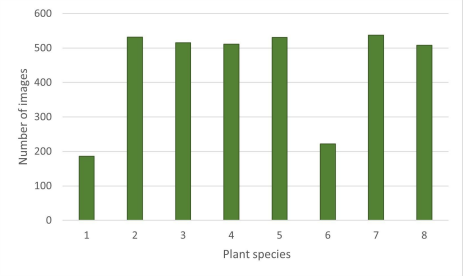
AN2DL - HOMEWORK 1

ARTIFICIAL NEURAL NOOBS: Luca Fornasari, Nicolò Fasulo, Tommaso Aiello

About the challenge

We were given a set of 3542 images of plants of size 96x96, representing 8 different species. More specifically, the dataset was **not balanced**, as it is shown by the histogram on the left.
Our goal was to implement a classifier through a Neural Network in order to achieve the best possible accuracy, paying attention to not overfit.



Data Preparation

Our first data manipulation has been the split of the dataset into a training dataset (80%) and a validation dataset (20%). We did it using the "***split-folders***" library and its function "***splitfolders.ratio***" that gave as output two folders containing training and validation data. Also as mentioned before we had an imbalance dataset so in order to solve this problem we tried at first to use oversampling but then we decided to use in each model the 'class_weight' parameter of the function 'fit' to make the model pay more attention to the samples from less represented classes because it gave us better performances.

Workflow and first approach

We started by using the **CNN model** seen in the Exercise Session 3, which was composed of 3 Convolutional blocks followed by a MaxPooling2D block, 1 Classifier layer and 1 Output layer. We changed and tweaked this model by adding additional layers, trying to improve it, but we couldn't get a validation above 0.55.
This pushed us to try with pre-trained models, using **Transfer Learning** and **Fine Tuning**.

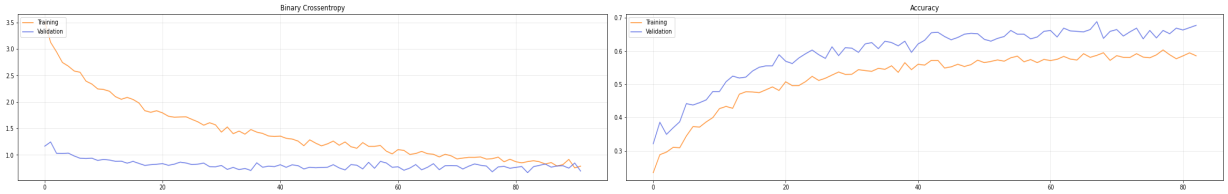
Ensemble Model

Before diving into the pre-trained models we implemented, it is important to clarify that we decided to use an **ensemble model** in order to take advantage of each model's best features. After training ResNET-50v2 and Xception models we couldn't exceed 0.8 of accuracy on the test set, so we first put those together having an increase in performances then we also added EfficientNetB1, EfficientNetB2 and VGG19. We weighted the contribution of each model to the final prediction according to its performances.

ResNET-50v2

Switching to pre-trained models made us think to perform some data augmentation in order to achieve better validation performances, so we implemented the function "ImageDataGenerator" from Keras.preprocessing library. This is the augmentation used in the ResNET Transfer Learning model: (rotation, height_shift, width_shift, zoom, brightness,

shear, horizontal and vertical flip, fill_mode=reflect) and finally we used the preprocessing function provided by the ResNET network.
The model we used contains a Resizing layer (224,224,3) to fit the images in the ResNet. Following the ResNet, which has 23 millions parameters and 189 layers, we added a Flattening layer, 3 Dense layers (with a L2 regularization penalty) and with 2 Dropout(0.15 rate) layers in between to add regularization.
At first we trained only the fully connected part, freezing the weights of the resnet50 pre-trained on 'imagenet'. We used Categorical Crossentropy as loss function and Adam as optimizer, and we trained for 100 epochs with EarlyStopping (patience=20).
Then we performed the fine tuning, freezing only the first 150 layers and these are the results obtained after fine-tuning.



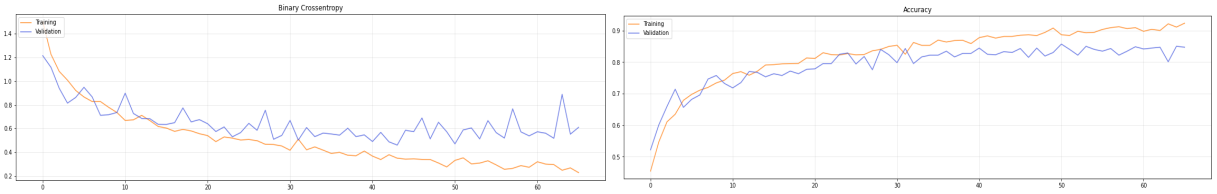
Xception

For the Xception model we used a similar data augmentation to the one used for Resnet50V2 but we augmented the brightness and shear range a bit. The input size of this model is (150,150,3) so we introduced a Resizing layer as input of the Xception, which is a network with 20 millions of parameters and 131 layers. Then we added a Flattening layer and 2 Dense layer with also 2 Dropout layers (0.25 rate) to regularize the model. We first froze all the weights of Xception, pre-trained on 'imagenet', and trained the model for 63 epochs. Then we kept frozen only the first 60 layers and did the fine-tuning, training the model for 120 epochs with EarlyStopping (25) using the same loss function, optimizer and learning rate used for ResNet50V2. These are the results obtained after fine-tuning:

VGG19

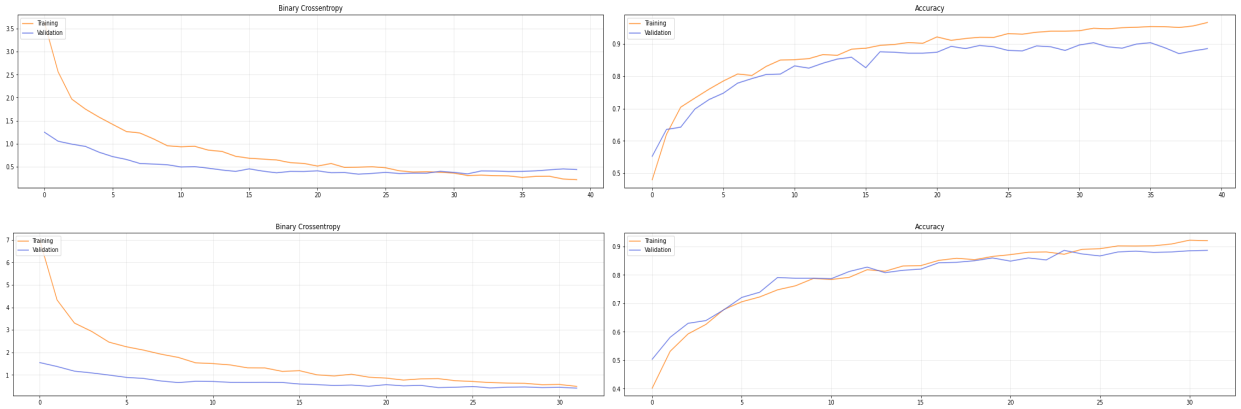
For the VGG19 model, we changed the data augmentation step; in particular, we noticed that VGG19 didn't handle rotations and zoom well, hence we removed them and increased the height and width shift range.
This model has more than 20 millions parameters and it's composed of 21 blocks. The input size is (144,144,3) and, as already said, a resizing layer was needed.
On top of VGG19, we also added a GlobalAveragePooling2D, a dropout layer, two dense layers (with 1024 and 512 units, respectively), a Leaky ReLU and a dense output layer.

The first phase of training (with all VGG19 layers frozen) led to a validation accuracy of 0.71, after 100 epochs and a mobile learning rate (we exploited the “ReduceLROnPlateau” function). After that, we proceeded with the fine tuning phase: we kept frozen only the first 13 layers and in 51 epochs of training the model reached a validation accuracy of 0.86.



EfficientNetB1 and EfficientNetB2

These two pre-trained models are the ones that gave us better results. They have way less parameters than the others but still very performing. They have respectively 6.5 and 7.7 millions parameters. The input size is respectively (240,240) and (260,260) and this time we didn’t add a Resizing layer but we did the resizing directly in the data augmentation step, similar to the resnet50 one but with no preprocessing needed.
Following these two models we simply added a GlobalAveragePooling2D, a Dropout layer with a high rate of 0.5 for B1 and 0.55 for B2 to prevent overfitting, and then an Output layer. We first froze all the weights of the pretrained models and then we performed the fine tuning. This time since the parameters were actually way less with respect to the other models we decided to train all the layers of EfficientNetB1 and to freeze only the first 40 layers of EfficientNetB2. We still used Categorical Cross Entropy as loss function and Adam as optimizer putting the learning rate to 1e-4 and trained for 50 epochs with Early Stopping (patience = 8). These are the results obtained (top:B1, bottom:B2):



Conclusion

After training these models, we changed the model.py file to ensemble them. We computed the prediction for each model and sum the result giving the following weights: EfficientNetB1 (33,33%), EfficientNetB2 (16,67%) , ResNet50V2 (16,67%), Xception (16,67%), VGG19 (16,67%). In the end we got 0.8854 accuracy on the test set.