POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Plants Binary Classification

AN2DL 1ST CHALLENGE
TEAM NAME: POLLIKUNGFU

Alessia Marino: 10933660, Antonino Fabio Coppola: 10926052, Lorenzo Manoni: 10740351

---

1. Introduction

This document is intended to explain the process behind the building of the final model for the first challenge of the Artificial Neural Network and Deep Learning course. The objective of the assignment was to build a model that correctly classifies images with the best accuracy we could reach by uploading the models to the CodaLab platform which will evaluate the model onto a test set that is unknown to us. The public dataset was composed of pairs of images of plants (96x96) and labels that could be either ”healthy” or ”unhealthy”.

2. Data and preprocessing

At the beginning of the challenge, we printed some of the images of the dataset, and was evident that there were some images not pertinent to plants: pics of Shrek and Trolls. For this reason, we cleaned the dataset from outliers and duplicates. In order to use the ”categorical cross-entropy” we converted the string labels into categorical form [1, 0] and [0, 1]. At the end of this process, we split the cleaned dataset into train (80%) and validation (20%) using the 80/20 technique and we were ready to use it.

2.1. Class Balancing

Imbalanced classification presents an issue where one class comprises significantly fewer samples compared to others within the classification. In deep learning, it’s crucial to train an estimator using balanced data to ensure equal exposure to all classes. In the pre-processed dataset, an imbalance was noted, with 63.1% of images categorized as ”healthy” and 36.9% as ”unhealthy.” To rectify this imbalance, the application of the following balancing techniques was undertaken:

2.1.1. Under Sampling

Within this methodology, a subset of examples was systematically removed from the majority class, aiming to achieve a more equitable distribution relative to the minority class. Despite conducting several tests, it was observed that the employment of this technique detrimentally impacted the performance of our models. The RandomUnderSampler function from the imblearn.under_sampling module was employed to implement this approach.

2.1.2. Smote

The SMOTE (Synthetic Minority Over-sampling Technique) function from the imbalanced-learn library was employed to synthetically generate new images for the minority class. Although this technique resulted in a marginal improvement in the EfficientNet model, noteworthy enhancements were not observed for the ConvNeXtXLarge model.

2.1.3. Class Weights

Classifiers in Scikit-Learn offer a 'class weights' parameter, enabling the declaration of the relative importance of imbalanced data. Rather than directly oversampling to achieve class balance, we can guide the estimator to modify its loss calculation method. This methodology has increased the number of true negatives predicted.

2.2. Augmentation

We tried different augmentation techniques for each model, we noticed that the more the model was complex, the more the images needed to be augmented.
At first, we tried to train the model without any augmentation techniques and we saw that the model was overfitting the train set since we did not have so much data (4850 image/label pairs). Therefore we decided to apply data augmentation to the train set (3880 image/label pairs), using different techniques from the Keras library for computer vision: keras_cv, to find an optimal combination for each model. Some of the most suitable transformations were:

- RandomFlip: Randomly flips the image horizontally and vertically;
- RandomShear: Stretches the image distorting it;
- RandomContrast: Change the contrast of the image;
- RandomHue: Change the hue of the image;
- GridMask: Apply black squares to the image covering some parts;
- CutMix: Generates an image from square patches of other images and gives percentages at labels;
- FourierMix: Generates an image from patches of other images and gives percentages at labels;
- MixUp: Generates a weighted combination of random image pairs from the training data.

3. Transfer Learning and Fine Tuning

Transfer learning involves taking a pre-trained network model designed for a specific task and adapting it to excel in a related but distinct second task. We used different pre-trained models from the Keras Application Library and their weights are initialized with the "ImageNet" dataset. These pre-trained models were integrated to our model, freezing the layers so none of the parameters were trainable. Following this layer, additional levels of batch normalization, dropout, and dense layers were introduced, adapting dynamically based on the specific pre-trained model employed. Subsequently, we transitioned to a phase where all batch normalization layers and the last 1/4 layers of the pre-trained model were set as trainable except for the BatchNormalization ones. This strategic adjustment aimed to tailor the model to our specific case. The fitting process was then re-executed to fine-tune the model for improved performance in our target domain.

4. Development Models

4.1. From Scratch

The initial strategy for the challenge involved crafting a customized CNN model, inspired by concepts covered in lectures. We created a sequential model with 4 convolutional layers, 3 Max Pooling layers, 1 Global Average Pooling, and a final dense layer. In total, the network had 97569 learnable parameters, which is rather small compared to millions of parameters of the most famous pre-trained networks. We trained our model for 84 epochs with early stopping. We tried to use the activation function Sigmoid instead of Softmax in the last Dense layer, however we achieved approximately the same results. The results are shown in the table below. After obtaining these results, we decided to explore additional models in an attempt to enhance the outcomes.

4.2. EfficientNetV2B3

After building a model from scratch we tried to use a pre-trained model from Keras. We chose EfficientNet because was one of the most popular on the web and we gave it a try with different versions of it but the most suitable for us was the "EfficientNetV2B3". We increased the augmentation on the train set and then fitted the model and we increased the accuracy with respect to the previous one. This approach had a big impact on the improvement toward our goal. We decided to use bigger models with respect to EfficientNet in order to see if there was a big achievement gap.

4.3. ConvNeXtBase

Subsequently, we thought to utilize the ConvexNetBase network, which yielded superior results compared to the EfficientNetV2B3. We constructed this network by implementing augmentation techniques while retaining the same layers within the sequential model. For optimization, we opted for AdamW as the preferred optimization function, recognizing a significant enhancement in performance compared to when we utilized the Adam optimizer. Initially, we conducted Transfer Learning with 1000 epochs and a batch size of 32. Afterward, we improved the model by implementing the Fine-Tuning technique with 4000 epochs, a learning rate of 0.5, and using the same batch size, with the early stopping. As we were unable to further enhance the model, we decided to try using the same network architecture but on a larger scale.

5. Final Model

Since we were not satisfied with our results, to achieve our best results we used the **ConvNeXtXLarge** model. In this model, the previously described methodologies are implemented. We built the usual Sequential model composed of the pre-trained model and the Fully Connected Layers. We balanced the classes with the class weights parameter and for the augmentation we applied RandomFlip, FourierMix, MixUp, and others. We started with a lower learning rate and reduce it on Plateau and we implemented the early stopping with an higher patience value. After the model had been fine-tuned, we noticed that was the one with the higher accuracy and in general the one with the best confusion matrix. The fully connected layer is structured by replication of a specific block of layers. The latter consists of a Dense layer with a predefined number of units and a ReLu activation function, followed by a Batch normalization layer and a dropout layer with a 50% rate. This block is repeated twice with a progressive decrease in the number of units. In conclusion, the output layer with the Softmax activation function is presented.

6. Comparison between models

To conclude, we compare the results obtained by the best models, for each technique used. The accuracy and the validation accuracy are based on the split of the dataset into the training and the validation set. With the final model we achieved 87.80% accuracy in the test set. The results of previous submissions were lost as they were deleted during the second phase of the challenge.

	Accuracy (%)	Validation Accuracy (%)	Test Accuracy (%)
From Scratch	69.12	74.02	-
EfficientNetV2B3	68.84	81.44	-
ConvNeXtBase	73.32	86.39	-
ConvNeXtXLarge	92.37	93.41	87.80

Table 1: Accuracy and Validation Accuracy comparison

7. Contributions

Each of the three team members significantly contributed to the project by delving into and experimenting with various types of neural networks. Each individual played a pivotal role in exploring and testing diverse functions, thereby making a substantial contribution to the research and development of the ultimate solution.
We thoroughly enjoyed the process of discovering and implementing new techniques, enabling us to enhance our models by leveraging the wealth of knowledge gained during our coursework, while also actively seeking out new insights and information.

References

[1] Classification on imbalanced data https://www.tensorflow.org/tutorials/structured_data/imbalanced_data?hl=it

[2] Over sampling methodshttps://imbalanced-learn.org/stable/references/over_sampling.html

[3] Keras Computer Vision https://keras.io/examples/vision/

[4] Keras Application Models https://keras.io/api/applications/