Artificial neural network and deep learning - First Challenge 2021

Leaf Classification

Marco Domenico Buttiglione, Luca De Martini, and Giulia Forasassi

Politecnico di Milano

November 30, 2021

1 Introduction

Image classification is one of the most common Machine Learning problems. Given a fixed set of classes, image classification is the task of predicting which label to assign to an input image. In this case, the purpose is to classify leaves images in order to predict their species using Deep Neural Networks.

In this project we have analyzed the data and experimented different techniques for training the network, including custom networks and transfer learning technique with fine tuning.

2 Data analysis

The training dataset that was provided consists in a set of 17728 pictures of leaves laying flat on a black background. They are organized in 14 directories, one for each plant species that is being analysed (apple, blueberry, cherry, corn, grape, orange, peach, pepper, potato, raspberry, soybean, squash, strawberry, tomato). The images are `256x256` resolution JPEGs (Figure 1) and the distribution of samples in the dataset is unbalanced, since there is a different number of samples for each of the classes. (Figure 3)

Figure 1: Example of a leaf image

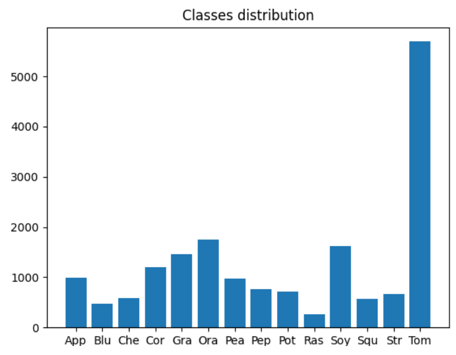

Figure 2: Classes distribution

3 Data preparation

3.1 Data splitting

We started by splitting the dataset into training, validation and test set using two approaches. For the first one, we divided the entire dataset into the three sets for training, validation and testing with a fixed ratio (80,10,10 or 64,16,20). As for the second one we took a fixed number of samples from each class of the dataset (50 or 70), keeping the rest for training. In both cases, the splitting is done taking random elements.

To overcome the class imbalance problem we used two alternatives methods: oversampling and class weighting. With oversampling, random images in the minority classes are duplicated until each class is the same size. On the other side, class weights allows to give different importance to predictions errors on a per class basis. By increasing the weight of a class the lower the number of its samples, the class imbalance problem is mitigated.

3.2 Data augmentation

To prevent overfitting and give more robustness to the network, we have considered appropriate to effectively increase the number of training samples provided by introducing data augmentation. By applying random transformations, for which the classification should be unaltered, new versions of the same image are created. We used rotation, zoom, vertical and horizontal flipping, translations and shears filling the empty space with 0 since the background of all images in the dataset is black.

We also tried to add a Gaussian noise to the background of the image, but we noticed a significant increase in training difficulty, without improvements in network performance, for this reason the relative experiments are not reported.

The data augmentation was performed initially with Keras `ImageDataGenerator`, later on, we switched to the TensorFlow `tf.data.Dataset` API. With this change we were able to optimize the preprocessing pipeline: using Keras layer for transformations and introducing prefetching and caching, we mitigated the storage bottleneck on cloud platforms.

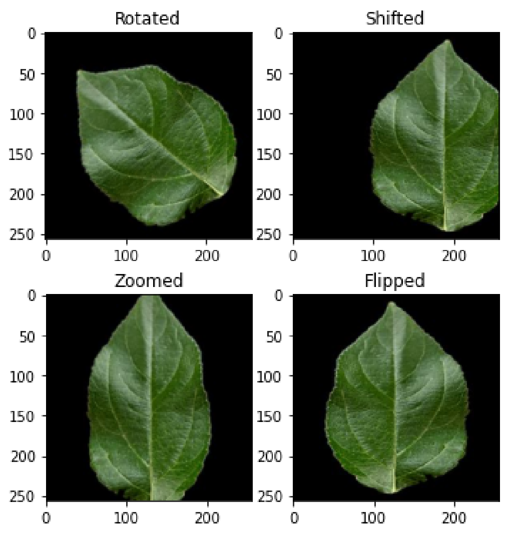

Figure 3: Augmented images

4 Models and techniques

The model selection and construction followed three main approaches, transfer learning using some known high performance pretrained networks for image classification, custom deep convolutional neural networks trained on the image classification task and some experiments with CNNs starting from an autoencoder in order to train the feature extraction part of the net.

4.1 Transfer learning

To establish a baseline performance, the first chosen method was transfer learning. By using a pretrained network, we were able to reach a viable model with sufficient performance in a short time.

First we tested by downsampling the images to `64x64` using VGG16 as network bottom and a simple classifier top, the training was fast, but there was room for improvements in accuracy. Working directly on the `256x256` and moving on to bigger networks yielded significant accuracy gains at the cost of training time. For classifier top, we hypothesised that recognition would be done mostly through surface patterns rather than the position of features, so, for the first experiments, we chose to use Global Average Pooling layers to flatten the image to generalize over the position of the features and this was positively reflected in the results. After obtaining satisfying results transfer learning with `Resnet152v2` we moved on to completely custom networks to see what we could achieve.

4.2 Custom networks

We tried using deep convolutional neural networks with `3x3` convolution filters, maxpooling, ReLU (normal and leaky) using dropout on the dense layers, then taking inspiration from ResNet and U-Net we tried adding skip connections by concatenating or adding (residual) earlier layers further ahead. The smaller models reached plateaus in loss and accuracy in few epochs with sub par performance, deeper models showed potential for more improvement by extending the training, however the experiments were limited by the available time and computing power.

4.3 Best models

In the end we returned to transfer learning since it showed the best results and experimented with other networks like `EfficientNetV7`, `DenseNet-201` and `Xception`, `Resnet50v2`, obtaining the best results with `EfficientNetV7` and `DenseNet-201` using Global Average Pooling and three hidden dense layers with dropout.

5 Training

The training was performed tracking the accuracy of the model with cross-validation via hold-out. By measuring the validation accuracy and loss, we were able to keep track of potential overfitting over the training data and observe the generalization properties. The training was performed mainly on local hardware and on Google Colab, although, due to the limited resources, the training for larger models had to be terminated before reaching the early stopping criterion.

For the transfer learning models, training was performed in two or more steps: first the pretrained network bottom was frozen, serving as feature extraction in order to prepare a sufficiently accurate classifier top, without risking to worsen the pretrained model. Then part of the convolutional layers were unfrozen, going from the deepest part back, this way, the basic edge and texture detection was left unaltered, but the higher level features could be fine tuned for the task at hand, all while reducing the training time.

For the custom models the training was performed on the full network, first with a higher learning rate, then lowering it after the model started closing in towards a minimum. Later on, we also experimented with training the feature extraction part of the model as an autoencoder, then removing the decoder part and replacing it with a classifier, however we couldn't notice significant improvements, possibly due to the limited training time.

6 Results

This section shows the results of the most successful models used for the classification task, and the accuracy values are reported in the table below.

Model	Accuracy
Xception	0.7264
Resnet50V2	0.8264
Resnet152v2	0.8736
EfficientNetV7	0.9038
DenseNet-201	0.9075

Table 1: Model accuracy on remote test data

As we can see in the accuracy data 1, the best results were obtained with transfer learning starting from a pre-trained `DenseNet-201`.

Following, we report additional metrics and results from the best model on local test data. The confusion matrix (Figure 5) shows classification performance on the local test set and the loss and accuracy metrics (Figure 4) show the training speed and metric trends.

7 Conclusions

From the experiments, we noticed that, due to the limited training resources, models using transfer learning with fine tuning obtained the best performance in terms of accuracy and training time. Further gains could possibly be achieved with fully custom networks and sufficient training time, nonetheless the transfer learning models showed outstanding results.

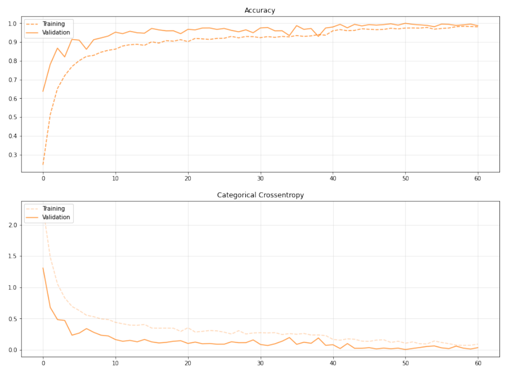

Figure 4: `DenseNet-201` model training

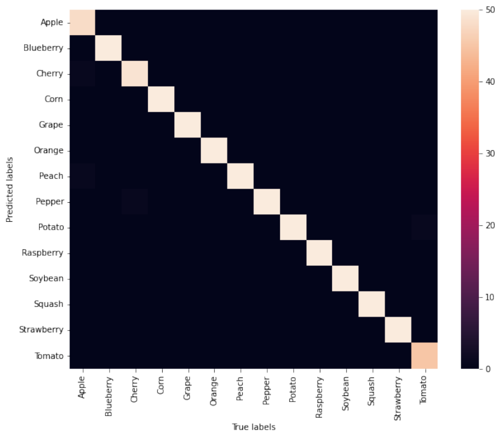

Figure 5: `DenseNet-201` model confusion matrix

Tools

- Tensorflow
- Keras
- Scikit-Learn
- Jupyter notebook