Self-learning software to identify slipper orchids from pictures of flowers to simplify identification for customs officers

Figure 1 *Brachypetalum bellatulum* [1]

Patrick Wijntjes, s1057924

University of Applied Sciences Leiden, Bio-informatics, Technology cluster
Zernikedreef 11
2333 CK Leiden

Period 02-09-2013 until 20-06-2014
ECTs: 59

Naturalis Biodiversity Center
Sylviusweg 72
2333 BE Leiden

Supervisors:
Dr. Barbara Gravendeel and Dr. Rutger Vos

Lecturer from school:
Drs. Jan Oliehoek

Abstract

Slipper orchids are very popular ornamentals among hobbyists and therefore highly endangered due to over collecting in the wild. To prevent them from going extinct all slipper orchids were placed on the international CITES list, which means that permits are required for trade all over the world. Some slipper orchid species are more endangered than others. This is reflected by different legislation as some species are placed on CITES Appendix 1 (highest level of protection) whereas others are placed on Appendix 2 (lower level of protection). Flowers of slipper orchids can be classified by characteristics like shape of the petals, the absence or presence of wards on the petals and the colour of the lip. During this internship the concept of classifying slipper orchids to section level from pictures was tested. To test the possibilities of picture-based classification, pictures of well-identified orchids were collected. From these, different sets were created, one for training Artificial Neural Networks (ANNs) and one for testing them. ANNs are computer models based on the nervous system of animals. ANNs can be trained in pattern recognition. In this project the ANNs were trained in recognition patterns in colour intensities of 50 vertical bins and 25 horizontal bins. During this training, connections between different “neurons” will be created. These connections together form a network, which can be used in pattern recognition. After training, the created network was tested. This means that the network received a new picture. This picture was translated to the colour intensities of 50 vertical bins and 25 horizontal bins. These intensities were used to find a pattern the network already knows. When the network had found a pattern, the sections that contains this pattern was saved as the output of the classification.

The results show that some sections were harder to classify than others. Adding new pictures to the training set could improve the classification, but could also worsen the classification, depending on the quality of the new pictures and how well the new networks were trained. In general, better-trained networks, trained with more than a single picture, were able to correctly classify slipper orchid flowers, showing that visual recognition software can be used for identification of these orchids.

To make the classification software available to the outside world, a first concept of a website was developed. Users can upload a picture of a slipper orchid flower to the website. The website then runs the classification software using a pre-trained ANN. The result of this classification is then translated to a human readable result. This result is returned to the website. With this application, customs officers can check whether the permits of imported slipper orchids are correct. As compared to classification using a book, the website improves the identification by laymen which will help customs officers to prevent illegal trade of these orchids.

Table of Contents

Introduction

Illegal trade in endangered orchids

There are thousands of different orchid species known all over the world [2]. None of these are allowed to be imported into the Netherlands without CITES permits. Since 1973 orchids are primarily protected by the Convention on International Trade in Endangered Species of Wild Flora and Fauna (CITES), which is signed by over 120 nations [3-4]. Despite this convention many orchids are traded illegally. To trade species that are protected by CITES, a licence or certificate is required.

It is difficult to monitor the illegal trade of orchids because some orchids look very similar to non-protected plants and so accurate identification can be very difficult. To improve identification, software that can identify orchids from pictures of tubers, leaves or flowers could be vital. This software would in particular aid customs officers and employees of nature conservation organizations involved in confiscating illegally-traded material. During this project, the focus was on slipper orchids and orchids from which *salep* is produced.

Slipper orchids

In Europe and Asia the slipper orchids (Cypripedioideae) are widely distributed between sea level up to 2000 m altitude. They prefer to live in calcareous environments and are found in deciduous or mixed deciduous and coniferous woods. They grow best in light to deep shade. The slipper orchid is an herbaceous perennial plant species that has a long lifespan. It can grow up to 60 cm and each season the slipper orchid will produce new growths. Each stem of the orchid can contain 3 to 4 leaves that often have upwardly curved sides. The flower stalk can be one-flowered or two-flowered with leaf-like bracts. The sepals and petals are rarely green but commonly brightly coloured. These sepals and petals are also often twisted [5]. Slipper orchids are highly desired ornamentals, wich is why they were placed on CITES Appendices, which means they are protected from unregulated export. Different levels of protection exist: the genera *Paphiopedilum* and *Phragmipedilum* and the species *Aerangis ellisii, Dendrobium cruentum, Laelia jongheana, Laelia lobata, Peristeria elata* and *Renanthera imschootiana* are placed on Appendix I (very strict control in trade) whereas all other species of ORCHIDACEAE are placed on Appendix II (less strict control in trade) [6].

Salep orchids

Ground orchid bulbs of the Orchidoideae, also known as *salep,* are very popular in Turkey and adjacent countries in central Asia. They are used to produce ice cream in summer and hot drinks during winter. *Salep* is also used as medicine. In the early 1990s the trade of *salep* increased strongly. Official statistics from the Turkish State Institute of Statistics shows that the export between 1995 and 1999 was 282.000 kg annually. To achieve this amount of *salep* 9.825.000 – 19.650.000 bulbs are required. It is unknown if this information is related to pure *salep*, substitutes or mixtures. However, as this harvesting is unsustainable, laws have been established to protect these orchids. In Turkey there are three laws that protect them: the first law is the Turkish Forest Law, which regulates the use of non-wood forest products. In short, this law states that it is forbidden to collect and remove any form of forest vegetation. The second law, the Turkish Law of Natural Parks, states that “The production of forest products, hunting and disturbing the natural balance is prohibited.” Since collecting *salep* is classified as production of forest products, it is prohibited in all protected areas. The final law in Turkey is The Regulation on Collection, Production and

Export of Bulbs of Wildflowers. As the title of this law suggests, this law regulates the production and the export of bulbs, roots and tubers of flowers. It also defines a list with species that may not be taken away from the wild for export [7]. The exact ingredients in *salep* cannot be identified without molecular identification tools, which makes it difficult to enforce these laws.

How a web application can improve control in illegal orchid trade

To make it easier to follow the trade routes of orchid smuggling, a web application that can identify illegally traded orchids is desired. This application could be used on laptops or desktops as well as mobile devices by taking pictures of flowers, leaves or underground tubers and uploading the pictures to a website. A simple flowchart of the application is shown in figure 2. In this project the focus was on creating the website and integrating the identification application for pictures taken of flowers of slipper orchids. This application is currently under development at Naturalis.

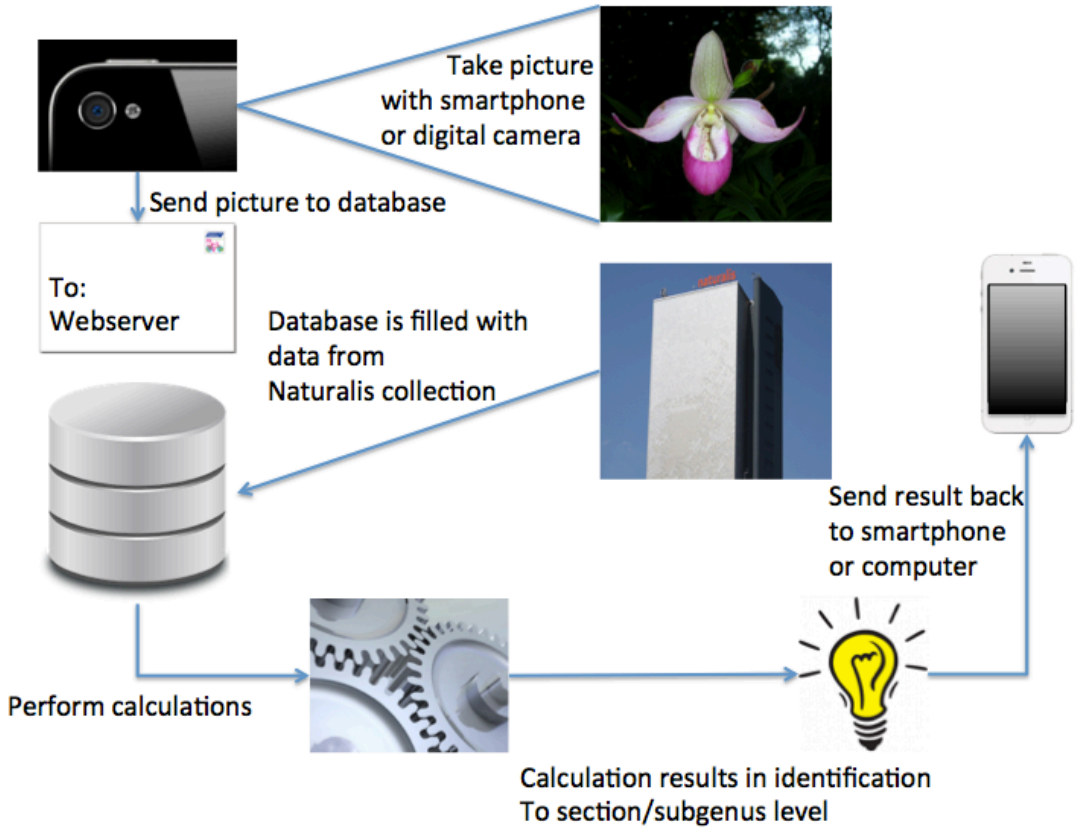

Figure 2 A simple flowchart of the application made during this project. Resources of the pictures: [8-15]

Comparable software

There is software available that can identify a person using face recognition, for example the software KeyLemon [16]. This software could be used to unlock a computer.
Essentially the software takes a picture or series of pictures of your face. When it takes a series of pictures it is almost always required to move your head up and down and / or left and right. The software saves this picture or pictures. When you use the software to unlock your computer the software takes a picture or series of pictures of your face and compares this with the saved picture(s). If it finds a match you will be logged-in to your account. The

existence of this kind of software demonstrates that it is possible to identify objects that look very similar to each other, such as two different persons. If this is possible, it would also be possible to identify orchids to section level. All sections have some very specific characteristics that are unique to that section.

Another example of comparable software is LeafSnap [17]. LeafSnap is an iPhone and android application that allows the user to take a picture of a leaf. After taking a picture the software turns the picture in a way that it can recognize the shape. The shape of the leaf will be used to create a list of possible trees.

These applications differ from the pipeline that was developing during this internship in several aspects. First of all, KeyLemon is made to recognize a human face and LeafSnap is made to recognize different leaves. The pipeline that was developed during this internship focused on slipper orchid flowers and *salep* orchid tubers. Secondly the pipeline will be available on a website, while the comparable software needs to be installed on a local machine (e.g. a computer or a smartphone). This makes the pipeline usable from different platforms. The comparable applications are specific for smartphones or computers, and also to the Operating System (OS) of the used device.

Artificial Neural Networks

Artificial Neural Networks (ANNs) are computational models based on the animal nervous system. They are capable of pattern recognition. To train a ANN, the pictures in the train data was translated to the colour intensities of Red, Blue and Green (RBG) for 50 vertical bins and 25 horizontal bins. These colour intensities were used to create connections between different “neurons”. These connections together form a network. A part of the training was coincidence. Coincidence caused different connections between networks. This may result in different classifications between two networks. Figure 3 shows a schematic representation of an ANN. In this study, the inputs were the colour intensities. In the hidden layer the network search for a known pattern. The outputs were the seven numbers that represents the seven different sections.

To test the created network, a new picture was load into the network. This picture was also translated to the colour intensities of RBG for 50 vertical bins and 25 horizontal bins. The network used its connections to search a pattern in the data of the new picture that the network already knows. Because the network was trained on recognizing patterns of seven sections of the slipper orchid, the output will be a list with seven numbers. These numbers corresponds with the seven sections. The seven sections were sorted from Z to A. the index of the positive number is the index of the section.

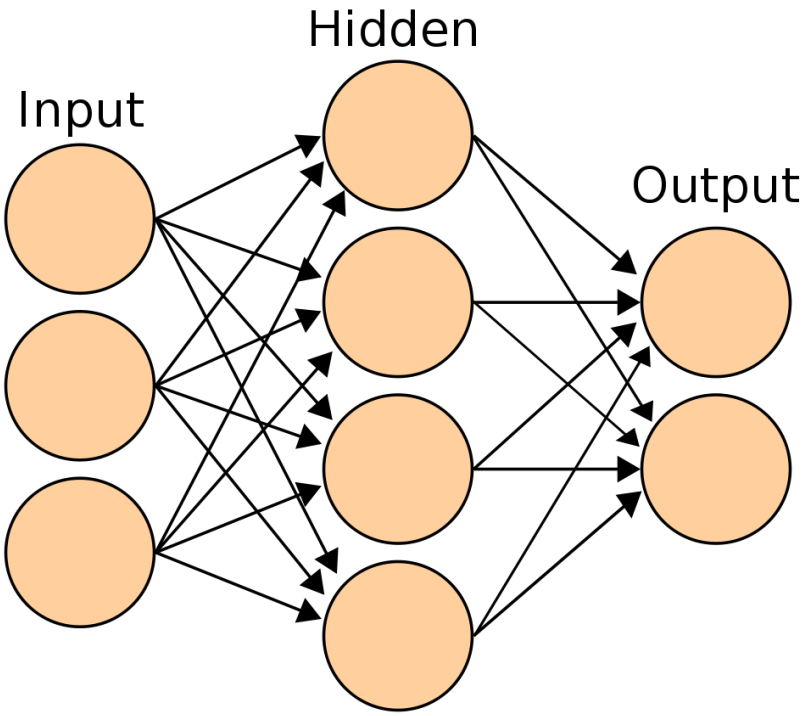

Figure 3 Schematic representation of an Artificial Neural Network [18]

Goal of this internship

The goal of this internship was to demonstrate that it is possible to classify slipper orchid flowers from pictures. Once the classification software works sufficiently accurate, it will be available online. Customs officers can then use this software to check whether accompanying permits are correct when confiscating material. If this control is simplified, it will help prevent illegal trade in slipper orchids, which will prevent them from going extinct in the wild.

Materials and Methods

Website

During this internship a website was developed. Users of this website can upload a picture to the server. On the server the software will classify the orchid on the picture to section level. The processes behind this website, like clicking on a button, were written in Python 2.7 using the Django package. The layouts of the webpages are written in HTML, using CSS style sheets. There are two versions of every HTML file, one for computers and one for mobile devices. The different CSS style sheets, HTML files and python scripts can be found in appendices 1.2, 1.3 and 1.6. To make the website usable, a well-trained Artificial Neural Network (ANN) is required. To train this network, pictures of slipper orchid flowers were needed. These pictures were saved on a private shared Flickr account on https://www.flickr.com/photos/113733456@N06/, to make the pictures accessible to everyone who needs them.

Flickr

To store the pictures in a safe place where they are accessible to whoever needs them, a shared Flickr account was created. Flickr is a website for saving and sharing pictures. Because the flower pictures are from a private collection of David Roberts, it is not allowed to freely share the pictures of the flowers, so the settings of the Flickr account were set to private. This means that only persons with the account name and password can access the pictures.

On Flickr it is possible to add metadata tags to the pictures. These tags were used later in the preparation process to save the pictures in the correct directory. To download the pictures and the metadata via the command line, a python script written by Hugo Haas, Offlickr.py, was modified and used (see appendix 1.5.4) [19].

Training

Preparation

Before training of the ANNs is possible, a preparation step is needed. First of all, pictures of the *salep* orchid tubers and the tubers of the look-a-likes were required. So pictures of *salep* orchid tubers and tubers of look-a-likes were taken at the Sylvius lab. It is required that the orientation is the same for all pictures. For instance, if the first picture of a tuber with appendices has the appendices on the right, all other tubers with appendices must have the appendices on the right as well. The user has to use the same orientation as the trainer. The background has to be one colour, like white or black, and this colour must be the same for every picture. The last requirement is that there is only one tuber on the picture.

The second step in this process is to download the pictures and metadata and search for the tags in the metadata. An example of the metadata can be found in supplementary 1.1. The blue square indicates the location of the tags. The next step is to convert the pictures from .jpg- to .png-format. This is needed for the training scripts, which work only with .png files. Finally the pictures will be placed into the correct directory using the tags. The tuber pictures were divided by shape and look-a-like or not, which resulted in six directories. Round, Spur and Oblong for the different *salep* orchid tubers, LRound, LSpur and LOblong for the different look-a-like tubers. The slipper orchid flowers were divided by section and within each section the pictures were divided by species. A section is a taxonomic level below a genus but above a species. Sections were created to obtain a better

overview of species rich genera. Species in he same section share a common ancestor. In this project the classification of Chochai, who carried out molecular phylogenetic research to come up with an evolutionary based list of sections, is used [20]. Some sections contain many species, such as *P. Barbata* and *P. Paphiopedilum*. Other sections just contain a few species, such as *P. Pardalopetalum* and *P. Cochlopetalum*.
After separating the pictures of the tubers from the flowers, the pictures of the tubers will be segmented. This means that the background is made entirely white and the tuber is cut out of the picture. It was not necessary to split the flower pictures, because these pictures were already segmented.

Slipper orchids

A total of 5 genera within the slipper orchids exist. The genera are *Cypripedium, Mexipedium Paphiopedilum, Phragmipedium* and *Selenipedium*. Due to time constraints, only the genus *Paphiopedilum* was used in this project. The sections of this genus are *Barbata, Brachypetalum, Chochlopetalum, Coryopedilmu, Paphiopedilum, Pardalopetalum* and *Parvisepalum*, which are based on molecular phylogenetic research [20]. The ANNs were trained on all these sections and tested on the four sections *Barbata, Brachypetalum, Coryopedilum* and *Parvisepalum* The ANNs were trained on more sections than they were tested with. This resulted in more realistic outcome, because the ANNs have to classify a new picture to all possible sections. If the ANNs were only trained on the sections used in the test runs, this could affect the outcome in a way that the results look much better because 3 of the 7 sections were missing.

The script that trained the network was already developed at Naturalis by Rutger Vos. This script accepts a parameter to specify the Desired Error. This is the maximum error rate that will be accepted. If the parameter is left empty the Desired Error will be 0.0001. When the error drops below this threshold the training is finished. To find the optimal Desired Error, seven networks were trained with a Desired Error in tenfold increments: the first network had a Desired Error of 0.0001, the second network had a Desired Error of 0.00001 and so on. The expectation was that a network with a lower Desired Error would classify a new picture better. To semi-automate the preparation for the training some scripts were written in python and bash. The workflow of these scripts can be found in figure 4. These scripts were for training the Artificial Neural Networks (ANNs), which is only done by the administrator of the website and not by the end-users of the website. The administrator has access to the shared Flickr account, so the manual check (and download) of the pictures that is required for one of the scripts, would not be a problem. Besides networks for classifying the slipper orchid flowers, ANNs for classifying *salep* orchid tubers were also created.

Salep orchids

Salep orchid tubers are very popular in Turkey. These tubers are used to make ice creams during summer and hot drinks during winter. Due to a lack of information about the collected tubers, the ANNs could only be trained to recognize differences between *salep* orchid tubers and look-a-likes obtained from non orchid families. The tubers of look-a-likes that were used were from *Arum maculatum* (Araceae), *Asparagus officinalis* (Asparagaceae), *Polygonatum verticillatum, Tulipa greigii,* and *T. sp.* (Liliaceae) [21]. Supplementary 2 contains pictures of these tubers.

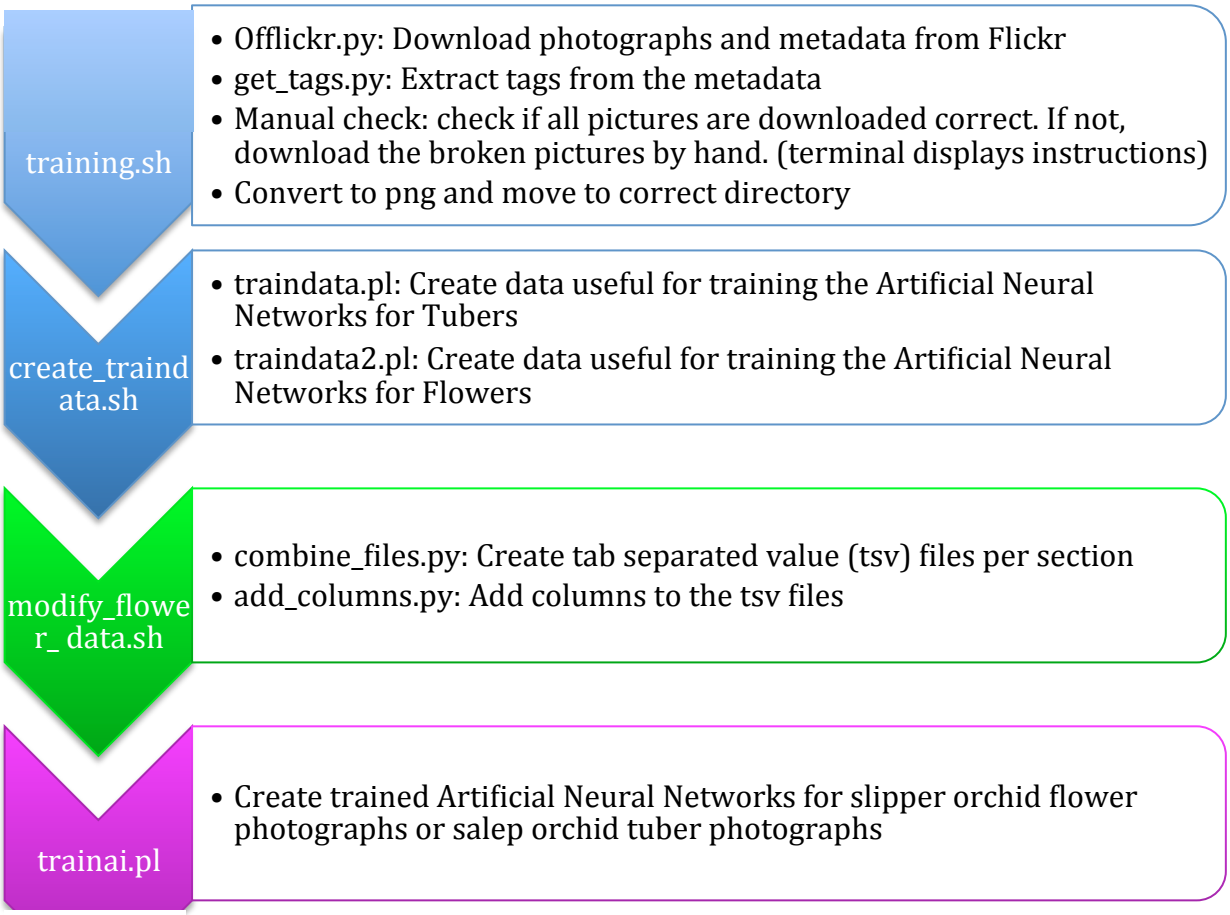

Figure 4 Flowchart of training preparation scripts. Blue: preparation steps. Green: processing step to produce useful data for training the flower Artificial Neural Networks. Purple: Training of the Artificial Neural Networks

Testing the networks

Slipper orchids

After training the networks for the slipper orchid flowers, they were tested on pictures of four sections. The tested sections were *Barbata, Brachypetalum, Coryopedilum* and *Parvisepalum*. The networks were also tested with pictures of some primary hybrids of *Parvisepalum* and *Brachypetalum* with the aim of testing if it is also possible to correctly identify hybrids with the software. The assumption was that for primary hybrids a trained network would ideally list the sections of both parents as possible identification. However, some results of the first test round were very poor. To find the source of these bad results, pictures of some poorly classified species were checked for detection ability of the defining characteristics of the corresponding section. Table 1 holds a list with the characteristics per section based on Cribb P. [22]. During this research it was discovered that some species had only one picture in the train data. Therefore, extra pictures were added to the train data and the networks were trained again to find out if better results could be obtained after more extensive training. These networks were tested again with the same test sets.

Table 1 characteristics per section

Characteristic	*Barbata*	*Brachypetalum*	*Coryopedilum*	*Parvisepalum*
Petals	Spathulate	Broad-elliptic, pale yellow or white	Long, hanging down, spirally twisted, tapering, glandular at tip	Broad-elliptic, Subcircular, Cream/yellow coloured
Spotted or warted?	Yes	Yes	Yes	No
Lip	Incurved side lobes	Small, ovoid with incurved margin	Incurved side lobes	Large, thin-textured
Staminode	Lunate	Ovate to transversely elliptic	Oblong	Large

Salep orchids

Due to time constraints, pictures of these orchids could not be investigated further during this internship.

Testing the website

To test the website, a working ANN was required. After testing the ANNs, a working ANN was implemented on the website. This website was tested on receiving a picture, running the classify script using the given network and sending back a result. First this was tested for only one device. Once this was working correctly, the website was also tested to receive pictures from more than one device at the same time and send a result back. Receiving the pictures was working correctly, but sending back a result was not. The result of the first uploaded picture was send back to all devices. So this part of the website was made operable during this internship. The IP-address was used to save the uploaded picture in a directory. The name of this directory will be the IP-address. Only the picture in the directory with the correct IP-address will be classified, and its result is sent back to the website.

Results

Website

To make the software available for end-users a website was developed. A first design of this website was finished during this internship. It consists of a homepage, a page for uploading a picture, a page to show the picture is uploaded correctly, and a results page. On the homepage it is possible to choose to upload a picture or remove unused files from the server. The last option requires to log in with a valid account. The administrator of the website has a valid username and password. After choosing the option of removing unused files, the user is redirected to the login page. A workflow of this website is shown in figure 5. Figure 6 zooms in on the uploading part of the website. After selecting the upload option the user is forwarded to the upload page. On this page the user can select a picture to upload. On iPhones it is also possible to take a picture after tapping the “select file” button. The website will check whether the selected file is a picture. If it is not, the user stays on the upload page and a warning is issued. After uploading the file, the picture will be renamed. When these modifications are done the user is directed to the upload success page. Here the user can see the uploaded picture. The user can choose to see the results or cancel. If the user selects the result option, the classify.pl script will run to classify the picture (see appendix 1.4.1). A flowchart of the classification step of the website can be found in figure 7. The output of this program is a list with numbers. An example of this output can be found in supplementary 1.2. This list is sent to a python script, result.py, that translates this list to a readable result (see appendix 1.6.2). Figure 8 shows a flowchart of this script. The output of this script is sent back to the result page, where the user can view it. Supplementary 3 contains screenshots of this website.

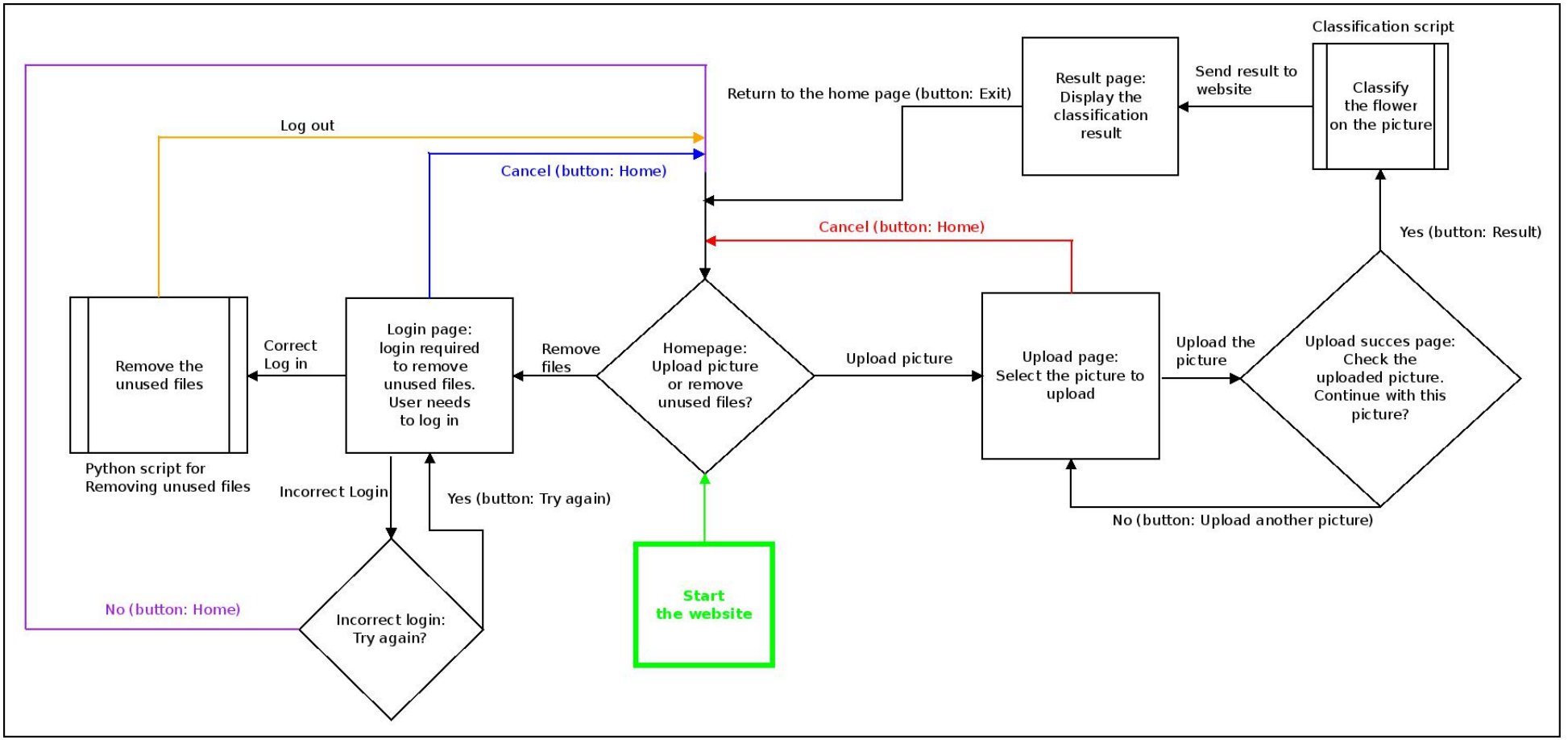

Figure 5 Flowchart of the website. Green: Start of the workflow, start the website.

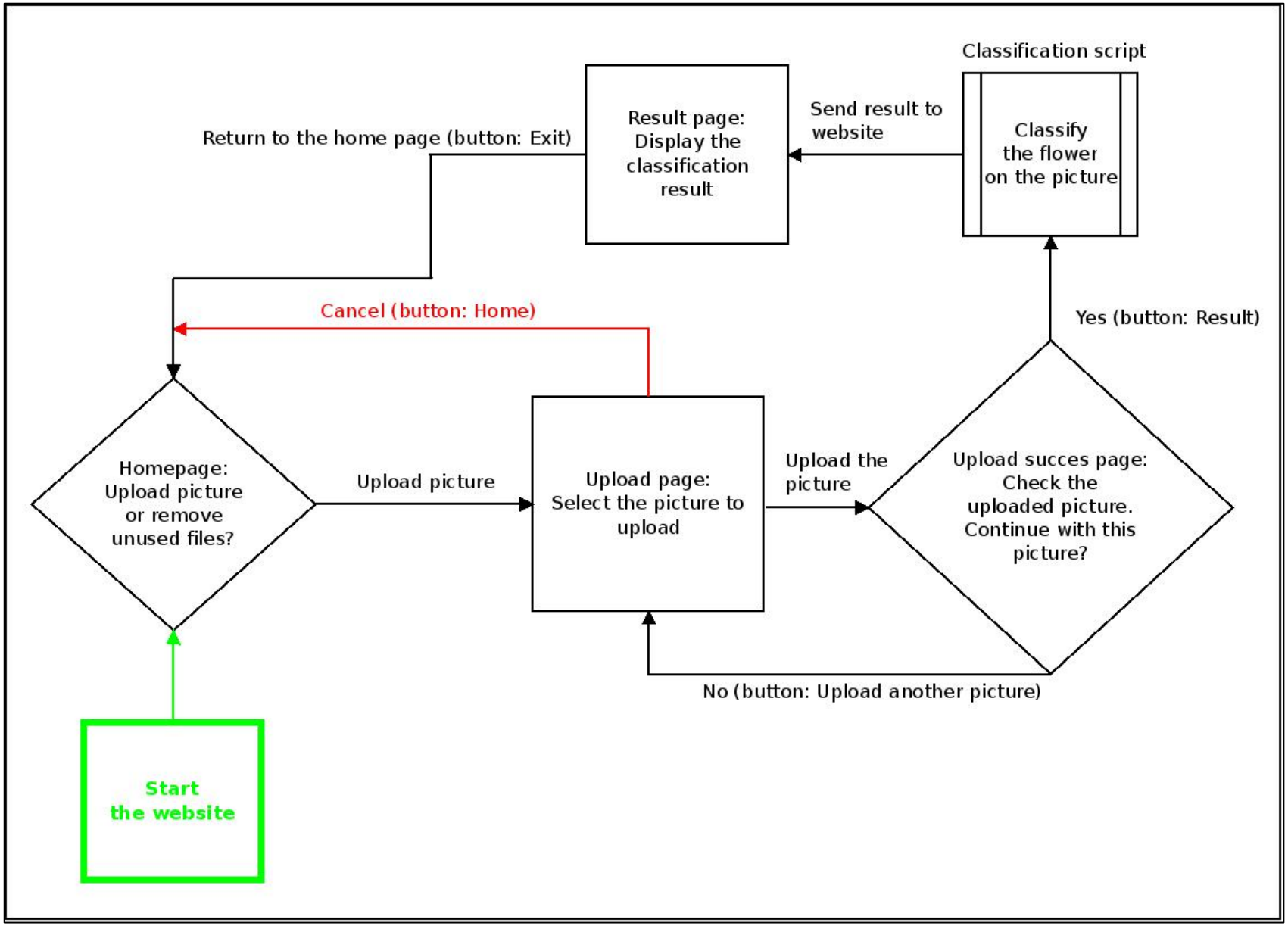

Figure 6 flowchart of the website, zoomed in on uploading. Green: Start of the workflow, start the website.

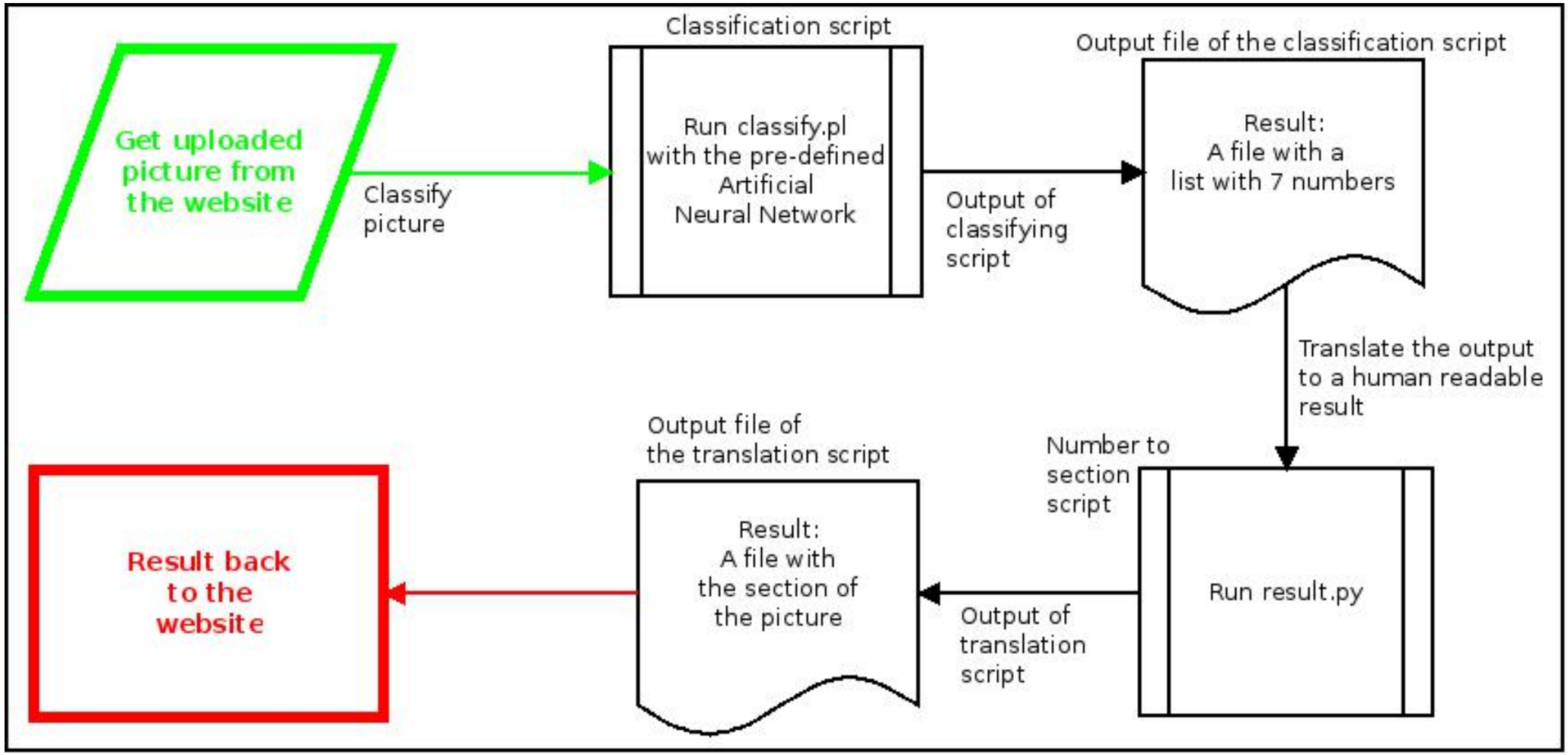

Figure 7 Flowchart of the classification step of the website. Green: Start of the workflow, get the picture, Red: End of the workflow, send the result.

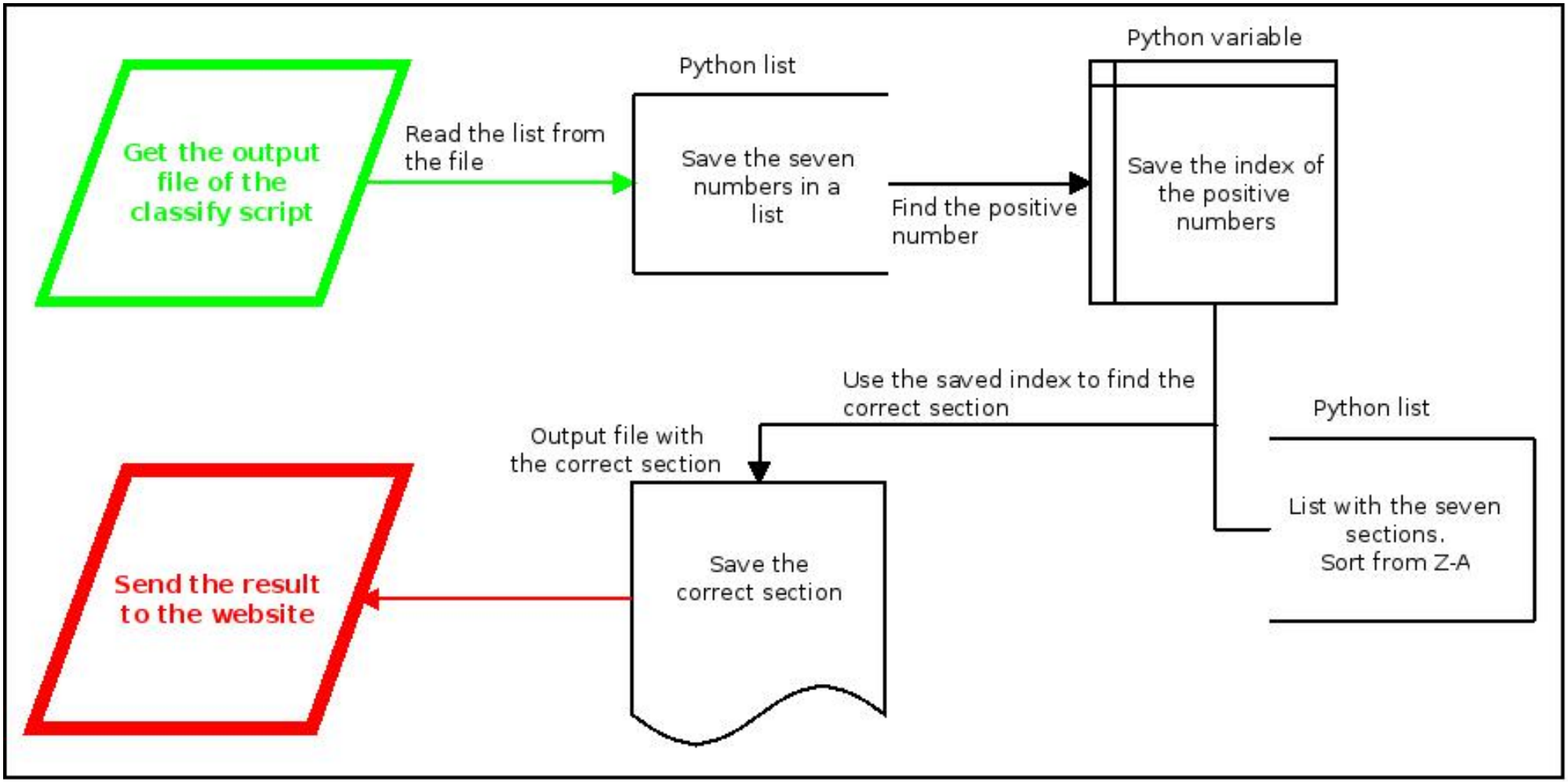

Figure 8 Flowchart of result.py. Green: Start of the workflow, get the input file. Red: End of the workflow, send the result.

Preparation scripts

To semi-automate the preparation process, a bash script, training.sh, was developed (see appendix 1.1.3, and the first script in figure 4). Figure 9 contains a flowchart of this script. The first step in this script was running Offlickr.py to download the pictures as .jpg and the metadata files as .xml. After downloading these files, the script will run another python script, get_tags.py (see appendix 1.5.3), to get the original names and the tags from the metadata. A workflow of this script can be found in figure 10. This info will be saved in .txt files, using the id of the picture as name. For example, if the picture name is 123456789.jpg the tag file of this picture is 123456789_tags.txt. At the end of this step the .xml files are removed. Example of different tag files can be found in supplements 1.3-1.5. The structure of these files is always the same: the first line is the original name of the picture, then an empty line, after that the shape for the tuber pictures or the section and species for the flower pictures. The last tag indicates whether the object is a slipper orchid flower or a *salep* orchid tuber. The knowledge of this structure can be used in the next step.

In the next step the pictures are divided between two directories, Flower and Tuber. Before the pictures and tags are moved to the correct directory, the pictures were converted from .jpg to .png. This conversion was required because the training scripts only work with .png files. After converting the pictures, the .jpg files were not used anymore, so all .jpg files were removed.

After dividing the pictures between the directories Flower and Tuber, the separation goes further. First the Flower pictures are divided between the different slipper orchid sections and species. After this division there are some directories with the section names inside the Flower directory, and every section directory contains further species directories. After dividing the Flower pictures, the Tuber pictures are divided between shape and Look-a-Like or orchid. This step will produce six directories: LOblong, LSpur, LRound, Oblong, Spur and Round. All directories starting with an “L” are for the Look-a-Like tubers.

The last step in the preparation process is segmenting the pictures of the tubers, using a Perl script developed by Rutger Vos: splitter.pl [23] (see appendix 1.4.2). This script uses the Perl package Image::Magick to modify the picture so that only the tuber is on the picture, with a minimum background. With an option of Image::Magick the background is modified to be completely white (see the square in appendix 1.4.2). Figure S-11 in supplementary 2 shows a picture before and after segmentation. The pictures of the flowers were already segmented, so this step was not required for these pictures.

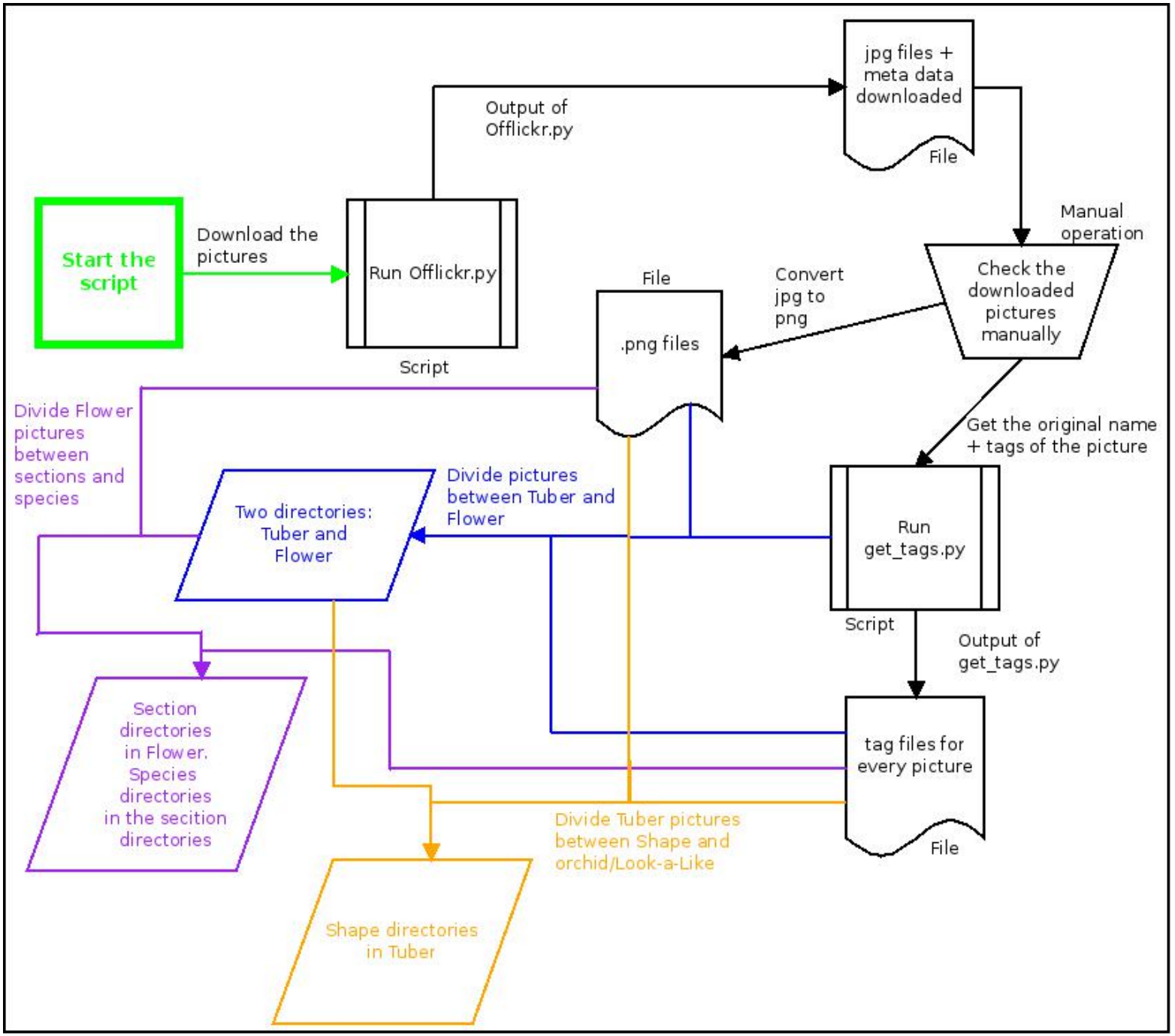

Figure 9 Flowchart of training.sh. The colour of the arrows indicates in which step they are involved.

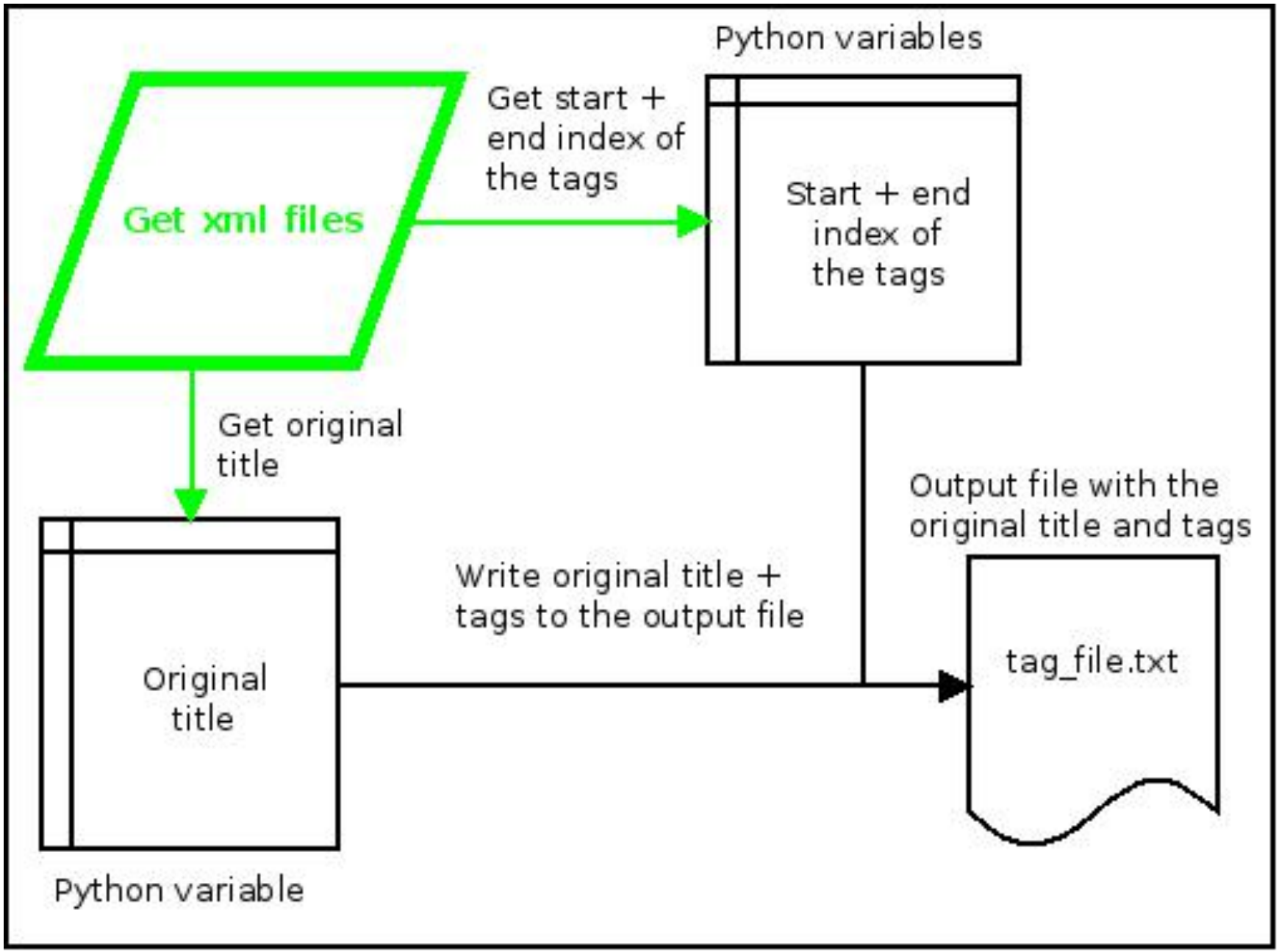

Figure 10 Flowchart of get_tags.py. Green: Start of the workflow.

Create training data

When the preparation script is finished the training data can be generated. To do this, a bash script, create_traindata.sh, was developed (see appendix 1.1.1, and the second script in figure 4). This script runs two different Perl scripts developed by Rutger Vos. One of these scripts, traindata.pl, is for the tubers and the other, traindata2.pl, for the flowers (see appendices 1.4.4 and 1.4.5). The only difference between these two scripts is the accepted input files. The script for the tubers only accept files with a name in the format <number>,<number>.png, while the other script accepts files with a name that contains at least one number. These scripts create .tsv (tab separated value) files for every directory. So after running create_traindata.sh there is a .tsv file for every species of the flower pictures and a .tsv file for every shape of the tuber. These .tsv files contain the intensities of Red, Blue and Green (RBG) of 50 vertical bins and 25 horizontal bins. This means that the classification is based on both colour and shape. The shape was used to create a digital contour of the flower. The colour was used to create a digital phenotype inside of the digital contour. Figure 11 shows an example of the translation from picture to digital phenotype.

From this point the .tsv files of the tuber can be used for training the neural networks. The .tsv files of the flowers need to be modified before using them for training the neural networks. To automate this modification process a bash script, modify_flower_data.sh, was developed that runs two python scripts (see appendix 1.1.2, the third script in figure 4 and appendices 1.5.1 and 1.5.1). The first python script, combine_files.py, creates .tsv files per section by combining the .tsv files in that specific section. When all sections have a .tsv file the second python script, add_columns.py, is run. This script creates extra columns, one column per section. After creating the columns they are filled in. The guideline for this step is: the first section gets 1's in the first column and -1's in all other columns. The second section gets 1's in the second column and -1's in all other columns and so on. The sections were sorted from A to Z. This convention was subsequently used in the classification step. This step produces a list with seven numbers, one for each possible section, sorted from Z to A. So when the last of these seven numbers is positive this means the pictures is classified as the first section and so on. After adding these columns the data can be used to train the neural networks.

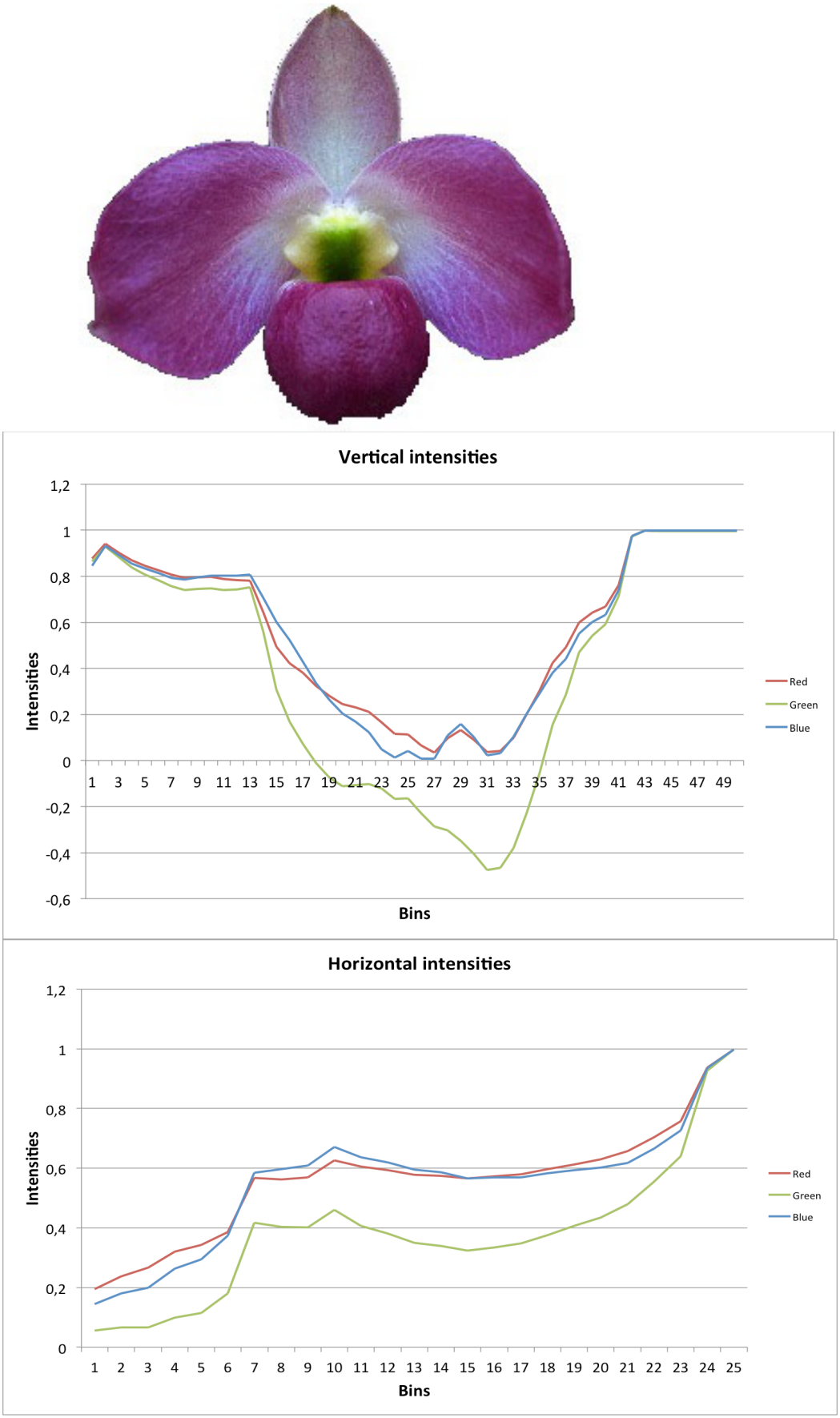

Figure 11 Example of translation from picture to digital phenotype. A purple phenotype results in high intensities of red and blue.

Neural networks

After completing these preparation process, a neural network can be trained. To train the network a Perl script written by Rutger Vos, trainai.pl [23], was used (see appendix 1.4.3). This script accepts a directory with .tsv files, the number of categories (the number of extra columns), the output file, a Desired Error and the maximal number of epochs. The last one is the maximum number of rounds the network will be trained. When either the Desired Error or the number of epochs is reached, the training process will stop. Because the network is trained with the .tsv files, it is trained to see patterns in the colour intensities of RBG. Now the network acts like it can see different colours and shapes.

During this project 14 ANNs were created for identifying slipper orchid flowers. Seven for the first test run and seven for the second test run. As mentioned earlier the Desired Error was incremented tenfold per network. The networks can be found on: https://github.com/naturalis/img-classify/tree/master/webapp/ann
All ANNs with a name ending with 4x.ann were used for the first test run and all ANN's with 5x.ann were used for the second test run. Due to time constraints, networks for tubers of *salep* orchids and look-a-likes were created, but not tested.

First test run

In the first test run four of the seven sections of *Paphiopedilum*, i.e. *Barbata, Brachypetalum, Coryopedilum* and *Parvisepalum* were tested. The ANNs were also tested with a few pictures of hybrids between species of *Parvisepalum* and *Brachypetalum*. The results of these tests can be found in supplementary 4. The different networks are shown horizontally and the different pictures vertically. A green cell with the text 'Correct' means the network classified the picture to the correct section. A yellow cell with the text 'Unknown' means the network was unable to classify the picture to any section. A yellow cell with the text 'Unknown <number>/<number>' means the network was unable to select a section from two options. When this text is green, one of the options was the correct section. A red cell with the text 'Incorrect <number>' means the network classified the picture to a wrong section. The number indicates to which section the network erroneously classified the picture. Below, the obtained results will be discussed in more detail.

Section *Barbata*

This section is characterized morphologically by spathulated, spotted and/or warted petals and incurved side lobes of the lip. Most of the species of the section *Barbata* were classified incorrectly. As seen in table 2 in Supplementary 4, most of the incorrectly classified pictures were classified as section *Paphiopedilum*. One explanation for this mistake could be that the species in both sections share several similar characteristics. The petals for instance have mostly the same shape. Another explanation is that both sections have many species so variation is high.

After characteristic pictures of three different species of the section *Barbata* were added to the test set, belonging to *P. argus, P. tonsum* and *P. wardii*, it turned out that even pictures with all characteristics of the section clearly visible were hard to classify correctly (see table 3 in supplementary 4). Like before, one of the possible explanations could be that species of the section *Barbata* share characteristics with the section *Paphiopedilum*. Another explanation could be that not enough train data of these species were added yet.

Section *Brachypetalum*

This section is characterized morphologically by broad-elliptic petals with purple spots, a small and ovoid lip with an incurved margin and pale yellow or white flowers. The visual recognition software developed during this internship identified this section very well (see table 4 in supplementary 3). None of the pictures were classified incorrectly and only a few were classified as unknown. But even then, most of the time the correct section was one of the options. A possible explanation could be that this section contains few species only so variation is low.

Section *Coryopedilum*

This section is characterized morphologically by long, tapering, spirally twisted and hanging petals with warts on the margins and glands at the tip and a lip with incurved side lobes. In this section one picture stands out, *P. rothschildianium*1. This picture was classified correctly with all networks (See table 5 in supplementary 4). So this is a good picture for classifying. Further, it is shown that the other two pictures of this species were much harder to identify. Possible explanations are that the first picture has more characteristics visible than the other two (i.e. spirally twisted petals and incurved side lobes) or that the resolution of the first picture is better than the other two.

Section *Parvisepalum*

This section is characterized morphologically by broad-elliptic, subcircular cream/yellow coloured petals and a large staminode. This section also has a striking picture, *P. emersonii*2_1. This picture was always incorrectly classified. Research on this picture shows that the petals are darker than the petals on the other pictures of this section. This is caused by some shadow on the flower. In addition, table 6 in supplementary 4 shows that *P. armeniacum* was the best recognized species. All test pictures of this species were classified correct by all networks. A possible explanation could be that this section contains few species only so variation is low.

Hybrids of species from sections *Parvisepalum* and *Brachypetalum*

Besides testing the ANNs with pictures of the sections, the networks were also tested with pictures of hybrids of species of different sections. This was done to test if it was possible to identify one or both of the parental species. It could be hard to identify both parental species because the hybrid shares a part of its morphology with the mother and another part with the father. Since the ANNs were not trained with pictures of hybrids, the assumption was that only one parental section would be found by the neural networks. Table 7 in supplementary 4 shows that most of the time the pictures were indeed classified as one of the parental sections. In just one case, *armeniacum* x *concolor* using the network with Desired Error 0.00001, both parental sections were found.

Characteristic research

Following the results of the first run, research on the characteristics of some species was done to understand why some species or sections were hard to classify. The results of this research can be found in supplementary 5. These results show that most of the test pictures do not have all section characteristics visible. The results indicate also that some characteristics were shared between different sections, like broad-elliptic petals in the sections *Brachypetalum* and *Parvisepalum*. This could be a reason why some species were classified very poorly. Essential characters necessary for correct identification turned out to

be the shape of the petals, the presence or absence of wards on the petals the colour of the flower and the shape of the lip (incurved or not). Especially the colour was often distorted (i.e. white flowers looked yellow or yellow flowers looked green) which might explain some of the poor results obtained. To improve the classification it is required to standardize the pictures of the train data. Knowledge of the different sections of slipper orchids is needed to standardize the pictures of the train data.

During this research it was also discovered that some species of the sections *Barbata* and *Coryopedilum* had only one picture in the train data. This could also be a reason why these species were classified poorly. Extra pictures of these species were added to the train data. This expanded train set was used to train the ANNs again. These new networks were tested again with the same test sets to find out if better classification results were obtained. The results of this experiment are described in more detail below.

Second test run

To demonstrate that more pictures in the train data improves the ability of the ANNs to classify pictures, a second test run was done. The results of this run can be found in supplementary 6. The structure of these tables is the same as in supplementary 4. Some pictures (i.e. 50 out of 124; roughly 40%) were indeed classified better, showing that more training pays of. Other pictures (i.e. 30 out of 124; roughly 25%) were classified worse than in the first test run. One explanation could be coincidence during training. When you train two networks with the same settings and the same train set, both networks will be different nonetheless. The cause of this phenomenon is that during the training the networks create connections. These connections will be different every time you train the network. To classify a new picture the connections in the network found at that moment are used to give a result. This means that different connections can result in a different classification. The remaining 44 pictures (roughly 35%) were classified as good as in the first test run.

Section *Barbata*

After the second test, species belonging to section *Barbata* were still difficult to classify (See tables 12 and 13 in supplementary 6). The results do show that many of the pictures of this section were classified better (i.e. 25 out of 45; roughly 56%). It is striking that some pictures of species with additional pictures in the train set were classified worse, though, because the assumption is that more pictures in the train set will improve the classification. These concerned the species *P. argus*, *P. hennisianum, P. tonsum* and *P. wardii*. A possible explanation is the different connections in the ANN's of the first test run and the second test run.

Section *Brachypetalum*

Brachypetalum is one of the sections that did not get additional pictures in the train set. Nevertheless, there are some pictures that were better classified than before (See table 14 in supplementary 6). These concerned some pictures of *P. concolor* and *P. godefroyae*. But also in this section there were some pictures that were classified worse. These concerned other pictures of *P. concolor* and *P. godefroyae* and also some pictures of *P. niveum*. A possible explanation is again the different connections between the ANNs of the first test run and the second test run. Another possible explanation could be that the patterns of the sections with additional pictures were more specific in the second test run. This means that in the first run the patterns of a picture of this section could be belong by two sections, but after adding pictures to the other section, this pattern does not fit anymore in this section.

Section *Coryopedilum*

This is a notable section. This section received additional pictures in the train data, but not of the species in the test data. Nevertheless, the pictures were much better classified than in the first run (See table 15 in supplementary 6). This is also the only section that has no longer incorrectly classified pictures. Still some pictures were classified as unknown. A possible explanation could be that the train data contains more variation after adding some pictures. The pictures in the test data looked more like the new variations, which made it possible to classify the picture to the correct section.

Section *Parvisepalum*

After the first test run this section was one of the best-classified sections. Although there were no additional pictures of this section added, some pictures were classified better during the second test run (See table 16 in supplementary 6). Again, one of the possible explanations could be the different connections between the first ANNs and the second ANNs. Also more specific patterns for the other sections could again be a possible explanation.

Hybrids of species from section *Parvisepalum* and *Brachypetalum*

Most of the hybrids were classified correctly in the first test run. During the second test run it was seen that some pictures were classified better to one of the parental sections (See table 17 in supplementary 6). The picture that was classified to both parental sections in the first run, was classified to only one parental section in this run. Because it was only seen one time in the first test run that both parental sections were recognized, it could be coincidence. This could be an explanation to the results of the second test run. A possible explanation for the better results of some pictures could be again the different connections between the different ANNs.

Discussion

Tubers

Because it is hard to extract DNA from a dried tuber, most of the tubers of confiscated *salep* orchids could not yet be identified to species level during my internship. During this project they could therefore only be used to train the neural network to see differences between orchid tubers and lookalike tubers up to the family (i.e. orchid versus non orchid) level. The time necessary to test this was lacking, though, so this is not worked out completely. A new student will use the pictures that were made to further train the neural network to identify them to orchids versus lookalikes.

Slipper orchids

Sections of slipper orchids that could be identified correctly included *Brachypetalum* (75-95% in the second test run), *Coryopedilum* (67-100% in the second test run), *Parvisepalum* (73-85% in the second test run) and primary hybrids of species of *Parvisepalum* and *Brachypetalum* (75-100% in the second test run). The only section that was less easy to identify was *Barbata* (24-41% in the second test run). The section *Barbata* has more characteristics in common with the section *Paphiopedilum* than the other sections, which might explain why pictures of the section *Barbata* were often classified as *Paphiopedilum*.

Training networks

During training of the networks it was found that it was very hard to find good pictures of flowers of some species. When this fact is combined with the fact that some species had few pictures in the train set, this could be a reason why these species were poorly classified. This assumption is supported by the fact that when more pictures were added to the train set, identification improved. This was especially found for *P. hennisianum, P. tonsum, P. violascens, P. glanduliferum, P. wilhelminiae* and *P. rothschildianium*.

Conclusions

Classifying slipper orchids from pictures is possible

Although some species were very hard to classify, the results show that it is possible for most species to be classified to section level correctly using a picture of a flower. Some species need more or better pictures in the train set, but the concept of classifying slipper orchids from a picture of a flower has been demonstrated during this project.

Differences between well classifiable vs hard classifiable sections of slipper orchids

The only section of those tested that remained hard to classify was the section *Barbata*. Most of the floral characteristics of this section are shared with the section *Paphiopedilum*. All other tested sections have unique characteristics. This might explain why some of the pictures of the section *Barbata* were classified as *Paphiopedilum*.
After adding pictures of this section to the training set the classification improved a little bin. Still many pictures were classified as *Paphiopedilum*. This supports the assumption that the shared characteristics make it hard to classify pictures of this section correctly.

Suggestions for follow-ups

To make this software available for the outside world, the website needs to become online. Before the website will be made online, the ANNs need to be tested with the sections *P. Cochlopetalum, P. Paphiopedilum* and *P. Pardalopetalum*. Depending on the results of these tests, the ANNs need to be trained again with more/better train data of one or more of these sections. The ANNs have to be re-trained with more/better train data of the section *Barbata* anyway.

When the website is online, custom officers can use this website to classify orchid flowers. To make the website more useful for custom officers, the software needs to be expended by classifying orchid tubers and orchid leafs.

Acknowledgements

I would like to thank the following people for helping me: Serrano Pereira for segmenting some pictures and test some of my scripts, David Roberts for providing slipper orchid flower pictures, Rogier van Vught for providing some look-a-like tubers and Benjamin Versteeg for helping me out with Django.

References

[1] David Roberts, Durrell Institute of Conservation and Ecology, University of Kent

[2] The Plant List, 2010, http://www.theplantlist.org/browse/A/Orchidaceae/

[3] Orchid Smuggling and Conservation (ORCHID), http://www1.american.edu/ted/orchid.htm

[4] http://www.cites.org

[5] Royal Botanic Gardens Kew, 2001, http://www.kew.org/plants-fungi/Cypripedium-calceolus.htm

[6]CITES Appendices, June 2013, http://www.cites.org/eng/app/appendices.php

[7] Kasparek M and Grimm U, 1999, European trade in Turkish Salep with Special Reference to Germany. Economic Botany 53(4): 396-406

[8] Camera: gsmnationblog, 2013, http://www.gsmnation.com/blog/2013/02/19/snap-away-5-best-smartphone-cameras-available/

[9] Envelop: Tuxx, 2004 - 2013, http://www.tuxx.nl/post/adressering/

[10] Naturalis tower: unityfm, 2013, http://www.unityfm.nl/nieuws.php?id=26906

[11] Database: introduction to query optimizer, 2013, http://prabathsql.blogspot.nl/2013/01/introdiuction-to-query-optimizer.html

[12] Wheelwork: op eigen kracht praktijk voor ergotherapie, 2013, http://www.ergotherapieopeigenkracht.nl/vergoeding-ergotherapie/

[13] Lamp: ledweeklampen, http://www.ledkweeklampen.nl

[14] iPhone: T-Mobile, http://shop.t-mobile.nl/eca/RAPRD/Apple-iPhone-4-8GB-wit/map48gbpwi.html?ab_agid=1021

[15] Slipper orchid: Wikimedia commons, 2009, http://commons.wikimedia.org/wiki/File:Slipper_orchid_wyn1.jpg

[16] KeyLemon, 2013, https://www.keylemon.com/product/

[17] LeafSnap, 2011, www.leafsnap.com

[18] gigaom, 2014, https://gigaom.com/2013/08/16/were-on-the-cusp-of-deep-learning-for-the-masses-you-can-thank-google-later/

[19] Offlickr, Hugo Haas, http://code.google.com/p/offlickr

[20] Chochai A, 2012, Molecular phylogenetics of *Paphiopedilum* (Cypripedioideae; Orchidaceae) based on nuclear ribosomal ITS and plastid sequences, In: Botanical Journal of the Linnean Society 170: 176-196

[21] Lawler LJ, 1984, Ethnobotany of the Orchidaceae, In: Arditti J (ed.), Orchid biology: reviews and perspectives: 27-149. Cornell University Press, Ithaca, New York, USA

[22] Cribb P, 1998, The genus *Paphiopedilum* second edition

[23] Rutger Vos, Bioinformaticist at Naturalis Biodiversity Center

[24] Photographs by Patrick Wijntjes

Supplements

1. Input and output files

1.1. Example of an xml file

<photo id="12342126885" secret="ca74c114fa" server="7451" farm="8" dateuploaded="1391690138" isfavorite="0" license="0" safety_level="2" rotation="0" originalsecret="29fddd19e8" originalformat="jpg" views="0" media="photo">
<owner nsid="113733456@N06" username="patrick_naturalis" realname="Patrick Wijntjes" location="" iconserver="0" iconfarm="0" path_alias=""/>
<title>charlesworthii5</title>
<description/>
<visibility ispublic="0" isfriend="0" isfamily="0"/>
<dates posted="1391690138" taken="2012-08-29 17:13:09" takengranularity="0" lastupdate="1391690248"/>
<permissions permcomment="0" permaddmeta="0"/>
<editability cancomment="1" canaddmeta="1"/>
<publiceditability cancomment="0" canaddmeta="0"/>
<usage candownload="1" canblog="1" canprint="1" canshare="0"/>
<comments>0</comments>
<notes/>
<people haspeople="0"/>
<tags>
<tag id="113688134-12342126885-10993197" author="113733456@N06" raw="genus:Paphiopedilum" machine_tag="0">genuspaphiopedilum</tag>
<tag id="113688134-12342126885-188931923" author="113733456@N06" raw="species:chariesworthii" machine_tag="0">speciescharieschariesworthii</tag>
<tag id="113688134-12342126885-535" author="113733456@N06" raw="Flower" machine_tag="0">flower</tag>
</tags>
<urls>
<url type="photopage">http://www.flickr.com/photos/113733456@N06/12342126885/</url>
</urls>
</photo>

1.2. Example of an output file of classify.pl

```
Entry: color-100,451.png
$VAR1 = [
          '-0.456072704368231',
          '-0.973874677375054',
          '-1',
          '-0.999999808175595',
          '0.990980698747592'
        ];
Entry: color-129,334.png
$VAR1 = [
          '-0.811027796569903',
          '-0.999812464339958',
          '0.99572448600037',
          '-0.996400768699541',
          '-1'
        ];
Entry: color-141,711.png
$VAR1 = [
          '0.971726551475096',
          '-0.999312299486709',
          '-0.981400879260904',
          '-0.99960776872456',
          '-1'
        ];
Entry: color-222,648.png
$VAR1 = [
          '0.996738096627912',
          '-0.919456120335819',
          '-0.999987677705016',
          '-0.999994486624424',
          '-0.99999973650751'
        ];
Entry: color-447,1931.png
$VAR1 = [
          '0.935067461526289',
          '-0.999821434281431',
          '-0.999766410047794',
          '-0.996420240471305',
          '-1'
        ];
Entry: color-46,379.png
$VAR1 = [
          '0.997483774166035',
          '-0.994043337286478',
          '-0.998988899850576',
```

```
        '-0.999999999293947',
        '-0.99999999995774'
      ];
Entry: color-50,380.png
$VAR1 = [
        '0.940404321606217',
        '-0.758403411429261',
        '-0.999999840072036',
        '-0.999999955862603',
        '-0.999999983567608'
      ];
Entry: color-56,651.png
$VAR1 = [
        '-0.897130009835269',
        '0.994303714419604',
        '-0.999999951703297',
        '-0.999999993987055',
        '-1'
      ];
Entry: color-64,737.png
$VAR1 = [
        '0.93481407861017',
        '0.345711139511172',
        '-0.999999999999977',
        '-0.999999985064081',
        '-1'
      ];
Entry: color-7,390.png
$VAR1 = [
        '-0.999853813787961',
        '-0.303503446288619',
        '-0.999984006201254',
        '-0.999999238570459',
        '-1'
      ];
```

1.3. Tag file of a flower

bellatulum12 phot

genus:Brachypetalum
species:bellatulum
Flower

1.4. Tag file of a *salep* tuber

TEH-1.1_5

Orchid_spur
Orchid
Spur
genus:Dactylorhiza
species:incarnata
Tuber

1.5. Tag file of a Look-a-Like tuber

tulipa red riding hood3

Look-a-Like_round
Look-a-Like
Round
genus:Tulipa
species:greigii
Tuber

2. Photographs of *salep* orchid look-a-likes [24]

Figure S-1 *Arum maculatum* [24]

Figure S-2 *Asparagus officinalis* [24]

Figure S-3 *Polygonatum verticillatum* [24]

Figure S-4 *Tulipa greigii* [24]

Figure S-5 *Tulipa sp.* [24]

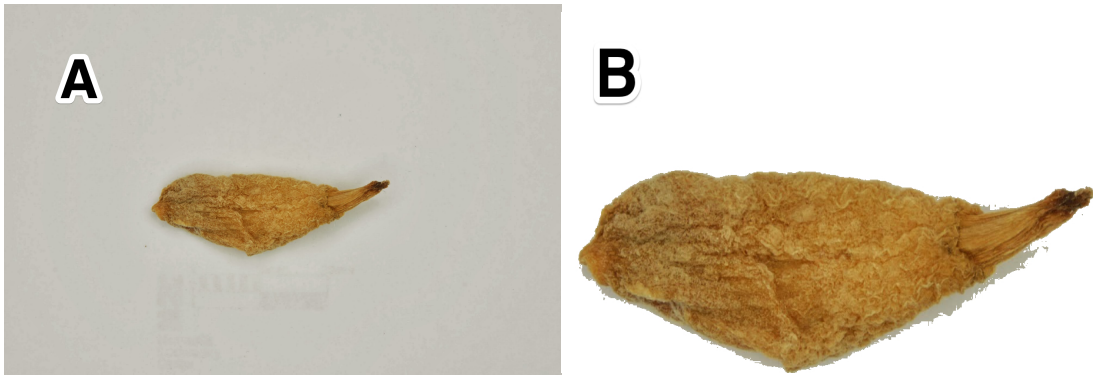

Figure S-6 A: A picture of a tuber, before segmenting. B: A picture of the same tuber after segmenting [24].

3. Screenshots of the classification website

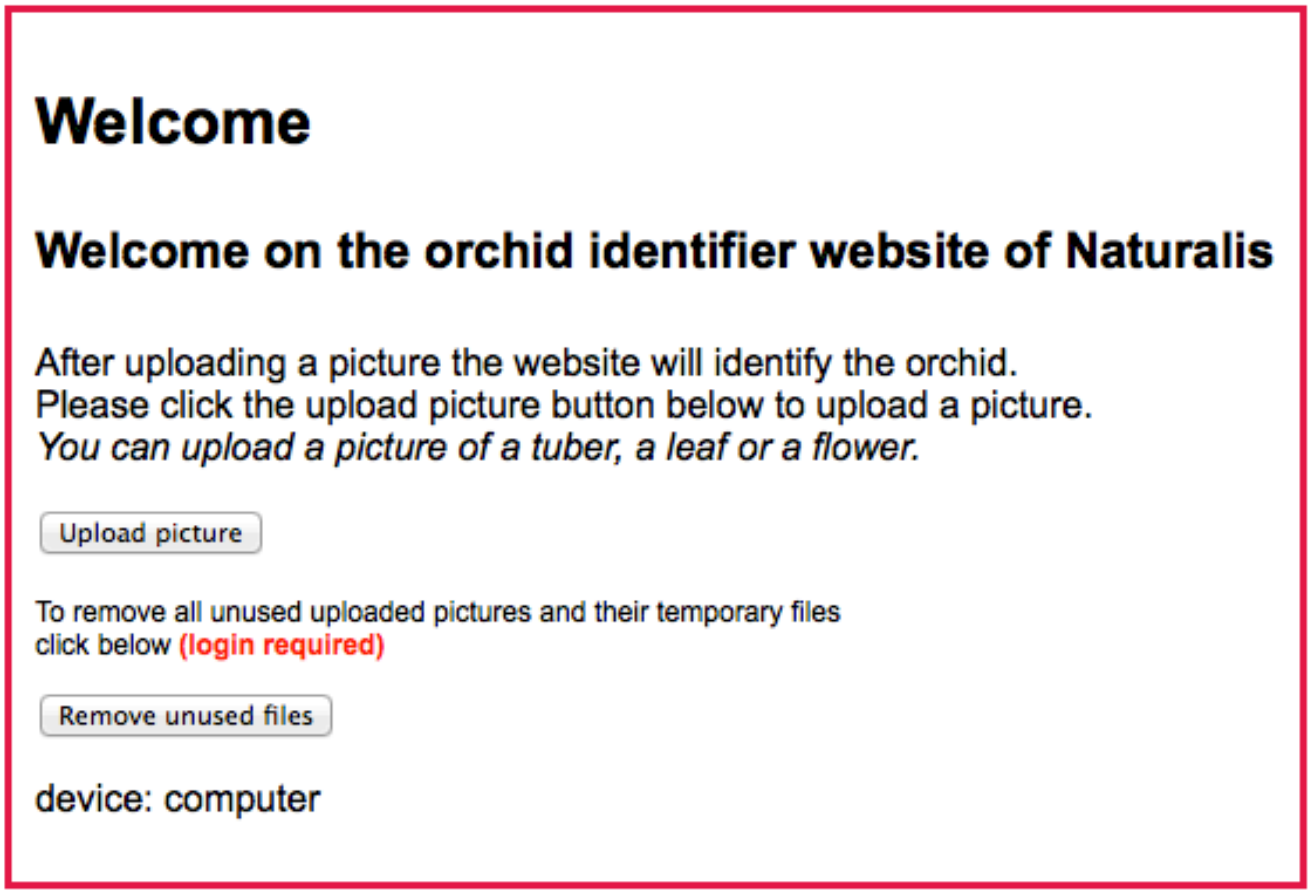

Figure S-7 Homepage of the classification website

Figure S-8 Upload page of the classification website

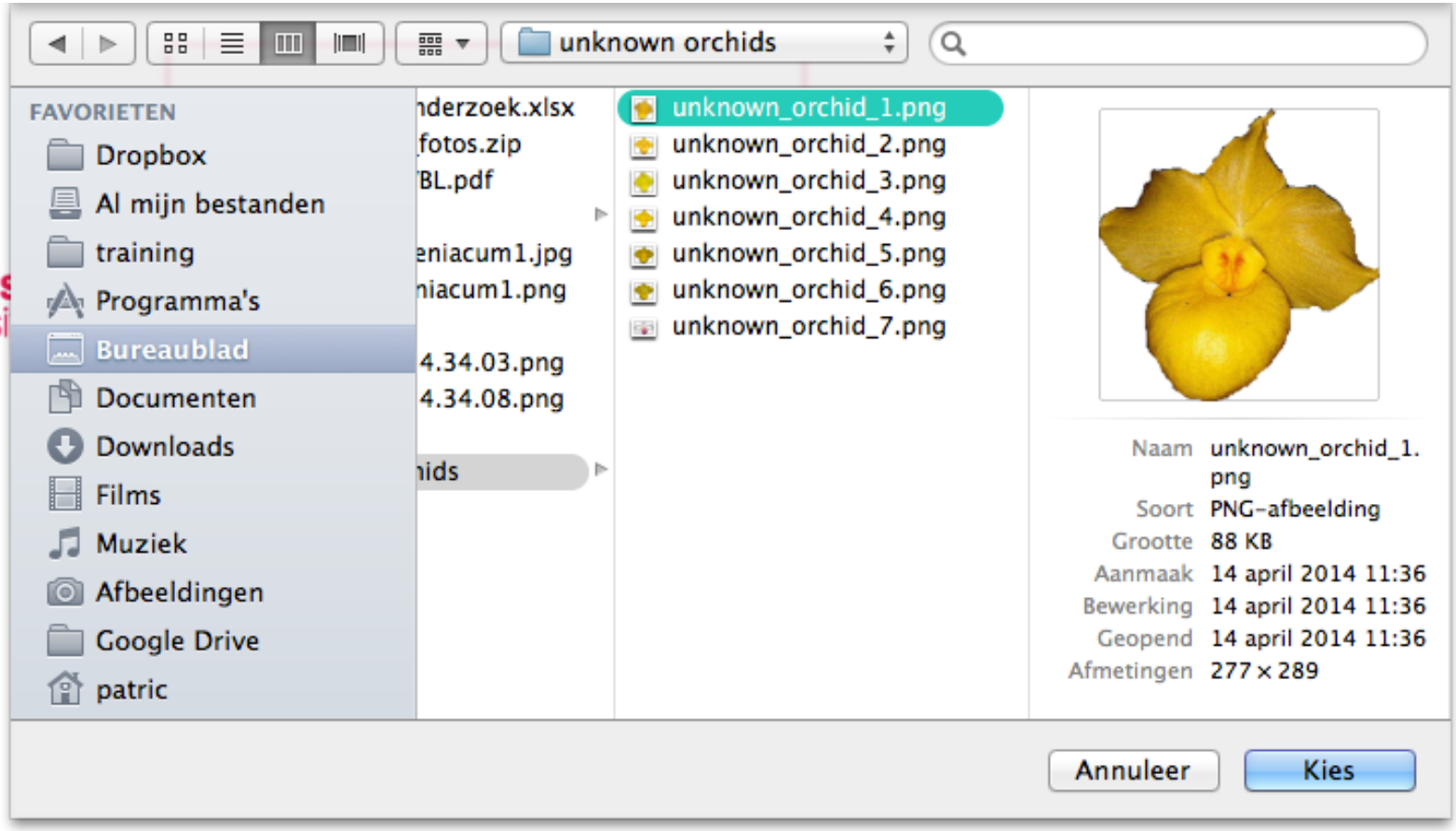

Figure S-9 Screen to select the picture to be classified

Naturalis Biodiversity Center

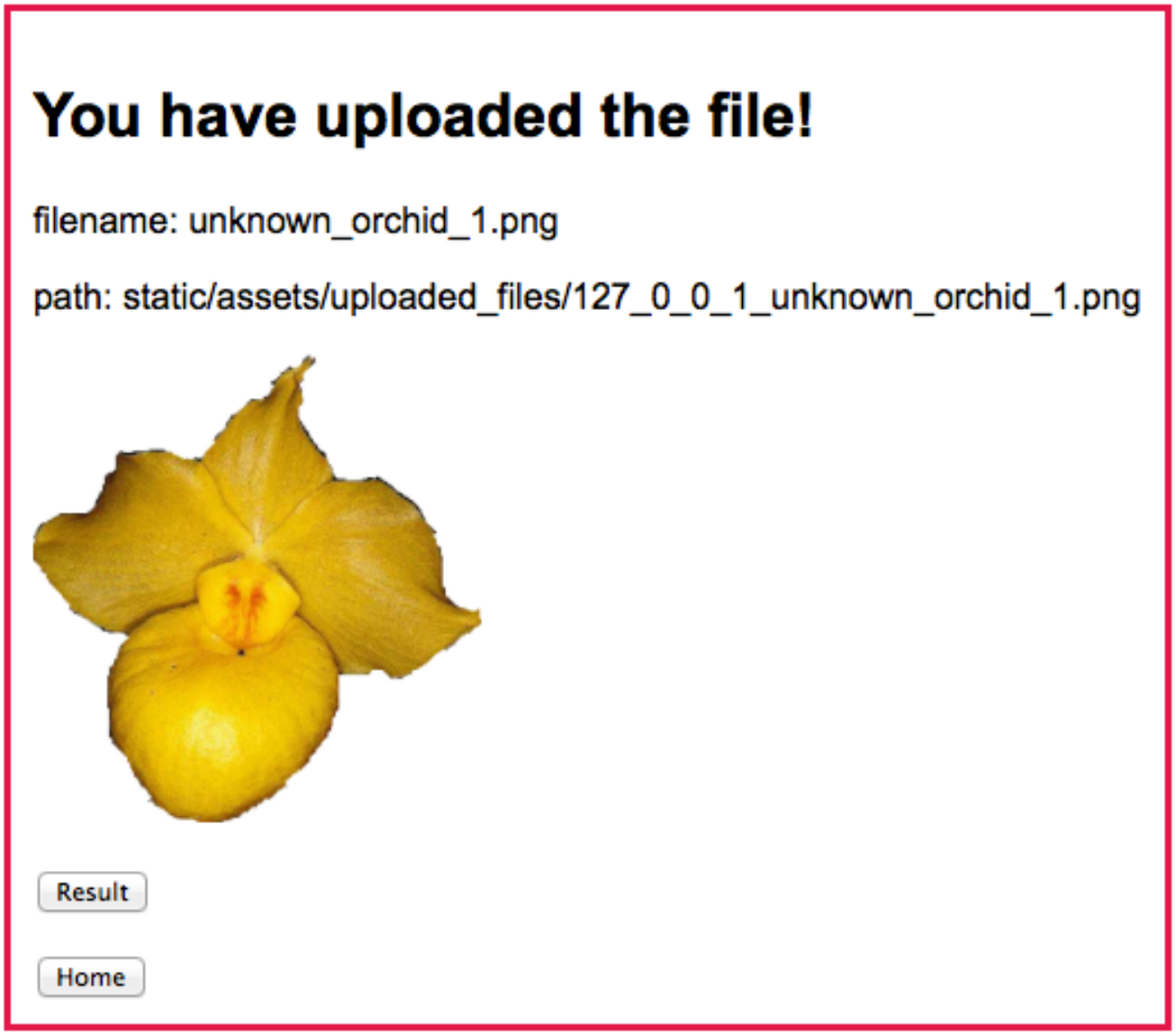

Figure S-10 page to check if the picture is uploaded correctly

Naturalis Biodiversity Center

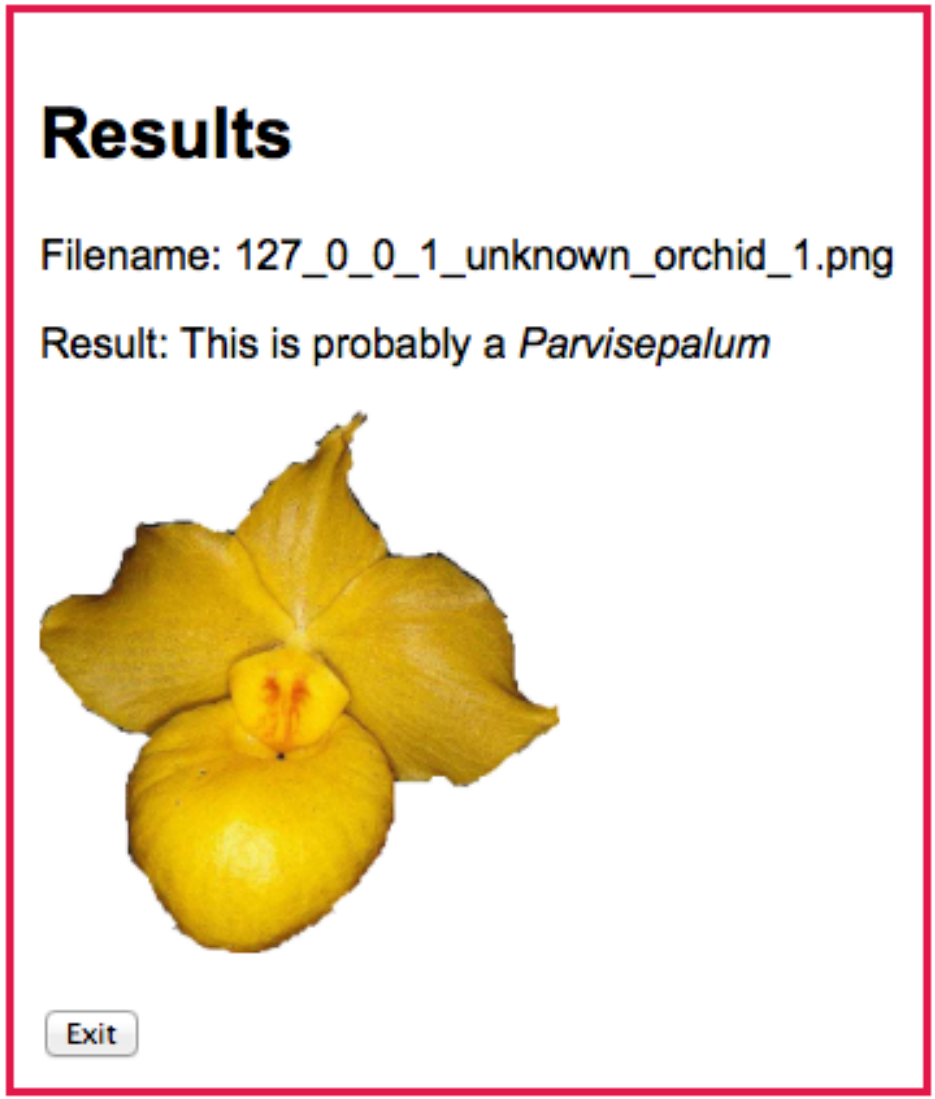

Figure S-11 Result page of the classification website

3. Results of first test run

Table 2 Results of first test of *Paphiopedilum* section *Barbata*. Red: incorrectly classified. Green: Correctly classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

	***Paphiopedilum* sect. *Barbata* run1_2**						
ANN	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	16	15	15	15	14	15	14
Pictures							
acmodontum1	Incorrect 1	Incorrect 1	Unknown 7/1	Incorrect 1	Incorrect 1	Incorrect 1	Incorrect 1
appletonianum1	Incorrect 3	Unknown	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 6	Incorrect 6
argus1	Unknown 7/3	Unknown	Correct	Unknown 7/3	Unknown 7/3	Correct	Unknown 3/7
argus2_2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
barbatum1	Incorrect 1 (low)	Unknown 7/3	Unknown 3/1(low)	Incorrect 3	Unknown 7/3	Incorrect 3	Unknown
barbatum2_2	Incorrect 3	Incorrect 3(low)	Unknown	Unknown	Incorrect 3	Unknown	Incorrect 3
bougainvilleanum1*	Unknown 7/1(low)	Incorrect 3	Unknown	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3
bullenianum1	Unknown 3/1(low)	Incorrect 3	Incorrect 1	Unknown 1/3	Incorrect 3	Incorrect 1	Unknown
callosum1	Incorrect 6	Unknown 6/3(low)	Unknown	Incorrect 1(low)	Incorrect 1	Incorrect 6	Incorrect 6
callosum2	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 1	Unknown	Incorrect 3
callosum3	Unknown	Incorrect 3	Incorrect 2	Unknown	Incorrect 3	Unknown	Correct
dayanum1	Unknown	Unknown	Unknown	Unknown	Correct	Incorrect 6	Unknown
dayanum2_3	Unknown 7/2	Correct	Unknown 7/3	Unknown	Incorrect 6	Correct	Unknown
dayanum2	Incorrect 3	Incorrect 3	Unknown	Unknown 3/1(low)	Unknown	Incorrect 1	Unknown
fowliei1*	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Unknown	Correct	Correct
fowliei2*	Unknown	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3
hennisianum1	Unknown 3/1	Incorrect 1	Incorrect 3	Incorrect 1	Incorrect 1	Incorrect 3	Unknown 1/7
hennisianum2_1	Unknown 3/7(low)	Incorrect 3	Correct	Unknown 7/3(low)	Unknown 7/3	Correct	Unknown
hennisianum2	Incorrect 3	Correct(low)	Incorrect 3	Unknown	Incorrect 3	Incorrect 5	Unknown
hookerae1	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3(low)	Incorrect 3	Incorrect 3
hookerae2_4	Correct	Correct	Incorrect 3	Correct	Correct	Correct	Correct
javanicum1*	Correct	Correct	Correct	Unknown 1/2/7(low)	Correct	Correct	Correct
javanicum2*	Unknown 7/1	Unknown	Unknown 7/1	Incorrect 1	Unknown	Correct	Unknown
lawrenceanum1	Correct	Unknown	Unknown	Incorrect 5	Unknown	Unknown	Unknown

mastersianum1	Incorrect 3	Incorrect 3	Unknown 3/1(low)	Incorrect 3	Incorrect 3	Unknown 3/7	Correct
mastersianum2_5	Correct	Correct	Correct	Correct	Correct	Correct	Incorrect 5
papuanum1*	Correct	Incorrect 1	Incorrect 1	Incorrect 1	Correct	Unknown	Correct
purpuratum1	Incorrect 1	Incorrect 1	Incorrect 6	Incorrect 1	Unknown 1/3	Incorrect 1(low)	Unknown 1/3
purpuratum2_1	Unknown 7/3	Correct	Unknown 3/7	Correct	Correct	Unknown 7/3	Correct
purpuratum2	Incorrect 1	Incorrect 1	Unknown 1/7	Incorrect 1	Incorrect 1	Incorrect 1	Unknown 1/7
schoseri1*	Incorrect 1	Unknown	Unknown	Unknown	Unknown	Incorrect 6(low)	Unknown
sukhakulii1	Incorrect 1	Unknown	Incorrect 1	Correct	Unknown	Incorrect 3	Unknown
superbiens1§	Unknown 3/7	Correct	Incorrect 3	Correct	Incorrect 3(low)	Unknown 7/3	Incorrect 3
tonsum1	Correct	Unknown 7/1	Correct	Correct	Incorrect 1	Correct	Correct
tonsum2_2	Correct	Correct	Correct	Correct	Unknown	Incorrect 3	Incorrect 3
tonsum2	Correct	Correct	Unknown 3/7	Correct	Unknown	Correct	Incorrect 3
tonsum3	Correct	Correct	Unknown	Unknown	Unknown	Correct	Unknown
urbanianum1	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3(low)	Unknown 1/7	Correct	Unknown 3/7
urbanianum2_3	Unknown	Correct	Correct	Unknown	Correct	Correct	Unknown
violascens1	Incorrect 1	Unknown 7/1	Incorrect 3	Correct	Incorrect 3	Incorrect 3	Unknown 3/1
violascens2	Unknown	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Unknown 3/7	Incorrect 3
wardii1	Incorrect 1 (low)	Unknown 3/1	Unknown 7/1	Incorrect 1	Incorrect 1	Incorrect 1	Unknown
wardii2_1	Correct	Correct	Unknown 7/1	Correct	Unknown 5/1	Unknown 7/5	Incorrect 5
wentworthianum1*	Incorrect 1	Unknown	Incorrect 4	Incorrect 1	Incorrect 1	Incorrect 1(low)	Unknown
wentworthianum2*	Unknown	Incorrect 3	Incorrect 4	Unknown	Unknown	Unknown	Unknown
Pictures	**41**	**42**	**43**	**44**	**45**	**46**	**47**

1:*Parvisepalum* 2:*Pardalopetalum* 3:*Paphiopedilum* 4:*Coryopedilum* 5: *Cochlopetalum* 6:*Brachypetalum* 7:*Barbata*

*= not in train set / on dear.smartweb.tw

§=only var. Curtisii in train set

The order in Unknown x/y is <best_sore>/<lower_score>. No numbers? All scores are negative

Table 3 Results of first test of extra pictures of *Paphiopedilum* section *Barbata*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

			***Paphiopedilum* sect. *Barbata* run1_2**				
ANN	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	2	2	2	2	2	1	2
Picture							
paph_argus	Correct	Unknown	Correct	Unknown 2/7	Correct	Correct	Correct
Paptonsum_blaine	Correct	Unknown	Unknown	Correct	Unknown	Unknown	Unknown
Paphtonsum_elle	Unknown	Incorrect 1	Incorrect 3	Unknown	Incorrect 3	Incorrect 1	Unknown
Paphwardii	Incorrect 3	Incorrect 3	Correct	Incorrect 3	Incorrect 3	Incorrect 3	Unknown

Legend Incorrect/Unknown

1 *Parvisepalum* 2 *Pardalopetalum*
3 *Paphiopedilum* 4 *Coryopedilum*
5 *Cochlopetalum* 6 *Brachypetalum*
7 *Barbata*

The order in Unknown x/y is <best_score>/<lower_score>. No numbers? All scores are negative!

Table 4 Results of first test of *Paphiopedilum* section *Brachypetalum*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

	***Paphiopedilum* sect. *Brachypetalum* run1**						
ANN	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	8	7	7	7	7	7	8
Pictures							
bellatum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
concolor1	Correct	Unknown 6/1	Correct	Correct	Unknown	Correct	Correct
concolor2_2	Correct	Correct	Unknown 6/1	Correct	Unknown 6/1	Correct	Correct
concolor2_4	Correct	Correct	Correct	Correct	Correct	Unknown 6/3	Correct
concolor2_6	Correct	Unknown 6/1	Unknown 6/1(low)	Correct	Unknown 6/1	Correct	Correct
concolor2	Correct	Correct	Correct	Correct	Unknown 6/3	Unknown	Correct
godefroyae1	Unknown 6/1(low)	Unknown	Unknown 6/7	Unknown 6/1	Unknown 7/6	Correct	Correct
godefroyae2_11	Unknown 6/3	Correct	Unknown 6/3	Correct	Correct	Correct	Correct
godefroyae2_13	Correct	Unknown	Unknown 1/3	Correct	Correct	Correct	Correct(low)
godefroyae2_15	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_3	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_5	Correct	Correct	Correct	Correct	Correct	Correct	Unknown 6/3
godefroyae2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_9	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
niveum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
niveum2_4	Unknown 6/2(low)	Correct	Correct	Correct	Correct	Correct	Correct
niveum2_5	Correct	Correct	Correct	Correct	Correct	Unknown	Correct
niveum2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct
			1:*Parvisepalum*	2:*Pardalopetalum*	3:*Paphiopedilum*	4:*Coryopedilum*	5: *Cochlopetalum*
			6:*Brachypetalum*	7:*Barbata*			
			The order in Unknown x/y is <best_score>/<lower_score>. No numbers? All scores are negative!				

Table 5 Results of first test of *Paphiopedilum* section *Coryopedilum*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

	***Paphiopedilum* sect. *Coryopedilum* run1**						
ANN	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	4	3	3	3	3	3	3
Pictures							
glanduliferum3	Incorrect 3	Unknown	Unknown	Incorrect 3	Incorrect 3	Unknown	Unknown
rothschildianium1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
wilhelminiae1	Unknown	Unknown 4/3	Unknown	Unknown	Correct	Correct	Correct
philippinense1	Correct	Correct	Unknown	Incorrect 1	Unknown	Unknown	Unknown
rothschildianium2	Unknown 4/5	Unknown	Incorrect 1	Correct	Correct	Correct	Correct
rothschildianium3	Correct	Correct	Incorrect 1(low)	Incorrect 1(low)	Unknown	Unknown 3/4	Correct

Legend Incorrect/Unknown

1 *Parvisepalum*	2 *Pardalopetalum*
3 *Paphiopedilum*	4 *Coryopedilum*
5 *Cochlopetalum*	6 *Brachypetalum*
7 *Barbata*	

Table 6 Results of first test of *Paphiopedilum* section *Parvisepalum*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

	***Paphiopedilum* sect. *Parvisepalum* run1**						
ANN	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	15	15	15	15	14	15	15
Pictures							
armeniacum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_4	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_5	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_6	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_8	Correct	Correct	Correct	Correct	Correct	Correct	Correct
delenatii1	Correct	Unknown	Unknown	Unknown	Correct	Correct	Correct(low)
delenatii2_2	Correct	Correct(low)	Correct	Correct	Correct	Correct	Unknown 1/7
delenatii2_4	Correct	Correct	Correct	Correct	Correct	Correct	Correct
delenatii2_6	Correct	Correct	Correct	Correct	Correct	Correct	Correct
delenatii2	Correct	Correct	Correct	Correct	Unknown 7/1	Correct	Correct
emersonii1	Unknown 3/1	Unknown	Correct	Incorrect 3(low)	Correct	Correct	Unknown
emersonii2_1	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Unknown	Incorrect 3
emersonii2_3	Correct	Correct	Correct	Correct	Correct	Correct	Correct
emersonii2_5	Correct	Correct	Unknown 1/3	Correct(low)	Correct	Correct	Correct
malipoense1	Correct	Correct	Unknown 3/1	Unknown	Incorrect 3(low)	Correct	Incorrect 3
malipoense2_10	Correct	Correct	Correct	Correct	Unknown 1/7	Correct	Correct
malipoense2_12	Correct	Correct	Unknown 3/1	Correct	Correct	Correct	Correct
malipoense2_14	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_16	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_18	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_2	Correct	Correct	Correct	Correct	Correct	Unknown 1/3	Correct

malipoense2_4	Unknown 1/7	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_6	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_8	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2	Unknown	Unknown	Unknown	Correct	Correct	Unknown	Correct
malipoense3	Correct	Unknown	Incorrect 7	Unknown	Correct	Unknown	Correct
micranthum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_11	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_13	Correct	Correct	Unknown	Correct	Correct	Unknown	Correct
micranthum2_15	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_17	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_19	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_1	Correct	Correct	Correct	Correct	Correct	Correct	Correct(low)
micranthum2_21	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_3	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_5	Correct	Correct	Incorrect 3	Correct	Incorrect 3	Unknown	Unknown
micranthum2_7	Correct (very low)	Correct	Unknown 7/1(low)	Correct	Incorrect 7	Incorrect 5	Unknown
micranthum2_9	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2	Correct	Unknown 1/6	Correct	Correct	Correct	Correct	Correct
micranthum3	Correct	Correct	Correct	Correct	Correct	Correct	Correct
Picture	**41**	**42**	**43**	**44**	**45**	**46**	**47**

1:*Parvisepalum* 2:*Pardalopetalum* 3:*Paphiopedilum* 4:*Coryopedilum* 5: *Cochlopetalum*
6:*Brachypetalum* 7:*Barbata*
The order in Un-known x/y is <best_sore>/<lower_score>. No numbers? All scores are negative

Table 7 Results of first test of hybrids of *Paphiopedilum* sect. *Parvisepalum* and *Brachypetalum*. Red: incorrect classified. Green: Correct classification to one of the parental sections. Green and bold: Correct classification to both parental sections. Yellow background and brown text: Unable to classify.

			***Paphiopedilum* sect. *Parvisepalum* X *Brachypetalum* run1**				
ANN	**41**	**42**	**43**	**44**	**45**	**46**	**47**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	7	5	5	5	5	4	5
Pictures							
armeniacumXconcolor2	Unknown	1	1	1	1	1	1
delenatiiXniveum2	1	Unknown	Unknown	6	Unknown	1	Unknown
armeniacumXconcolor1	1	1	1	1	1	1	1
armeniacumXconcolor6	6	***1&6***	1	1	1	1	1
emersoniiXbellatulum	3&1	1	1&7	Unknown	Unknown	3	1
delenatiiXniveum1	6	6	6	Unknown	6(low)	6	6
armeniacumXconcolor5	1	1	1	1	1	1	1
armeniacumXconcolor4	1	1	1	1	1	1	1

Legend

1	*Parvisepalum*	2	*Pardalopetalum*
3	*Paphiopedilum*	4	*Coryopedilum*
5	*Cochlopetalum*	6	*Brachypetalum*
7	*Barbata*		

4. Results of characteristic research

Table 8 Characteristics *of Paphiopedilum* section *Barbata*

Paphiopedilum sect. *Barbata*

Characteristics	argus1*	argus2_2§	mastersianum1*	mastersianum2_5§	tonsum1*$	tonsum2_2§
spathulate petals	!	✓	✓	✓	!	✓
spotted and/or warted petals	✓	✓	×	✓	✓	✓
incurved side lobes	✓	✓	✓	✓	✓	✓
Note	artificial white patches caused by scanning		artificial white patches caused by scanning		artificial white patches caused by scanning + Small pic.	artificial white patches caused by scanning

Characteristics	tonsum2*	tonsum3*	wardii1*	wardii2_1§
spathulate petals	✓	✓	×	✓
spotted and/or warted petals	✓	✓	✓	✓
incurved side lobes	✓	✓	✓	✓
Note	artificial white patches caused by scanning	artificial white patches caused by scanning	artificial white patches caused by scanning	

✓	Visible
!	Difficult to see
×	Invisible
*	Picture scanned from Cribb, 1998
§	Picture from dear.smartweb.tw
Name	Incorrectly identified to sectional level in max 1 of the 7 different networks
Name	Incorrectly identified to sectional level in max 3 of the 7 different networks
Name	Incorrectly identified to sectional level in more than 3 of the 7 networks

Table 9 Characteristics of *Paphiopedilum* section *Brachypetalum*.

Paphiopedilum sect. *Brachypetalum*

Characteristics	concolor1*	concolor2_2§	concolor2_4§	concolor2_6§	concolor2*
variously ornamented with purple spots	✓	✓	✓	✓	✓
broad-elliptic petals	✓	✓	✓	✓	!
small lip	✓	✓	✓	✓	✓
ovoid lip	✓	✓	✓	×	✓
lip with incurved margin	✓	×	✓	✓	✓
pale yellow or white flower	×	✓	✓	✓	✓

✓	Visible
!	Difficult to see
×	Invisible
*	Picture scanned from Cribb, 1998
§	Picture from dear.smartweb.tw
Name	Incorrectly identified to sectional level in max 1 of the 7 different networks
Name	Incorrectly identified to sectional level in max 3 of the 7 different networks
Name	Incorrectly identified to sectional level in more than 3 of the 7 networks

Table 10 Characteristics of *Paphiopedilum* section *Coryopedilum*.

Paphiopedilum sect. *Coryopedilum*

Characteristics	rothschildianium1*	rothschildianium2*	rothschildianium3*
Long petals	✓	✓	✓
Petals hanging down	✓	✓	✓
Tapering petals	✓	✓	✓
Spirally twisted petals	✓	✓	!
Warts on margins of petals	×	×	×
Glandular at tip	×	×	×
incurved side lobes	✓	×	×

✓	Visible
!	Difficult to see
×	Invisible
*	Picture scanned from Cribb, 1998
§	Picture from dear.smartweb.tw
Name	Incorrectly identified to sectional level in max 1 of the 7 different networks
Name	Incorrectly identified to sectional level in max 3 of the 7 different networks
Name	Incorrectly identified to sectional level in more than 3 of the 7 networks

Table 11 Characteristics of *Paphiopedilum* section *Parvisepalum*.

Paphiopedilum sect. *Parvisepalum*

Characteristics	delenatii1*	delenatii2_2§	delenatii2_4§	delenatii2_6§	delenatii2*
broad-elliptic petals	✓	✓	✓	✓	✓
subcircular petals	✓	✓	✓	✓	!
large staminode	✓	✓	✓	✓	✓
cream/yellow coloured petals	×	✓	✓	✓	×
Note	artificial white patches caused by scanning				artificial brown patches caused by scanning

Characteristics	emersonii1*	emersonii2_1§	emersonii2_3§	emersonii2_5§	malipoense1*
broad-elliptic petals	✓	✓	✓	✓	✓
subcircular petals	✓	✓	✓	✓	×
large staminode	✓	✓	✓	✓	✓
cream/yellow coloured petals	×	×	✓	×	✓
Note	artificial white patches caused by scanning	Flower dark		Flower grey/ blue like	

Characteristics	malipoense2_10§	malipoense2_12§	malipoense2_14§	malipoense2_8§	malipoense2*
broad-elliptic petals	✓	✓	✓	✓	!
subcircular petals	✓	✓	✓	✓	×
large staminode	✓	✓	✓	✓	✓
cream/yellow coloured petals	✓	✓	✓	✓	×
Note					Not fully frontal

Characteristics	malipoense3*	malipoense2_16§	malipoense2_18§	malipoense2_2§	malipoense2_4§
broad-elliptic petals	!	✓	✓	✓	✓
subcircular petals	✗	✓	✓	✓	✓
large staminode	✓	✓	✓	✓	✓
cream/yellow coloured petals	✗	✓	✓	✓	✓
Note	Not fully frontal				

Characteristics	malipoense2_6§
broad-elliptic petals	✓
subcircular petals	✓
large staminode	✓
cream/yellow coloured petals	✓

✓	Visible
!	Difficult to see
✗	Invisible
*	Picture scanned from Cribb, 1998
§	Picture from dear.smartweb.tw
Name	Incorrectly identified to sectional level in max 1 of the 7 different networks
Name	Incorrectly identified to sectional level in max 3 of the 7 different networks
Name	Incorrectly identified to sectional level in more than 3 of the 7 networks

5. Results of second test run

Table 12 Results of second test run of *Paphiopedilum* section *Barbata*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

	***Paphiopedilum* sect. *Barbata* run2_1**						
ANN	**51**	**52**	**53**	**54**	**55**	**56**	**57**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	17	15	14	15	16	14	14
Pictures							
acmodontum1	Unknown 1/2	Incorrect 1	Incorrect 1	Incorrect 1	Unknown 7/1	Incorrect 1	Incorrect 1
appletonianum1	Incorrect 1	Unknown	Incorrect 3(low)	Incorrect 6	Unknown	Unknown	Incorrect 1
argus1	Unknown 7/3	Correct	Correct	Unknown	Incorrect 3	Correct	Unknown 7/3
argus2_2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
barbatum1	Unknown	Unknown	Incorrect 3	Incorrect 3	Incorrect 1	Unknown 3/7	Incorrect 3
barbatum2_2	Unknown	Incorrect 5	Unknown 7/3(low)	Unknown	Incorrect 4	Incorrect 5	Unknown
bougainvilleanum1*	Unknown 7/3	Incorrect 3	Correct	Correct	Unknown 3/7	Unknown 7/3	Unknown 7/3
bullenianum1	Incorrect 1	Incorrect 1(low)	Unknown	Unknown	Incorrect 1	Incorrect 3	Unknown
callosum1	Unknown 6/3	Incorrect 3	Incorrect 6	Unknown 3/6	Incorrect 6	Incorrect 6	Unknown
callosum2	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3
callosum3	Correct	Unknown	Unknown	Correct	Correct	Correct	Correct
dayanum1	Unknown	Unknown	Unknown	Unknown	Incorrect 6	Incorrect 6(low)	Unknown
dayanum2_3	Incorrect 4(low)	Unknown 1/7	Unknown	Unknown	Unknown	Unknown 4/7	Unknown
dayanum2	Incorrect 3(low)	Incorrect 3	Unknown 5/4	Unknown	Unknown	Unknown	Incorrect 5
fowliei1*	Unknown low:3/7	Unknown 3/7(low)	Correct	Incorrect 3	Incorrect 3(low)	Incorrect 3	Unknown
fowliei2*	Incorrect 3	Incorrect 3	Incorrect 3(low)	Unknown	Incorrect 3	Incorrect 3(low)	Incorrect 3
hennisianum1	Unknown	Incorrect 1(low)	Unknown	Unknown	Incorrect 1	Incorrect 1	Incorrect 1
hennisianum2_1	Correct	Incorrect 3	Correct	Correct	Unknown 7/3	Correct	Correct
hennisianum2	Incorrect 3(low)	Unknown	Unknown	Unknown	Unknown	Unknown	Incorrect 3
hookerae1	Incorrect 3	Incorrect 3	Unknown 3/7	Incorrect 3	Incorrect 3	Unknown	Incorrect 3
hookerae2_4	Correct	Correct	Correct	Correct	Correct	Correct	Correct
javanicum1*	Correct	Correct	Correct	Correct	Correct	Unknown 7/3	Correct

javanicum2*	Unknown	Incorrect 1	Correct	Correct(low)	Unknown 7/1	Unknown 7/1	Incorrect 1(low)
lawrenceanum1	Incorrect 4	Unknown	Correct	Unknown	Unknown	Unknown	Unknown
mastersianum1	Unknown	Correct	Unknown 7/3(low)	Correct	Correct	Correct	Correct(low)
mastersianum2_5	Correct	Correct	Correct	Correct	Correct	Correct	Correct
papuanum1*	Unknown 4/7/1(L)	Unknown	Unknown 7/6/1	Unknown	Correct	Unknown 3/7(low)	Incorrect 5
purpuratum1	Incorrect 1	Incorrect 1	Incorrect 1	Unknown 1/3(low)	Unknown	Incorrect 1	Incorrect 1
purpuratum2_1	Unknown 7/3	Unknown 7/5	Correct	Correct	Correct	Correct	Correct
purpuratum2	Incorrect 1	Unknown 7/1	Unknown 1/7	Unknown 1/7	Unknown 7/1	Incorrect 1	Incorrect 1
schoseri1*	Unknown	Unknown	Unknown	Unknown	Incorrect 6	Unknown	Incorrect 1
sukhakulii1	Incorrect 1	Unknown 1/7(low)	Incorrect 3	Incorrect 1	Incorrect 4	Incorrect 1	Unknown
superbiens1§	Unknown	Unknown 7/3	Correct	Correct(low)	Unknown 3/7(L)	Unknown	Correct
tonsum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
tonsum2_2	Correct	Correct	Incorrect 5	Correct	Correct	Correct	Correct
tonsum2	Unknown	Correct	Correct	Unknown	Correct	Correct	Correct
tonsum3	Correct	Incorrect 3	Correct	Correct	Unknown	Incorrect 3	Unknown
urbanianum1	Correct	Unknown 7/1(low)	Incorrect 3	Unknown	Unknown	Unknown	Incorrect 3(low)
urbanianum2_3	Correct(low)	Correct	Correct	Correct(low)	Correct	Correct(low)	Correct
violascens1	Unknown 3/7(low)	Incorrect 3	Incorrect 3	Unknown	Correct(low)	Unknown	Incorrect 3
violascens2	Unknown 7/3	Incorrect 3(low)	Incorrect 3(low)	Correct	Unknown 3/7	Unknown	Incorrect 3
wardii1	Incorrect 1	Unknown 1/3(low)	Incorrect 1	Incorrect 1	Unknown 3/1	Incorrect 1	Unknown
wardii2_1	Correct	Correct	Correct	Correct	Correct	Correct	Unknown 1/7(low)
wentworthianum1*	Correct	Unknown	Unknown	Correct(low)	Unknown	Unknown	Unknown
wentworthianum2*	Unknown	Unknown	Correct	Unknown	Unknown	Unknown	Unknown
Pictures	**51**	**52**	**53**	**54**	**55**	**56**	**57**
			1:*Parvisepalum*	2:*Pardalopetalum*	3:*Paphiopedilum*	4:*Coryopedilum*	5: *Cochlopetalum*
	*= not in train set / on dear.smartweb.tw		6:*Brachypetalum*	7:*Barbata*	L:low		
	§= only var. Curtisii in train set.		The order in Unknown x/y is <best_sore>/<lower_score>. No numbers? All scores are negative				

Table 13 Results of second test of the extra pictures of *Paphiopedilum* section *Barbata*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

	***Paphiopedilum* sect. *Barbata* run2_2**						
ANN	**51**	**52**	**53**	**54**	**55**	**56**	**57**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	2	2	2	2	2	2	2
paph_argus	Correct	Correct	Correct	Correct	Correct	Correct	Correct
Paptonsum_blaine	Incorrect 1(low)	Unknown	Unknown	Correct(low)	Correct	Correct	Correct
Paphtonsum_elle	Incorrect 1	Incorrect 3(low)	Unknown	Unknown	Unknown	Incorrect 1	Incorrect 1(low)
Paphwardii	Incorrect 3	Incorrect 3	Unknown 3/6	Incorrect 3	Unknown 3/7	Unknown	Incorrect 3

Legend Incorrect/Unknown

1 *Parvisepalum* 2 *Pardalopetalum*
3 *Paphiopedilum* 4 *Coryopedilum*
5 *Cochlopetalum* 6 *Brachypetalum*
7 *Barbata*

The order in Unknown x/y is <best_score>/<lower_score>/ No numbers? All scores are negative

Table 14 Results of second test run of *Paphiopedilum* section *Brachypetalum*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

		***Paphiopedilum* sect. *Brachypetalum* run2**					
ANN	**51**	**52**	**53**	**54**	**55**	**56**	**57**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	7	7	8	7	7	6	7
Pictures							
bellatum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
concolor1	Correct	Correct	Correct	Unknown	Correct	Correct	Correct
concolor2_2	Correct	Unknown 6/1	Unknown 6/1	Correct	Unknown 6/1	Unknown 6/1	Correct
concolor2_4	Correct	Correct	Unknown 6/1	Unknown 6/1	Correct	Unknown 6/1	Correct
concolor2_6	Correct	Correct	Unknown 6/1	Correct	Unknown 6/1	Unknown 6/1	Correct
concolor2	Correct	Correct	Correct	Correct	Unknown	Correct	Correct
godefroyae1	Unknown	Unknown	Unknown 6/7	Correct	Unknown 6/1	Incorrect 1	Incorrect 1
godefroyae2_11	Correct	Unknown	Correct	Correct	Correct	Correct	Correct
godefroyae2_13	Correct	Unknown	Correct	Correct	Correct	Correct	Correct
godefroyae2_15	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_3	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_5	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2_9	Correct	Correct	Correct	Correct	Correct	Correct	Correct
godefroyae2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
niveum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
niveum2_4	Unknown	Correct	Correct	Correct	Correct	Unknown 6/1	Correct
niveum2_5	Correct	Correct	Unknown 6/3(low)	Unknown	Correct	Correct	Correct
niveum2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct

1:Parvisepalum 2:Pardalopetalum 3:Paphiopedilum 4:Coryopedilum
5: Cochlopetalum 6: Brachypetalum 7:Barbata
The order in Unknown x/y is <best_sore>/<lower_score>. No numbers? All scores are negative!

Table 15 Results of second test of *Paphiopedilum* section *Coryopedilum*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

			***Paphiopedilum* sect. *Coryopedilum* run2**				
ANN	**51**	**52**	**53**	**54**	**55**	**56**	**57**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	3	3	3	2	2	3	2
Picture							
glanduliferum3	Correct	Correct	Unknown	Correct	Unknown 4/3	Unknown	Correct
rothschildianium1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
wilhelminiae1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
philippinense1	Correct	Unknown 4/1	Correct	Correct	Unknown 4/6	Correct	Correct
rothschildianium2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
rothschildianium3	Correct	Correct	Correct	Unknown	Correct	Correct	Correct

Legend Incorrect/Unknown

1 *Parvisepalum* 2 *Pardalopetalum*
3 *Paphiopedilum* 4 *Coryopedilum*
5 *Cochlopetalum* 6 *Brachypetalum*
7 *Barbata*

The order in Unknown x/y is <best_score>/<lower_score>/ No numbers? All scores are negative

Table 16 Results of second test of *Paphiopedilum* section *Parvisepalum*. Red: incorrect classified. Green: Correct classified. Yellow background and brown text: Unable to classify (to one section). Yellow background and green text: Unable to classify to one section, but the correct one is in the options.

		***Paphiopedilum* sect. *Parvisepalum* run2**					
ANN	**51**	**52**	**53**	**54**	**55**	**56**	**57**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	15	14	14	14	15	14	14
Picture							
armeniacum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_4	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_5	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_6	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct
armeniacum2_8	Correct	Correct	Correct	Correct	Correct	Correct	Correct
delenatii1	Unknown	Correct	Unknown	Correct	Correct	Correct	Unknown
delenatii2_2	Correct	Correct	Correct	Correct	Unknown	Correct	Unknown
delenatii2_4	Correct	Correct	Correct	Correct	Correct	Correct	Correct
delenatii2_6	Correct	Correct	Correct	Correct	Correct	Correct	Correct
delenatii2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
emersonii1	Unknown low:1/3	Correct	Correct	Correct	Incorrect 3	Incorrect 3	Unknown
emersonii2_1	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3	Incorrect 3
emersonii2_3	Unknown 1/3(low)	Correct	Correct	Unknown	Unknown 3/1	Unknown 1/3	Correct
emersonii2_5	Unknown	Correct	Unknown	Unknown	Unknown	Unknown	Correct
malipoense1	Correct	Unknown 1/3	Unknown	Correct	Unknown 3/1(low)	Correct	Correct
malipoense2_10	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_12	Correct	Correct	Correct	Correct	Incorrect 3	Correct	Correct
malipoense2_14	Correct	Correct	Correct	Unknown	Correct	Correct	Correct
malipoense2_16	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_18	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_2	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_4	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2_6	Correct	Correct	Correct	Correct	Correct	Correct	Correct

malipoense2_8	Correct	Correct	Correct	Correct	Correct	Correct	Correct
malipoense2	Correct	Correct	Correct	Unknown	Correct	Correct	Correct
malipoense3	Unknown	Unknown	Incorrect 7	Unknown	Unknown	Incorrect 7	Incorrect 7
micranthum1	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_11	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_13	Unknown	Unknown	Unknown	Correct	Unknown	Unknown	Unknown
micranthum2_15	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_17	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_19	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_1	Correct(low)	Incorrect 5	Incorrect 6	Correct	Correct	Correct	Unknown
micranthum2_21	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_3	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2_5	Unknown	Incorrect 7	Unknown	Correct	Unknown	Unknown	Unknown 3/1
micranthum2_7	Correct	Correct	Correct	Correct	Correct	Correct	Correct(low)
micranthum2_9	Correct	Correct	Correct	Correct	Correct	Correct	Correct
micranthum2	Correct	Correct	Incorrect 4	Correct	Unknown	Correct	Correct
micranthum3	Correct	Correct	Unknown 1/4	Correct	Correct	Correct	Correct
Picture	**51**	**52**	**53**	**54**	**55**	**56**	**57**

1:*Parvisepalum* 2:*Pardalopetalum* 3:*Paphiopedilum* 4:*Coryopedilum* 5: *Cochlopetalum*
6:*Brachypetalum* 7:*Barbata*

The order in Un-known x/y is <best_sore>/<lower_score>. No numbers? All scores are negative!

Table 17 Results of second test of hybrids of *Paphiopedilum* sections *Parvisepalum* and *Brachypetalum*. Red: incorrect classified. Green: Correct classification to one of the parental sections. Green and bold: Correct classification to both parental sections. Yellow background and brown text: Unable to classify.

			***Paphiopedilum* sect. *Parvisepalum* X *Brachypetalum* run2**				
ANN	**51**	**52**	**53**	**54**	**55**	**56**	**57**
Desired Error	1,00E-04	1,00E-05	1,00E-06	1,00E-07	1,00E-08	1,00E-09	1,00E-10
Time to classify (s)	5	4	4	4	4	4	3
Picture							
armeniacumXconcolor2	1	1	1	1	1	1	1
delenatiiXniveum2	Unknown	Unknown	1	2(low)	Unknown	1	6
armeniacumXconcolor1	1	1	1	1	1	1	1
armeniacumXconcolor6	1	1	1	1	1	1	1(low)
emersoniiXbellatulum	1	1&3	1&3	1	Unknown	Unknown	Unknown
delenatiiXniveum1	6	6	6	6	6	6	6
armeniacumXconcolor5	1	1	1	1	1	1	1
armeniacumXconcolor4	1	1	1	1	1	1	1

Legend

1 *Parvisepalum* 2 *Pardalopetalum*
3 *Paphiopedilum* 4 *Coryopedilum*
5 *Cochlopetalum* 6 *Brachypetalum*
7 *Barbata*

Appendices

1. Codes

1.1. Bash

1.1.1. Create_traindata.sh

```
#!/usr/bin/env bash

#Loop throug the directories
for directories in $(ls -d */)
do
    directory=${directories%%/}
    #The tuber pictures use another Perl script for training the
neural network
    #than the flower pictures. So check the name of the directory.
    if [[ $directory == T* ]]
    #If it starts with T (=Tuber)
    then
        #Go into the directory
        cd $directory
        #Loop through the directories
        for directories2 in $(ls -d */)
        do
            #Create variables
            catagory=0
            shape=${directories2%%/}
            echo "run traindata.pl for $shape"
            #Give the Look-a-like tubers catagory -1
            if [[ $shape == L* ]]
            then
                catagory=-1
            #Give the salep tubers catagory 1
            else
                catagory=1
            fi

            #Run the traindata script and write the output to a tsv
file.
            #Give this file the name of the current directory
            perl ../traindata.pl -d ./$shape -c $catagory >
$shape.tsv
        done
    #After looping through the directories go out of the Tuber
directory
    cd ..
    elif [[ $directory == F* ]]
    #If it starts with F (=Flower)
    then
        #Go into the directory
        cd $directory
        #Loop through the section directories
```

```
        for section in $(ls -d */)
        do
            #Go into the section directory
            cd $section
            #Loop through the species directories
            for species_dir in $(ls -d */)
            do
                #Create variables
                species=${species_dir%%/}
                echo "run traindata2.pl for $species"
                #Run the traindata2 script and redirect the output
to a tsv file.
                #Give this file the name of the current directory
                perl ../../traindata2.pl -d ./$species -c 0 >
$species.tsv
            done
            #After loopging through the species directories go back
to the Flower directory
            cd ..
        done
    #After looping through the section directories go out of the
Flower directory
    cd ..
    fi
done

#Clear the screen
clear

#Print a finish message
echo "Done"
echo "Please modify the flower data:"
echo "Run: ./modify_flower_data.sh"
```

1.1.2. modify_flower_data.sh

```
#!/usr/bin/env bash

#Go into the Flower directory
cd Flower

#Loop through the directories
for directory in $(ls -d */)
do
    #Go into the directory
    cd $directory
    #Run combine_files.py
    echo "Run combine_files.py"
    python ../../combine_files.py
    #Go back to the Flower directory
    cd ..
done

#Run add_columns.py
echo "Run add_columns.py"
python ../add_columns.py

#Print a finising message
echo "Finished"

echo "To train a neural network use the folowing command:"
echo "perl trainai.pl -d <directory_with_traindata> -c
<number_of_categories> -o <output>"
echo "To change the Desired Error add -t <desired_error>"
echo "To change the number of epoch add -e <epochs>"
echo "To follow the run time add date; before and ;date after the
command"
```

1.1.3. training.sh

```
#!/usr/bin/env bash

#Clear the screen.
clear

#=================================================================
================================#
#                                    Download pictures from Flickr
#
#=================================================================
================================#

#Download pictures from Flickr
python Offlickr.py -p -n -i 113733456@N06 -d .

#List all xml files and save it in a
ls | egrep xml > xml_files.txt

#Get the tags for every picture
python get_tags.py

#remove all xml files and the list of xml files saved in
xml_files.txt
rm *.xml xml_files.txt


#Clear the screen, ask the user to check manualy if al pictures are
download correctly. If not, download them by hand.
clear
echo "Pictures downloaded"
echo "Please check the downloaded pictures."
echo "Download all broken pictures by hand:"
echo "log in on Flickr with the shared naturalis account and go to:"
echo
"https://www.flickr.com/photos/113733456@N06/<picture_id>/sizes/o/"
echo "Press enter to continue"
read

#=================================================================
================================#
#                                Divide pictures between Flower and
Tuber                            #
#=================================================================
================================#

#Create the required directories
mkdir Flower Tuber Tuber/LOblong Tuber/LSpur Tuber/LRound
Tuber/Oblong Tuber/Round Tuber/Spur

#Loop through every jpg file
for i in *.jpg
```

```
do
    name=(${i//./ }$0)
    tags="$name""_tags.txt"
    #Conver from jpg to png
    echo "Convert $i to $name.png"
    convert $i $name.png
    content=$(cat $tags)
    echo "-------------------------------------------"
    #Divide the pictiure and tags between Tuber and Flower
    if [[ $content == *Tuber* ]]
    then
        mv $name.png Tuber
        mv $tags Tuber
    elif [[ $content == *Flower* ]]
    then
        mv $name.png Flower
        mv $tags Flower
    else
        echo "No correct tag found"
        echo "================================================"
    fi
done

#Remove the jpg that will not be used anymore
rm *.jpg

#Clear the screen, print a message and wait for input from the user.
clear
echo "Pictures divided between flower and tuber"
echo "Press enter to continue"
read

#================================================================
================================#
#        Divide the pictures in Flower between the different genera
and the different species       #
#================================================================
================================#

#go into the Flower directory
cd Flower

#Loop through the png files
for files in *.png
do
    name=(${files//./ }$0)
    tags="$name""_tags.txt"
    #Create variables that will be used to create directories.
    content=$(cat $tags)
    sectioni=$(sed -n '3p' < $tags)
    speciesi=$(sed -n '4p' < $tags)
    section=(${sectioni##*:})
    species=(${speciesi##*:})
    #Create a direcory with the name of the section
```

```
    mkdir $section
    #Move the picure and tag file to the the correct directory
    mv $files $section
    mv $tags $section
    #Go into the section directory and create a directory wiht the
name of the species
    cd $section
    mkdir $species
    #Move the picture and tag file to the correct directory
    mv $files $species
    mv $tags $species
    #Go back to the Flower directory
    cd ..
done
#After looping through the png files in the Flower directory go out
of this directory
cd ..

#Clear the screen, print a message and wait for input from the user
clear
echo "Flower pictures divided"
echo "Press enter to continue"
read

#================================================================
================================#
#                        Devide the pictures in Tuber between
shape(L-a-L)                              #
#================================================================
================================#

#Go into the Tuber Directory
cd Tuber

for files in *.png
do
    #Create the variables to divide the pictures.
    name=(${files//./ }$0)
    tags="$name""_tags.txt"
    content=$(cat $tags)
    #Divide the pictures to the correct directory
    if [[ $content == *Look-a-Like_round* ]]
    then
        mv $name.png  LRound/
        mv $tags LRound/
    elif [[ $content == *Look-a-Like_oblong* ]]
    then
        mv $name.png LOblong/
        mv $tags    LOblong/
    elif [[ $content == *Look-a-Like_spur* ]]
    then
        mv $name.png LSpur/
        mv $tags LSpur/
    elif [[ $content == *Round* ]]
```

```
    then
        mv $name.png Round/
        mv $tags Round/
    elif [[ $content == *Oblong* ]]
    then
        mv $name.png Oblong/
        mv $tags Oblong/
    elif [[ $content == *Spur* ]]
        then
        mv $name.png Spur/
        mv $tags Spur/
    else
        echo "No correct tag found"
    fi
done
#After looping through the png files in the Tuber directory go out
of this directory
cd ..

#Clear the screen, print a message and ask the user if he wants to
split the Tuber pictures.
clear
echo "Tuber pictures divided"
echo "Press enter to continue"
read

#================================================================
================================#
#                                Split all pictures of Tuber
#
#================================================================
================================#

#Go into the Tuber directory
cd Tuber

#Loop through all directories in this folder
for directories in $(ls -d */)
do
    directory=${directories%%/}
    #Set standard parameter values
    size=10000
    t=0.65
    #Some directories requires other values for size or for t
    if [[ $directory == *LO* ]]
    then
     size=50000
    elif [[ $directory == *LS* ]]
    then
     size=50000
    elif [[ $directory == R* ]]
    then
     t=0.6
    fi
```

```
    #Loop through all pictures
    for files in ./$directory/*.png
    do
     #Split the picture, using the given parameter values
     echo "Splitting $files"
     perl ../splitter.pl -t $t -i $files
     #Remove the noise pictures using the file size
     for FILENAME in *,*.png
     do
          FILESIZE=$(stat -f%z $FILENAME)
          if (( FILESIZE > size ))
          then
               mv $FILENAME ./$directory
          else
               rm $FILENAME
          fi
     done
    done
done
#When this step is finished go out of the Tuber directory
cd ..

#Clear the screen and print a message
clear
echo "Tuber pictures splited"

#Print a finish message
echo "The program is finished"
echo "To create traindata"
echo "Please run ./create_traindata.sh"
```

1.2. CSS

1.2.1. computer.css

```
/* This is the default lay-out for computers */

/* Lay-out for the body */
body {
            /* Place the text in the middel of the page */
            text-align: center;
}

/* Lay-out for the page */
#page {
    /* the width */
    width: 960px;

    /* place the text left */
    text-align: left;

    /* clear automatically the area around the page */
    margin: 10px auto 20px auto;

    /* Set the background color to white */
    background-color:white;
}

/* Lay-out for the logo */
#logo {
    /* Place the logo on the leftside of the page */
    float: left;

    /* the width of the logo is 200 pixels */
    width: 200px;
}

/* Lay-out for sidebar, not used, but available for later */
/*#sidebar {
    float: right;
    width: 200px;
    border: 1px solid #000;
}*/

/* Lay-out for the content */
#content {
    /* place the content left, because logo is described first, the
content will
    be displayed right of the logo */
    float: left;

    /* The width of this box will be automaticly changed to the
content self */
    width: auto;

    /* Place a border around the content
```

```
    3 pixels widht with a pink-like color, The same color of the
    Naturalis logo */
    border: 3px solid #E3004A;

    /* Clear 10 pixels around the content, inside de border */
    padding: 10px;
}

/* Lay-out for the footer */
#footer {
    /* Place the footer on the bottom */
    position:absolute;
    bottom:10px;
}

/* For all Font-settings below the Naturalis housestyle is aplied.
Which means that the Arial font is everywhere used */

/* Font-settings of p */
p {
    font-family: Arial;
    font-size: 1em;
}

/* Font-settings of h1 */
h1 {
    font-family: Arial;
    font-size: 1.9em;
}

/* Font-settings of h2 */
h2 {
    font-family: Arial;
    font-size: 1.7em;

}

/* Font-settings of h3 */
h3 {
    font-family: Arial;
    font-size: 1.5em;
}

/* Font-settings of h4 */
h4 {
    font-family: Arial;
    font-size: 1.3em;
}

/* id small, used to make text in p smaller than the settings for p
*/
#small {
    font-size: 0.75em;
}
```

```
/* Font-settings of form */
form {
    font-family: Arial;
    font-size: 1em;
}

/* Font-settings of inut */
input {
    font-family: Arial;
    font-size: 0.75em;
}

/* id button, for styling the individual buttons */
#button {
    font-family: Arial;
    font-size: 0.75em;
}
```

1.2.2. mobile.css

```
/* This is the default lay-out for mobile devices */

/* Lay-out for the body */
body {
            /* Place a margin of 10 pixels around the text */
            margin: 10px
}


/* Lay-out for the logo */
#logo {
    /* Place the logo on the leftside of the page */
    float: left;
}

/* Lay-out for sidebar, not used, but available for later */
/*#sidebar {
    float: right;
    width: 200px;
    border: 1px solid #000;
}*/

/* Lay-out for the content */
#content {
    /* place the content left, because logo is described first, the
content will
    be displayed under the logo */
    float: left;

    /* Place a border around the content
    13 pixels widht with a pink-like color, The same color of the
    Naturalis logo */
    border: 13px solid #E3004A;

    /* Clear 10 pixels around the content, inside de border */
    padding: 10px;
}

/* Lay-out for the footer */
#footer {
    /* Place the footer on the bottom */
    float:left;
    bottom:10px;
}

/* For all Font-settings below the Naturalis housestyle is aplied.
Which means that the Arial font is everywhere used */

/* Font-settings of p */
p {
    font-family: Arial;
```

```
    font-size: 40px;
}

/* Font-settings of h1 */
h1 {
    font-family: Arial;
    font-size: 120px;
}

/* Font-settings of h2 */
h2 {
    font-family: Arial;
    font-size: 100px;

}

/* Font-settings of h3 */
h3 {
    font-family: Arial;
    font-size: 75px;
}

/* Font-settings of h4 */
h4 {
    font-family: Arial;
    font-size: 50px;
}

/* id small, used to make text in p smaller than the settings for p
*/
#small {
    font-size: 25px;
}

/* Font-settings of form */
form {
    font-family: Arial;
    font-size: 30px;
}
/* Font-settings of inut */
input {
    font-family: Arial;
    font-size: 40px;
}

/* id button, for styling the individual buttons */
#button {
    font-family: Arial;
    font-size: 40px;
}
```

1.3. HTML

1.3.1. computer_invalid_login.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Invalid login{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling the login details are incorrect -
->
    <h2>Your login details are invalid!</h2>

    <!-- Button to login again -->
    <form action="/accounts/login/">
        <!-- The submit button with the text Try again inside -->
        <input type="submit" value="Try again" id="button">
    </form>

    <p></p>
    <!-- Button to go back to the welcome page -->
    <form action="/welcome/">
        <!-- The submit button with the text Home inside -->
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.2. computer_login.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Login{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header -->
    <h3>Login required to remove files!</h3>

    <!-- Ask for the username and password -->
    <form action="/accounts/auth/" method="post">{% csrf_token %}
        <label for="username">User name:</label>
        <input type="text" name="username" value="" id="username">
        <p></p>
        <label for="password">Password:</label>
        <input type="password" name="password" value=""
id="pasword">
        <p></p>
        <input type="submit" value="login" id="button">
    </form>

    <p></p>
    <!-- Button to go back to the home page -->
    <form action="/welcome/">
        <!-- The submit button with the text Home inside -->
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.3. computer_remove.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Files removed{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling the task has been completed -->
    <h2>Removing complete!</h2>
    <!-- Give detailed information -->
    <p>All uploaded pictures and their results<BR>
    are moved to the results directory</p>
    <p>You have removed the following files:</p>
    <p>{{uploads}}</p>
    <p>{{temps}}</p>

    <!-- Button to logout -->
    <form action="/accounts/logout/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Logout" id="button">
    </form>

{% endblock %}
```

1.3.4. computer_result.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Result{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header to tell this is de result page -->
    <h2>Results</h2>

    <!-- Display the result -->
    <p>Result: This is probably a <em>{{result}}</em> </p>

    <!-- Display the picture, give it a link to view the picter on a
new
    webpage / tab -->
    <p><a
href="/static/assets/uploaded_files/{{ip}}/{{filename}}"><img
src="/static/assets/uploaded_files/{{ip}}/{{filename}}"
width="200"/></a></p>

    <form action="/exit/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Exit" id="button">
    </form>
{% endblock %}
```

1.3.5. computer_sorry.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Error{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Header -->
    <h2>Sorry!</h2>

    <!-- Explain the error and ask to try again -->
    <p>During the process, your picture has been removed.<BR>
    Please try again</p>

    <!-- Button to try again -->
    <form action="/upload/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Upload picture" id="button">
    </form>

    <p></p>
    <!-- Button to go back to the homepage -->
    <form action="/welcome/">
        <!-- The submit button with the text Home inside -->
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.6. computer_upload_succes.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Upload succeeded{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling the file is uploaded -->
    <h2>You have uploaded the file!</h2>

    <!-- Display the picture, give it a link to view the picter on a
new
    webpage / tab -->
    <p><a href="/{{path}}"><img src="/{{path}}"
width="200"/></a></p>

    <form action="/result/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Result" id="button">
    </form>
    <!-- Display an empty line betwee both buttonts-->
    <p></p>
    <form action="/upload/">
        <!-- The submit button with the text Upload another picture
inside -->
        <input type="submit" value="Upload another picture"
id="button">
    </form>
{% endblock %}
```

1.3.7. computer_upload.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Upload picture{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header to tell this is de upload page -->
    <h2>Upload picture</h2>

    <p style="color:red; font-size:0.9em"><em>{{message}}</em></p>

    <!-- Use the form from orchid.views.upload to receive a picture
-->
    <form action="" method="post" style="{{style}}"
enctype="multipart/form-data">{% csrf_token %}
    {{form}}
    <p></p>
    <!-- Place a submit button under the form, with the text Upload
picture inside.
    To place it under the form, it is required to display an empty
line first-->
    <input type="submit" value="Upload picture" id="button">

    <!-- End of form -->
    </form>

    <p></p>
    <!-- Place a button under the form to go back to the home screen
-->
    <form action="/welcome/">
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.8. computer_welcome.html

```
<!-- Use the computer template and override the content block -->
{% extends "computer.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Home{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling this is the welcome page -->
    <h2>Welcome</h2

    <!-- Tell the user what this webapplication will do -->
    <h4>Welcome on the orchid identifier website of Naturalis</h4>
    <p>After uploading a picture the website will identify the
orchid.<BR>
    Please click the upload picture button below to upload a
picture.<BR>
    <em>You can upload a picture of a tuber.</em></p>

    <!-- Create a form which contains two buttons.
    The first button is for uploading the file -->
    <form action="/upload/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Upload picture" id="button">
    </form>
    <!-- display a white line -->
    <p></p>
    <!-- Second button to remove all leftover files -->
    <p id="small">To remove all unused uploaded pictures and their
temporary files<BR>
    click below <span style="color: red"><strong>(login
required)</strong></span></p>
    <form action="/admin/remove">
        <input type="submit" value="Remove unused files"
id="button">
    </form>

{% endblock %}
```

1.3.9. computer.html

```
<!-- THIS IS THE STANDARD HTML FOR THE COMPUTER LAY-OUTS -->

<!-- Load in the static function -->
{% load static %}

<!-- declare the DOCTYPE and html language-->
<!DOCTYPE html>
<html lang="en">

<!-- Create the head -->
<head>
    <!-- Create a title using a block -->
    <!-- This title needs to be changed! The best way to change the
title
    is in the other html files witch extends this html -->
    <title>{% block title %} My Base Template{% endblock %}</title>

        <!-- Use the compyter.css stylesheet for all the lay-outs --
>
        <link rel="stylesheet" type="text/css" href="{% static
"assets/css/computer.css" %}">

<!-- end of the head -->
</head>

<!-- Create the body -->
<body>
        <!-- Create a page -->
        <div id="page">

    <!-- Create a logo block, using the logo lay-out from
compyter.css -->
    <div id="logo">
        {% block logo %}
        <!-- Place the Naturalis logo inside the logo block -->
        <ul>
            <a href="http://www.naturalis.nl"><img src="{% static
"assets/images/Naturalis_logo.png" %}" width="100"/></a>
        </ul>
        <!-- End of the logo block -->
        {% endblock %}
    </div>

    <!-- Create a content block, using the content lay-out from
computer.css -->
    <div id="content">
        <!-- Place a standard text in the content -->
        <!-- This block will be override for all different html
pages -->
        {% block content %}This is the content of this block{%
endblock %}

    </div>
```

```
    <!-- Create a footer block, using the footer lay-out from
computer.css -->
    <div id="footer">
        <!-- Place some text in the footer block -->
        {% block footer %}<p>&copy;2013-2014 Patrick Wijntjes</p>{%
endblock %}

        <!-- End of the page -->
        </div>

<!-- End of the body and of the html -->
</body>
</html>
```

1.3.10. mobile_invalid_login.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Invalid login{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling the login details are incorrect -
->
    <h2>Your login details are invalid!</h2>

    <!-- Button to login again -->
    <form action="/accounts/login/">
        <!-- The submit button with the text Try again inside -->
        <input type="submit" value="Try again" id="button">
    </form>

    <p></p>
    <!-- Button to go back to the welcome page -->
    <form action="/welcome/">
        <!-- The submit button with the text Home inside -->
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.11. mobile_login.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Login{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header -->
    <h3>Login required to remove files!</h3>

    <!-- Ask for the username and password -->
    <form action="/accounts/auth/" method="post">{% csrf_token %}
        <label for="username">User name:</label>
        <input type="text" name="username" value="" id="username">
        <p></p>
        <label for="password">Password:</label>
        <input type="password" name="password" value=""
id="pasword">
        <p></p>
        <input type="submit" value="login" id="button">
    </form>

    <p></p>
    <!-- Button to go back to the home page -->
    <form action="/welcome/">
        <!-- The submit button with the text Home inside -->
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.12. mobile_remove.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Files removed{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling the task has been completed -->
    <h2>Removing complete!</h2>
    <!-- Give detailed information -->
    <p>All uploaded pictures and their results<BR>
    are moved to the results directory</p>
    <p>You have removed the following files:</p>
    <p>{{uploads}}</p>
    <p>{{temps}}</p>

    <!-- Button to logout -->
    <form action="/accounts/logout/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Logout" id="button">
    </form>

{% endblock %}
```

1.3.13. mobile_result.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Result{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header to tell this is de result page -->
    <h2>Results</h2>


    <!-- Display the result -->
    <p>Result: This is probably a <em>{{result}}</em> </p>

    <!-- Display the picture, give it a link to view the picter on a
new
    webpage / tab -->
    <p><a
href="/static/assets/uploaded_files/{{ip}}/{{filename}}"><img
src="/static/assets/uploaded_files/{{ip}}/{{filename}}"
width="200"/></a></p>

    <form action="/exit/">
        <!-- The submit button with the text Exit inside -->
        <input type="submit" value="Exit" id="button">
    </form>
{% endblock %}
```

1.3.14. mobile_sorry.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Error{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Header -->
    <h2>Sorry!</h2>

    <!-- Explain the error and ask to try again -->
    <p>During the process, your picture has been removed.<BR>
    Please try again</p>

    <!-- Button to try again -->
    <form action="/upload/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Upload picture" id="button">
    </form>

    <p></p>
    <!-- Buttont to go back to the homepage -->
    <form action="/welcome/">
        <!-- The submit buttont with the thext Home inside -->
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.15. mobile_upload_succes.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Upload succeeded{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling the file is uploaded -->
    <h2>You have uploaded the file!</h2>

    <!-- Display the picture, give it a link to view the picter on a
new
    webpage / tab -->
    <p><a href="/{{path}}"><img src="/{{path}}"
width="200"/></a></p>

    <form action="/result/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Result" id="button">
    </form>
    <!-- Display an empty line betwee both buttonts-->
    <p style="font-size:10;"></p>
    <form action="/upload/">
        <!-- The submit button with the text Upload another picture
inside -->
        <input type="submit" value="Upload another picture"
id="button">
    </form>
{% endblock %}
```

1.3.16. mobile_upload.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Upload picture{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header to tell this is de upload page -->
    <h2>Upload picture</h2>

    <!-- Font-size needs te be changed!!!!! -->
    <p style="{{style}};"><em>{{message}}</em></p>

    <!-- Use the form from orchid.views.upload to receive a picture
-->
    <form action="" method="post" style="{{style}}; font-size:60px"
enctype="multipart/form-data">{% csrf_token %}
    {{form}}
    <p></p>
    <!-- Place a submit button under the form, with the text Upload
picture inside.
    To place it under the form, it is required to display an empty
line first-->
    <input type="submit" value="Upload picture" id="button">

    <!-- End of form -->
    </form>

    <p></p>
    <!-- Place a button under the form to go back to the home screen
-->
    <form action="/welcome/">
        <input type="submit" value="Home" id="button">
    </form>

{% endblock %}
```

1.3.17. mobile_welcome.html

```
<!-- Use the mobile template and override the content block -->
{% extends "mobile.html" %}

<!-- Override the title -->
{% block title %}Orchid Identifier - Home{% endblock %}

{% block content %}
    <!-- All lines in this block will be diplayd in the content
block -->

    <!-- Display a header, telling this is the welcome page -->
    <h2>Welcome</h2

    <!-- Tell the user what this webapplication will do -->
    <h4>Welcome on the orchid identifier website of Naturalis</h4>
    <p>After uploading a picture the website will identify the
orchid.<BR>
    Please click the upload picture button below to upload a
picture.<BR>
    <em>You can upload a picture of a tuber.</em></p>

    <!-- Create a form which contains two buttons.
    The first button is for uploading the file -->
    <form action="/upload/">
        <!-- The submit button with the text Upload picture inside -
->
        <input type="submit" value="Upload picture" id="button">
    </form>
    <!-- display a white line -->
    <p></p>
    <!-- Second button to remove all leftover files -->
    <p id="small">To remove all unused uploaded pictures and their
temporary files<BR>
    click below <span style="color: red"><strong>(login
required)</strong></span></p>
    <form action="/admin/remove">
        <input type="submit" value="Remove unused files"
id="button">
    </form>

{% endblock %}
```

1.3.18. mobile.html

```
<!-- THIS IS THE STANDARD HTML FOR MOBILE DEVICES -->

<!-- Load in the static function -->
{% load static %}

<!-- declare the DOCTYPE and html language-->
<!DOCTYPE html>
<html lang="en">

<!-- Create the head -->
<head>
    <!-- Create a title using a block -->
    <!-- This title needs to be changed! The best way to change the
title
    is in the other html files witch extends this html -->
    <title>{% block title %} My Base Template{% endblock %}</title>

        <!-- Use the mobile.css stylesheet for all mobile lay-outs -
->
        <link rel="stylesheet" type="text/css" href="{% static
"assets/css/mobile.css" %}">

<!-- end of the head -->
</head>

<!-- Create the body -->
<body>
        <!-- Create a page -->
        <div id="page">

    <!-- Create a logo block, using the logo lay-out from mobile.css
-->
    <div id="logo">
        {% block logo %}
        <!-- Place the Naturalis logo inside the logo block -->
        <ul>
            <a href="http://www.naturalis.nl"><img src="{% static
"assets/images/Naturalis_logo.png" %}" width="250"/></a>
        </ul>
        <!-- End of the logo block -->
        {% endblock %}
    </div>

    <!-- Create a content block, using the content lay-out from
mobile.css -->
    <div id="content">
        <!-- Place a standard text in the content -->
        <!-- This block will be override for all mobile html pages -
->
        {% block content %}This is the content of this block{%
endblock %}

    </div>
```

```
    <!-- Create a footer block, using the footer lay-out from
mobile.css -->
    <div id="footer">
        <!-- Place some text in the footer block -->
        {% block footer %}<p>&copy;2013-2014 Patrick Wijntjes</p>{%
endblock %}

        <!-- End of the page -->
        </div>

<!-- End of the body and of the html -->
</body>
</html>
```

1.4. Perl

1.4.1. classify.pl [23]

```
#!/usr/bin/perl
use strict;
use warnings;
use Getopt::Long;
use Data::Dumper;
use AI::FANN ':all';
use Fingerprint 'make_fingerprint';
use Bio::Phylo::Util::Logger ':levels';

# process command line arguments
my $verbosity  = WARN;
my $resolution = 50;
my $dir;
my $ai;
GetOptions(
    'verbose+'     => \$verbosity,
    'dir=s'        => \$dir,
    'ai=s'         => \$ai,
    'resolution=i' => \$resolution,
);

# instantiate helper objects
my $log = Bio::Phylo::Util::Logger->new(
    '-level' => $verbosity,
    '-class' => 'main',
);
$log->info("going to instantiate AI from file $ai");
my $ann = AI::FANN->new_from_file($ai);

# read from the directory
$log->info("going to classify images in dir $dir");
opendir my $dh, $dir or die $!;
while( my $entry = readdir $dh ) {
    if ( $entry =~ /.png$/ ) {
     $log->debug("going to classify $entry");

     # analyse the input file
     my @fingerprint = make_fingerprint(
          'file'       => "$dir/$entry",
          'resolution' => $resolution,
     );
     $log->debug("made fingerprint of file");

     # do the classification
     my $result = $ann->run(\@fingerprint);
     print "Entry: $entry\n";
     print Dumper($result);
    }
}
```

1.4.2. splitter.pl [23]

```
#!/usr/bin/perl
use strict;
use warnings;
use Data::Dumper;
use Getopt::Long;
use Image::Magick;
use List::Util 'sum';
use Bio::Phylo::Util::Logger ':levels';

# will have deep recursions
no warnings 'recursion';

# process command line arguments
my $threshold = 0.7;
my $fuzzyness = 100; # pixels
my $verbosity = WARN;
my $infile;
GetOptions(
    'threshold=f' => \$threshold,
    'fuzzyness=i' => \$fuzzyness,
    'verbose+'    => \$verbosity,
    'infile=s'    => \$infile,
);

# instantiate helper objects
my $log = Bio::Phylo::Util::Logger->new(
    '-level' => $verbosity,
    '-class' => 'main',
);
my $img = Image::Magick->new;
my %seen;
my %area;

# read the image
$log->info("going to read image '$infile'");
my $msg = $img->Read($infile);
$log->warn($msg) if $msg;

# get width and height
my $width  = $img->Get('columns');
my $height = $img->Get('rows');
$log->info("width: $width height: $height");

# iterate over all pixels
for my $x ( 0 .. $width ) {
    for my $y ( 0 .. $height ) {
     my $nucleus = "$x,$y";
     recurse( 'x' => $x, 'y' => $y, 'nucleus' => $nucleus );
     if ( $area{$nucleus} ) {
          my $size = scalar @{ $area{$nucleus} };
          if ( $size > $fuzzyness ) {
               $log->info("found area of $size pixels around
```

```
nucleus $nucleus");
            }
      }
    }
}

# write large areas
for my $nucleus ( grep { scalar @{ $area{$_} } > $fuzzyness } keys
%area ) {
    my ($min_x) = sort { $a <=> $b } map { [ split(/,/, $_) ]->[0] }
@{ $area{$nucleus} };
    my ($max_x) = sort { $b <=> $a } map { [ split(/,/, $_) ]->[0] }
@{ $area{$nucleus} };
    my ($min_y) = sort { $a <=> $b } map { [ split(/,/, $_) ]->[1] }
@{ $area{$nucleus} };
    my ($max_y) = sort { $b <=> $a } map { [ split(/,/, $_) ]->[1] }
@{ $area{$nucleus} };

    # compute new area
    my $new_width  = $max_x - $min_x;
    my $new_height = $max_y - $min_y;
    $log->info("going to write $nucleus to
${new_width}x${new_height} file");

    # create new image, set dimensions, make white background
    my $new_img = Image::Magick->new( 'size' =>
${new_width}x${new_height}" );
    $msg = $new_img->Read('xc:white');
    $log->warn($msg) if $msg;
    $log->info("instantiated new image");

    # assign pixels
    for my $x ( 0 .. $new_width ) {
      for my $y ( 0 .. $new_height ) {
            my $loc = ( $x + $min_x ) . ',' . ( $y + $min_y );
            if ( $seen{$loc} ) {
                  $msg = $new_img->SetPixel( 'x' => $x, 'y' => $y,
'color' => $seen{$loc} );
                  $log->warn($msg) if $msg;
            }
      }
    }
    $log->info("assigned new pixels");

    # write image
    $msg = $new_img->Write("${nucleus}.png");
    $log->warn($msg) if $msg;
    $log->info("wrote image ${nucleus}.png");
}

sub recurse {
    my %args = @_;

    # get sub args
```

```
    my $nucleus   = delete $args{nucleus};
    my ( $x, $y ) = @args{qw(x y)};

    # sample the focal pixel
    my @pixel = $img->GetPixel(%args);

    # if pixel is darker than threshold and not yet seen...
    if ( sum(@pixel)/scalar(@pixel) < $threshold && ! $seen{"$x,$y"}
) {
      $log->debug("$x,$y");

      # store the pixel
      $seen{"$x,$y"} = \@pixel;

      # initialize area around current nucleus
      $area{$nucleus} = [] if not $area{$nucleus};

      # store id of the focal pixel
      push @{ $area{$nucleus} }, "$x,$y";

      # start growing the area
      if ( $x > 0 ) {
            recurse( 'x' => $x - 1, 'y' => $y, 'nucleus' => $nucleus
);
      }
      if ( $y > 0 ) {
            recurse( 'x' => $x, 'y' => $y - 1, 'nucleus' => $nucleus
);
      }
      if ( $x < $width ) {
            recurse( 'x' => $x + 1, 'y' => $y, 'nucleus' => $nucleus
);
      }
      if ( $y < $height ) {
            recurse( 'x' => $x, 'y' => $y + 1, 'nucleus' => $nucleus
);
      }
    }
}
```

1.4.3. trainai.pl [23]

```
#!/usr/bin/perl
use strict;
use warnings;
use Getopt::Long;
use AI::FANN ':all';
use Bio::Phylo::Util::Logger ':levels';

# process command line arguments
my $verbosity = WARN;
my $epochs  = 50000;
my $target  = 0.0001;
my $datadir = 'data/traindata';
my $outfile = 'data/ai/butterbeetle.ann';
my $categories = 1;
GetOptions(
    'verbose+'     => \$verbosity,
    'datadir=s'    => \$datadir,
    'epochs=i'     => \$epochs,
    'target=f'     => \$target,
    'outfile=s'    => \$outfile,
    'categories=i' => \$categories,
);

# instantiate helper objects
my $log = Bio::Phylo::Util::Logger->new(
    '-level' => $verbosity,
    '-class' => 'main',
);

# read the data files
my @interdigitated;
my $neurons;
$log->info("going to read traindata from $datadir");
opendir my $dh, $datadir or die $!;
while( my $entry = readdir $dh ) {
    if ( $entry =~ /\.tsv$/ ) {

     # read the table
     $log->info("going to read $datadir/$entry");
     open my $fh, '<', "$datadir/$entry" or die $!;
     my @header;
     LINE: while(<$fh>) {
          chomp;
          my @fields = split /\t/, $_;
          if ( not @header ) {
               @header = @fields;
               next LINE;
          }

          # first cell
          my $file = shift @fields;
```

```
            # last cells
            my @categ;
            for my $i ( 1 .. $categories ) {
                 push @categ, pop @fields;
            }

            # see AI::FANN docs for datastructure
            push @interdigitated, \@fields, \@categ;
            $neurons = scalar @fields; # +1 in hidden layer
            $log->info("read fingerprint for $file (@categ)");
        }
    }
}

# create the training data struct
my $train = AI::FANN::TrainData->new(@interdigitated);

# create the AI
my $ann = AI::FANN->new_standard( $neurons, $neurons + 1,
$categories );
$ann->hidden_activation_function(FANN_SIGMOID_SYMMETRIC);
$ann->output_activation_function(FANN_SIGMOID_SYMMETRIC);

# train the AI
$ann->train_on_data( $train, $epochs, $epochs / 100, $target );

# save the result
$ann->save($outfile);
```

1.4.4. traindata.pl [32]

```
#!/usr/bin/perl
use strict;
use warnings;
use Getopt::Long;
use Fingerprint 'make_fingerprint';
use Bio::Phylo::Util::Logger ':levels';

# process command line arguments
my $verbosity = WARN;
my $resolution = 50;
my $dir;
my $category;
GetOptions(
    'category=i'   => \$category,
    'resolution=i' => \$resolution,
    'dir=s'        => \$dir,
    'verbose+'     => \$verbosity,
);

# instantiate helper objects
my $log = Bio::Phylo::Util::Logger->new(
    '-level' => $verbosity,
    '-class' => [ 'main', 'Fingerprint' ],
);

# print the header
print "image\t";
for my $axis ( qw(vert horiz) ) {
    for my $color ( qw(red green blue) ) {
     my $max = $axis eq 'horiz' ? $resolution / 2 : $resolution;
     for my $i ( 1 .. $max ) {
          print "${axis}.${color}.${i}\t";
     }
    }
}
print "category\n";

# start reading the images
$log->info("going to read images from $dir");
opendir my $dh, $dir or die $!;
while( my $entry = readdir $dh ) {

    # only read png files created by splitter.pl
    if ( $entry =~ /(\d+,\d+)\.png/ ) {
     my $nucleus = $1;
     my @row = ( $nucleus );

     # read image
     my $img = Image::Magick->new;
     push @row, make_fingerprint(
          'file'       => $dir . '/' . $entry,
          'resolution' => $resolution,
     );
```

```
      $log->info("created fingerprint for $entry");

      push @row, $category;
      print join("\t", @row), "\n";
    }
}
```

1.4.5. traindata2.pl [23]

```
#!/usr/bin/perl
use strict;
use warnings;
use Getopt::Long;
use Fingerprint 'make_fingerprint';
use Bio::Phylo::Util::Logger ':levels';

# process command line arguments
my $verbosity = WARN;
my $resolution = 50;
my $dir;
my $category;
GetOptions(
    'category=i'   => \$category,
    'resolution=i' => \$resolution,
    'dir=s'        => \$dir,
    'verbose+'     => \$verbosity,
);

# instantiate helper objects
my $log = Bio::Phylo::Util::Logger->new(
    '-level' => $verbosity,
    '-class' => [ 'main', 'Fingerprint' ],
);

# print the header
print "image\t";
for my $axis ( qw(vert horiz) ) {
    for my $color ( qw(red green blue) ) {
     my $max = $axis eq 'horiz' ? $resolution / 2 : $resolution;
     for my $i ( 1 .. $max ) {
          print "${axis}.${color}.${i}\t";
     }
    }
}
print "category\n";

# start reading the images
$log->info("going to read images from $dir");
opendir my $dh, $dir or die $!;
while( my $entry = readdir $dh ) {

    # only read png files created by splitter.pl
    if ( $entry =~ /(\d+,?\d*)\.png/ ) {
     my $nucleus = $1;
     my @row = ( $nucleus );

     # read image
     my $img = Image::Magick->new;
     push @row, make_fingerprint(
          'file'       => $dir . '/' . $entry,
          'resolution' => $resolution,
     );
```

```
      $log->info("created fingerprint for $entry");

      push @row, $category;
      print join("\t", @row), "\n";
    }
}
```

1.5. Python for training

1.5.1. Add_columns.py

```
import os

#Collect all tsv files and save the names in files.txt
os.system("ls | egrep '.tsv' > files.txt")

#Function for creating the variables
def create_variables():
    #Read the names of the tsv files and save them as a list in
python
    files = open("files.txt", 'r').readlines()

    #Create variables
    number = len(files)
    header = []
    out_list = []
    counter = 0
    maxi = 0

    #Run the create_output_list function
    create_output_list(number, header, out_list ,files, counter,
maxi)

#Function for creating the output lists
def create_output_list(number, header, out_list, files, counter,
maxi):
    #add information to the output lists
    for q in range(number):
        out_list.append(-1)
        header.append("C%s"%(q+1))
    #Run the change_files function
    change_files(header, out_list, number ,files, counter, maxi)

#Functin for changing the files. This function will add the columns
to the files.
def change_files(header, out_list, number, files, counter, maxi):
    #Loop through the list of files
    for x in range(number):
        #Create an output file
        y = files[x].strip()
        name = y.split(".")[0] + "_new.tsv"
        output = open(name, 'a')
        #Read the content of the file, save it as a list
        z = open(y, 'r').readlines()
        #Loop through the list of content
        for a in range(len(z)):
            #Create variables, b is the content list, i is the
output list
            b = z[a].split("\t")
            i = b[:-2]
            #When a is 0, it is the header
            if a == 0:
```

```
                #Add the header list to the output list
                for c in range(number):
                    i.append(header[c])
                #Write the content of the output list to the output
file
                for d in i:
                    output.write(d.strip() + "\t")
                #After looping through the output list write an
enter to the output file.
                output.write("\n")
            #When a is not 0 it is a normal line
            else:
                #When the counter (=file number) is equal to x,
column x gets an 1
                if counter == x:
                    out_list[x] = 1
                #Otherwise it keeps a 0
                else:
                    pass
                #Add the list of -1's (and 1's for column x ==
counter) to the output list
                for f in range(number):
                    i.append(out_list[f])
                #Write the contentn of the output list to the output
file
                for g in i:
                    output.write("%s\t" %(g))
                #After looping through the output list write an entr
to the output file.
                output.write("\n")
        #Search to the hights number of columns
        if maxi < len(i):
            maxi = len(i)
        #Set the column with 1 back to 0 for the next file
        out_list[x] = -1
        #Add 1 to the counter.
        counter += 1
        os.system("mv %s %s"%(y, y.split(".")[0]))
    print "\nnumber of categories: ", counter
    clean_up(output)

def clean_up(output):
    #Close the output file
    output.close()
    #Remove the temporary file
    os.system("rm files.txt")

#Run the create_variables function
create_variables()
```

1.5.2. Combine_files.py

```
import os

#function for creating the variables
def create_variables():
    #Save all tsv files created with add_columns.py in files.txt
    os.system("ls | egrep '.tsv' > files.txt")

    #Read the content of files.txt and save it as a list
    files = open('files.txt', 'r').readlines()

    #Create variables
    number = len(files)
    counter = 0
    #Get the path working directory
    paht = os.path.dirname(os.path.realpath("combine_files.py"))
    #The output file will named as the directory
    name = paht.split("/")[-1] + ".tsv"
    #print name
    output = open(name, 'a')
    #Run the process_files function
    process_files(number, files, counter, output)
    #Run the clean_up function
    clean_up(output, name)

#Function for combining the files
def process_files(number, files, counter, output):
    #Loop through the files list
    for file_index in range(number):
        file = files[file_index].strip()
        #Read the content of the file, save it as a list.
        content = open(file, 'r').readlines()
        #Loop through the lines
        for row_number in range(len(content)):
            #create a list, every columns is an entry of the list
            row = content[row_number].split("\t")
            #When counter is 0, this is the first file.
            if counter == 0:
                #The whole content of the file is written to the
output file
                for entry in row:
                    output.write(entry.strip() + "\t")
                #After looping through the content, write an enter
                output.write("\n")
            #When counter isn't 0, it is not the first file
            else:
                #When row_number is 0 it is the header
                if row_number == 0:
                    #The header will not be written to the output
file, because it is already there
                    pass
                #When row_number isn't 0 it is not the header
                else:
                    #The content will be written to the output file
```

```
                    for entry in row:
                        output.write(entry.strip() + "\t")
                    #After looping through the content, write an
enter to the output file.
                    output.write("\n")
        #Add 1 to the counter
        counter += 1
#Function for removing temporary files.
def clean_up(output, name):
    #Close the output file
    output.close()
    #Remove the temporary files.txt file
    os.system("rm files.txt")
    #Move the output file out of the Flower directory
    os.system("mv %s ../"%(name))

#Run the create variables function
create_variables()
```

1.5.3. get_tags.py

```
import os

#Function for getting all xml files
def get_xml_files():
    #Open xml_files.txt in read mode
    files = open("xml_files.txt", 'r')

    #Create a list of all the xml files
    infiles = files.readlines()

    #Close the temporary file
    files.close()

    #Run the create_tag_files function
    create_tag_files(infiles)

#Function for creating the tag files
def create_tag_files(infiles):
    #Loop through the list
    for file in infiles:
        #Remove all enters at the back of the filename
        infile = file.strip()
        #Get the picture id, to save the tags with the same number
as the picture
        #Example the name of the meta file is 123456789.xml so the
id is 123456789
        number = infile.split(".")[0]
        #Print a message
        print "Collecting the tags of file %s"%(infile)
        #Open the meta data file in read mode
        open_file = open(infile, 'r')
        #Make a list of the meta data
        read_file = open_file.readlines()
        #Try to find the tags
        try:
            '''One line befor the first tag you can find
"/t<tags>\n"
    So the first tag will be the index of "/t<tags>\n" +1'''
            #Get the index of the first tag
            start = read_file.index("\t<tags>\n") +1
            '''One line after the last tag you can find
"\t</tags>\n"
    So the last tag will be the index of "/t</tags>\n" -1. Since a
for-loop
    loops from start to end, EXCLUDING the end, you use the index of
"/t</tags>\n"'''
            #Get the index of the end of the tags
            end = read_file.index("\t</tags>\n")
            #Get the original name of the picture
            title = read_file[2].split(">")[1].split("<")[0]
            #Save the output name (using the id of the picture)
            out_name = "%s_tags.txt"%(number)
            #Open the output file in write mode
```

```
            output = open(out_name, 'w')
            print "The tags will be saved in %s"%(out_name)
            #Write the name of the picture and a white line to the
output file
            #The title will always be the first line of the output
file
            output.write("%s\n\n"%(title))
            #Loop through the tags
            for tag_line in range(start, end):
                #Write the text between <tag> and </tag> to the
output file
                output.write(read_file[tag_line].strip().split("
")[4].split('"')[1])
                #Write an enter to the output file
                output.write("\n")
            #When the loop ends, close the output file
            output.close()
            '''If there are no tags, a ValueError arise. Except this
Error and print
    a message that the file has no tags'''
        except ValueError:
            print "%s has no tags"%(infile)
        #Close the output file
        open_file.close()
        #break

#Run the get_xml_files function
get_xml_files()
```

1.5.4. Offlickr.py [18]

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# Offlickr
# Hugo Haas -- mailto:hugo@larve.net --
http://larve.net/people/hugo/
# Homepage: http://larve.net/people/hugo/2005/12/offlickr/
# License: GPLv2
#
# Daniel Drucker <dmd@3e.org> contributed:
#   * wget patch
#   * backup of videos as well
#   * updated to Beej's Flickr API version 1.2 (required)

import sys
import libxml2
import urllib
import getopt
import time
import os.path
import threading

# Beej's Python Flickr API
# http://beej.us/flickr/flickrapi/

from flickrapi import FlickrAPI
import logging

__version__ = '0.22 - 2009-03-20'
maxTime = '9999999999'

# Gotten from Flickr

flickrAPIKey = '1391fcd0a9780b247cd6a101272acf71'
flickrSecret = 'fd221d0336de3b6d'


class Offlickr:

    def __init__(
        self,
        key,
        secret,
        uid,
        httplib=None,
        dryrun=False,
        verbose=False,
        ):
        """Instantiates an Offlickr object
        An API key is needed, as well as an API secret and a user
id."""

        self.__flickrAPIKey = key
        self.__flickrSecret = secret
```

```
        self.__httplib = httplib

        # Get authentication token
        # note we must explicitly select the xmlnode parser to be
compatible with FlickrAPI 1.2

        self.fapi = FlickrAPI(self.__flickrAPIKey,
self.__flickrSecret,
                              format='xmlnode')
        (token, frob) = self.fapi.get_token_part_one()
        if not token:
            raw_input('Press ENTER after you authorized this
program')
        self.fapi.get_token_part_two((token, frob))
        self.token = token
        self.flickrUserId = uid
        self.dryrun = dryrun
        self.verbose = verbose

    def __testFailure(self, rsp):
        """Returns whether the previous call was successful"""

        if rsp['stat'] == 'fail':
            print 'Error!'
            return True
        else:
            return False

    def getPhotoList(self, dateLo, dateHi):
        """Returns a list of photo given a time frame"""

        n = 0
        flickr_max = 500
        photos = []

        print 'Retrieving list of photos'
        while True:
            if self.verbose:
                print 'Requesting a page...'
            n = n + 1
            rsp = self.fapi.photos_search(
                api_key=self.__flickrAPIKey,
                auth_token=self.token,
                user_id=self.flickrUserId,
                per_page=str(flickr_max),
                page=str(n),
                min_upload_date=dateLo,
                max_upload_date=dateHi,
                #The next line is added by Patrick Wijntjes, 14-01-
2014
                privacy_filter=5
                )
            if self.__testFailure(rsp):
                return None
```

```
            if rsp.photos[0]['total'] == '0':
                return None
            photos += rsp.photos[0].photo
            if self.verbose:
                print ' %d photos so far' % len(photos)
            if len(photos) >= int(rsp.photos[0]['total']):
                break

        return photos

    def getGeotaggedPhotoList(self, dateLo, dateHi):
        """Returns a list of photo given a time frame"""

        n = 0
        flickr_max = 500
        photos = []

        print 'Retrieving list of photos'
        while True:
            if self.verbose:
                print 'Requesting a page...'
            n = n + 1
            rsp = \

self.fapi.photos_getWithGeoData(api_key=self.__flickrAPIKey,
                    auth_token=self.token,
user_id=self.flickrUserId,
                    per_page=str(flickr_max), page=str(n))
            if self.__testFailure(rsp):
                return None
            if rsp.photos[0]['total'] == '0':
                return None
            photos += rsp.photos[0].photo
            if self.verbose:
                print ' %d photos so far' % len(photos)
            if len(photos) >= int(rsp.photos[0]['total']):
                break

        return photos

    def getPhotoLocation(self, pid):
        """Returns a string containing location of a photo (in
XML)"""

        rsp = \

self.fapi.photos_geo_getLocation(api_key=self.__flickrAPIKey,
                auth_token=self.token, photo_id=pid)
        if self.__testFailure(rsp):
            return None
        doc = libxml2.parseDoc(rsp.xml)
        info = doc.xpathEval('/rsp/photo')[0].serialize()
        doc.freeDoc()
        return info
```

```
    def getPhotoLocationPermission(self, pid):
        """Returns a string containing location permision for a
photo (in XML)"""

        rsp = \

self.fapi.photos_geo_getPerms(api_key=self.__flickrAPIKey,
                auth_token=self.token, photo_id=pid)
        if self.__testFailure(rsp):
            return None
        doc = libxml2.parseDoc(rsp.xml)
        info = doc.xpathEval('/rsp/perms')[0].serialize()
        doc.freeDoc()
        return info

    def getPhotosetList(self):
        """Returns a list of photosets for a user"""

        rsp =
self.fapi.photosets_getList(api_key=self.__flickrAPIKey,
                auth_token=self.token, user_id=self.flickrUserId)
        if self.__testFailure(rsp):
            return None
        return rsp.photosets[0].photoset

    def getPhotosetInfo(self, pid, method):
        """Returns a string containing information about a photoset
(in XML)"""

        rsp = method(api_key=self.__flickrAPIKey,
                     auth_token=self.token, photoset_id=pid)
        if self.__testFailure(rsp):
            return None
        doc = libxml2.parseDoc(rsp.xml)
        info = doc.xpathEval('/rsp/photoset')[0].serialize()
        doc.freeDoc()
        return info

    def getPhotoMetadata(self, pid):
        """Returns an array containing containing the photo metadata
(as a string), and the format of the photo"""

        if self.verbose:
            print 'Requesting metadata for photo %s' % pid
        rsp = self.fapi.photos_getInfo(api_key=self.__flickrAPIKey,
                auth_token=self.token, photo_id=pid)
        if self.__testFailure(rsp):
            return None
        doc = libxml2.parseDoc(rsp.xml)
        metadata = doc.xpathEval('/rsp/photo')[0].serialize()
        doc.freeDoc()
        return [metadata, rsp.photo[0]['originalformat']]
```

```
    def getPhotoComments(self, pid):
        """Returns an XML string containing the photo comments"""

        if self.verbose:
            print 'Requesting comments for photo %s' % pid
        rsp = \

self.fapi.photos_comments_getList(api_key=self.__flickrAPIKey,
                auth_token=self.token, photo_id=pid)
        if self.__testFailure(rsp):
            return None
        doc = libxml2.parseDoc(rsp.xml)
        comments = doc.xpathEval('/rsp/comments')[0].serialize()
        doc.freeDoc()
        return comments

    def getPhotoSizes(self, pid):
        """Returns a string with is a list of available sizes for a
photo"""

        rsp = self.fapi.photos_getSizes(api_key=self.__flickrAPIKey,
                auth_token=self.token, photo_id=pid)
        if self.__testFailure(rsp):
            return None
        return rsp

    def getOriginalPhoto(self, pid):
        """Returns a URL which is the original photo, if it
exists"""

        source = None
        rsp = self.getPhotoSizes(pid)
        if rsp == None:
            return None
        for s in rsp.sizes[0].size:
            if s['label'] == 'Original':
                source = s['source']
        for s in rsp.sizes[0].size:
            if s['label'] == 'Video Original':
                source = s['source']
        return [source, s['label'] == 'Video Original']

    def __downloadReportHook(
        self,
        count,
        blockSize,
        totalSize,
        ):

        if not self.__verbose:
            return
        p = ((100 * count) * blockSize) / totalSize
        if p > 100:
            p = 100
```

```
        print '\r %3d %%' % p,
        sys.stdout.flush()

    def downloadURL(
        self,
        url,
        target,
        filename,
        verbose=False,
        ):
        """Saves a photo in a file"""

        if self.dryrun:
            return
        self.__verbose = verbose
        tmpfile = '%s/%s.TMP' % (target, filename)
        if self.__httplib == 'wget':
            cmd = 'wget -q -t 0 -T 120 -w 10 -c -O %s %s' %
(tmpfile,
                    url)
            os.system(cmd)
        else:
            urllib.urlretrieve(url, tmpfile,
                               reporthook=self.__downloadReportHook)
        os.rename(tmpfile, '%s/%s' % (target, filename))


def usage():
    """Command line interface usage"""

    print 'Usage: Offlickr.py -i <flickr Id>'
    print 'Backs up Flickr photos and metadata'
    print 'Options:'
    print '\t-f <date>\tbeginning of the date range'
    print '\t\t\t(default: since you started using Flickr)'
    print '\t-t <date>\tend of the date range'
    print '\t\t\t(default: until now)'
    print '\t-d <dir>\tdirectory for saving files (default: ./dst)'
    print '\t-l <level>\tlevels of directory hashes (default: 0)'
    print '\t-p\t\tback up photos in addition to photo metadata'
    print '\t-n\t\tdo not redownload anything which has already been
downloaded (only jpg checked)'
    print '\t-o\t\toverwrite photo, even if it already exists'
    print '\t-L\t\tback up human-readable photo locations and
permissions to separate files'
    print '\t-s\t\tback up all photosets (time range is ignored)'
    print '\t-w\t\tuse wget instead of internal Python HTTP library'
    print '\t-c <threads>\tnumber of threads to run to backup photos
(default: 1)'
    print '\t-v\t\tverbose output'
    print '\t-N\t\tdry run'
    print '\t-h\t\tthis help message'
    print '\nDates are specified in seconds since the Epoch
(00:00:00 UTC, January 1, 1970).'
```

```
    print '\nVersion ' + __version__


def fileWrite(
    dryrun,
    directory,
    filename,
    string,
    ):
    """Write a string into a file"""

    if dryrun:
        return
    if not os.access(directory, os.F_OK):
        os.makedirs(directory)
    f = open(directory + '/' + filename, 'w')
    f.write(string)
    f.close()
    print 'Written as', filename


class photoBackupThread(threading.Thread):

    def __init__(
        self,
        sem,
        i,
        total,
        id,
        title,
        offlickr,
        target,
        hash_level,
        getPhotos,
        doNotRedownload,
        overwritePhotos,
        ):

        self.sem = sem
        self.i = i
        self.total = total
        self.id = id
        self.title = title
        self.offlickr = offlickr
        self.target = target
        self.hash_level = hash_level
        self.getPhotos = getPhotos
        self.doNotRedownload = doNotRedownload
        self.overwritePhotos = overwritePhotos
        threading.Thread.__init__(self)

    def run(self):
        backupPhoto(
            self.i,
```

```
            self.total,
            self.id,
            self.title,
            self.target,
            self.hash_level,
            self.offlickr,
            self.doNotRedownload,
            self.getPhotos,
            self.overwritePhotos,
            )
        self.sem.release()


def backupPhoto(
    i,
    total,
    id,
    title,
    target,
    hash_level,
    offlickr,
    doNotRedownload,
    getPhotos,
    overwritePhotos,
    ):

    print str(i) + '/' + str(total) + ': ' + id + ': '\
         + title.encode('utf-8')
    td = target_dir(target, hash_level, id)
    if doNotRedownload and os.path.isfile(td + '/' + id + '.xml')\
         and os.path.isfile(td + '/' + id + '-comments.xml')\
         and (not getPhotos or getPhotos and os.path.isfile(td + '/'
               + id + '.jpg')):
        print 'Photo %s already downloaded; continuing' % id
        return

    # Get Metadata

    metadataResults = offlickr.getPhotoMetadata(id)
    if metadataResults == None:
        print 'Failed!'
        sys.exit(2)
    metadata = metadataResults[0]
    format = metadataResults[1]
    t_dir = target_dir(target, hash_level, id)

    # Write metadata

    fileWrite(offlickr.dryrun, t_dir, id + '.xml', metadata)

    #The following lines were commented out by Patrick Wijntjes, 14-
01-2014
    '''# Get comments
```

```
    photoComments = offlickr.getPhotoComments(id)
    fileWrite(offlickr.dryrun, t_dir, id + '-comments.xml',
              photoComments)'''

    # Do we want the picture too?

    if not getPhotos:
        return
    [source, isVideo] = offlickr.getOriginalPhoto(id)

    if source == None:
        print 'Oopsie, no photo found'
        return

    # if it's a Video, we cannot trust the format that getInfo told
us.
    # we have to make an extra round trip to grab the Content-
Disposition
    isPrivateFailure = False
    if isVideo:
        sourceconnection = urllib.urlopen(source)
        try:
            format = sourceconnection.headers['Content-
Disposition'].split('.')[-1].rstrip('"')
        except:
            print 'warning: private videos cannot be backed up due
to a Flickr bug'
            format = 'privateVideofailure'
            isPrivateFailure = True

    filename = id + '.' + format

    if os.path.isfile('%s/%s' % (t_dir, filename))\
         and not overwritePhotos:
        print '%s already downloaded... continuing' % filename
        return
    if not isPrivateFailure:
        print 'Retrieving ' + source + ' as ' + filename
        offlickr.downloadURL(source, t_dir, filename, verbose=True)
        print 'Done downloading %s' % filename


def backupPhotos(
    threads,
    offlickr,
    target,
    hash_level,
    dateLo,
    dateHi,
    getPhotos,
    doNotRedownload,
    overwritePhotos,
    ):
    """Back photos up for a particular time range"""
```

```
    if dateHi == maxTime:
        t = time.time()
        print 'For incremental backups, the current time is %.0f' %
t
        print "You can rerun the program with '-f %.0f'" % t

    photos = offlickr.getPhotoList(dateLo, dateHi)
    if photos == None:
        print 'No photos found'
        sys.exit(1)

    total = len(photos)
    print 'Backing up', total, 'photos'

    if threads > 1:
        concurrentThreads = threading.Semaphore(threads)
    i = 0
    for p in photos:
        i = i + 1
        pid = str(int(p['id']))  # Making sure we don't have weird
things here
        if threads > 1:
            concurrentThreads.acquire()
            downloader = photoBackupThread(
                concurrentThreads,
                i,
                total,
                pid,
                p['title'],
                offlickr,
                target,
                hash_level,
                getPhotos,
                doNotRedownload,
                overwritePhotos,
                )
            downloader.start()
        else:
            backupPhoto(
                i,
                total,
                pid,
                p['title'],
                target,
                hash_level,
                offlickr,
                doNotRedownload,
                getPhotos,
                overwritePhotos,
                )


def backupLocation(
```

```
    threads,
    offlickr,
    target,
    hash_level,
    dateLo,
    dateHi,
    doNotRedownload,
    ):
    """Back photo locations up for a particular time range"""

    if dateHi == maxTime:
        t = time.time()
        print 'For incremental backups, the current time is %.0f' %
t
        print "You can rerun the program with '-f %.0f'" % t

    photos = offlickr.getGeotaggedPhotoList(dateLo, dateHi)
    if photos == None:
        print 'No photos found'
        sys.exit(1)

    total = len(photos)
    print 'Backing up', total, 'photo locations'

    i = 0
    for p in photos:
        i = i + 1
        pid = str(int(p['id']))  # Making sure we don't have weird
things here
        td = target_dir(target, hash_level, pid) + '/'
        if doNotRedownload and os.path.isfile(td + pid + '-
location.xml'
                ) and os.path.isfile(td + pid
                 + '-location-permissions.xml'):
            print pid + ': Already there'
            continue
        location = offlickr.getPhotoLocation(pid)
        if location == None:
            print 'Failed!'
        else:
            fileWrite(offlickr.dryrun, target_dir(target,
hash_level,
                      pid), pid + '-location.xml', location)
        locationPermission =
offlickr.getPhotoLocationPermission(pid)
        if locationPermission == None:
            print 'Failed!'
        else:
            fileWrite(offlickr.dryrun, target_dir(target,
hash_level,
                      pid), pid + '-location-permissions.xml',
                      locationPermission)
```

```
def backupPhotosets(offlickr, target, hash_level):
    """Back photosets up"""

    photosets = offlickr.getPhotosetList()
    if photosets == None:
        print 'No photosets found'
        sys.exit(0)

    total = len(photosets)
    print 'Backing up', total, 'photosets'

    i = 0
    for p in photosets:
        i = i + 1
        pid = str(int(p['id']))  # Making sure we don't have weird
things here
        print str(i) + '/' + str(total) + ': ' + pid + ': '\
             + p.title[0].text.encode('utf-8')

        # Get Metadata

        info = offlickr.getPhotosetInfo(pid,
                offlickr.fapi.photosets_getInfo)
        if info == None:
            print 'Failed!'
        else:
            fileWrite(offlickr.dryrun, target_dir(target,
hash_level,
                      pid), 'set_' + pid + '_info.xml', info)
        photos = offlickr.getPhotosetInfo(pid,
                offlickr.fapi.photosets_getPhotos)
        if photos == None:
            print 'Failed!'
        else:
            fileWrite(offlickr.dryrun, target_dir(target,
hash_level,
                      pid), 'set_' + pid + '_photos.xml', photos)


        # Do we want the picture too?


def target_dir(target, hash_level, id):
    dir = target
    i = 1
    while i <= hash_level:
        dir = dir + '/' + id[len(id) - i]
        i = i + 1
    return dir


def main():
    """Command-line interface"""
```

```
# Default options

flickrUserId = None
dateLo = '1'
dateHi = maxTime
getPhotos = False
overwritePhotos = False
doNotRedownload = False
target = 'dst'
photoLocations = False
photosets = False
verbose = False
threads = 1
httplib = None
hash_level = 0
dryrun = False

# Parse command line

try:
    (opts, args) = getopt.getopt(sys.argv[1:],
            'hvponNLswf:t:d:i:c:l:', ['help'])
except getopt.GetoptError:
    usage()
    sys.exit(2)
for (o, a) in opts:
    if o in ('-h', '--help'):
        usage()
        sys.exit(0)
    if o == '-i':
        flickrUserId = a
    if o == '-p':
        getPhotos = True
    if o == '-o':
        overwritePhotos = True
    if o == '-n':
        doNotRedownload = True
    if o == '-L':
        photoLocations = True
    if o == '-s':
        photosets = True
    if o == '-f':
        dateLo = a
    if o == '-t':
        dateHi = a
    if o == '-d':
        target = a
    if o == '-w':
        httplib = 'wget'
    if o == '-c':
        threads = int(a)
    if o == '-l':
        hash_level = int(a)
    if o == '-N':
```

```
            dryrun = True
        if o == '-v':
            verbose = True

    # Check that we have a user id specified

    if flickrUserId == None:
        print 'You need to specify a Flickr Id'
        sys.exit(1)

    # Check that the target directory exists

    if not os.path.isdir(target):
        print target + ' is not a directory; please fix that.'
        sys.exit(1)

    offlickr = Offlickr(
        flickrAPIKey,
        flickrSecret,
        flickrUserId,
        httplib,
        dryrun,
        verbose,
        )

    if photosets:
        backupPhotosets(offlickr, target, hash_level)
    elif photoLocations:
        backupLocation(
            threads,
            offlickr,
            target,
            hash_level,
            dateLo,
            dateHi,
            doNotRedownload,
            )
    else:
        backupPhotos(
            threads,
            offlickr,
            target,
            hash_level,
            dateLo,
            dateHi,
            getPhotos,
            doNotRedownload,
            overwritePhotos,
            )


if __name__ == '__main__':
    main()
```

1.6. Python for website

1.6.1. forms.py

```
# Import the required modules
from django import forms
from models import Orchid

# Class for uploading pictures
class UploadPictureForm(forms.ModelForm):

    # The meta data
    class Meta:
        # The used model, Orchids in this case
        model = Orchid
```

1.6.2. result.py

```
#Import the required module
import sys

#Function for creating the variables
def create_variables():
    #Save the ip-adress which is the second commandline argument
    ip = sys.argv[1]

    #Create a list with the possible sections.
    #Sort them so form Z-A. The index of the section
    #Now is corresponding the index of the list from the
classify.pl script
    sections = ["Parvisepalum", "Pardalopetalum", "Paphiopedilum",
"Coryopedilum", "Cochlopetalum", "Brachypetalum", "Barbata"]

    #Open the output file from the classify.pl script.
    infile = open("%s_out.txt"%(ip), 'r')

    #Read in the content of the output file
    file1 = infile.readlines()

    #Create a variable for the index
    index = -1

    #Create an empty list. This list will be used
    #To save the values of the output list form classify.pl
    values = []

    #Create a counter
    counter = -1

    #Create an output file
    output = open("%s_result.txt"%(ip) , 'w')

    #Run the get_section function, give it all the created
variables.
    get_section(sections, file1, index, values, counter, output)

def get_section(sections, file1, index, values, counter, output):
    #Loop through the lines of the file
    for line in file1:
        #For every loop add 1 to the counter
        counter += 1
        #To accept error use a try-except
        try:
            #Find the lines which contains the values
            if counter >= 1 and counter <= 7:
                #Add these values to the values list

values.append(float(line.strip().strip(",").strip("'")))
        #If an error occur, except this and continue
        except:
            continue
```

```
    #Loop through the values
    for number in values:
        #When the value is bigger than 0, the picture
        #Is classified to this section.
        if number > 0:
            #Get the index of this value
            index = values.index(number)
        #Otherwise
        else:
            #Go on
            continue

    #The result of the cassification script is the section
    #Which index corresponds to the index of the positive value
    result = sections[index]


    #Write the result to this file and close the file
    output.write("%s"%(result))
    output.close()

#Run the create_variables function to create the variables
create_variables()
```

1.6.3. views.py

```
# import the required modules
from django.http import HttpResponseRedirect
from django.shortcuts import render_to_response
from forms import UploadPictureForm
from django.core.context_processors import csrf
from django.contrib import auth
from time import time
from django.contrib.auth.decorators import login_required
import os

# Function to get the used devise.
def get_device( request ):
    """ Redirect to the servers list. """
    #Initiate the device variable
    device = ""
    #If the used device is in the list, the device is a mobile phone
    '''I have test both html-styles on the iPad. The results shows
that the iPad can
    better show the computer style'''
    if 'HTTP_USER_AGENT' in request.META and (
      request.META['HTTP_USER_AGENT'].startswith( 'BlackBerry' ) or
\
      "Opera Mobi" in request.META.get('HTTP_USER_AGENT') or \
      "Opera Mini" in request.META.get('HTTP_USER_AGENT') or \
      "Windows CE" in request.META.get('HTTP_USER_AGENT') or \
      "MIDP"       in request.META.get('HTTP_USER_AGENT') or \
      "Palm"       in request.META.get('HTTP_USER_AGENT') or \
      "NetFront"   in request.META.get('HTTP_USER_AGENT') or \
      "Nokia"      in request.META.get('HTTP_USER_AGENT') or \
      "Symbian"    in request.META.get('HTTP_USER_AGENT') or \
      "UP.Browser" in request.META.get('HTTP_USER_AGENT') or \
      "UP.Link"    in request.META.get('HTTP_USER_AGENT') or \
      "WinWAP"     in request.META.get('HTTP_USER_AGENT') or \
      "Android"    in request.META.get('HTTP_USER_AGENT') or \
      "DoCoMo"     in request.META.get('HTTP_USER_AGENT') or \
      "KDDI-"      in request.META.get('HTTP_USER_AGENT') or \
      "Softbank"   in request.META.get('HTTP_USER_AGENT') or \
      "J-Phone"    in request.META.get('HTTP_USER_AGENT') or \
      "IEMobile"   in request.META.get('HTTP_USER_AGENT') or \
      "iPod"       in request.META.get('HTTP_USER_AGENT') or \
      "iPhone"     in request.META.get('HTTP_USER_AGENT') ):
        device = "mobile"
    #Otherwise it is a computer.
    else:
        device = "computer"
    #Return the device
    return device

# Function to check if the uploaded file is a picture
def check_upload(upload):
    #Create a list with picture extentions
    picture = ["jpg","png","jpeg"]
```

```
    #Get the extension from the uploaded file
    name = str(upload)
    extension = name.lower().split(".")[-1]

    #Check if the extension is in the picture list
    if extension in picture:
        #If it is, return true
        return True
    #Otherwise
    else:
        #Remove the file from the server
        name = name.replace(" ","\ ")
        os.system("rm static/uploaded_files/%s"%(name))
        #Retrun false
        return False


# Welcome view (homepage)
def welcome(request):
    #Get the used device, using the get_device function
    device = get_device(request)

    # Create the args dictionary and save the csrf in this dictonary
    args = {}
    args.update(csrf(request))
    # Save the html name, with the used device
    html = device+"_welcome.html"

    # Call the html, for the correct device, for de welcome page.
    return render_to_response(html, args)


#Function to give the uploaded file a variable part in front of the
filename
def processUpload(request, filename):

    #To use paths spaces need to be replaced by \<space>
    filename = str(filename).replace(" ","\ ")
    #To make the new filename easier to acces repalce the spaces by
"_"
    filename2 = str(filename).replace(" ","_")


    # Get the IP-adres of the computer
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        ip = x_forwarded_for.split(',')[0]
    else:
        ip = request.META.get('REMOTE_ADDR')

    # Replace the '.' in the ip-adres to '_'
    ip = ip.replace('.', '_')

    # Create an output file named <ip>.filename.txt
```

```
    outfile = open('%s.filename.txt' %(ip), 'w')

    # Place the variable part (the ip) in front of the filename of
the uploaded file
    os.system("mv static/uploaded_files/%s
static/uploaded_files/%s_%s"%(filename, ip, filename2))

    # Write the new filename to the outputfile
    outfile.write("%s_%s" %(ip, filename2))

    # Close the outputfile
    outfile.close()


# The upload view (choice file and upload it)
def upload(request):
    #Get the used device, using the get_device function
    device = get_device(request)

    # Get the IP-adres of the computer
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        ip = x_forwarded_for.split(',')[0]
    else:
        ip = request.META.get('REMOTE_ADDR')

    # Replace the '.' in the ip-adres to '_'
    ip = ip.replace('.', '_')

    #If the user uploaded a file that isn't a picture, a message
will be displaid.
    #Also the color of the upload button will be red.
    #To make this posible, the variables need to be created befor
the if statement.
    message = ""
    style = ""
    # Check if the method is POST
    if request.method == 'POST':

        #If the method is POST give the message and style variables
the
        #correct values
        message = "You didn't select a picture"
        style = "color:red"

        # Save the user input from the form
        form = UploadPictureForm(request.POST, request.FILES)

        # Check if the form is valid
        if form.is_valid():

            # Save the form
            form.save()
```

```
            #Check if the uploaded file is a picture
            is_picture = check_upload(request.FILES["picture"])

            #When the uploaded file is a picture
            if is_picture:

                # run the processUpload function to place the ip in
front of the name of the uploaded file
                processUpload(request, request.FILES["picture"])

                ''' save the filename and path in python variables
                use the variable part (the ip) to create the path'''
                filename = str(request.FILES["picture"]).replace("
", "_")
                path = ("static/assets/uploaded_files/%s_%s" % (ip,
filename))

                # Create the args dictionary and save the csrf in
this dictonary
                args = {}
                args.update(csrf(request))

                # save the filename and path in the dictionary
                args['filename'] = filename
                args['path'] = path

                # Save the html name, whit the used device
                html = device+"_upload_succes.html"

                # Call the upload_succes html, for the correct
device and give it the args dictonary
                return render_to_response(html, args)

            #When the uploaded file isn't a picture
            else:
                # Create the args dictionary and save the csrf in
this dictonary
                args = {}
                args.update(csrf(request))

                # Save the empty form, message and style in the
dictionary
                args['form'] = UploadPictureForm()
                args['message'] = message
                args['style'] = style

                # Save the html name, with the used device
                html = device+"_upload.html"
                # Call the upload html, for the correct device and
give it the args dictionary
                return render_to_response(html, args)


    # When the method is not POST
```

```
    else:
        # Create a form to upload a picture
        form = UploadPictureForm()

    # Create the args dictionary and save the csrf in this dictonary
    args = {}
    args.update(csrf(request))

    # Save the empty form, message and style in the dictionary
    args['form'] = UploadPictureForm()
    args['message'] = message
    args['style'] = style

    # Save the html name, with the used device
    html = device+"_upload.html"
    # Call the upload html, for the correct device and give it the
args dictionary
    return render_to_response(html, args)

# The result view (to display the result of the analysis)
def result(request):
    #Get the used device, using the get_device function
    device = get_device(request)
    try:
        # Get the IP-adres of the computer
        x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
        if x_forwarded_for:
            ip = x_forwarded_for.split(',')[0]
        else:
            ip = request.META.get('REMOTE_ADDR')

        # Replace the '.' in the ip-adres to '_'
        ip = ip.replace('.', '_')

        #Run converter.sh to convert jpg files to png
        os.system("sh converter.sh %s"%(ip))

        # Read in the filename from <ip>.filename.txt
        infile = open('%s.filename.txt' %(ip), 'r')
        filename = infile.read().strip()

        # Close the infile
        infile.close()

        # Run the program to classify the orchid
        os.system("python classify.py %s %s" % (filename, ip))
        #After the previous step a list with numbers is created.
        #Runt result.py to translate this list to a readable result.
        os.system("python result.py %s" % (ip))

        # Open the file with the result from the result.py program
        result = open('%s_result.txt' %(ip), 'r')

        # Read in the result
```

```
        read_result = result.read()

        # Close the file
        result.close()

        # Create the args dictionary and save the csrf in this
dictonary
        args = {}
        args.update(csrf(request))

        # Save the filename, the result and the ip in the args
dictionary
        args['filename'] = filename
        args['result'] = read_result
        args['ip'] = ip

        # Save the html name with the used device
        html = device+"_result.html"
        # Call the result html, for the correct device, with the
args dictionary
        return render_to_response(html, args)
    except IOError:
        '''If an IOError occure, the picture is uploaded just when
the administrator removed all
        unused files. So the uploaded picture is also removed. Send
the user to the sorry page,
        which tells the user to try uploading again.'''
        # Save the html name with the used device
        html = device+"_sorry.html"
        # Go to the sorry html, for the correct device
        return render_to_response(html)


# The exit view (to "close" the app and remove all created temporary
files)
def exit(request):
    # Get the IP-adres of the computer
    x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
    if x_forwarded_for:
        ip = x_forwarded_for.split(',')[0]
    else:
        ip = request.META.get('REMOTE_ADDR')

    # Replace the '.' in the ip-adres to '_'
    ip = ip.replace('.', '_')

    ''' Create the variable part for the filename using a timestamp.
    replace all . into _ to prevent errors for the extension '''
    var_part = str(time()).replace('.', '_')

    # Read in the filename from <ip>.filename.txt, save it and close
the file
    infile = open('%s.filename.txt' %(ip), 'r')
    filename = infile.read().strip()
```

```
    infile.close()

    # Remove the temporary file <ip>.filename.txt
    # Move the uploaded picture and its result to the result
directory,
    # Save it as timestamp_ip.png, timestamp_ip_result.txt and
timestamp_ip_section.txt
    os.system("rm %s.filename.txt" %(ip))
    os.system("mv static/uploaded_files/%s/%s results/%s_%s" %(ip,
filename, var_part, filename))
    os.system("rm -r static/uploaded_files/%s" %(ip))
    os.system("mv %s_out.txt results/%s_%s_result.txt" %(ip,
var_part, ip))
    os.system("mv %s_result.txt results/%s_%s_section.txt" %(ip,
var_part, ip))

    # Go back to the welcome page
    return HttpResponseRedirect('/welcome')


# To remove all leftover files, login is required
def login(request):
    #Get the used device, using the get_device function
    device = get_device(request)
    # Create a dictionary and put the csrf in it
    args = {}
    args.update(csrf(request))

    #Save the html name with the used device
    html=device+"_login.html"
    #Go to the login html, for the correct device, give it the
dictionary
    return render_to_response(html, args)

# Function to check the username and password
def auth_view(request):
    # Get the username and password
    username = request.POST.get('username', '')
    password = request.POST.get('password', '')
    ''' If the username and password are incorrect user will be None
    Otherwise it will be the user '''
    user = auth.authenticate(username=username, password=password)

    ''' Go to the correct page (admin/remove for correct login,
accounts/invalid for
    invalid login)'''
    if user is not None:
        #Login the user
        auth.login(request, user)
        return HttpResponseRedirect('/admin/remove')
    else:
        return HttpResponseRedirect('/accounts/invalid')

# function for logout
```

```
def logout(request):
    #Log the user out
    auth.logout(request)
    #Go back to the welcome page
    return HttpResponseRedirect('/welcome/')

# Function for invalid login
def invalid_login(request):
    #Get the used device, using the get_device function
    device = get_device(request)
    # Go to the invalid login html, for the correct device
    html = device+"_invalid_login.html"
    return render_to_response(html)

@login_required
#User need to be registreded. Even when the user is not active this
user can login and remove the files.
def remove(request):
    #Get the used device, using the get_device function
    device = get_device(request)
    # List all the files that will be removed using a command line
command (ls)
    '''Save the name(s) of the picture(s) that will be removed in
uploads.txt and the
     name(s) of the temporary file(s) in temps.txt'''
    os.system("ls static/uploaded_files > uploads.txt")
    os.system("ls | egrep *.filename.txt > temps.txt")

    #Remove all the unused pictures and their temporary files
    os.system("rm -r static/uploaded_files/*")
    os.system("rm *filename.txt")

    #Read the content of the uploads.txt file and the temps.txt file
and save the content in
    # Python variables
    uploads_in = open("uploads.txt", 'r')
    temps_in = open("temps.txt", 'r')
    uploads = uploads_in.read()
    temps = temps_in.read()

    # Create the args dictionary and save the csrf in this dictonary
    args = {}
    args.update(csrf(request))

    #Save the list of the pictures that will be removed in the
dictionary
    args['uploads'] = uploads
    #Save the list of the temporary files that will be removed in
the dictionary
    args['temps'] = temps

    # Remove the text files wich contain the lists
    os.system("rm uploads.txt temps.txt")
```

```
    # Save the html name with the used device
    html = device+"_remove.html"
    # Call the html, for the correct device, and give it the args
directory
    return render_to_response(html, args)
```