Homework #1 - Report

I. Introduction

A. Classification Problem

The problem was a multiclass Image classification problem where we were required to classify images of leaves, divided into categories according to the species of the plant to which they belong.
The classes were the following ones, they are 14 and at the side of each class the respective representation:

1. `Apple : 988`
2. `Blueberry : 467`
3. `Cherry : 583`
4. `Corn : 1206`
5. `Grape : 1458`
6. `Orange : 1748`
7. `Peach : 977`
8. `Pepper : 765`
9. `Potato : 716`
10. `Raspberry : 264`
11. `Soybean : 1616`
12. `Squash : 574`
13. `Strawberry : 673`
14. `Tomato : 5693`

From this we can already notice a substantial imbalance in the dataset we were provided, but more on this in the next parts of this document.

II. Training phase

After the brief introduction to the problem we can start to take a look at the main body of the challenge that was the training of our neural network.
To do this we made use of Tensorflow and more specifically Keras that we used both to build and manage our model and its ImageDataGenerator to load and manage the images of the dataset.
As it regards the data one thing we initially tried to do was trying to do some **data cleaning**, we looked at the images in the dataset, and we removed all images that were erroneous (ex. Img 14219, that was full black) or that we thought it would not help the training. Lastly, we argued and tried to see if the fact that leaving the small number of images that were with the background would benefit the training or not. Moreover we used only the images that were given to us.

A. First approach

As a first way to approach the challenge we simply used one of the basic models we viewed in class, without changing anything but a modification on the output layer to match the number of classes that we have. So, it was composed of five convolution layers, one flattening layer, one dense layer and the final output layer.
The split between training and validation set was simply made by the *validation_split* parameter in the ImageDataGenerator and it was set with a ratio of 0.8 and 0.2.
The results obtained from such a network after an average of 40-50 epochs were not good at all, the final score of this trial was between 0.14 and 0.16. This low accuracy on the test set made us think that one of the problems that we first had to solve could have been the images contained in the test set were modified or put in a way that our simple learner couldn't predict, and this brought us to the next part of our training process.

B. Data Augmentation

We started applying data augmentation on our dataset in order to increase the amount of data by modifying the data in our possession through Keras' ImageDataGenerator. In particular the ways we augmented data were by applying *rotation*, *height shift*, *width shift*, *zoom*, *horizontal flip*, *vertical flip* and *filling mode* with reflection, in order to have images that were presented in the most diverse way possible.
Results improved a little but the results were still not satisfactory at all. The score on the test set was around 0.24 that is a considerable improvement from the first approach but not enough.
As a result of that and by seeing a high accuracy on the Tomato class, we deduced that the real problem of our trained model was the imbalance of the dataset.
Data Augmentation was from now on always included in our Data Generation.

C. Balancing the dataset

We tried to solve the class imbalance problem since the ratio of samples between the class that had the highest number of samples and the class that had the least was 20:1.
Here are the ways we tried to adopt in order to solve this problem.

a. Class weights

At first we tried solving the problem leaving everything as is but changing the way we fit the model by adding the precomputed class weights.
In this case the training process had bad results and a slower learning rate that made us discard quickly the option. The loss was always at high values and so the learning process didn't go well, in fact the results on the test set decreased.

b. Manual balancing

Another solution that we tried was to remove some images from the class images with the highest number of images and copying with data augmentation the images of the classes with the least number of images. In order to have a ratio between the highest sampled and lowest sampled class of approximately 2: 1.

This method turned out to be successful and obtained a way better score of 0.63 on the test set. But the effort it implied was too much for a result that was not so outstanding.

c. Final solution

The final solution was to precompute a division of the training and validation folders, we did this by using the library split-folders, a library which solved the problem that was that the ImageDataGenerator divided in training and validation sets before shuffling, thus creating imbalanced sets.
After this little addition the training of our model improved significantly going up to a precision of 0.7 on the test set and with little to no effort with only 30 epochs of training.

D. Alternative metrics

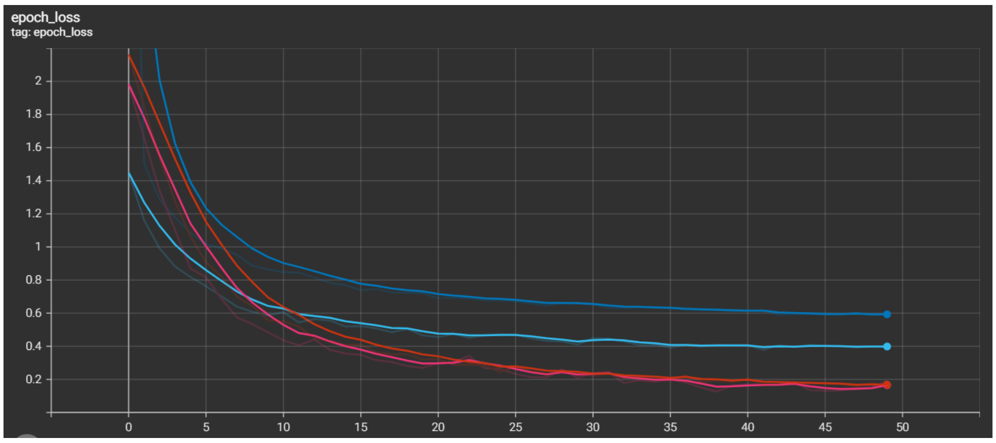
In addition to the simple accuracy, during the previously discussed parts, we also tried to use other metrics instead of the simple accuracy. The metrics we used for this purpose were F1 score, precision and recall. These additional metrics further improved the performance of our trained model because such metrics can be more significant than simple accuracy when dealing with imbalanced data, and the score went up to approximately 0.75.

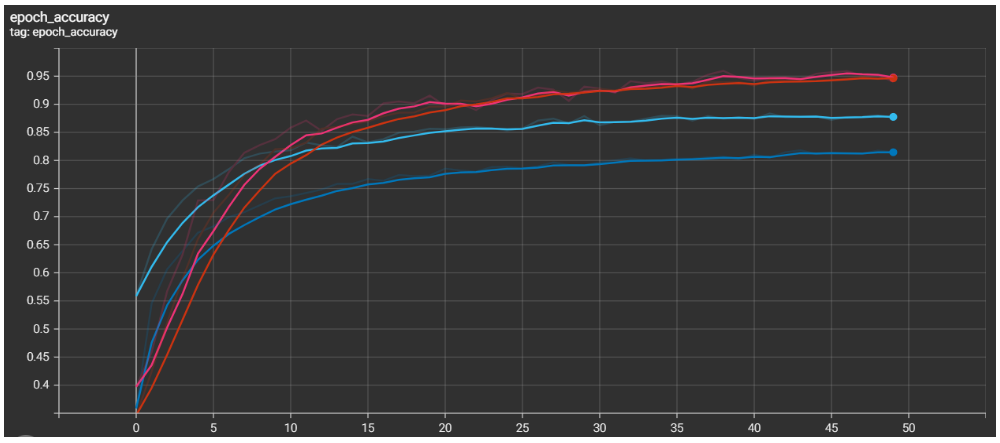
E. Transfer Learning

As a final approach we also tried to use transfer learning, using the knowledge gained while solving one problem and applying it to our related problem.
We used the VGG16 model and, immediately applying all the previous adjustments, the model we built reached the accuracy of the network we created and surpassed it obtaining a score of 0.76 after just 35 epochs. We didn't manage to further improve this model given that time was running low.

III. Conclusions

The following graphs represent the loss and accuracy of the two best models we got during 50 epochs of training from zero.
As we can see from the legenda the bluish ones are the train and validation of the VGG model while the reddish ones come from our built network.
From these graphs we can see how the VGGmodel for the first few epochs fit our set with better results than our model that was build from zero, but going on with the epochs it begins to stall at an average of 0.82 while the other model reaches and maintains an accuracy of approximately 0.95 . By improving the fine tuning on the vgg network , we could improve the performances, letting it adapt better to our specific case.
From the final values on loss and accuracy, it would seem that the VGG based network should perform worse than the other, especially since the overall accuracy is substantially lower. However, the most performant one on the test set was the VGG, we assume because it did not overfit the given dataset as much as the other.


epoch_loss
tag: epoch_loss


epoch_accuracy
tag: epoch_accuracy