Project report
Deep learning: binary image classification

LUCA DONGHI
luca.donghi1@studenti.unimi.it, 982862

30/03/2022

1. Task

The task is to implement a deep-learning-based system discriminating between real faces and comics, using the *Comics faces* dataset. This dataset is published on Kaggle and released under the CC-BY 4.0 license, with attribution required. It contains 10.000 real faces images and 10.000 comics faces images in 1024x1024 format.

2. Introduction

This report will briefly explain the process undergoing the creation of a well-performing binary classifier. The chosen deep-learning system is the Convolutional Neural Network (CNN), a class of Artificial Neural Network, largely applied to analyze visual imagery. The report will cover all the relevant part of the algorithm implementation: data organization and preprocessing, creation of the CNN, model selection and the discussion of the results.

3. Data

3.1. *Dataset*

The *Comics faces* Kaggle dataset is composed of 20.000 face images: 10.000 real faces images and 10.000 comics faces images in 1024x1024 format. The real faces and comics faces images are paired: for each real image there is its comics version.

3.2. *Data organization*

The complete dataset has been used to train, validate and test the models. It has been downloaded directly from Kaggle to Google Colab using the Kaggle API. For the Jupyter Notebook attached to this report is directly executable on Google Colab by loading your Kaggle API Token in the designated code cell. To each downloaded images a label has been applied: 0 for real faces and 1 for comics faces. Then, the images have been separated from the labels and respectively stored in an X and y variable.
After being preprocessed (see Subsection 3.3), the complete dataset, composed by pairs image-labels, has been splitted in two set: Training Set and Test Set.
From the training set (x_train, y_train) two smaller training set has been created in order to speed up the computation for the Grid Search. The first subset of the training set (x_train_try1, y_train_try1) composed of 4800 examples is used in the first round of model selection for training 27 models.
The second subset of the training set (x_train_try2, y_train_try2) composed of 10.800 examples is used in the second round of model selection for training 6 models. I decided to implement this system of subset of the training set to be able to try a larger number of models in the available time. The second round with larger number of examples is used because the best models could be different for different training set sizes.
The training set is further splitted in the each training section in Training Set and Validation Set. The validation set is automatically created by Tensorflow to plot the performances of the trained models and perform model validation.

3.3. *Preprocessing*

The 1024x1024 images has been reshaped into 150x150 images for computational and storage capacity issues. This specific shape has been selected, after different manual visualizations, because the images still remained reasonably detailed and the size is not too large.
For the same reasons the images have been processed in gray scale.
The entire dataset has been shuffled. This is very important because initially our images are sequentially ordered for each category. This would lead the CNN to learn "always predict 0 and at a certain point 1" leading to terrible performances on new data.
The last preprocessing step consist in scaling the images such that their pixel val-

ues are normalized between 0 and 1. This makes the work a lot easier for the CNN.

4. Algorithm implementation

4.1. *Convolutional Neural Network*

The Convolutional Neural Network is a Neural Network to which a convolutional layer has been added. This technology is very powerful when processing images because it is able of extract features from images. The convolutional layer, through a filter that scan (convolve) the pixel of the images, is able to detect pattern: the deeper you go with the convolutions, the most likely is that the patterns become features of the images. Working with face images it would probably start from very easy shapes like lines and circles ending up with noses, eyes, mouths and so on and so fourth. Each convolutional layer is associated to a pooling layer (typically max pooling) which is needed to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network.
For this reasons the CNN was the most reasonable choice for the task

4.2. *how the proposed solution scales up with data size?*

The CNN I modeled, and more in general all the code I wrote in the Jupyter Notebook, easily scale up with data size. Indeed, to process a larger number of examples you only need grater computational and storage resources. For example using Google Colab you only need to pay for larger memory space and faster GPUs.

4.3. *Model structure*

All the models I trained have a common structure and only some hyperparameters have been tuned.

Common structure:

- Every model have at least 1 convolutional layer
- Every convolutional layer have a 3X3 window
- Every convolutional layer is associated to a Relu activation function

- Every convolutional layer is associated to a 2X2 Max Pooling layer
- Every model have a Flatten layer after the convolutional layers
- Every model have a final dense layer of size 1 and a final Sigmoid activation function
- If a model have additional dense layers they are associated to a Relu activation function
- Every model uses a binary crossentropy loss function
- Every model uses Accuracy as metric
- Every model uses Adam Optimizer for the learning rate

While many of the parts composing the common structure are simply chosen because they are efficient choices used in the literature for this kind of classifier, some of them could have been tuned. Only some Hyperparameters have been tuned because of the resources (time, computational, storage). The value chosen for the other optimizable Hyperparameters are simply standard choices.

Adam Optimizer is almost always used in order to optimize the learning rate based on the momentum of the Stochastic Gradient Descent making the computation more efficient
The Relu activation function is a non-linear function which main advantage is that it does not activate all the neurons at the same time. It is more computationally efficient than the Sigmoid function (Another natural choice). For this reason the Relu function has been used for each internal layer, while the Sigmoid function has been only used at the final step.
The Flatten layer is needed to convert from the 2-dimensional output of the convolutional layer to the 1-dimensional imput of the dense layer.
The binary crossentropy loss function and the Accuracy metric are simply obvious choices for this task.
The 3X3 size of the convolutional window and the 2X2 size of the max pooling layer, as well as the choice of a max pooling as pooling layer, are standard choices I decided to use.

4.4. *Model Selection*

To train all the Hyperparameters for a large number of values was definitely not a feasible path. In order to decide which kind of Grid Search to perform I previously experimented with few, almost randomly, chosen values for 2 epochs.

The results lead me to some reasonable values for the three Hyperparameters I wanted to tune: the number of convolutional layers, the size of the layers and the number of additional dense layers

Values used for the first Grid Search:

- Number of Convolutional layers: 1, 2, 3
- Layer sizes: 32, 64, 128
- Number of additional dense layers: 0, 1, 2

To train and evaluate on the entire training set all the 27 models that are output by this Grid Search a very large amount of time is required. I decided perform the model selection through two round of Grid Search.

For the first round I used the first subset of the training set (x_train_try1, y_train_try1) composed of 4800 examples. On this small dataset all the 27 models have been evaluated for 10 epochs and using 30% of the data as validation set.

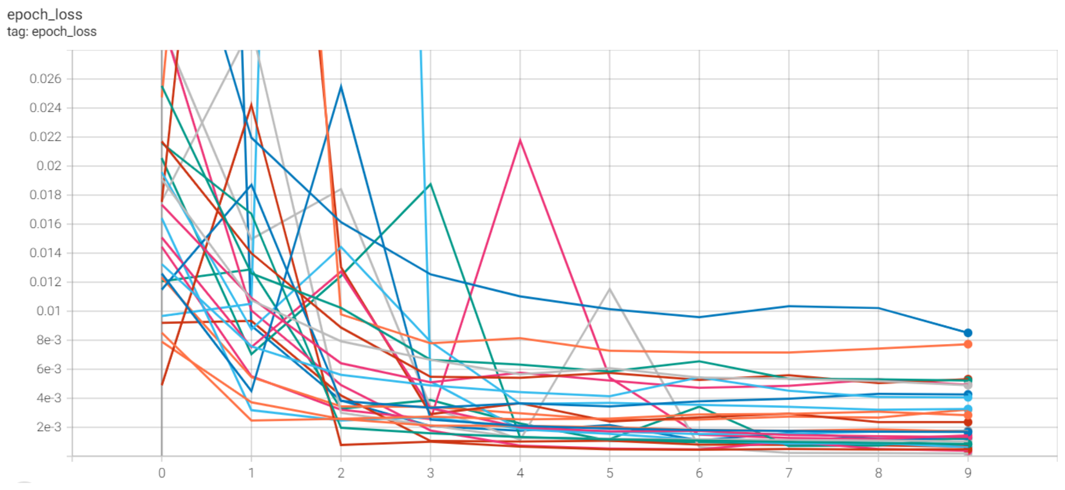


Figure 1. First round validation loss by epochs- 27 models using 4800 examples

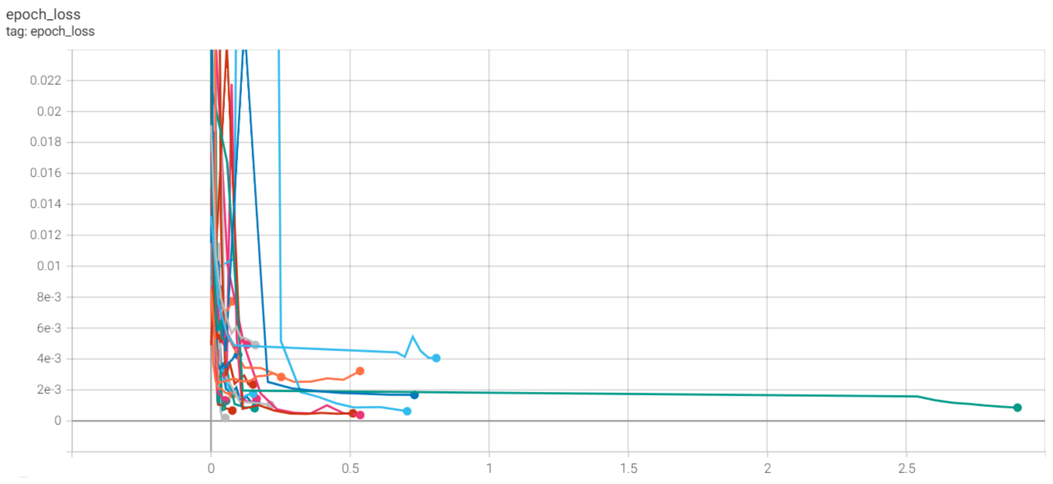
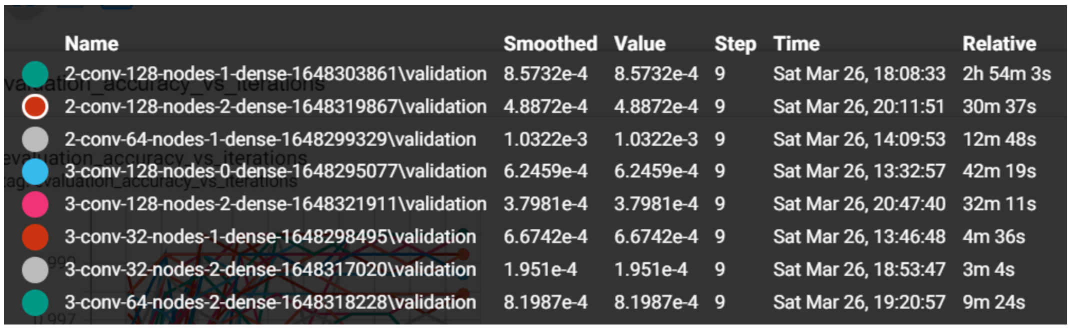


Figure 2. First round validation loss by time- 27 models using 4800 examples



	Name	Smoothed	Value	Step	Time	Relative
●	2-conv-128-nodes-1-dense-1648303861\validation	8.5732e-4	8.5732e-4	9	Sat Mar 26, 18:08:33	2h 54m 3s
●	2-conv-128-nodes-2-dense-1648319867\validation	4.8872e-4	4.8872e-4	9	Sat Mar 26, 20:11:51	30m 37s
●	2-conv-64-nodes-1-dense-1648299329\validation	1.0322e-3	1.0322e-3	9	Sat Mar 26, 14:09:53	12m 48s
●	3-conv-128-nodes-0-dense-1648295077\validation	6.2459e-4	6.2459e-4	9	Sat Mar 26, 13:32:57	42m 19s
●	3-conv-128-nodes-2-dense-1648321911\validation	3.7981e-4	3.7981e-4	9	Sat Mar 26, 20:47:40	32m 11s
●	3-conv-32-nodes-1-dense-1648298495\validation	6.6742e-4	6.6742e-4	9	Sat Mar 26, 13:46:48	4m 36s
●	3-conv-32-nodes-2-dense-1648317020\validation	1.951e-4	1.951e-4	9	Sat Mar 26, 18:53:47	3m 4s
●	3-conv-64-nodes-2-dense-1648318228\validation	8.1987e-4	8.1987e-4	9	Sat Mar 26, 19:20:57	9m 24s

Figure 3. First round validation loss data - best 8 models using 4800 examples

By inspecting the performances of these 27 models on Tensorboard we can draw some considerations.

Models without any additional dense layer are completely instable
The Validation Loss curves completely instable that are displayed in Figure 1 are all models without any additional dense layer

Models with 128 as layer sizes have very large training duration without too much improvement The Validation Loss curves with the longer training durations that are displayed in Figure 3 are all models with 128 as layer sizes.

Models with only 1 convolutional layer have bad performances and models with 3 convolutional layers are definitely the best
Indeed we can see in Figure 2, which display the best 8 models performances, that no models with only 1 convolutional layer appears and that models with 3 convolutional layers definitely prevail.

For the second round I used the second subset of the training set (x_train_try2, y_train_try2) composed of 10.400 examples. On this dataset 6 models have been evaluated for 8 epochs (I realised that after 8 epochs there were no relevant improvement) and using 20% of the data as validation set.

Values selected for the second Grid Search based on previous considerations:

- Number of Convolutional layers: 3
- Layer sizes: 32, 64, 128
- Number of additional dense layers: 1, 2

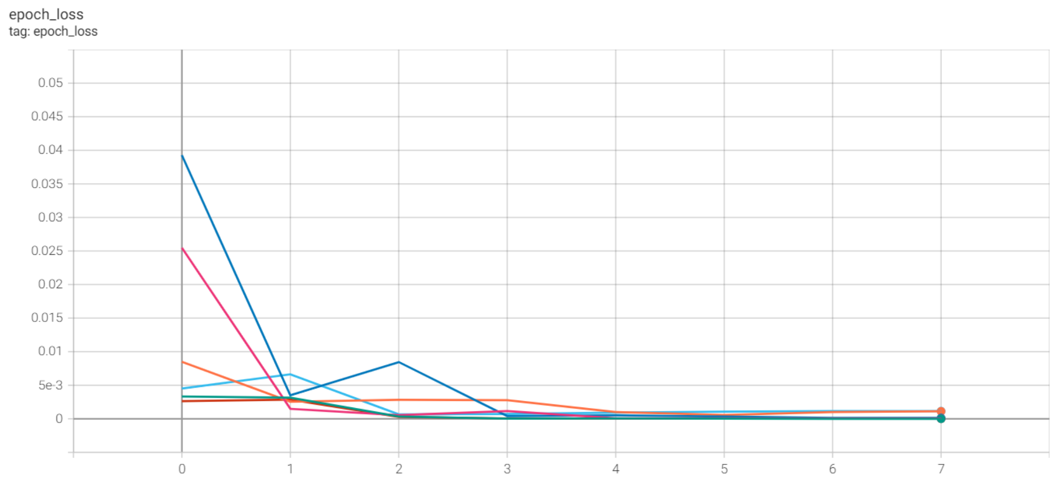


Figure 4. Second round validation loss by epochs- 6 models using 10.800 examples

Not including models with zero dense layer was definitely a good choice: all the models are now stable as we can see in Figure 4.

The best 4 models in terms of validation loss are:

1. 3 Convolutional Layers, Layer sizes 128 and 1 dense layer. Very large training duration: 1h, 34 min and 18 sec
2. 3 Convolutional Layers, Layer sizes 64 and 2 dense layer. Quick training duration: 18 min and 25 sec
3. 3 Convolutional Layers, Layer sizes 128 and 2 dense layer. Large training duration: 1h, 5 min and 33 sec
4. 3 Convolutional Layers, Layer sizes 32 and 1 dense layer. Definitely the most quick for training: 6 min and 36 sec

This 4 best models have been trained on the entire training set (x_train, y_train) composed of 16.000 examples for 10 epochs.

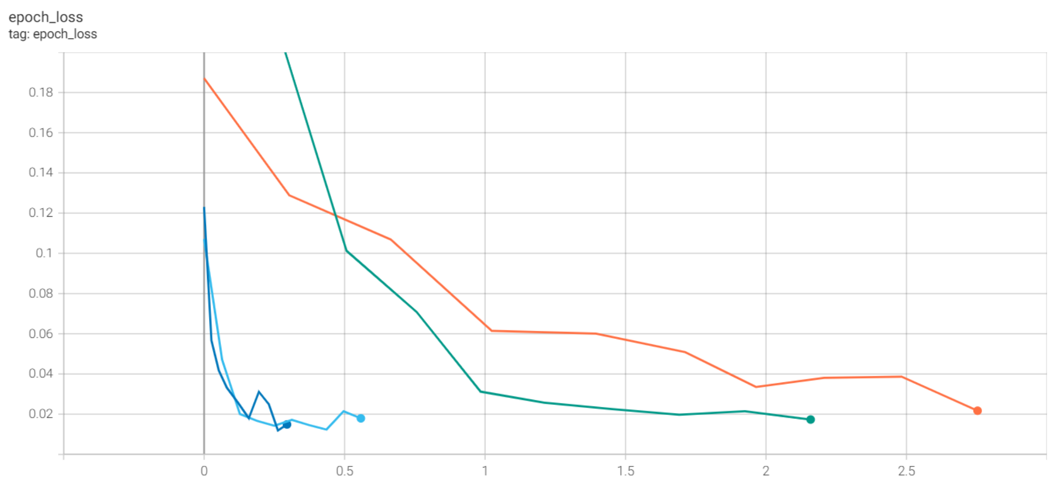


Figure 5. Third round validation loss by time- 6 models using 16.000 examples

Looking at Figure 5 we can notice that the best two models, the ones with 32 and 64 as layer sizes, are also by far the faster in the training session. This two models will be my 2 final models to be tested on the test set

4.5. *Can we further improve our models?*

With a larger dataset we could have definitely improved our trained models. However we don't have it.
Instead, something we could have definitely done is a deeper hyperparameters tuning.
I have only perform model selection on few hyperparameter values. For this project we don't have neither the computational capacity nor the time to perform a larger Grid Search.
However based on what we learned from previous steps there is something we can try.
We know that the layer sizes can not be enlarge for time resources and it seemed not to improve too much performances. Excluding the case with zero dense layers, this hyperparameter value alone seemed not to change the performances.
The number of convolutional layers instead could definitely be a leading factor in

model performances.

We can try our best performing model (3 conv, 64 layer sizes, 2 dense) with 4 or 5 convolutional layers!
Indeed, they perform even better than our best models. The last attempt I want to do is to see if reducing the layer sizes of the model with 5 convolutional layers would reduce the training duration without worsening the accuracy: we try 5 convolutional layers, 32 as layer sizes and 2 additional dense layers.

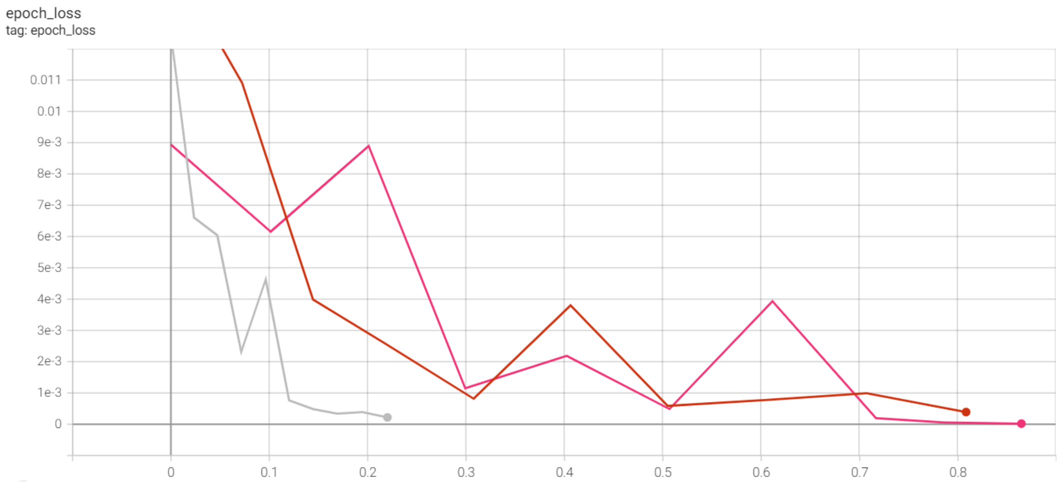


Figure 6. Experiment with more convolutional layers - validation loss by epochs - 3 models using 16.000 examples

As we can see in Figure 6 the last model I experimented is the by far the faster and is only slightly worse performing than the best one.

4.6. *Testing the models on the Test Set*

After the training the 3 best models have finally been tested on the test set (x_test, y_test) composed of 4000 examples. Their performances are represented with: the test loss,the test accuracy and the number of misclassified images:

- **4 Convolutional Layers, Layer sizes 64 and 2 dense layer**.
  test loss: 0.250842422246933
  test accuracy: 0.9992499947547913
  It misclassified only 3 comics faces as real faces and nothing else

- **5 Convolutional Layers, Layer sizes 64 and 2 dense laye**r.
  test loss: 0.05062811076641083
  test accuracy: 0.999750018119812
  It misclassified only 1 comics faces as real faces and nothing else

- **5 Convolutional Layers, Layer sizes 32 and 2 dense layer**.
  test loss: 0.0555519200861454
  test accuracy: 0.999750018119812
  It misclassified only 1 comics faces as real faces and nothing else

5. Conclusions

In conclusion, a larger number of convolutional layers could be tested. Other technologies could have been added to try to improve the performances.
A typical choice for this kind of models is to add a Drop Out Layer. It is used to prevent overfitting from happening. I didn't include it because definitely I had no overfitting issues.
The best model I trained is definitely the last one because it has slightly worse performance than the same model with 64 as layer sizes, but it is a lot more fast to train.

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.