Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Flowers recognition

Author:
David Bertoldi - 735213 - d.bertoldi@campus.unimib.it

January 20, 2023

Abstract

The ABSTRACT is not a part of the body of the report itself. Rather, the abstract is a brief summary of the report contents that is often separately circulated so potential readers can decide whether to read the report. The abstract should very concisely summarize the whole report: why it was written, what was discovered or developed, and what is claimed to be the significance of the effort. The abstract does not include figures or tables, and only the most significant numerical values or results should be given.

1 Introduction

The aim of this work is to build a machine learning model able to learn from a small knowledge base and to classify similar images but belonging to different classes. The main issue to overcome was the high chance that the model could overfit and fail to classify unseen samples. The strategy adopted was to find the model exploiting transfer learning the best and tryed to freeze the model such that it maintained similar performances.

This document describes the research on trained models, hyperparameters and generalization techniques that allowed the model to operate on a large variety of images.

2 Datasets

The dataset used is the *102 Category Flower Dataset* [1] created by the researchers of the *Visual Geometry Group* of *Oxford*. The dataset is composed of 8 189 RGB images of variable size, each image contains one or more flowers on a neutral background and is labeled with a single category extracted from a set of 102 possible categories. The original dataset also contains the flowers segmented from the background (Figure 1); these images can be used for example as further input for the neural network. In this work they were not used in order to force the model to be more elastic with respect to the background of the images.

The subdivision of the dataset defined in the original publication has been maintained, in particular there are 1 020 images in the training set, 1 020 images in the validation set and 6 149 images in the test set.

Figure 1: Training images and their segmentation

Each category is represented by 10 images in the training set and validation set, while the proportion of images for each category varies in the test set. The difficulty of operating on a dataset of this type is evident: the number of images for training is limited while the test set is larger.

Another peculiarity of the dataset is the presence of similar images belonging to different categories.

3 The Methodological Approach

In order to classify images correctly two types of experiment were taken in accont: the first one aimed to find the best CNN[1] architectures in literature that could fit the available hardware (see Section 3.1) and the second tried to find a good trade-off between number of trainable layers and accuracy by freezing the layers' weights during training.

All the proposed architecture were pre-trained on *ImageNet* and used as feature extractors for a new classifier that operates over 102 classes.

3.1 Technology and implementation

The hardware used for this work was composed by a GPU NVIDIA 2070 SUPER with 8GB of VRAM, a CPU Intel i7-9700K 3.6GHz and 32GB of RAM.

The libraries used were *Keras*, based on *Tensorflow*, for learning and preprocessing, *sklearn* for metrics and *numpy* for generic calculation. Github

[1]Convolutional Neural Network

3.2 Preprocessing and Data Augmentation

Before proceeding with the description of the experiments, there was a preliminar implicit objective: how to overcome the scarcity of training data.

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 of these images by applying none, one or more of the following transformations:

- Horizontal flipping
- Rotation of angle $\alpha \in [-20°, 20°]$
- Shift of brightness $\gamma \in [0.7, 1.3]$
- Zoom $\zeta \in [0.8, 1.2]$

Additionally every image is transformed with the original preprocessing function used for the training of the original architecture on *ImageNet*. In Sections 3.4.1, 3.4.2 and 3.4.3 there are more details about it.

3.3 Hyper-parameters

To train the perceptron on the extracted features it was decided to use the SGD with momentum $\beta = 0.9$ or Adam with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, with dimension of batch equal to 64. Since the task is a classification over multiple categories the chosen objective function to minimize was *categorical cross-entropy*.

The learning rate η is calculated according to the algorithm described in *"Cyclical Learning Rates for Training Neural Networks"*. This algorithm made η fluctuate forward and back in a fixed interval I at each iteration during the training phase; this strategy had a dual purpose: to reduce the bias introduced by choosing a non-optimal η during the design phase and to help the optimizer to escape from saddle points or from local minima that could block its correct covergence.

To calculate the aforementioned interval I of values the authors described the *Learning Rate Finder* algorithm (LRF):

1. Choose an interval J on which to make vary η. For this work $I = [10^{-10}, 10^{-1}]$ was used

2. Train the model for few epochs (*e.g.* 10) starting with the smallest $\eta' \in J$. At the end of each epoch exponentially increase η'

3. Stop the training when η' reached the upper bound of J

4. Plot the fluctuation of the loss function realtive to η'

As an example, the algorithm generated the plot in Figure 2 for optimizer Adam and the architecture described in section 3.4.1.

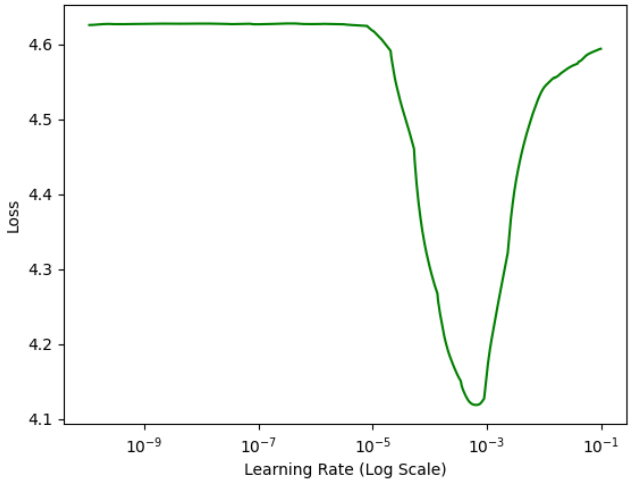

Figure 2: Output of the learning rate finder algorithm. $I = [10^{-5}, 10^{-3}]$ is the optimal solution

The graph shows how the network begins to learn starting from $\eta \simeq 10^{-5}$ and diverges once η exceeded $\sim 10^{-3}$; these two values were used as extremes of I for the cyclical learning rate algorithm on this particular architecture.

The drawback of this strategy is that it added two additional hyper-parameters: the step size s, that determined the number of iterations required to go from the minimum η to the maximum, and the cyclical learning rate schedule, that defined the way η is modified.

Figure 3 shows the two schedules used for this work: *Triangular* and *Triangular2*; the difference between the two is that the second method halves the upper bound of I at each cycle completion.

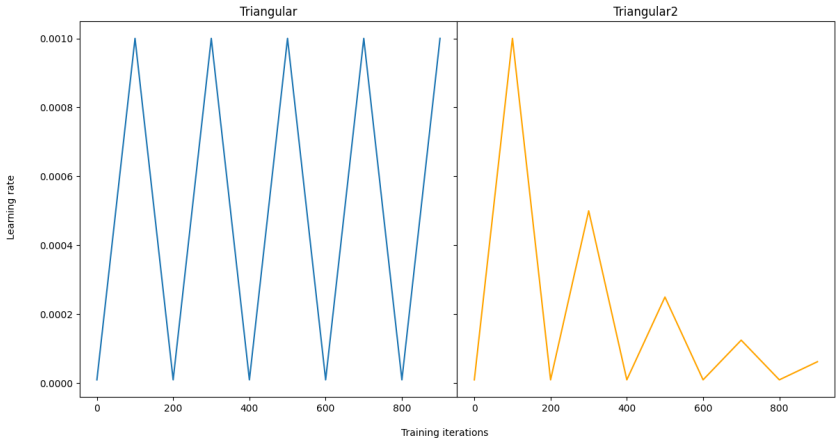

Figure 3: Plot of the learning rate with *Triangular* and *Triangular2* methods

3.4 Experiment 1: Choice of the architecture

The first experiment aimed to find the architecture that could achieve the best accuracy on the test set. This work tested *ResNet18*, *InceptionV3* and *EfficientNetB4*. All of them presented diffent peculiarities and could be used for training with the available hardware.

Since the training on *ImageNet* of the networks is not sufficient to use them as feature extractors, these networks were fine-tuned in order to update their weights and to fit better the task. For each networks the original classi-

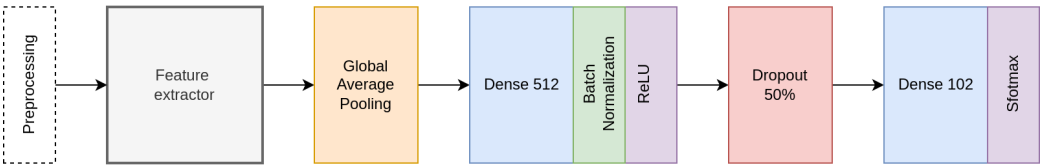

Figure 4: Plot of the learning rate with *Triangular* and *Triangular2* methods

fication layers were removed and substituted with a new classifier composed by the following layers:

1. **Global Average Pooling**: instead of using a Flatten layer that could raise the risk of overfitting and forced the use of Dropout, this layer applied average pooling on the spatial dimensions until each spatial dimension is one, and leaves other dimensions unchanged. It added 0 new parameters to the model

2. **Dense 512**: a dense layer with a **Batch Normalization** before the activation function (ReLU). This helped to avoid vanishing/exploding gradients when training the model. It added $512 \cdot (n + 1) + 4 \cdot 512 = 512n + 2\,560$ parameters

3. **Dropout** as regularization technique for reducing overfitting during training, with a drop rate $\rho = 0.5$. It added 0 new parameters to the model

4. **Dense 102**: a dense layer that used softmax as activation function. It added $102 \cdot (512 + 1) = 52\,326$ new parameters

The total number of parameters added to the original feature extractor is $512n + 54\,886$, where n is the dimensionality of the last layer of the feature extractor.

The strategy adopted for training is the same for all the three networks: each network is trained with 3 different step sizes, two different optimizers and 2 different learning rate schedulers. So each network is trained 12 times in order to find the best configuration. For this phase it was implemented the Early Stopping technique in order to prevent overfit and to speed up the training by stopping prematurely the process. This strategy monitored the validation accuracy and stopped everytime a model could not improve by 1% in the last 10 epochs.

3.4.1 Fine-tuning of ResNet-18

ResNet-18 is a convolutional neural network that is 18 layers deep and accepts images of input size of 224×224. It is a residual network, that is it implements skip connections to help to address the problem of vanishing gradients.

The preprocessing function applied on the images converted them from RGB to BGR, then each color channel was zero-centered with respect to the *ImageNet* dataset, without scaling.

From the output of the *LRF* algorithm the learning rate was varied in the interval $[10^{-5}, 10^{-3}]$ for SGD and $[10^{-6}, 10^{-4}]$ for Adam.

3.4.2 Fine-tuning of InceptionV3

InceptionV3 is a convolutional neural network that is 42 layers deep and accepts images of input size of 299×299. This architecture factorizes the

larger convolutions into smaller ones and and uses an auxiliary classifer to propagate label information lower down the network.

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 $[10^{-5}, 10^{-2}]$ for SGD and $[10^{-3}, 10^{-3}]$ for Adam.

3.4.3 Fine-tuning of EfficientNetB4

EfficientNetB4 is a convolutional neural network that is 237 layers deep and accepts images of input size of 224×224. This type of architecture was built using automatic search and scaling techniques aimed to simultaneously maximize the accuracy of the network and the number of operations per second.

There was no need to apply the ad-hoc preprocessing function because it was embedded in the Keras implementation.

From the output of the *LRF* algorithm the learning rate was varied in the interval $[10^{-5}, 10^{-3}]$ fboth or SGD and Adam.

3.5 Evaluation of the training

To verify the quality of the training two techniques were applied: *grad-CAM* and *Saliency Map*. Confusion matrices as well helped indentifying issues in the first stages of the implementation.

3.5.1 grad-CAM

grad-CAM uses the gradient of the last convolutional layer so that the spatial information are preserved. An heatmap is displayed over the original image that shows the main component identified by the network.

Figure 5 demonstrates that the model could distinguish the flowers from the background and that the background is not an important feature to be learned.

3.5.2 Saliency Map

Like the *grad-CAM*, *Saliency Maps* analyze a specific layer of the model, but in this case it is a classification layer. It shows the gradients of an image with

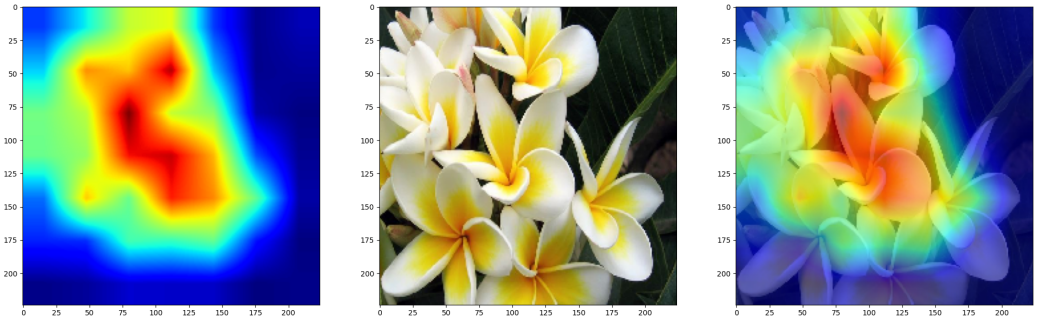

Figure 5: *grad-CAM* of multiple flowers extracted from EfficientNetB4

respect with the label. The map indicates the points that affected more the correct classification of the image.

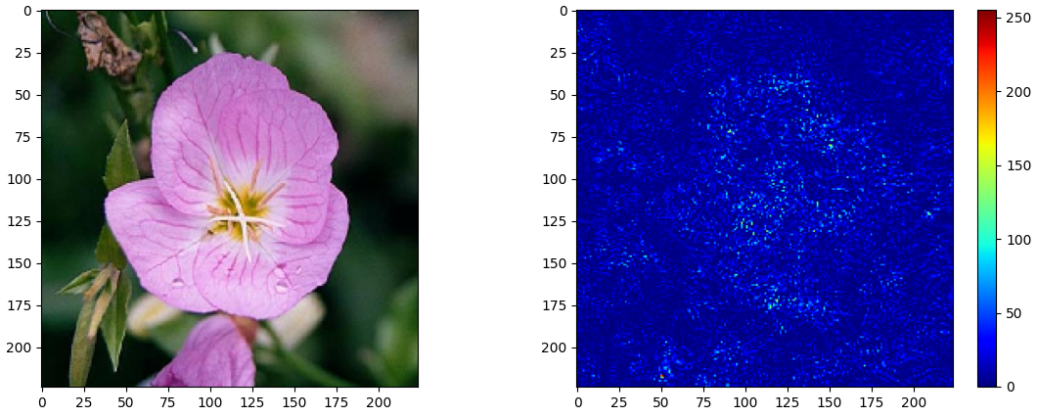

Figure 6: Example of *Saliency Map* extracted from EfficientNetB4

As can be seen in figure 6 most of the pixels that affect the classification is contained within the flower and this is a good indicator of the quality of the model.

3.6 Experiment 2: Freeze layers

The aim of this second experiment was to find a good trade-off between the number of trainable parameters and test accuracy: after choosing the best model from Section 3.4, its layers were frozon so that the weight would not be updated during training. The layers were frozen starting from the input

to the deeper layers and because the training dataset is small and different from *ImageNet* dataset it made sense to freeze only a small portion of layers.

4 Results and Evaluation

This section reports the results obtained on the test set:

Table 1:

Archtecture	**Batch size**	**Optimizer**	**Learning rate**	**Scheduler**	**Step size**	**Test accuracy**
ResNet-18	64	Adam	$[10^{-5}, 10^{-3}]$	Triangular	2	87.64%
InceptionV3	64	Adam	$[10^{-5}, 10^{-3}]$	Triangular	2	91.86%
EfficientNetB4	16	Adam	$[10^{-5}, 10^{-3}]$	Triangualr2	4	94.89%

5 Discussion

The discussion section aims at interpreting the results in light of the project's objectives. The most important goal of this section is to interpret the results so that the reader is informed of the insight or answers that the results provide. This section should also present an evaluation of the particular approach taken by the group. For example: Based on the results, how could the experimental procedure be improved? What additional, future work may be warranted? What recommendations can be drawn?

6 Conclusions

Conclusions should summarize the central points made in the Discussion section, reinforcing for the reader the value and implications of the work. If the results were not definitive, specific future work that may be needed can be (briefly) described. The conclusions should never contain "surprises". Therefore, any conclusions should be based on observations and data already discussed. It is considered extremely bad form to introduce new data in the conclusions.

References

[1] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *Indian Conference on Computer Vision, Graphics and Image Processing*, Dec 2008.

A Exerimental results

Table 2:

ResNet-18			
Optimizer	**Scheduler**	**Step size**	**Test accuracy**
SGD	Traingular	2	
		4	
		8	
	Triangular2	2	
		4	
		8	
Adam	Traingular	2	**86.39%**
		4	85.78%
		8	85.65%
	Triangular2	2	82.48%
		4	
		8	

Table 3:

ResNet-18			
Optimizer	**Scheduler**	**Step size**	**Test accuracy**
SGD	Traingular	2	
		4	
		8	
	Triangular2	2	
		4	
		8	
Adam	Traingular	2	**86.39%**
		4	85.78%
		8	85.65%
	Triangular2	2	82.48%
		4	
		8	

Table 4:

ResNet-18			
Optimizer	**Scheduler**	**Step size**	**Test accuracy**
SGD	Traingular	2	
		4	
		8	
	Triangular2	2	
		4	
		8	
Adam	Traingular	2	**86.39%**
		4	85.78%
		8	85.65%
	Triangular2	2	82.48%
		4	
		8	