Università degli studi di
Milano-Bicocca

Advanced Machine Learning

Final Project

Transfer Learning on Oxford 102 Category Flower Dataset

Author:
Simone Giuseppe Locatelli, 816781,
s.locatelli29@campus.unimib.it

February 21, 2022

Abstract

The study of flower classification system is a very important subject in the field of Botany, also flower plays an extremely important role in our life, which has both high value in research and applications. However, because of the complex background of flowers, the similarity between the different species of flowers, and the differences among the same species of flowers, there are still some challenges in the recognition of flower images. The traditional methods are mainly based on the three features: color, shape and texture. These methods needs people to manually select features, thus handcrafting a proper features space. In this paper, we investigated the use of Convolutional Neural Networks (CNN) leveraging the Oxford 102 flowers dataset in variuos approaches, such as: building a base model trained from scratch or adapting a pretrained model, DenseNet-169, on a new dataset as feature extractor or as fine-tuned model; and see we how the use of transfer learning and optimization techniques can help improving the accuracy of flower classification. The results show that transfer learning can overcome the basic workflow: compared with the traditional methods, the accuracy of flower recognition on Oxford flowers dataset is improved, and has better robustness and generalization ability.

1 Introduction

In the environment the flowers surround us, they play such an important role for many activities ranging from research to simply curiosity. Therefore, a good understanding of flowers is essential to help in identifying several species when came across. Because of the importance of this task, many different approaches have already been developed. Methods like Scale Invariant Features [1], Histogram of Oriented Gradients [2] or manual feature engineering were initially used as features in order to train other classifiers, such as Support Vector Machines. Nowadays state-of-art performance is achieved by the use of Convolutional Neural Networks, they have ensured the demand of robustness and generalization and have removed the need of hand crafted features, but, in order to achieve these objectives, a lot of data is required for training. The literature comes in aid thanks to the Oxford 102 flowers dataset [3] having 8.189 images divided into 102 categories. But because as it will be shown all those images are not sufficient to train a model from scratch, pre-trained model on other similar task will be used. To summarize, the work

done in this project will show different approaches mainly based on transfer learning, and see to what extent which one performs better comparing the results obtained.

2 Dataset

The dataset chosen for this task is the Oxford 102 flower dataset by the Visual Geometry Group [3]. The dataset consist of 8189 images of 102 flower's categories. The images have large scale, pose and light variations. In addition, there are categories that have large variations within the category and several very similar categories, making the problem fall into the Fine Grained Visual Classification (FGVC) category. The datasets is composed of the images (of different sizes), and other useful resources that will define each split of the dataset (train, validation and test), and the labels of each image. As we can see from Fig.1 below the training set is balanced, as well for the validation set, both contains 10 images per class (1020 images respectively), while the test set is way larger and unbalanced (6149 images).

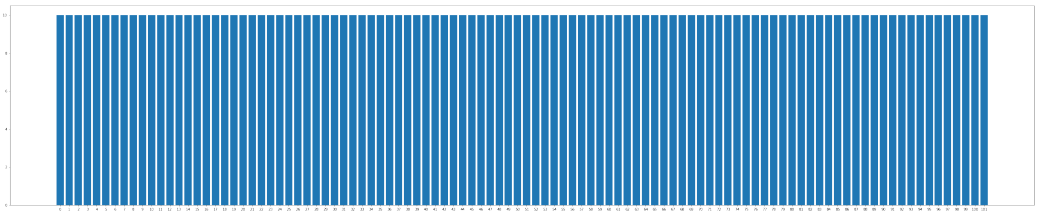


Figure 1: Histogram over the number of images in each class in the dataset.

Examples of images in the dataset are shown below.

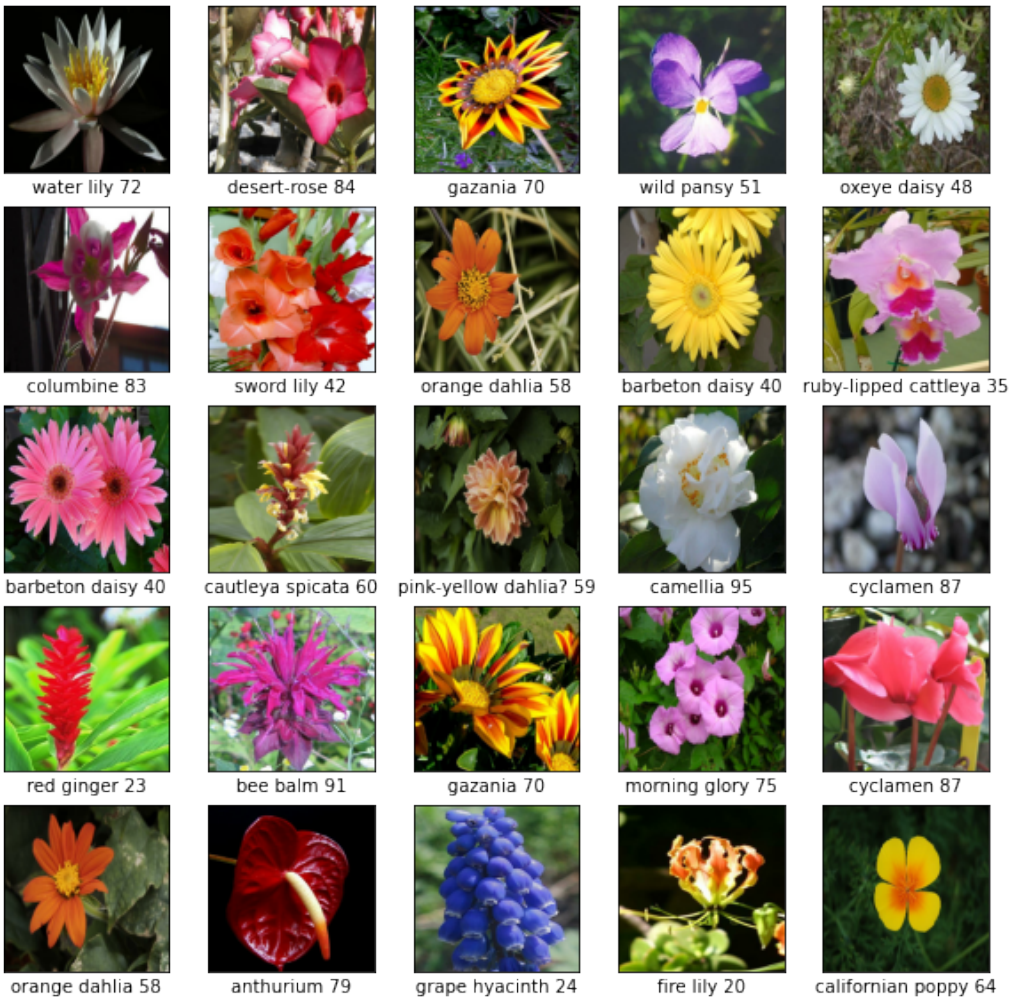


Figure 2: Examples of image in the dataset with associated labels.

3 The Methodological Approach

In this section the project development process is presented. Moreover, a mention of the tools that have allowed the development of this project must be made. Python was the only programming language used with some useful libraries, such as: Tensorflow, Scikit-Learn and KerasTuner and others. The hardware platform was both based on a local laptop and on cloud thanks to Google Colab.

3.1 Dataset pre-processing and Data Augmentation

Since the dataset was already balanced, it was not touched, it was only divided into the appropriate splits to facilitate subsequent operations. As already mentioned, the test split is way larger, 6 times than the train and validation ones, this led some difficulties in the training phase of some models because of the lack of data necessary to avoid overfitting or underfitting. But, in order to make correct comparision between other models it was left unchanged. In order to properly use the dataset some preprocessing was done. First, each image have been rescaled in the range of [0-1], resized to 224x224 pixels and pre-processed in order to be used by a CNN (normalization to 0 mean and unit standard deviation). Also, the labels were one-hot encoded. Next, only for images belonging to training set, was applied some Data Augmentation. In particular was applied a random horizontal flip and a random rotation, changes in the hue of the images were not made because of the importance of this feature in this domain. Applying a Data Augmentation on the training set helps the model being more robust and able to learn a more generalized feature representation, acting as an implicit regularizer.

3.2 Training settings

For the training phase of the CNNs different hyperparameter were selected. As optimization procedure the Adam[4] algorithm with a learning rate of 0.01 for the model trained from scratch and 0.001 for the other experiments, was selected, while a Categorical Cross-Entropy as loss function because of the one-hot encoded labels and because of the multiclass classification problem:

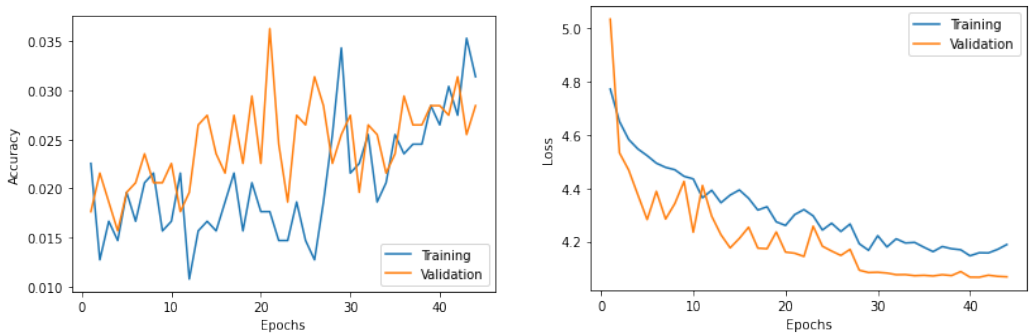
$$\mathrm{CE} = -\sum_{i=1}^{N} \mathrm{y_i} \cdot \log \hat{\mathrm{y}}_\mathrm{i} \tag{1}$$

where $\hat{\mathrm{y}}_\mathrm{i}$ is the i-th scalar value in the model output, $\mathrm{y_i}$ is the corresponding target value, and N is the number of scalar values in the model output (the number of labels). For the training phase we set the epochs to 50. Other regularization techniques were used: an Early Stopping criteria, with a patience of 8 epochs and minimum improvement of 0.001, and a learning rate scheduler that will reduce the learning rate when the metric evaluated (the validation loss) has stopped improving after 5 epochs by a factor of 0.1.

3.3 A model from scratch

The starting point for the evaluation of the performances has been done training a CNN from scratch. The small training dataset will not help this process, but it is needed to demonstrate how transfer learning can be useful in such cases. For this task was built a model with about 1M parameters, more precisely the model is composed of:

- An input layer with shape (224, 224, 3).
- 3 consecutive blocks defined as:
  - A convolutional layer ranging from 64 channels as output space to 256 of the last block, a kernel size of 3x3 with a stride of 2 on both dimensions, a ReLu activation function to leverage non-linearity.
  - A Max pooling layer with size of 2x2 to extract the maximum activations. In total we have: 1792, 73856, 295168 parameters for each block respectively.
- A flattening layer to flatten our tensor to obtain a 1-D vector.
- A fully connected layer with 256 neurons, a kernel weight matrix initialized from a uniform distribution and a ReLu activation followed by a Dropout layer (only in training phase) with 30% of probability to introduce some noise (switch off a neuron) and so making the model more robust, totalling 590080 parameters.
- Same as before, this time with half neurons and a Dropout with probability of 20%, with 32896 parameters.
- A Normalization layer to normalize the output of the previous layer (and so minimizing the variance), 512 parameters.
- Lastly, the final classifier with 102 neurons and the Softmax output function because of the multiclass classification problem, 13158 parameters.



a) Accuracy b) Loss

Figure 3: Accuracy and Loss trends during training of the baseline model.

3.4 Tranfer Learning

The poor performance obtained with the previous method guide the workflow through the use of Transfer Learning to exploit pretrained models. When adapting a pretrained CNN to new datasets, we can either use it directly as a feature extractor [5] in order to train another classifier, or fine-tune it [6]. The backbone chosen for these tasks was a DenseNet-169 [7] because of its robust and lightweight architecture. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters, moreover, the large density alleviates overfitting problems making it a suitable model for this problem with such a small dataset.

3.4.1 Feature extractor

Different cutting points have been chosen for the feature extraction, specifically from (in ascending order):

1. Layer "*conv5_block25_1_bn*" a Batch Normalization layer right after a Convolutional one with a feature vector, once flattened, of dimension 6.272

2. Layer "*conv5_block30_1_bn*" like above but higher in the architecture structure

3. Layer "*avg_pool*" which is the Average Pooling layer right before the Dense classifier with a feature length of 1.664

Since the dataset is small and quite similar to the one where the model was trained on, features extracted from higher layer should be more expressive. Once the features have been extracted, a Support Vector Machine with a linear kernel was trained on them so we can evaluate how good they are in the classification task on the new dataset. The best performance were obtained with the "*conv5_block30_1_bn*" features which led to an accuracy of 24% in the validation phase and 20% in the test phase, a great improvement.

3.4.2 Fine-tune

Fine-tuning a pretrained model could be made in 2 different ways:

1. Freezing the backbone model (meaning that its weights remains fixed through the training) and to use it as a feature extractor for a trainable classifier built on top of it.

2. Fine-tune the weights of the top layers of the pre-trained model alongside the training of the classifier added, in this way the training process will force the weights to be tuned from generic feature maps to features associated specifically with the new dataset.

Freezed Backbone Because of the small dataset, we take advantage of features learned by a model trained on a larger dataset: this is done by instantiating the pre-trained model and adding a fully-connected classifier on top. The pre-trained model is "frozen" and only the weights of the classifier get updated during training. In this case, the DenseNet-169 base extracted all the features associated with each image, thus a classifier with 102 neurons determines the image class given the features extracted. The image below shows the concept behind this technique.

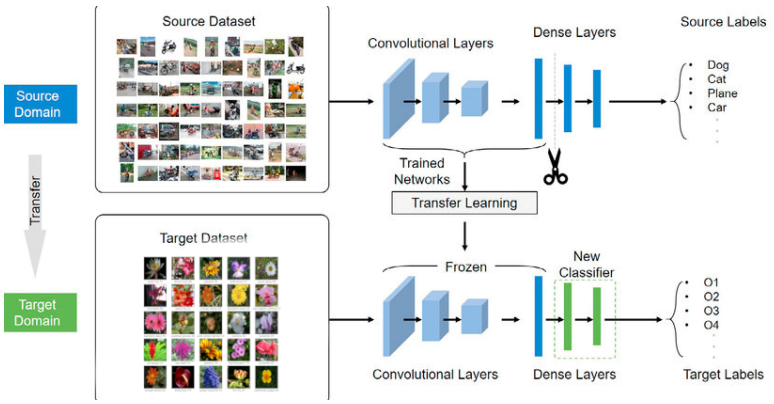
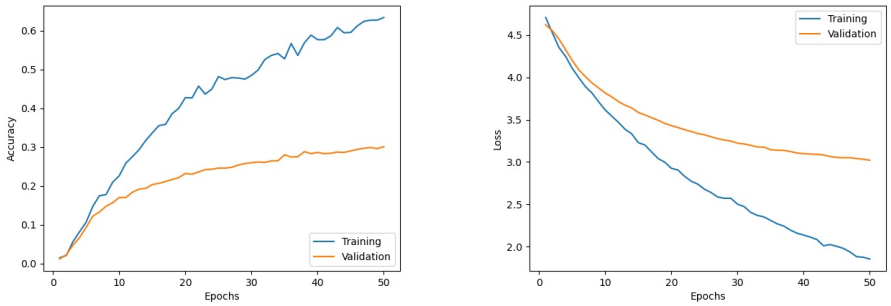


Figure 4: Fine-tune with freezed backbone.
Source: https://www.researchgate.net/figure/The-architecture-of-our-transfer-learning-model_fig4_342400905

As we can see from Fig.5, with this approach the performance of the classification have been improved.



a) Accuracy b) Loss

Figure 5: Accuracy and Loss trends during training with a freezed backbone.

Fine-tuning Even if this technique is usually recommended when the training dataset is large and similar to the original one that the pre-trained model was trained on, to further improve performance, the top-level layers (the last dense block) of the pre-trained models were repurposed to the new dataset via fine-tuning, in this way, the weights are tuned such that the model learned high-level features specific to the dataset.

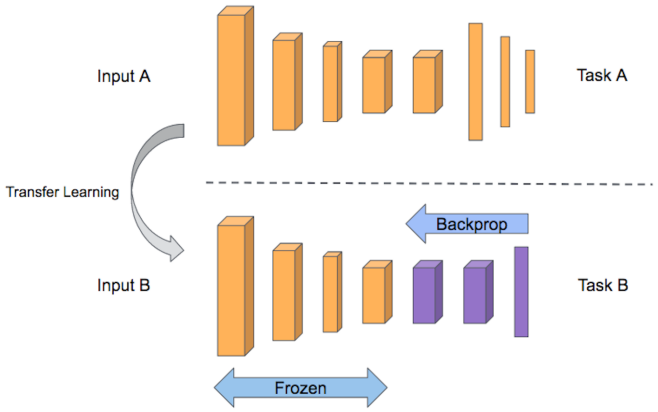
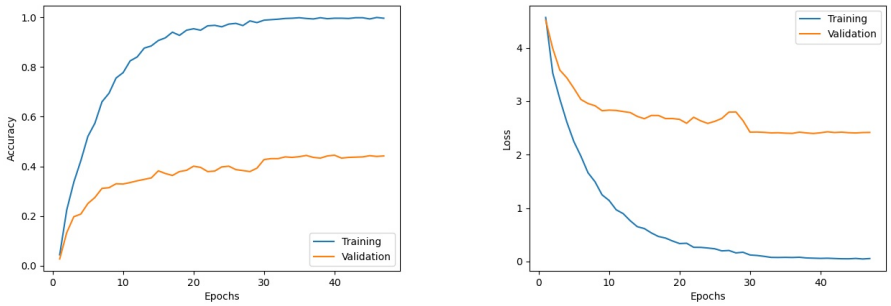


Figure 6: Fine-tuning example.
Source: https://paperswithcode.com/task/transfer-learning

As the Fig.7 shows, counterintuitively, some new improvements have been obtained.



a) Accuracy b) Loss

Figure 7: Accuracy and Loss trends during training of fine-tuned model.

3.5 Hyperparameter optimization of fine-tuned model

Driven by the good results obtained in the fine-tuning procedure, another approach was tested. We have combined that approach with an hyperparameter optimization algorithm in order to obtain the best model architecture. In particular, the DenseNet169 backbone was left unchanged, what we were trying to optimize was from which dense block tune the weights and

the classifier built on top of it. The hyperparameters to be optimized were the following:

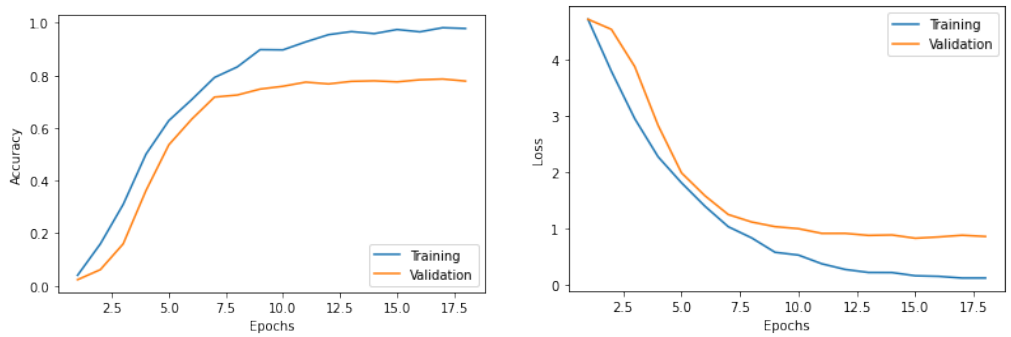
- The number of Dense block with tunable weight: from [last, penultimate, third last]
- For each Dense Layer:
  - The number of Dense layer: [1, 2, 3]
  - The activation function: [ReLu, LeakyReLu]
  - The dimension of the layer: [64, 128, 256]
  - The Dropout layer [0, 0.2, 0.4]
- Optimization algorithm: [Adam, RmsProp]
- Learning rate: [0.01, 0.001, 0.0001]

Of course, on top of what we got there will be the classification layer with 102 neurons. The search space was big enough (given the hardware limits) to opt for a Bayesian optimization, thus, exploiting the Bayes Theorem, we do not have to inspect all the search space. After the optimization procedure, which took 34 trials, these were the hyperparameters that gave the best results:

Table 1: Best hyperparameter obtained from Bayesian Optimization.

Hyperparameter	Choice
Dense block	third last
Dense Layer	1
Activation	ReLu
Layer size	256
Dropout	0.4
Optimization	Adam
Learning rate	0.0001

After the training, lasted only 18 epochs for the Early Stopping, as we can see from the trends below, this approach boost the performance even with a poor training set.



a) Accuracy b) Loss

Figure 8: Accuracy and Loss trends during training of the optimized model.

4 Results and Evaluation

For the evaluation of the results obtained during the experiments for the classification of 102 flowers classes, the test dataset provided us with 6149 images was used. The metrics chosen to discriminate the models is the accuracy. The graph below summarizes the performances obtained from all the trials.

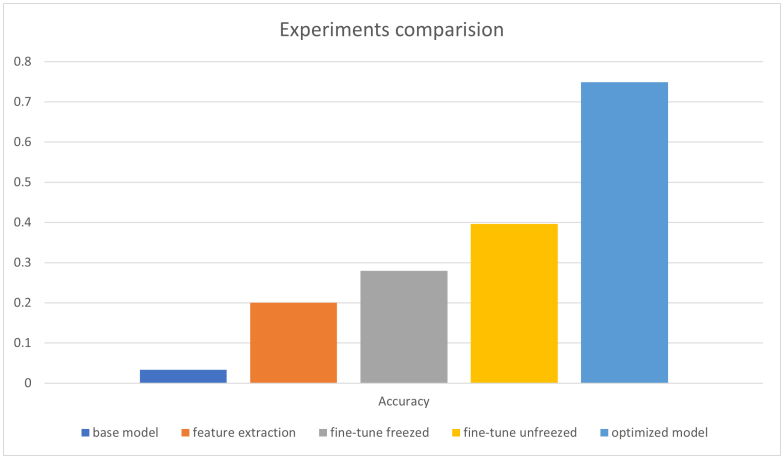


Figure 9: Accuracy obtained in the experiments.

Table 2: Accuracy obtained in the experiments.

	Base model	Feature extraction	Fine-tune freezed	Fine-tune unfreezed	Optimized model
Accuracy	0.0331	0.2	0.28	0.396	0.749

5 Discussion

As shown in Fig.9, going on with the experiments we managed to obtain better and better performances. The model trained from scratch performed poorly, 3.3% of accuracy: the dataset, in particular, the training set was too small to let the model learn the correct weights and thus be able to generalize on unseen data. Using a pretrained model, a DenseNet169, as feature extractor in order to later train an SVM, has proved to be a correct approach, but due to the different context and the limiting dataset only 20% of accuracy was obtained. The descriptors used were too much tuned on the original task, hence not expressive enough to allow a meaningful representation. So, tuning the model could be an effective approach: the first trial where the backbone was completely freezed and only the weights of the classifier built on top were trainable took us to 28% of accuracy, while allowing even the last dense block of DenseNet169 to be trainable has led to even better results: 39.6%. In each trial, as shown in Fig.5 and Fig.8 respectively, there are clear evidence of underfitting since the models given the small dataset cannot continue learning. The best model turned out to be the one optimized on the new problem with a final accuracy of 74.94%. The ability to search first from which layer tune the weights and second for an optimal architecture of the classifiers at the top of the model, in addition to the previous knowledge of DenseNet169, have allowed to exceed the results obtained through handcrafted features in [3]. Of course, manipulating the dataset to make it more available for the training of CNNs, thus increasing the training set and reducing the test set leaving the distribution of images unaffected, can be a solution to improve overall performance. Also, the availability of having a more performing hardware would have allowed to test more configurations both in the architecture/training of the model and in the hyperparameter optimization.

6 Conclusions

This project presented a way to address flower image classification where data poorness presents a major limitation. However, the results obtained are encouraging and show how the use of deep learning techniques can outperform a more traditional approach constituted by features engineering. Moreover, the previous knowledge of a pretrained DenseNet169 combined with classifiers appropriately optimized, proved to be a compelling approach. In this sense, this framework could be generalized to other datasets with low cardinality. Future developments could consider an enriched dataset composed of a larger set of different flower images to improve generalization capabilities, and oriented towards attention mechanism, like self-attention implemented in Visual Transformers, in order to exploit more global informations.

References

[1] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1. Ieee, 2005, pp. 886–893.

[3] M.-E. Nilsback and A. Zisserman, "Automated flower classification over a large number of classes," in *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*. IEEE, 2008, pp. 722–729.

[4] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[5] A. Sharif Razavian, H. Azizpour, J. Sullivan, and S. Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 806–813.

[6] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on knowledge and data engineering*, vol. 22, no. 10, pp. 1345–1359, 2009.

[7] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993