Artificial Neural Networks and Deep Learning: Homework 1

Teka Kimbi Ntimanputu (10673197), William Stucchi (10661711) and Lorenzo Veronese (10654901)
{teka.kimbi, william.stucchi, lorenzo1.veronese}@mail.polimi.it

I. Introduction

Automation in agriculture is crucial in current years, where famines, global warming and bad weather will cause the loss of more and more crops in the next few decades. A possible step in this direction is to automatically localize and tackle the diseases before they spread across the entire field. This would ensure that all the plants will grow healthy and strong, resulting in a successful harvest and decreasing losses and costs of curing plants.

A plant can be classified as healthy or unhealthy based on a picture of its leaves. This task is not trivial and requires the evaluation of an expert. Due to the lack of structure in the data and the absence of a detailed rule to assess the healthiness of a plant, this problem can be solved using a neural network model, in particular, since the task revolves around images, a convolutional neural network (CNN).

II. Dataset description

Due to the challenge format of the project, the dataset is split into three parts:

- local dataset, which can be analyzed and used to train the models;
- remote private dataset of phase 1, which is not accessible and is used to evaluate the first models;
- remote private dataset of phase 2, also not accessible, which can be used a very limited amount of times, hence it is used for a final evaluation of the model.

The local dataset is composed of 5200 $96 \times 96 \times 3$ RGB images of plants' leaves, each one labelled as healthy or unhealthy. The objective of this work is to train a CNN in order to perform binary classification on unseen images.

Inspecting the local dataset the presence of obvious outliers was noticed, which were images of Shrek and Trololo. In the preprocessing stage, they have been removed by finding the first occurrences of each one of them and then removing the others by simple comparison.

Another characteristic of this dataset is the unbalanced presence of healthy and unhealthy examples. This might deviate the learning of the model towards the majority class, leading to big quantities of false positives and false negatives. Various solutions to this problem are explained in Section III-A.

The local dataset has been split between the train and validation set, with a ratio of 80-20%.

III. Experiments

This Section describes the steps, trials and implementations that led to the final implementation that is shown in Section IV.

A. Preprocessing and data augmentation

The first problem that has been dealt with is the unbalanced condition of the dataset. In order to tackle this issue, three possible solutions were implemented on the train set:

- upsampling of the minority class: sampling without replacement of the unhealthy class up to when the two groups have almost the same length.
- downsampling of the majority class: randomly remove some images of the healthy class up to when the two groups have almost the same length.
- class weights: computed in such a way that during the training phase, the model pays more attention to the minority class examples with respect to the others.

The solution that has seemed the most successful is the class weight one. This is probably due to the disadvantages of the other two techniques: the upsampling of the minority class results in not independent and identically distributed samples due to the replication of some images, while the downsampling of the majority class leads to the loss of too many examples of the training set, hence to a higher risk of overfitting. Instead, the class weights work directly on the loss function without decreasing the quality of the training set.

To increase the generalization capabilities of the model, some techniques to artificially increase the number of samples were applied. In particular, the following were implemented:

- introduction of noisy pixels: some random pixels are randomly set to random values.

- random cut: some black squares are randomly inserted inside the image.
- blurring: some filters like the Gaussian one or the uniform one are convolved through the entire image.
- cutmix and mixup. Cutmix works by cutting some images are inserting them inside other images. Mixup, instead, creates new images as linear combinations of the other two belonging to the same class (this method can be generalized to consider different classes, but this was not applied).

The results are shown in Figure 1. Using these image augmentation techniques, which are described by Shorten et al. [1], it turned out that they led to oscillations and irregularities of the loss function during the training. This behaviour can be explained by observing that the transformations that have been listed above hide some useful details that are essential to classify the images correctly.

In addition to the preprocessing described above, some image augmentation techniques can be performed during the training phase, modifying for each batch the input images. The following were used: horizontal and vertical flip, rotation, zoom and translation. Brightness and contrast were not successful instead. The success of these techniques is probably due to the fact that no corruption is added to the data points, unlike the preprocessing described in the previous paragraph.

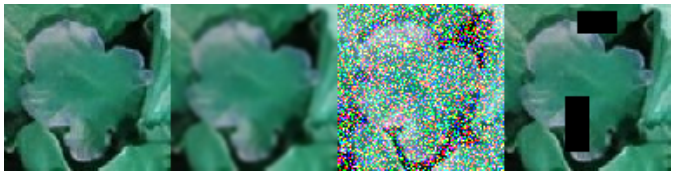


Fig. 1. From left to right: original image, blurred image, image with noise, cut image.

B. Hand-crafted models

A simple model has been chosen as a starting point, with the objective of making it more and more complex according to its performance on the validation set. The first architecture is a LeNet (LeCun et al. [2]), which is composed of two convolutional 5×5 filters interspersed by two 2×2 max-pooling layers followed by a flattening that prepares the input to the three dense layers.

Another model that has been tested, in parallel with the former, is composed of 5 convolutional layers with an increasing number of 3×3 filters, with ReLU activation function and a max-pooling layer: these last two are useful to add non-linearities to the model. A final global average pooling layer allows the transition from the feature extraction network to the fully connected network.

To enhance the complexity of the previous model, a new one was proposed. In this novel architecture, each convolutional layer was duplicated, effectively increasing the depth of feature extraction. Furthermore, the model was upgraded by incorporating two dense layers, consisting of 1024 neurons with ReLU activation functions. This addition aimed to capture and comprehend more intricate patterns within the images. To mitigate overfitting, dropout and data augmentation layers were implemented.

The previous three networks have shown a volatile pattern of training accuracy and validation accuracy, with high jumps in absolute value, due to the fact that they could not correctly detect the features of the images they were given. To overcome this issue, different values of the learning rate have been tested in order to allow the parameters of the models to better adapt to the data, but instead, they have shown poor capabilities even in these cases. Also, the training procedure revealed that while the training loss consistently decreases and stays low, the validation loss begins an upward trend. The latter suggests that the model has become overly specialized on the training data and losing the ability to generalize to unseen data. Due to these issues, attention has been shifted towards pre-trained models using transfer learning and fine-tuning in order to exploit the power offered by previous training on massive amounts of data.

C. Transfer learning and Fine-tuning

The lack of learning capabilities found working on hand-crafted architectures, has brought into focus on transfer learning and fine-tuning techniques. All the architectures analyzed here, are provided by the Keras API through Tensorflow (see [3]) and are all exploited using the imagenet-learnt parameters (since an extreme overfitting would have happened otherwise). Most of them were used with a final average pooling layer since it seems they worked better with respect to the max pooling layer.

The most important trained models are displayed in the table below, together with some of their hyperparameters: fine-tuning was done or not (column ”FT”), the learning rates and weight decay coefficients for transfer learning and fine-tuning (”l.r. TL/FT”, ”w.d. TL/FT”. Only the final ones are displayed), the additional fully connected layers (besides the output one), and the best validation loss and accuracy. The models utilized the categorical cross-entropy and were optimized with Adam. Note that the hyperparameters of worst architec-

tures were not fully optimized, since in those cases the attention was shifted towards more promising solutions. Furthermore, some models were fine tuned starting from the best epoch of the transfer learning process.

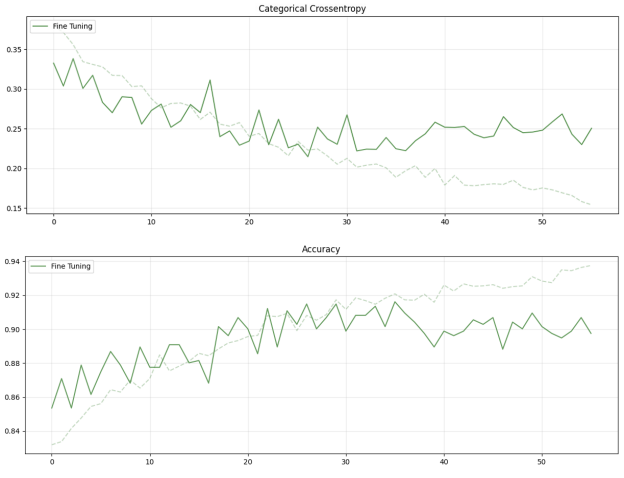
For a matter of space, some information is omitted from the table, such as the number of unfrozen layers during the fine-tuning phase (layers were sequentially unfrozen), the patience of the early stopping (used almost always in every model), the batch size (ranging from 16 to 128) and the type of upsampling done. A general result regarding the upsampling preprocessing described in Subsection III-A is that it was not increasing the model performances, hence most of the time it was skipped (a hypothesis of this is explained in the same subsection). Instead, the class weights seemed useful for the class unbalance, as well as the data augmentation performed during training.

Some experiments were done also in terms of weight initialization. In some cases, the dense layers seemed to benefit from the HeUniform, while the softmax output layer benefited from GlorotUniform initialization.

Also, ensemble techniques were tried, in particular, the average on the ConvNeXtLarge and EfficientNetB7, but it didn't lead to any interesting results, as well as AdaBoost applied on ConvNeXtBase.

IV. Final Model

All the paths described in Section III led to the result that the pre-trained architecture that best fits this context is the convNeXtBase. This model led to a "test" accuracy on the remote private dataset of phase 1 of 0.9000, precision of 0.9375, recall of 0.7895, and F1 of 0.8571, while on the dataset of phase 2 0.8420, 0.8227, 0.7447, and 0.7818 respectively. Its results and parameters are also shown in the table below. The model was trained using a batch size of 16. No preprocessing upsampling was applied, and since they surprisingly seemed to not increase the performances, class weights were discarded: this is a possible reason for the low recall. The data augmentation performed during the training comprised horizontal and vertical flip, rotation, zoom and translation. During the fine-tuning phase, the last 260 layers were unfrozen to allow comprehensive adjustment to the new task. Figure 2 shows the validation graphs during the fine-tuning of this architecture (transfer learning is omitted for a matter of space): thanks to the early stopping method overfitting was avoided.



(a) (b)

Fig. 2. Figure (a) shows the validation loss (solid line) and the training loss (dashed line) during the transfer learning of the ConvNeXtBase model. The accuracy is shown in Figure (b).

V. Conclusions

With this work, a model for the classification of plants between healthy and unhealthy based on their images has been proposed. The model shows great generalization capabilities, performing 0.8420 of accuracy in phase 2.

Transfer learning and fine-tuning have been found to provide the best performances, highlighting the generality of the feature extraction phase learnt from different images (imagenet) than the ones of the final application. The unbalanced dataset issue has been tackled using class weights while upsampling and downsampling methods were not successful. Image augmentation seemed also useful, thanks to its ability to allow the network to learn more general features and avoid overfitting.

Architecture	FT	l.r. TL	l.r. FT	w.d. TL	w.d. FT	Additional layers	Val. loss	Val. accuracy
ResNet50	n	0.001	none	0.05	none	Drop. + Dense 512	0.3612	0.8387
Xception	n	0.0001	none	0.05	none	Drop.+ Dense 64	0.4948	0.8472
MobileNetV2	y	0.0005	0.00001	none	none	none	0.48	0.7
EfficientNetB7	y	0.001	0.00001	none	none	Drop.	0.354	0.8735
DenseNet169	y	0.0001	0.0001	none	0.05	Dense 1024 + Dense 512 + Drop.	0.35	0.84
DenseNet201	y	0.0001	0.0001	none	0.05	Dense 1024 + Dense 512 + Drop.	0.33	0.85
ConvNeXtBase	y	0.001	0.00005	none	0.00005	none	0.2146	0.9148
ConvNeXtLarge	y	0.0001	0.00005	0.00001	0.00001	2 Dense 1024 + Drop. + Weight init.	0.2318	0.9040
ConvNeXtXLarge	y	0.0001	0.00001	0.00005	0.05	Dense 1024 + Dense 512	0.28	0.87

VI. Contributions

We worked on the project in a pretty parallel way, with each team member taking the initiative to try novel tracks and implementations. At the end of every iteration, each one shared its results, issues and doubts with the others in order to find together the best following steps. All members equally contributed to the work.

References

[1] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning. j big data," *Journal of Big Data*, 2019.

[2] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] Keras. Keras apllications. [Online]. Available: https://keras.io/api/applications/