Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Flowers recognition

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

January 25, 2023

Abstract

This work describes a supervised approach to classification of images on a small dataset, but with a large number of classes. In particular, it is shown how the combination of modern training techniques and recent developments in the field of research of architectures for neural networks are able to obtain high performances even when the model is partially trained.

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 then trying to freeze the model such that it could maintain 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 8189 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 1020 images in the training set, 1020 images in the validation set and 6149 images in the test set.

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. Figure 2 plots the distribution of the three datasets: the distribution of

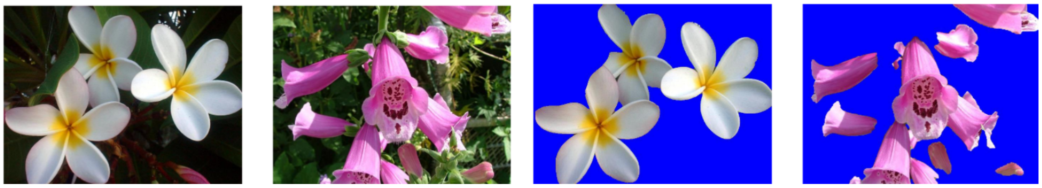


Figure 1: Training images and their segmentation

the samples is non-uniform across the categories, with a mean $\mu \simeq 60$ and standard deviation $\sigma \simeq 44$. In fact some classes had more than 200 samples, like 50 (*Common Dandelion*) and 78 (*Lotus*) and other just 20 per class, like 1 (*Pink Primrose*), 17 (*Purple Coneflower*) or 23 (*Fritillary*).

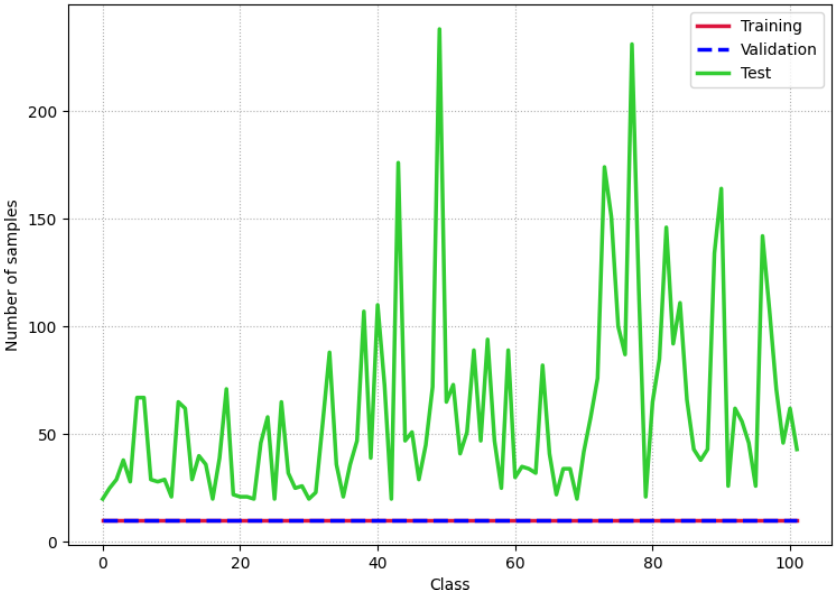


Figure 2: Distribution of the samples for each category in each dataset

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 account: the first one aimed to find the best CNN[1] architectures in literature that could fit the available hardware (see Section 3.1) and fine-tuned to accomplish the task; the second one tried to find a good trade-off between number of trainable layers, training time and accuracy by freezing the layers' weights during training.

All the proposed architecture were pre-trained on *ImageNet*[2] and used as feature extractors for a new classifier that operates over 102 classes (see Section 3.4 for details about the architecture).

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 calculations.

3.2 Data Augmentation and Preprocessing

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`[2] 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 $\gamma \in [0.7, 1.3]$

[1] Convolutional Neural Network

[2] Although it is deprecated, the API is easy to use and already optimized. The suggested way with `tf.keras.utils.image_dataset_from_directory` and `tf.data.Dataset` is more flexible, but harder to optimize properly. `ImageDataGenerator` API was preferable in terms of development speed.

- 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 classifier on the extracted features it was decided to use the SGD [3] with momentum $\beta = 0.9$ or Adam [4] 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"* [5]. 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 $J = [10^{-10}, 10^{-1}]$ 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 3 for optimizer Adam and the architecture described in section 3.4.1.

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

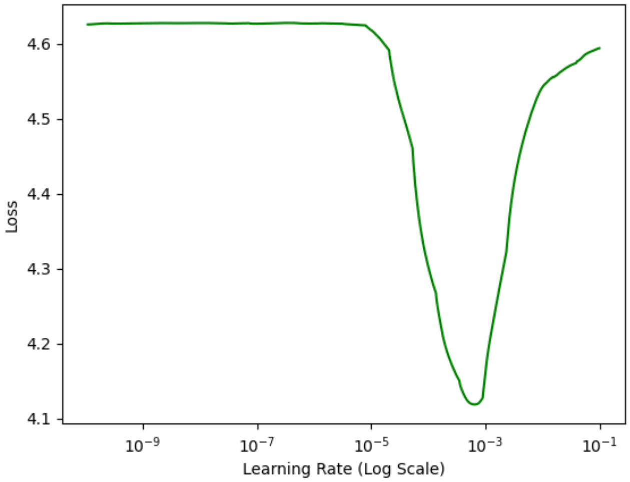


Figure 3: Output of the learning rate finder algorithm. $I = [10^{-5}, 10^{-3}]$ is the optimal solution

to go from the minimum $\eta \in I$ to the maximum, and the cyclical learning rate schedule, that defined the way η is modified.

Figure 4 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.

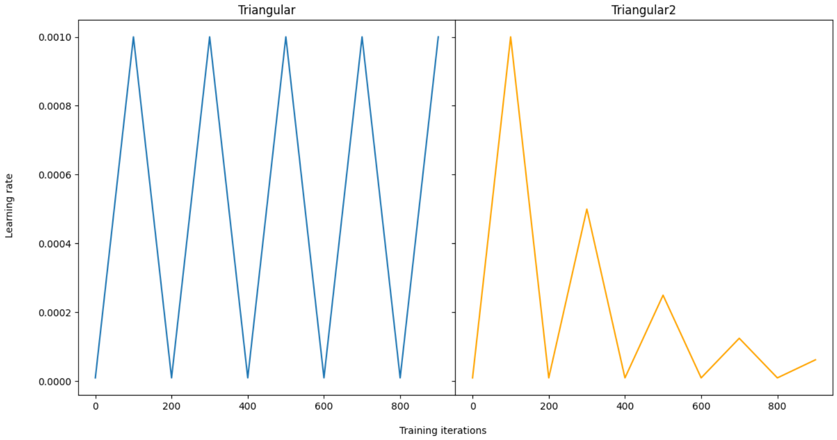


Figure 4: Plot of the learning rate with *Triangular* and *Triangular2* methods

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* [6], *InceptionV3* [7] and *EfficientNetB4* [8]. 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.

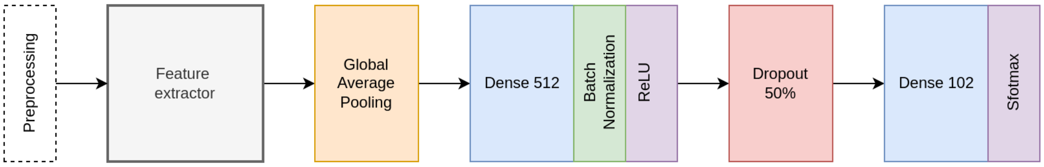


Figure 5: High level architecture of the network: in grey the pretrained network used as feature extractor, followed by the classifier's layers.

For each networks the original classification 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 $512 \cdot (n+1) + 4 \cdot 512 = 512n + 2560$ 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 $102 \cdot (512+1) = 52326$ new parameters

Figure 5 gives an overview of the architecture described above. The total number of parameters added to the original feature extractor is $512n+54886$, 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 $s \in \{2, 4, 8\}$, two different optimizers (*SGD* and *Adam*) and 2 different learning rate schedulers (*Triangular* and *Triangular2*). 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 overfitting 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 0.5% 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 $[10^{-4}, 10^{-2}]$ 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* [9] and *Saliency Map* [10]. Confusion matrices [11] 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.

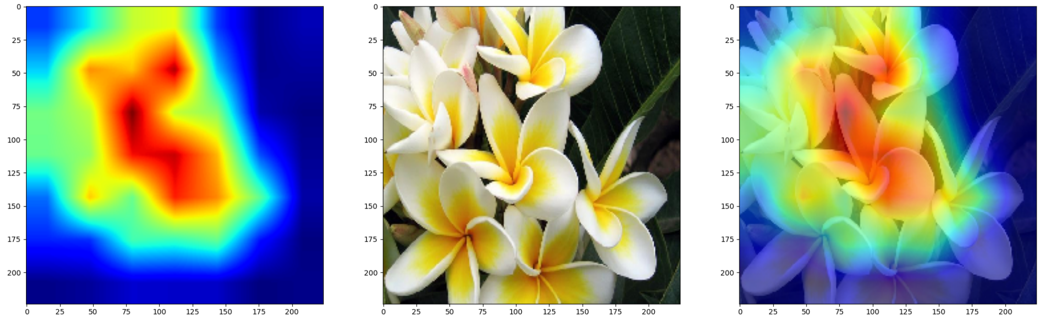


Figure 6: *grad-CAM* of multiple flowers extracted from EfficientNetB4

For example Figure 6 demonstrates that the model could distinguish multiple 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 respect with the label. The map indicates the points that affected more the correct classification of the image.

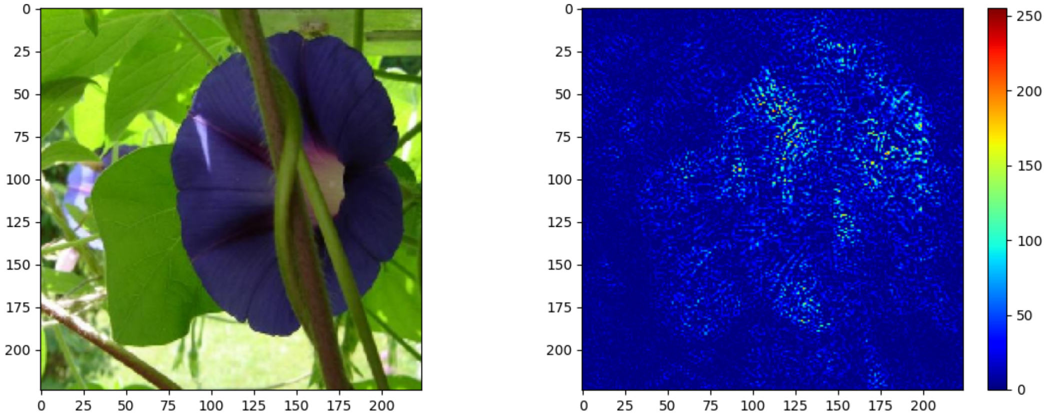


Figure 7: Example of *Saliency Map* extracted from EfficientNetB4 of a flower with an obstacle in front of it

As can be seen in figure 7 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, test accuracy and training time: after choosing the best model from Section 3.4, its layers were frozen 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 was expected to have better performances when only a small portion of layers is locked. Only the feature extractor was affected by this process, while the classificator reamined fully trainable. This allowed to freeze from 10% to 100% of the feature extractor's layers.

The results were evaluated considering, in order of importance, the accuracy on the test dataset, the training time and the number of trainable parameters. These three parameters were chosen because the first two could be strictly related, but the third and the other two could present some interesting relation *e.g.* the training time could be stable before dropping along with the accuracy.

4 Results and Evaluation

This section reports the results obtained for both the experiments.

4.1 Experiment 1

Table 1 shows the configuration that gave the best results for each architecture. Figure 8 shows the confusion matrix on the test set for EfficientNetB4.

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	$[10^{-4}, 10^{-2}]$	Triangular	2	91.86%
EfficientNetB4	16	Adam	$[10^{-5}, 10^{-3}]$	Triangualr2	4	94.89%

Appendix A shows in tabular form all the results from the 12 trainings of each model.

4.2 Experiment 2

It is immediately noticeable that EfficientNetB4 from the first experiment performed better than the other models and for this reason it was chosen for the second experiment. Figure 9a compares the accuracy on the test dataset and the number of trainable parameters as the number of frozen layers increased. Figure 9b compares accuracy and training time with respect to the number of frozen layers.

In order to have an accurate estimation of the timings the *Early Stopping* strategy was disabled because it could terminate prematurely the training process. This limitation afflicted the accuracy because the model could have overfitted more than the regularized version.

5 Discussion

Despite the shallowness of *ResNet-18*, the model based on it performed with quite high accuracy ($\sim 88\%$) but it was predictably surpassed by the other

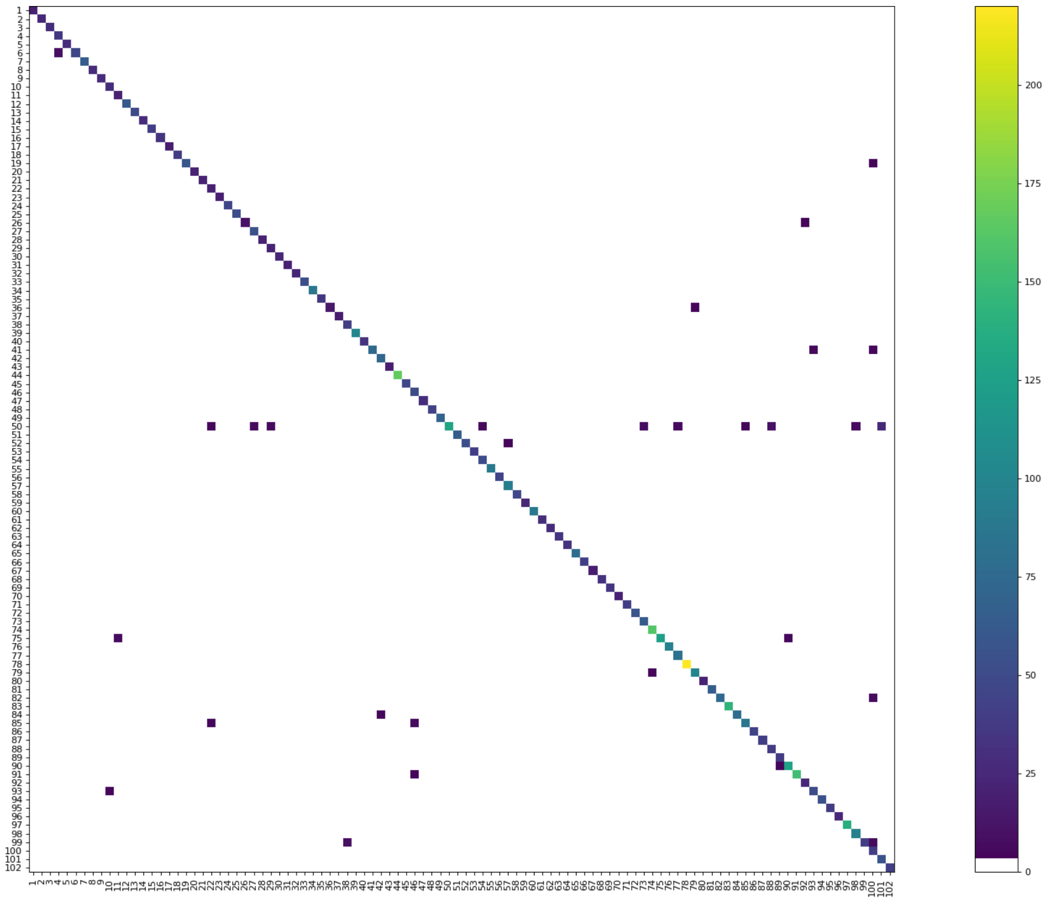
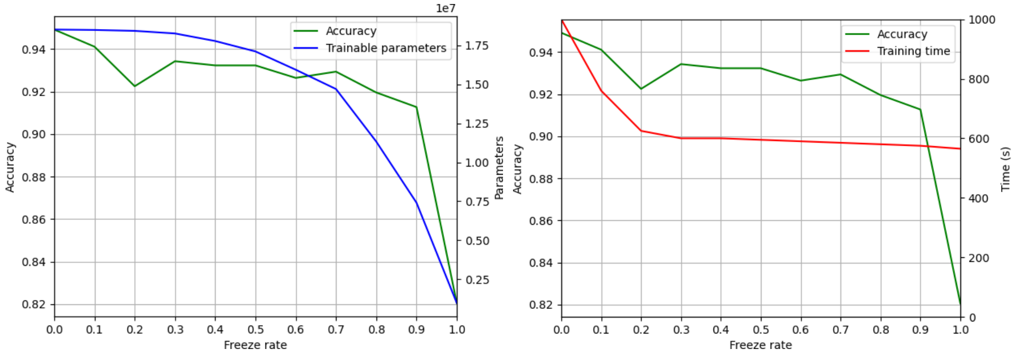


Figure 8: Confusion matrix of EfficientNetB4 on the test dataset

two models. In particular EfficientNetB4, that performed better on *ImageNet* [8], was more efficient than *InceptionV3* in terms of accuracy and number of parameters. The main obstacle represented by a small dataset was overcome by data augmentation: all the models performed with an average of +3% in accuracy. The *Transfer Learning* also played a fundamental role and without it the model would had been less stable and the training would had required more time before reaching the same qualitative results.

Figure 8 shows that the model had more problems dealing with class 50 (*Common Dandelion*) in the test dataset; this was the class with more samples and some images contained insects landed on the flower and in others the background covered more than half of the image. A strenght point of the



(a)

(b)

Figure 9: Plot of the accuracy compared to the number of trainable parameters (a) and compared to training time (b) as the number of frozen layers varies

model is that is able to recognize obstacles and foreign objects like in Figure 7, where the *Saliency Map* shows that the gradient were smaller along the branch.

In the second phase models with less frozen layers performed better as expected: the shallow layers contained generic knowledge learned from *ImageNet*, like lines, basic shapes, *etc.* and they were useful for this new task. As the number of frozen layers increased more complex concepts reamained stored in the network nad they were less useful to the classifier and thus the accuracy started to drop. This behaviour is shown in Figure 9. The reason why on 20% the accuracy dropped and then raised again it could be related to the fact that a block of layers was partially frozen and then less consistent during the training *i.e.* the entire block could be capable of learning a feature, but a part of it learned from a dataset and the other from another.

Finally the best model in terms of trade-off between accuracy, training time and number of trainable parameters had 30% of frozen layers.

In order to improve the experiments the segmented images could be used as well, but in a parallel gated input so that the network could learn that the images without background contain just additional information about the original image and that they are not part of the original training dataset. In this way the model could reinforce the concepts learned but it could be less flexible if the secondary input layer is not designed carefully.

An interesting evolution of the second experiment is the implementation of pruning strategies: in this way it would be possible to find an optimal network that can be deployed in smaller devices and have advantages in terms of time of inferece.

6 Conclusions

This work proved that images similar to each other belonging to a large number of classes was possible even with a small dataset; this result was achieved thanks to *Transfer Learning* and *Data Augmentation* techniques, that allowed the models to preserve, fine-tune concepts learned on other datasets and made the network not overfitting due to data scarcity.

Even shallow model could reach high accuracy on a dataset six time larger than the one used for training and more complex ones could almost reach 95% in accuracy. The use of automatic procedures, like the cyclical learning rate, speeded up the search for an optimal solution in the space of hyper-parameters, but the human intervention was still needed when it came to make high level choices.

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.

[2] J. Deng, W. Dong, R. Socher, L. Li, Kai Li, and Li Fei-Fei, “Imagenet: A large-scale hierarchical image database,” in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, June 2009, pp. 248–255.

[3] J. Kiefer and J. Wolfowitz, “Stochastic estimation of the maximum of a regression function,” *Ann. Math. Statist.*, vol. 23, no. 3, pp. 462–466, 09 1952. [Online]. Available: https://doi.org/10.1214/aoms/1177729392

[4] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” 2014.

[5] L. N. Smith, “Cyclical learning rates for training neural networks,” 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” 2015.

[7] C. ”Szegedy, W. ”Liu, Y. ”Jia, P. ”Sermanet, S. ”Reed, D. ”Anguelov, D. ”Erhan, V. ”Vanhoucke, and A. ”Rabinovich, “Going deeper with convolutions,” 2015.

[8] M. Tan and Q. V. Le, “Efficientnet: Rethinking model scaling for convolutional neural networks,” 2019.

[9] 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,” *International Journal of Computer Vision*, Oct 2019. [Online]. Available: http://dx.doi.org/10.1007/s11263-019-01228-7

[10] K. Simonyan, A. Vedaldi, and A. Zisserman, “Deep inside convolutional networks: Visualising image classification models and saliency maps,” 2013.

[11] S. V. Stehman, “Selecting and interpreting measures of thematic classification accuracy,” *Remote Sensing of Environment*, vol. 62, no. 1, pp. 77–89, Oct. 1997. [Online]. Available: https://doi.org/10.1016/s0034-4257(97)00083-7

A Exerimental results

Table 2

ResNet-18			
Optimizer	**Scheduler**	**Step size**	**Test accuracy**
SGD	Traingular	2	76.27%
		4	75.28%
		8	76.04%
	Triangular2	2	26.63%
		4	50.77%
		8	66.38%
Adam	Traingular	2	**86.39%**
		4	85.78%
		8	85.65%
	Triangular2	2	82.48%
		4	84.85%
		8	84.72%

Table 3

IneptionV3			
Optimizer	**Scheduler**	**Step size**	**Test accuracy**
SGD	Traingular	2	76.27%
		4	75.28%
		8	76.04%
	Triangular2	2	26.63%
		4	50.77%
		8	66.38%
Adam	Traingular	2	**86.39%**
		4	85.78%
		8	85.65%
	Triangular2	2	82.48%
		4	84.85%
		8	84.72%

Table 4

EfficientNetB4			
Optimizer	**Scheduler**	**Step size**	**Test accuracy**
SGD	Traingular	2	76.27%
		4	75.28%
		8	76.04%
	Triangular2	2	26.63%
		4	50.77%
		8	66.38%
Adam	Traingular	2	**86.39%**
		4	85.78%
		8	85.65%
	Triangular2	2	82.48%
		4	84.85%
		8	84.72%