Artificial Neural Networks and Deep Learning 2023-Homework 1

Team name: NCSP

Data Inspection

The first step we took was inspecting the dataset provided, finding immediately some outliers that would've compromised the training and validation process. Another inspection revealed the presence of some duplicate images that we removed to avoid giving more weight to some samples and prevents ambiguity in validation and testing.

The next step was looking at the images and labels to analyse them and find any notable properties. The split between negative and positive classes provided is 62% and 38% so the data is a little unbalanced. Our final considerations regarding the dataset were that the number of samples wasn't very high and that we might need to balance the two classes, two problems we solved respectively with augmentation and oversampling.

Data Augmentation & Oversampling

Data augmentation is a technique used to reduce overfitting when training a neural network. There are multiple transformations that can be applied to the training data, we selected the following: translation, flip, zoom and rotation. We greatly limited the transformation factors to avoid creating distorted or unusable images that would only hinder the training process and compromise the results. We avoided applying augmentations like varying the contrast and brightness of the images because we felt that it would impact the network's ability to differentiate between the two classes, since the leaves' colour is a very important factor.

Another augmentation technique we tried was MixUp: this way two images and their respective labels are merged with a random linear combination creating a new sample. We tried both implementing it by hand as described on the Keras Documentation (see .ipynb for details) and by using a layer as implemented in the keras_cv library (applying it to the input before feeding it into the model, as to not modify the input shape of the model).

Oversampling is a technique used to adjust the class distribution of a data set and we chose to use it in combination with some augmentation techniques to increase the number of samples in the positive class. Additionally, we tried to solve this problem by also manually tuning the class weights to different values, but we didn't see many improvements, so we only used the previously described techniques, which we felt was more than enough.

We split the provided data into train validation and test sets with the following distribution: 200 samples for testing, 20% for validation and the rest for training.

First Model

Our very first approach was trying to use a simple model created by us. We clearly didn't expect high performances, but we wanted to get some familiarity with the problem while testing some of our ideas.

The model was composed of 2 convolutional layers followed by 2 max pooling layers. Then 2 inception modules each one with 3 convolutional layers with increasing size of the filter and a max pooling. These two layers are by a max pooling. Deeper in the net we also have other 2 inceptions modules with a different number of filters. The last part of the FEN is composed by 3 inception modules and 1 average pooling. The Fully connected part of the net is made of 2 dense layers (with respectively 256 and 128 units). In the FEN part of the net, we have also used two skip connections to avoid the mitigate the vanishing gradient problem and facilitate the training. We used 'relu' activation function and a low dropout rate. The network's output, like all following ones, was a dense layer with two units with a 'softmax' to compute the class output.

Transfer Learning

Seeing as our own model was struggling to get good generalization capabilities, we began to exploit transfer learning to try to achieve better performance. We explored several families of networks made available by the Keras library, using the imported weights pre-trained on the imagenet dataset.

We tried different architectures from the module Keras.applications: MobileNet, EfficientNet, ConvNext, Xception,

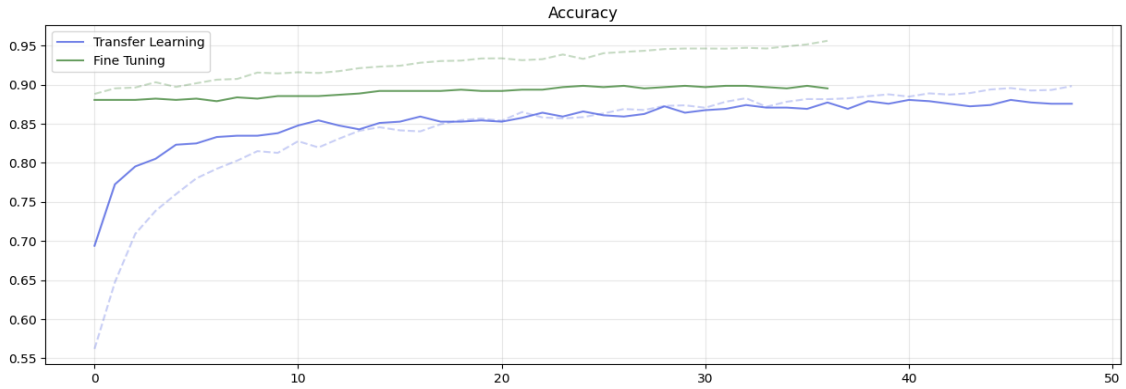
DenseNet, ResNet.
One of the family of networks that gave us the best results is the EfficientNet one. In particular, we tried different sizes of the V2 network completing it with a varying number of dense layers to perform classification on the extracted features with the V2L giving us the best result regarding Transfer Learning.

We immediately noticed that with the biggest networks (such as EfficientNetV2L) our model tended to overfit easily with the addition of dense layers and that the best performance was often obtained with few to no added dense layers. By only adding a softmax activated output layer we managed to get a performance of about 85% of accuracy on our test set.

Fine Tuning

The step that naturally followed transfer learning was that of fine tuning, I.e. the technique of repeating the training process after transfer learning with part of the imported convolutional network unfrozen and made trainable with a very low learning rate. As for before, we tried this process with many of the different families of networks and with a varying number of frozen convolutional layers.

- EfficientNet: The application of fine tuning gave no appreciable increase in performance to the transfer learning model and, rather, usually quickly led to overfitting.
- ConvNext: We start trying to unfreeze the last 30 layers of the network, but this wasn't useful. Thus, we have decided to unfreeze more layers, some improvements came only when we unfreeze more around 150 layers and training them to convergence. As a final step we decide to unfreeze the whole net with a very low learning rate to reach the best convergence; this, on the XLarge model, managed to increase performance from 86% to around 90% on the local test set.



Cross Validation

There were two main reasons for which we decided to try using K-fold cross validation for the MobileNet and EfficentNet models. The first reason was that our training dataset wasn't particularly big so using K-fold cross validation would allow us to use the entire dataset for training and validation. The second reason is that on most trial we would have a large difference between the validation dataset accuracy and the test dataset accuracy, giving rise to the doubt that we were slightly overfitting the validation dataset.

We first trained the MobileNet model using ten folds obtaining a small improvement, of around 4-5%, compared to the MobileNet model trained without cross validation but since MobileNet had worse results compared to more complex model it did not give us any advantage.

Because of this we tried using cross-validation on EfficientNet, a more complex model that gave better result than MobileNet, but due to memory and time constraints we had to reduce the number of folds to five. This brought about an improvement of around 1-2% to the model accuracy but it was not satisfactory compared to the time and computational power taken.

Ensemble

Even if we knew that our resources would not be sufficient to perform a top-notch ensemble model, we still tried anyway with a small number of our best models, employing a majority voting model between them. It managed to increase by about 10% the average accuracy; still, we don't have enough statistics to back this up accurately since it would have been too resource demanding to be performed in the time provided by the challenge.

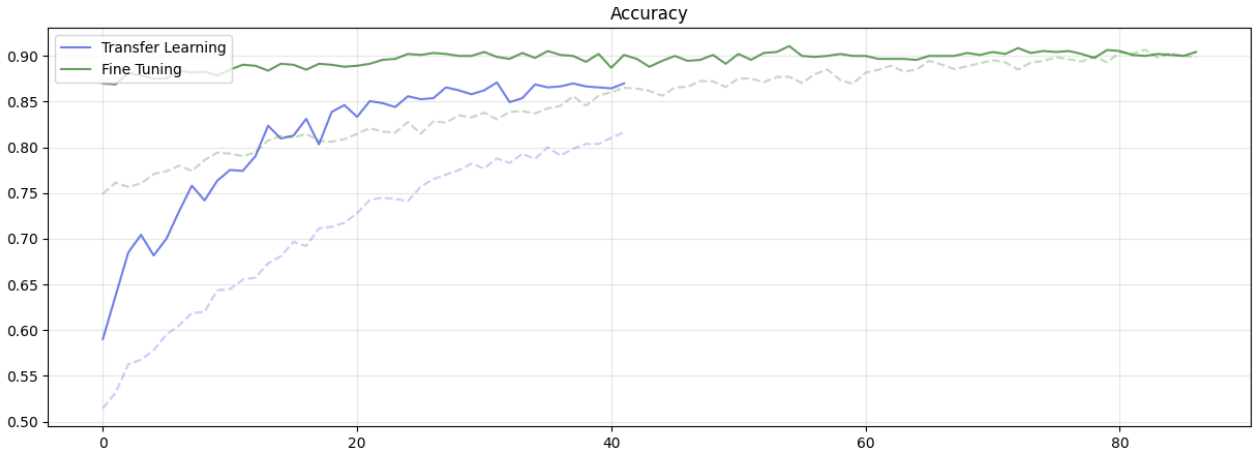
What we didn’t use (and why)

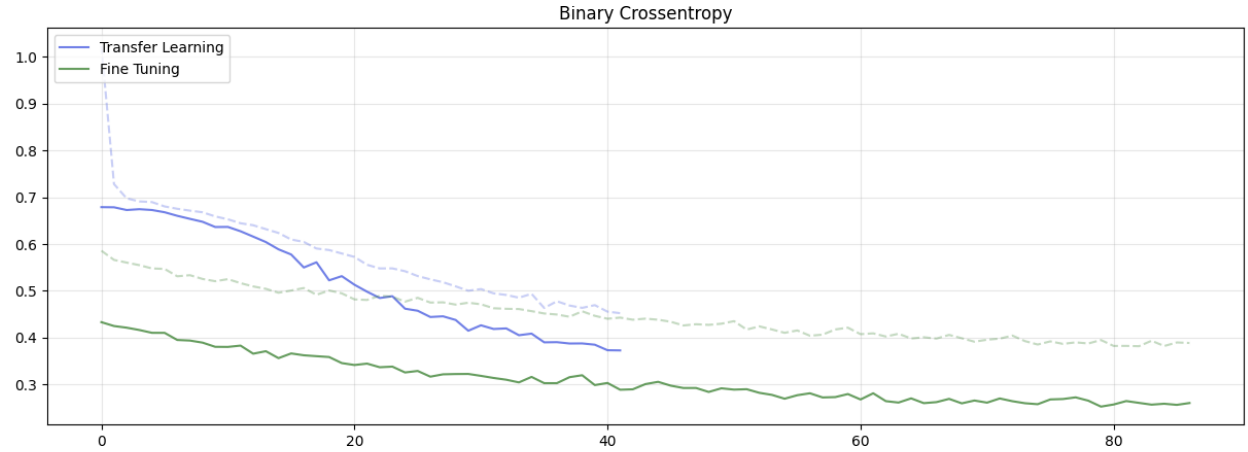
With more resources, we could have performed a more accurate training of an ensemble model by training a larger number of models and by using ad-hoc techniques to handle the sampling of the different training sets of the single models, providing also stronger guarantees on its performance.

We weren’t able to try using k-fold cross validation on the fine tuning of the model due to memory and time constraint. Duplicating the trained model multiple times was not possible so we were limited to three folds and since we had to train a certain number of layers of the transfer learning model it took significantly more time for each fold, due to the size and lower learning rate necessary for fine-tuning.

Final Model

In the end we settled on a model trained starting from the ConvNextBase architecture. We added a Dense layer with 64 units with a ‘relu’ activation and a dropout rate of 0.7. We performed image augmentation as described earlier and introduced callbacks to stop early and to reduce the learning rate on plateau. We then performed transfer learning with a learning rate of 1e-4 and fine tuning with a much lower 1e-6. We also used a small batch size to improve the training performance in particular during fine tuning. We chose to freeze around 200 layers since in our tests this gave the best performance.





Conclusions

In conclusion, our best results were obtained thanks to fine tuning on ConvNext and EfficientNet even though the performances on the Codalab test set weren’t on par with the ones on our set. We saw a drop between 5/10% between the two measures in accuracy and this could be due to a number of reasons but since the online test set wasn’t available we could only speculate. It’s possible that our models were overfitting the provided dataset, lacking the generalisation capabilities required to perform well on unseen data. As previously described, we employed many possible solutions to this problem and we tried many combinations but time and resources constraints limited our available approaches and we couldn’t obtain significantly better results. With more resources we probably could have been able to also provide better guarantees for our results since it would have been possible to perform a more precise validation.

Contributions

Niccolò Maria Rizzi: Data Augmentation and Mixup, Transfer Learning and Fine-tuning on ConvNextBase, ConvNextXLarge and EfficientNetV2L networks.

Marco Baratto: Data Augmentation and Mixup, Transfer Learning and Fine tuning on ConvNextBase and EfficientNet, Transfer Learning test on multiple models

Stefano Bonfanti: Data inspection, Removal of the outliers, Oversampling, Transfer Learning and Fine tuning on ConvNextBase and EfficientNet, Transfer Learning test on multiple models

Giacomo Savazzi: Transfer Learning with K-fold cross-validation on MobileNet and EfficientNet with fine-tuning, Simple ensemble model.

References

Keras Applications - https://Keras.io/api/applications/

Mixup implementation: Keras Documentation - https://keras.io/examples/vision/mixup/

Mixup layer: Keras Documentations - https://keras.io/api/keras_cv/layers/preprocessing/mix_up/

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Going Deeper with Convolutions - https://arxiv.org/abs/1409.4842

Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Deep Residual Learning for Image Recognition - https://arxiv.org/abs/1512.03385

Classificazione su dati sbilanciati - https://www.tensorflow.org/tutorials/structured_data/imbalanced_data