REPORT CHALLENGE1 – ARTIFICIAL NEURAL NETWORKS AND DEEP LEARNING

Group “Pinguino”: Pietro Caforio, Federico G. Ciliberto, Luca Civitavecchia

- 1. INTRODUCTION

This report serves as a comprehensive documentation of our journey through the challenge, aiming to present the most crucial steps undertaken, the challenges encountered, and the wealth of knowledge gained along the way.

The first five sections are not to be intended as a chronological explanation of the development of our model, but instead as a detailed analysis of the most important aspects of the developing, which instead are only briefly mentioned in section six. This last section is a chronological summary of the development process that led us to the final model.

- 2. DATA PREPROCESSING

Our first approach with the problem involved analyzing thoroughly the dataset. We manually analyzed each image, in order to get a more hands-on understanding of the data. We immediately noticed the presence of two repeated images, depicting a good-hearted ogre, and an amazing Russian baritone. Obviously, we proceeded removing all of them. We also normalized the images in order to have values in the 0-1 range. After doing this, the dataset accounted for around five thousand images.

We observed an imbalance within the dataset, and to address this issue, we employed first data augmentation techniques and then SMOTE [7] as class balancing methods. However, we found that our models did not show significant improvements as a result of these efforts. In particular, data augmentation techniques appeared to make the model learn wrong features, because some transformations were exclusively applied to the minority class, while SMOTE didn’t change our performance on the test set.

More on this can be found in the attached notebook “DatasetBalancing_FINAL.ipynb”.

In all of our models we used a data augmentation layer as input to the network. We tried to find the best transformations to use, using both empirical observations based on trial and errors, and also theory. We used rotations, zoom, horizontal and vertical flips, because they don’t mask or change the features important to understand if a leaf is healthy or unhealthy.

In our most advanced models, we also used some data preprocessing based on Cutmix, Mixup and RandAugmentation, offered by KerasCV . This was done as a preprocessing step, because our starting dataset was composed of roughly five thousand images, which we thought weren’t enough to effectively train deep models with many parameters. Also, we wanted to increase the ability of our model to generalize. By doing this, the size of our dataset increased a lot, reaching around 19 thousand images (we applied the augmentation techniques only after splitting the dataset. They were never applied to the validation set, to maintain it as similar as possible to the unkown test set). To successfully train a model with this many images, we used big, deep, complex models with many parameters (mainly ConvNextLarge and ConvNextExtraLarge). Our best results were obtained with the combination of these advanced augmentation techniques and complex models.

- 3. SELF-SUPERVISED LEARNING

We tried adopting some self-supervised learning techniques. The idea was to force in the convolutional part of our networks the extraction of features which were meaningful for our classification task. This was done by pre-training our networks on a pretext task, which used data automatically labelled (in different ways, depending on the specific technique adopted) starting from the original dataset [1]. Our first approach was mainly guided by the documentation we found online [2]. We applied 3 different rotations to the images, in order to obtain four times the size of the initial dataset. Images were now classified in 4 classes, based on the rotation applied (1 class was for the original version). We tried it on effnetb0, since we wanted a network which wasn’t too big, in order to train all the layers. The network learned very well the pretext task, however when we tried to train the network (with a new dense part) on the downstream task (the initial classification problem) the results were disappointing. We tried to reason on them, and we thought that the problem could be due to the fact that the features enforced in the network weren’t very meaningful for the downstream task. So, we reasoned on which features to enforce, and on which technique could be useful to enforce them. To

understand if a leaf is healthy or not, the network needs to focus on colours gradient, darker stains, and irregularities on the borders. Considering this, we decided to try a self-supervised learning with a jigsaw technique [3]. We cut every image in 4 patches. From each image, we built 24 new images by combining the 4 patches. Then these images were fed to the network, with an output layer composed of 24 neurons. The pretext task was to "solve" the puzzle by understanding the permutation applied on the original image. Also in this case, the network learned very well the pretext task. However, once again, we obtained low results in the downstream task. We thought this was due to the fact that we used a pretext task which was too easy for the network. In fact, in all the papers we found online, researchers were using 9 patches instead. However, this would have required much more computational power and time, which we didn't have, so we abandoned this approach.

More on this can be found in the attached notebook "SelfSupervised_Final.ipynb".

The images above display some of the samples used for the pretext task in the two self-supervised techiques analyzed (jigsaw at the left, rotations at the right)

- 4. HYPERPARAMETERS SEARCH

Our first approach to hyperparameters tuning was done by hand. We simply tried different variants of our models. This obviously required a considerable effort, exploring only a small portion of the possible values. So, we searched for more advanced tools, and we discovered keras tuner [6]. We applied it on a relatively small network, mobile net, to find the best number of dense layers. The results were encouraging, however the time required was excessive considering the time we had available, so we consider it only as a proof of concept of an effective hyperparameter search.

More on this can be found in the attached notebook "KerasTuner_final 2.ipynb".

- 5. ENSEMBLE MODELS

To achieve higher performances, we tried to exploit the main characteristics of all the models that gave us the best accuracy. In order to do so, we implemented majority voting between three of our best models so far, respectively, EfficientNetV2L, EfficientNetV2S trained on the unbalanced dataset, and EfficientNetV2S trained on the balanced dataset obtained using SMOTE (as discussed in section 2). We observed a slight improvement in the accuracy, but unfortunately it wasn't as impressive as we hoped. We thought this could be caused by the fact that simple majority voting was considering as equal all the models, without prioritizing the best ones, so we decided to try weighting the results of the models before using the function argmax to retrieve the prediction of the complete model. First, we used as weights the respective accuracies on the validation set achieved by the three models, normalized. Then we realized that by doing so, the weights were very close, since the models' performances were close as well. Therefore, we manually adjusted the weights a bit, to prioritize the best models even more. This style of model ensembling was more effective, and it is performed in our final solution as well.

- 6. MODELS DEVELOPMENT AND FINAL SOLUTION

Our first approach to the challenge was to try some handmade models. Since they did not provide great results, we decided to experiment with transfer learning and fine tuning on some pretrained models available in the keras applications framework. After some trial&error with lightweight models like Resnet and MobileNet, which didn't achieve great results, we tried working on EfficientNetV2L, getting 0.87 accuracy on test set, which was our best result up to then. Then we tried tweaking some hyperparameters using the smaller version EfficientNetV2S, to get a deeper understanding of what was affecting mostly the performances. We thought our model could benefit from a more balanced dataset, therefore we tried some

techniques to solve the unbalance, but as explained in section 2, they did not provide great improvement. Once this was done, we started exploring the advanced data augmentation techniques we discussed in section 2. Unfortunately, we noticed that most of the time the performances would decrease by applying complex data augmentation. At first, we hypothesized that we were doing the wrong augmentations, but after a while we concluded that most probably the models we've tried were not complex enough to learn from the augmented data (we were experiencing some sort of underfitting, in fact also the training accuracy was not very high). This led us to try heavier models like ConvNeXtLarge or ConvNeXtXLarge, obtaining some great results both on the validation and submission test hence suggesting good generalization capabilities.

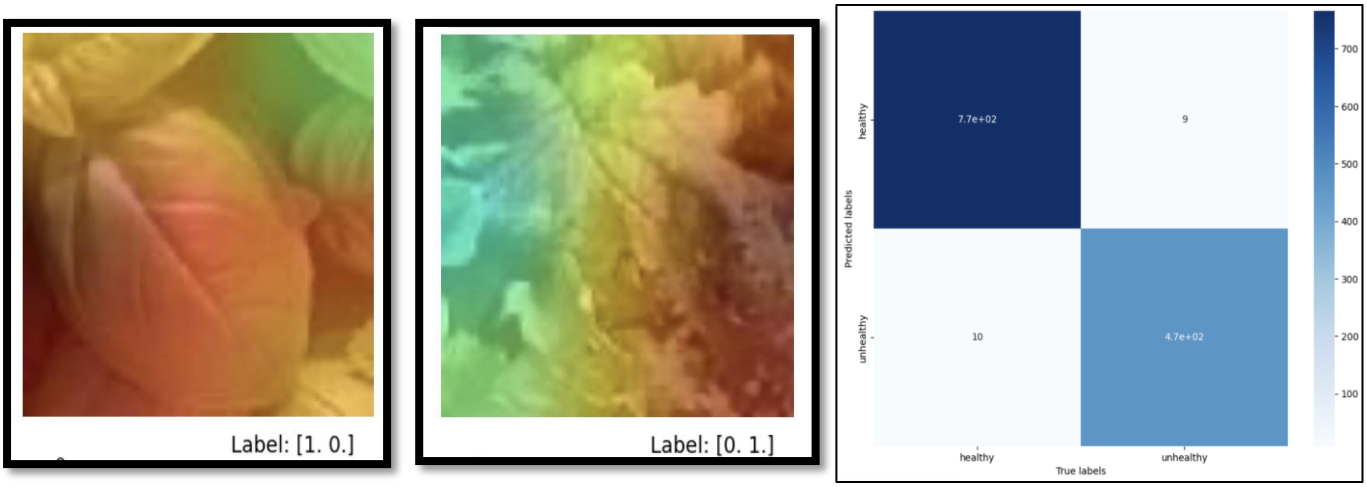
To boost the performances even more, we then ensambled the best models, exploring various possibilites as already analyzed in section 5.

Our final solution is composed of:

- ConvNextLarge, trained with KerasCV augmentation techniques (Cutmix, Mixup, RandAugmentation)
- ConvNextXLarge, trained with KerasCV augmentation techniques (Cutmix, Mixup, RandAugmentation)
- ConvNextLarge, trained with KerasCV augmentation techniques (Cutmix, Mixup, RandAugmentation, ColorDegeneration, FourierMix)

The detailed implementation and training of these three models can be found in the attached notebook "FinalModels.ipynb"

These three were ensembled using weighted average, with weights tuned on their respective performances on the validation set, and manually adjusted a bit to prioritize more the best models.



The images above display the gradCAM we obtained from one of our best models. In the healthy leaf, we can notice how our model is focused on the center of the main leaf. In the unhealthy leaf, the model is clearly focusing on the final part of the leaf, which is darker, rotten and with many holes. On the right of the gradCAMs we have the confusion matrix obtained from our final ensemble model.

REFERENCES

[1]Self-Supervised Representation Learning | Lil'Log (lilianweng.github.io)

[2]Implementing Image Rotation Pretext Training for Self Supervised Learning using Keras (deeplearninguniversity.com)

[3]1603.09246.pdf (arxiv.org)

[4]CutMix, MixUp, and RandAugment image augmentation with KerasCV

[5]How to Develop a Weighted Average Ensemble With Python - MachineLearningMastery.com

[6] KerasTuner

[7] https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.SMOTE.html

TEAM CONTRIBUTIONS

All the team members worked together throughout the challenge. In the following we just state some of the focuses of each member, but please note that we all equally contributed to the tasks and most of the times we worked side by side.

Pietro Caforio:

Self-supervised techniques research, Self-supervised learning with rotations, models research&implementation, augmentation techniques research.

Federico Giuseppe Ciliberto:

Dataset preprocessing and cleaning, data augmentation techniques, dataset balancing implementation, self-supervised jigsaw techniques, models research&implementation, hyperparameters tuning research.

Luca Civitavecchia:

Dataset balancing attempts (SMOTE), model ensembling research&implementation, hyperparameters tuning implementation, models research&implementation.