Artificial Neural Networks and Deep Learning - Homework 1 Report

Alberto Aniballi, Lorenzo Campana, Michele Pio Prencipe (Group: Alan Tuning)

November 13, 2023

1 Dataset inspection

The first activity we performed to complete the task was to analyze the dataset trying to understand what the nature of the problem was to solve, whether there was an imbalance between the classes and whether there were any outlier images that could compromise the learning process of our model. The problem to be solved was classification of 5200 plant images with a resolution of (96 x 96 pixels) and three color channels. The classification was binary so it was clear that the target classes were only two, 'healthy' and 'unhealthy': the dataset consisted of 3199 occurrences of the class 'healthy' and 2001 occurrences of the class 'unhealthy'. Although the images were not very large, the object to be classified, the plant, occupied a central role in the images, very often covering the entire available space with very good lighting conditions, little space was left for the background, which therefore did not influence the successful classification of the plant. The two classes were characterized by a slight imbalance in favor of the "healthy" class however we initially decided not to perform oversampling of the minority class as we first wanted to test whether the developed models were able to learn the relationship between the image and the target without being affected by the introduction of duplicates of the "unhealthy" class thus reducing the risk of overfitting. Visualizing the dataset, it was clear that outliers were present; in particular, there were 196 outlier images attributable to two distinct groups: 98 identical images depicting the protagonist of the famous shrek cartoon and 98 identical images depicting Eduard Khil, the author of the well-known song "Trololo". To eliminate these outliers, we performed two procedures, the first of deterministic algorithmic type while the second of clustering type, verifying that both produced the same result thus confirming their correctness.

2 Outliers Removal

The first outlier detection technique is based on a deterministic procedure that leverages the fact that within the two outlier groups the images were identical to each other. Once we identified one image of “Shrek” protagonist and one of Eduard Khil, we defined a Boolean mask that returns true if each image in the dataset is equal to one of the two different outlier images; this mask makes it easy to find the indices of the images to be removed. The second technique used to identify outliers is of the clustering type and is called DBSCAN, Density-Based Spatial Clustering of Applications with Noise. This clustering technique works on vectors, so before using it we used a pre-trained model to flatten each image into its feature vector. The pre-trained model chosen to extract the feature vector was ConvNextBase. DBSCAN defines clusters based on the density of points in a specific area, the region in which the two groups of outlier images are located turns out to be of high density and therefore can be clustered separately from the other images, since the 98 images in one group and the other are formed by vectors of pixels identical to each other and therefore are located in the same area of hyperdimensional space. After a short period of parameter selection, DBSCAN was able to successfully identify in two specific clusters the two groups of outliers.
Once the outliers were removed, we used stratified sampling to divide the dataset into training, validation and test set. Using the training set to learn the model, while the validation set to evaluate its performance and at the same time limiting overfitting by using early stopping technique.

3 Models Development

We initially decided to build several CNNs together from scratch to test the effect of introducing new layers on the performance of the model (section 3.1). After reaching a threshold of validation accuracy beyond which our 'custom' models could not go, we started working in parallel using pre-trained models (section 3.3). In particular, in this second phase we focused on using hyper parameter tuning and overfitting reduction techniques to try to improve our results. Before developing each model, we normalized the dataset, dividing each sample by 255.

3.1 Custom Models

The first CNNs developed consisted of a low number of layers: we used various convolution layers combined with max pooling and activation layers, reducing the spatial dimensions of our image each time, thus obtaining a latent representation, or feature vector, of our initial image. Following advice received in class, our activation layers introduce Relu or Leaky Relu non-linearities in order to avoid the well-known vanishing gradient problem. Once the latent representation was obtained, dense, or fully connected layers were added, with the task of classifying the feature vector as healthy or unhealthy. These models were simple and as we expected the training phase was quick, however we still decided to use validation accuracy to trigger early stopping in case the performance on the metric did not improve within a predefined number of epochs. In doing so, we never reached the 200 epochs initially set for training. Once we saw that these first CNNs failed to exceed the threshold of 0.65 in val_accuracy, we decided to implement more complex custom models by combining multiple parallel convolution branches each with a different number of filters and a final GlobalAveragingPooling layer that would allow us to aggregate the result of multiple parallel convolution branches into a single vector then passed to the classification network. We used this parallel approach since it could improve the generalization ability of the model as the network leverages several latent representations arising from each branch. Initially, the parallel blocks consisted of a 2d convolution layer, followed by a Batch Normalization layer so as to alleviate the internal covariate shift problem and at the same time stabilize learning, then an activation layer followed by dropout to try to prevent overfitting and thus improve the generalization ability of the model. This first parallel approach succeeded in exceeding the threshold of 0.65 in validation accuracy by reaching 0.71, however, we noticed that after a few tens of epochs the model could no longer learn despite being much deeper than the initial simple networks. Therefore, we combined the parallel convolution approach with the residual connection technique. Residual connections in fact help combat the vanishing gradient phenomenon by using connection shortcuts that avoid passing through certain layers, allowing the gradient to flow more directly. This new model using parallel convolutional blocks containing residual connections achieved an accuracy of 0.76 in our validation set. Unfortunately, the performance achieved in the training set was higher, which directed us towards the fact that the CNN was overfitting the data, so we implemented a further technique called Data Augmentation that introduced further variability to the training data set by generating new images through transformations of the original data set. Before applying this technique, it is important to realize that the new images must still retain their original label so it is necessary to understand what type of transformation preserves the label. Images could certainly be rotated, flipped vertically and horizontally without losing important information about their class. Within the image, the plant tends to occupy almost all the space, so we thought that zooming in would not improve the model, as each portion of space could potentially contain information relevant to the label and zooming in could cause many parts of the plant to be lost. At a later stage, see section 3.2, we also decided to apply new augmentation techniques such as CutMix() and MixUp(). After performing augmentation, on the one hand we recorded improvements as CNN probably improved its ability to be invariant to the transformations applied during training, on the other hand increasing the number of images highlighted the inadequacy of our model to learn optimally as overfitting decreased however validation accuracy also decreased returning to the levels of 0.71. In the face of such evidence we decided to start using an approach based on transfer learning and fine tuning of much more complex models pre-trained on the imagenet dataset, this would allow us to give a boost to our performance.

3.2 Introduction of Focal Cross Entropy

Before delving into transfer learning we wanted to address the problem of the slight imbalance between the two classes, as previously mentioned 3199 samples belonged to the healthy class and 2100 to the unhealthy class. Looking at the metrics of our models, we noticed that the models had similar accuracy, precision and F1 scores, while recall was lower. This implies that our network struggled more to identify all positive images, unhealthy in our case, and had a higher false negative value, thus identifying healthy plants even if the true label was actually unhealthy. For this reason we decided to start using a different loss function: Categorical Focal Cross Entropy. This loss function allows us to address the problem of slightly unbalanced classes and put the focus of our model on images that are more difficult to classify, as it assigns a higher weight to misclassified samples thereby potentially improving recall without significantly compromising other metrics.

3.3 Transfer Learning

The first pre-trained model we chose was DenseNet in two variants proposed by keras: DenseNet121 and DenseNet201. This architecture is particular in that each layer of the network is connected to the layers after it in a feed-forward fashion guaranteeing a high flow of information between the different layers, also facilitating the flow of the gradient by reducing the vanishing gradient problem. We built the first model by freezing the weights of the pre-trained network and adding two fully connected layers and a dropout to the end of it. The validation accuracy improved significantly compared to our custom models, reaching values above 0.8 after a few epochs. After finishing the first transfer learning procedure, we saved the model, reloaded it and proceeded with the implementation of fine-tuning, making the weights of the last 50 layers of DenseNet121 trainable. This fine tuning allowed us to achieve a validation_accuracy of 0.9122 and a score of 0.83 on Codalab. We then moved on to the transfer learning of an even more powerful architecture, EfficientNet, again testing different versions: EfficientNetV2B0, EfficientNetV2B1, EfficientNetV2S and EfficientNetV2M. The approach taken was the same as described above for DenseNet, this time we lowered the learning rate at each new tuning depth. The best result was produced by EfficientNetV2S with a fine tuning of 40 layers, value of validation accuracy achieved was 0.9162, an overall result of 0.84 on Codalab. Before moving to a new architectural network, we delved into hyperparameter tuning of our current networks. For each new pre-trained model, we tested various combinations of learning rate, batch size, weights initialization technique, dropout rates and whether or not to use Gaussian noise within our classification model. It turned out that the best parameters for EfficientNet were: 1e-4 as learning rate, 16 of batch size, he_uniform for initialization, 0.3 dropout rates and gaussian noise with 0.1 standard deviation of the distribution. In the fine-tuning stage, the learning rate was lowered to 1e-5. Then, we implemented transfer learning and fine tuning with a final architecture: ConvNext.
After several attempts at improving the models by doing hyperparameter tuning, we took an alternative route that would allow us to further reduce the overfitting of our CNN to the training dataset: implementing new data augmentation techniques, specifically CutMix and MixUp. CutMix was removing small regions, pixels, of the original image and replacing them with patches of pixels from other images. These small cuts prevent the CNN from focusing too much on some small regions of the input image by introducing random pixel removal and thus training the model to identify the image using a partial view of it and soiled with pixels from another image. MixUp combines the different features of the image and at the same time interpolates their labels as well so that the network does not become too sure of the relationship between the features and their labels.The best result was recorded using ConvNextBase with fine tuning of 150 layers: 0.93 in validation accuracy and 0.91 overall score on Codalab. In the final stage of the competition this same model instead achieved a score of 0.85 on Codalab. To increase the score we decided to try a new technique as explained below.
Finally, we implemented an ensembling technique assuming that using multiple models to classify images would reduce the variance of the prediction by containing the errors or biases that each CNN taken individually might exhibit. We divided the ensembling approach into four steps. First step is the choice of models: we used ConvNextSmall, EfficientNetV2L, ConvNextLarge. Second step is the summation of the predictions that each convolutional architecture makes; in our case, we also assigned an importance weight to each model based on the validation accuracy that each had individually. This weight is multiplied to the prediction of each network: weight of 0.4 at ConvNextSmall, weight of 0.2 at EfficientNetV2L, weight of 0.4 at ConvNextLarge. The predictions were then summed element-wise. Third step is to make the final prediction: this was done using argmax, then selecting the class with the highest value. This ensembling technique achieved an overall score of 0.86 on Codalab during the final phase, proving to be the best predictor so far. We know that using less correlated models would have been better, however, we did not have the computational resources to tow so many different architectures.
As a final experiment, we performed 5-fold cross validation: for each model used in ensembling we performed five times fine tuning with a different training set and validation each time. In each of these five trainings, the model stopped at a different number of epochs. We computed the average of these five epoch numbers and then trained the model on the entire dataset stopping at the average epoch number just computed without using early stopping since we did not have a validation set. Finally we used these trained models on the whole dataset to perform ensembling again.

CONTRIBUTIONS

Alberto Aniballi

-Led the dataset inspection phase and implemented outlier detection technique
-Contributed to the development of custom CNN models
-Collaborated in the implementation of the ensembling technique
-Structured the report, ensuring a clear presentation of the workflow
-Performed fine-tuning on DenseNet and EfficientNet

Lorenzo Campana

-Coordinated the organization of the team
-Implemented the data augmentation techniques and unsupervised outlier detection (DBSCAN)
-Implemented the various pre-trained versions of ConvNext and DensNet and finally fine-tuned them.
-Experimented with hyperparameter tuning for various models, optimizing learning rates, batch sizes, and initialization techniques.
-Implemented the introduction of Focal Cross Entropy to address class imbalance
-Collaborated in the implementation of the ensembling technique

Michele Pio Prencipe

-Participated in the initial dataset inspection
-Implemented the various pre-trained versions of EfficientNet
-Engaged in the analysis of the competition results, refining models based on performance feedback.
-Collaborated in the implementation of the ensembling technique