Università degli studi di Milano-Bicocca

Advanced Machine Learning

Final Project

A small flower classifier: from a little dataset to 102 categories

Authors:
Eleonora Cicalla - 851649- e.cicalla@campus.unimib.it
Alessia Rubini - 851890- a.rubini5@campus.unimib.it
Matteo Severgnini - 851920- m.severgnini9@campus.unimib.it
February 2, 2024

Abstract

This project involves a task of image classification of flowers using the *102 Category Flower Dataset*(link). This dataset is characterized by an unusual division of train, validation, and test sets, with small train and validation sets (1020 images) and a very large test set (6149 images). The goal of this work is to create a high-performance model while maintaining the unusual split of the dataset and to reduce its dimension while keeping the same performances, so as to be usable even on devices with limited resources. Three models were implemented exploiting the principles of *transfer learning*, namely performing *feature extraction and fine tuning* using three different models pretrained on *ImageNet* (*VGG16, InceptionV3, ResNet50*). After identifying the most performant model, a *model explanation* was carried out implementing *Grad-CAM* and *LIME*, to understand and demonstrate the functioning of the model, and *quantization* was performed in *float-32, float-16, and int-8*. The thesis thus demonstrates that, despite the dataset having unusual and suboptimal train, validation, and test set splits, it is still possible to obtain a high-performance model, with an accuracy of about 86%, and compressed, with a significant reduction in terms of memory from 80MB to 11MB without compromising on performance.

1 Introduction

This project's main goal is to build a discrete classification model for the *102 Category Flower Dataset* dataset and reduce its dimension.
The biggest challenges in classifying the images in this dataset were having significantly fewer samples in the train set than the test set, and having particularly similar types of flowers. Having little data available for model training often leads to overfitting, so several techniques were adopted so as to prevent this scenario. The strategy adopted was to perform transfer learning by starting with pre-trained neural models and then specializing them on the flower classification task. In particular, three different architectures were used and compared in order to evaluate which one performed better to our data in terms of accuracy and loss. Then, some analysis was performed to better understand the way of employing the models used. In detail, grad-CAM and LIME analyses were carried out.
Once the best neural architecture was identified for the data at hand, quan-

tization was performed so that the final model could be used even on devices with limited computational resources.

2 Datasets

The *102 Category Flower Dataset* created by Maria-Elena Nilsback and Andrew Zisserman is a dataset containing high-resolution images of flowers commonly occurring in the United Kingdom. The dataset is divided into 102 classes. Along with the dataset of all the images, a matrix is provided that divides the data into a train set (1020 samples), a validation set (1020 samples), and a test set (6149 samples), and specifies the classes to which each image belongs. In this arrangement, for instance, the categories labeled Petunia have been merged as identical, etc... Hence, the original dataset was divided into the three parts just mentioned, and then each of them was further divided into 102 sub-folders so that each sub-folder represented a flower category and contained only the images related to that specific category. Furthermore, a dataset with the segmentations for the images was provided but not used in this project. The reason for not using it is that the pixels were not removed but replaced with blue background pixels, which could negatively impact the model's performance. Each class in the train set comprises only 10 images, making it extremely necessary to carry out data augmentation to train the model with a sufficient amount of data. Train, test, and validation sets have been preprocessed using the corresponding preprocessing function of each analyzed model.

3 The Methodological Approach

In this section the key points of the thesis are shown. In particular, one of the main objectives is to identify the best CNN architectures in the literature that can perform better on the available data. The training took place first by keeping all the weights relating to the pre-trained models frozen, and then also updating the weights of some layers of the pre-trained models, trying to identify a good compromise between accuracy and training time - as suggested by the official documentation. All proposed architectures were pre-trained on ImageNet and used as feature extractors for a new classifier that operated on 102 classes.

3.1 Data augmentation and Preprocessing

The data augmentation operation was performed on the train set to try to fill the data shortage and, consequently, reduce the risk of overfitting. In order to raise the number of samples it was used *ImageDataGenerator* from *Keras*. This allowed to virtually loop infinitely on the images during the training and most importantly to transform each image by applying none, one or more of the following transformations: rotation range of 20, brightness range between 0.7 and 1.3, zoom range between 0.8 and 1.2, and horizontal flip. Furthermore, each image is transformed with the original preprocessing function used for training the original architecture on *ImageNet*, paying attention to the fact that different models may require different operations (e.g. the size of the image it accepts as input).

3.2 Scheduler for learning rate

A *Cyclical Learning Rate (CLR)* is used in this project since it has been discovered that a varying learning rate during training is beneficial overall and thus proposes to let the global learning rate vary cyclically within a band of values (*min_lr, max_lr*) instead of setting it to a fixed value. In addition, this cyclical learning rate method (CLR) practically eliminates the need to tune the learning rate yet achieve near optimal classification accuracy. Furthermore, unlike adaptive learning rates, the CLR methods require essentially no additional computation. To determine the optimal range of values for CLR, an *LR Range Test* was conducted. A *Linear Learning Rate Scheduler* was implemented, which involves running the model for several epochs allowing the learning rate to increase linearly between low and high learning rate values. During this experiment, it is important to observe the trend of the loss function as the CLR range should be chosen where the decrease in loss is greatest. This range is not unique for all models, but varies for each one. Another essential parameter for the proper functioning of CLR is *step_size*, which specifies the number of iterations required to complete half a cycle, i.e., the increase from the minimum learning rate value to the maximum learning rate value or vice versa. A full cycle is therefore carried out by doubling the number of *step_size*. It is important, therefore, to balance the number of *step_size* with the number of iterations per epoch, calculable as the *size of the trainset* divided by *batch_size*, so that multiple cycles are carried out during the training of the model. The paper recommends using a *step_size* from

2 to 10 times the number of iterations per epoch, but it can vary following empirical tests. CLR allows the use of policies, i.e., techniques for varying the learning rate as the epochs change. The main policies are: **triangular**, which varies the learning rate in the range [*min_lr, max_lr*] as the epochs change, keeping *min_lr* and *max_lr* constant; **triangular2**, which varies the learning rate in the range [*min_lr, max_lr*] but setting *max_lr* to $\frac{\text{max\_lr}-\text{min\_lr}}{2}$ at the end of each cycle; **exp range**, which varies the learning rate in the range [*min_lr, max_lr*] reducing the value of *max_lr* following an exponential factor. These policies are useful for increasing model performance. The choice of which to set depends solely on the results of empirical tests. [1]

3.3 Transfer learning and fine tuning

For this project it was chosen to use the fine tuning technique: starting from pre-trained CNN architectures on a larger dataset, a cutting point is identified between the layers closest to the output and then adding any layers useful for specializing the pre-trained model on the task of interest. For each model the *batch_size=64* was set. In order to reduce the risk of overfitting, during training, the scheduler for the learning rate illustrated in the previous section was used, and an early stopping which monitors the validation_loss and stops training if this does not improve within 5 epochs. For all three architectures, training was first performed keeping all the weights of the pre-trained models frozen. After that, the training continued by also updating the weights of some of the layers for some epochs; in this second training phase the cyclic learning rate scheduler was not used, but a very low learning rate.

3.3.1 VGG16

VGG16 is a convolutional neural network model that is 16 layers deep. It takes in input tensor size as 224, 244 with 3 RGB channels. The *preprocess_input* function converts the input images from RGB to BGR, then will zero-center each color channel with respect to the *ImageNet* dataset, without scaling. The best configuration obtained involves the use of a model cutting function that cuts the model at the second last layer "*block5_conv2*", for performance reasons. After cutting the base model, all the layers are frozen and some classifier layers are added to the *VGG16* base model, since the *VGG16* base model is loaded without the top, after adding the classifier the model is compiled and trained. The layers added are: *GlobalAveragePooling2D*, *Dense* with 1024 weights, *BatchNormalization*, a *ReLu* activation layer, *Dropout* of magnitude 0.5 and a final *Dense* layer with 102 weights and *Softmax* as the activation function. Five layers are then unfrozen and the model is recompiled and retrained.

3.3.2 ResNet50

ResNet50 is a convolutional neural networks that is 50 layers deep and accepts images of input size of 224×224 px. As in the previous case, the *preprocess_input* function convert the input images from RGB to BGR, then will zero-center each color channel with respect to the *ImageNet* dataset, without scaling. The best configuration was obtained by making a cut on the convolutional layer *'conv4_block6_out'*, an intermediate layer that maintains a good balance, preserving general features useful for various tasks, but not too specific that might not adapt well to new data. After making the cut, the weights were frozen and subsequently layers were added to make the model suitable for the *102 Flowers Recognition* dataset. Specifically, the following were added: a *GlobalAveragePooling2D()* layer; two *Dense* layers, the first with 512 nodes and the second with 256 nodes, both followed by a *BatchNormalization()* layer, an *Activation* layer, which uses the *ReLU* activation function, and finally a *Dropout* layer equal to 0.5; and finally a last *Dense* layer, the output layer, composed of 102 nodes, one for each class of the problem, and *softmax* as activation function. L2 normalization was performed, that is, the addition of a penalty term to the loss function of the model based on the L2 norm of the network weights, equal to 0.005, in the first two *Dense* layers. The optimizer, after a series of experiments, was chosen to be SGD. The first phase of training involves training the model after the cut with all the weights frozen so as to preserve these useful features when adapting the model to a new task or dataset. This initial training was carried out for 40 epochs, using the learning range *[0.08; 0.4]*, a *step_size* of 4, and the *triangular* policy. During the second phase, instead, the last 10 layers were unfrozen, after the output level, only if of the type *Conv2D* or *Activation*. The first model with the unfrozen weights was trained for 80 epochs, still using the SGD activation function but with a small learning rate, equal to *1e-5*. The reason why *CLR* was not used is that having already defined weights, it is essential to use small and constant learning rates to avoid overwriting the useful features learned previously. In a final phase, the previous phase was repeated, but unfreezing the weights of another 10 layers, reaching a total of 20 layers with trainable weights. This incremental weight unfreezing approach was chosen as it provides a good balance between maintaining the useful features learned and adapting the model to the new task efficiently and effectively.

3.3.3 InceptionV3

In InceptionV3, there are several inception blocks, each of which contains multiple layers. In total, InceptionV3 has a total of 48 layers, including both convolutional layers, pooling layers, batch normalization, and fully connected (dense) layers. It takes as input images of size 299x299 px. The preprocessing function applied on the images scaled their pixels between -1 and 1, sample-wise. From the output of the LRF algorithm the learning rate was varied in the interval $[0.005, 0.06]$ for the optimizer SGD with 'mode' set to *triangular* and '*momentum*' set to 0.9. The selected cutting point was *conv2d_84*, which is one of the convolutional layers preceding the last two 'inception blocks'. This resulted in a model pre-trained on *ImageNet* but not yet specialized for the specific task of interest. Subsequently, the following layers were sequentially added: *GlobalAveragePooling2D*, *Dense* with 512 units, *BatchNormalization*, and an activation function set to *'relu'*, followed by a *Dropout* layer with a dropout rate of 0.5 to attempt to reduce overfitting. Finally, a *Dense* layer with 102 units and *'softmax'* activation was added for the final classification of the 102 flower categories. At this point, the first training phase was initiated while keeping all the layers of the pre-trained model "cut" with frozen weights, leading to promising results. Afterwards, for proper fine-tuning, some of the layers of the cut pre-trained model were made trainable. This allowed the model to further specialize for the final classification task, resulting in improved performance. To determine how many layers to unfreeze, several experiments were conducted, ultimately deciding to unfreeze the last 10 layers of the model closest to the output, taking care not to make the BatchNormalization layers trainable.

3.3.4 Training evaluation: Grad-CAM

Grad-CAM [2] utilizes the gradients of the last convolutional layer to display an heatmap over the original image that highlighted the main component identified by the model. Convolutional layers naturally retain spatial information which is lost in fully-connected layers, so we can expect the last convolutional layers to have the best compromise between high-level semantics and detailed spatial information. The neurons in these layers look for semantic class-specific information in the image and *Grad-CAM* uses the gradient information flowing into the last convolutional layer of the CNN to assign importance values to each neuron for a particular decision of interest. [3]

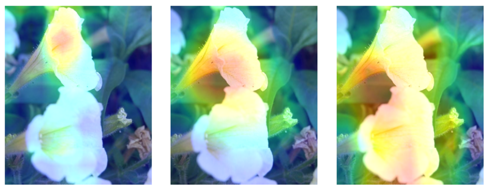

Figure 1: *Grag-CAM* results obtained on the image_01315(class 51) from VGG16, InceptionV3 and ResNet50

For example, the image 1 shows that the models are capable of distinguishing flowers from the background and understand that highlighting the background is not useful for classification.

3.3.5 Model explanation: LIME

LIME (Local Interpretable Model-agnostic Explanations) is a technique used to explain the behavior of machine learning models. Its main goal is to provide local and interpretable explanations of the predictions made by the machine learning model. It is applied to CNNs to explain predictions on individual images. For a given input image, *LIME* identifies the significant regions of the image that contributed the most to the model's prediction. These regions are highlighted and clarified using segmentation and visualization techniques. The interpretation provided by *LIME* is local and focuses on interpreting predictions for a specific input instance rather than for the entire model. This allows users to understand the model's behavior in detail for specific cases, rather than just having a general overview of the model. Furthermore, being model independent, *LIME* can be applied to any machine learning model, including CNNs, regardless of their architecture or complexity. The implementation has facilitated the observation, based on a given input, of the extent to which a particular region influenced the model's classification of that specific image as belonging to a certain class, in terms of probabilities. The project team decided to apply this model explanation technique only to clarify the InceptionV3 classification model because it was identified as the best performing one based on the available data.

3.4 Quantization

Another of the main goals of the paper is to make a more efficient compression of the model while maintaining its performance. Quantization is the process of representing values with a reduced number of bits. In neural networks, this can be applied to weights, activations and gradient values. There are two main forms of quantization: post-training quantization and quantization aware training. In this work it was performed the first one. First, the already trained TensorFlow float model was optimizated by conversion to *TensorFlow Lite* format using the option TensorFlow Lite converter. This resulted in a TensorFlow Lite model, which still uses 32-bit float values for all parameter data. By enabling the default optimization flag, all fixed parameters (such as weights) were quantized. Instead, to quantize variable data (such as model inputs/outputs and intermediates between levels), it was necessary to provide a RepresentativeDataset. This is a generating function that provides a set of input data large enough to represent typical values. This allows the converter to estimate a dynamic range for all variable data. At this point the following quantization options were:

- **16-bit floating-point quantization (FP16)**: The model uses 16-bit floating-point quantization to represent the weights and operands of the model.
- **8-bit integer quantization (INT8)**: The model uses 8-bit integer quantization to reduce model size and improve inference efficiency.

As was done for the explanation of the model, it was decided to apply quantization only to the InceptionV3 model. The results allow us to observe how the size of the model changes based on the type of quantization applied and whether this impacts (in theory not) its performance.

4 Results and Evaluation

4.1 Fine tuning results

The section reports the results obtained for the classification models.

Transfer learning									
Model	**Batch size**	**Step size**	**Policy**	**Optimizer**	**Test loss before unfreeze**	**Test accuracy before unfreeze**	**Unfreeze layers**	**Test loss post unfreeze**	**Test accuracy post unfreeze**
VGG16	64	16	triangular2	SGD	0.7686	0.8037	5	0.6731	0.8291
ResNet50	64	4	triangular	SGD	2.0619	0.7441	20	1.2464	0.8446
InceptionV3	64	2	triangular	SGD	0.6209	0.8333	10	0.5154	0.8614

Table 1: Results from fine tuning before and after the unfreeze of some pre-trained model layers.

4.1.1 Misclassification and image-similarity-search

On each model's *ipynb* notebook, it's possible to visualize a report detailing the performance in a detailed manner, class by class. In particular, from the InceptionV3 report, some classes were identified where the model did not achieve excellent performance, for examples classes 1, 3 and 11. Therefore, it was decided to report some examples of misclassifications related to these categories. To understand these misclassifications, an *image-similarity-*

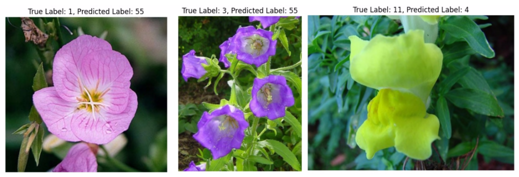
True Label: 1, Predicted Label: 55
True Label: 3, Predicted Label: 55
True Label: 11, Predicted Label: 4

Figure 2: Three examples of misclassification made by InceptionV3

search task was performed. In this task, by inputting the misclassified image, feature extraction and Euclidean distance calculation are conducted to find the most similar image to the given input from the class with which it was mistakenly labeled. Below is an example: feature extraction was done with InceptionV3 model.

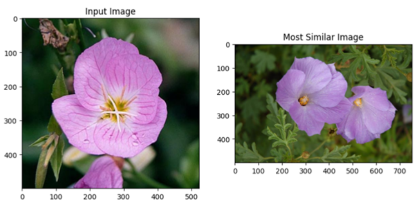

Figure 3: The most similar image to the misclassified image with true label equal to 1, but belonging to class 55.

4.1.2 Model explanation results

On the three misclassification examples, the model explanation with LIME was applied to understand which areas of the images the classifier had used to determine (incorrectly) the flower category.

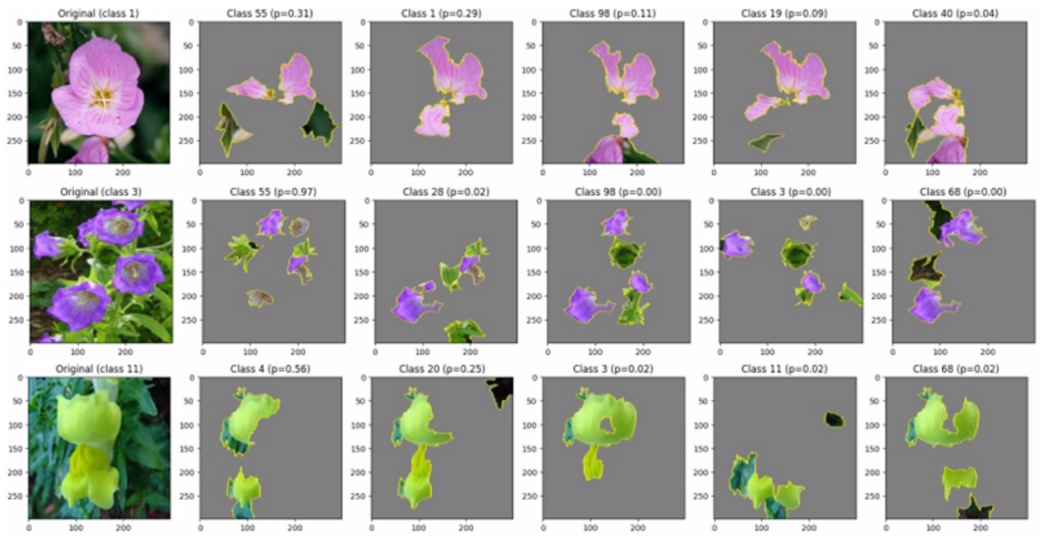

Figure 4: LIME applied on three InceptionV3 model misclassification examples

4.2 Quantization model results

The results related to the second main objective of the project are presented: selecting and "quantizing" the most performing model in order to significantly reduce its size.

Quantization				
	Keras	**TFLite**	**Float-16**	**Int-8**
Size	80.6M	42.2M	21.1M	10.9M
Accuracy	0.8614	0.8614	0.8612	0.8611

Table 2: Quantization results on InceptionV3

5 Discussion

Given the small size of the training set and the disparate class distribution between the training and test sets, the approach of transfer learning was chosen to achieve good performance. As extensively discussed, various techniques were experimented with in this phase to mitigate the risk of overfitting. The strategy involved selecting three pre-trained models on ImageNet: VGG16, ResNet50, and InceptionV3. A cutting point at a layer that preserved features useful for the flower classification task was identified, and the models were customized by adding necessary layers, which varied from model to model.
Table 1 illustrates the performance achieved by training the three classification models through fine-tuning. This allows for comparison and demonstrates how the performance varies with the models used. The accuracy values obtained before unfreezing the weights indicate that the ResNet50-based model appears to be the least performing, with relatively high loss and unsatisfactory accuracy. Conversely, the InceptionV3-based model appears to be the most performing, with accuracy reaching up to approximately 83% on the test set. After making some of the layers of the pre-trained models trainable, an increase in performance is observed across all models. Particularly, this training phase led to significant performance improvements in the ResNet50-based model. However, the classifier built on InceptionV3 continues to exhibit the highest performance even after this phase.
For a deeper understanding of this model, three misclassification examples were selected for model explanation with LIME. Additionally, an image similarity search was conducted. This analysis, presented in the results section, helped understand which regions of those three images misled the model, also providing estimates of the probability of belonging to other classes. It is also noted that for the three examined samples, the highest probability of belonging is indeed to the class with which the model classified them (incorrectly).

Having established that the most performing model is based on InceptionV3, it was utilized for the second task of the project, namely model quantization. Table 2 displays the results of quantization applied to the classifier built on InceptionV3. As expected, this phase significantly reduced the size of the model compared to the original model, with no decrease in performance. Quantization enables the use of models on devices with very limited computational resources.

6 Conclusions

This project's main goal is to build a classification model for the *102 Category Flower Dataset* and to reduce its dimension while maintaining the same performance. Despite the dataset's unusual splitting, both objectives were achieved: to create a model with excellent performance, and to realize a reduced-size model with great performance, enabling its use on devices with limited computational resources. The optimal model indeed achieves an accuracy of 86.14% with a size of 80.6MB. The same quantized model reaches an accuracy of 86.11% with a size of 10.9MB.
The work proposed in this thesis can be extended by conducting new experiments such as using additional pre-trained models for transfer learning, implementing an *ensembled model*, or by modifying the original task, creating a *GAN* model for generating new images of flowers.

References

[1] L. N. Smith, "No more pesky learning rate guessing games," *CoRR*, vol. abs/1506.01186, 2015. [Online]. Available: http://arxiv.org/abs/1506.01186

[2] Keras Documentation. (2024) Keras grad-cam example. [Online]. Available: https://keras.io/examples/vision/grad_cam/

[3] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-cam: Visual explanations from deep networks via gradient-based localization," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 618–626.