AN2DL Homework 1 – Recognizing leaves

Guadagno Antonio – 10561018

Emanuele Claudio Alfredo – 10682738

➢ Pre-Processing

First, we have reorganized the dataset that has been provided to us (see attached notebook "*Dataset_pre_processing"*). We have created 2 new folders, "Training" and "Validation", each one containing 14 sub-folders, one for each class that we have to recognize. We have used the 80% of data for the training and the 20% of data for the validation. Of course, the division of images between training and validation set has been made in a **random** way (using appropriate functions and not by hand) and class by class (in a **stratified** way).

By analyzing the statistics regarding the training dataset, we have noticed the problem of **class imbalance** (for instance, in the training set we have 4611 pictures labeled as "Tomato" and only 378 pictures labeled as "Blueberry"), so we have computed the class weights in order to use them during the training of all the models that we have developed.

Considering class i, to compute its weight, we have used this formula:

$$Weight(i) = \frac{1}{num\_of\_photo\_for\_class\_i} * \frac{tot\_num\_of\_photo}{14}$$

➢ Model with no augmentation and no transfer learning

We started by building a very simple model without data augmentation. The network was built in this way:

$$Input\ layer\ +\ 5\ (\ Convolution\ +\ Max\ Pooling\ )\ +\ Flattening\ +\ 2\ (\ Dense\ Layer\ )\ +\ Output\ layer$$

To limit overfitting, we have used early stopping and dropout layers. We have obtained an accuracy of 29,43% on the test set so we have decided to add data augmentation in order to improve performances.

➢ Model with augmentation and no transfer learning

The model used is the same as before (in order to do a comparison). We used these settings for the augmentation:

```
train_data_gen_data_aug = ImageDataGenerator(rotation_range=5,
                                             height_shift_range=15,
                                             width_shift_range=15,
                                             zoom_range=0.1, # Useful to capture the veins (zoom in) and the edges (zoom out) of the leaf
                                             horizontal_flip=True,
                                             vertical_flip=True, # To recognize leaves when they're upside down
                                             fill_mode='constant', # To avoid multiple leaves in the same image
                                             rescale=1/255.) # rescale value is multiplied to the image
```

Using this model, we have obtained a 50,94% of accuracy on the test set. We have decided to improve our model by the usage of transfer learning (ResNet50).

➢ Transfer Learning and Fine Tuning

We have tried VGG16 and ResNet50, both with and without fine tuning. Our final model uses ResNet50 and fine tuning.

Also in this case, we have used class weights (computed in the same way we showed before). We have done various A/B tests to tune the parameters of the data augmentation, then we have used the ones that have maximized our performances on the validation set:

```
train_data_gen_data_aug = ImageDataGenerator(rotation_range=30,
                                             height_shift_range=50,
                                             width_shift_range=50,
                                             zoom_range=0.3, # To better recognize both veins (zoom in) and edges (zoom out) of the leaves
                                             horizontal_flip=True,
                                             vertical_flip=True, # To recognize leaves when they're upside down
                                             fill_mode='constant') # To avoid multiple leaves in the same image
```

The model we have built presents:

- Input layer
- **Resizing layer**: we wanted images 224x224 (height and width of images used by ResNet50)
- **Gaussian Noise layer**: to add some noise and so to generalize better
- **Preprocessing**: to apply the preprocess_input function of ResNet50 to the images
- **ResNet50** (supernet)
- **Max Pooling layer**: to reduce the number of parameters to be trained and to generalize better
- Flattening layer + Dropout (0.5)
- Dense (128 neurons) + Dropout (0.3)
- Dense (64 neurons) + Dropout (0.3)
- Output layer

We have divided the **training into 2 steps**:

- **First step**: 20 epochs in which all the layers of ResNet50 are set to “trainable = False”. In this way we train a little bit the dense layers dedicated to classification (the ones after ResNet50) before the fine tuning. In this phase we use a larger learning rate (1e-3).
- **Second step**: 100 epochs in which the last 17 layers of ResNet50 are set to "trainable = True". In this way we fine tune ResNet50 together with our dense layers. In this phase we use a smaller learning rate (1e-4), precisely for the fine tuning.

By splitting the training into 2 steps, we can fine tune the last layers of ResNet50 having our dense layers trained a little bit.

To avoid overfitting, we have used early stopping with a patience of 20.

With this model we have a 92.08% of accuracy on the test set.

(See attached notebook “*RN50_MAX_POOL_FINE”*)

Once we have checked that everything was working correctly, we have trained our final model using all the data present in the dataset that has been provided to us. Validation set and early stopping were not needed anymore. Seen that Kaggle allows us only runs that lasts less than 9 hours, we have split the whole training in 2 parts, reloading the partial model we have obtained after the first step of training.

With this model we have obtained a 93.22% of accuracy on the test set of the first part of the competition. In the final phase of the challenge, we have an accuracy of 91.32% on the test set.

(See attached notebooks *“RN50_MAX_POOL_MORE_FINE_pt1”* and *“RN50_MAX_POOL_MORE_FINE_pt2”*)

The 91.51% that you can see in the finale ranking has been obtained using adaptive learning, but this is an argument that we have seen with image segmentation, so we have decided to not present that model, just to try it for our personal interest.

➢ Other models tried and discarded

Before arriving to our final model, we have tried many different combinations. Among the others, we have tried a model which presented 3 dense layers after the flattening:

- Dense (512 neurons) + Dropout (0.3)
- Dense (256 neurons) + Dropout (0.3)
- Dense (128 neurons) + Dropout (0.3)

But the number of parameters was too high (because of the 512 neurons of the first dense layers) and so we were not able to generalize in a good way.

Another thing that we have tried is fine tuning in 1 step (setting the last 17 layers of ResNet50 to “trainable = True” since from the beginning, without first training a little bit our dense layers) but performances were worst.

We have also tried transfer learning without fine tuning, but of course with fine tuning performances were better.