Project Himanshu

January 8, 2019

1 Introduction

The goal of this kernel is to build a Keras cnn model to classify 12 kinds of plant seedlings.

We will use the V2 Plant Seedlings Dataset. This dataset contains 5,539 images distributed between 12 classes. The images show plant seedlings at different growth stages. Of the 12 classes, 3 classes are crop seedlings and 9 are weed seedlings.

The images are in different sizes. We will resize all images to 96x96 and use only 250 images from each class. We won't do any image augmentation.

This notebook will focus on:

```
Creating the folder structure that Keras generators need.
Creating generators to feed the images from the folders into the model.
Model building and training.
Assessing the quality of the model by generating a confusion matrix and a classification report
```

Results

This model will produce a validation accuracy that is greater than 90% and an F1 score of approximately 0.75.

```
In [ ]: from numpy.random import seed
        seed(101)
        from tensorflow import set_random_seed
        set_random_seed(101)

        import pandas as pd
        import numpy as np

        import tensorflow
        import keras

        from tensorflow.python.keras.models import Sequential
        from tensorflow.python.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D
                                                    , Flatten
        from tensorflow.python.keras.optimizers import Adam
        from tensorflow.python.keras.metrics import categorical_crossentropy
        from tensorflow.python.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.python.keras.models import Model
```

```
from tensorflow.python.keras.callbacks import EarlyStopping, ReduceLROnPlateau
                                              , ModelCheckpoint

import os
import cv2

import imageio
import skimage
import skimage.io
import skimage.transform

from sklearn.utils import shuffle
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
import itertools
import shutil
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
# Number of samples we will have in each class.
smpl_sz = 250

# The images will all be resized to this size.
img_sz = 96
```

In [3]:
```
# Listing all the directories in the folder
os.listdir('/home/himanshu/Desktop/MyFolder/Python/Keras/Project/NonsegmentedV2')
```

Out[3]:
```
['Common wheat',
 'Scentless Mayweed',
 'Cleavers',
 'Shepherds Purse',
 'Loose Silky-bent',
 'Charlock',
 'Maize',
 'Black-grass',
 'Fat Hen',
 'Small-flowered Cranesbill',
 'Sugar beet',
 'Common Chickweed']
```

In [6]:
```
# Create a new directory to store all available images
all_images_dir = 'all_images_dir'
os.mkdir(all_images_dir)
```

In [7]:
```
# This code copies all images from their seperate folders into the same
# folder called all_images_dir, changing the file name at the same time
# as some of the files have similar names
```

```
folder_list = os.listdir
('/home/himanshu/Desktop/MyFolder/Python/Keras/Project/NonsegmentedV2')

for folder in folder_list:

    # create a path to the folder
    path = '/home/himanshu/Desktop/MyFolder/Python/Keras/Project/NonsegmentedV2/'
            + str(folder)

    # create a list of all files in the folder
    file_list = os.listdir(path)

    # move the 0 images to all_images_dir
    for fname in file_list:

        # source path to image
        src = os.path.join(path, fname)

        # Change the file name because many images have the same file name.
        # Add the folder name to the existing file name.
        new_fname = str(folder) + '_' + fname

        # destination path to image
        dst = os.path.join(all_images_dir, new_fname)
        # copy the image from the source to the destination
        shutil.copyfile(src, dst)
```

In [8]:
```
# Get a list of all images in the all_images_dir folder.
image_list = os.listdir('all_images_dir')

# Creating a dataframe of all the images.
df_data = pd.DataFrame(image_list, columns=['image_id'])

df_data.head()
```

Out[8]:

	image_id
0	Common Chickweed_710.png
1	Scentless Mayweed_347.png
2	Scentless Mayweed_78.png
3	Scentless Mayweed_13.png
4	Maize_235.png

In [9]:
```
# Each file name has this king of format: E.g.
# Loose Silky-bent_377.png

# This function will extract the class name from the file name of each image.
def extract_target(x):
    # split into a list
```

```
    a = x.split('_')
    # the target is the first index in the list
    target = a[0]

    return target


# create a new column called 'target'
df_data['target'] = df_data['image_id'].apply(extract_target)

df_data.head()
```

Out[9]:

	image_id	target
0	Common Chickweed_710.png	Common Chickweed
1	Scentless Mayweed_347.png	Scentless Mayweed
2	Scentless Mayweed_78.png	Scentless Mayweed
3	Scentless Mayweed_13.png	Scentless Mayweed
4	Maize_235.png	Maize

In [10]:

```
# Checking the class distribution

df_data['target'].value_counts()
```

Out[10]:

```
Loose Silky-bent             762
Common Chickweed             713
Scentless Mayweed            607
Small-flowered Cranesbill    576
Fat Hen                      538
Sugar beet                   463
Charlock                     452
Cleavers                     335
Black-grass                  309
Shepherds Purse             274
Maize                        257
Common wheat                 253
Name: target, dtype: int64
```

In [11]:

```
# Get a list of classes
target_list = os.listdir
('/home/himanshu/Desktop/MyFolder/Python/Keras/Project/NonsegmentedV2')

for target in target_list:

    # Filter out a target and take a random sample
    df = df_data[df_data['target'] == target].sample(smpl_sz, random_state=101)

    # if it's the first item in the list
    if target == target_list[0]:
        df_sample = df
```

```
            else:
                # Concat the dataframes
                df_sample = pd.concat([df_sample, df], axis=0).reset_index(drop=True)
```

In [13]:
```
# Display the balanced classes.
df_sample['target'].value_counts()
```

Out[13]:
```
Scentless Mayweed            250
Common Chickweed             250
Loose Silky-bent             250
Charlock                     250
Shepherds Purse             250
Fat Hen                      250
Cleavers                     250
Common wheat                 250
Maize                        250
Sugar beet                   250
Black-grass                  250
Small-flowered Cranesbill    250
Name: target, dtype: int64
```

In [14]:
```
# train_test_split

# stratify=y creates a balanced validation set.
y = df_sample['target']

df_train, df_val = train_test_split(df_sample, test_size=0.10, random_state=101
                                    , stratify=y)

print(df_train.shape)
print(df_val.shape)
```

```
(2700, 2)
(300, 2)
```

In [20]:
```
print("\n Train set class distribution")
print(df_train['target'].value_counts())


print("\n Val set class distribution")
print(df_val['target'].value_counts())
```

```
 Train set class distribution
Scentless Mayweed            225
Common Chickweed             225
Loose Silky-bent             225
Charlock                     225
```

```
Shepherds Purse            225
Fat Hen                     225
Cleavers                    225
Common wheat                225
Maize                       225
Sugar beet                  225
Black-grass                 225
Small-flowered Cranesbill   225
Name: target, dtype: int64

 Val set class distribution
Fat Hen                     25
Loose Silky-bent            25
Common wheat                25
Cleavers                    25
Common Chickweed            25
Shepherds Purse            25
Charlock                    25
Sugar beet                  25
Scentless Mayweed           25
Black-grass                 25
Small-flowered Cranesbill   25
Maize                       25
Name: target, dtype: int64
```

In [21]:
```
#Creating a Direcotry Structure
folder_list = os.listdir
('/home/himanshu/Desktop/MyFolder/Python/Keras/Project/NonsegmentedV2')

folder_list
```

Out[21]:
```
['Common wheat',
 'Scentless Mayweed',
 'Cleavers',
 'Shepherds Purse',
 'Loose Silky-bent',
 'Charlock',
 'Maize',
 'Black-grass',
 'Fat Hen',
 'Small-flowered Cranesbill',
 'Sugar beet',
 'Common Chickweed']
```

In [23]:
```
#Checking the direcotry structure
os.listdir('/home/himanshu/Desktop/MyFolder/Python/Keras/Project/NonsegmentedV2')
```

Out[23]:
```
['Common wheat',
 'Scentless Mayweed',
```

```
 'Cleavers',
 'Shepherds Purse',
 'Loose Silky-bent',
 'Charlock',
 'Maize',
 'Black-grass',
 'Fat Hen',
 'Small-flowered Cranesbill',
 'Sugar beet',
 'Common Chickweed']
```

In [24]:
```
# Create a new directory
base_dir = 'base_dir'
os.mkdir(base_dir)


#[CREATE FOLDERS INSIDE THE BASE DIRECTORY]

# now we create 2 folders inside 'base_dir':

# train_dir
    # Maize
    # Fat Hen
    # Shepherds Purse
    # Common Chickweed
    # Cleavers
    # Charlock
    # Loose Silky-bent
    # Small-flowered Cranesbill
    # Black-grass
    # Scentless Mayweed
    # Sugar beet
    # Common wheat

# val_dir
    # Maize
    # Fat Hen
    # Shepherds Purse
    # Common Chickweed
    # Cleavers
    # Charlock
    # Loose Silky-bent
    # Small-flowered Cranesbill
    # Black-grass
    # Scentless Mayweed
    # Sugar beet
    # Common wheat
```

```
# create a path to 'base_dir' to which we will join the names of the new folders
# train_dir
train_dir = os.path.join(base_dir, 'train_dir')
os.mkdir(train_dir)

# val_dir
val_dir = os.path.join(base_dir, 'val_dir')
os.mkdir(val_dir)


# [CREATE FOLDERS INSIDE THE TRAIN AND VALIDATION FOLDERS]

# create new folders inside train_dir

for folder in folder_list:

    folder = os.path.join(train_dir, str(folder))
    os.mkdir(folder)


# create new folders inside val_dir

for folder in folder_list:

    folder = os.path.join(val_dir, str(folder))
    os.mkdir(folder)

# check that the folders have been created

os.listdir('base_dir/train_dir')
```

Out[24]:
```
['Common wheat',
 'Scentless Mayweed',
 'Cleavers',
 'Shepherds Purse',
 'Loose Silky-bent',
 'Charlock',
 'Maize',
 'Black-grass',
 'Fat Hen',
 'Small-flowered Cranesbill',
 'Sugar beet',
 'Common Chickweed']
```

In [25]:
```
# Set the id as the index in df_data
df_data.set_index('image_id', inplace=True)
```

In [26]:
```
df_data.head()
```

```
Out[26]:                                                target
         image_id
         Common Chickweed_710.png    Common Chickweed
         Scentless Mayweed_347.png  Scentless Mayweed
         Scentless Mayweed_78.png   Scentless Mayweed
         Scentless Mayweed_13.png   Scentless Mayweed
         Maize_235.png                          Maize
```

```
In [27]: # Getting a list of train and val images
         train_list = list(df_train['image_id'])
         val_list = list(df_val['image_id'])

         # Transferring the train images

         for image in train_list:

             # the id in the csv file does not have the .tif extension
             # therefore we add it here
             fname = image
             # get the label for a certain image
             folder = df_data.loc[image,'target']


             # source path to image
             src = os.path.join(all_images_dir, fname)
             # destination path to image
             dst = os.path.join(train_dir, folder, fname)

             # resize the image and save it at the new location
             image = cv2.imread(src)
             image = cv2.resize(image, (img_sz, img_sz))
             # save the image at the destination
             cv2.imwrite(dst, image)



         # Transfer the val images

         for image in val_list:

             # the id in the csv file does not have the .tif extension
             # therefore we add it here
             fname = image
             # get the label for a certain image
             folder = df_data.loc[image,'target']


             # source path to image
```

```
            src = os.path.join(all_images_dir, fname)
            # destination path to image
            dst = os.path.join(val_dir, folder, fname)

            # resize the image and save it at the new location
            image = cv2.imread(src)
            image = cv2.resize(image, (img_sz, img_sz))
            # save the image at the destination
            cv2.imwrite(dst, image)
```

2 Starting Modeling

In [28]:
```
train_path = 'base_dir/train_dir'
valid_path = 'base_dir/val_dir'


num_train_samples = len(df_train)
num_val_samples = len(df_val)
train_batch_size = 10
val_batch_size = 10


train_steps = np.ceil(num_train_samples / train_batch_size)
val_steps = np.ceil(num_val_samples / val_batch_size)
```

In [30]:
```
datagen = ImageDataGenerator(rescale=1.0/255)

train_gen = datagen.flow_from_directory(train_path,
                                        target_size=(img_sz,img_sz),
                                        batch_size=train_batch_size,
                                        class_mode='categorical')

val_gen = datagen.flow_from_directory(valid_path,
                                        target_size=(img_sz,img_sz),
                                        batch_size=val_batch_size,
                                        class_mode='categorical')

# Note: shuffle=False causes the test dataset to not be shuffled
test_gen = datagen.flow_from_directory(valid_path,
                                        target_size=(img_sz,img_sz),
                                        batch_size=1,
                                        class_mode='categorical',
                                        shuffle=False)
```

```
Found 2700 images belonging to 12 classes.
Found 300 images belonging to 12 classes.
Found 300 images belonging to 12 classes.
```

3 Creating the model Architecture

In [31]:
```
# Source: https://www.kaggle.com/fmarazzi/baseline-keras-cnn-roc-fast-5min-0-8253-lb

kernel_size = (3,3)
pool_size= (2,2)
first_filters = 32
second_filters = 64
third_filters = 128

dropout_conv = 0.3
dropout_dense = 0.3


model = Sequential()
model.add(Conv2D(first_filters, kernel_size, activation = 'relu',
                 input_shape = (img_sz, img_sz, 3)))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(Conv2D(first_filters, kernel_size, activation = 'relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(Conv2D(second_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(Conv2D(third_filters, kernel_size, activation ='relu'))
model.add(MaxPooling2D(pool_size = pool_size))
model.add(Dropout(dropout_conv))

model.add(Flatten())
model.add(Dense(256, activation = "relu"))
model.add(Dropout(dropout_dense))
model.add(Dense(12, activation = "softmax"))

model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 94, 94, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 92, 92, 32)        9248
_________________________________________________________________
```

```
conv2d_3 (Conv2D)            (None, 90, 90, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 45, 45, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 45, 45, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 43, 43, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 41, 41, 64)        36928
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 39, 39, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 19, 19, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 17, 17, 128)       73856
_________________________________________________________________
conv2d_8 (Conv2D)            (None, 15, 15, 128)       147584
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 13, 13, 128)       147584
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 6, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               1179904
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 12)                3084
=================================================================
Total params: 1,663,756
Trainable params: 1,663,756
Non-trainable params: 0
_________________________________________________________________
```

4 Training Model

In [32]:
```
model.compile(Adam(lr=0.0001), loss='binary_crossentropy',
                      metrics=['accuracy'])
```

In [33]:
```
filepath = "model.h5"
checkpoint = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
```

```
                                 save_best_only=True, mode='max')

reduce_lr = ReduceLROnPlateau(monitor='val_acc', factor=0.5, patience=3,
                                   verbose=1, mode='max', min_lr=0.00001)


callbacks_list = [checkpoint, reduce_lr]

history = model.fit_generator(train_gen, steps_per_epoch=train_steps,
                    validation_data=val_gen,
                    validation_steps=val_steps,
                    epochs=20, verbose=1,
                   callbacks=callbacks_list)
```

```
Epoch 1/20
269/270 [============================>.] - ETA: 0s - loss: 0.2730 - acc: 0.9164
Epoch 00001: val_acc improved from -inf to 0.91694, saving model to model.h5
270/270 [==============================] - 33s 123ms/step - loss: 0.2727 - acc: 0.9165 - val_lo
Epoch 2/20
269/270 [============================>.] - ETA: 0s - loss: 0.2256 - acc: 0.9181
Epoch 00002: val_acc improved from 0.91694 to 0.91861, saving model to model.h5
270/270 [==============================] - 19s 72ms/step - loss: 0.2257 - acc: 0.9181 - val_los
Epoch 3/20
269/270 [============================>.] - ETA: 0s - loss: 0.2025 - acc: 0.9218
Epoch 00003: val_acc did not improve
270/270 [==============================] - 19s 70ms/step - loss: 0.2026 - acc: 0.9218 - val_los
Epoch 4/20
269/270 [============================>.] - ETA: 0s - loss: 0.1909 - acc: 0.9248
Epoch 00004: val_acc improved from 0.91861 to 0.92056, saving model to model.h5
270/270 [==============================] - 20s 72ms/step - loss: 0.1909 - acc: 0.9248 - val_los
Epoch 5/20
269/270 [============================>.] - ETA: 0s - loss: 0.1786 - acc: 0.9293
Epoch 00005: val_acc improved from 0.92056 to 0.92972, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.1782 - acc: 0.9294 - val_los
Epoch 6/20
269/270 [============================>.] - ETA: 0s - loss: 0.1656 - acc: 0.9324
Epoch 00006: val_acc improved from 0.92972 to 0.93278, saving model to model.h5
270/270 [==============================] - 20s 75ms/step - loss: 0.1656 - acc: 0.9324 - val_los
Epoch 7/20
269/270 [============================>.] - ETA: 0s - loss: 0.1541 - acc: 0.9371
Epoch 00007: val_acc improved from 0.93278 to 0.93389, saving model to model.h5
270/270 [==============================] - 20s 72ms/step - loss: 0.1541 - acc: 0.9371 - val_los
Epoch 8/20
269/270 [============================>.] - ETA: 0s - loss: 0.1437 - acc: 0.9423
Epoch 00008: val_acc improved from 0.93389 to 0.94028, saving model to model.h5
270/270 [==============================] - 20s 74ms/step - loss: 0.1437 - acc: 0.9423 - val_los
Epoch 9/20
269/270 [============================>.] - ETA: 0s - loss: 0.1346 - acc: 0.9459
```

```
Epoch 00009: val_acc improved from 0.94028 to 0.94222, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.1345 - acc: 0.9460 - val_los
Epoch 10/20
269/270 [============================>.] - ETA: 0s - loss: 0.1258 - acc: 0.9479
Epoch 00010: val_acc improved from 0.94222 to 0.94528, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.1257 - acc: 0.9479 - val_los
Epoch 11/20
269/270 [============================>.] - ETA: 0s - loss: 0.1144 - acc: 0.9522
Epoch 00011: val_acc improved from 0.94528 to 0.94611, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.1144 - acc: 0.9521 - val_los
Epoch 12/20
269/270 [============================>.] - ETA: 0s - loss: 0.1071 - acc: 0.9546
Epoch 00012: val_acc did not improve
270/270 [==============================] - 20s 73ms/step - loss: 0.1071 - acc: 0.9546 - val_los
Epoch 13/20
269/270 [============================>.] - ETA: 0s - loss: 0.0996 - acc: 0.9579
Epoch 00013: val_acc improved from 0.94611 to 0.94861, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.0995 - acc: 0.9579 - val_los
Epoch 14/20
269/270 [============================>.] - ETA: 0s - loss: 0.0945 - acc: 0.9599
Epoch 00014: val_acc did not improve
270/270 [==============================] - 20s 73ms/step - loss: 0.0944 - acc: 0.9600 - val_los
Epoch 15/20
269/270 [============================>.] - ETA: 0s - loss: 0.0865 - acc: 0.9648
Epoch 00015: val_acc improved from 0.94861 to 0.95056, saving model to model.h5
270/270 [==============================] - 20s 74ms/step - loss: 0.0863 - acc: 0.9649 - val_los
Epoch 16/20
269/270 [============================>.] - ETA: 0s - loss: 0.0814 - acc: 0.9656
Epoch 00016: val_acc improved from 0.95056 to 0.95139, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.0816 - acc: 0.9655 - val_los
Epoch 17/20
269/270 [============================>.] - ETA: 0s - loss: 0.0761 - acc: 0.9686
Epoch 00017: val_acc did not improve
270/270 [==============================] - 20s 75ms/step - loss: 0.0761 - acc: 0.9686 - val_los
Epoch 18/20
269/270 [============================>.] - ETA: 0s - loss: 0.0714 - acc: 0.9699
Epoch 00018: val_acc did not improve
270/270 [==============================] - 20s 73ms/step - loss: 0.0713 - acc: 0.9699 - val_los
Epoch 19/20
269/270 [============================>.] - ETA: 0s - loss: 0.0645 - acc: 0.9732
Epoch 00019: val_acc improved from 0.95139 to 0.95500, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.0644 - acc: 0.9732 - val_los
Epoch 20/20
269/270 [============================>.] - ETA: 0s - loss: 0.0602 - acc: 0.9745
Epoch 00020: val_acc improved from 0.95500 to 0.95556, saving model to model.h5
270/270 [==============================] - 20s 73ms/step - loss: 0.0601 - acc: 0.9746 - val_los
```

5 Evaluate Model using the Val Set

In [35]:
```
model.metrics_names
```

Out[35]:
```
['loss', 'acc']
```

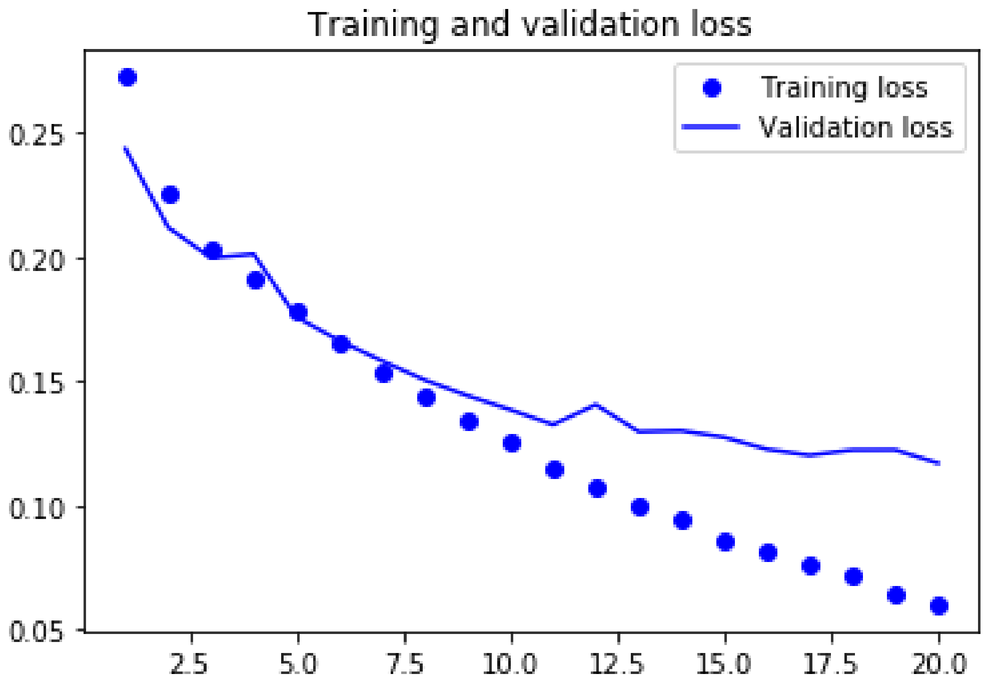
In [36]:
```
# Print the validation loss and accuracy.

# Here the best epoch will be used.
model.load_weights('model.h5')

val_loss, val_acc = \
model.evaluate_generator(test_gen,
                        steps=len(df_val))

print('val_loss:', val_loss)
print('val_acc:', val_acc)
```

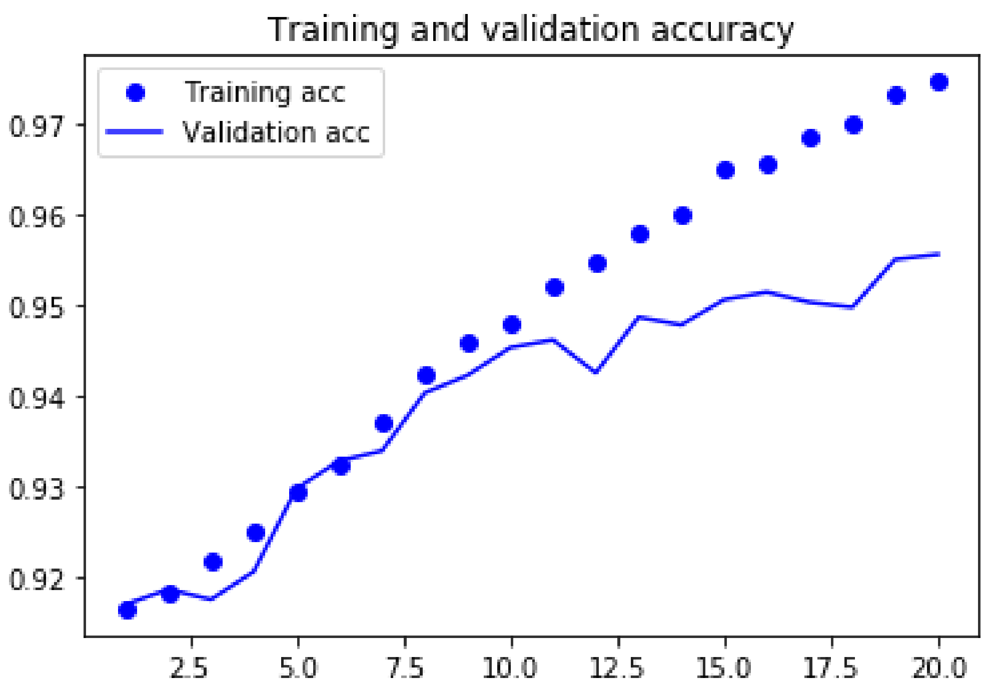
```
val_loss: 0.11705364881115633
val_acc: 0.955555555621783
```

6 Plot the Training Curve

In [37]:
```
# display the loss and accuracy curves

import matplotlib.pyplot as plt

acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1, len(acc) + 1)

plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.figure()

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.legend()
plt.figure()
```

Out[37]:
```
<Figure size 432x288 with 0 Axes>
```






```
<Figure size 432x288 with 0 Axes>
```

7 Make a prediction on the val set

We need these predictions to print the Confusion Matrix and calculate the F1 score.

In [38]:
```
# make a prediction
predictions = model.predict_generator(test_gen, steps=len(df_val), verbose=1)
```

```
300/300 [==============================] - 1s 4ms/step
```

In [39]:
```
predictions.shape
```

Out[39]:
```
(300, 12)
```

In [40]:
```
# This is how to check what index keras has internally assigned to each class.
test_gen.class_indices
```

Out[40]:
```
{'Black-grass': 0,
 'Charlock': 1,
 'Cleavers': 2,
 'Common Chickweed': 3,
 'Common wheat': 4,
 'Fat Hen': 5,
 'Loose Silky-bent': 6,
 'Maize': 7,
 'Scentless Mayweed': 8,
 'Shepherds Purse': 9,
 'Small-flowered Cranesbill': 10,
 'Sugar beet': 11}
```

In [41]:
```
# Put the predictions into a dataframe.
# The columns need to be ordered to match the output of the previous cell

class_dict = train_gen.class_indices

# Get a list of the dict keys.
cols = class_dict.keys()

df_preds = pd.DataFrame(predictions, columns=cols)

df_preds.head()
```

Out[41]:

	Black-grass	Charlock	Cleavers	Common Chickweed	Common wheat	Fat Hen \
0	0.182492	0.000369	0.005700	0.000122	0.593382	0.013446
1	0.468289	0.000008	0.000479	0.000002	0.160517	0.014424
2	0.082417	0.000829	0.412537	0.000011	0.005066	0.303426
3	0.000738	0.000164	0.002487	0.003655	0.000009	0.002779
4	0.428237	0.000036	0.000265	0.000012	0.120407	0.000982

```
   Loose Silky-bent     Maize  Scentless Mayweed  Shepherds Purse  \
0          0.199016  0.001122           0.002284         0.000166
1          0.355947  0.000012           0.000113         0.000001
2          0.161298  0.000003           0.014975         0.000003
3          0.000249  0.000203           0.000005         0.000152
4          0.448889  0.000179           0.000566         0.000009

   Small-flowered Cranesbill  Sugar beet
0                   0.000755    0.001146
1                   0.000051    0.000157
2                   0.011455    0.007981
3                   0.857591    0.131968
4                   0.000067    0.000349
```

8 Creating a Confusion Matrix

In [42]:
```python
# Get the labels of the test images.

test_labels = test_gen.classes
```

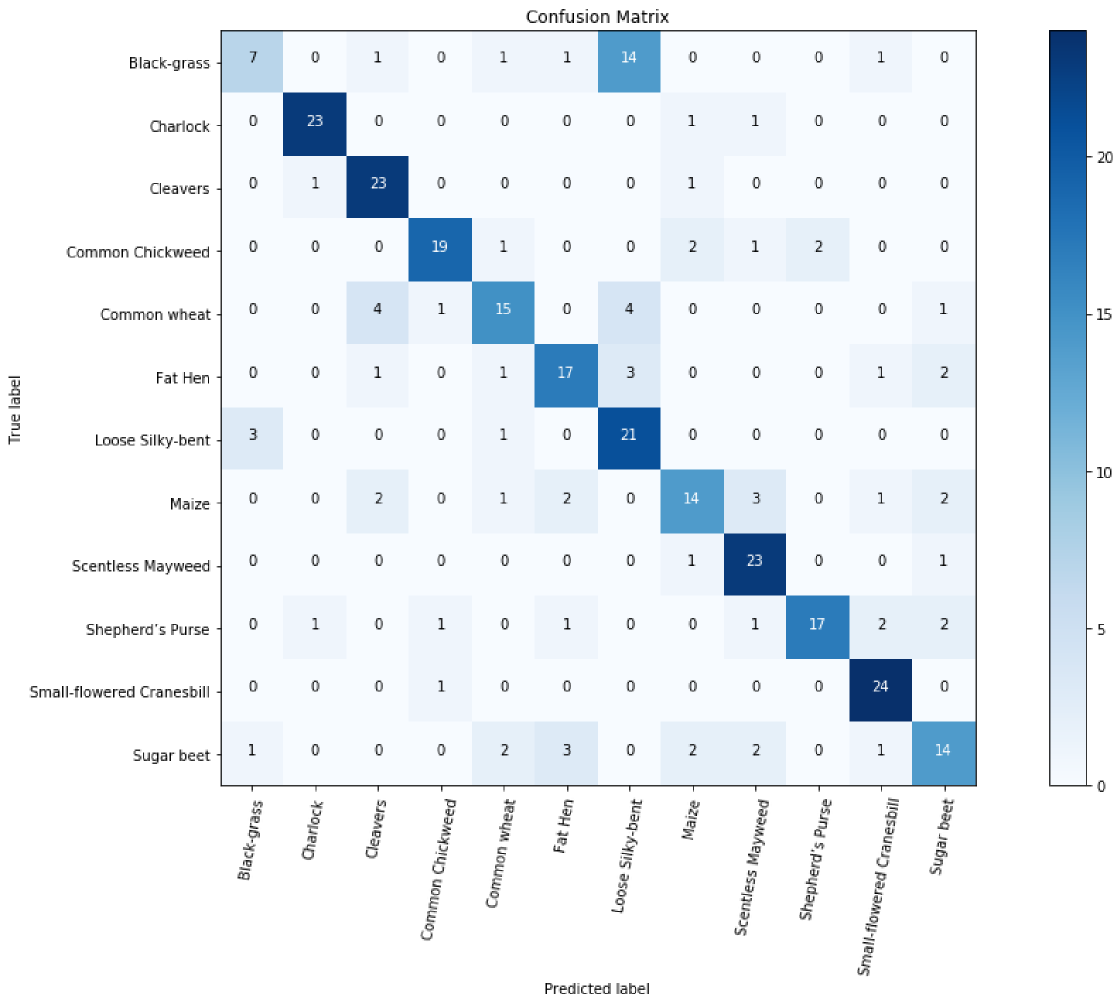
In [43]:
```python
# Source: Scikit Learn website
# http://scikit-learn.org/stable/auto_examples/
# model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-
# selection-plot-confusion-matrix-py


def plot_confusion_matrix(cm, classes,
                          normalize=False,
                          title='Confusion matrix',
                          cmap=plt.cm.Blues):
    """
    This function prints and plots the confusion matrix.
    Normalization can be applied by setting `normalize=True`.
    """
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    # set the size of the figure here
    plt.figure(figsize=(15,10))

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
```

```
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=80) # set x-axis text angle here
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.tight_layout()
```

In [44]:
```
# argmax returns the index of the max value in a row
cm = confusion_matrix(test_labels, predictions.argmax(axis=1))
```

In [45]:
```
# Define the labels of the class indices. These need to match the
# order shown above.
cm_plot_labels = cols

plot_confusion_matrix(cm, cm_plot_labels, title='Confusion Matrix')
```

```
Confusion matrix, without normalization
[[ 7  0  1  0  1  1 14  0  0  0  1  0]
 [ 0 23  0  0  0  0  0  1  1  0  0  0]
 [ 0  1 23  0  0  0  0  1  0  0  0  0]
 [ 0  0  0 19  1  0  0  2  1  2  0  0]
 [ 0  0  4  1 15  0  4  0  0  0  0  1]
 [ 0  0  1  0  1 17  3  0  0  0  1  2]
 [ 3  0  0  0  1  0 21  0  0  0  0  0]
 [ 0  0  2  0  1  2  0 14  3  0  1  2]
 [ 0  0  0  0  0  0  0  1 23  0  0  1]
 [ 0  1  0  1  0  1  0  0  1 17  2  2]
 [ 0  0  0  1  0  0  0  0  0  0 24  0]
 [ 1  0  0  0  2  3  0  2  2  0  1 14]]
```




```
In [46]: from sklearn.metrics import classification_report

         # Generate a classification report

         # Get the true labels
         y_true = test_gen.classes

         # For this to work we need y_pred as binary labels not as probabilities
         y_pred_binary = predictions.argmax(axis=1)

         report = classification_report(y_true, y_pred_binary, target_names=cm_plot_labels)

         print(report)

                        precision    recall  f1-score   support
```

Black-grass	0.64	0.28	0.39	25
Charlock	0.92	0.92	0.92	25
Cleavers	0.74	0.92	0.82	25
Common Chickweed	0.86	0.76	0.81	25
Common wheat	0.68	0.60	0.64	25
Fat Hen	0.71	0.68	0.69	25
Loose Silky-bent	0.50	0.84	0.63	25
Maize	0.67	0.56	0.61	25
Scentless Mayweed	0.74	0.92	0.82	25
Shepherds Purse	0.89	0.68	0.77	25
Small-flowered Cranesbill	0.80	0.96	0.87	25
Sugar beet	0.64	0.56	0.60	25
micro avg	0.72	0.72	0.72	300
macro avg	0.73	0.72	0.71	300
weighted avg	0.73	0.72	0.71	300