Artificial Neural Networks & Deep Learning

Politecnico di Milano

Challenge 1

Valeria Panté (10712755), Luca Tombesi (10865393), Adriana Vella (10864553)

The assigned task is the multi-class classification of images from different species of plants. The paper aims to analyze the dataset, the image pre-processing, the choices and the steps made for the development of the classification model.

1. Dataset

The dataset assigned is composed of 3542 images belonging to 8 different classes, labeled as Species1, Species2, Species3, Species4, Species5, Species6, Species7 and Species8. The size of each input image is 96x96. All the images are not equally distributed among all the classes, in fact the number of samples for each label is the following:

- 186 for Species1;
- 532 for Species2;
- 515 for Species3;
- 511 for Species4;
- 531 for Species5;
- 222 for Species6;
- 537 for Species7;
- 508 for Species8.

To avoid overfitting during the training of the model, we chose to split the dataset in two subsets: training and validation.
Based on the previous distribution, we noticed that images from Species1 and Species6 were in large minority with respect to the other classes, so we decided to keep all the images of Species1 (the one with the smallest number of samples) in the training set, while for Species 6 we decided to keep a small fraction of samples (10%) in the validation set. This operation was done in order to increase the accuracy of the classification of the mentioned classes.
For all the other classes the images were splitted in the following way: 80% for the training set and 20% for the validation set.

2. Preprocessing

The first step we considered for the image preprocessing was the Data Augmentation, which involved:

- ➔ rescale, with a factor of 1/255, in order to normalize the pixels of the images;
- ➔ rotation of 15°, horizontal and vertical flip, to avoid the dependence of the network on the position of the found features.

Initially, we also considered the following augmentations, which then were discarded:

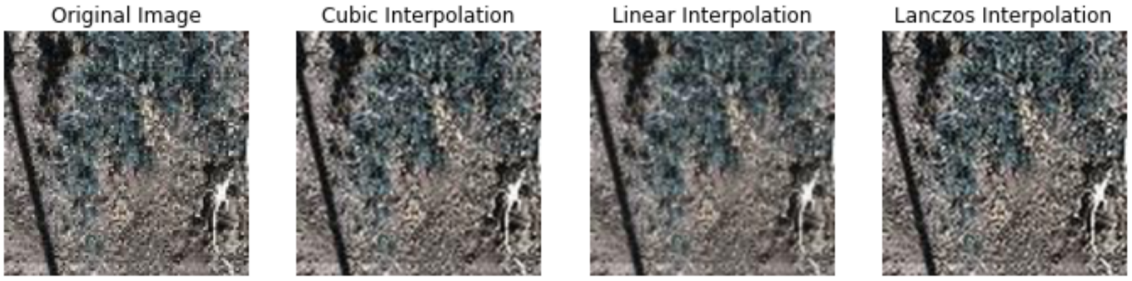
- ➔ zoom, removed due to the low resolution of the images;
- ➔ brightness: first we thought of applying a random brightness to balance the brightness and the darkness of the images, but this operation didn't improve the performance of the network.

In order to enable the network to more easily learn the features from the images, in the last models we added two levels of preprocessing at the beginning of the architecture:

- `tensorflow.keras.layers.Resizing(256,256,interpolation="lanczos5")`

  The Resizing layer is used for increasing the size of the image. As interpolation, we chose the `"lanczos5"` one because it maintains a good quality resolution and gives also an higher

contrast to the image, useful for detecting its edges. This decision was made after analyzing different interpolation methods, applying them to sample images:



The dimension of the output image is functional to the final model we developed.

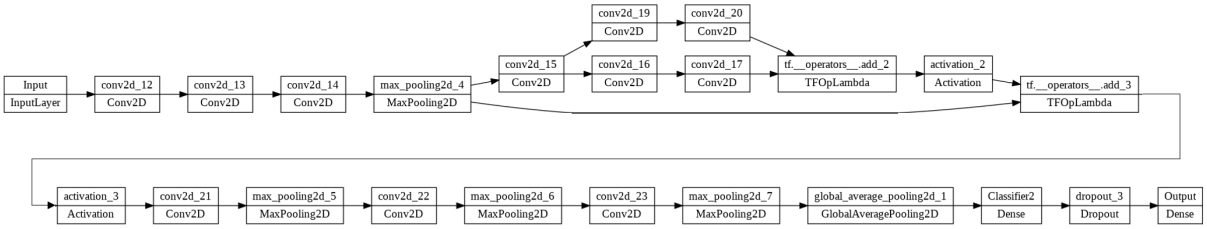
- `tensorflow.keras.layers.RandomContrast(0.3)`

  The RandomContrast layer aims at randomly enhancing the contrast of the image with a maximum factor of 0.3 in order to highlight the edges and facilitate the network in finding the features.

The rescale, resize and random contrast preprocessing steps are applied both in the training phase of the model and in the prediction one, when the model is deployed.

3. Model development

We started by developing our custom CNN from scratch.
Initially it was composed of 6 convolutional layers and 3 fully connected layers. Then, after some modeling, we ended up with the following configuration:



In this configuration, besides the serialization of different convolutional layers, we introduced a skip connection and some convolutions performed in parallel and then added with each other to simulate the ResNeXt configuration seen in class. After each sum of such layers, we put an Activation Layer with ReLU as activation function to avoid vanishing gradient. Furthermore, at the end of the convolutional layers, we put a GAP layer in order to make our network more robust to shifts of the features in the input images. Finally, we kept 2 fully connected layers: the first one with 128 neurons, ReLU activation function, *HeUniform* initialization of the weights and a Dropout of 0.5 to avoid overfitting; the second one with 8 neurons, SoftMax activation function and *GlorotUniform* initialization of the weights.
This model gave us an accuracy on our test set of 0.75.

After these attempts, we opted to use the Transfer Learning and Fine Tuning techniques.
First of all we tried with the VGG16 network as supernet, adding at the end a GAP layer, which already improved our performance in the previous architecture, and 3 fully connected layers, where the first two are made respectively of 1024 and 256 units, both sharing ReLU activation function, *HeUniform* initialization of the weights and a Dropout of 0.5 to avoid overfitting; the third one is made of 8 units, SoftMax activation function and *GlorotUniform* initialization of the weights.
As previously mentioned, before the VGG16 layer, the preprocessing layers were added.

After performing the Transfer Learning with the “imagenet” weight initialization, we fine tuned by freezing the first 6 layers and trained the rest of the net. This decision was made after some attempts, as we noticed it had the best performance, which had an accuracy of 0.91 on our test set and 0.82 on the CodaLab test.

The last improvement was made thanks to the change of the supernet from VGG16 to DenseNet.
After Transfer Learning, we performed Fine Tuning with the same number of layers frozen, because we thought that our input dataset is different from the “imagenet” one, so the weights needed to be retrained, but we kept only the first six layers untrainable as they capture the high level features of the image. The accuracy reached by this architecture on our test set was 0.92 while the one on the CodaLab test was 0.89.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing (Resizing)         (None, 256, 256, None)    0

 random_contrast (RandomCont  (None, 256, 256, None)   0
 rast)

 densenet169 (Functional)    (None, 8, 8, 1664)        12642880

 global_average_pooling2d (G  (None, 1664)             0
 lobalAveragePooling2D)

 dense (Dense)               (None, 1024)              1704960

 dropout (Dropout)           (None, 1024)              0

 dense_1 (Dense)             (None, 256)               262400

 dropout_1 (Dropout)         (None, 256)               0

 dense_2 (Dense)             (None, 8)                 2056

=================================================================
Total params: 14,612,296
Trainable params: 14,444,360
Non-trainable params: 167,936
_________________________________________________________________
```

Besides the choices made on the architecture of the network, we also performed hyperparameter tuning by making several attempts in changing the number of fully connected layers, the number of neurons per layer, the amount of dropout. Thanks to these attempts, we discovered that a number of FC layers larger than two did not improve the network performance, thus we kept 2 FC layers to reduce the number of parameters needed. Furthermore, a larger number of neurons in the first fully connected layer increased the accuracy, so we opted for 1024 units. For what concerns the dropout, we tried several values in the range from 0.3 to 0.7 to still avoid overfitting without doing it too much, but the parameter that maximized our performance was 0.5.

As optimizer we chose the Adam one, first with a learning rate of 10^{-4} but it got stuck in local optima, thus we changed it to 10^{-5} getting much better results. Lower learning rates did not prove to find solutions that increased the performances.