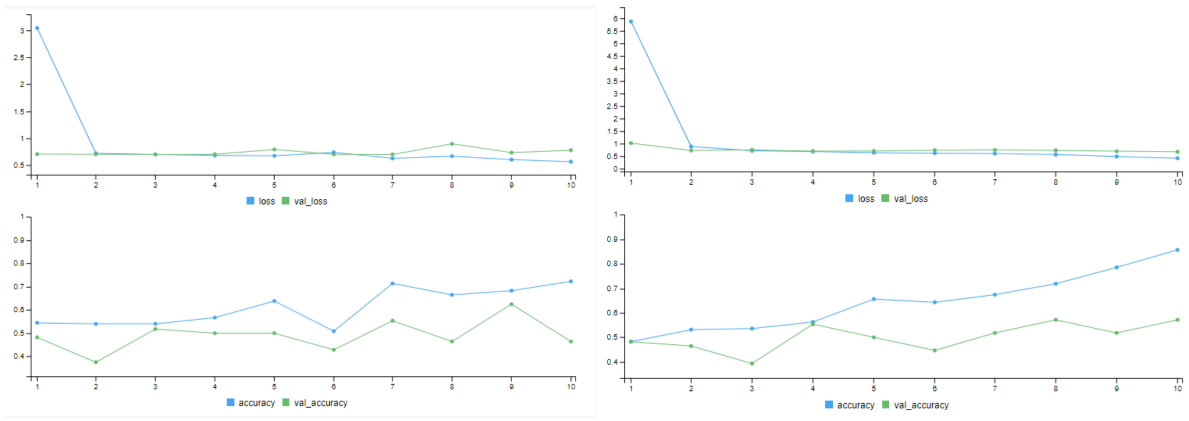

Figure 4. Training history of the model 4a (left) and model 4b(right)

4.5 Pool sizes

Table 4. Summary of model performance in changing pool sizes

Model	**Loss**	**Accuracy**
Baseline	0.8599	54.17%
5a - Increase	0.8141	55.00%
5b - Decrease	1.0035	51.67%

5a - Increase: In this model, the pooling size is increased from 2x2 to 3x3. Pooling layers are used to reduce the spatial size of the representation, making the model more invariant to small translations and reducing the amount of parameters and computation in the network. Increasing the pool size is supposed to capture more global features in the image. In this case, the model's accuracy slightly increases to 55.00% and the loss decreases to 0.8141. The improvement suggests that the larger pooling size might be extracting more beneficial features for this specific task and dataset.

5b - Decrease: In this model, the pooling size is decreased from 2x2 to 1x1. This means the pooling operation is applied to each individual pixel, which essentially doesn't reduce the spatial dimension. It leads to a decrease in accuracy to 51.67% and an increase in loss to 1.0035. The larger loss and decreased accuracy suggest that this model might be less effective at extracting the necessary features to improve classification, possibly due to overfitting or a high variance problem.

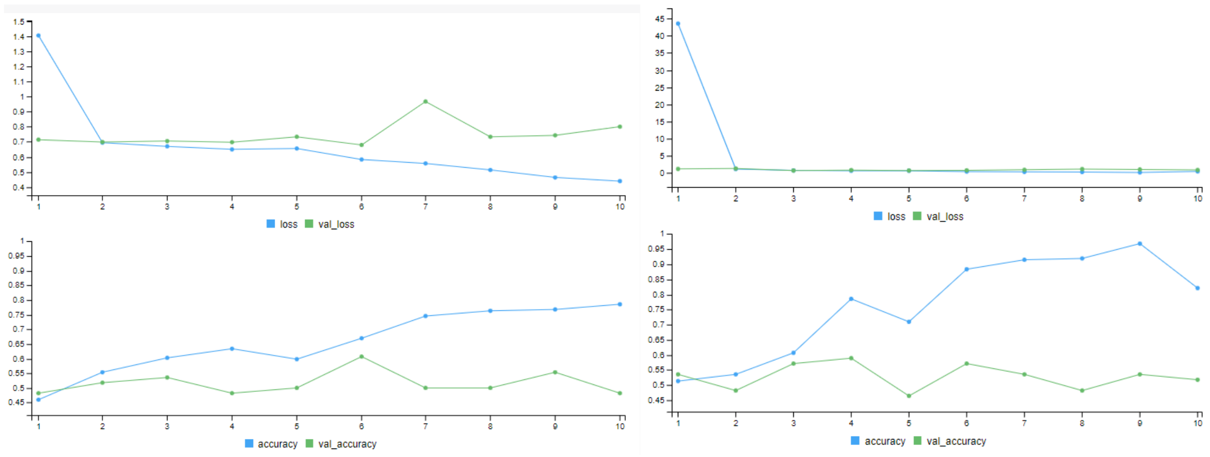

Figure 5. Training history of the model 3a (left) and model 3b(right)

In figure 4, again, both the loss and accuracy curves of the two models follows the similar trend of the baseline model, indicating potential overfitting.

5. Conclusion

While CNN models learned some patterns from game iconography, their moderate performance and lack of significant improvement over the baseline suggest limited predictability of user rating count solely from game icons. The relationship between iconography and rating count is influenced by factors like game quality, gameplay, and developer reputation. Thus, game iconography might have some predictive potential, but a more comprehensive model could provide more accurate predictions.

The investigation of CNN model parameters showed each significantly impacts performance. Extra layers or filters often led to overfitting and decreased accuracy, while reducing these simplified the model at an accuracy cost. Altering kernel size from the baseline 3x3 size impacted accuracy, and pool size adjustment revealed that larger pooling slightly improved accuracy. The results highlight the importance of hyperparameter tuning, balancing model complexity and generalization, and exploring other optimization strategies or model architectures.

Bibliography

Carremans, B. (2020) *How to Handle Overfitting in Deep Learning Models*, *freeCodeCamp.org*. Available at: https://www.freecodecamp.org/news/handling-overfitting-in-deep-learning-models/ (Accessed: 29 May 2023).

Chollet, F., & Allaire, J. J. (2018). Deep Learning with R. Manning Publications.

Kerim, A. and Genç, B. (2022) 'Mobile games success and failure: mining the hidden factors', *Neural Computing and Applications* [Preprint]. Available at: https://doi.org/10.1007/s00521-022-07154-z.