REPORT HOMEWORK 1 – ARTIFICIAL NEURAL NETWORK AND DEEP LEARNING

Gatti Carlotta – Montenegro Martina – Zanella Francesco

INTRODUCTION:

This report explains step by step how our final model shaped itself, showing the reasoning we followed to pre-process data and to build architectures by hand. It also points out the techniques we used to exploit previous networks found in the literature.

DATA ANALYSIS AND PREPROCESSING:

Pre-processing data is almost a mandatory first step in deep learning, in order to prepare raw data to be trained in the model. For instance, when we deal with convolutional layers, all images must be of the same size, we may also want to raise image quality or, by shrinking the size, to decrease the amount of time needed to train the model. Another example could be to flip or rotate images to introduce variance in the dataset.

Normalization is fundamental - in fact, our goal in normalizing the set of data was rescaling it such that all pixel values were between 0 and 1, thus helping the training of neural networks since different features are on a similar scale and – therefore – gradient descent is stabilized. As a consequence, we can use larger learning rates, we help the model to converge faster and, we reduce overfitting (since normalization also acts a regularization factor). Without normalization the input layer could have certain features which dominate the process, due to the high numerical values, creating bias in the network (only those features contribute to the outcome).

In the image domain, it means dividing each pixel by 255 – also avoiding exploding gradient descent.

It is also useful to use the layer of Batch Normalization, which standardizes the input layer across a single batch to reduce the problem of internal covariate shift, meaning that the distribution of each layer's inputs changes during training since parameters change. Still, it can help in using a higher learning rate and regularizing the model. This process consists in calculating the mean and variance for each of the different activations across a batch, then the mean is subtracted, and each feature is divided by the standard deviation. After that, a scale and a shift of the data are done, to introduce new learnable new parameters. Note that by standardizing the output of a layer we standardize the inputs to the next layer. Drawbacks could arise combining it with DropOut, since both have the regularization effect, or with too small batch sizes: the quality of the statistics calculated is affected by the batch size.

Using normalization, we got better accuracy values. It was particularly pointed out while building the base model, in which we experimented different solutions. While, regarding Batch Normalization, it increased convergence time in the base model.

We further noticed that classes in the proposed dataset were not only low represented, but also strongly imbalanced - few samples and badly distributed among classes. Therefore, data augmentation was needed. The problem of having imbalanced classes [1] stands in biases linked to the learner with respect to the most represented classes, ignoring the minorities (i.e. in our case: class 1 was strongly low sampled and also class 6), thus being misclassified more often (highlighted by the confusion matrix we built).

Different data augmentation techniques have been tried in order to find the best combination: Random Flip, Rotation, Zoom and Contrast (decision was supported by [2]). The first three were implemented to increase the number of samples, while the last one to improve the quality of data (since after analysing images we noticed there was a low contrast).

In order to address the problem of class 1 and class 6, we tried to augment more only these classes. Unluckily it did not make remarkable improvements in the accuracy outcome.

BUILDING THE BASE MODEL:

After data augmentation and pre-processing, we tried to build a simple model based on three convolutional layers and three fully connected layers. The architecture from which we started is the same present in the final upload, upon which we implemented different regularization techniques to improve the initially low results. In fact, at first, the outcome was accounting for an overfitting pattern: on one hand training loss was approaching zero, on the other hand the validation one was increasing.

Therefore, we added regularization parameters: the goal is to constrain model freedom, making slight modifications in the algorithm, so that the model can generalize better, by penalizing the weights (adding a term to the loss function leads to the decrease of weights and thus to simpler networks less prone to overfitting). We tried both Lasso and Ridge regression. Nevertheless, DropOut proved to be better, in fact it is the most frequently used regularization technique, since, statistically, it produces better results.

At each iteration, it randomly selects some nodes and removes them and their connections. Therefore, it could remove informative pixels. Still, better accuracy values were obtained through DropOut.

During training, early stopping has been applied – it stops the training when the performance on validation is worse than the one on training, out of a dead band of acceptance. We can tune the patience, i.e. the number of epochs where we see no further improvements and, therefore, after which the training is stopped.

Hyperparameters were chosen by taking into consideration that when the learning rate is too small it leads to overfitting and when too large makes the training diverge. We tried to change the learning rate, decreasing it after epoch 75 and after epoch 200 of training, seeing no encouraging improvements. The explanation we found was in the intrinsic nature of the optimizer we used, i.e. Adam, which is adaptive by itself.

One thing that has not been done is setting boundaries for the learning rate and making it cyclically vary between boundaries.

Seeing the performance, one step more was trying to tune the hyperparameters of the network, no more by hand, but through the function Keras Tuner: this was never fully exploited due to time constraints, but the partial results (3 dense layers, 0.3 as dropout parameter) were added to the 'Base Model' allowing us to better the performance on the accuracy.

In the end, Global Average layer was added since fully connected are prone to overfitting (they have a huge number of parameters), thus enforcing correspondences between feature maps and categories and reducing number of parameters. This led to more interpretability, regularization and increased robustness to spatial transformation. By doing this, we got further improvement in the accuracy of validation.

Overall, we reached the final validation accuracy of the 'Base Model' of about 0.76.

TRANSFER LEARNING AND FINE-TUNING:

Going deeper into our work, we implemented Transfer Learning and Fine Tuning.

Transfer Learning consists in re-using models trained very well on a huge amount of data. It assumes that training from scratch would be time-consuming on a smaller dataset, such as the one we have for homework. Usually, fine-tuning is applied after transfer learning: unfreezing some layers (optimal number chosen by a trial-and-error procedure) to be re-trained having as initialization weights the ones of the original models. [3]

We tried different famous architectures which distinguished themselves for their performances in the past years to improve robustness, since: being deeper could improve the performance (more local minima with good performance leads to ease the location of acceptable solutions).

We first tried VGG16 which consists of 13 convolution layers, 3 dense layers, pooling and three ReLu units. It is not so deep when compared to other architectures and, moreover, it has many more parameters. In fact, using the feature extraction part of VGG16 not retrained and the fully connected of the Base Model, without re-tuning hyperparameters, led to suboptimal results.

Therefore, we wanted to exploit deeper layers, which are not affordable without residual connection or infinite time to train the network. In fact, we tried ResNet50 and Xception: we focused more on these two, which have fewer parameters (VGG16 – 138,4 mln, ResNet50 – 25,6 mln, Xception – 22,9 mln) and which are deeper. The goal was improving the performance through re-training also part of the convolutional layers, since these networks have been trained to extract features on a different type of dataset with respect to ours.

Resnet50 was first trained completely frozen, adding a dense layer (256 units), Batch Normalization, DropOut (0.3) and one dense layer as output layer (therefore 8 units for 8 classes). We named the parameters since they have been tuned through Keras Tuner. After, unfreezing the last 33 layers (15.503.512 trainable parameters) we obtained the best result (on validation 0.8192).

We used the same approach with Xception: we first trained the model using our Base Model for the dense part. Afterwards, starting from the weights previously obtained, we unfroze both models: we firstly re-trained all the layers and then only the last 35 layers. In the end we obtained an accuracy of 0.81 on the validation data.

ENSEMBLE MODELS

The final step of our work was to ensemble learning methods to improve performances, thanks to the fact that it helps reduce the variance of Neural Networks. In fact, Neural Networks are non-linear methods able to learn complex relationships, thus being sensitive to initial weights and initial noise. [4]

The procedure adopted was to simply stack models in a single tensor in ‘model.py’ and extract the best prediction among them.

CONCLUSION

We ensembled the models with the best results: ResNet, Xception (fine-tuned as described above) and the Base Model (exploiting the hyper-parameters optimized by Keras Tuner, DropOut as a regularization term and Global Average Pooling).

The final result on the test set provided is 0.79.

REFERENCES

[1] Johnson, J.M., Khoshgoftaar, T.M. Survey on deep learning with class imbalance. J Big Data 6, 27 (2019). https://doi.org/10.1186/s40537-019-0192-5

[2] https://www.ai.rug.nl/~mwiering/GROUP/ARTICLES/plantaugment.pdf

[3] Alhawas, N. & Tüfekci, Z. (2022). The Effectiveness of Transfer Learning and Fine-Tuning Approach for Automated Mango Variety Classification. European Journal of Science and Technology, (34), 344-353.

[4] https://machinelearningmastery.com/ensemble-methods-for-deep-learning-neural-networks/