Homework 1 – Image classification

The aim of this assignment is the classification of images of leaves according to the species of the plant to which they belong, using TensorFlow and Keras python packages.

1. Data Pre-Processing and Data Visualization

We were provided with a dataset consisting in a folder containing 17728 images already divided in sub-folders, based on their class. By means of python **split-folder** module we split the original folder into training and validation sets with stratified sampling, with a proportion of 0.85 and 0.15 respectively, which resulted the best choice for the majority of built models with respect to other trials with different percentages (0.8-0.2, 0.9-0.1, 0.7-0.3).

After importing libraries and fixing the seed for assuring reproducibility, we firstly exploited Keras **ImageDataGenerator** to read pictures in our source folders without the need of saving them in memory and, at the same time, avoiding filling the RAM, and to perform transformations.

Data that goes into neural networks should usually be normalized in some way to make it more amenable to be processed by the network. In our specific case, in the experiments without the use of transfer learning, we performed **scaling** technique in order to normalize pixel values to be in the [0, 1] range.

Moreover, we implemented **image augmentation** on training data (but not on validation set, since we wanted it to be as close as possible to the test set containing the "standard cases", without strangeness) which consists in applying different transformations to original images, leading to multiple transformed copies of the same image. Each copy, however, is different from the other in certain aspects depending on the augmentation techniques applied like shifting, rotating, zooming, restricting brightness range etc. This technique was very useful from different perspectives since it allowed us to make our models more robust and invariant against the considered transformations, giving them a higher capability to generalize, but also to increase the number of samples used for the training (which was very significative considering that for some leaves' classes there were few images, especially for the unhealthy ones). After observing that models trained with data augmentation performed significatively better, we focused the attention on trying different combination of transformations for augmentation.

The inspection of the labels revealed a strong unbalance between classes in the training set, as it is shown in Fig.1. Obviously, the same proportions are found in the validation set because of the stratification employed in the split.

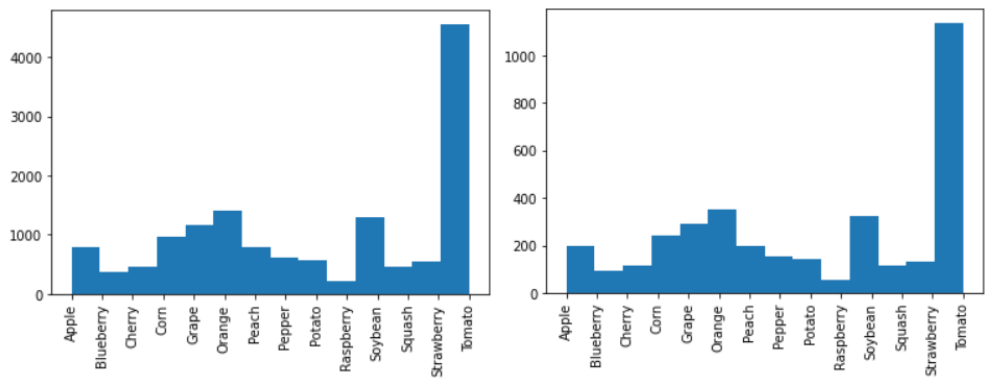

Fig 1: Classes distribution in training set (left) and validation set (right)

To deal with this problem, we considered different solutions, such as **oversampling** and using **penalized models**. Instead, we discarded undersampling due to the extremely reduced cardinality of the minority class

combined with the requirement for a sufficiently large amount of samples for the training of deep learning models. The best option was found to be the **assignment of different weights** to the different classes, in order to penalize more the errors in the prediction of samples belonging to the classes with lower cardinality, as it can be seen in the computed class weights:

```
[1.28230887 2.71681097 2.17344877 1.04961672 0.86832699 0.72448292
 1.29621343 1.65516484 1.76950188 4.80293367 0.78358131 2.20915224
 1.88086913 0.22233047]
```

Referring these weights to the histogram in Fig. 1, it is evident for example that the lowest weight is associated to Tomato class, which is the one with the highest cardinality.

2. Model implementation

2.1 Build and train the model from scratch

The following step consists in building the model architecture and tuning hyperparameters. Among the Convolutional Neural Networks tested, we have decided to report the one which achieved the best score on the test set in phase1 and that we have submitted also in phase2. The architecture is made of 3 convolutional layers with the nonlinear activation function 'ReLu', kernel size 3x3 corresponding to the width and height of the 2D convolution window, alternated with Dropout layer and pooling layer (Max Pooling 2D). After these 3 groups of layers there is a GlobalMaxPooling2D layer, and finally two Dense layers interleaved by a Dropout one to prevent overfitting.

Next step is model compiling: being a multiclass classification problem, we chose the categorical cross entropy as loss function, which shows the sum of all individual losses, and Adam as optimizer algorithm with learning rate 0.0001, since it's able to adjust the learning rate based on the history of the gradient to both speed up the learning and increase the performance at the same time. The metric we chose to evaluate the model's performance was accuracy, which is meaningful when the model is not biased toward the majority class, situation obtained with data weighting method described in the previous section.
Varying the hyperparameters of the built model, such as number of filters in convolutional layers, kernel size, kernel initializer, learning rate, number of units in dense layers, number of epochs for the training, activation functions and several other, allowed us to improve the performance of our model. Moreover, we explored combinations of different techniques with the aim of preventing and reducing overfitting, such as: **early stopping,** monitoring the accuracy on validation set (since mean accuracy on test set was known to be determinant for the final evaluation) and restoring the best weights; **l2 regularization** to avoid the creation of priority path and large weights trying different orders for the regularization factor; **drop out technique**, which we tried with values between 0.25 and 0.45 to force the model not to have favourite neurons, **scheduling the learning rate** using the LearningRateScheduler as additional callback.

2.1.1 Results

We obtained 0.624 as mean accuracy on the hidden test set in the phase2.

2.2 Transfer Learning

2.2.1

A technique we experienced to be more efficient for our specific application was Transfer Learning, which consists in transposing the experience acquired by a model on a previous task by freezing the weights of its layers responsible for feature extraction and attaching some other trainable layers aimed at performing the actual classification of interest. Among the supernets developed during previous ImageNet competitions and then stored as Keras models, we searched in literature which models achieved better performances in leaves classification, which brought us to try: Xception, VGG16, ResNet152, MobileNet, EfficientNet, DenseNet.

For each trial, we put a particular attention on applying the same **preprocessing** originally used for training the specific supernet and on trying different combinations of the additional layers attached to perform the classification, varying the kind of layers (Flatten, GlobalAveragePooling, GlobalMaxPooling), the hyperparameters, and the techniques to reduce overfitting already listed in section 2.1.

2.2.2 Fine tuning

We applied fine tuning technique to the most promising models built with transfer learning in order to better adapt them to our specific task. Starting from the models described in the previous section, we have done several trials by varying, as well as hyperparameters described in previous sections, also the number of unfreezed layers of the supernet.

In general, since we desired small changes in the pretrained layers, when we applied fine tuning, we chose lower values of the learning rate with respect to the previous learning phase.

2.2.3 Results

Submission Name	Train-Validation %	Supernet	Fine Tuning	Peculiarity	Accuracy phase2
Submission18	0.15-0.85	EfficientNet	Yes		0,947
Submission15	0.15-0.85	MobileNet	Yes		0,905
Submission9	0.15-0.85	MobileNet	Yes	GaussianNoise layer	0,89
Submission13	0.30-0.70	EfficientNet	No		0,896
Submission12_DENSE_ft	0.20-0.80	DenseNet	Yes		0,894
Submission9Dense	0.20-0.80	DenseNet	No		0,86

Nb. The results reported in the table are relative to the performances achieved by each submitted model (employing transfer learning technique) on the test set of phase2.

In the end, we chose as best model the one based on the EfficientNet with fine tuning since the accuracy on the test set is the highest. Moreover, it doesn't present overfitting because the score gained in the train set equal to 0.9898 is similar to the one in the test set, as a proof that the algorithm generalizes well on new data.

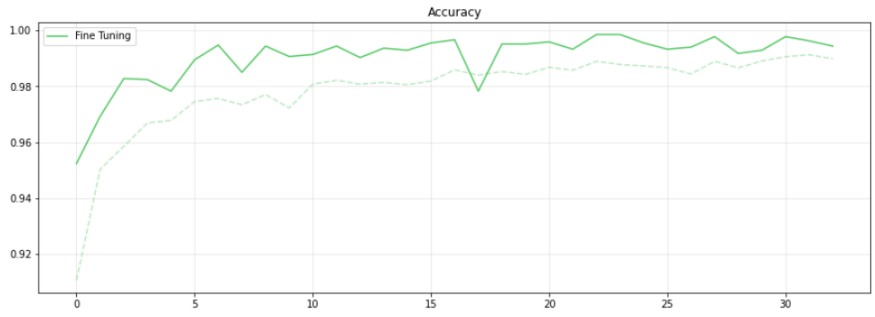

Fig2: Plot of the accuracy on train and validation set of the best submitted model.