Artificial Neural Networks and Deep Learning 2022
Homework 1

Team: All Is Well
Fatma Hamila - 10913201, Kodai Takigawa - 10808008, Zheng Maria Yu - 10596129

1. Introduction

The goal of this homework is to classify images of plants belonging to 8 species. It is a supervised task given that all images are labeled with the corresponding class. We tried different approaches to improve the classification results and different methods to overcome some obstacles that we will discuss in this report.

2. Data splitting

The given dataset consisted of 3542 RGB images (size 96x96), divided into 8 classes. Generally, we splitted the whole dataset into a training set (80%) and a validation set (20%) preserving the distribution of classes. Sometimes, we used a splitting ratio of 80%:10%:10% (training:validation:test) to evaluate the performance locally.

3. Data preprocessing

The first step was to preprocess the data according to the type of the model used. Firstly, the images are loaded with ImageDataGenerator. For models implemented from scratch, the value of pixels are normalized to [0, 1] through the rescale function. When using pretrained models instead, we applied their corresponding built-in preprocessing function from keras.applications.

During the training of models, we realized that the number of samples were insufficient and the dataset was imbalanced. In order to deal with the data scarcity issue, we resorted to data augmentation methods to artificially increase the number of samples. At the beginning, the augmented samples were generated through ImageDataGenerator, but then we decided to use the Albumentations library for its higher variety of operations and efficiency as well. The main transformations we applied were HorizontalFlip, VerticalFlip, Transpose, Rotate, RandomBrightnessContrast, GaussNoise and CoarseDropout. They successfully helped us to improve the models' validation accuracy and to prevent overfitting due to the high variety, but they also led to the decrease of the training accuracy and the convergence speed.

4. Model structure

4.1. Handmade models

At first we approached this multi-class classification problem by implementing convolutional neural networks from scratch, as learned during the practical sessions. We noticed that too simple models with few hidden layers were not able to capture the features well, while too deep models struggled in learning. In the end, the best model we built is composed by:

- Input layer: the required input image size is 96x96.

- Feature extractor: 5 stacks of Conv2D + MaxPooling2D + BatchNormalization layers, with ReLU as the activation function.
- Flatten layer: to flatten the input into a one-dimensional vector.
- Classifier: a dense layer with 128 units, followed by the output layer of size 8 (equal to the number of classes).

4.2. Pretrained models

We also handled the problem through transfer learning, by importing some popular architectures with pre-trained Imagenet weights from keras.applications.
The pretrained models we used were VGG19, ResNet50V2, MobileNet50V2, EfficientNetB7, EfficientNetV2B0 and EfficientNetV2B1. Considering their performance and the computational time and resources they required, we decided to keep using EfficientNetV2B0 since it represents a good compromise, and our final model was based on it. Moreover, after adding a Resize layer to (224, 224, 3) to the final model, we saw that it performed better: it might be due to the fact that it was trained on this image shape.
The pretrained models were integrated to our fully connected layers, and we tried to train them in different ways:

- Freeze the entire pretrained network for the training, then unfreeze it and use a lower learning rate to fine tune.
- Freeze / unfreeze only some layers of the pretrained network.
- Keep the entire model trainable.

We observed that the first approach gave us worse results than the other ones.

5. Model Training

We kept adjusting the training settings and tuning the hyperparameters with respect to the model type, and the monitoring indicators obtained in previous sessions.

- Batch size: A batch size of 100-128 worked with the handmade models, while its value was decreased to 16-64 if deep pretrained models were used.
- Optimizer & learning rate: Generally we chose Adam as optimizer, which worked well with a learning rate between 1e-3 and 1e-5. The default value of epsilon 1e-7 was changed to 1e-1 when using pretrained models, because greater values of epsilon seemed to make the weight updates smaller, so the training became more stable.
- Loss function: Categorical cross entropy was used during training. Later on, we decided to fine-tune the best models with categorical focal loss to compensate for the class imbalance. It allowed us to slightly improve the total accuracy.
- Scheduler: In the last few days, we used the Cosine Annealing Scheduler in training our final model. It determined periodical variations of the learning rate.

6. Model Evaluation

In order to achieve better results, many experiments were carried out to find valid techniques.

6.1. Ensemble Learning

It consists in solving a problem by using more neural networks jointly, and here we will describe our approaches.

- Building a binary classification model for Species1: since the F1 score of Species1 was much lower than the others, we had the idea to build a binary classification model to predict Species1 only. We would use its output when this model classified the image as Species1, otherwise the prediction of the main model would be considered. The 7 remaining species were grouped into the same class "Others" for training purposes. We tried to use different structures, also adapting our best multi-class classification model by changing the output layer's units, or using transfer learning and fine tuning. We also implemented the undersampling by matching the number of samples in the two classes, given their evident imbalance (186 samples for Species1 and 3356 for Others).

  Unfortunately, its validation accuracy (around 70-75%) remained lower than the multi-class models, maybe due to the scarcity and irregularity of Species1 features.

- Ensembling two different architectures: We ensembled two well-performing models with different architectures: one was based on EfficientNet, and another one was made from scratch. Then their predictions were summed up to output the class having the highest probability. Considering that the two models were trained on the same data split, there were no great variations in the results.

- Ensembling models obtained from K-Fold cross validation: Splitting the dataset into 5 folds, we trained 5 different models based on the K-Fold cross validation technique. Their predictions were averaged later on. This approach gave us the best result.

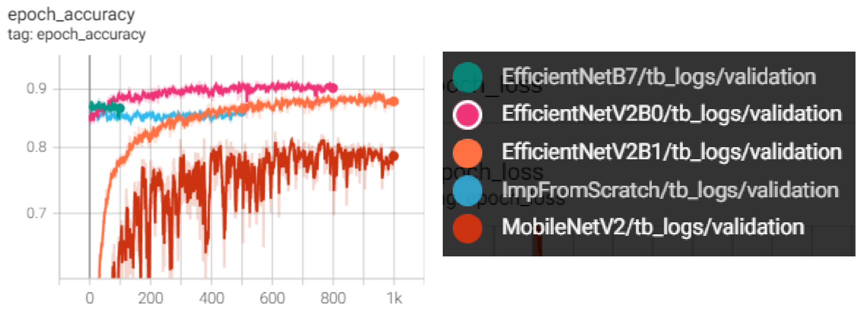
6.2. Test Time Augmentation

An attempt of test time augmentation is realized by applying horizontal flip, vertical flip and transposition for each input in model.py, after the preprocessing steps. The final guess is determined by averaging the predictions of the original image and the transformed ones.

7. Results & Conclusion

7.1. Validation accuracy graph

Our main models' training tendency comparison is plotted here by Tensorboard. Note: some models were not saved at the first epochs, to reduce the training time.



7.2. Conclusion

By keeping learning about neural networks and training with different approaches, our models reached satisfactory testing accuracies in both phases on CodaLab platform: the best handmade model achieved 0.8626 in Phase 1 and 0.8484 in Phase 2, while the best ensemble model based on EfficientNetV2B0 achieved 0.9119 and 0.9065 respectively.

In particular, we noticed that the use of data augmentations had big impacts on the models, and deeper models did not necessarily lead to better performance.