Multilabel Classification of Orchid Features based on Deep Learning

C.A. Post (s2010771)
Enschede, Netherlands
June 26th 2020

***Abstract*—There are many different types of orchids, with many inter-class similarities. Classifying them using the conventional method of comparing the features to registered orchid types is very time consuming. Thus it would be useful to have an image based orchid feature classifier. In this paper a single output multilabel classifier using transfer learning is designed to classify six different orchid features. The design is based on experiments on different loss functions, pre-trained models, number of neurons in the dense layer, dropout rates and fine-tuning. The final model uses an Xception model with one untrainable layer as feature extractor. The classifier consists of two dense layers with a dropout of 0.5 in between. The final model gets a macro average f1-score of 0.85. The model is reliable for the non-color features, however it doesn't perform well on some rare classes of the color features.**

I. Introduction

The Orchidaceae are one of the two largest flowering plants species, with 736 different recognized genera in 2015 [1]. The Orchidaceae are not only popular as house plants but also have economic importance. For example, the orchids from the genus Vanilla fragrans are used to make vanilla extract, another type of orchid has edible roots and some types are even used for culinary medical purposes [2]. The classification of flowers has been done by looking at the habitation, the morphological structure and other features of the plant. Using the observations a comparison can be made with the registered types and the orchid can be classified. This is a time consuming task and a lot of expertise is needed. Thus it would be very useful to have an image based classifier. In recent years, research has been done to let the computer do classification based on an image. There are some difficulties within image recognition. Images of the same flower might look very different because of viewpoint variation, illumination conditions, scale variations, occlusion and background clutter. Above all, no flower is the same, so intra-class variation also needs to be taken into account. Deep learning has proven to be successful in identifying flower species [3]–[7]. Deep learning uses an artificial neural network, in which a convolutional network can learn to recognize features and the dense layers use these features to classify the image. There are however still some challenges within image based classification based on deep learning. The first problem is the data imbalance in the datasets. If there is a big difference between the minority and majority classes, the neural network will under-classify the minority classes. For example, suppose there are a hundred pictures of the colors blue and red. Divided in 99 red and one blue sample. Then, by always predicting red, the model easily obtains an accuracy of 99%.

Another challenge is that properly training a convolutional neural network (CNN) is very computationally demanding and needs a large dataset. One common solution to this problem is the use of transfer learning [8], in which a CNN is pre-trained on a large dataset and can be re-trained on another dataset. Another problem that occurs is over-fitting. When this happens, the model is not generalizing anymore, but learning each picture in the training set to achieve maximum accuracy. If the model is too complex, over-fitting occurs, however if the model is too simple, under-fitting occurs. Consequently the model needs to be neither too complex nor too simple. All these problems need to be taken into account while designing a classifier based on deep learning.
In most research so far the species of the flower have been classified for example on the 102 Oxford Flower Dataset [3]–[5], using a multiclass classifier. Also specific research on orchid species classification has been done [6], [7]. In contrast to these classifiers that have the species as output, the classifier described in this research predicts six features of orchids. These are also the features, which taxonomists use to classify orchids:

- The Color of the Flower
- The Color of the Labellum
- The Inflorescence
- The Texture of the Labellum
- The Labellum Characteristics
- The Number of Flowers

These features make up a total of 38 classes. The different combination of these features make up certain species. With the huge number of species within the orchid family, it would be useful to not have to train on all the separate species, but be able to classify them using the features. This way less samples per species are needed. All the above mentioned papers of flower classification use a multiclass classifier. There are multiple labels in this research, so a multilabel classifier is designed. This is a novel approach within flower classification. This is a feasibility study with the goal of obtaining a classifier that delivers the predictions of the orchid features, where each class gets at least an f1-score of 0.80. The f1-score will be explained in section IV-B. To achieve this goal, the following research question and subquestions will be used:
How well can a single output multilabel orchid feature classifier using transfer learning perform?

1) Can a cost sensitive loss function prevent under-classifying of the minority classes?
2) Which pre-trained model performs best in classifying orchid features?
3) Does changing the number of neurons in the dense layer and the dropout value improve the performance?
4) Is there an improvement in performance by changing the amount of trainable layers in the pre-trained model
5) Does using more data augmentation improve the performance?

The structure of this paper is as follows: In section II, related works is described. Section III describes the design of the model, followed by the experiments and results in section IV. This section also includes the analysis. Finally, the conclusion including future works is found in section V.

II. Related work

A. ImageNet

ImageNet is a large scale image database [9]. This large database makes it easier to make classification datasets to experiment on. ImageNet also held the Large Scale Visual Recognition challenge from 2010-2017 [10]. The goal of these challenges was to classify and detect from hundreds of object categories and millions of images. These challenges yielded some very good deep neural network architectures, for example the VGG16 [11], ResNet50V2 [12], Xception [13] and InceptionV3 model [14]. These models can be reused for feature extraction and classification purposes using a method called transfer learning [8]. In transfer learning, a model is trained on a large general dataset, like in the imagenet challenges, to recognize certain features. These models can then be re-trained to recognize related features on a smaller dataset.

B. Flower Identification

The use of neural networks has proven to be successful in flower classification tasks [3]–[7]. The Oxford Flower Datasets [15] have been used to build flower classifiers. Using this set, multiple solutions to make an accurate classifier were invented.
One common solution is the use of pre-trained models. Xia et al [3] successfully classifies flower species using transfer learning on the inception-v3 model. Wu et al [4] showed that the use of transfer learning increases the generalization ability and the robustness of the model. The pre-trained models implemented were the VGG16, VGG19, InceptionV3 and ResNet50. The use of these pre-trained models gave a significant improvement in the accuracy of the flower recognition.
Cibuk et al [5] uses two pre-trained models, AlexNet and VGG16, to extract features. These features are then concatenated and fed into a minimum redundancy maximum relevance method to select the features. Finally, a support vector machine (SVM) is used as classifier.

There are also some classifiers designed specifically for orchid species. Sani et al [6], designed a neural network with three hidden layers. This network is used to classify two similar looking orchid species. Zhang et al [7] constructed a large database with 2608 orchid species. The results show that using joint learning of a deep neural network and a tree classifier in an end to end fashion, achieves competitive results in large-scale plant identification.

C. Imbalanced Dataset

Imbalanced datasets are still a challenging topic in image based classification. Proposed methods to tackle this problem can be divided into data-level and algorithm-level techniques [16]. Data-level methods use pre-processing of the data, for example by up- or under sampling. In case of multilabel classification, each sample can have multiple labels. By upsampling an image with a specific label, the number of samples of other labels also changes as they belong to the same image.

Algorithm methods modify the algorithm for handling the class imbalance, for example by implementing class penalties or weights. Most of these solution are based on cost sensitive learning.
Wang et al [17] introduced the loss functions mean false error (MFE) and mean false square error (MSFE), which are based on the false positive error and the false negative error. By computing the errors on different classes separately, the loss becomes more sensitive to the loss of the minority classes. The MSFE is an improved version of the MFE. The MFE is not sensitive to the error of the positive class, whereas MSFE minimizes the errors on the positive class and the negative class at the same time.

Lin et al [18] introduced the Focal Loss. This loss function adds a factor to the standard cross entropy function. This factor makes it possible to put more focus on misclassified examples and less focus on the relative loss of the well classified examples.

III. Orchid Feature Classifier design

The model needs to be able to classify all six labels properly. There are two main options to design the model based on this:

- Making a multi-output multiclass model.
- Making a single output multilabel model.

Generally two methods are used within multilabel classification: The Binary Relevance approach and the Label Powerset method [19]. The latter makes each combination of labels a separate class, and thus transforms the multilabel model into a multiclass model. This results in having a lot of classes, with few samples each.
The Binary Relevance approach makes each class into a binary problem. This means the relationships between the labels won't be taken into account while making predictions. For orchids, changing one class usually means having a different species. Thus, it might be useful to not take the relationships between the labels into account. As the rare species might be predicted correctly if the model had enough samples of all the features separately.
The multi-output multi-class model would need a separate classifier for each label. The advantage is that of each label

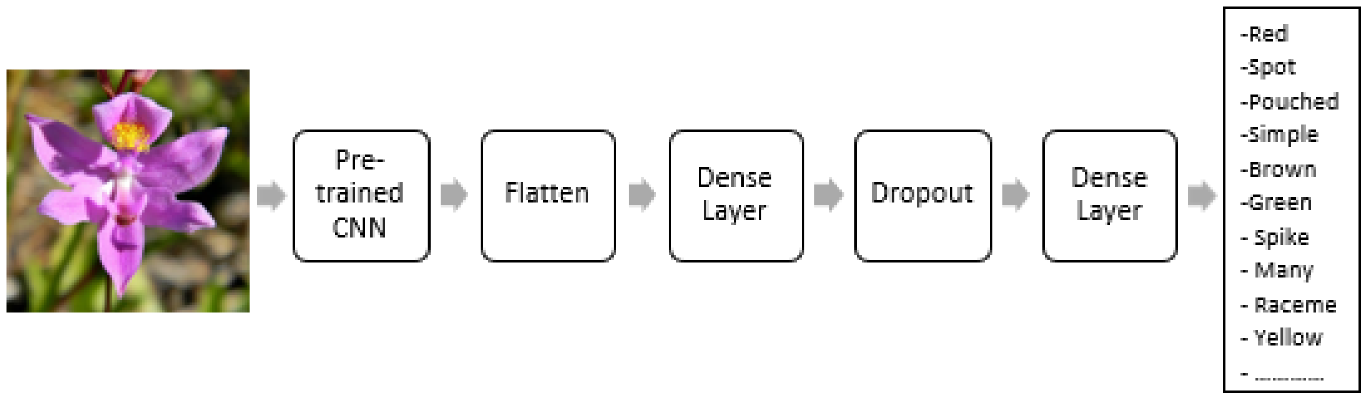

Fig. 1. Block diagram of the model

only one class will be predicted positive and each classifier is specialized in a certain feature. The multi-output multi-class model and the single output multilabel model both seem promising. In this research the single output multilabel is used.
The design of this model is based on putting together existing techniques. The proposed model consists of two parts: A convolution neural network (CNN) used as a feature extractor and a classifier. Two of the output features are colors, thus the input needs to be an RGB image. A schematic of the model can be seen in figure 1.

The first part of the model is the CNN. A CNN generally consists of multiple convolutional and pooling layers. In the convolutional layer a filter is applied over the input image. The output of this layer is linear, thus at the end of the convolutional layer an activation function is used to introduce non-linearity. Common activation functions are Tanh, ReLu and Sigmoid. By applying multiple filters, multiple features can be extracted. The first convolutional layers extract the low level features like edges. By having more convolutional layers, middle and high level features can also be extracted.
In the CNN, a pooling layer is commonly used after the convolutional layer. The role of the pooling layers is to reduce dimensionality of the output of the convolution. This is done to decrease the computational power needed to process the data while maintaining spatial invariance.
As this research needs multiple features as output, a deep CNN is deemed most suitable. However, a deep CNN needs a lot of computational power and data to properly train the model. Both of which are not available for this research. Consequently, transfer learning will be used. Using transfer learning a pre-trained model can be retrained. There are multiple pre-trained models available trained on the ImageNet data set. The pre-trained CNN can be fine-tuned to recognize orchid features by making layers trainable. An experiment will be performed on four different winners from the ImageNet challenge to decide on a pre-trained model for the final model.
After the CNN, fully connected layers are applied. In this fully connected layer, also known as dense layer, all the neurons are connected to all neurons of the previous layer. The function of the dense layer is to classify the sample based on the features extracted by the CNN. The proposed classifier consists of two fully connected layers. First a flatten layer is applied to get the right dimensions, followed by a dense layer with a certain number of neurons. In between the dense layers a dropout is applied to prevent over-fitting. The last dense layer has 38 neurons, one per class. The sigmoid activation function is used in the final layer. The sigmoid function is chosen as the classes aren't mutually exclusive. The number of neurons in the first dense layer and the dropout value might influence the performance and will be decided on the basis of an experiment.

The model can be trained using different parameter settings. The training process is based on changing the weights of our model in order to try to minimize the amount of wrongly predicted labels. The loss function is used to tell the model how wrong the predictions are. To decide on a loss function for this model, the best performing one will be decided on in experiment 1. Based on the loss, the optimizer tells the model how to change the weights for example by adding or subtracting a certain value to the weights. The learning rate is used to decide how much these weights are changed. By adding data-augmentation each epoch each image is slightly changed in for example the brightness or scale. This way the model sees a slightly different image each time, which can prevent over-fitting. The effect of using more data augmentation will be experimented on in section IV-F.

IV. Experiments and Results

The goal of the experiments is to find the optimal parameter settings for the model. To build the model, the open source machine learning system called Tensorflow [20] is used as platform, with high level API Keras [21] running on top of it. Keras is a deep learning API written in python. The first experiment is performed to decide on an appropriate loss function. The second is done to find the most promising working pre-trained model. In the third experiment, the number of neurons of the first dense layer will be changed and different dropout values will be tested on. The next experiment is

performed to fine-tune the model by changing the amount of data augmentation, trainable layers and to see whether changing the test set improves the performance. The final experiment will test the reliability of the model.
After each experiment the most promising parameter is chosen to continue on with to the next experiments. This decision is based on the precision, recall and f1 scores, which will be explained in subsection B. A more detailed explanation of the experiments is given in the subsections.

The model as shown in figure 1 is used. The base model used to start the first experiment with uses the VGG16 as pre-trained CNN, with all layers untrainable. Followed by a flatten layer. The first dense layer contains 512 neurons with ReLu as activation function. The dropout is 0.5. The last dense layer has 38 neurons, with sigmoid as activation function. Adam is used as the optimizer and the binary cross entropy as loss function. The input image has size 224x224x3. The batch size is 64 and the training data is shuffled. The model is trained for 60 epochs. All results are derived from the independent test set using a threshold of 0.5.

The structure of this section is as follows: Subsection A describes the data set used in the experiments. The second subsection explains the evaluation methods used. Finally, all the experiments will be elaborated on in their own subsections. In the subsection of the experiment, first the experiment set-up is explained, secondly the results are shown and lastly the analysis is done.

A. Dataset

The Orchid Flower Dataset [22] is used to train and test the model. This dataset consists of 7156 images of 156 different orchid species. Some samples of the dataset can be seen in figure 2. Every image has seven different labels: The Texture of the Labellum, the Labellum Characteristics, Inflorescence, Number of Flowers, Color of Flower, Color of the Labellum and the Species. In this research only the features will be used. In total there are 38 classes in these six features.
The data set is imbalanced, which means that there are some underrepresented classes. The smallest class belongs to the ”color of flower” label, namely the class Brown with 15 images. The largest color of flower class consists of 1670 samples, which has over a hundred times more images than brown. There are also some noisy images. For example a random picture which shows no similarities to the flower it is labeled to be, or an image of a spider, as can be seen in the lower middle picture in fig. 2. There could also be some wrongly labeled images. The images of this data set are varied and show the orchids from different viewpoints, in different background settings, with different brightness and for example with insects on the flower. Some images show only the buds of the orchids, whereas others show the flower in full bloom. This also means that some images are labelled with having a certain color, while there is no flower to be seen in the picture. The bud is the only part shown and this might have a different color than the flower. There are also drawings of the species included in this dataset.

TABLE I
CONFUSION MATRIX

True/Predicted	predicted negative	predicted positive
Negative	TN	FP
Positive	FN	TP

B. Evaluation Method

The f1-score, along with the precision, recall and the confusion matrix are used for evaluating the performance of the model. Table I presents the confusion matrix, which is the multilabel confusion matrix of scikit learn [23]. The confusion matrix for binary classification consists of four variables:

1) True Positives (TP) :The amount of labels predicted positive while in reality they are positive.
2) False Positive (FP) :The amount of labels predicted positive while in reality they are negative.
3) True Negatives (TN) :The amount of labels predicted negative while in reality they are negative.
4) False Negatives (FN) :The amount of labels predicted negative while in reality they are positive.

The other evaluation methods are based on these four variables, which each show a different relationship between them [24]. The precision is the ratio between the correctly predicted positive labels versus the total of predicted positive labels, as defined in equation (1). When the precision is 1.00 all labels predicted positive are actually positive.

$$Precision = \frac{TP}{TP + FP} \tag{1}$$

The recall is the ratio between the correctly predicted positive labels versus the total of positive labels, as defined in equation (2) When the recall is 1.00, no labels are predicted negative while they should have been positive.

$$Recall = \frac{TP}{TP + FN} \tag{2}$$

The f1-score is the harmonic mean between the precision and recall, as defined in equation (3).

$$f1 - score = 2\frac{Precision * Recall}{Precision + Recall} = \frac{2TP}{2TP + FP + FN} \tag{3}$$

For multilabel classification the average score over the labels can be calculated in different ways. The two ways used in this paper are [23]:

- The ”macro” average calculates the scores for each label and their unweighted mean.
- The ”micro” average calculates the scores by counting the total amount of True negatives, False negatives, True positives and False positives.

C. Experiment 1: Loss function

The aim of this experiment is to tackle the data imbalance problem using different cost sensitive loss functions: The Focal Loss, the Mean Squared Error (MSE) and the Binary Cross Entropy (CE). For the Focal loss alpha is 0.25 and gamma is 2. The MSE and Binary CE loss functions are given weights

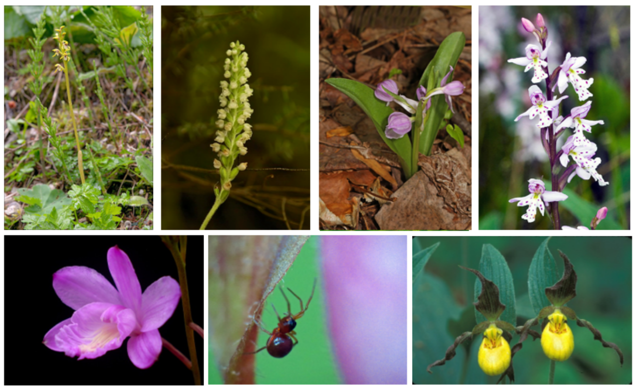

Fig. 2. Samples of the dataset

calculated by the compute_class_weight function of scikit-learn [25]. These weighted and non-weighted loss functions are compared. The number of labels never predicted positive is added to see whether the model over-fits on the majority classes. The results can be seen in table II

TABLE II
DIFFERENT LOSS FUNTIONS

Loss function	**f1 micro**	**f1 macro**	**Number of labels never predicted positive**
Binary CE	0.66	0.41	10
MSE	0.68	0.45	8
Focal Loss	0.65	0.42	7
Weighted binary CE	0.70	0.56	2
Weighted MSE	0.69	0.57	2

As expected, the focal loss performs better than the non-weighted loss functions as it is designed specifically for data imbalance. However only one value for gamma and alpha is used in this experiment and might not be the optimal values. It can be seen that the weighted loss functions have more than three times as little classes that are always predicted negative. This means that giving weights to the loss function improves the number of positive predictions of the minority classes. However it should be noted that in the test set this time, there are some classes with very few samples. For example the class brown, which only has 2 samples. There is also another class with three samples and two other classes with four samples. Thus the validity can be argued. However looking at these results, it can be seen that the weighted binary CE and MSE perform similarly. Consequently the weighted binary CE was chosen arbitrarily to continue with.

D. Experiment 2: Pre-trained models

As aforementioned, it was decided that the test set did not contain enough samples of certain classes to reliably give an estimate of the performance. Thus the images of all classes in the test set were increased to at least 10 images per class, except for "Brown", which ended with seven images. Otherwise the training amount would be insufficient. These samples were taken from the training and validation set, which resulted in a decrease of training samples, for example brown only has six samples left. Consequently it was decided that from now onward data augmentation will be used to rectify this decrease of rare classes.
The aim of this experiment is to see whether a certain pre-trained model performs better than the others. For the training only the first layer was made untrainable.
The pre-trained models were imported from Keras applications using the ImageNet weights. The chosen pre-trained models are VGG16, Xception, InceptionV3 and ResNet50V2. Since they all showed great capability in both the ImageNet Challenges and in other flower classification papers, as mentioned in section II. The models were trained with a learning rate of 0.00005. The results can be seen in table III.

TABLE III
DIFFERENT PRE-TRAINED MODELS

pre-trained model	**Precision macro**	**Recall macro**	**F1 macro**	**lowest precision**	**lowest recall**
VGG16	0.76	0.76	0.75	0.50, 0.53	0.20, 0.20
InceptionV3	0.80	0.74	0.75	0.00, 0.46	0.00, 0.31
ResNet50V2	0.82	0.71	0.73	0.57, 0.57	0.10, 0.14
Xception	0.86	0.77	0.79	0.59, 0.63	0.10, 0.30

Notable is that the first layer of the VGG16 model does not contain any trainable parameters. To be able to compare it better with the other models, the model was retrained with three layers trainable. These had a similar number of untrainable parameters as the other models had with one untrainable layer. This did improve the performance. However the results were still worse than the Xception model. Interestingly enough this shows that zero untrainable parameters performed worse than a few untrainable parameters. In the fine tune section it will be tested whether making more than one layer untrainable also improves the final model. In the results it can be seen that the Xception model outperforms all the other pre-trained models. Thus the Xception model is deemed most promising to continue with. The Xception model is the newest out of all four models and according to the paper [13], it should outperform the InceptionV3 model. In this research the performance is measured specifically on flower feature recognition so the results on the imagenet dataset could not be representative.

E. Experiment 3: Dense Layers & dropout rate

As a result of the previous experiment, the pre-trained Xception model with one layer untrainable is used for this experiment. All other configurations stay the same. Now the model is first tested with a different number of neurons in the first dense layer: 256, 512, 1024 neurons. The results can be seen in table IV. Next the most promising number of neurons is chosen and three different dropout rates are tested: 0.3, 0.5 and 0.7. The results are shown in table V.

TABLE IV
DIFFERENT AMOUNT OF NEURONS IN THE DENSE LAYER

Dense Layers	**Precision macro**	**Recall macro**	**f1 macro**	**f1 micro**	**Lowest precision**	**Lowest Recall**
256	0.87	0.77	0.80	0.87	0.60,0.68	0.20,0.30
512	0.86	0.77	0.79	0.86	0.59,0.63	0.10,0.20
1024	0.87	0.77	0.80	0.86	0.60,0.70	0.20,0.30

TABLE V
DIFFERENT DROPOUT VALUES

Dropout	**Precision macro**	**Recall macro**	**f1 macro**	**f1 micro**	**Lowest precision**	**Lowest Recall**
0.3	0.82	0.76	0.78	0.86	0.00,0.60	0.00,0.30
0.5	0.87	0.77	0.80	0.87	0.60,0.68	0.20,0.30
0.7	0.85	0.76	0.78	0.86	0.47,0.67	0.10,0.20

The results show an insignificant difference between the different number of neurons. It was expected that a large number of neurons might lead to quick over-fitting or a smaller number could lead to under-fitting. It is also noteworthy that the validation scores of the 256 and 1024 neurons models reach convergence almost at the same epoch. As a result 256 neurons were chosen to continue with, as it decreases the number of trainable parameters and thus decreases the training time.
As can be seen in table V, there is an insignificant difference between the different dropout values. Notable is that the lowest value in recall and precision is 0.00 for the dropout of 0.3. This means that one class is never predicted positive correctly. This could be caused by the fact that a dropout of 0.3 isn't enough to prevent generalization errors. The regularization parameter is maximum when the dropout is 0.5. Using that knowledge and seeing as the difference between the dropout 0.5 and 0.7 is insignificant. It was chosen to continue with a dropout of 0.5.

F. Fine-tuning

These experiments are performed in order to fine-tune the model. First, extra data augmentation is used. Second, a different number of untrainable layers is experimented on. Lastly, the test set is analysed in order to see what goes wrong with the lowest scoring classes. Then, the samples of the test set are interchanged with samples of the training and validation set and the model is trained again on these new datasets.

1) Extra Data Augmentation: This experiment is performed in order to improve the performance of the model even further. In this experiment the amount of data augmentation is increased. The results can be seen in table VI.

TABLE VI
EXTRA DATA AUGMENTATION

Data augm.	**Precision macro**	**Recall macro**	**f1 macro**	**f1 micro**	**Lowest precision**	**Lowest Recall**
Usual	0.87	0.77	0.80	0.87	0.60,0.68	0.20,0.30
Extra	0.85	0.80	0.81	0.87	0.57,0.66	0.30,0.38

Table VI shows that there is very little difference on the overall performance between the original amount of data augmentation and the extra data augmentation. This could be because the original amount of data augmentation was already enough to change the pictures and the extra augmentation does not make that much of a difference anymore. What can be noticed is that there is less difference between the recall and precision. This means that there is a better balance between the false negatives and false positives. High precision means that most of the predicted positive labels are actually positive and a high recall tells that the number of correctly positive predicted samples is close to the number of true positives. Both of which is needed. Although there is only a slight difference in performance, it was decided to continue with the extra data augmentation.

2) Untrainable layers: In this experiment more layers are made untrainable. This improved the performance of the VGG16 model as mentioned in experiment 2. By making more layers untrainable, it might make the model less prone to over-fitting. The number of trainable parameters was made two and five times as big, which means that it will be tested with 1, 13 and 25 untrainable layers. The extra data augmentation is used too. The results can be seen in table VII.

TABLE VII
DIFFERENT AMOUNT OF TRAINABLE LAYERS

Layers Untrain-able	**Precision macro**	**Recall Macro**	**f1 macro**	**f1 micro**	**Lowest precision**	**Lowest Recall**
1	0.85	0.80	0.81	0.87	0.57,0.66	0.30,0.38
13	0.85	0.78	0.79	0.85	0.55,0.60	0.20,0.30
25	0.84	0.78	0.79	0.85	0.57,0.67	0.30,0.30

Looking at the performance of the different number of untrainable layers, it can be seen that there is a slight decrease in performance. It should be noted however that the number of trainable parameters is 0.26%, 0.52% and 1.17% of the total number of trainable parameters of the Xception model respectively. The difference in percentage of untrainable layers in the whole model is very small. Furthermore the performance between 13 and 25 untrainable layers is very similar, thus it cannot be said that the reason for the decrease is having more layers untrainable. In future works it might be worth it to test whether making half or almost all layers untrainable will make a significant diffence, as it would decrease the training time significantly. For now, the one layer untrainable performed best. Consequently, this was chosen to be used in the final model.

3) Low scoring classes: The final model is decided on: An Xception model is used as pre-trained CNN, with one

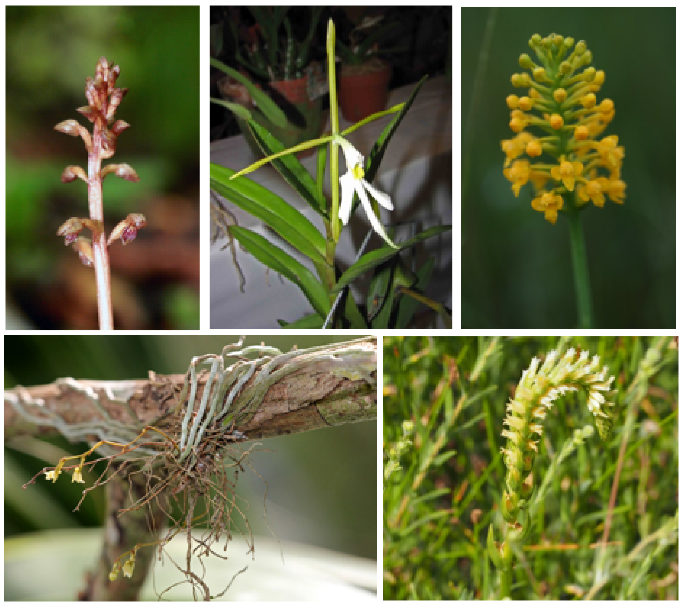
Fig. 3. Samples of the different groups belonging to the class Yellow

layer untrainable. Followed by a flatten layer. The first dense layer has 256 neurons and the dropout is 0.5. Extra data augmentation is used and all the other parameters are as described in the beginning of this section. While looking at the results, it is clear that the model has most trouble with predicting certain colors. To see whether the cause is the model or the test-set, the following experiment is performed.
For this experiment the classes of the "color of flower" label with an f1-score of less than 0.70 are used. First, the images of the training and test set of these classes are compared. The first thing noticed is the way the data sets are divided: The first 70% is of the training set, the next 20% of the validation set and the last 10% of the test set. So suppose in a certain class there are 100 photos named 1.jpeg until 100.jpeg, then up to 70.jpeg is part of the training set, 70-90.jpeg of the validation and 90-100.jpeg is part of the test set. The data set consists of sequences of images of the same flower from different viewpoints, brightness and either more zoomed in or out. This could result in training on a sequence of images of one flower, and then testing on a completely different flower. So the hypothesis is that the way the images are divided over the test, validation and training set, makes the performance go down. To see whether this hypothesis is right, the following is done: First, all images of a certain color class were gathered. Next the similar looking images were grouped. This grouping was done by looking at what the flowers looked like, in which setting they are and the viewpoint of the pictures. The similarities are also based on the species. An example of some of the different groups belonging to the color yellow can be seen in figure 3.

Next, the samples of each group were split between the training, validation and test set. Using this new data set, the model was trained again. Next, half of the samples in this new test set was swapped with samples from the new training set. The model was trained again using this new data set. As all the lowest predicted classes were minority classes, the model was trained again on not only the training augmented with the validation set. The results can be seen in table VIII.

TABLE VIII
THE SCORES OF THE LOWEST PREDICTED COLOR OF FLOWER CLASSES BEFORE AND AFTER SWITCHING SAMPLES

Color	**f1 original**	**f1 switch 1**	**f1 switch 2**	**f1 switch val1**	**f1 switch val2**
YellowPurple	0.63	0.88	0.79	0.85	0.88
Yellow	0.67	0.85	0.76	0.88	0.82
PinkWhite	0.53	0.76	0.67	0.82	0.74
GreenYellow	0.63	0.83	0.82	0.89	0.92
GreenWhite	0.53	0.62	0.82	0.76	0.76

It can be seen that the scores significantly improve by switching samples per group. Which implies that the model performs well, as long as the test and training set have similar looking pictures. The first switch was done by dividing the similar looking pictures over the training, validation and test set. However in the second switch half of the samples were arbitrarily swapped. As a consequence, the lower performance of the second set could be explained by the fact that there are less similar looking images in the test and training set. Furthermore it should be noted that although the samples are labeled a certain color, it doesn't mean that just looking at the color always works. There are for example photos of just the buds, which don't always have the color of the flower. Some photos are located in a forest, in which some are zoomed in on the flower and some plants only make up a very small part of the image. In the latter the background noise could play a big role in predicting the images wrongly. Furthermore the significant difference in performance by swapping some samples could be caused by the few amount of samples in these classes. After the grouping was done, there were multiple groups with less than 6 samples. So all these six samples could end up being in the test and validation set. As can be seen in fig. 3, even within a group with very few test samples, there is a variety of different looking flowers, which are also shot at different viewpoints and angles. This means that there are very few similar samples to train and test on. There is also still a difference in performance between the classes, for example GreenYellow got good scores twice after the switching, while GreenWhite got two very different scores. This can be explained by looking at the groups. GreenYellow has a high number of samples per group, with only two groups containing less than 10 samples. In comparison GreenWhite its biggest group contains 18 samples and all other groups less than 7 samples. When the model is trained on both the training and validation set, the performance improves as expected. This again shows that the low performance is caused by too few samples to train on.

G. Reliability

Using the acquired knowledge that swapping images of the training set and test set improves the performance, the final experiment will check the reliability of the model. This experiment is performed to see the influence of randomly switching half of the test set with samples of the training set. The switching is done per species, to make sure all classes keep the same amount of test samples. The first set was made by changing all the even numbered images from the test set with even numbered images from the training set. If there was only one sample of the species in the test set, it would be changed with a even numbered sample of the training set regardless the test sample number being even or uneven.
The second set was made by switching the uneven numbered samples of the test set with uneven numbered samples of the training set. If there was only one sample of the species in the training set, then this time this species would be swapped with a sample from the validation set.
The third set was randomly switched. In table IX, the average scores of each class are shown with the deviation: the difference between the average and the lowest/highest score.

The table shows that the overall performance only changes slightly, with a maximum deviation of 0.01. Now let's look at each label separately.

1) Texture of the labellum: The "No Spot" class is the majority class with 676 test samples, in contrast to the 171 test samples in the "Spot" class. The model predicts "No Spot" very well and reliably. The "Spot" class also gets predicted well, however significantly less than the "No Spot". The deviation is also higher, especially for the recall. The lower recall makes sense as it is a minority class. This would imply that the model predicts "Spot" as positive less often than it should.

2) Labellum Characteristics: In this label, all the classes are predicted with an f1-score of at least 0.83. The scores of "Lobed", "Pouched" and "Simple" show very little deviation, whereas "Fringed" has considerably more deviation. Again the class with the most deviation in the scores is the minority class of the label. "Fringed" only has 71 samples whereas the second smallest class has 176. This also implies that the model is still not fully capable of handling imbalanced data.

TABLE IX
THE AVERAGE SCORES PER CLASS WITH DEVIATION

Label	**Class**	**Precision**	**Recall**	**f1-score**
Texture of the Labellum	Spot	0.88 ± 0.03	0.84 ± 0.06	0.86 ± 0.03
	No spot	0.96 ± 0.01	0.97 ± 0.01	0.96 ± 0.01
Labellum characteristics	Fringed	0.88 ± 0.06	0.85 ± 0.07	0.87 ± 0.04
	Lobed	0.90 ± 0.01	0.91 ± 0.01	0.91 ± 0.01
	Pouched	0.92 ± 0.02	0.93 ± 0.02	0.92 ± 0.02
	Simple	0.87 ± 0.03	0.88 ± 0.01	0.88 ± 0.01
Inflorescence	Panicle	0.90 ± 0.02	0.79 ± 0.04	0.84 ± 0.02
	Raceme	0.92 ± 0.01	0.95 ± 0.02	0.93 ± 0.02
	SingleOrPair	0.96 ± 0.02	0.94 ± 0.01	0.95 ± 0.01
	Spike	0.85 ± 0.02	0.86 ± 0.02	0.86 ± 0.02
Number of Flowers	A few	0.80 ± 0.02	0.84 ± 0.03	0.83 ± 0.02
	Many	0.88 ± 0.01	0.89 ± 0.01	0.88 ± 0.02
	SinglePair	0.96 ± 0.02	0.94 ± 0.01	0.95 ± 0.01
Color of Flower	Brown	0.94 ± 0.11	0.43 ± 0.28	0.55 ± 0.12
	Green	0.80 ± 0.05	0.92 ± 0.01	0.86 ± 0.03
	GreenBrown	0.88 ± 0.03	0.92 ± 0.02	0.90 ± 0.02
	GreenWhite	0.97 ± 0.05	0.74 ± 0.18	0.83 ± 0.09
	GreenYellow	0.84 ± 0.16	0.84 ± 0.07	0.83 ± 0.07
	Orange	0.93 ± 0.07	0.93 ± 0.07	0.93 ± 0.07
	Pink	0.90 ± 0.01	0.97 ± 0.02	0.93 ± 0.01
	PinkWhite	0.73 ± 0.07	0.53 ± 0.26	0.60 ± 0.16
	Purple	0.85 ± 0.03	0.75 ± 0.05	0.80 ± 0.03
	PurpleWhite	1.00 ± 0.00	0.96 ± 0.06	0.98 ± 0.03
	Red	0.84 ± 0.04	0.79 ± 0.12	0.81 ± 0.08
	White	0.87 ± 0.04	0.93 ± 0.03	0.89 ± 0.03
	Yellow	0.82 ± 0.18	0.63 ± 0.05	0.71 ± 0.04
	YellowBrown	0.83 ± 0.01	0.88 ± 0.04	0.85 ± 0.03
	YellowPurple	0.80 ± 0.04	0.87 ± 0.04	0.83 ± 0.03
Color of the labellum	BluePurple	0.79 ± 0.06	0.79 ± 0.07	0.79 ± 0.05
	GreenCL	0.74 ± 0.04	0.89 ± 0.03	0.80 ± 0.02
	OrangeCL	0.92 ± 0.08	0.93 ± 0.07	0.93 ± 0.07
	PinkRed	0.88 ± 0.02	0.90 ± 0.03	0.89 ± 0.02
	PinkRedWhite	0.91 ± 0.04	0.84 ± 0.06	0.87 ± 0.05
	PurpleYellow	0.98 ± 0.03	0.97 ± 0.03	0.97 ± 0.03
	WhiteCL	0.88 ± 0.04	0.91 ± 0.02	0.89 ± 0.02
	WhiteYellow	0.94 ± 0.11	0.45 ± 0.05	0.61 ± 0.06
	YellowCL	0.79 ± 0.05	0.81 ± 0.05	0.80 ± 0.01
	YellowCLBrown	0.91 ± 0.03	0.94 ± 0.06	0.89 ± 0.07
Total Average	Macro avg	0.88 ± 0.01	0.85 ± 0.01	0.85 ± 0.01

3) Inflorescence: All inflorescence labels achieve an f1-score of at least 0.82. "Panicle" is the minority class of this label, with 27 samples versus the second smallest class "Spike" with 166 samples. The highest deviation is again in the minority class, however it is still only 0.04. "Panicle" is also the class with the lowest scores, similar to the minority classes of the previous two labels.

4) Number Of Flowers: The Number of Flower classes are predicted well consistently with a maximum deviation of 0.03. Interestingly enough, this time the minority class has the best performance. This could be because the "SinglePair" feature is easier to recognize for the model. By looking at the evaluation on the training set, it can be seen that the "SinglePair" class has scores of 1.00, whereas "A Few" has a precision of 0.97 and "Many" has scores of 0.99. Thus, these other two classes probably contain some difficult to identify samples. These samples could be noisy samples. The other reason could be that the difference in samples between the classes is relatively small compared to the difference in the other models.

5) Color of Flower: Although the majority of the scores is above the 0.80, this label contains the worst performing class and also the biggest deviations. This is also the label with the largest number of classes, and the biggest difference in the number of samples per class. It can be observed that the majority classes perform well and have a relative small deviation compared to the rare classes. All the low scoring classes belong to the minority classes. However, there are also minority classes that perform well with relatively low deviation, for example ”PurpleWhite” and ”YellowPurple”. This could be because these features don't look similar to other features for the model and are thus easy to classify. It could also be because the images within these classes look more alike than the images of the other minority classes, which makes switching the images have less influence on the performance. The big deviations are caused by the low number of samples. If the number of samples is very low, the difference that one wrong predicted sample makes, is significantly bigger than in a class with a high number of samples.

6) Color of the Labellum: This label has a relatively big number of classes, similar to the color of flower label. The lowest scoring classes are again the minority classes. ”WhiteYellow” is the smallest class with 10 samples, followed by ”PurpleYellow” with 20 samples. The very low recall of ”WhiteYellow” can be explained by the number of training samples. WhiteYellow only has 11 training samples, whereas the second smallest number of training samples is 78 for ”PurpleYellow”.
Looking back at all the separate labels, it shows that the model is not able to reliably predict some minority classes. This can be caused by the lack of ability of the model to handle data imbalance, and the lack of samples in the data set.

V. CONCLUSION

In this paper the difficulties of image based classification and the importance of flower classification were discussed. The aim of this study was to see how well a classifier based on deep learning could predict the six different orchid features. The goal was to reach a f1-score of at least 0.80 per class. The final model was designed to see how well a single output multilabel classifier can perform. This design was based on experiments on different loss functions, pre-trained models with different number of trainable layers, number of neurons in the first dense layer and the dropout value inbetween the dense layers.

The final model consists of a pre-trained Xception model, with 1 layer untrainable. Followed by a flatten layer, a dense layer with 256 neurons, a dropout of 0.5 and a final dense layer with 38 neurons using the sigmoid activation function. The loss function used to train the model is a weighted binary CE.
This final model predicts all classes of the non-colored labels reliably with a f1-score of at least 0.81. The predictions of the color labels are less reliable and contain classes with f1-scores less than 0.80. The weighted loss function did improve the performance on the minority classes, however there is still a trend in the results where the lowest scores are from the rare classes. Thus the model is still not able to fully handle the data imbalance. For the feature extraction, the Xception model performed better than the VGG16, ResNet50 and the InceptionV3 model. Changing the number of neurons and the dropout value showed an insignificant difference in f1-scores. Also changing the number of trainable layers did not improve the model. More data augmentation did not have a significant effect on the overall performance. The division of the samples over the test, training and validation set showed to have a significant influence on the performance. The model its overall performance is well, however there are still some classes with fairly low and unreliable scores. These classes all have few test and training samples. The cause of these low scores is probably the lack of training and test samples in combination with the model not being able to completely tackle data imbalance. This means that the model might predict all these features well, if more samples of these rare classes were used. At least the results would be more reliable with more samples and the performance can be properly evaluated.

A. Future works

The current model is not good enough in the prediction of some of the rare classes. This could be solved by augmenting the data set with more samples of the minority classes. This way the test scores will be more reliable and the model can properly train on a range of different looking images in that class. Furthermore, the weights within the loss function could be optimized, which would probably imply giving the minorty classes more importance than the current model does. According to Wang et al [17], the MFE and MSFE perform better than the mse loss function. The weighted mse and binary cross entropy performed similarly. Thus by implementing MSFE the performance on the minority classes could improve.
T-SNE could also be used to visualize the dataset, such that it can be seen which classes look similar and are more easily be confused by the model [26]. Also the noisy samples should be removed from the dataset. To improve the efficiency of the training, it could be tested whether the model performs similarly with only a few layers trainable. Furthermore a segmentation system can be designed, such that only the relevant parts of the image are used as input.

Instead of using a multilabel classifier, a multi-output multiclass model could be used as well. The use of segmentation to extract only a certain feature from the flower could increase the performance, as it greatly decreases the background noise. Furthermore using a multiclass classifier, the data imbalance problem could be solved by using data level techniques or ensemble techniques, as described in [16].

When a good enough performance on the feature classifier is achieved, a species classifier can be added to the model. This classifier can use the predicted features to predict to which species it belongs to by using for example a support vector machine. Next, this model could be applied to make a real life application, for example a smart phone app. In which you could make a photo of the orchid and the species get returned.

References

[1] J. V. F. A. M. P. G. S. C. v. d. B. A. S. Mark W. Chase, Kenneth M. Cameron, “An updated classification of orchidaceae,” *Botanical Journal of the Linnean Society*, vol. 177, pp. 151–174, 2 2015.

[2] O. Sharma, *The title of the work*. 7 West Patel Nagar, New Delhi 1 10 008: Tata McGraw-Hill Publishing Company Limited, 17 ed., 2007. page 396.

[3] Xiaoling Xia, Cui Xu, and Bing Nan, “Inception-v3 for flower classification,” in *2017 2nd International Conference on Image, Vision and Computing (ICIVC)*, pp. 783–787, 2017.

[4] Y. P. C. Y. Yong Wu, Xiao Qin, “Convolution neural network based transfer learning for classification of flowers,” *2018 IEEE 3rd International Conference on Signal and Image Processing (ICSIP)*, 2018.

[5] M. Cıbuk, U. Budak, Y. Guo, M. C. Ince], and A. Sengur, “Efficient deep features selections and classification for flower species recognition,” *Measurement*, vol. 137, pp. 7 – 13, 2019.

[6] M. M. Sani, S. B. Kutty, H. A. Omar, and I. N. M. Isa, “Classification of orchid species using neural network,” in *2013 IEEE International Conference on Control System, Computing and Engineering*, pp. 586–589, 2013.

[7] H. Zhang, G. He, J. Peng, Z. Kuang, and J. Fan, “Deep learning of path-based tree classifiers for large-scale plant species identification,” in *2018 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*, pp. 25–30, 2018.

[8] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” in *ICANN*, 2018.

[9] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[10] D. J. S. H. e. a. Russakovsky, O., “Imagenet large scale visual recognition challenge.,” *Int J Comput Vis*, vol. 115, pp. 211–252, 2015.

[11] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *CoRR*, vol. abs/1409.1556, 2015.

[12] K. He, X. Zhang, S. Ren, and J. Sun, “Identity mappings in deep residual networks,” *CoRR*, vol. abs/1603.05027, 2016.

[13] F. Chollet, “Xception: Deep learning with depthwise separable convolutions,” *CoRR*, vol. abs/1610.02357, 2016.

[14] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the inception architecture for computer vision,” *CoRR*, vol. abs/1512.00567, 2015.

[15] 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.

[16] K. Johnson, J.M., “T.m. survey on deep learning with class imbalance,” *Journal of Big Data*, vol. 6, no. 27, 2019.

[17] S. Wang, W. Liu, J. Wu, L. Cao, Q. Meng, and P. Kennedy, “Training deep neural networks on imbalanced data sets,” pp. 4368–4374, 07 2016.

[18] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, “Focal loss for dense object detection,” *CoRR*, vol. abs/1708.02002, 2017.

[19] J. Read and F. Pérez-Cruz, “Deep learning for multi-label classification,” *ArXiv*, vol. abs/1502.05988, 2015.

[20] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. Software available from tensorflow.org.

[21] F. Chollet *et al.*, “Keras.” https://keras.io, 2015.

[22] D. Apriyanti, L. Spreeuwers, P. Lucas, and R. Veldhuis, “Orchid Flowers Dataset,” 2020.

[23] L. Buitinck, G. Louppe, M. Blondel, F. Pedregosa, A. Mueller, O. Grisel, V. Niculae, P. Prettenhofer, A. Gramfort, J. Grobler, R. Layton, J. VanderPlas, A. Joly, B. Holt, and G. Varoquaux, “API design for machine learning software: experiences from the scikit-learn project,” in *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

[24] A. Tharwat, “Applied computing and informatics,” tech. rep., Frankfurt University of Applied Sciences, 2018. https://doi.org/10.1016/j.aci.2018.08.003.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, “Scikit-learn: Machine learning in Python,” *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[26] L. van der Maaten and G. Hinton, “Visualizing data using t-SNE,” *Journal of Machine Learning Research*, vol. 9, pp. 2579–2605, 2008.