Artificial Neural Network & Deep Learning

Image Classification Homework - First Challenge

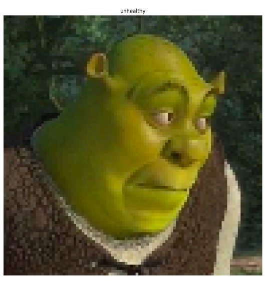
Introduction

The task involves classifying plants into two categories based on their health status using a binary classification network. The dataset consists of 5200 RGB images, each measuring 96x96 pixels. We adopted a systematic approach to tackle the challenge, starting with the LeNet architecture and gradually moving to more complex convolutional neural network (CNN) architectures, ultimately utilizing EfficientNet, to establish a baseline, analyze performance, and introduce more complex architectures for improved classification accuracy.

Data Inspection and Preprocessing

Outliers Removal

After loading the dataset, we inspected the pictures plotting a few samples. We immediately noticed the presence of two types of outliers (definitely not plants), for a total of 196 pictures, which we removed in order to train our net with samples more suitable to the problem faced. The removal was performed by visual inspection, then we used a script to detect all the duplicates and remove them.



Class Balancing

After a quick inspection on the proportion of the training set classes, we noticed that there was a strong imbalance: the healthy pictures were 62.22%, meanwhile the percentage of the unhealthy pictures was 37.78%. Inspecting the confusion matrix made us notice that predictions performed with the earliest architectures were erroneous mostly on unhealthy images. To mitigate mispredictions on the less represented class, we decided to balance them by oversampling the unhealthy pictures, instead of undersampling the healthy class, due to the not so big dataset. Performances increased as shown in the table below.

Architecture	**Accuracy**	**Precision**	**Recall**	**F1**	**Optimizer**	**Classes**	**Submission**
EfficientNetB4	0,8589	0,8801	0,8336	0,8457	Adam	Unbalanced	0,88
EfficientNetB4	0,8708	0,8852	0,8499	0,8605	Adam	Balanced	0,9

Another attempt to balance the data was made by using *class_weight*, which applies different weights to the healthy and unhealthy classes. Unfortunately, no significant improvement was achieved, so we opted for the oversampling procedure.

Work environment set-up & Image Augmentation

After the necessary pre-processing, we moved on to building an end-to-end training environment for the networks. We started by fixing the random seed in order to ensure a certain level of reproducibility, then splitted the available dataset in 3 parts: training, validation and test set using different dimensions for the partition (mostly 80%, 10%, 10%), being careful to shuffle the dataset and preserving the data distribution between the sets.

After testing different image augmentation techniques, we found the most suitable augmentation layer for our problem, namely a combination of Random Rotation, with a factor of 0.15, Random Translation, with both height and width factor of 0.1, Random Flipping, Random Contrast, with a factor of 0.1. We also tried to preprocess the images with a Global Contrast Normalization and a Brightness layer but we didn’t notice any performance gain.

Architectures

Ad Hoc Architecture, LeNet, AlexNet & VGG

We first drew inspiration from a suggested CNN model on plant classification to create an ad-hoc architecture for our problem, with three sets of 2D convolutional layers, succeeded by batch normalization, ReLU layers and max-pooling, ending with a softmax classifier.

As LeNet is a famous CNN architecture, we chose it as our second try, training the network from zero, using Adam with default settings. Even after some regularization, hyperparameter tuning and data augmentation the results were not satisfactory, yielding only 66% accuracy on the submission test set, a symptom that the architecture was not able to generalize well.

Then we proceeded with AlexNet, which was built from scratch too, creating 5 convolutional layers, some of which were followed by max-pooling layers, and 3 fully connected layers with a final 2-way softmax. We have observed through previous experiments with LeNet that CNNs with ReLU as the activation function train faster and yield superior results compared to their alternatives with tanh and sigmoid, so in our architectures we will mainly apply the ReLU non-linearity to the output of each convolutional and fully-connected layer. The size of the network posed a significant problem with overfitting and to reduce this problem we implemented regularization with dropout. This approach proved effective in comparison to previous attempts. The network reached only 68% accuracy.

Characteristic of VGG16 and VGG19 is the use of 3x3 convolutional filters in cascade to increase the depth of the architecture, introducing more non-linearity and creating more expressive and discriminative functions. This approach also decreases the number of parameters and reduces the risk of overfitting. However, we didn’t get good enough results with this architecture, even with transfer learning.

Transfer Learning & Fine-Tuning with EfficientNet

After obtaining suboptimal results with previous architectures, we turned to transfer learning and fine-tuning, techniques that are proved to be effective for small datasets. As a base model we opted for the EfficientNet families of networks since they are more accurate and smaller, therefore less computationally expensive, exploiting the uniform and balanced scaling of the network.

We exploited Keras pre-trained weights (ImageNet dataset), first by freezing the backbone of the network and building new top layers (GAP layer, Batch Normalization layer, Dropout layer, Dense layer) functioning as the classifier layer; we put only a dense layer connecting the GAP to the two output neurons because since the topology of the FC was not really much influential on the end result, but choosing how many layers to unfreeze was more important. To correctly process images, we also added on top of everything a resizing layer, responsible to scale images to the dimensions expected from the network (*i.e.* 380x380 for B6). We proceed by training the model with 50 epochs, then as a second step, we fine-tuned, by unfreezing the top 20 layers while leaving the Batch Normalization layers frozen. The number of layers to unfreeze has been chosen by being careful to unfreeze a whole logic block of the network; we found that the range between 20-40 layers (trying to unfreeze an entire logical block of the network) was the optimal number for our use case. At this point we trained for another 50 epochs and observed a 6-8% gain in accuracy on the validation set. Obtaining a 90% accuracy score on the submission.

To test the most suitable network, we began from the B0 up to the B6 network, trying for each solution different combinations in terms of optimizers (mostly Adam vs AdamW). B0 network has shown an improvement with respect to other architectures, leading to good results even before class balancing and outliers removal; B4 showed further improvements on every metric, so we proceeded to fine tune it searching for an adequate learning rate and regularization strategies. On the contrary, B6 increased performances locally, but worsened them on the private remote test set, stopping us from trying more complex architectures.

Ensemble

When submitting for the second phase of the challenge our best model dropped 12 percentage points in accuracy falling, to 78%. One of the reasons for this drop might be that the model was skewed towards the data of the first test set, so in reality it wasn't that good at generalizing. At this point, to improve the generalization capabilities of the model, we decided to try an ensemble of two networks: one trained with a balanced data set, and one with an unbalanced one. The training for this network was done with different hyperparameters and with different shuffling of the dataset. The result was an increase in accuracy up to 82%.

ConvNeXtLarge

In the latter stages of the challenge, we also decided to try another architecture: ConvNeXt, since according to the latest studies it performs very well in solving binary classification problems.
Our network is built by sequencing the input layer, a data augmentation layer, a data preprocessing layer compatible with the network architecture, then the whole ConvNeXtLarge deprived of its last layers, which we rebuilt with a FC layer according to our needs.

During training of the classifier, the ConvNeXtLarge model prioritized identifying significant features from images in the training set, which had undergone transformation through augmentation tools. At this stage, the accuracy of the network saturated at almost 80%. The fine-tuning phase was conducted by initializing the first epoch as the best epoch of the training phase (the one with highest validation accuracy) so as to prevent the algorithm from reprocessing the same images already processed in the training phase, and we also unfreezed the entire model, except for the BatchNormalization layers. As we previously did with all the other experiments, we set the learning rate to be one order of magnitude lower during fine-tuning than during the initial training phase. At this stage, fine-tuning achieved an accuracy of about 95% on the validation set over about 45 epochs and achieved a score of 86.5% in accuracy on the test set.

Contributions

- **Irfan Cela** (@IrfEazy - 10694934): outliers removal, tests and hyperparameters tuning on ConvNeXt (Base and Large), data augmentation through CutMix and MixUp, incorporating caching, shuffling, and prefetching to enhance training efficiency.
- **Alessandro Cogollo** (@Frigobar123 - 10571078): performed class balancing, tests and hyperparameters tuning on EfficientNets (B0, B4, B6, and some of the V2 family), tests on transfer learning and fine-tuning.
- **Fabio Lusha** (@AlexOberstein - 10882532): LeNet, inception architecture, hyperparameters tuning with EfficientNet (B0 and B4), ablation studies on EfficienNet (B0 and B4) about the efficacy of freezing/unfreezing layers in block
- **Bianca Christiana Savoiu Marinas** (@BexRedpill - 10684465): LeNet, AlexNet, Ad Hoc CNN, data augmentation exploration using SMOTE, hyperparameters tuning with EfficientNet, ensembling to mitigate overfitting.

References

- EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. ICML 2019, Mingxing Tan, Quoc V. Le.
- Going Deeper with Convolutions. 2014, Christian Szegedy, Wei Liu, Yangqing Jia et al.
- ImageNet Classification with Deep Convolutional Neural Networks. 2012, Alex Krizhevsky, Geoffrey E. Hinton, Ilya Sutskever.