AN2DL 2023 Homework 1
Team_FEMS

Fabio Cabeccia - 10936531, Simone Bortolotti - 10935857,
Elahe Houshmand - 10701966, Mohammadreza Javadi Namin - 10893373

November 2023

1 Dataset Specification

Our first approach to the presented problem was very trivial and mainly involved a very swift human eye exploration of the given dataset. This is composed of 5200 pictures of plants, labeled as 'Healthy' or 'Unhealthy'. Most pictures are close-ups and clearly show the leaves in the foreground, but the foliage is often different in shape, pose and colour. In addition, the light condition in which the pictures are taken can change. Most pictures are not easily classifiable at a first look, since there is no defining feature that characterize each class.

1.1 Dataset Pre-processing

The first action taken on the dataset was to convert the images from .npy to .png format using the Python *Pillow* library, then we looked for outliers or any other disturbing element: as expected our first outlier was found in the first 100 images. To address the situation, since we were expecting more instances of the same outlier plus some others, we decided to use a library called Image Deduplicator to find and print in .png all duplicates inside the dataset, finding not only all instances of the first outliers, but also the ones of a second outlier that was not seen at first. We then proceeded to delete all duplicates obtaining the final filtered dataset used for training our models, which counted a total of 4850 images. Being the dataset this small, we expected to face over-fitting during the training process.

1.2 Dataset Balancing

After the filtering the dataset was clearly unbalanced in favour of the 'healthy' class, with 3060 versus 1790 elements. To avoid under-representation of the 'Unhealthy' class [1], we computed the weights of both classes and then fed them to our training function, in order to work as if our class were already balanced when introducing new models. The function assigned weight 0.71 to the 'Healthy' class and 1.35 to the 'Unhealthy' class.

1.3 Training and validation sets

Knowing that the model would have been tested on the server, we decided to split the already small amount of data into only training and validation sets. Since we wanted to keep as much data as possible for training (hoping to fix overfitting with other techniques), we chose to use only a small fraction of the data for validation. In practice, we tried validation splits ranging from 0.1 to 0.05. No test set was needed during the first phase, but during the second phase we decided to also add a 0.05 test split to the data for both testing the old models and the new before submission.

2 Dataset Augmentation

Finally, before starting to explain how we chose our CNN, we have to mention we focused on enlarging the dataset as much as possible, to give to the feature extraction layer more information to work on. We worked in several directions:

- **Data Augmentation Layers**: All data augmentation layers were inserted after the input one, each implementing the wanted transformation(Zoom, Rotation, Shift, Brightness...). This way, every batch during training was subject to random transformations, further reducing overfitting without having to enlarge the dataset beforehand.

- **ImageDataGenerator**: Same concept as above: the transformation are applied online, but this time using the Image Data Generator module from keras.
- **CutMix and MixUp**: CutMix [8] and MixUp [9] are two advanced data augmentation techniques that were used in addition or to substitute the other two strategies during our testing. In the end, they were not used since they introduced too much complexity and did not increase models' performance.

2.1 Test-time augmentation

To further improve our model accuracy, we also performed a self-ensemble technique called Test Time Augmentation [2]. This technique consists in creating multiple augmented copies of each image in the test set, having the model make a prediction for each, and then returning an ensemble of these predictions. This ensemble is made by aggregating the posterior vectors, keeping the class with biggest sum of the prediction.

3 Development

Here we briefly list our attempts, to highlight the strategy that brought us to the final model. Firstly, we decided to build models without any regularisation technique, to determine how much overfitting we would have had to tackle. Then, by taking into account the results of the base models, we started to add different techniques, one by one. In practice, we considered using early stopping, learning rate scheduling, dropout and batch normalization on the classification layers, plus regularisation in the form of l1 and l2 regularisation. Moreover, we focused on tuning: the learning rate of the model, the number of neurons in the hidden layers, the number of layers to freeze when using fine-tuning and the percentage of neurons to freeze when using dropout. During this period of testing, we also decided to give k-fold Cross Validation a try, but since the training time was incredibly long for even the smallest networks, we decided to scrap the idea and only rely on the aforementioned techniques.

3.1 Custom Model

The first attempt in solving this classification problem was made using a VGG16 model [5] written using keras Sequential API. One could argue that a better approach could have been to directly refer to transfer learning models, but we wanted to set a a baseline that could be compared to the results obtained with more complex models. The classification section of the model was also the classic one, made of, in order, a Flattening layer, two Dense layers of 4096 neurons with *ReLU* activations, and finally an Output layer made of two nodes with *softmax activation*. In addition, we tried to tweak the architecture by adding and removing convolutional layers.

3.2 Transfer Learning and Fine Tuning

After running some tests with our first custom model we decided to return to transfer learning, to be able to extract the image features in a better way, using models pre-trained on the *Image Net* dataset. For the classification part of the network we decided to experiment with different combinations of a different amount of fully-connected classification layers. In particular we never went over three dense layers, with number of neurons ranging from 512 to 4096.
Considering all the above, we ended up testing VGG16, which was already used to solve a similar problem [7], MobileNet [3], and Efficient Net v2 [6]. The latter brought an increase in accuracy over the other two, so we decided to keep tuning it up until we introduced ConvNeXt [4].

4 Final model

The best results were obtained using a ConvNeXtBase pre-trained model as feature extractor, initialised with ImageNet weights.

4.1 Layers

The first layer of the model is the *Input* layer, used to instantiate a keras tensor. The third is the *transfer_net* layer, which contains the ConvNeXtBase as base model but in place of its top layers we included the following layers:

- Global Average Pooling, to flatten the input using a pooling average strategy;
- Batch Normalization, to further increase stability during training;

- Dropout, with a rate of 0.4 to reduce overfitting;
- Two Dense with 1024 and 512 nodes respectively, each with a ReLU activation function and L2 regularisation, plus an *He Uniform* variance scaling initializer;
- Dropout, with a rate of 0.26 to reduce overfitting;
- Dense layer with 64 nodes, ReLU activation function and L2 regularisation, plus an *He Uniform* variance scaling initializer;
- Dense output layer with two nodes (we used one-hot encoding for the outputs) and a *softmax* activation function.

4.2 Pre-processing

No pre-processing was needed, since ConvNeXt already includes pre-processing layers in its architecture.

4.3 Training

As stated, we use the *ImageNet* weights to initialize the transfer_net and then we trained the model in two steps: first, we only trained the head of the classifier, and then we moved to fine tuning. In general, we tried to find the find the appropriate number of layers to unfreeze to get the best results possible, which were surprisingly achieved by retraining the whole model (all layers unfreezed).

4.4 Hyperparameters

- **Validation Split**: 0.1 of Training set.
- **Test Split**: 0.05 of the entire original dataset (4850 images).
- **Batch Size**: 64.
- **Max epochs**: 100, never actually reached due to early stopping.
- **Optimizer**: Adam optimizer.
- **Learning Rate**: in the first training we started from 1e-3, with a a reduction on plateau of 0.1 down to a minimum of 1e-6. In fine tuning we used a much smaller 1e-6.
- **Loss**: Categorical Cross Entropy.

5 Results

Model	Validation Accuracy (%)	Phase 1 Test Accuracy (%)	Phase 2 Test Accuracy (%)
ConvNeXtBase	91.76	\	81.70
Custom VGG16	84.00	60.00	\
Transfer VGG16	79.18	76.00	\
EfficientNetV2	87.22	82.00	\
MobileNetV2	83.47	\	\

Table 1: Validation and Test accuracy comparison

A small note: due to the congestion of CodaLab during the last day of the challenge we were not able to test the latest ConvNext model we built, which achieved over 90% on our local test set and that was expected to vastly outperform our latest submission; thus, we put as accuracy value the last evaluation we had on a very similar architecture, again a ConvNextBase but with a different combination of classification layers.

6 Contributions

Following are the contribution of each member to the homework. Note that the majority of the testing process was done with constant communication inside the team.

- Fabio: filtered the dataset, built the general model framework in the form of the class *model*, introduced various data augmentation techniques, such as CutMix and MixUp, and tested Efficient Net V2 Small and ConvNeXt Base.
- Elahe: tested and tuned MobileNet, EfficientNetV2 Large and VGG16. Regarding VGG16, applying oversampling on the minor class, followed by early stopping , data augmentation, a resizing layer to have 224*224 input shape, exponential decay and batch normalization lead to best results for the proposed VGG16 model considering both computational load as well as performance.
- Mohammadreza: explored and tested different models, including MobileNet and EfficientNet. For each model, different approaches were employed to maximize performance. Additionally, a hand-crafted method was experimented with, but it did not yield favorable results.
- Simone: explored the dataset in .png and ensured the dimension was right. Started the modelling by tuning a MobileNet implementation, following what was done in class, then passed to EfficientNetV2 Medium and finally ConvNeXt Large. Introduced the test split for the second phase and tested the ImageDataGenerator Data Augmentation technique. Explored differences between the use of the given seed and a random seed, and finally researched techniques to prevent overfitting.

References

[1] Classification on imbalanced data. `https://www.tensorflow.org/tutorials/structured_data/imbalanced_data`. Accessed: 2022-11-28.

[2] How to use test-time augmentation to make better predictions. `https://machinelearningmastery.com/how-to-use-test-time-augmentation-to-improve-model-performance-for-image-classificatio` Accessed: 2022-11-28.

[3] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL `http://arxiv.org/abs/1704.04861`.

[4] Z. Liu, H. Mao, C.-Y. Wu, C. Feichtenhofer, T. Darrell, and S. Xie. A convnet for the 2020s, 2022. URL `https://arxiv.org/abs/2201.03545`.

[5] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition, 2014. URL `https://arxiv.org/abs/1409.1556`.

[6] M. Tan and Q. V. Le. Efficientnetv2: Smaller models and faster training. 2021. doi: 10.48550/ARXIV.2104.00298. URL `https://arxiv.org/abs/2104.00298`.

[7] W. Wenxuan, W. Qianshu, H. Chaofan, S. Xizhe, B. Ruiming, and T. T. Toe. Leaf disease image classification method based on improved convolutional neural network. In *2022 IEEE International Conference on Industry 4.0, Artificial Intelligence, and Communications Technology (IAICT)*, pages 210–216, 2022. doi: 10.1109/IAICT55358.2022.9887392.

[8] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo. Cutmix: Regularization strategy to train strong classifiers with localizable features, 2019. URL `https://arxiv.org/abs/1905.04899`.

[9] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz. mixup: Beyond empirical risk minimization, 2017. URL `https://arxiv.org/abs/1710.09412`.