Unsupervised classification of video games styles

Camille Montemagni, Hugo Hueber, Luca Joss
Computer Science, EPFL, Switzerland
{*camille.montemagni, hugo.hueber, luca.joss*}*@epfl.ch*

***Abstract*—We explored an unsupervised model, using an autoencoder and k-means clustering layer to check if it was possible to classify the graphic style of video games in order to find similarities. We focused on the classification of games from two different consoles, Nintendo's Super Nintendo and Sega's Genesis, to verify the power of the classification. We are getting satisfactory results in terms of balancing, and this tool could be used for further research.**

I. Introduction

Overview: For this project we explore how an unsupervised classification of video game screenshots behaves. We focus on images from consoles of the same generation, the Super Nintendo released in 1991 in North America and 1992 in Europe, and the Sega Genesis, released in 1989 in North America and 1990 in Europe.

The initial idea is to observe and identify what the unsupervised classifier uses to distinguish two clusters after being trained on a data set balanced in the amount of images from different video game consoles.

If the results are satisfactory this could lead to researching more sophisticated models which could be applied to larger and more diverse data sets.

II. Models and methods

A. Method selection

Autoencoder: Since the resolution of the Super Nintendo is between 256x224 and 512x448 pixels, depending on the games, and the resolution of the Sega's Genesis in NTSC regions is 320x224 pixels we first chose to set all the images of our database to 320x224 pixels. This makes every image have 71'680 values in grayscale, which is too high for k-means clustering with the computational power we have access to.

To reduce the amount of features per image, we built an autoencoder which encodes images to 4480 features, which is 16 times smaller than the original images. But the required layers and number of neurons to train a successful autoencoder were too high, so we reduced the images to 160x112 pixels, resulting in 17'920 values per input image, and having an encoded image being 4 times smaller than the new original image. In addition to encoding the images in smaller size, the autoencoder's purpose is to extract meaningful data from the images, instead of simply reducing the size of the images.

For the number of hidden layers and the size of these layers, we used the table from Jeff Heaton's post [1], and chose to use 3 hidden layers for the encoder (and 3 hidden layers for the decoder) since we wish to learn complex representations, making it a deep autoencoder. We started using a number of neurons for the hidden layers equal to $\frac{2}{3}$ Input layer neurons plus the number of Output layer neurons, and then proceeded to trial and error to find the final Hidden layer sizes as seen in the figure [1].

Clustering: Once the autoencoder has been trained, we can encode the images down to 4480 features, and thus feed this output as input to a k-means clustering layer. The new model will be composed of the encoder part of the autoencoder with a new layer taking the 4480 output as input and output the student's t-distribution for each image.

B. Exploratory data analysis

Data Collection: To generate the training and testing data set, multiple python scripts were used to extract images from videos. We had to select videos which had the purest images of multiple games from the desired consoles and extract them. By pure, we mean with as little as possible watermarks or other additional text on the images which aren't related to the video game. The training and testing data sets are smaller than what would ideally want, and the method to extract the images isn't the best, but is enough for us to run the experiment.

Autoencoder: Since we wish to train the unsupervised model to cluster video game images, and observe the behavior of the clustering on images from Super Nintendo and Sega's Genesis, we removed any images which had nothing to do with the video games (such as screenshots from transitions between two games).

Clustering: The same data set for training the autoencoder was used for training the clustering. If we had a larger database, it would have been interesting to use different data sets for training the models.

For the k-means clustering layer added to the model, the number of features had to be decreased, this was done by using the autoencoder.

C. Data augmentation

No rotation or translation was used for data augmentation, since we train and test on images taken from videos of the games and not photos, which may not be centered.

D. Cross-validation

Autoencoder: Validation is done by splitting the training data set into 80% training data and 20% validation data. By giving the training data and validation data to the keras[1] library, it will compute the validation loss while fitting the model.

As mentioned in the Method selection II-A, the number of hidden layers was set to 3, but for the number of neurons, as encouraged by Jeff Heaton [1], was done by running different dimensions for the hidden layers and finally choosing the dimensions which got the lowest validation loss.

Clustering: Since we observe the clustering done by the unsupervised model, we don't have labels which could serve as validation for the clustering. The model was trained for a fixed number of iterations or until the number of labels which changed is less than a fixed tolerance value.

III. Results

Autoencoder: The autoencoder was trained for 1000 Epochs in the Model shown in figure [1].

The training loss and validation loss went down to 0.0043 for the loss and 0.0071 for the validation loss, which is a satisfying number, and as seen in figure [2], the validation loss isn't increasing, so it is not over fitting.

The results of the reconstruction, shown in figure [3] aren't as satisfying as we hoped, this might be due to the fact we have less neurons in the hidden layers as we would have hoped to have. But we are still able to have recognizable reconstructions, which is promising. And as mentioned in the Method selection II-A, our data sets are in grayscale, which was due to lack of memory on the machine used to train the model. It would have been interesting to be able to keep the RGB and observe the results.

Clustering: After training the clustering model, feeding it our test set results in the following cluster distribution [4]. The images are clustered in a 60/40 ratio between the two clusters. If the random seeds are removed, the clustering keeps the same ration, but the clusters are inverted at some runs. For example, first run cluster 1 has 60% of the images with majority of Super Nintento images and cluster 2 has 40% of the images, with majority of Sega's Genesis images such as figure [4]. While a later run would have cluster 1 with 40% of images and a majority of Sega's Genesis images, while cluster 2 has 60% of images with a majority of Super Nintendo images.

IV. Discussion

Data collection: The data collection could be improved by setting up a machine running emulators of game consoles combined with a script which would regularly take screenshots. This method would avoid watermarks or other data which has nothing to do with the games, as well as decrease

[1]https://keras.io/

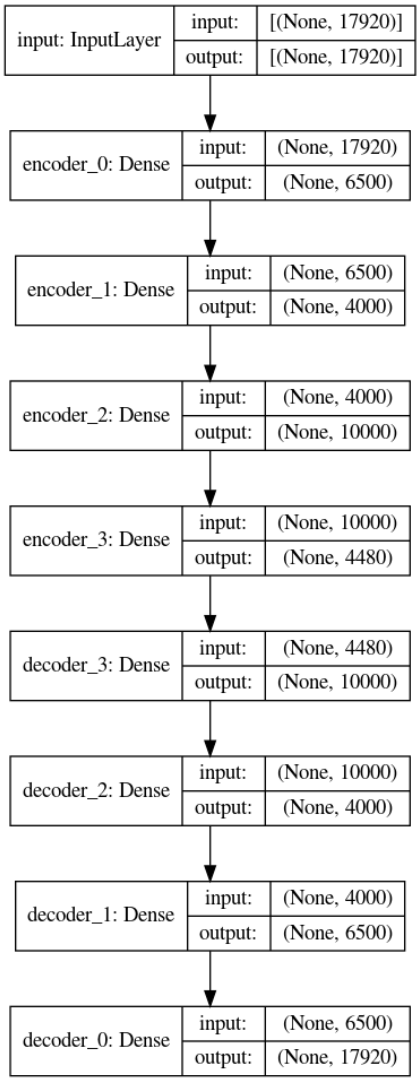

Figure 1. Autoencoder Model

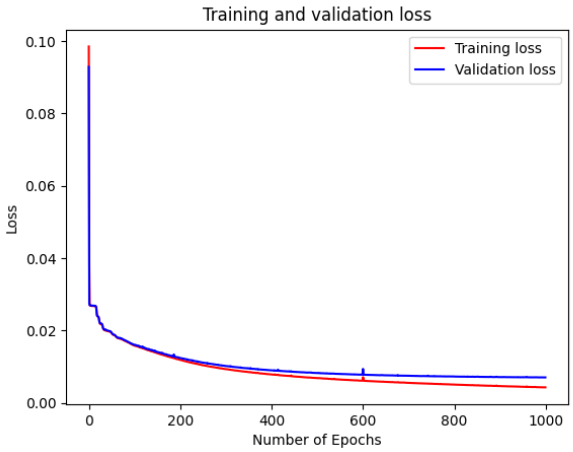

Figure 2. Training and validation loss when fitting the autoencoder

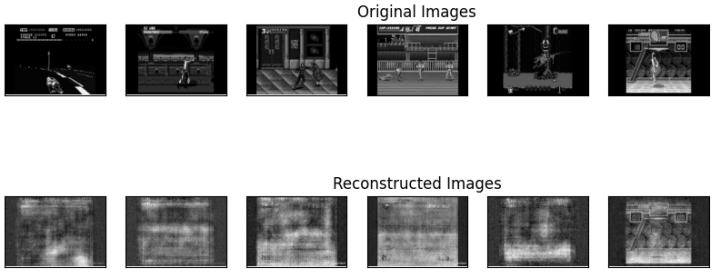

Figure 3. Input and output of our autoencoder after training

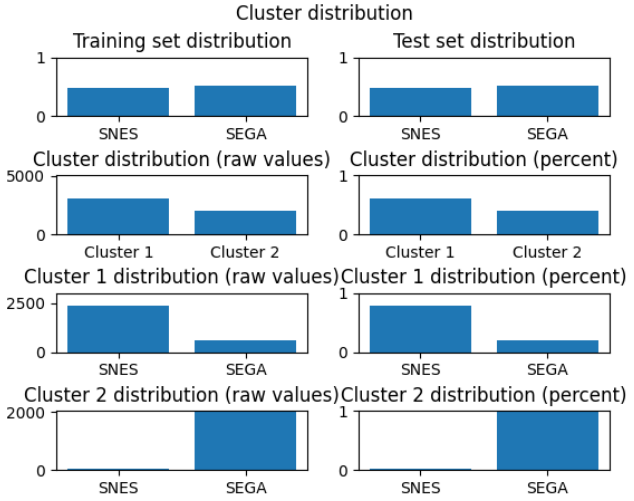

Figure 4. Cluster distribution when given our test data as input

the manual pre-processing of removing images which had nothing to do with video games.

Increasing the database would help train more robust models, but would come to the price of requiring more computational power, which was already a very big limiting factor.

Computational power: A GTX 1060[2], which has 6 GB of memory and 1280 CUDA cores, was used with CUDA to train the autoencoder. Having more computational power, such as a GTX 3080 [3] hich has 8704 CUDA cores and 10 GB of memory, would allow more exploration when training the autoencoder. And as discussed in the Method selection II-A, it would allow to have images reflecting the resolution of the console and to see if the color palette has an effect on the clustering of the images.

Data Augmentation: Training the autoencoder with noisy images could help with the watermarks, but if a data collection system which could generate clean images is set up, but same as mentioned in the data collection, this increase would come at memory cost.

Autoencoder: The results show us that for a good autoencoder, it is quickly required to have a sophisticated model, but we still have acceptable results with our simplified model. The final model used makes it possible to have more consistent clustering. When the clustering model was run with an encoder trained to reduce too much, encoding images of 71'680 pixels down to 400 features, the clustering would not be consistent between different runs. This is a sign of a bad model, even with the random variables, the clustering should be consistent. To encode images with such a big difference requires more layers and neurons, and thus shows the important of the autoencoder model for our classification experiment.

After the reduction of the original images to 17'920 pixels, using an encoder which would reduce the images from 17'920 pixels to 400 features would still have a decent clustering, but less performant and consistent as when the encoder reduces the images from 17'920 pixels down to 4480 features. Since we have access to a machine that could run the later, the final model was chosen to encode down to 4480.

Clustering: Seeing the results are similar from different runs after removing the random seed fixation shows that our model works as we want it to. By observing the global clustering, we notice the model seems to notice a difference between games of different console. We will refer to cluster 1 as the Nintendo cluster and cluster 2 as the Sega cluster.

When observing the Sega images in the Nintendo cluster, we see that most of them are a few "random" images from some games, which makes us think of the error factor in the clustering model. But we can also see whole games inside this cluster, such as the game "Super Street Fighter II: The New Challengers" [5], which is images `test_sega1723.png` to `test_sega1806.png` in the test set. The same can be observed for the other way around, but with less samples. There is one full game of Super Nintendo inside the Sega cluster, which is Mario Paint [6], the interesting detail with this Mario game is the amount of black and white, which will result in extreme points in grayscale. Another hypothesis is the more 2D style of the Mario game which could be seen similar to most of the Sega's Genesis games, such as Sonic [7].

V. Summary

What was done: The main issues encountered were related to training an autoencoder so it would perform decently. Even if the resolution of the game consoles we focused on weren't huge, around 70'000 pixels per image,

[2]https://www.nvidia.com/en-sg/geforce/products/10series/geforce-gtx-1060/

[3]https://www.nvidia.com/en-gb/geforce/graphics-cards/30-series/rtx-3080/

Figure 5. Screenshot from the Sega game Super Street Fighters II: The New Challengers

Figure 6. Screenshot from the Super Nintendo game Mario Paint

the computers we had access to couldn't manage this amount of data. By applying reductions on specific parts, such as the images resolution, using grayscale instead of keeping the RGB and the number and size of hidden layers, we managed to obtain interesting results. With these reductions, we still managed to get satisfactory results with the autoencoder, and thus trained the clustering model.

The clustering model has a global performance better than what we had predicted. It managed to separate the test data well between Super Nintendo and Megadrive games as can be seem in figure [4].

Potential future steps: Seeing the promising results from our simplified model, it is encouraging to explore more this road for classification of video game styles. By having a more sophisticate model, and then increasing the number of clusters, it could be interesting to observe if game types could be classified, or if any notable features could be extracted from the games.

A potential path would be to compare the clustering when keeping the RGB images and when converting to grayscale, to observe if the color palette might be a factor taken into account by consoles for their games.

Furthermore, this model could be extended to more recent consoles, or to compare recent games which try to imitate old school styles, such as Shovel Knight or Valfaris.

Figure 7. Screenshot from the Sega game Sonic the Hedgehog 2

ACKNOWLEDGEMENTS

Thank you to Nicolas Flammarion and Martin Jaggi for the teaching. Thank you Yannick Rochat and Selim Krichane for the supervision of our unsupervising. Thank you to our respective roommates for putting up with us during this confinement. A special thanks to Gustave Pistache for the exchanges and help provided.

REFERENCES

[1] J. Heaton, "The Number of Hidden Layers," https://heatonresearch.com/2017/06/01/hidden-layers.html, 2017, [Online; accessed December-2020].