Artificial Neural Networks and Deep Learning 2023 - Politecnico di Milano
3 Pentium 4 in a Trenchcoat
10667728 Lorenzo Morelli, 10676573 Davide Edoardo Pellegrino, 10703747 Pier Luigi Porri

Image Recognition on plant health with ConvNeXt

The primary objective of this year's challenge revolves around a binary classification problem focused on predicting the health status of plants depicted in labeled images. The task entails assigning each image to one of two classes: *'Healthy'* or *'Unhealthy*.' The team goal is to develop and test models tailored for this binary classification task.

1 How we approached the task

As an initial exploration of the challenge, our team conducted a thorough examination of the provided data, by analyzing the distribution of labels. As expected from such a small dataset, approximately 62% of labeled data belonged to one of the two classes, *'Healthy'*. Initially, we didn't expect this aspect to pose a challenge. Further exploration revealed the presence of numerous outliers, in the form of unrelated images, which we promptly deleted.

Additionally, we began speculating on how our model would perform on images formatted differently from the ones at our disposal. In fact, during the data inspection phase, it was observed that the plants were all well-centered in the images and with a similar framing. Thus, some architectural choices were deemed necessary in the models to address potential issues related to these aspects, to help mitigate the risk of overfitting. Specifically, we opted for *random translation* and *random rotation*, as these were the best performing transformations. Together with the *Global Average Pooling* implementation, we were aiming at making our model as robust as possible.

During the dataset splitting phase, we agreed to devote 10% of the entire dataset each to *testing* and *validation*. Since the labels had textual value, encoding them through *one-hot encoding* was needed, since our initial goal was to build a network with two output neurons. Consequently, we considered the use of the softmax activation function, as well as the categorical cross entropy as our loss function.

2 Experiments and failures

2.1 CNNs from scratch

As for the network, we began our work with a vanilla *LeNet*. After building our network, we compiled the model using the Adam optimizer, as well as implementing early stopping, on validation accuracy to mitigate potential overfitting, for the initial training.

Each of our initial attempts all yielded the same result: a test accuracy of 62%, coincidentally the exact same percentage of *'Healthy'* samples in our dataset. This correlation was further confirmed by the confusion matrix plotted on the predictions on the test set, which showed an almost total distribution on True and False Positives and little to no *'Unhealthy'* predictions. At this point, we hadn't figured out the reason for this problem yet, but it was already clear to us that a simple LeNet network would not be enough.

We then decided to pivot to a more complex network, this time of our own creation, consisting of 5 convolutions in which the number of filters doubled each time, starting with 32. This new model was the first to provide us with enough accuracy to break the 62% barrier, although still being quite unreliable on unseen data such as the evaluation dataset on Codalab.

2.2 K-Fold Crossvalidation attempts

At this point, we had two options to increase performance: either change our model, or try to improve the current one. We initially opted for the latter, and turned to *K-fold Crossvalidation* as our optimization method of choice.

However, despite some improvements on validation and test accuracy (especially w.r.t *smoothness* and *steepness* of the curve), our newly validated network failed to reach more than 62% on the evaluation dataset. Thus, considering the cost in terms of resources and time that this technique requires, we ultimately decided on pivoting to the first original option: changing to a different model.

Meanwhile, a crucial design decision was made: opting for one single neuron in the output layer, rather than two, and consequently switching to a *sigmoid* activation function. This radical change in strategy was influenced by our research: many of the binary classification models that we analyzed online presented this particular structure, while the multi-neuron architecture was more common in multiclass classification problems.

3 Transfer Learning and Fine Tuning

3.1 First TL attempts

Feeling that the one that we had built was overfitting and too weak, we decided to move on to a more complex model, turning our attention to the wide variety that Keras offers, and further improving it with the *Transfer Learning* technique.

3.1.1 Mobile

Our initial venture into transfer learning involved utilizing the *Mobile* architecture. We ultimately decided on this particular network mainly because of its manageable size and efficient performance on image recognition.

The subsequent logical step involved *fine-tuning*. After watching some informative material online - particularly, the guide provided by Keras to fine-tuning proved to be quite helpful - we went for a single fine-tuning round, freezing the entire Mobile network but the last block. Despite the fact that this was our first attempt with transfer learning, it yielded satisfactory results, in particular it shot us above the 80% barrier on the Codalab evaluation dataset for the first time.

However, applying fine-tuning multiple times, after a certain point, started to have an adverse impact, degrading performance and generalization on unseen data despite improvements on our training set. Our final recourse involved adjusting hyperparameters to potentially enhance performance. Despite thoughtfully chosen parameter values influenced by observed model behavior, such as reducing the learning rate and batch size, the resulting improvement was marginal and not statistically significant.

3.1.2 EfficientNetV2L

Meanwhile, we tried to branch out and test multiple different models to more efficiently find the best one. The next on our list was *EfficientNetV2L*.

This particular network presents the peculiarity of a much larger size, an aspect that we thought would translate to a better performance. However, just like K-Fold Crossvalidation, the little improvement in accuracy did not justify the sheer increase in resource consumption and training time. Our last attempt at Transfer Learning resulted in our final model of choice.

3.2 Fine Tuning on ConvNeXt

While researching on the topic of image recognition on NN forums, we noticed that the name *ConvNeXt* kept appearing in articles and discussions. Thus, we figured we could give it a try.

In the meantime, we decided it was time to figure out why most of our past non-transfer learned models kept assuming that all samples belonged to the same label. The answer, as we learned not long after, was the steep imbalance of the dataset: as a matter of fact, the *'Healthy'* samples were 3101, while the *'Unhealthy'* ones clocked in at just 1903. That's when we agreed on using some kind of resampling to balance the dataset, specifically oversampling the minority class. Aside from solving our *"62% accuracy"* problem, resampling could also give us the chance of reducing the batch size, since we don't have the risk of training with a batch full of the same class.
We tried many samplers from the *imblearn* library, and after thorough testing we settled on a simple *RandomOverSampler*, which gave us the best performance overall.
ConvNeXt and the new preprocessing method gave us our best model to date: none of our previous tries came close, even with some kind of resampling applied.
Some more testing revealed that we reached a plateau, thus confirming our final model.

4 Final Model

- ConvNeXt Neural network
- Preprocessing: data augmentation, oversampling and the built-in preprocessing layer in the NN
- 32 batch size, early stopping with 30 patience, dropout layer with a rate of 0.35
- 92.11% accuracy on local test data, 91% on evaluation data

References

- Classification on imbalanced data.
  https://www.tensorflow.org/tutorials/structured_data/imbalanced_data.
- Guide by Tensorflow Keras on transfer learning.
  https://keras.io/guides/transfer_learning.
- Data augmentation by Tensorflow.
  https://www.tensorflow.org/tutorials/images/data_augmentation
- RandomOverSampler by imblearn.
  https://imbalanced-learn.org/dev/references/generated/imblearn.over_sampling.RandomOverSampler.html

Contributions

- Davide Pellegrino: LeNet and ConvNeXt, and helped with other models. Mostly responsible for the fine-tuning phase and parameter tweaking, traffic management in Cities Skylines II during training wait times, and contributions to the final report.
- Lorenzo Morelli: EfficientNetV2 and Xception (not mentioned in the report due to unsatisfactory results), and helped with other models. Mostly responsible for experimenting with new different models/networks during the exploratory phase, analyzing the results obtained, and writing the final report.
- Pier Luigi Porri: MobileNet and helped with other models. Mostly responsible for the discovery of new methodologies for image augmentation, strategies to deal with unbalanced dataset, and also helped with the final report.

3 Pentium 4 in a Trenchcoat are Lorenzo Morelli, Davide Edoardo Pellegrino and Pier Luigi Porri.
See attached notebook for more information.