Mushroom classification with MobileNetV2 and Xception

Lorenzo Vainigli
lorenzo.vainigli@studio.unibo.it
matr. 0000842756

Course of Machine Learning
Laurea Magistrale in Informatica
University of Bologna
A.Y. 2020-2021

Abstract. The task of classifing mushrooms is difficult beacuse there are a lot of species to know and one might differ from another for very little details. This fact make it difficult to a human to recognize a species while looking at a photo. In this paper is presented a method to use two convolutional neural networks, MobileNetV2 and Xception, to solve this classification problem. Results we got shows that with machine learning we can handle this task but it requires a lot of work.

1 Introduction

The purpose of this project is to study and build a classifier that is able to recognize images of mushorooms and categorieze them. To

do this we used two pre-trained convolutional neural networks that represent the state of the art: MobileNetV2 [1] and Xception [2]. Both models were trained on ImageNet database [3]: MobileNetV2 achieves a 71% top-1 accuracy and Xception achieves 79% on the same metric. As regarding top-5 accuracy, Xception will perform better with a score 95%, while MobileNet has a score of 90%. The trade-off is that ImageNet is six times smaller than Xception.

2 Methods

In this section we describe the process of building these models. We begin with an analysis of the dataset to take a look at the data that we will use for train, validation and test. Then we describe the steps followed to obtain the model for our purpose.
Briefly, we used the pre-trained convolutional layers of the two models to make a first training and test phase, we analized results and then we did another training and test phase, this time enabling the training of some convolutional layers. Details are explained in the next sections. We have done it both for ImageNetV2 and for Xception with 3, 10 and 20 classes.

2.1 Software

This experiment was done using Python in the Google Colab platform [4] and Google GPUs. Tensorflow [5] is the core tool used and the neural networks models are loaded from the Keras library.

2.2 Dataset

The image dataset is composed by an hierarchical structure of folders named with the category name of the images that they contains. Precisely each folder name has the format `ID_super-category_category`, so we use this information to associate these two properties to the images. This dataset is very large: it contains about 90,000 high resolution images belonging to about 1,500 classes. For our work we used 6,739 images belonging to 20 classes. Unfortunately, as shown in fig. 1, the dataset is very unbalanced; to create balanced sets without using class weights, we used the following amount of images depending on the number of classes: 414 if we use the first 3 classes, 340 if

we use the first 10 classes and 255 if we use all the 20 classes.
There are some annotation files but we ignore it, since our purpose is to associate each image of mushroom to its category and no more.

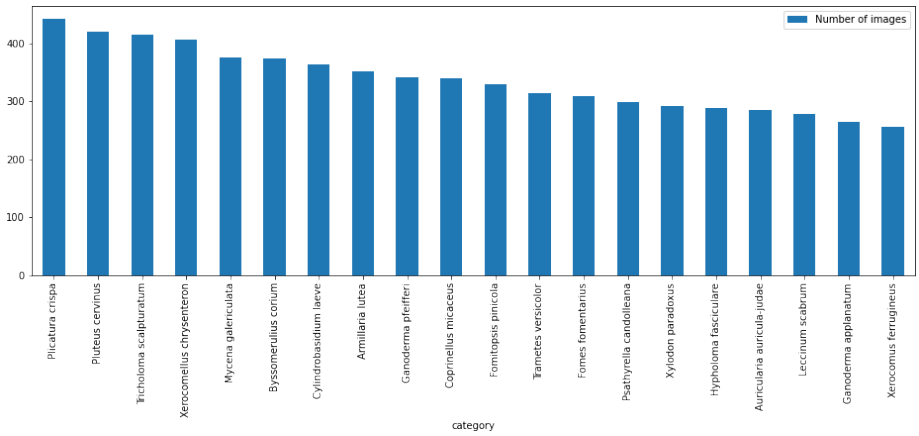

Fig. 1: Number of samples for each class sorted in descending order. The 20^{th} class has 38% less images than the first class.

2.3 Image processing

In this project we have followed the official TensorFlow tutorials for transfer learning [6]. These guidelines suggests to add at the beginning of the model a random flip layer and a random rotation layer for perform augumentation on input images. In addition, an auxiliary input preprocessing layer was added because each Keras model expects a specific kind of input preprocessing. In our case these functions are:

- `tf.keras.applications.mobilenet_v2.preprocess_input` for MobileNetV2;
- `tf.keras.applications.xception.preprocess_input` for Xception.

As regarding image size, despite the models accept custom image sizes, we leave it as default: 224x224 for MobileNetV2 and 299x299 for Xception.

2.4 Transfer learning

We will take advantage from the possibility to access the pre-trained models in a very simple way with Keras. Once we have loaded the model, we add a top dense layer to adapt it to our number of classes preceded by a pooling layer and a dropout layer. The complete architecture of the model is shown in fig. 2.

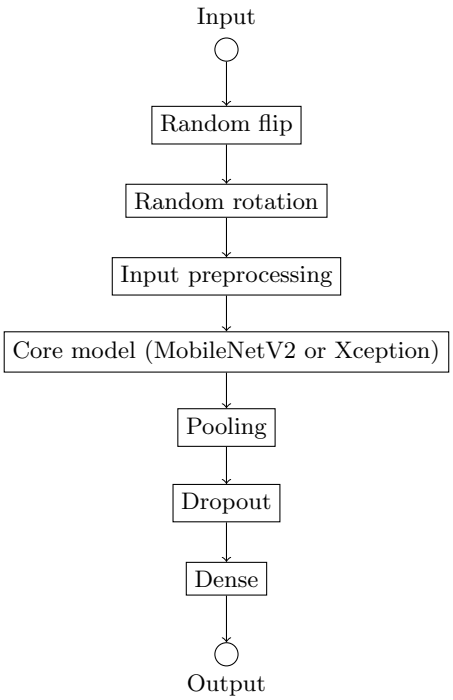

Fig. 2: Architecture of the model.

2.5 Fine tuning

As suggested in [6], after the first training and evaluation of the model, a second training phase (fine tuning) has been performed to get better results. In fine tuning, some convolutional layers were unlocked for training and learning rate was decreased.

2.6 Hyperparameters

The experiments that were made share the same settings of hyperparameters: the split between training-validation set and test set was set to 80%-20% as the split between training set and validation set; batch size was, as default, fixed to 32; learning rate is 10^{-4} for base training and 10^{-5} for fine tuning. Epochs were set to 100 for base training and 20 for fine tuning, but an early stopping with patience 10 has been used to prevent overfitting.

3 Results

Table 1 show that MobileNet performs better than Xception if we train the model with 3 classes, but if we choose 10 or 20 classes (and this means that the complexity is greater) Xception achieves better results. In figure 3 we can clearly notice that the fine tuning helps a lot in the improvement of the model, especially if we have a high number of classes.
Considering the top-5 accuracy, instead, we can conclude that both models perform pretty well and thay achieve the same results.

Classes	**Samples per class**	**MobileNetV2**		**Xception**	
		Top-1	**Top-5**	**Top-1**	**Top-5**
3	414	92%	-	90%	-
10	340	77%	98%	81%	98%
20	255	65%	95%	70%	95%

Table 1: Result values for top-1 accuracy and top-5 accuracy.

4 Discussion

This work is only a basic application of two neural networks in image classification tasks and considering results achieved by these models with the 1,000 classes of ImageNet, the results on this dataset were not so good. We notice that with increasing the number of classes, the accuracy tends to decrease and 20 classes are a very few number.

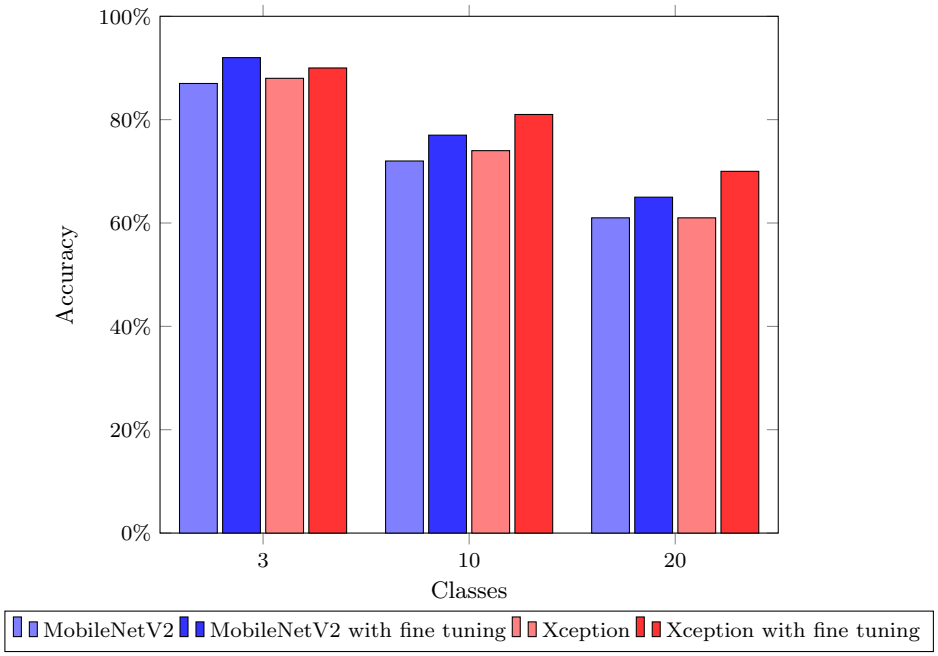

Fig. 3: Comparison of performances between MobileNetV2 and Xception.

The dataset is not optimal and we can say that if we have a larger amount for each class, as the ImageNet dataset, results will be better. Nevertheless, the dataset used is very interesting beacuse sometimes it will be very difficult for a human to recognize the species of a mushroom among all those that exist. Here the challenge is not to improve the models (even if there is room for improvement, it's more or less proven that they are good), but to improve the dataset with more samples.

Further directions

The results shown were obtained training the two models with 3, 10 and 20 classes, but hyperparameters were fixed. It might be interesting to try different setting of hyperparameters to see how results change.

5 Conclusions

In this paper was presented a method to use pre-trained neural networks to classify mushrooms. The work begun with an analysis of the dataset and then the model was implemented following the official guidelines. In addition to the classical training phase, a fine tuning phase was performed to achieve better performances. Results shows that models can correctly predict the true mushroom species for a given image with noticeable accuracy.

References

[1] Mark Sandler et al. *MobileNetV2: Inverted Residuals and Linear Bottlenecks*. 2019. arXiv: 1801.04381 [cs.CV].

[2] François Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017. arXiv: 1610.02357 [cs.CV].

[3] Jia Deng et al. "ImageNet: a Large-Scale Hierarchical Image Database". In: June 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848.

[4] Tiago Pessoa et al. "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications". In: *IEEE Access* PP (Oct. 2018), pp. 1–1. DOI: 10.1109/ACCESS.2018.2874767.

[5] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[6] *Transfer learning and fine-tuning — TensorFlow Core*. URL: https://www.tensorflow.org/tutorials/images/transfer_learning.