Homework 01

ELE 2346 - DEEP LEARNING

Pedro Henrique Cardoso Paulo
pedrorjpaulo.phcp@gmail.com

Professor: Raul Queiroz Feitosa

Departamento de Engenharia Mecânica
PUC-RJ Pontifícia Universidade Católica do Rio de Janeiro
April, 2023

Homework 01

ELE 2346 - DEEP LEARNING

Pedro Henrique Cardoso Paulo

April, 2023

1 Introduction

1.1 Objectives

The main objectives of this exercice is to provide the students some experience with:

- Convolutional Neural Networks (CNNs)
- VGG16
- Transfer learning

1.2 Dataset

The ibeans dataset is a dataset containing images of leaves classified according to their health condition. The images are splitted into 3 different classes (angular leaf spot, bean rust and healthy) as shown in Figure 1.

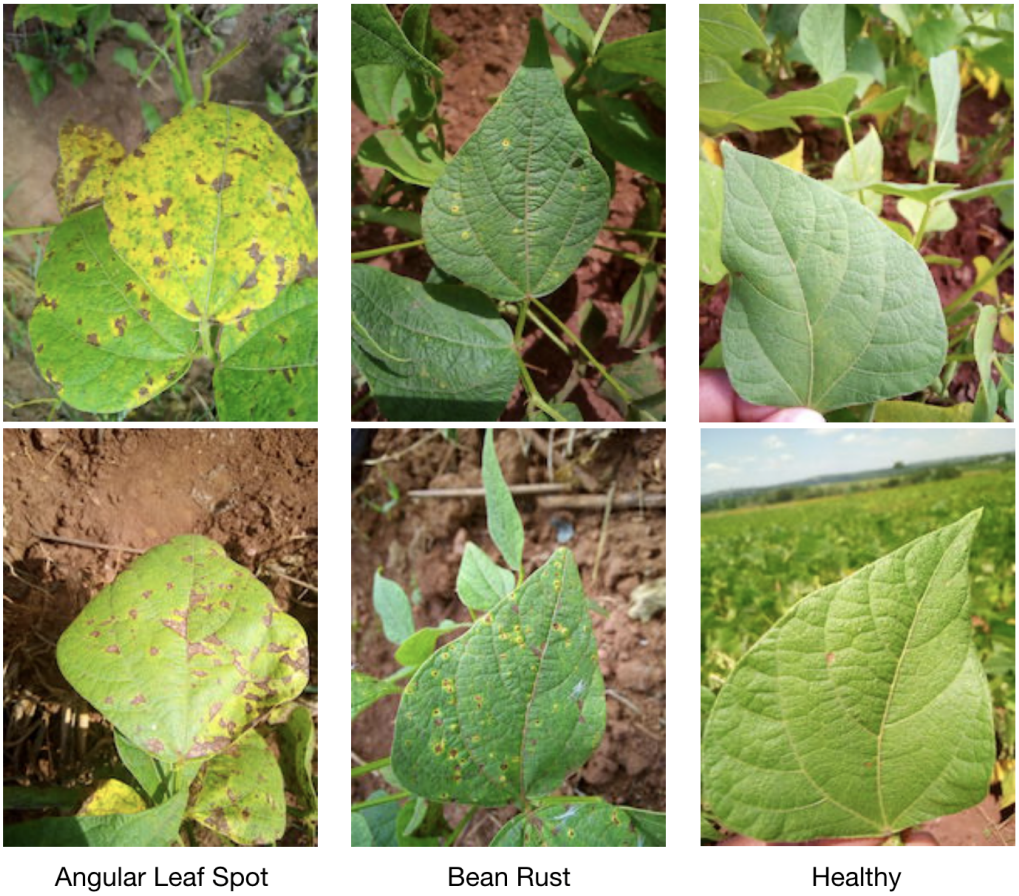


Figure 1: Dataset examples

Since the dataset consists of images of leaves, it is possible to affirm that operations such as rotation, reflection and zooming of the image should generate a new image with the same end label as the original one.

This kind of data augmentation techniques are useful to make the network more robust and also provide a larger dataset for the final trainning, which is useful for deep learning application. In this exercice, the same data augmentation pipeline proposed for the in class exercice will be used.

1.3 Exercice

The main objective of this homework is to create a Classifier for the ibeans dataset using a VGG16 CNN model as a base. Three cases should be evaluated for this study:

1. Training from scratch: Use the VGG16 model with no prior training, remove the latest layer (prediction layer) and add a classifier. Train the model from scratch and evaluate
2. Using pre-trained network as feature extractor: Use the same model of 1, but with the VGG16 model pre-trained on ImageNet. Freeze all layers of ResNet50, train and evaluate the model
3. Fine-tuning the latest layers: take a pre-trained VGG16 model, unfreeze the last blocks acording to the following:
   a Unfreeze the last convolutional blocks (from `block5_conv1`), train and evaluate the model
   b Unfreeze the last convolutional blocks (from `block4_conv1`), train and evaluate the model
   c Unfreeze all convolutional blocks, train and evaluate the model

Figure 2 shows an schematic of the three case studies listed above. It is worth noticing that, since the hyperparameters for the classifier should be the same for all three cases, the choice of hyperparameters will also be a part of this study.

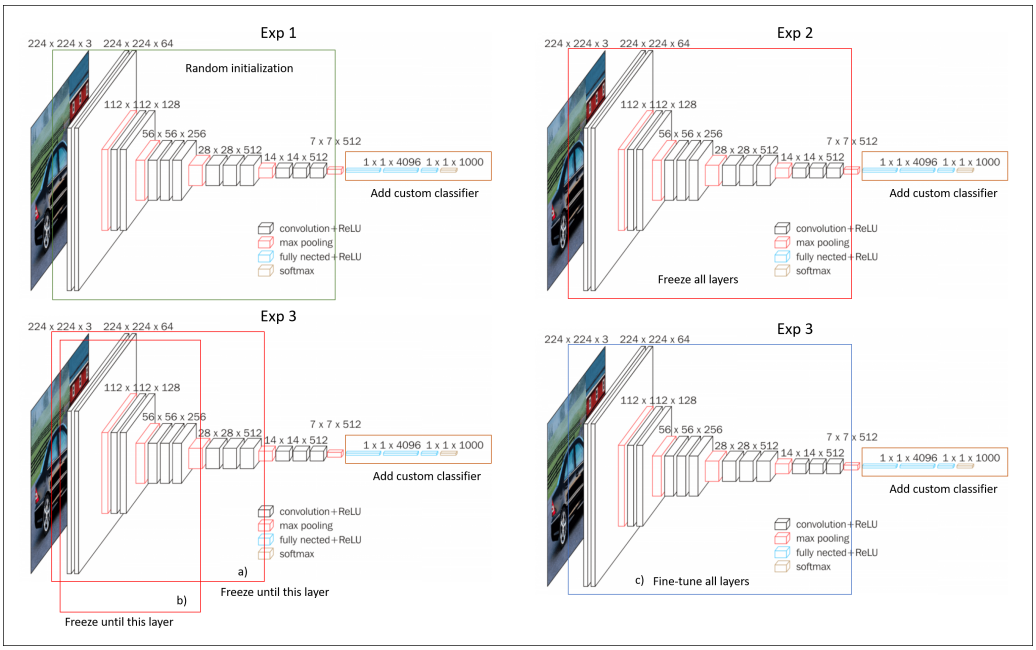


Figure 2: Exercices summary

Some differences should also be noticed when comparing this homework with the practical exercice performed in class about transfer learning. First, the homework dataset is a multi-class case, so the output layer has to be converted to a softmax-activated, 3-neuron layer. The loss and accuracy metrics used also hat to be replaced to the multi-class compatible `CategoricalCrossentropy` and `CategoricalAccuracy`, respectively.

The multi-class nature of the problem also impacted the loading of the dataset since the problem demanded a one-hot encoding of the label in order to be compatible with the new output layer proposed. An example of the updated data-reading routine can be seen in the code below, where the train dataset is loaded:

```
#Loading train
_URL = 'https://storage.googleapis.com/ibeans/train.zip'
path_to_zip = tf.keras.utils.get_file(os.path.basename(_URL), origin=_URL, extract=True)
print(path_to_zip)
train_dir = os.path.join(os.path.dirname(path_to_zip),'train')
BATCH_SIZE = 32
IMG_SIZE = (224, 224)
```

```
train_dataset = tf.keras.utils.image_dataset_from_directory(train_dir,
                                                            #Here we perform the encoding
                                                            label_mode='categorical',
                                                            shuffle=True,
                                                            seed=1337,
                                                            batch_size=BATCH_SIZE,
                                                            image_size=IMG_SIZE)
```

2 Results and discussions

2.1 Colab Notebook

All the code, examples and tests made are documented on the following Colab Notebooks. The second notebook is a copy of the first created only to exercice the plot of confusion matrixes in Colab.

- Link to the Colab Notebook
- Link to the Colab Notebook (only Confusion Matrix plots)

2.2 Hyperparameter selection

The overall implemented classifier will be based on the one used in the classrom example and will be composed of:

- A new `Input` layer with the expected image shape
- The VGG16 model imported or initialized with random values
- A `GlobalAveragePooling2D` layer to take the average of the VGG16 output channels
- An optional `Dense` layer with `"relu"` activation
- An output `Dense` layer with 3 neurons and `"softmax"` activation

The trainning will be performed in 50 epochs using the `Adam` optimizer and `CategoricalCrossentropy` loss. The base learning rate will be assumed as 0.0001, as in the classrom example.

The main hyperparameters tuned in this study were the number of nerons on the hidden layer of the classifier. For the hyperparameter selection, the case described in exercice 1 was used as a base. An architecture with no hidden layer, an architecture with 32 neurons on the hidden layer and one with 64 neurons were tested, and the results are reported in table 1.

	train acc	val acc	test acc
No hidden layer	92.07%	95.49%	89.84%
32 neurons hidden layer	94.00%	94.74%	92.19%
64 neurons hidden layer	91.88%	93.23%	89.06%

Table 1: Hyperparameter test results

Based on this results, it is possible to conclude that the hidden layer with 32 neurons was the best choice, having good overall accuracies and a lower difference in performance between train, test and validation datasets. In the Colab Notebook some tests with different learning rates were performed, but the overall conclusion was that the initial proposed learning rate of 0.0001 was the most adequate.

2.3 Exercice 1

The global results of loss and accuracy for the train and validation datasets for the exercice 1 are shown in Figure 3 for the best architecture selected in the hyperparameter selection study performed. The evolution of the metrics are shown along the epochs of the trainning.

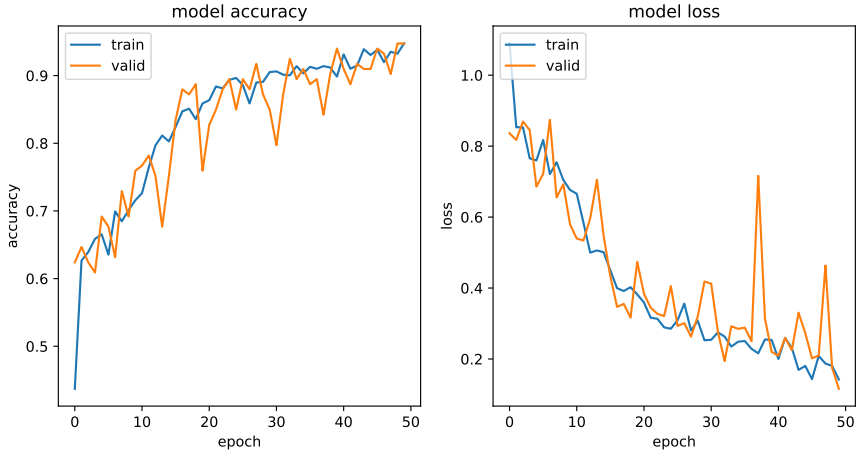


Figure 3: Results for exercice 1

2.4 Exercice 2

The global results of loss and accuracy for the train and validation datasets for the exercice 2 are shown in Figure 4 for the best architecture selected in the hyperparameter selection study performed. The evolution of the metrics are shown along the epochs of the trainning.

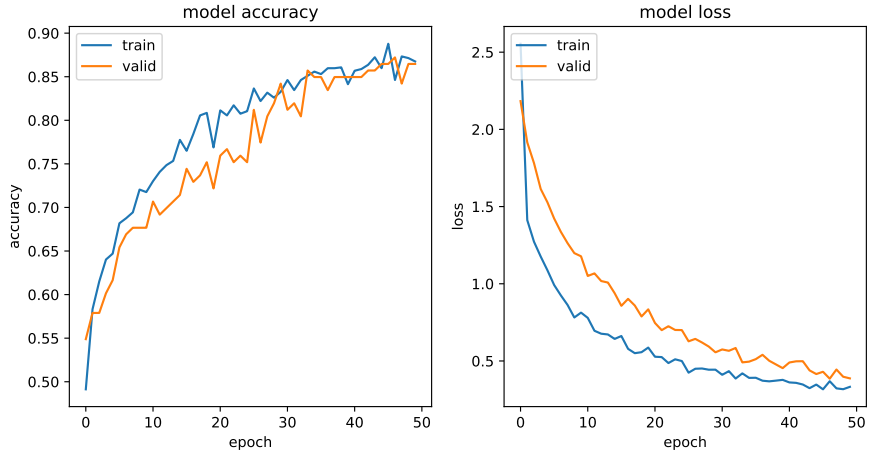


Figure 4: Results for exercice 2

2.5 Exercice 3

2.5.1 Item 3a

The global results of loss and accuracy for the train and validation datasets for the exercice 3a are shown in Figure 5 for the best architecture selected in the hyperparameter selection study performed. The evolution of the metrics are shown along the epochs of the trainning.

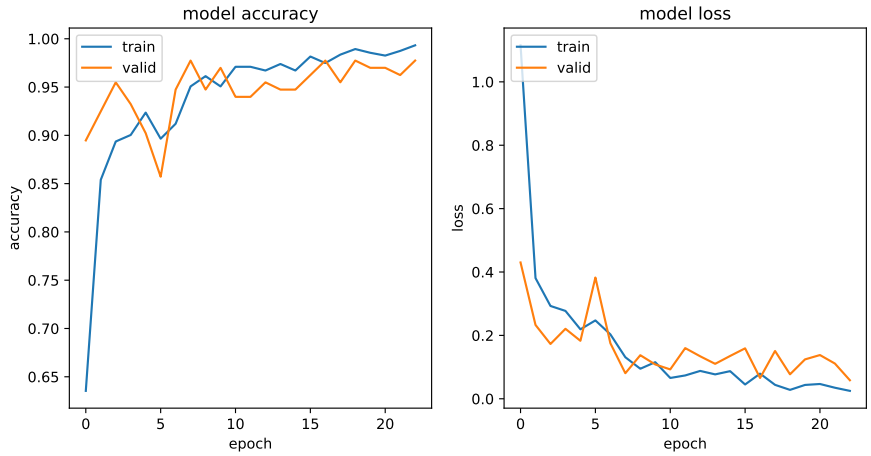


Figure 5: Results for exercice 3a

2.5.2 Item 3b

The global results of loss and accuracy for the train and validation datasets for the exercice 3b are shown in Figure 6 for the best architecture selected in the hyperparameter selection study performed. The evolution of the metrics are shown along the epochs of the trainning.

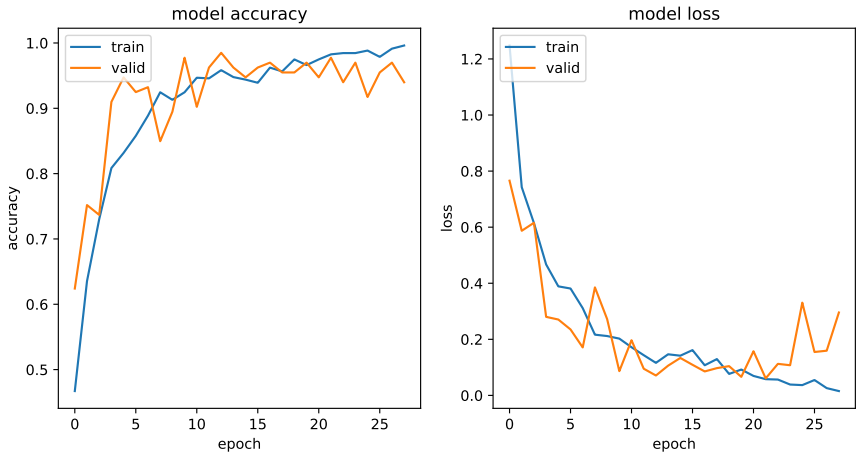


Figure 6: Results for exercice 3b

2.5.3 Item 3c

The global results of loss and accuracy for the train and validation datasets for the exercice 3c are shown in Figure 7 for the best architecture selected in the hyperparameter selection study performed. The evolution of the metrics are shown along the epochs of the trainning.

2.6 Results Summary and Conclusions

The results summary can be seen in Table 2. As a general rule, it is possible to see that, while cases that have more patameters to tune such as case 1 and case 2 tend to have a great difference between the train and test

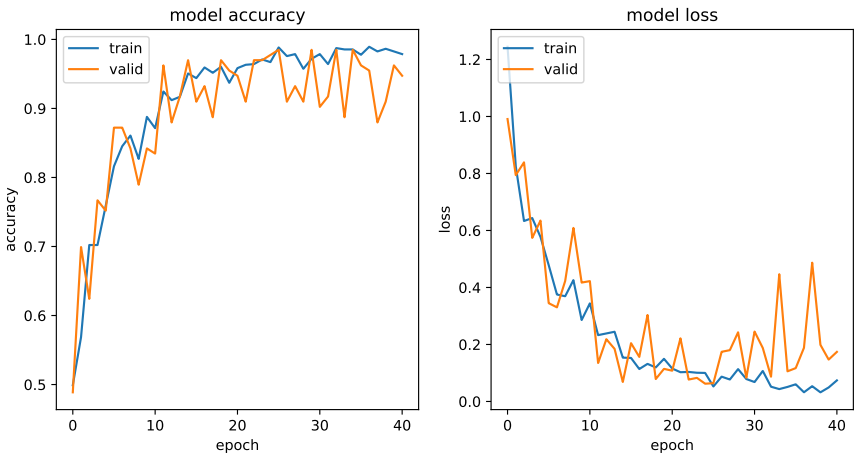


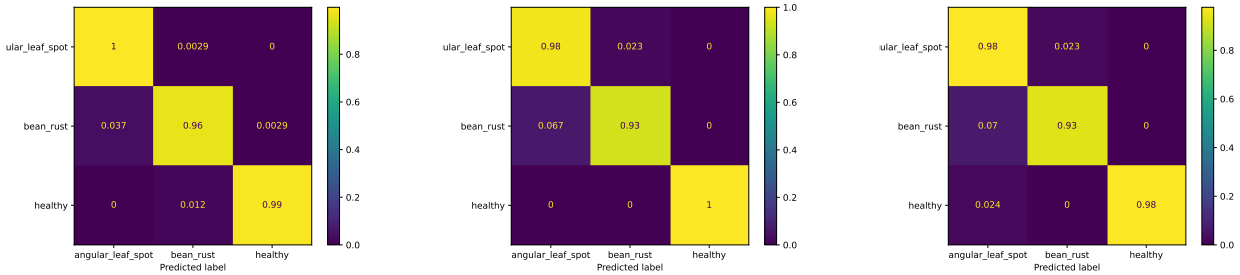
Figure 7: Results for exercice 3c

performance (a sign of overfitting), cases with less hyperparameters to tune such as 2 and 3a do not suffer form the same issue, having closer performences tin the train and test results. Also it is worth noticing that the use of the ImageNet weights as the initial weights for our problem allowed lower trainning times, specially when trainning the whole network. It is also worth mentioning that, while the use of the ImageNet weights provide a good first guess for our study, some fine-tuning of the convolutional layers is needed in order to obtain a performance that can be compared with trainning the whole nwtwork, with little to no loss in efficiency and even a performance increase thanks to the early stop condition adopted.

Exp	train acc	train loss	val acc	val loss	test acc	test loss	training time
1	94.00%	0.159551	94.74%	0.174357	92.19%	0.229864	14 min
2	87.43%	0.329751	92.48%	0.233386	85.94%	0.390936	06 min
3a	98.26%	0.055598	98.50%	0.056195	96.09%	0.112185	03 min
3b	95.84%	0.121377	96.24%	0.082859	91.41%	0.176876	04 min
3c	95.84%	0.116910	98.50%	0.076800	94.53%	0.131344	11 min

Table 2: Results summary

Considering all the results and comments, the case 3a can be considered the best strategy for the current dataset since it presented good accuracy, low discrepancy in train and test performance and an overall low training time. The results per class can also be seen in Figure 8, where we can notice that all classes are well-represented by the model, with most of the misclassifications ocurring between unhealthy leaves, specially `bean_rust`.



(a) Train set **(b)** Validation set **(c)** Test set

Figure 8: Confusion matrices plot for the 3a classifier (normalized by total real data per class)