NOVA IMS Information Management School

Deep Learning Project

MASTER DEGREE PROGRAM IN DATA SCIENCE AND ADVANCED ANALYTICS

Flower Classification

GROUP 14

Inês Ribeiro m20210595

José Dias m20211009

Matias Neves m20211000

Nicola Andreatta m20211297

Peter Hamori m20191467

NOVA Information Management School
Instituto Superior de Estatística e Gestão de Informação
Universidade Nova de Lisboa

INDEX

1. Introduction

In this project, Convolutional Neural Networks were implemented to build a model able to classify flower species from a dataset of images.

1.1 Motivation

There are countless flower species on earth. Recognising and classifying them is a hard task without botany knowledge, as well as time-consuming if flower manuals are used. With the following project we approached this classification problem by implementing an image classification algorithm to solve this task and automate the process. The output was a system able to provide the flower species with the smallest error possible.
Hence, this could meet the needs of inexperienced people simply by providing a picture as an input to the system. An appropriate recognition of flower species is also essential for biodiversity protection. In fact, flowers are the main source of nourishment for insect pollinators and such a tool would help mankind to safeguard the habitat of these insects.

1.2 Task definition

The purpose of the current work was to build a Convolutional Neural Network (CNN) model to perform multi-class image classification to differentiate between five different types of flowers. To build, validate and assess CNN models, publicly available data was used from the Kaggle dataset " Flower Recognition CNN Keras" *(1)*. This dataset consists of 4326 images, featuring the following flower species: daisy, dandelion, rose, sunflower and tulip.
Upon feeding an image as input to the CNN models, the expected output was the label of the class that the model predicts. The dataset was slightly imbalanced, a property which was taken into account for throughout the development of the models.

2. Data Preprocessing

Resizing: The images of the dataset came in different shapes regarding height and width. Since neural networks receive inputs of the same size, the initial motion of preprocessing was to resize the images. Based on interpolation methods, some resizing tools can produce relatively accurate representations of images with arbitrary height and width parameters. First of all, a model was developed using a fixed size of 150x150 pixels, commonly used in CNNs. Nevertheless, in a later stage it was realized that the variety of the ratio of images in the data set was not negligible, hence pictures with a ratio much different from 1:1 (i.e. square ratio) were subjected to a considerable stretch.

The main intent was to select the most suitable height and width in a way that required the least amount of modifications overall, in order to preserve the integrity of the picture. Other than this, it was taken into account the RAM size provided by Google Colaboratory. Given these considerations, it was selected the largest size possible of those that have *height / width = 0.75*; said size was half of the average size across all images. The third dimension of each image was kept unchanged, as that represented the RGB channels. The final input image size was: (126, 169, 3).
Scaling: Secondly, the images were rescaled according to the concept of Min-Max Scaling. Given that the range of pixel values was [0; 255], the images were altered by dividing the value of each pixel by 255, thus shifting the domain of the pixel values to the [0; 1] interval.
Train-test split: Thirdly, the data set was divided into three distinct sets. The training, validation and test sets held 60, 20 and 20 percent of the images, respectively. The test set was set aside throughout the whole training process. Some of the models were trained directly using the original training and validation sets. Others were trained on an augmented

training and an augmented validation set, which were constructed by performing transformations to the original sets. These transformations were rotation, zooming, width - and height - shifting, and horizontal flipping.
One-hot encoding: The target labels were one-hot encoded so that they matched the size of the outputs of the models. The CNNs were constructed in a way that their output was a five element long vector. The i-th element of an output vector indicated the probability of the given input image to belong to class i. The error of the prediction of a CNN could then be derived as the summation of element-wise differences between the output vector and the target vector.
Dataset validation: An additional challenge regarding the target labels was the following: after scrolling through the images manually, it was found that some of them were misclassified, showing either none of the five flowers (e.g.: a house) or showing a different flower than the one indicated by the corresponding target label. The image set used was not collected by the group, so it would be important to find a way to validate the images and the labels. After further research on this topic, it was concluded that no effective algorithm exists that accounts for this type of sanity check, so the only option was to do this task manually. However, the amount of misclassified images in each category was not significant and therefore, trusting the robustness of neural networks, no image was removed.

3. Modeling

To carry out the desired prediction task, several models were firstly built from scratch. Then, pre-trained models were also used. Throughout the entire project, the standard Graphical Processing Unit (GPU) of Google Colaboratory was used for training the CNN models.

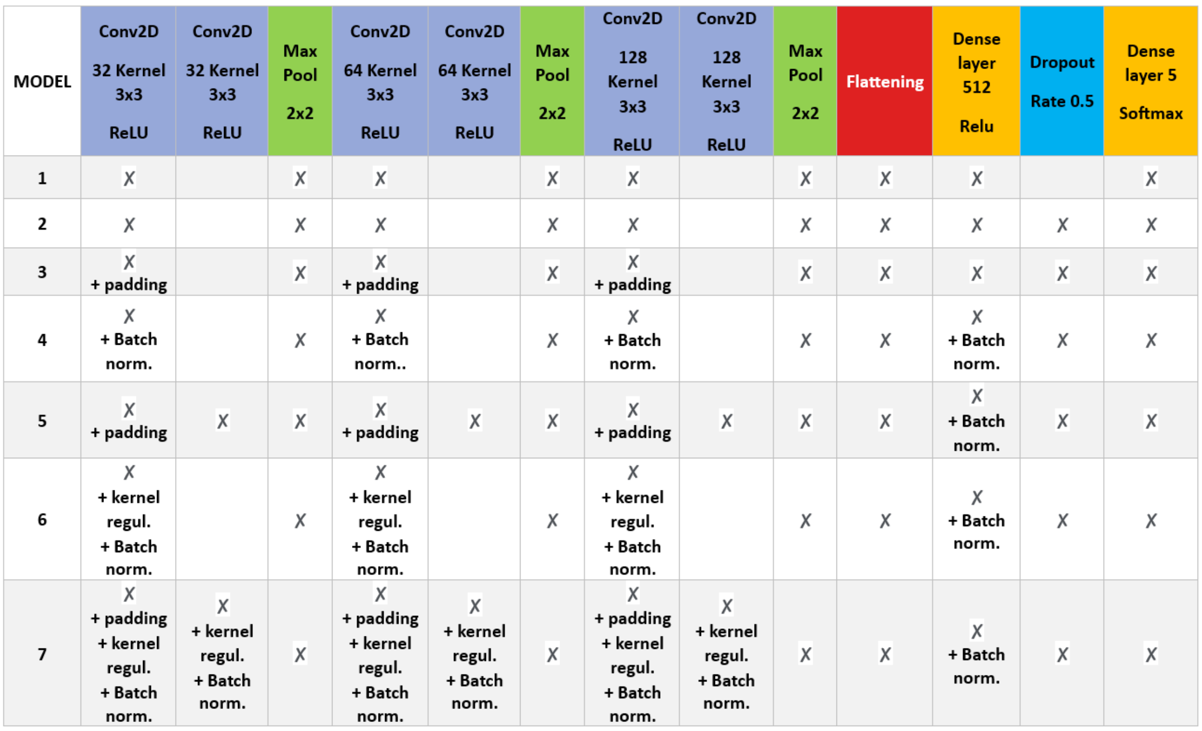
3.1 Normal data vs. Augmented data

The dataset used for training was slightly imbalanced, and not particularly big. Data augmentation was a possible solution to this problem. Data augmentation consists of taking an image and making copies of said image with random transformations applied to them, such as rotating them, resizing them, zooming in, and so on. This is done at every epoch during training: while the dataset doesn’t increase in size, its contents change at every epoch, which means that a lot more different scenarios are explored in training. Models trained with augmented data are therefore usually more robust.
When it comes to models built from scratch, for architectures 1, 2, and 3, models with data augmentation performed better than models without data augmentation, whilst for the remaining architectures the difference in performance was not significant. Since the models with the best performances among all models built from scratch were all trained with augmented data, all pre-trained models used augmented data.

3.2 Building Models from scratch

There were 14 models built from scratch in this project with 7 different architectures: each architecture was used for 2 models, one to be trained with regular data and the other one to be trained with augmented data.

In the following table we present a summary of the 7 different architectures:



MODEL	Conv2D 32 Kernel 3x3 ReLU	Conv2D 32 Kernel 3x3 ReLU	Max Pool 2x2	Conv2D 64 Kernel 3x3 ReLU	Conv2D 64 Kernel 3x3 ReLU	Max Pool 2x2	Conv2D 128 Kernel 3x3 ReLU	Conv2D 128 Kernel 3x3 ReLU	Max Pool 2x2	Flattening	Dense layer 512 Relu	Dropout Rate 0.5	Dense layer 5 Softmax
1	X		X	X		X	X		X	X	X		X
2	X		X	X		X	X		X	X	X	X	X
3	X + padding		X	X + padding		X	X + padding		X	X	X	X	X
4	X + Batch norm.		X	X + Batch norm..		X	X + Batch norm.		X	X	X + Batch norm.	X	X
5	X + padding	X	X	X + padding	X	X	X + padding	X	X	X	X + Batch norm.	X	X
6	X + kernel regul. + Batch norm.		X	X + kernel regul. + Batch norm.		X	X + kernel regul. + Batch norm.		X	X	X + Batch norm.	X	X
7	X + padding + kernel regul. + Batch norm.	X + kernel regul. + Batch norm.	X	X + padding + kernel regul. + Batch norm.	X + kernel regul. + Batch norm.	X	X + padding + kernel regul. + Batch norm.	X + kernel regul. + Batch norm.	X	X	X + Batch norm.	X	X

Figure 1.: Overview of the architectures of the seven basic models

The structure of the models was a sequence of 3 convolutional layers, each followed by a pooling layer. Then a flattening layer and 2 fully connected layers, the first to introduce nonlinearity and the second to create the outputs. Besides the first model, all the others also included a dropout in between the two dense layers, in which half of the neurons were dropped out at each iteration to make the model generalize better for unseen data. The following visualization shows the main structure used in most of the models.

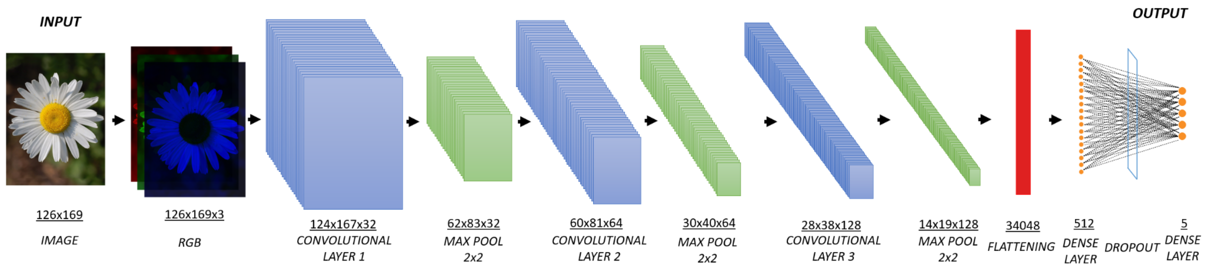


Figure 2.: Overview of the pipeline components of the seven basic models

Every architecture had three sets of convolutional and pooling layers, even though every architecture but 5 and 7 had one convolutional layer per set. With three convolutional layers it was possible to reliably detect high level features, and the max pooling layers after each one highlighted the more important features. In architectures 5 and 7 two convolutional layers in a row were used, and then max pooling, to try to detect more features from the output of a layer before taking the highest valued features from the output.

In architectures 3, 5 and 7, a technique called “padding” was used. Padding is used in convolutional layers, and it consists of adding a border of null pixels around an image. Usually, the pixels in the borders of images are touched less often in the convolutional phase, but with padding, the actual borders of the image are brought closer to the center,

therefore their pixels are more likely to have an impact in training, as they're touched more often. This can increase the quality of a model.
Architectures 4, 6 and 7 also added batch normalization after each pooling layer, in order to standardize the outputs. This made networks easier to train and increased their learning speed, whilst keeping the data's dimensional structure.
Another technique used was kernel regularization. This technique is used to reduce overfitting: L1 regularization removes some features from the model, and L2 regularization reduces the impact of insignificant features, resulting in a more generalized model.
At the end of every architecture there were two fully connected layers. The first one was used to produce non-linear combinations of the features obtained at that point, the second one simply compressed everything to produce five element long output vectors. In some architectures, there was a dropout layer after the first fully connected layer. This dropout layer randomly dropped output variables from the previous layer, to improve model performances and decrease chances of overfitting. Dropout was not used for convolutional layers because it would disturbed the structure of multi-dimensional outputs since it would randomly remove variables; Batch normalization did not have this issue since normalization keeps all the variables, and therefore doesn't lead to changes in the dimensional structure.

3.3 Pre-trained Models

In a second and more sophisticated attempt to tackle the flower classification problem, pre-trained CNN models were used. The idea of transfer learning is to reuse already existing models and build on top of them. This approach was beneficial for mainly two reasons: first, the core CNN architecture of the model was already defined and built; second, this core part has already been trained on a typically large dataset (e.g. ImageNet) for a relatively long period of time (i.e. number of epochs). Instead of initializing the parameters of a transfer learning model arbitrarily, it is possible to use the pre-trained weights and thus save time and resources. The model is then trained on the custom dataset corresponding to a given problem. This way, the weights of the model are adjusted specifically to perform well on the given task.
In the current project, the following 7 transfer learning models were used: VGG16, EfficientNetB7, ResNet152V2, Xception, MobileNetV3Small, ResNet50V2 and InceptionV3.
A short description of each model can be found below.

- VGG16: 3x3 filters, convolutional stride is fixed to 1 pixel, three fully-connected layers follow a stack of convolutional layers, the final layer is a soft-max layer and all hidden layers use the ReLu activation function; main downsides: slow to train and network architecture weights are quite large;
- EfficientNetB7: usually has a higher accuracy and efficiency over traditional CNN´s, it has 813 layers, all these layers are made from only 5 modules, and these blocks are all combined in sub blocks;
- ResNet152V2: it has 152 layers; uses batch normalization before each weight layer;
- MobileNetV3Small: lightweight in its architecture, performs depthwise separable convolutions (performs a single convolution on each color, instead of doing it in all 3 and then flattening); it performs with high speed;
- ResNet50V2 : different from the traditional networks, because it used a different architecture that relies on micro-architecture modules (network-in-network architectures); it has 50 weight layers;

- InceptionV3: acts as a multilevel feature extractor by computing 1x1, 3x3, and 5x5 convolutions within the same module of the network (the outputs are then stacked together before passing to the next layers);
- Xception: very similar to the Inception model but it replaces the inception modules with depthwise separable convolutions;

4. Model Selection

The best model was selected based on the validation set performance.
Theoretically, it is not guaranteed that the selected model was actually the best choice. The reason for this is that, when constructing the training and validation set, stochasticity was introduced on two occasions: first, when deciding which instances were used for training and which were for validation; second, when augmenting the instances. The model selection was thus based on (i.e. overfitted to) one specific transformation of one specific data partition. To ensure statistical significance when performing the model selection, each model would have to be trained and validated at least 30 times, so that the Central Limit Theorem would hold, using the same number of epochs and a different split for each. This process would require a lot more processing effort. For this reason, in the scope of the current project, the less time-consuming approach of selecting the best-performing model based on only one training-validation split was taken.
In order to speed up the training process, callback functions were used when training every model. Callbacks could be used to halt training when certain conditions were met in order to avoid spending any unnecessary amount of time. In this project, only the former was applied: whenever a model's validation F1-score dropped too much compared to the highest registered score in multiple epochs, training was stopped.
As mentioned earlier, the models were optimized for F1-score, so that was the basis of the model selection. However, to find the most robust solution, the confusion matrix of each model's prediction, as well as precision and recall scores for each class, and the F1-score for each class individually, were also taken into account. As the dataset was not heavily imbalanced, the accuracy score could arguably be used, along with the other metrics, to enhance the performance analysis. An additional informal technique that was applied was taking a look at the output of the models when misclassifying an image to analyze how certain a given model was about predictions that proved to be wrong.

The final choice was the ResNet152V2 model.

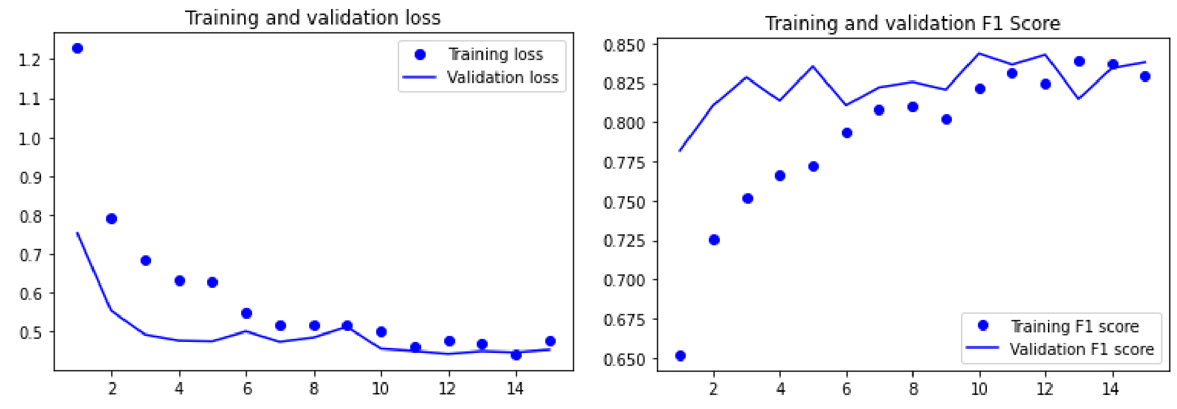


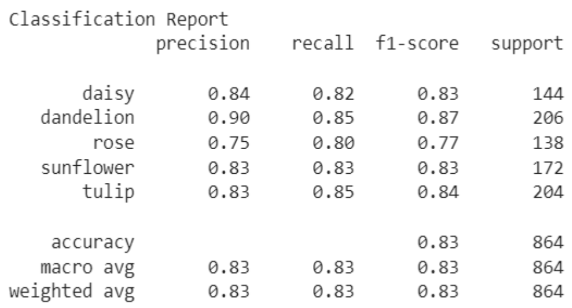
Figure 3.: Training and validation history of ResNet152V2

5. Hyperparameter tuning

After the most promising model was selected, grid search was performed to find the best hyperparameter settings for ResNet152V2 in an automated fashion. Three hyperparameters were tested together to find their optimal combination: the number of neurons in the first hidden dense layer, the dropout rate of the first hidden dense layer and the possible addition of a second hidden dense layer (with fixed number of neurons and dropout rate).
It was noted that the aforementioned grid search procedure was not extensive as the ResNet152V2 model has further tunable hyperparameters.
After concluding the grid search, the final model was re-trained using the optimal hyperparameter set. To make use of all the available data, this training was done on the merged training and validation set. However, this meant that it was not possible to apply early stopping. To avoid overfitting, the model was trained for a number of epochs that proved to be optimal during the grid search process.

6. Model Assessment

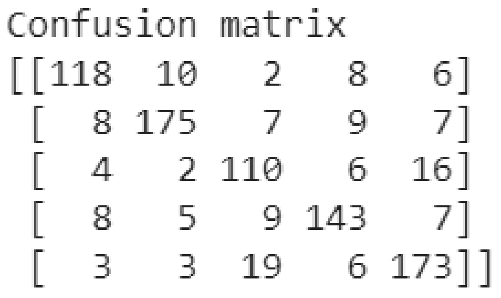
As a final motion of the project, once the hyperparameters and the weights of the selected model were both fine-tuned, the performance of the model was assessed on the unseen test set which held 20 percent of the total dataset. The model achieved an F-1 score of 0.8347. To gain more specific insights about the performance of the model, the classification report and the confusion matrix of the prediciton were once again assessed.



```
Classification Report
              precision    recall  f1-score   support

       daisy       0.84      0.82      0.83       144
   dandelion       0.90      0.85      0.87       206
        rose       0.75      0.80      0.77       138
   sunflower       0.83      0.83      0.83       172
       tulip       0.83      0.85      0.84       204

    accuracy                           0.83       864
   macro avg       0.83      0.83      0.83       864
weighted avg       0.83      0.83      0.83       864
```



```
Confusion matrix
[[118  10   2   8   6]
 [  8 175   7   9   7]
 [  4   2 110   6  16]
 [  8   5   9 143   7]
 [  3   3  19   6 173]]
```

Figure 4.: Classification report and Confusion matrix of ResNet152V2 on the test set

From this more detailed assessment, it can be seen that the performance of the model on predicting the classes of daisy, dandelion, sunflower and tulip are all high and fairly similar. The one class on which the ResNet152V2 model achieves a significantly lower F-1 score is the class of rose. More specifically, it can be derived that the reason for a lower F-1 score lies in having a lower precision, since F-1 score is a combination of precision and recall. The precision score shows the fraction of those roses that were correctly classified. It was also noted that a lower precision indicates a higher Type I. error. This means that the chance of misclassifying a rose is higher. To better understand this phenomena, the corresponding five element long output vectors were examined for the 28 misclassified roses. It was found that there are two distinct types of misclassification. In some of the cases, the model assigns similar probabilities to two classes which practically sum up to 1 (i.e. the model classifies the remaining three classes as highly unlikely). When this happens, the class of rose always had the second highest probability, oftentimes only slightly lower than the most probable class. In the second case of misclassification, the class of rose was assigned a rather low probability. Since the majority (20 times out of 28) of the rose misclassification instances are of case 1 (i.e. the second most probable class was rose), it can be argued that more rose training instances could effectively help the model correctly predict the labels for the aforementioned misclassified images and thus the precision and F-1 scores corresponding to the rose class could be further increased.

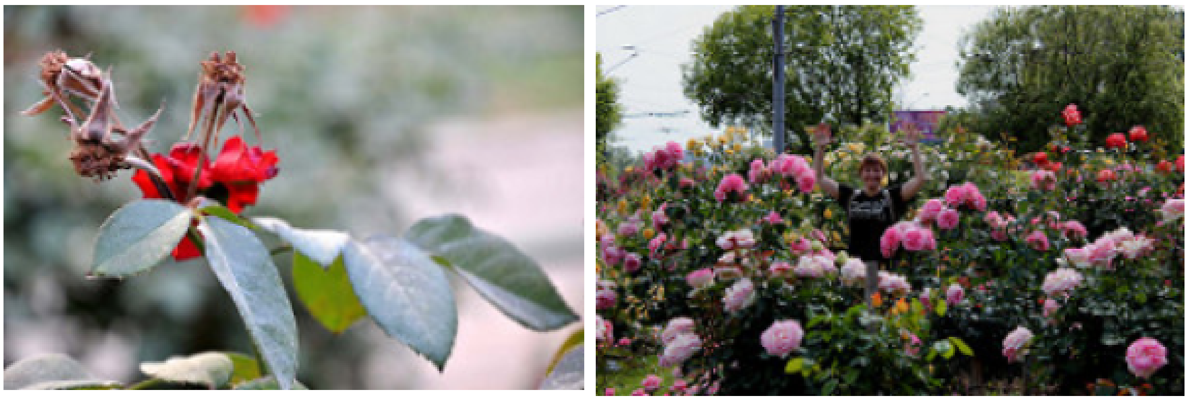


Figure 5.: Example of the first and second case of rose misclassification, respectively

7. Discussion and Future Work

The next step of this project would be to construct a test set manually and assess the performance of our model on that. The way to do this is to take pictures of the five types of flowers whenever we encounter them during our everyday lives. The collected images then need to be labeled manually. Once we collected a reasonable amount of images, we could perform predictions using our model and assess its performance to see if it was applicable for our initial purpose.

Other models already exist for this purpose, some basing predictions on flowers as well, and some of those were trained with the same dataset used in this project, and said models typically yield high results. However, out of those that also used flower data, none have been found that use accuracy as their main evaluation metric, meaning that their results are not comparable to the ones of the current project.

After further research on the topic, it was found that currently existing predictors work better based on the leaves of the plants instead of their flowers. Thus, in a second attempt at tackling the plant recognition problem, the objective would be to find a suitable dataset incorporating images of plant leaves and exploit the opportunities of training the already constructed models based on this new type of input.

8. Conclusions

The goal of this project was to develop a Convolutional Neural Network, to classify 5 different types of plants (daisy, dandelion, rose, sunflower and tulip).

There are multiple real world applications for this model, such as facilitating people to the recognition of flower species and allowing them to safeguard the species that are important for the biodiversity of the environment.

First of all, the dataset was imported from Github, divided it into 3 different datasets, training dataset, validation dataset, and test dataset and also one hot encoded the labels.

Callbacks were also set, to decrease the training time by using the early stopping method. This way, the training stopped if the F1-score decreased for 5 epochs in a row.

Data augmentation was used to create a more robust model that generalizes better to unseen data.

After all this, the modeling part was reached. 7 models without data augmentation and 7 models with data augmentation were built. The models without data augmentation and the

models with data augmentation had the same architecture, so this was a good way to check whether data augmentation improves the scores.
After finding that data augmentation increased performance, the 7 pre-trained models from keras were also trained, but only with data augmentation, given that it yields to the best performance.
Based on the performance of the validation set, it was found that the best model is ResNet152V2 with an F1-score of 0.8381.
Finally, a grid search was done to find the best parameters for the ResNet152V2 model, which was then re-trained with all data available to train (training dataset and validation dataset). That yielded a final F1-score of 0.822 on the test dataset.

9. References

Link to the dataset:

1) https://www.kaggle.com/code/rajmehra03/flower-recognition-cnn-keras/data
2) https://github.com/HitGobba2/deep-learning-

Other references:

3) https://www.baeldung.com/cs/batch-normalization-cnn - article on batch normalization
4) S Sachar and A Kumar 2021. Automatic Plant Identification Using Transfer Learning. *IOP Conf. Ser.: Mater. Sci. Eng.* 1022 012086
5) https://www.educative.io/edpresso/pre-trained-models-for-transfer-learning-in-keras
6) https://neurohive.io/en/popular-networks/vgg16/
7) https://keras.io/api/applications/
8) https://pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/
9) https://towardsdatascience.com/transfer-learning-using-mobilenet-and-keras-c75daf7ff299
10) https://ai.googleblog.com/2019/05/efficientnet-improving-accuracy-and.html
11) https://towardsdatascience.com/complete-architectural-details-of-all-efficientnet-models-5fd5b736142
12) https://www.mygreatlearning.com/blog/resnet/
13) https://medium.com/analytics-vidhya/implementation-of-regularization-techniques-l1-l2-in-keras-e0b39cb6a81d