Image Classification Challenge Report

Filippo Brozzi, Manuel Cecere Palazzo, Daniele Manfredonia

November 28, 2021

Abstract

In this document we'll shortly present how we approached the challenge, which was our workflow and the issues we faced, how we decided to overcome some of them, and finally describe our best models, explaining the rationale of our choice and the results they produced

1 The challenge and the dataset

We were given a dataset of over 15000 images of leaf, distributed over 14 different type of plants, and asked to produce a classification model over that categories.
Our first intuition was to better analyze the dataset, in order to understand the best way to use it to reach our goal. At a first glance we noticed a high imbalance in our data: certain classes were highly represented, while others had just a few photos.
With this in mind, we expected a simple convolutional model trained on this dataset to produce good results only on the 'popular' classes, and indeed, our first submission got a pretty bad score overall, but a decent one on tomatoes, which was the plant with over 5000 photos in our training set, almost a third of the entire set.

2 Ideas to solve the imbalance

This intuition led us to the need to solve this imbalance, and so we inspected and tried several options

2.1 Class weights

Our first try was to instruct the model to focus on the training on certain categories, by applying a set of weights that would increase or decrease the contribution to the loss score based on the category on which the misprediction was made. Obviously, the idea was to highlight errors on less represented plants, and not to focus too much on mispredicted tomatoes.

$$w_i = \frac{1}{\#\text{samples of class i}}$$

We also tried to scale this weight set by the total number of photos:

$$w_i = \frac{\#\text{ of all photos}}{\#\text{samples of class i}}$$

However results showed no improvement, and on the other hand accuracy on tomatoes actually dropped.We got the opposite of what we were looking for, as the net trained less on tomatoes, but did not manage to improve on less represented categories [Figure 1]

2.2 Data augmentation

We came to the conclusion that our dataset was not really well-suited to achieve good results, and so we tried to include some data augmentation in order to introduce more training material to our network.
By taking a deeper look at the photos we assumed that translation, rotation, flips and brightness shifts were the most suited transformations for the model to be invariant to, since most likely different leafs of a same plant differ only according to a combination of these transformations.
This was our first step toward better results, since we obtained an improvement of almost 45% of accuracy in the test set. [Figure 2]

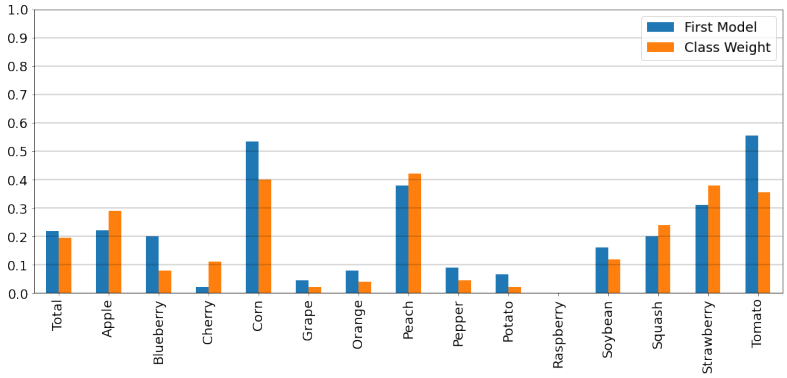

Figure 1: Accuracy per class with class weighting and with our first simple model

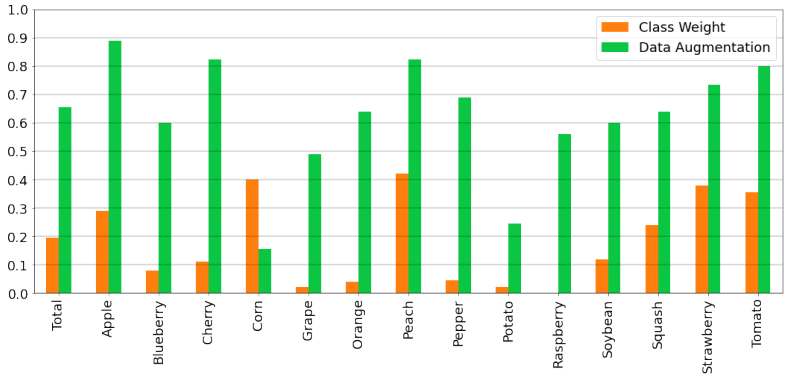

Figure 2: Accuracy per class with class weighting and with data augmentation

2.3 Undersampling and oversampling

Still, accuracy on tomatoes remained far better than on other categories, leading us to the conclusion that the imbalance was only partly solved.
Our next step, was then to precisely balance the dataset via random undersampling, cutting the number of images per plant to one of the less represented one: Raspberry.
We then trained our model with this small dataset and data augmentation, and obtained pretty leveled results, but a worse overall accuracy.
We concluded that our idea was valid, but that the dataset had been shrunk too much, and so we added random oversampling by simple duplication to the preprocessing of our training set in order to create a well-balanced dataset with a significant number of images.
Once developed and used for training in addition to do data augmentation, this dataset showed significant improvements in the overall score while still maintaining partial scores balanced. [Figure 3]

2.4 Other techniques

During this phase of experiments we also tried some other techniques, but being them not effective, and for the sake of brevity, we are just going to briefly explain this approaches and the motivation behind them.

- *Sigmoid Focal Cross Entropy:* Focal loss was first introduced in the RetinaNet paper. Focal loss is described as extremely useful for classification when you have highly imbalanced classes in the TensorFlow documentation. It down-weights well-classified examples and focuses on hard examples.
- *Area Under the Curve:* The accuracy metric can be misleading to evaluate the performance of a model with imbalanced classes. Therefore, we tried to monitor the AUC in the Early stopping procedure.

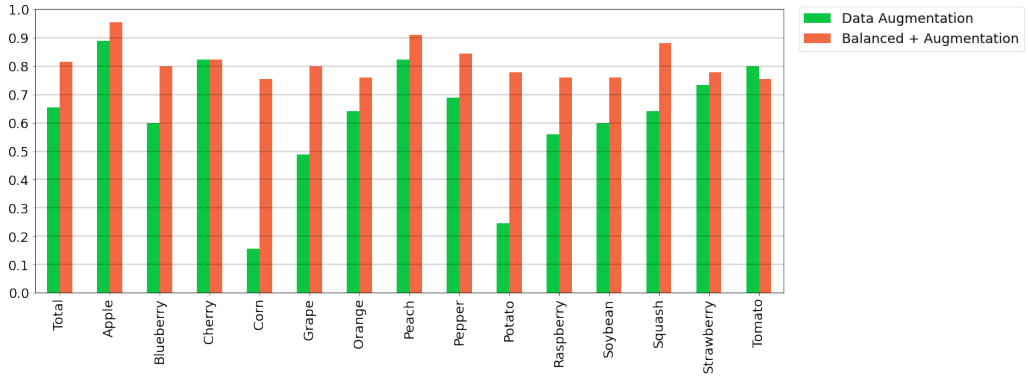

Figure 3: Accuracy per class using only augmentation and using also our balanced dataset for training

3 Final Models

All of our final models were trained over a set of 20451 images. Each category was represented with 1398 images, except Tomato which had 2277.
Our validation set consisted of 5120 images, which were different from the ones of the training set. Each class had 350 images, except Tomato which had 570.
We chose not to extract a test set out of the original dataset, since after a few tries we observed that it wasn't a good representation of the real test set.The accuracy on the validation set has been enough to compare candidate submissions, when needed.

3.1 MarkZuckerberg

Our model is a CNN, with 6 alternate layers of convolution, batch normalization and max-pooling, taking as input a 256*256*3 normalized tensor and reducing it to one with shape 4*4*128.
Once flattened, the tensor passes through three fully connected layers, where dropout is introduced at each step.The first two layers consists of 512 neurons, while the last one has 256.
To obtain better results, we introduced a callback to decrease learning rate if training reaches a plateau.
With this hand crafted model, we reached an overall accuracy of 81,5 % [Figure 3 and Figure 4] In this phase to navigate the space of hyperparameters we also used the Keras Tuner framework with the Hyperband algorithm. Unfortunately, this approach didn't give us the best results, probably

3.2 Fine-Tuning and Transfer Learning

At this point, we decided to use more famous networks through Fine-Tuning and Transfer Learning. We selected the most promising networks to experiment with browsing the keras.applications documentation, and picked VGG19, Xception and EfficientNet due to their top-1 and top-5 performance on the ImageNet validation dataset.We used their own preprocessing function and replaced the top classification network. Xception and EfficientNet B7, and especially the second one, gave us the most interesting results.

3.2.1 SteveJobs

This model is made through the use of the Xception network for the convolutional part, taking as input a 256*256*3 preprocessed tensor. Once flattened, the tensor passes through one fully connected layer of 256 neurons, with dropout. The fine-tuned application gave us the best results, with an 83.7% accuracy on the final test set. [Figure 4]

3.2.2 BillGates

This model was the one characterized by the best overall results of all models. We used EfficientNet B7 in the convolutional part of the network, taking as input a 256*256*3 preprocessed tensor. The output of EfficientNet is then passed through a GAP layer, a BatchNormalization Layer and then the output layer. Also in this case we used a callback to decrease learning rate if training reaches a plateau. The Transfer learning and Fine-tuned models yield us similar results, respectively 89.2%

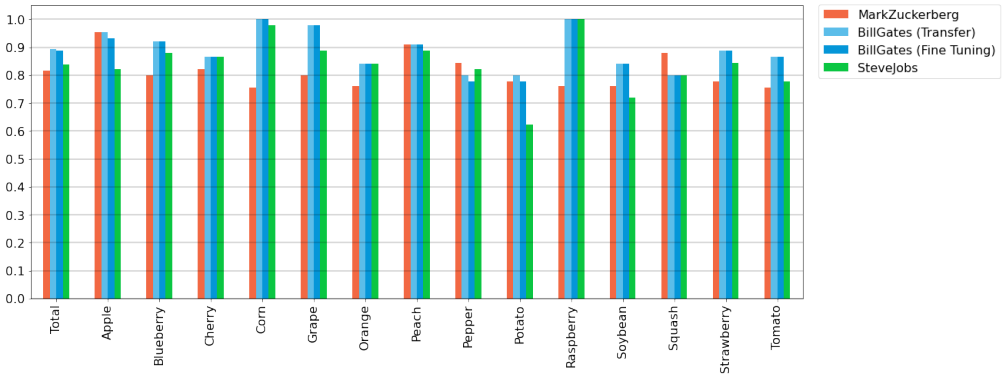

Figure 4: Accuracy per class in all of our final models

and 88.6%. [Figure 4] This difference, if statistically significant, could be due to the big number of parameters against the relatively small dataset.