Intermediate report

Self learning software to identify illegally traded orchid material

Figure 1 ***Cypripedium calceolus*** **[1]**

Patrick Wijntjes, s1057924

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

Period 02-09-2013 until 27-06-2014

Naturalis Biodiversity Center
Sylviusweg 72
2333 BE Leiden

Supervisors:
Dr. Barbara Gravendeel and
Dr. Rutger Vos

Lecturer from school:
Drs. Jan Oliehoek

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]. Despite this convention many orchids are illegally traded. 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 custom officers and employees of nature conservation organizations involved in confiscating illegally traded material.

Slipper orchids

During this project the focus is on slipper orchids and orchids from which *salep* is produced. 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 [4]. Slipper orchids are highly desired ornamentals.

Ground orchid bulbs of the Orchidoideae, also known as *salep,* are very popular in Turkey. 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 show 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 is unsustainable, laws have been established to protect these orchids. In Turkey there are three laws that would 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 “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 last 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 holds a list with species that may not be taken away from the wild for export [5]. As mentioned earlier, the exact ingredients in *salep* cannot be identified without molecular identification tools, therefore making 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 different orchid species would be handy. This application could be used on

laptops/desktops and smartphones/tablets by taking pictures of flowers, leaves or underground tubers and uploading the pictures to a website. A simple workflow of the application can be found in figure 2. In this project the focus is on creating the website and integrating the identification application. This application is currently under development at Naturalis.

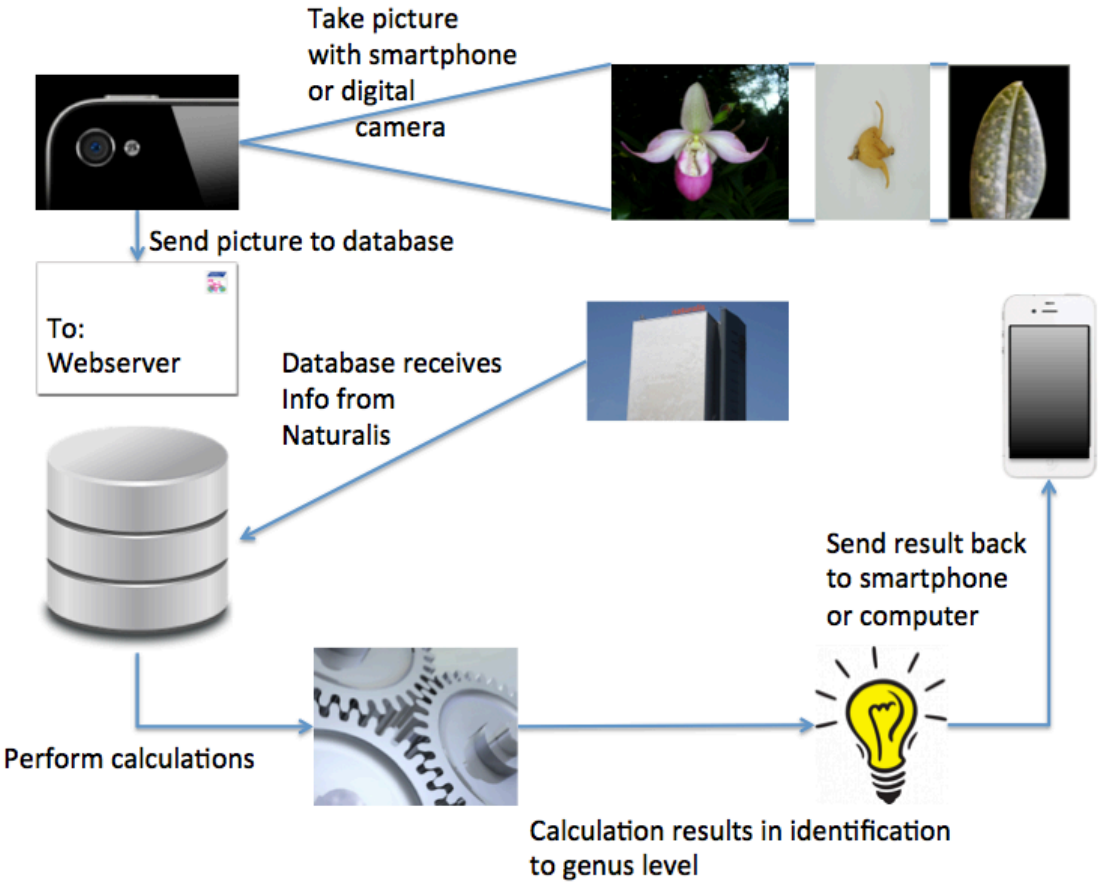

Figure 2 A simple workflow of the application made during this project. Resources of the pictures: [6-15]

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 / these pictures. When you use the software to unlock your computer the software takes a picture / a series of pictures of your face and compares this picture / these pictures with the saved picture(s). When it finds a match you will be logged-in to your own account.

Materials and Methods

Website

During this internship a website is developed. Users of this website can upload a picture to the server. On the server the software will check if the picture is an orchid or a look-a-like and, when possible, what the genus of the orchid is.

The processes behind this website are 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 3.2, 3.3 and 3.6.

Training

During this research the neural network is trained to see the differences between *salep* orchid tubers and tubers of orchid look-a-likes. The tubers of look-a-likes that are used are from *Arum maculatum*, *Asparagus officinalis, Polygonatum verticillatum, Tulipa greigii,* and *T. sp.* [17]. Appendix 1 contains pictures of these tubers. The neural network will also be trained on pictures of slipper orchid flowers.

Before training of the software is possible there are many steps to prepare the training. First of all, pictures of the *salep* orchid tubers and the tubers of the look-a-likes are required. So in the first few weeks of this research project pictures of *salep* orchid tubers and tubers of look-a-likes were taken at the Sylvius lab. Although the orientation of these tubers is unregulated, it is required that it is the same for all pictures. For instance, if the first picture of a tuber with spurs has the spurs on the right, all other tubers with spurs must have the spurs on the right as well. The user has to use the same orientation as the trainer, so this can be found in the user guide. 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.

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. It is not allowed to share the pictures of the flowers, so the settings of the Flickr account were set on private. This means that only persons with the account name and password can access the pictures.

On Flickr it is possible to add tags to the pictures. These tags are 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, was modified and used (see appendix 3.5.1) [18].

Before training of the neural network is possible, a preparation process is needed. The first 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 appendix 2.1. the blue square indicates the location of the tags. The next step is to convert the pictures from jpg- to png-format. Finally the pictures will be split and placed into the correct directory (Round, Spur and Oblong for the different *salep* orchid tubers, LRound, LSpur and LOblong for the different look-a-like tubers and the correct genus and species for the slipper orchid flowers) using the tags.

Results

Website

To make the software available for end-users a website was developed. The first design of this website is finished. It contains a homepage, a page for uploading a picture, a page to show the picture is uploaded correctly, and a result page. On the homepage it is possible to choose to upload a picture or remove the unused files from the server. For the last option it is required to log in with a valid account. 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 some modifications of the name of the picture are made behind the scenes. When these modifications are done the user is sent 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, a program will run to generate a result. The output of this program is sent back to the result page, where the user can view it.

Preparation script

To automate the preparation process a bash script was developed (see appendix 3.1.1). The first step in this script is 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 3.5.2), to get the original names and tags from the meta data. 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 the different tag files can be found in appendices 2.2-2.4. 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 genus, the species and lastly a tag that 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 this step the pictures are divided between two directories, Flower and Tuber. Before the pictures and tags are moved to the correct directory, the pictures are converted from .jpg to .png. After converting the pictures, the .jpg files remain in the training directory. These files are not used anymore, so after this step all .jpgs are removed.

After dividing the pictures between Flower and Tuber, the separation goes further. First the Flower pictures are divided between the different slipper orchid genera species. After this division there are some directories with generic names in Flower, and every genus 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 splitting the pictures of the tubers, using a Perl script developed by Rutger Vos: splitter.pl (see appendix 3.4.1). This script used a Perl package, named 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 normalized to be completely white (see the square in appendix 3.4.1). Figure 8 shows a

picture before and after splitting. The pictures of the flowers are already split, so this step was not required for these pictures.

Create training data

When the preparation script finishes the training data can be generated. To do this, a bash script, create_traindata.sh, is currently being developed (see appendix 3.1.3). On this moment, this script runs two different Perl scripts developed by Rutger Vos. One of these scripts is for the tubers and the other for the flowers (see appendices 3.4.2 and 3.4.3). The only difference between these two scripts is the accepted input files. 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.

The tsv files of the tuber can be used for training the neural network. The tsv files of the flowers need to be modified before using them for training the neural network. To automate this modification process a bash script is developed that will run three python scripts (see appendix 3.1.2 and appendices 3.5.3-3.5.5). The first python script adds some columns to the tsv files. To determine the number of columns the number of species per genus is used. After creating the columns they will be filled in. The guideline for this step is: the first species per genus gets 1's in the first column and 0's in all other columns. The second species gets 1' in the second column and 0' in all other columns etc.

The second python script combines the modified tsv files to one tsv file per genus. For training the neural network it is required that all used tsv files have the same number of columns. So a third python script is written to complete the missing columns. The first step is to find the file with the maximum number of columns. This number of columns is used to complete the other files to the same amount of columns. The new columns are filled in with 0's for every picture.

Neural networks

After completing all these preparation steps a neural network is trained. To train the network a Perl script, trainai.pl, is used (see appendix 3.4.4). There are two neural networks created using this script, one network for the tubers and another for the flowers. These networks will be used for identifying species from new pictures. The neural networks can be found using the next two links.
Network for flowers: https://github.com/naturalis/img-classify/blob/master/webapp/training/flower.ann
Network for tubers: https://github.com/naturalis/img-classify/blob/master/webapp/training/tuber.ann

Discussion

Because it is hard to extract DNA from a dried tuber, most of the tubers of confiscated orchids could not yet be identified to species level. During this project they could therefore only be used to train the neural network to see differences between orchid tubers and Look-a-Like tubers up to the generic level. When more of the tubers are identified correctly, it will be possible to use the pictures to further train the neural network to identify them to species level.

On this moment scripts of Rutger Vos are used to create the training data and to train the neural networks. On 6 February 2014 Serrano Pereira started as bioinformatics at Naturalis Biodiversity Center. He works on the same project and will develop new scripts to create the trainings data and train the neural networks. When these scripts are finished they will be used in my project as well.

References

[1] Royal Treatment for the Lady's slipper, 2013, http://www.empowernetwork.com/justiniskandar/blog/royal-treatment-for-the-ladys-slipper/

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

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

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

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

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

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

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

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

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

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

[14] Orchid leaf: A Close-up View of a Lady's Slipper Orchid, Brian Johnston, 2012, http://www.microscopy-uk.org.uk/mag/artjan12/bj-slipper3.html

[15] Photographs by Patrick Wijntjes

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

[17] 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

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

[19] Rutger Vos, Bio-informatican at Naturalis Biodiversity Center

Appendices

1. Figures

Figure 3 *Arum maculatum* [15]

Figure 6 *Tulipa greigii* [15]

Figure 4 *Asparagus officinalis* [15]

Figure 7 *Tulipa sp.* [15]

Figure 5 *Polygonatum verticillatum* [15]

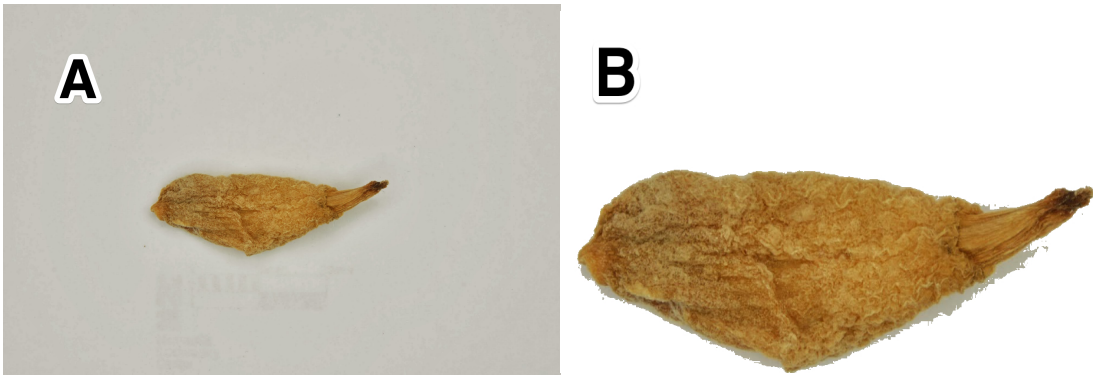

Figure 8 A: A picture of a tuber, before splitting. B: A picture of the same tuber after splitting [15].

2. Input and output files

2.1. 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>

2.2. Tag file of a flower

bellatulum12 phot

genus:Brachypetalum
species:bellatulum
Flower

2.3. Tag file of an *salep* tuber

TEH-1.1_5

Orchid_spur
Orchid
Spur
genus:Dactylorhiza
species:incarnata
Tuber

2.4. 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

3. Codes

3.1. Bash

3.1.1. training.sh

```
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 > a

#Get the tags for every picture
python get_tags.py

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

echo "Done"

clear

#====================================================================
================================#
#                                  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
#echo "File: $i"
    var=(${i//./ }$0)
#   echo "Var: $var"
    tags="$var""_tags.txt"
#   echo "Tag: $tags"
    #Conver from jpg to png
    echo "Convert $i to $var.png"
    convert $i $var.png
    content=$(cat $tags)
#   echo "Content: $content"
```

```
    echo "-----------------------------------------------"
    #Divide the pictiure and tags between Tuber and Flower
    if [[ $content == *Tuber* ]]
    then
        mv $var.png Tuber
        mv $tags Tuber
    elif [[ $content == *Flower* ]]
    then
        mv $var.png Flower
        mv $tags Flower
    else
        echo "No correct tag found"
        echo "================================================"
    fi
    #rm $tags
done

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

echo "Done"

clear

#================================================================
================================#
#        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 f in *.png
do
#echo "File: $f"
    var=(${f//./ }$0)
e#cho "Var: $var"
    tags="$var""_tags.txt"
#   echo "Tag: $tags"
    #Create variables that will be used to create directories.
    content=$(cat $tags)
    genusi=$(sed -n '3p' < $tags)
    speciesi=$(sed -n '4p' < $tags)
    genus=(${genusi##*:})
    species=(${speciesi##*:})
#echo "Speciesi: $speciesi"
#   echo "Species: $species"
    #Create a direcory with the name of the genus
    mkdir $genus
    #Move the picure and tag file to the the correct directory
    mv $f $genus
```

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

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

#Go into the Tuber Directory
cd Tuber

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

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

clear

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

#Go into the Tuber directory
cd Tuber

#Loop through all directories in this folder
for i in $(ls -d */)
do
    y=${i%%/}
#           echo "Y1: $y"
    #Set standard parameter values
    size=10000
    t=0.65
    #Some directories requires other values for size or for t
    if [[ $y == *LO* ]]
    then
        size=50000
    elif [[ $y == *LS* ]]
    then
        size=50000
    elif [[ $y == R* ]]
    then
        t=0.6
    fi
    #Loop through all pictures
    for f in ./$y/*.png
    do
        #Split the picture, using the given parameter values
        echo "Splitting $f"
        #       echo "-t: $t"
```

```
        #       echo "Size: $size"
#               pwd
#               echo "++++++++++++++++++++++++++++++++++++++++++++"
        perl ../splitter.pl -t $t -i $f
        #Remove the noise pictures using the file size
        for FILENAME in *,*.png
        do
            FILESIZE=$(stat -f%z $FILENAME)
            #           echo "$FILENAME: $FILESIZE"
            if (( FILESIZE > size ))
            then
            mv $FILENAME ./$y
            else
            rm $FILENAME
            fi
        done
    done
done

clear
echo "Done"
```

3.1.2. modify_flower_data.sh

```
#Go into the Flower directory
cd Flower

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

#run complete_columns.py
python ../complete_columns.py
#Remove the txt files with the length of the tsv files
rm *.txt
```

3.1.3. create_traindata.sh

```
#Loop throug the directories
for i in $(ls -d */)
do
    y=${i%%/}
#    echo ${i%%/}
    #The tuber pictures use another Perl script for training the
neural network
    #than the flower pictures. So check the name of the directory.
    if [[ $y == T* ]]
    #If it starts with T (=Tuber)
    then
        #Go into the directory
        cd $y
        #Loop through the directories
        for x in $(ls -d */)
        do
            #Create variables
            catagory=0
            a=${x%%/}
            echo "run traindata.pl for $a"
            #Give the Look-a-like tubers catagory -1
            if [[ $a == L* ]]
            then
                catagory=-1
            #Give the salep tubers catagory 1
            else
                catagory=1
            fi
#pwd
#           echo "perl ../traindata.pl -d ./$a -c $catagory >
$a.tsv"
            #Run the traindata script and redirect the output to a
tsv file.
            #Give this file the name of the current directory
            perl ../traindata.pl -d ./$a -c $catagory > $a.tsv
        done
    #After looping through the directories go out of the Tuber
directory
    cd ..
    elif [[ $y == F* ]]
    #If it starts with F (=Flower)
    then
        #Go into the directory
        cd $y
        #Loop through the genus directories
        for x in $(ls -d */)
        do
            #Go into the genus directory
            cd $x
#            echo "X: $x"
            #Loop through the species directories
            for z in $(ls -d */)
            do
```

```
                #Create variables
                b=${z%%/}
#                echo "Z: $z"
#                pwd
#                echo "perl ../../traindata2.pl -d ./$b > $b.tsv"
                #Run the traindata2 script and redirect the output
to a tsv file.
                #Give this file the name of the current directory
                perl ../../traindata2.pl -d ./$b -c 0 > $b.tsv
            done
            #After loopging through the species directories go back
to the Flower directory
            cd ..
         done
    #After looping through the genus directories gou out of the
Flower directory
    cd ..
    fi
done
#pwd

clear

echo "Done"
```

3.2. Css

3.2.1. 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;
}
```

3.2.2. 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;
}
```

3.3. Html

3.3.1. 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 %}
```

3.3.2. 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 %}
```

3.3.3. 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 %}
```

3.3.4. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

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

    <!-- FOR TESTING: Display the filename -->
    <p>Filename: {{filename}} </p>

    <!-- Display the result -->
    <!-- WARNING: this is a test result, and will always be a random
sentence
    This needs to be changed for the real version! -->
    <p>Result: {{result}} </p>

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

3.3.5. 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 calculation, 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 %}
```

3.3.6. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

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

    <!-- FOR TESTING: Display the filename -->
    <p>filename: {{filename}}</p>

    <!-- FOR TESTING: Display the path to the file -->
    <p>path: {{path}}</p>

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

3.3.7. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

    <!-- 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 %}
```

3.3.8. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

    <!-- 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, a leaf or a
flower.</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>

    <!-- FOR TESTING: Print the used divise -->
    <p>device: {{device}}</p>
{% endblock %}
```

3.3.9. 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>
```

3.3.10. 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 %}
```

3.3.11. 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 %}
```

3.3.12. 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 %}
```

3.3.13. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

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

    <!-- FOR TESTING: Display the filename -->
    <p>Filename: {{filename}} </p>

    <!-- Display the result -->
    <!-- WARNING: this is a test result, and will always be a random
sentence
    This needs to be changed for the real version! -->
    <p>Result: {{result}} </p>

    <!-- Display the picture, give it a link to view the picter on a
new
    webpage / tab -->
    <!-- Misschien de grootte aanpassen! -->
    <p><a href="/static/assets/uploaded_files/{{filename}}"><img
src="/static/assets/uploaded_files/{{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 %}
```

3.3.14. 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 calculation, 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 %}
```

3.3.15. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

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

    <!-- FOR TESTING: Display the filename -->
    <p>filename: {{filename}}</p>

    <!-- FOR TESTING: Display the path to the file -->
    <p>path: {{path}}</p>

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

3.3.16. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

    <!-- 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 %}
```

3.3.17. 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 -->
    <!-- The displaying content is only for testing and needs to be
updated
    for the real version -->

    <!-- 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, a leaf or a
flower.</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>

    <!-- FOR TESTING: Print the used divise -->
    <p>device: {{device}}</p>
{% endblock %}
```

3.3.18. 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>
```

3.4. Perl

3.4.1. splitter.pl [19]

```
#!/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
);
      }
    }
}
```

3.4.2. traindata.pl [19]

```
#!/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";
    }
}
```

3.4.3. traindata2.pl [19]

```
#!/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";
    }
}
```

3.5. Python for training

3.5.1. 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()
```

3.5.2. get_tags.py

```
import os

#Open a in read mode
files = open("a", 'r')

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

#Close and remove the temporary file
files.close()
#Loop through the list
for x in infiles:
    #Remove all enters at the back of the filename
    infile = x.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]
        #print title
        #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 y in range(start, end):
```

```
            #print read_file[y].strip().split(" ")[3].split('"')[1]
            #Write the text between <tag> and </tag> to the output
file
            output.write(read_file[y].strip().split("
")[3].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
```

3.5.3. add_columns.py

```
import os

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

#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 = []
counter = 0
maxi = 0
#Get the path working directory
path = os.path.dirname(os.path.realpath("add_columns.py"))
#The output file for the lenght will named as the directory
file = path.split("/")[-1] + ".txt"
lenght = open(file, 'w')

#add information to the output lists
for q in range(number):
    out.append(0)
    header.append("C%s"%(q+1))

#Loop through the list of files
for x in range(number):
    #print "file:", files[x].strip()
    #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
        #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[x] = 1
            #Otherwise it keeps a 0
            else:
                pass
            #Change the last column (=catagory) from a zero with an
enter to a zero without an enter
            i[-1] = 0
            #Add the list of 0's (and 1's for column x == counter)
to the output list
            for f in range(number):
                i.append(out[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")
    #print "lengte: ", len(i)
    #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[x] = 0
    #Add 1 to the counter.
    counter += 1
    #os.system("rm %s"%(y))

#Close the output file
output.close()
#Remove the temporary file
os.system("rm files.txt")
#print "Maxi:", maxi
#Writhe the highest number of columns to the lenght output file
lenght.write("%s"% (maxi))
#Close the lenght output file
lenght.close()
#Move the output file out the Flower directory
os.system("mv %s ../"%(file))
```

3.5.4. combine_files.py

```
import os

#Save all tsv files created with add_columns.py in files.txt
os.system("ls | egrep 'new' > 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')

#Loop through the files list
for x in range(number):
    y = files[x].strip()
    #print "file:", y
    #Read the content of the file, save it as a list.
    z = open(y, 'r').readlines()
    #Loop through the lines
    for a in range(len(z)):
        #create a list, every columns is an entry of the list
        b = z[a].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 d in b:
                output.write(d.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 a is 0 it is the header
            if a == 0:
                #The header will not be written to the output file,
because it is already there
                pass
            #When a isn't 0 it is not the header
            else:
                #The content will be written to the output file
                for d in b:
                    output.write(d.strip() + "\t")
                #After looping through the content, write an enter
to the output file.
                output.write("\n")
    #Add 1 to the counter
    counter += 1
```

```
    #os.system("rm %s"%(y))
#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))
```

3.5.5. complete_columns.py

```
import os

#Save a list of the length files in len_files
os.system("ls | egrep '.txt' > len_files")

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

#Create variables
name_list = []
names = []
max = 0
infile = ''

##print len_files

#Loop through the len_files and add the content to the list.
for x in len_files:
    name_list.append(x.strip())

##print name_list

#Loop through the name_list
for y in name_list:
    #Read the lenth of from the file
    file = int(open(y, 'r').readline())
##    print file
    #Find the maximum lenth
    if max < file:
        max = file
        #Save the name of the file with the maximum lenth
        infile = y

#print "Max:", max
#print "infile:", infile
##print "1:", name_list
#Remove the file wiht the maximum lenght from the name_list
name_list.remove(infile)
# name_list contains only the files that needs to be extended
##print "2:", name_list

#Loop through the name list
for z in name_list:
    #Add the names of the tsv files to the names list.
    name = z.split(".")[0] + ".tsv"
    names.append(name)
# Dit zijn de echte files die aangevuld moeten worden.
#print names

#Loop through the names list
for a in names:
##    print "-----------------------------------------------"
    #Read the content of the file and save it as a list
```

```
    content = open(a, 'r').readlines()
    #Loop through the content
    for c in range(len(content)):
        #When c is zero it is the header line
        if c == 0:
            #Save the header as list
            header = content[c].split("\t")
            #Remove the empty field and the enter from the header
            header.pop(-1)
            header.pop(-1)
            #Save the lenth of the header line (= number of columns)
            file_length = int(len(header))
        ##    print file_length
            #Calculate the difference between the max lenght and the
lenth of the current file
            difference = max-file_length
        ##    print difference
        ##    print header[-1]
            #Save the C-number of the last column
            number = int(header[-1].split("C")[1])
            #Create an output file
            outname = "%s_new.tsv" %(a.split(".")[0])
            #print outname
            output = open(outname, 'w')
            #Use the difference to add information to the header
list
            for q in range(difference):
                adding = "C" + str(number + q + 1)
                header.append(adding)
            #Write the content of the header list to the output file
            for i in header:
                output.write("%s\t"%(i))
            #After looping through the header list, write an enter
to the output file
            output.write("\n")
            #Close the output file
            output.close()
        #When c is not zero, it is not the header line
        else:
            # save the content of the line as a list.
            line = content[c].split("\t")
            #Remove the empty field and the enter from the header
            line.pop(-1)
            line.pop(-1)
            #Save the lenth of the line (= number of columns)
            file_length = int(len(line))
            #Calculate the difference between the max lenght and the
lenth of the current file
            difference = max-file_length
            #Open the output file again, using the append mode
            outname = "%s_new.tsv" %(a.split(".")[0])
##            print outname
            output = open(outname, 'a')
            #Use a the difference to add enough 0's
```

```
            for v in range(difference):
                line.append(0)
            #Write the content of the line list to the output file
            for w in line:
                output.write("%s\t"%(w))
            #After looping through the line list, write an enter to
the output file
            output.write("\n")
            #Close the output file
            output.close()
    #print "mv %s %s"%(outname, a)
    #Rename the output file
    os.system("mv %s %s"%(outname, a))
#Remove the temporary len_files file
os.system("rm len_files")
```

3.6. Python for website

3.6.1. 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
```

3.6.2. 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

def check_upload(upload):
    picture =
["jpg","tif","bmp","gif","png","jpeg","psd","pspimage","thm","yuv"]
```

```
    name = str(upload)
    extension = name.lower().split(".")[-1]

    if extension in picture:
        return True
    else:
        name = name.replace(" ","\ ")
        os.system("rm static/uploaded_files/%s"%(name))
        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))
    # ONLY FOR TESTING! Save the device in the args dictionary
    args['device']=device
    # 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):

    filename2 = str(filename).replace(" ","_")
    filename = 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('.', '_')

    message = ""
    style = ""
    # Check if the method is POST
    if request.method == 'POST':

        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()

            is_picture = check_upload(request.FILES["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"]) #
zie hier nog een extra regel

                ''' 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)

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

                # Save the empty form 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 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('.', '_')

        # 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 identify the orchid
        # Warning: The program now used is only a test program!
        os.system("python resultaat.py %s %s" % (filename, ip))

        # Open the file with the results from the identify program
        result = open('%s_test.txt' %(ip), 'r')

        # Read in the results
        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 and the result in the args dictionary
        args['filename'] = filename
        args['result'] = read_result

        # 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 . in _ 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.jpg and timestamp_ip_result.txt
    os.system("rm %s_filename.txt" %(ip))
    os.system("mv static/uploaded_files/%s results/%s_%s.jpg"
%(filename, var_part, filename))
    os.system("mv %s_test.txt results/%s_%s_result.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,
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 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)
```