UNIVERSITÀ
DEGLI STUDI
DI MILANO

Department of Economics, Management and Quantitative Methods
Università degli Studi di Milano
Data Science and Economics

ALGORITHMS FOR MASSIVE DATA MODULE

Comics faces

Shaikyp Ulan (944462)

September 18, 2022

Abstract
The goal of this project is to develop a deep-learning-based system to discriminate between real faces and comics, using the “Comics faces” dataset available on Kaggle. For this task I explore the Convolutional Neural Networks (CNN), which are largely used for image classification.

Introduction
The performance of well-known convolutional neural networks (CNNs) for categorizing images into two classes—Real Face Pictures and Comic Pictures—is empirically examined in this research. This experimental project's main objective is to train a computer to recognize image categories using Deep Learning methods.

Dataset
The “Comic faces” dataset is published on Kaggle and released under the CC-BY 4.0 license, with attribution required. It contains pictures organized in two folders. One containing 10’000 pictures of real images. The other one containing 10’000 corresponding drawings of the real images, that are the comics. For a total of 20’000 images in 1024x1024 format.

Methodology

A Convolutional Neural Network is a Deep Learning algorithm which can take an image as input, detect its main characteristics and distinguish it from others. Hence, the algorithm reduces the images without losing critical features for a good prediction. A layer in the CNN is composed by a “convolutional layer” and a “pooling layer”. Where the number of layers defines the level of details captured. The higher is this number, the higher is the complexity and the more computational power is required. The convolutional operation takes place in the Convolutional Layer and involves the “filter” element. This shifts through the image until it is considered in its entirety. The number of times this filter shifts depends on the “stride length”. The objective is to extract high-level features from the image. Actually, the first convolutional layer captures the low-level features. The high-level features are captured adding more layers, which define the network architecture. On the other hand, the Pooling Layer reduces the dimensionality of the convolved features, helping in decrease the computational power required. Also, it extracts the dominant features. It is a “max pooling” if it returns the maximum value of the filter’s window. Or it is an “average pooling” if it returns the average of the values of the filter’s window. Generally, max pooling performs better. Once the model can understand the features, the image must be flattened into a column vector in order to be passed to a “Dense layer”, also called “fully-connected layer”. Each neuron of this layer receives input from all the neurons of previous layer. This is given as input to a neural network able to perform the classification using a Softmax function. This technique has two interesting results: the obtained values sum up to 1 and they are all non-negative. This means that we can interpretate them as a probability distribution over the classes. In our case, the probability of being a real face image or a comics image.

Model.

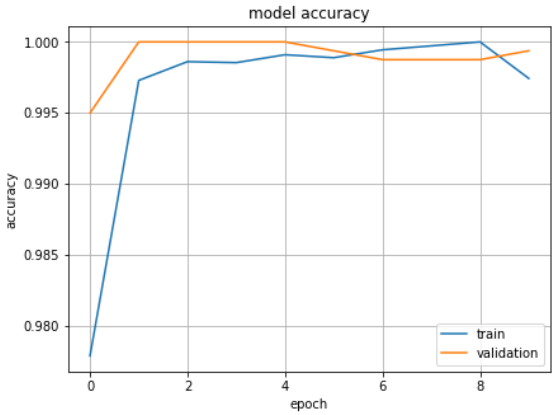
Convolution Neural Networks (CNNs) are a class of Neural Networks that have excelled in the areas of image recognition, processing, and classification, and I have employed them in my experimental project. In order to determine training weights for the CNN model and to evaluate its performance, training data is necessary.

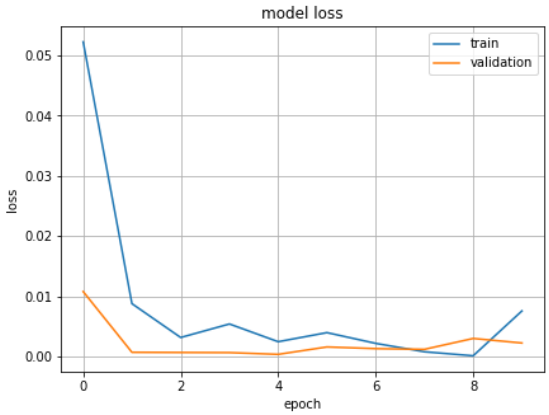
A neural network has many hyperparameters to set. But there is not a rule of thumb to define the values that best fit any dataset. Hence, the tuning is performed to find sets of hyperparameters to build a good model over specific data. In this work, I divide the tuning into two parts. One aimed to find a good architecture. For this step perform the tuning with a Grid Search over the number of neurons in each layer and the number of convolutional layers. Once the appropriate initial architecture is set, I try to further optimize the best model evaluating more learning rates for the Adam optimizer and probabilistic rates for the newly add drop out layer. For both phases I

use different subsets of the original training set, according to the computational resources and the time they require. The choice of evaluating no less than two convolutional layers comes from the expected performances and the computational resources available. The evaluation of too many models would become too costly in terms of time, and it is unluckily to obtain a better performance with a single convolutional layer, as it is more difficult to detect high-level features. Moreover, we know from literature that to improve the performance is better to add layers instead of neurons to each of them. Based on this last consideration, I also decide to explore a range of layers sizes with just three values (32, 64 and 128). Similarly, I decide to evaluate models always built with a single dense layer. Indeed, the performance could be optimized to avoid overfitting with dropout layers, rather than multiple fully connected layers. After an attempt with a training set containing 16000 elements, I notice that the time required to provide an output with these parameters is still too high. Therefore, I decide to reduce the training set size to sub-training-set of 4000 elements for the Grid Search process. From these results, I will consider the best performing hyperparameters.
Finally, the best model is obtained with 3 convolutional layers. For what concerns the number of neurons, a higher number even worsens the results and doubles the time needed to obtain it. It is around 60 seconds for 32 neurons and over 120 seconds for 64 or 128 neurons.

Results.
The best accuracy is obtained with 3 convolutional layers and layers' size 32. Once confirmed this architecture I want to further optimize the model. Hence, I add a dropout layer to lower the risk of overfitting. I set the drop out probabilistic rates to 0.5 and 0.7. Then I evaluate two different learning rates with the Adam optimizer [0.01 and 0.001], since in the previous Grid Search I only considered the default 0.01 learning rate. It comes out that the model with the best performance has a 0.5 drop out probabilistic rate and a 0.01 learning rate within the optimizer. Next, I create the model with the best hyperparameters found so far. I randomly set the number of epochs to 10. Once the model is fit, it is possible to evaluate the performances as the number of epochs changes.

The best number of epochs is 7, where the test and validation accuracies are closer and higher in the graph. Finally, I just train the best model again using the best number of epochs as well. Despite the computational time is still quite high, it is anyway a bit lower then using the random number of epochs.





In fact, a greater complexity of the network leads to a greater computational effort and above all to the risk of overfitting. Given the results obtained with the best model, which test accuracy is 99.80%, we can consider these final settings reasonably good.

Declaration

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.