ACADEMIC YEAR 2023/2024

POLITECNICO
MILANO 1863

Artificial Neural Networks and Deep Learning

Challenge 1 - Image Classification

Stefano Bruni - 10660827
Matteo Pisani - 10921568
Flaminia Telese - 10931641

Prof. Matteo Matteucci

Problem definition

In the first homework, we were faced with a binary image classification problem.

The aim was to build a model able to distinguish healthy plant images from unhealthy ones. To solve this problem, each team had to design and train a convolutional neural network which processes images and assigns them a label indicating their health state. An annotated dataset was available and it was to be used for model training and validation. The quality of the designed model was to be assessed over a separate test set, which was not accessible to the competitors. The metric used for evaluating the model was the test set accuracy; other metrics (precision, recall and F1 score) were also computed.

Data analysis and cleaning

The dataset we were given to train our models consisted of 5200 images of dimension 96x96 pixels representing plants divided in two categories: 'healthy' and 'unhealthy' (respectively '0' and '1').

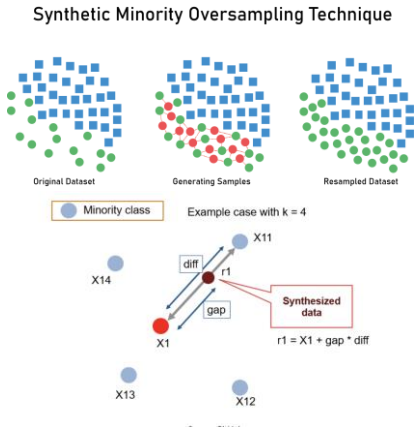
First we visually inspected the dataset and noticed some outliers which didn't represent any plant at all. We deleted all of them since they could lead our model to learn wrong features.

Once all the outliers were eliminated we also deleted the duplicated images because learning on those could lead our model to overfit.

At this point our cleaned dataset consisted of 4850 images: 63.1% of the images represented healthy plants while the rest represented unhealthy ones, showing an imbalance towards the healthy class.

Class unbalance

Unbalanced datasets are prone to overfitting and yield low accuracy. We tried and tested many balancing techniques (like random undersampling or oversampling and two phase-training). Amongst them, two were particularly effective: class weighting (which despite being simple to implement, needed to be heuristically tuned to achieve satisfactory results) and SMOTE [2], a more sophisticated technique to synthetically class-balance the training set. Normally, SMOTE is used to deal with structured data rather than with images, but nonetheless it helped generalize our model in a considerable way. SMOTE can be rapidly described as follows: take the difference between a sample and its nearest neighbor, multiply the difference by a random number between 0 and 1, add the difference to the sample to generate a new synthetic example in the feature space and repeat until the dataset is balanced.



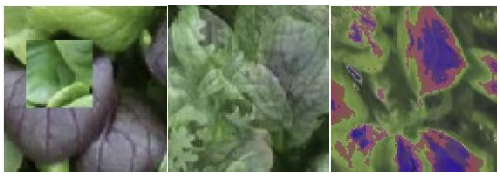
How SMOTE works

Synthetic image using SMOTE

Image augmentation

We performed data augmentation to increase the model's generalization capabilities. To mimic a larger, more diverse dataset, augmentation was performed on the validation set too. On the training set, we used CutMix, where two images are combined so that one is put inside the other. This helped increase

regularization and enhance the prediction accuracy on new data, as the network learns to focus on smaller sections of the image [3]. We also employed MixUp [4] and it further improved the generalization error if used in combination with the former. Finally, Keras CV library function RandAugment was specifically used on the validation set to simulate a broader, more complex dataset by means of random geometrical transformations. In this way, we made it a better proxy for test data, thus the validation error was more indicative of the performance over new samples. This helped us in our design decisions.

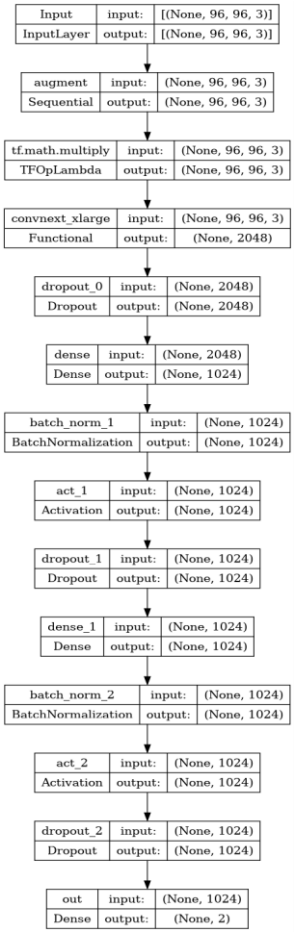


Examples of, respectively, CutMix, MixUp and Randaugment augmentations.

The model architecture

The first requirement was to make the CNN powerful enough to understand the separation of data in the image space. For that reason we employed a ConvNeXtLarge network for feature extraction, after trying smaller networks which yielded worse performance. Moreover, we used a non trivial feature classifier which consisted of three fully-connected layers. Another challenge was avoiding overfitting, considering we had little data for a relatively complex network, so we employed different techniques to obtain a strong degree of regularization. The overfitting problem was further exacerbated by the fact that the training set didn't follow the same distribution of the test data.

The final architecture first had a preprocessing layer, which applied many geometric transformations, in order to train the network to further generalize to images distant from the training ones. It then had the FEN mentioned before. It was followed by the Feature Classifier part, which consisted of two blocks containing a dense 1024 neurons layer (each using an L2 kernel regularizer), a batch normalization layer, a ReLU activation and then a dropout layer. A last dense layer with softmax activation and two neurons provided the output. The model was compiled with a CrossEntropy loss function and the AdamW optimizer, to help prevent overfitting. The overall model had more than 300 million parameters, the majority inside the FEN. The model's hyperparameters were manually tuned to optimize performance over the validation set.



Schema of the CNN architecture

Model training and validation

The training consisted of the standard transfer learning and fine tuning phases. To further prevent overfitting, in both steps we used early stopping and learning rate scheduling, using the callbacks provided

by Keras. We imposed a lower patience during fine tuning to have a stronger regularization, together with a faster learning rate decrease. As we observed many models had a low recall, and we wanted to have balanced metrics, we traded precision for recall by giving more weight to positive (unhealthy) samples in the loss function. This parameter was tuned manually to 1.3.

For validation we used the classical hold-out approach, which was the only feasible one given the long training time; moreover the validation set was augmented randomly. We did not use any of the available data for testing, as this practice could have promoted overfitting of the given dataset. Our best model managed to obtain almost 80% accuracy on the validation test after transfer learning and up to 90% after fine tuning. The training process took around five hours and the final compressed model weighed more than 2 GB; for that we used Kaggle, which had more free-tier RAM and GPU than Colab.

Alternative Architectures/Experiments

Noisy Student

We also built a model using a semi self-supervised approach named *Noisy Student* [1]. We first built a 'teacher model' implementing transfer learning and fine tuning on an *EfficientNetV2M* with a Dense layer with 'ReLU' activation, a BatchNormalization layer, another Dense layer and finally a Dropout layer. We trained this model using a partition of the dataset, with only geometric manipulations and balancing by SMOTE.

Then we trained the 'student model' on the previous dataset, but adding noise with RandAugment and labeling by the teacher's predictions. Our 'student model' reached an accuracy of 81.5% on our local test set and one of 78% on the first CodaLab test set.

We found this approach interesting, especially because it proved itself to be quite robust to overfit. We decided not to explore it any further since we thought it could be improved only by using an external, unlabeled and larger dataset, which was not allowed by the rules.

Ensemble of EfficientNets

We trained several models by using transfer learning and fine tuning on EfficientNets of different versions. We trained the models with different methods: using different augmentation techniques, different partitions of the dataset in the two phases, different techniques to manage class imbalance, different callbacks and a different number of Dropout and BatchNormalization layers.

We built our ensembles with three models: one that used an *EfficientNetV2L* for transfer learning, one an *EfficientNetV2M* and the last an *EfficientNetV2S*; the final prediction was decided by the majority voting of the three models.

The ensembles reached an accuracy of around 85% on our local test and 87-88% on the CodaLab first phase test and one of 80.9% and 80% on the second phase test.

Results and conclusions

Our best model used the ConvNeXtLarge net as FEN and obtained almost 86% score in the second phase of the competition. During the first phase we reached 88% using a slightly weaker model which employed EfficientNetV2L. We tried many more techniques, but the simple yet solid model illustrated before gave us the best results, as well as the most theoretical guarantees.

Individual contributions

On the premise that all teammates contributed to every step mentioned in this report, the main contributions were:

- Data Analysis and Cleaning, Noisy Student and Ensemble of EfficientNets for Flaminia Telese;
- Best Submitted Model that used Transfer Learning and Fine Tuning on a ConvNeXtLarge for Matteo Pisani;
- Class Imbalance and Image Augmentation for Stefano Bruni.

References

[1] Paul, S. (2021, April 19). *Consistency training with supervision*. Keras. Retrieved November 18, 2023, from https://keras.io/examples/vision/consistency_training/

[2] SMOTE: Synthetic Minority Over-sampling Technique: https://arxiv.org/abs/1106.1813

[3] CutMix: Regularization Strategy to Train Strong Classifiers: https://arxiv.org/abs/1905.04899

[4] Mixup: Beyond Empirical Risk Minimization:https://arxiv.org/abs/1710.09412

[5] RandAugment: Practical automated data augmentation with a reduced search space:https://arxiv.org/abs/1909.13719