REPORT GRADIENTERS

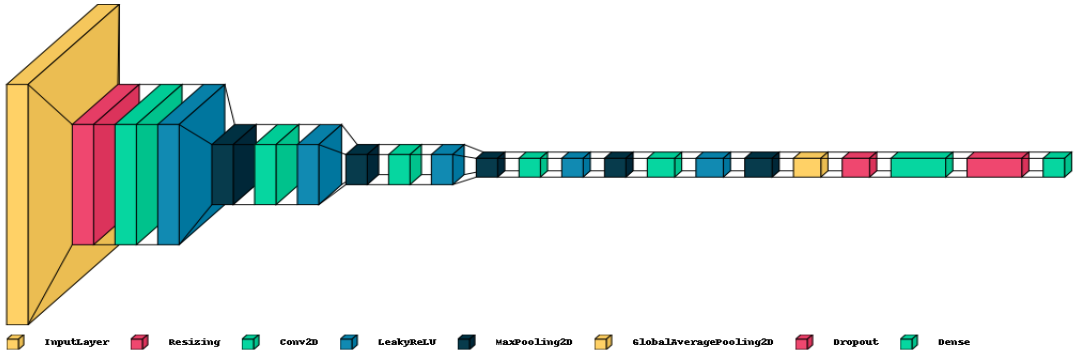
Matteo Braceschi, Alice Cariboni, Claudio Corradini

Our choices about the architecture of the models and the parameter optimization have been made through experimental trials, relying on architectures and layers shown during the lectures and suggestions found in specialised sites.

While implementing the architecture by trial and error, we developed Python scripts to improve our productivity and the performance of the models. In particular:

- A program that would build automatically and train sequentially several models, each one given in input as a list of layers with the corresponding parameters.
- A script to split the dataset into folders “train” and “validation”, keeping the same label structure.
- There have been attempts to implement Hyperas (a wrapper for convenient hyperparameter optimization) to improve our model. Unfortunately, this implementation would have required more development time.

Among our models, the one which obtained the best results (named FinalModel in the following) gave a score of 0,764150 when evaluated with the test set after the submission. The architecture is the following:



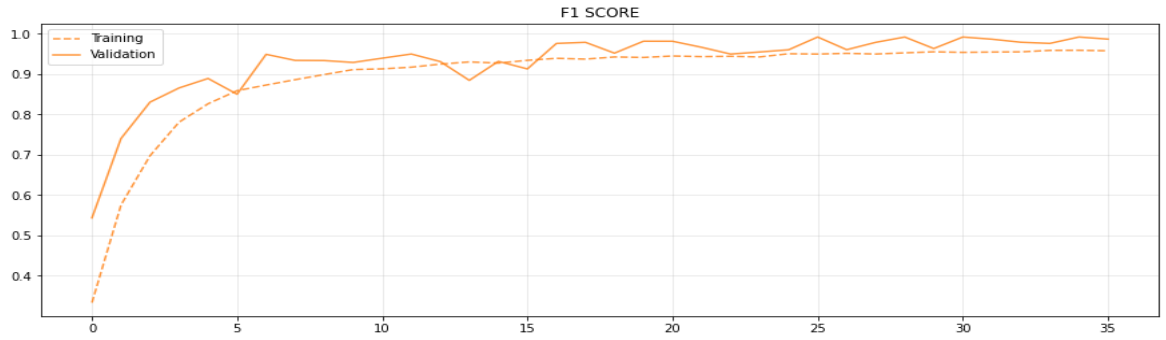
Relevant design choices have been:

- A normalisation of the values in input, dividing them by 255.
- The batch size that we used is 32, because as we observed and as we find on the web using a larger batch contributes a significant degradation in the quality of the model, as measured by its ability to generalise.
- choice of up to 50 epochs per training, to shorten the production time of a model
- Use of Data Augmentation in order to amplify our dataset and to avoid overfitting.
- An initial resize of the images to half of the original resolution, thus reducing the total number of parameters and speeding up the training, assuming that it wouldn’t change the performance significantly.
- A definition of a general convolutional block as composed by convolutional layer, activation layer and pooling layer. For feature extraction five instances of the convolutional block have been added, with an increasing number of convolutional filters going deeper into the architecture, as seen in the lectures.
- GlorotUniform as kernel initializer and LeakyReLU as activation function for the convolutional block, LeakyReLU instead of ReLU solve the problem of the dying neurons. According to specialised sites the Sigmoid should perform better when using GlorotUniform,

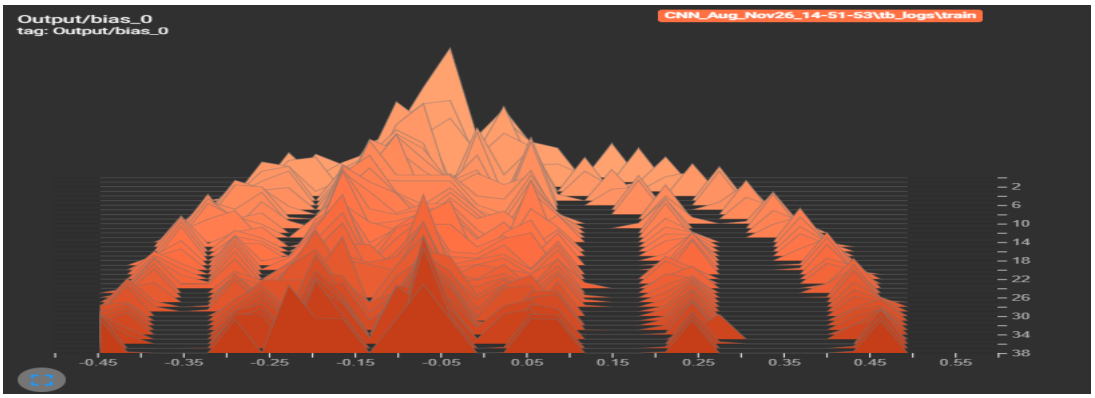
while LeakyRelu should be better with HeUniform kernel initializer. We tried both combinations without obtaining a higher score.

- We tried to use L2 regularization, or Weight decay, in order to add a term to the loss and try to decrease the weight values during the training process but we have more or less the same result in the valuation (0,7434). Another attempt is using the Batch Normalization with the purpose of speeding up the training and decreasing the importance of the initial weights. We use it with Momentum = 0.99 and after each Pooling layer but we don't notice better performance.
- To minimize overfitting, we applied GlobalAveragePooling instead of the flatten layer,in order to generate one feature map for each corresponding category of the classification task and take the average of each feature map.
- In the end, we used two Fully-Connected layers, the first with 512 neurons and the last one with 14 neurons and obviously with softmax as activation function.
- After GAP and FC layer we use the Dropout technique in order to reduce overfitting.
- As optimizers we tried to use Adam (with batch size =32) and SGD (with batch size = 1) and we demonstrated that Adam is better than SGD, probably because SGD is usually used with a big dataset.
- We used F1-score as a metric to evaluate the training, considering it as a more informative metric than accuracy in case of unbalanced datasets. We also use the F1-Score of the validation for Early Stopping.
- After that we have assured that our model avoided overfitting with a split between training set and validation set of 0.2, we re-trained the model with a split of 0.01. This improved our valuation after the submission by 0,1.

The following graph shows how training and validation converge, proving lack of overfitting:

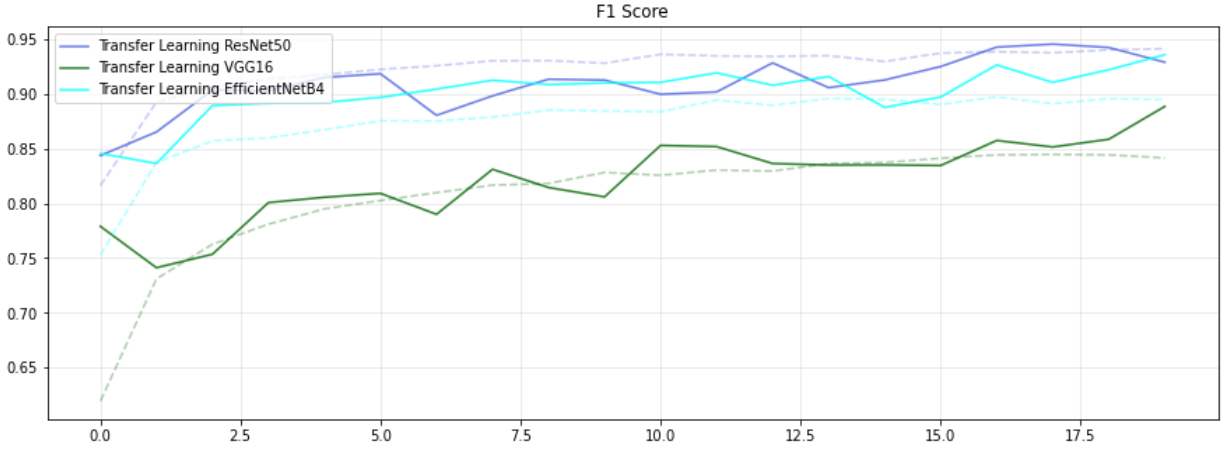


With Tensorboard, we can check the histograms in which we can see the changes of the weights distribution during all the epochs. A longer training, with more epochs would have shown whether our model suffers from problems like vanish gradient.

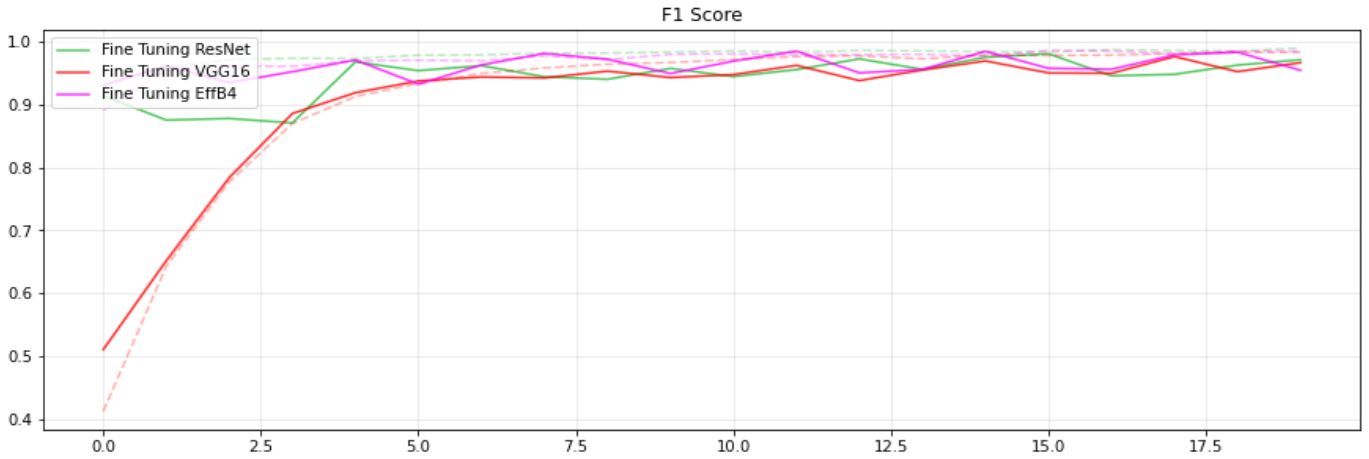


To compare our result with other architectures at the state of the art, we applied transfer learning in several architectures: Resnet50, VGG16 and EfficientNetB4. We trained only the fully connected layers to specialize on our classification task, keeping the weights of imagenet for feature extraction.

The graph below shows the training of the TL architectures. slightly better results have been made by ResNet.



Our last step was a comparison between our model and the TL architectures, applying fine tuning in them. We reloaded the previously trained models and we let only the last convolutional block be trainable (for example: in ResNet50 we freezed just the first 143 layers that are the first 4 convolutional blocks, keeping the last convolutional block to be trainable).



The graph below shows the training of the architectures with fine tuning:

The submission of these models has given the best results. Respectively:

- ResNet50: 0.8491
- EfficientNetB4: 0.8472
- VGG16: 0.8339
- FinalModel: 0,7641

In the end, even knowing that more improvements can be done, we are satisfied with our result, which requires less parameters than well-known architectures, with a relatively small decrease of the performances.