POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Politecnico di Milano

054307 - Artificial Neural Networks and Deep Learning

2021

Homework 1

Student:	**Student ID**
Rasmus Tuxen	10844184
Davide Carnevali	10635813
Riccardo Morandi	10609940

Leaderboard Nickname:
DRTNet

November 27, 2021

Data Preprocessing

The original dataset was composed of 17728 256x256 rgb images belonging to 14 different classes. The dataset though was heavily unbalanced, as it can be seen in histogram (1), the class `tomato` was overrepresented being a third of the data while other classes had only a few hundred images. Further analysing the data we noticed that the classes `blueberry, orange, raspberry, soybean` and `squash` were composed only of images of healthy leaves, the class Corn on the other hand was composed predominantly of unhealthy leaves, the other classes where overwhelmingly healthy.
We decided to create a custom balanced dataset that contained 1206 images per class, using a combination of oversampling and undersampling for underrepresented and overrepresented classes respectively, trying in the process to balance the proportion of healthy and unhealthy leaves when possible.
We also noticed that in the pictures the leaves where singled out and had a black background in the training set but in real world images this would not be the case, so we trued to take it into account in the data augmentation that we performed.

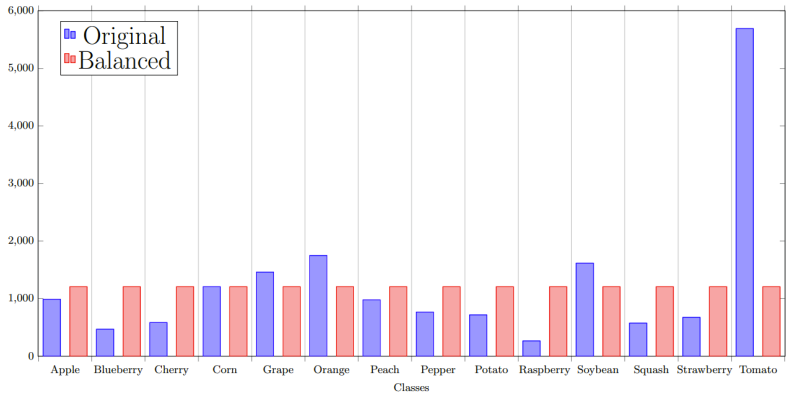


Figure 1: Number of samples in the two datasets

Strategy for training and validation

During or training we decided to split the dataset in training (85%) and validation (15%) sets using the same random seed to enable reproducibility and to make sure that the training and validation set did not overlap. We decide not to extract a test set from the provided data but used the hidden dataset in Codalab as our test set since we were allowed many submissions.

Data augmentation

Early in the process we noticed that there was a rather large discrepancy between the train/validation accuracy and the achieved test accuracy. Looking at the images in the given training set we suspected this was due to a couple of factors; class-imbalance in the training set (discussed earlier), many images of the same class were taken from the same zoom-distance e.g. `blueberry`, black backgrounds in all images and lack of unhealthy plants in multiple classes. Especially the last two concerned us, as the CodaLab challenge had categories 'Healthy'/'Unhealthy' and 'Wild' which would be images taken in natural environments of plant with different health-status.
We therefore decided to introduce a lot of data augmentation in order to align the training/validation accuracy with the corresponding test accuracy.
The augmentation was added in two stages: The first with a number of keras preprocessing functions like shifts, brightness change, shearing, rotations and flipping. The second was with a custom preprocessing function which could: add salt/pepper noise to the image, add noise to the background, swap color channels, change the hue of the image and blurring. The values and probabilities of these augmentations can be seen in the code.

In figure 2 some examples of data augmentation performed on the training set can be seen.

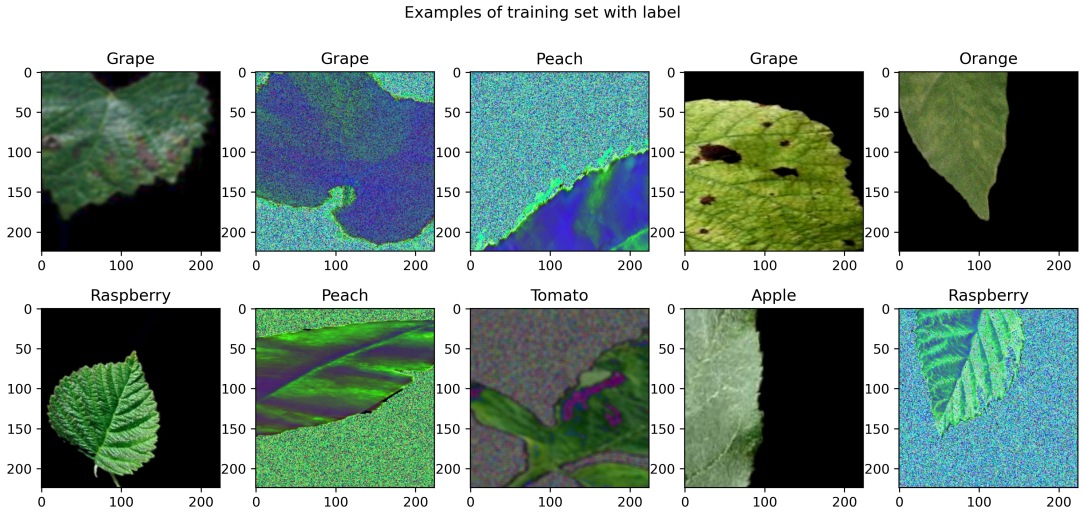
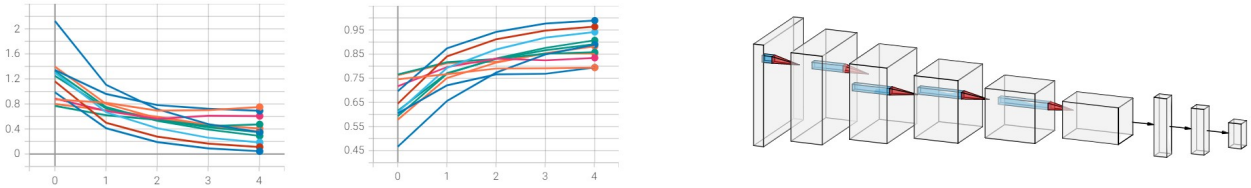


Figure 2: Examples of images from the training set with augmentations

This augmentation was only performed on the training set however. This way all the validation images were still normal and if the validation error would begin dropping during training this would have meant that the model was learning a wrong representation of what plants looked like.

Experimentation with model architecture

We first tried to build a convolutional neural network without relying on foundation models, therefore we decided to implement a gridsearch to select architecture. The hyperparameters were the number of `(conv2D-> Relu-> MaxPool2D)` blocks, each with kernel size (3,3) and the number of dense layers in the classifier, the number of units per layer in this case was 256 and 128 respectively (if present), and we were using a flattening between the convolutional part and the classifier. The results, shown in 3a and 3b, were unsurprising since the more we increased the size of the network, the better their performance got. We therefore decided to adopt an architecture as shown in 3c with 5 `conv2D` layers and one dense layer that with trained with mild data augmentation obtained a score of 60% on the test set.
We then decided to test the same network architecture but with a `GlobalAveragePooling2D` instead of the flattening layer. The training, shown in figure 4, was performed on the balanced dataset and with the custom preprocessing described above and the model reached a much higher accuracy of 75% on the test set.



(a) gridsearch loss (b) gridsearch accuracy (c) custom convolutional model architecture

Figure 3

Transfer Learning

With the previously described architectures we were only ably to achieve a 75% accuracy on the test set. In order to get a better performance we decided to perform transfer learning with fine tuning.
The transfer learning models are constructed with models from tf.keras.applications. The general structure of the transfer learning architectures is as follows; first the input is normalized to be between $[0, 255]$ such that the transfer model's own `preprocess_input(...)` can be used, then the input is fed through the transfer model. Afterwards the input is flattened with `GlobalAveragePooling2D` followed by a single hidden layer using batch normalization and dropout. Since the output is multi-class we use a softmax activation in the last layer.
After training the first time with the weights of the transfer model frozen, we perform fine tuning where some of the layers in the transfer model are unfrozen; the model is then trained again with a lower learning rate.

In Figure 4 the learning curves for three transfer-learning based models as well as the best custom CNN can be seen. They were both trained for a maximum of 60 epochs. The transfer-learning models used two rounds of fine-tuning beginning at epochs 20 and 40. All are trained on the augmented dataset where the validation set was left without augmentations, this is the main reason for the discrepancy between the training and validation error. The performance scores of our models can be found in table 1 below.

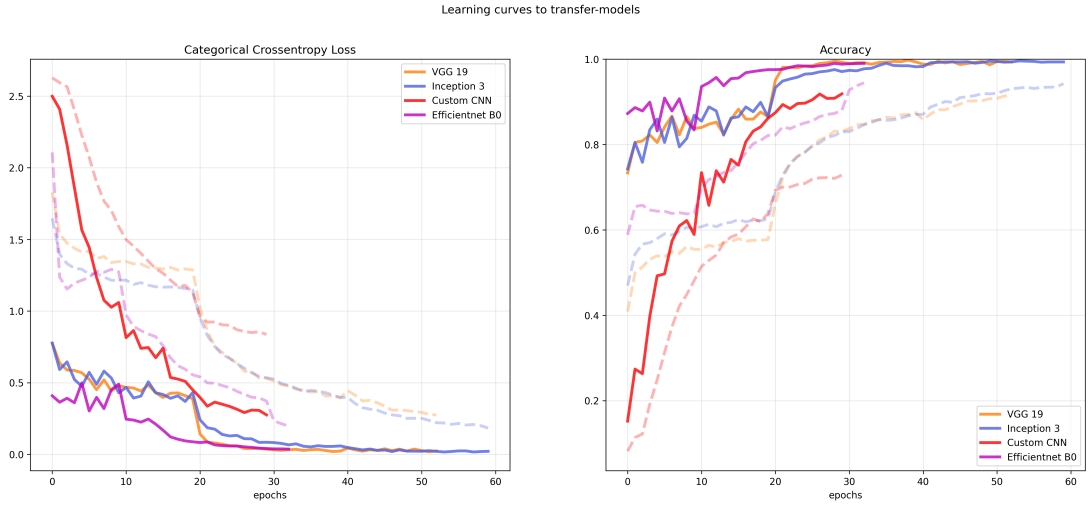


Figure 4: Learning curves from two transfer-learning based models trained on the augmented dataset. The dotted lines are training errors and the solid are validation errors

Conclusions

For the Final phase of the challenge, we decided to submit the three best models which performed best during the Development phase, and in addition trying with the custom fully convolutional model we made. The selected are the ones trained with transfer learning and fine tuning, specifically VGG19, EfficientNetB0 and InceptionV3 (for this one we choose to perform fine-tuning on the last two Inception modules). The results showed us similar accuracies to the previous phase (as shown in Table 1). The InceptionV3 did worse while the other two slightly improved.

	Custom CNN	InceptionV3	EfficientNetB0	VGG19
Development	0.7491	0.9245	0.9208	0.9151
Final	0.7358	0.9283	0.9245	0.9094

Table 1: Test accuracy comparison in the two phases

Regarding our custom model we think that the limiting factor to the performance was the architecture itself which was too small to be able to learn obtain good results, specially on 'wild' images.

On the other side, we figured that the problem for our fine tuned models could be the lack of images, and a more balanced and complete dataset would help to reach best performance.

Another thing to note, is that all the learning curves for the training set have not converged while the ones for the validation set have - mainly because the accuracy is at 99%. In order to achieve a better result, one could consider to train for more epochs and augment the validation images, which would hopefully let the model learn a more robust representation of different leaves.