Report challenge 1

Image recognition for ANN and Deep Learning

Francesca Forbicini
Vincenzo Greco
Nicolò Fontana

26 November 2022

1 Introduction

The aim of the challenge was to build and train a neural network to perform a classification task over a small and imbalanced dataset of low resolution images, belonging to 8 different classes.

2 Evolution of the network

2.1 First attempts

Our first attempts were based on applying transfer learning using VGG16 and VGG19 along with padding of the images as preprocessing. This led to a test F1-score of 0.6683.
After this we switched to EfficientNet due to a better theoretical guaranteed performance and we focused on improving the preprocessing.

2.2 EfficientNet

In particular we started with EfficientNetB7, but because of the padding we went back to the B0 version; indeed we were getting images mainly composed of black pixels due to the high ratio between the size of the required input (not smaller than 224) and the original size of our images (96 pixels). Using a network, B0, with smaller input size (224 pixel against 600 of B7) allowed us to achieve better results.
At this point, to avoid the limits imposed by the padding, we decided to drop it and resort to resizing. The only potentially relevant advantage of padding was to avoid any loss of information during the augmentation phase with zoom, rotation or translation (the original image, being just a small central portion, never overflowed the borders), but clearly it was not sufficient to counterbalance the drawbacks.
After getting rid of the padding, we focused on exploring the possible combinations of some hyperparameters such as the number of layers, the number of neurons per each layer and the dropout rate. This research was done mainly using EfficientNetV2B0 (the improved and more recent version of EfficientNetB0) and led us to an F1-score on the test set around 0.9.

2.3 Final model

The next step was to use a much larger network for the transfer learning, EfficientNetV2L, to which we applied everything we had learnt, reaching an interesting score of 0.92 on the test set.
The major difficulties at this stage were given by the scarcity of data and the imbalance in the dataset, which we already tried to face: respectively using data augmentation (with extremely positive outcomes) and class weighting (with terribly poor effects).

3 Discarded ideas

These are relevant approaches we tried because of a strong theoretical support, but that, for our case, revealed not so impactful on the score of our model.

3.1 Stratified K-folding

When still using B0, we tried to apply K-folding techniques to reduce the possibility of any eventual overfitting. After some handcrafted attempts we resorted to the model_selection.StratifiedKFold method from sklearn library.
Despite it giving us a strong tool against overfitting we had to discard this approach due to our limited computational resources; in particular the amount of total epochs (and the corresponding time) required to fully perform it would have drained all the compute units provided to us by Colab.
So we decided to use it performing just the first split only to divide the dataset into training and validation exploiting the embedded feature of stratification.

3.2 Class weighting

Another potentially useful technique we tried was the class balancing. This time we started using the built-in parameters and functions of Keras and then, after still achieving poor results, we decided to implement our version. Since also this one lead to no better score (even during the validation phase) we dropped it.

4 Worthy ideas

These ideas, in contrast with the previous ones, proved themselves very useful since the beginning of our research and allowed us to reach higher performances.

4.1 Data augmentation

One crucial step of preprocessing has been the implementation of layers dedicated to data augmentation. In particular, due to the strongly orientation-independent nature of the dataset and the distribution of relevant information over the whole image, it was possible for us to use many types of transformations: rotations, flipping and color corrections are just some of them.
This approach, combined with an elevated dropout rate, allowed us to train the network for high amounts of epochs reducing the risk of incurring into overfitting.
In addition we also added a GaussianNoise layer which we considered helpful for generalization.

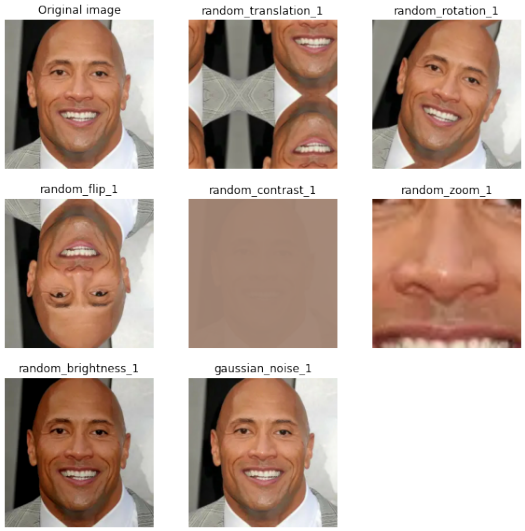

Figure 1: Data augmentation on a rock

4.2 Transfer learning and fine tuning

The main characteristic of our training was the application of transfer learning (as mentioned before) and fine tuning. In detail, an important aspect of fine tuning that we had to take in consideration was the care when unfreezing the source model layers.
Indeed, if done during the first phase of training, this would lead to a complete change in the weights of the base model (due to the extreme variability of the top network) making the whole point of transfer learning useless.
The consequence was to face an initial period of training with the whole convolutional transferred model frozen and a high learning rate for the optimizer (Adam was the choice as it is the state of art for our problem). After that would follow a second step, characterized by an extremely low learning rate (1e-5), dedicated to fine tuning, where also some of the layers of the convolutional network were unfrozen and their weights learned. Also, since the first version of our model, we used a Global Average Pooling layer between the base model and the fully connected part for a better position invariance of the network with respect to the input.

4.3 Early stopping

Another fundamental part of all our networks was the application of the early stopping method to avoid overfitting. This, applied both in the first and the second phase of training (obviously with different patience parameter, respectively 10 and 30) allowed us to cut many epochs of training, saving a lot of time and resources, while giving a better insights on the validation performances of our model.

5 Results and conclusions

5.1 Best model choices and scoring

In the end, the best model we managed to train was the one with EfficientNetV2L as base model and a top part made of 3 dense layers (respectively with 512, 256, 128 neurons) interspersed with 3 pairs of batch normalization and dropout layers (using a 0.4 dropout rate). As optimizer we used Adam with a learning rate of 1e-3 for the first phase (with the convolutional part frozen) and a much lower 1e-5 for the second part, when we unfroze all the layers of EfficientNet.
Regarding the activation function, the kernel initializer and monitored metrics we just used the default ones of Keras (ReLU, Glorot uniform and accuracy) because any change in them (we tried ELU, He uniform, ROC AUC and categorical accuracy) did not provide any improvement whatsoever.
This network reached a notable score of 0.92 in both phases of testing, proving itself a valuable solution for the commissioned task.

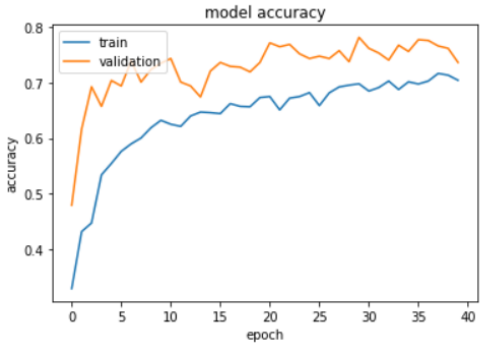

Figure 2: Validation and training accuracy during first transfer learning phase

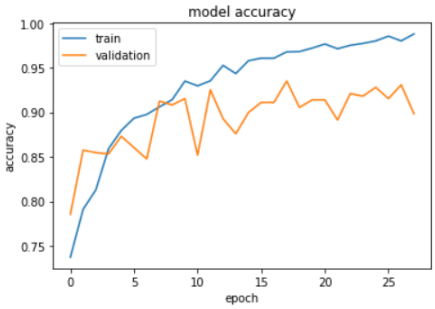

Figure 3: Validation and training accuracy during fine tuning

5.2 Final relevant reflections

Some interesting insights are other possible approaches we evaluated to tackle the faced problems, but that for a reason or another we had to drop.
In particular we considered to create many networks, each one focused on the classification of one class (vs the others) which then could have been ensembled to get better results (boosting approach) or, slightly differently, to create a hierarchical structure of networks with some of them specialized into distinguish between two (or more) bundle of classes, while the others trained to classify the classes inside each bundle (hierarchical approach).
Another possibility was to apply oversampling to the underrepresented classes to face the imbalance in the dataset (we immediately discarded the undersampling options due to the already small dimension of the dataset provided). As said, these options have been carefully pondered and abandoned mainly for time and resource constraints.