Flowers Recognition

Rémi Ang
Udacity.com - Machine Learning Engineering Nano Degree

May 27, 2018

1 Definition

1.1 Project Overview

Object recognition is a central topic in the field of computer vision. The numerous applications can be applied in many different domains such as self-driving vehicles environment detection, skin cancer screening or reality augmented tourism.

This project focus on the automatic recognition of flower species from pictures. The potential applications are:

- population tracking and preservation
- crop and food supply management
- toxicity detection
- education
- and more...

Figure 1: application example: it's a Daisy!

1.2 Problem Statement

Is it possible to accurately identify the variety of a flower from a picture ?

The today state of the art solution to tackle pictures classification problems is the creation and training of a deep Convolution Neural Network (CNN). The training has to be performed on a dataset of flower pictures classified by species.

1.3 Dataset

The dataset used in this project is the "Flowers Recognition Dataset" of Kaggle.com given by Alexander Mamaev.

The zip archive is 230MB and contains about 4300 flower images (.jpg) classified in 5 classes: Daisy, Dandelion, Rose, Sunflower and Tulip.

1.4 metrics

The evaluate our classification model is based on the *Accuracy ACC* and *Classification Cross-Entropy* $H(y, \hat{y})$ defined as follows:

$$ACC = \frac{number\ of\ correct\ predictions}{total\ number\ of\ prediction} \cdot 100 \text{ [2]}$$

$$H(y, \hat{y}) = -\sum_{k=1} y_i log(\hat{y}_i) \text{ [4]}$$

2 Analysis

2.1 Data Exploration and Visualization

The pictures have been automatically scraped from the online picture portals Flickr, Google Images and Yandex Images.

They represent single or multiple flowers under various angles, zoom, brightness conditions, life cycle stages, etc.

Figure 2: samples pictures

The complete set of pictures distribution is as in Fig. 3:

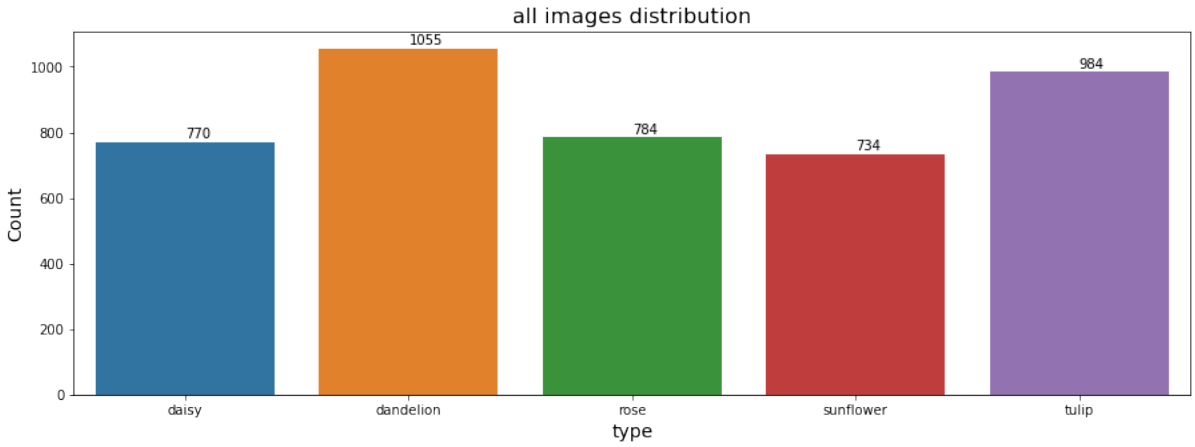

Figure 3: All images distribution by label

We observe that the classes distribution is slightly unbalanced. There is an average of 865 pictures per classes to train the model, which is reasonably enough. The *dandelion* class is the most represented with 1055 pictures. However, this flower variety has the particularity to be represented in 2 radically distinct state: the yellow flowers and the white "blow balls". For this particular case, it is beneficial to have more sample to expect the model to be able to recognize both *dandelion*'s aspects.

Nevertheless, a significant number of pictures contains other subjects (persons, insects, other flowers, vase) or seem to be miss-classified. Such pictures can potentially impact negatively the model learning. A more detailed approach of the samples selection is described in chapter 3.1: Data Preprocessing

2.2 Algorithms and Techniques

2.2.1 pictures selection and split

As already mentioned in the previous chapter, a large number of pictures seems inappropriate to train a model. A "blacklisting" approach will be put in place to discard them from the dataset.

On top of that, the selected pictures will be randomly split into three distinct training, validation and testing datasets. The random splitting is made in a stratified fashion in order to preserve the initial classes distribution in each set.

2.2.2 image augmentation

Image augmentation techniques will be used to enrich the training and validation sets.

2.2.3 Deep Convolutional Neural Network and transfer learning

A deep CNN has to be trained to recognize flowers variety from pictures. Rather than designing and training a full new CNN, we will take advantage of existing reference models by applying transfer learning. The weights of all convolutional layers of the base model will be loaded and frozen as-is. The top layers will be replaced by some of our own in order to specialize the mode to flowers types recognition.

An important part of this project will be dedicated to determine the most suitable base model to use.

2.3 Benchmark

The benchmark for this project is the "Flowers Species Recognition" work from Yuning Chai, Victor Lempitsky and Andrew Zisserman from the University of Oxford in 2011 [5, 6] .

This two-steps approach consist of the segmentation of the pictures and the training of a kernelized SVM classifier. The best model reach a prediction accuracy of 80.0% and is able to recognize flowers among 102 different variety.

The goal of this project is to reach at least this accuracy, however only 5 flower species are considered.

3 Methodology

3.1 Data Preprocessing

3.1.1 Pictures scan

3.1.2 Sample blacklist

As already mentioned in 2.1: Data Exploration and Visualization, a significant amount of pictures in the dataset are likely to negatively impact the learning of the mode. Our flower recognition model is intended to be used in an application where the images are supposed be focused on one or few flowers of the same type, from a close to middle-range distance (1 to 10 meters), in natural colours.

The black-listing of pictures aim to discard samples:

- which don't have a ".jpg" extension (some python web scrapping routine can be found in the *dandelion* subfolder)
- not representing a flower
- wrongly classified
- containing other subjects (persons, insects, objects,...)
- close ups
- wide shots with fields, mountains, landscapes...
- drawings
- with artistic filters significantly denaturing the original colour

After the manual visualization of all samples, the blacklist contains 1059 items (24.5% of the total dataset) stored in a text file (`blacklist.txt`). The data loading implementation read this file and discards any file found in the list.

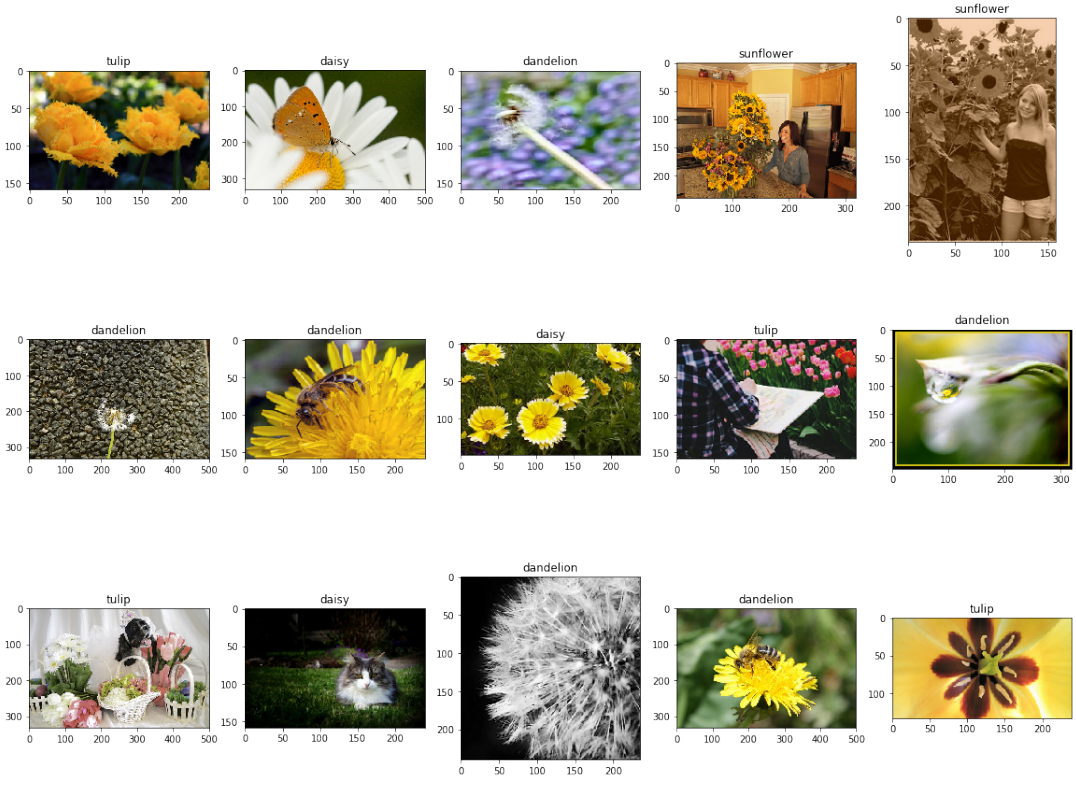

Figure 4: samples of black listed pictures

Python source code 1: picture file scan and filtering

```
from sklearn.datasets import load_files

path = "./flowers"

blacklist_file = "blacklist.txt" # list of files to be removed drm the dataset

#read blacklist
with open(blacklist_file, 'r') as f:
        blacklist = f.readlines()
blacklist = [path + s.strip() for s in blacklist]

# scan all files stored in the data folder
data = load_files(path, load_content=False)
all_flowers_files = np.array([s.replace('\\', '/') for s in data["filenames"]])

# identify blacklist indexes
isvalid_file = [True if f not in blacklist else False for f in all_flowers_files]
flowers_files = all_flowers_files[isvalid_file]

# flowers_targets = np_utils.to_categorical(data["target"],5)
flowers_targets = data["target"][isvalid_file]
flowers_target_names = data["target_names"]
```

After filtering, we observe that the new classes distribution is comparable to the original one. The classes are ow represented by 654 samples on average:

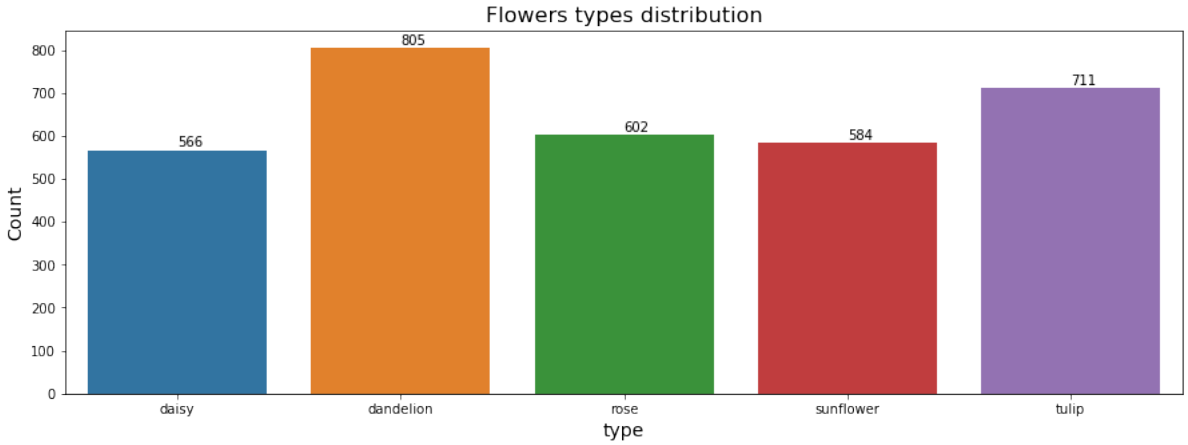

Figure 5: Images distribution after blacklist filtering

3.1.3 Training, validation and testing split

The dataset is randomly split in 3 parts using the `train_test_split` from `sklearn.model_selection` package, using the `stratify` option in order to preserve the original classes distribution in each split:

- Training (76.5% - 2500 samples)
- Validation (8.5% - 278 samples)
- Testing (15% - 491 samples)

Python source code 2: picture files train/valid/test split

```
from sklearn.model_selection import train_test_split

# train+valid / test split
test_size = .1

flowers_files_train_valid, flowers_files_test, flowers_targets_train_valid, flowers_t
flowers_files, flowers_targets,
test_size = test_size,
stratify=flowers_targets)

# train / valid split
valid_size = .2
flowers_files_train, flowers_files_valid, flowers_targets_train, flowers_targets_vali
flowers_files_train_valid, flowers_targets_train_valid,
test_size = valid_size,
stratify=flowers_targets_train_valid)
```

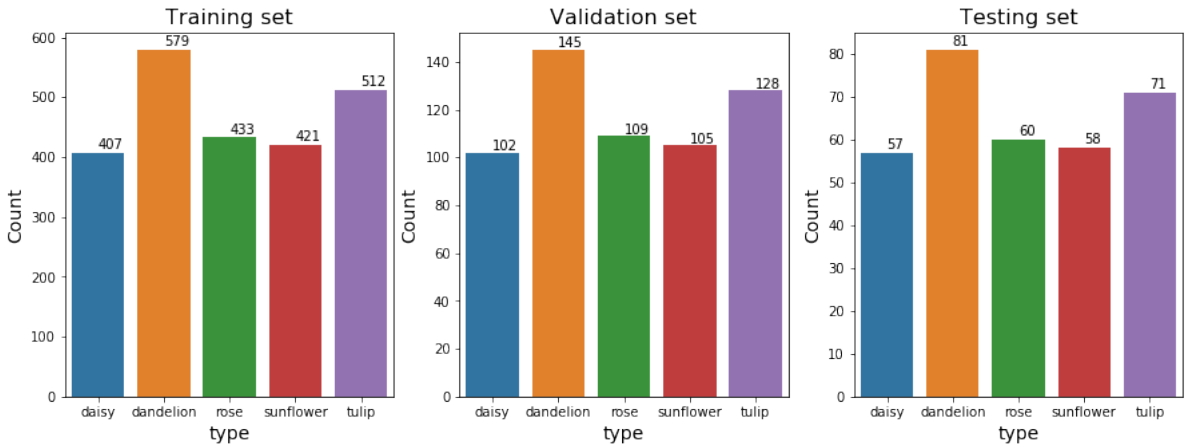

Figure 6: Stratified images distributions after train/valid/test split

3.1.4 Dataset loading in memory

The images are loaded in RBG colors and resized in a 299x299 shape using OpenCV "`cv2`" python package. Targets labels are one-hot-encoded using the `to_categorical` method from the `keras.utils.np_utils` package.

Python source code 3: Image loading with openCV

```
from keras.utils import np_utils
from tqdm import tqdm
import cv2

def loadimg(path):
        """
        load and resize an image
        return a (299, 299, 3) array
        """
        img = cv2.imread(path)
        img_r = cv2.resize(img, (299, 299))
        return img_r

x_train = np.array([loadimg(f) for f in tqdm(flowers_files_train)]).reshape(-1, 299,
y_train =  np_utils.to_categorical(flowers_targets_train)

x_valid = np.array([loadimg(f) for f in tqdm(flowers_files_valid)]).reshape(-1, 299,
y_valid =  np_utils.to_categorical(flowers_targets_valid)

x_test = np.array([loadimg(f) for f in tqdm(flowers_files_test)]).reshape(-1, 299, 29
y_test =  np_utils.to_categorical(flowers_targets_test)
```

3.1.5 Image augmentation

In order to enrich the dataset, image augmentation is applied to the training and validation datasets using the `ImageDataGenerator` utility from the `keras.preprocessing.image` package. This step is also used to normalize the features: all values are divided by 255. No image augmentation is applied on the testing set which is meant to represent real case images for the model evaluation.

Python source code 4: Augmented image generators

```
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
rescale=1./255,
shear_range=0.2,
zoom_range=0.2,
horizontal_flip=True)
```

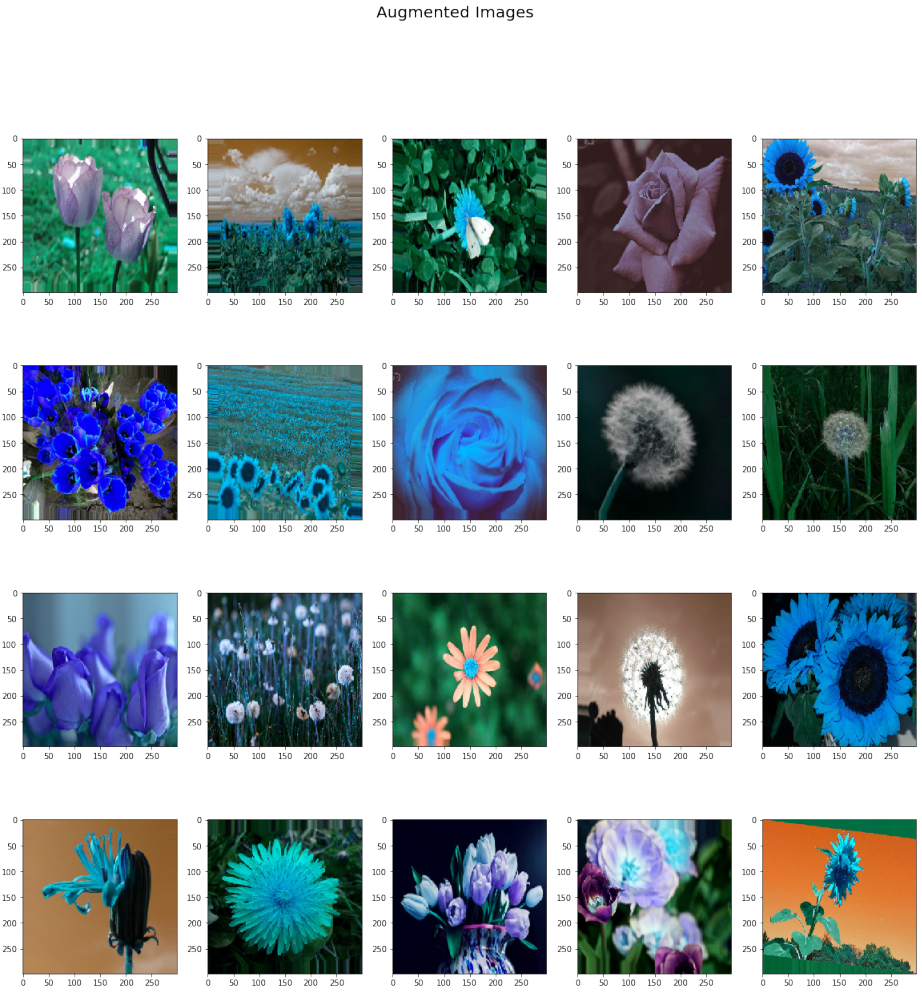

Figure 7: samples of augmented training images

3.2 Model Implementation

3.2.1 CNN architecture

The targeted CNN architecture is the combination of a fixed weights base model and custom top layers specialized in the prediction of flower varieties.

The flower recognition CNN final architecture has been implemented as following:

- Base model
- Flatten layer
- Dense layer, 500 units, ReLu activation
- Dropout layer, 0.3 drop probability
- Output layer: Dense layer, 5 units, Softmax activation
- Optimizer: Adam with learning rate $\alpha = 1e^{-}3$ and decay $d = 1e^{-}6$
- loss function: categorical cross-entropy
- metrics : accuracy

Python source code 5: construction of a CNN with VGG16 as base model

```
from keras.applications.vgg16 import VGG16
from keras.layers import Flatten, Dense, Dropout
from keras.models import Model
from keras.optimizers import Adam

# load the base model, do not include top layers
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(299,299,3))

# freeze the layers of the base model
for layer in base_model.layers:
        layer.trainable = False

# top layers:
x = Flatten()(base_model.output)
x = Dense(500, activation='relu', name='fc1')(x)
x = Dropout(0.3)(x)
x = Dense(5, activation='softmax', name='fc2')(x)

model = Model(inputs=base_model.input, outputs=x)

# set up the optimizer
opt = Adam(lr = 1e-3, decay=1e-6)

# compile the model
model.compile(loss = 'categorical_crossentropy', optimizer=opt, metrics=['accuracy'])
```

This architecture is inspired by the solution proposed by A. Akashnian in his "Flowers are mesmerizing" notebook published on Kaggle.com [3].

3.2.2 CNN training

The model is trained with the augmented pictures generator specified in 3.1.5: Image augmentation, on 50 epochs with batches of 32 samples (= 73 steps/epoch). The model weights are saved via a `ModelCheckpoint` each time that the validation loss each a new minima. The training historic is stored in order to later analyse the accuracies and losses evolution during the training. In order to speed-up the processing time, the model has been trained on an Amazon Web Services GPU instance (50 to 60 seconds per epochs).

Python source code 6: model training

```
from keras.callbacks import ModelCheckpoint

batch_size = 32
epochs = 50
saved_models_dir = 'saved_models'

#if it doesn't exist, create the saved model directory
if not os.path.isdir(saved_models_dir):
        os.mkdir(saved_models_dir)

# define the model weight h5 storage
best_weights_h5 = os.path.join(saved_models_dir,
        'weights.best.flowers_recognition.{}.hdf5'.format(base_model_name))

# model checkpointer
checkpointer = ModelCheckpoint(filepath=best_weights_h5,
        verbose=1, save_best_only=True)

# training
training_hist = model.fit_generator(train_datagen.flow(x_train, y_train, batch_size=b
        steps_per_epoch=x_train.shape[0] // batch_size,
        epochs=epochs,
        verbose=1,
        callbacks=[checkpointer],
        validation_data=valid_datagen.flow(x_valid, y_valid, batch_size=batch_size),
        validation_steps=x_valid.shape[0] // batch_size)
```

3.3 Model Refinement

3.3.1 Base Model selection

The `keras.applications` package offers some pre-trained models. In frame of this project, the VGG16, VGG19 and Resnet50 based models have been individually tested on 10 epochs.

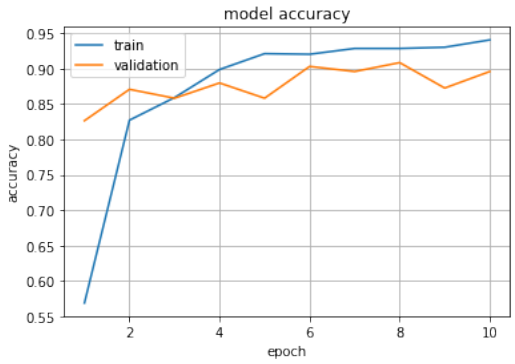

Figure 8: VGG16 - 10 epochs Accuracy

Figure 9: VGG16 - 10 epochs Loss

According to Fig. 8 and 9, VGG16 based model seems to learn extremely fast after the first back-propagation at epoch 1. It then constantly improves during the next epochs. After 10 epochs, the best validation loss achieved is 0.2549 and the accuracy 0.91 at epoch 8. This base model seems to be a good candidate for our flowers classifier.
— VGG19

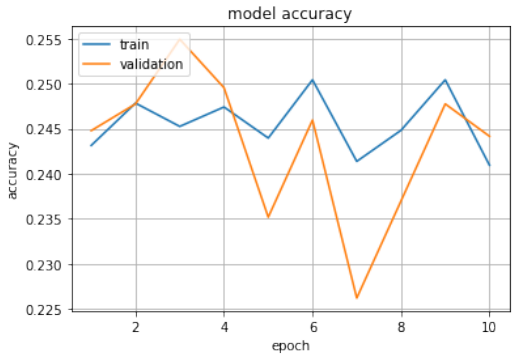

Figure 10: VGG19 - 10 epochs Accuracy

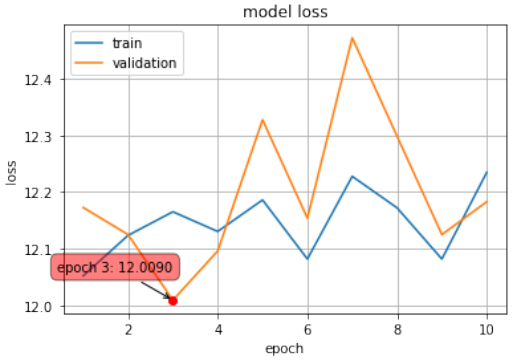

Figure 11: VGG19 - 10 epochs Loss

According to Fig. 10 and 11, the VGG19 based model doesn't seem able to learn. The average training accuracy of 24% is comparable to random guesses performance. This base model will not be considered for the flower classifier.

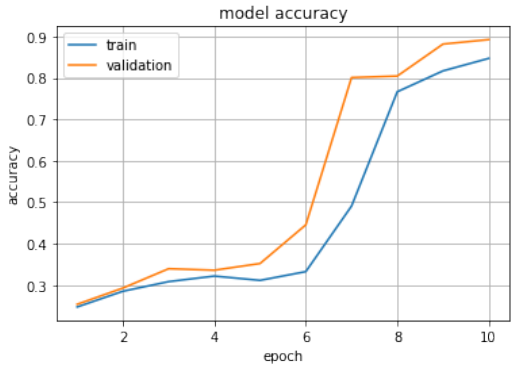

Figure 12: Resnet50 - 10 epochs Accuracy

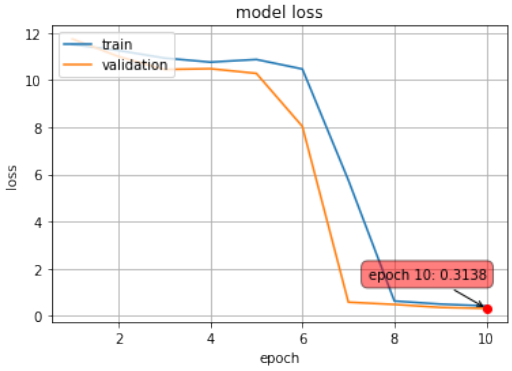

Figure 13: Resnet50 - 10 epochs Loss

According to Fig. 12 and 13, it takes up to 7 epochs for this model to achieve reasonably good performances. It reach its best validation loss at the 10th epoch with 0.3138 and an accuracy of nearly 90%, letting suggest that it could even more improve with longer training.

base model selection - conclusion:

The best performing flower classifier on 10 epochs is the **VGG16 based model**. Therefore, it has been selected to perform a full training on 50 epochs.

4 Results

The VGG16 based flower classifier is trained on 50 epochs (about 45 minutes on GPU instance). According to Fig. 14 and 15, the model achieve the best validation loss at epoch 33 with 0.2122. The validation accuracy stagnate at 90% after the 10th epoch.

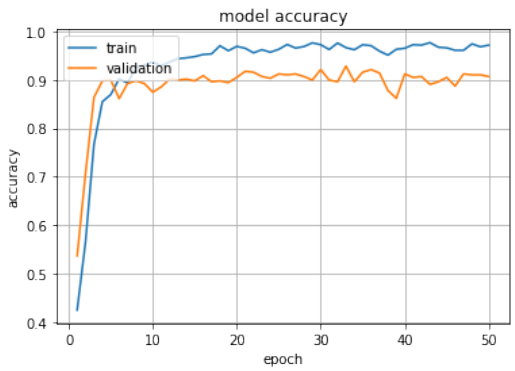

Figure 14: VGG16 - 50 epochs Accuracy

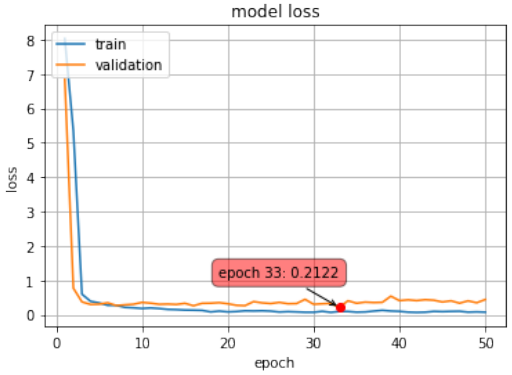

Figure 15: VGG16 - 50 epochs Loss

4.1 Model Accuracy

The model accuracy is compute by loading the best loss weights, scaling the test set and evaluating the model with its `evaluate` method on the scaled test dataset:
The VGG16 based flower classifier achieve an accuracy of **89.30%** on the testing set.

Python source code 7: best weights loading and evaluation of the model accuracy

```
base_model_name = 'VGG16'
best_weights_h5 = 'saved_models/weights.best.flowers_recognition.{}.hdf5'.format(base

# load best loss weights
model.load_weights(best_weights_h5)

# scale test dataset
xtest_scaled = x_test.astype('float32')/255

# evaluate model accuracy
score = model.evaluate(xtest_scaled , y_test, verbose=1)

# display accuracy in stdout
print('\nTest accuracy: {:.2f}%'.format(score[1] * 100))
```

4.2 Justification

The predictions are evaluated for each sample of the test dataset:

Python source code 8: test pictures prediction

```
import numpy as np

predictions = np.array([np.argmax(model.predict(np.expand_dims(x, axis=0))) \
        for x in tqdm(xtest_scaled)])
```

The confusion matrix displayed in Fig. 16 confirms the good performance of the model.

- 80.7 % of the test *daisies* are correctly classified. The type of flower is mostly confused by the model by *dandelions*.
- 92.6 % of the test *dandelions* are correctly classified. This type of flower is mostly confused by the model with *sunflowers*. This remarquable score demonstrate that the model is able to identify the *dandelions* both in yellows flower state and white "blow-balls" (see discussion in Chapter 2.1: Data Exploration and Visualization).
- 85.0 % of the test *roses* are correctly classified. This type of flower is mostly confused by the model with *tulips*.
- 94.8 % of the test *sunflowers* are correctly classified. This the most accurately predicted type of flower.
- 91.5 % of the test *tulips* are correctly classified. This type of flower is mostly confused by the model with *roses*.

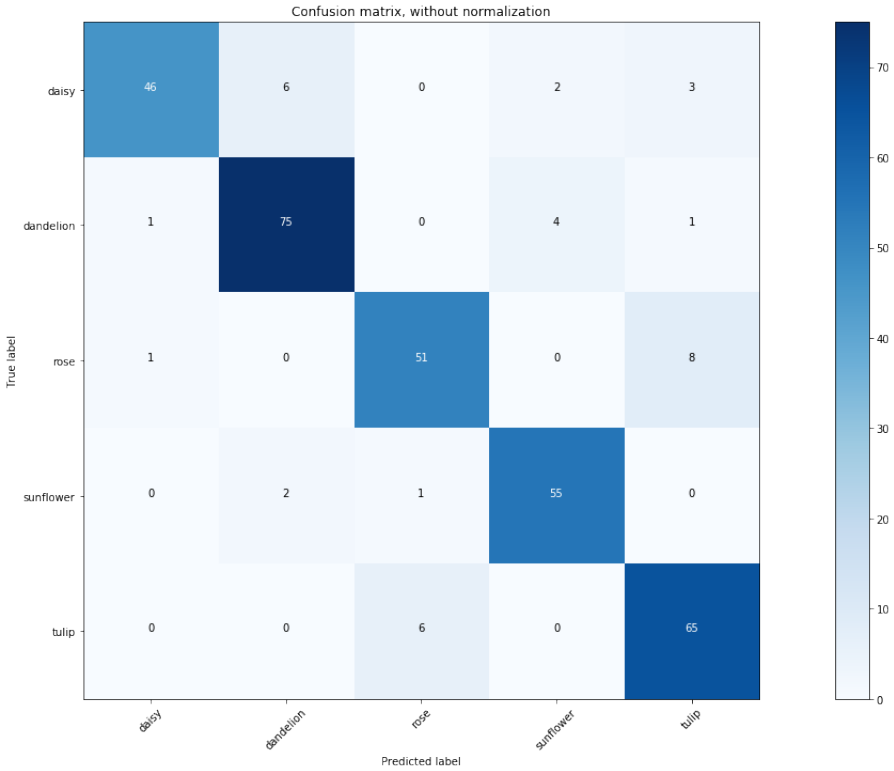

Figure 16: test pictures predictions - confusion matrix

5 Conclusion

5.1 Free-Form Visualization

5.1.1 Test samples prediction

A sample of successful tests predictions is displayed in Fig. 17 with the predicted label and the true label shown in the title. Those samples demonstrate that the model is able to identify flowers from different angles, various life state and to perform the identification with a variable amount of flowers (one to some dozens). The picture on middle row right show a correctly identified *dandelion* with only two seeds on the bulb!

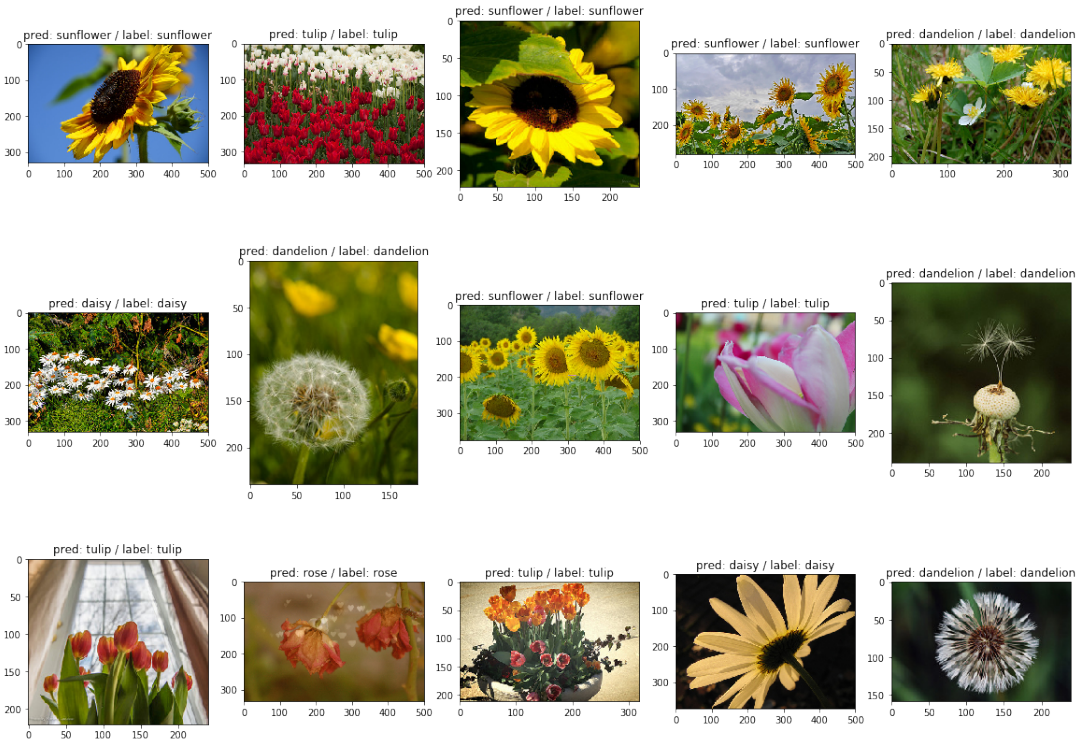

Figure 17: Sample of test picture flower recognition

5.1.2 Failed prediction

All the test samples that the model failed to identify correctly are display on Fig. 18.

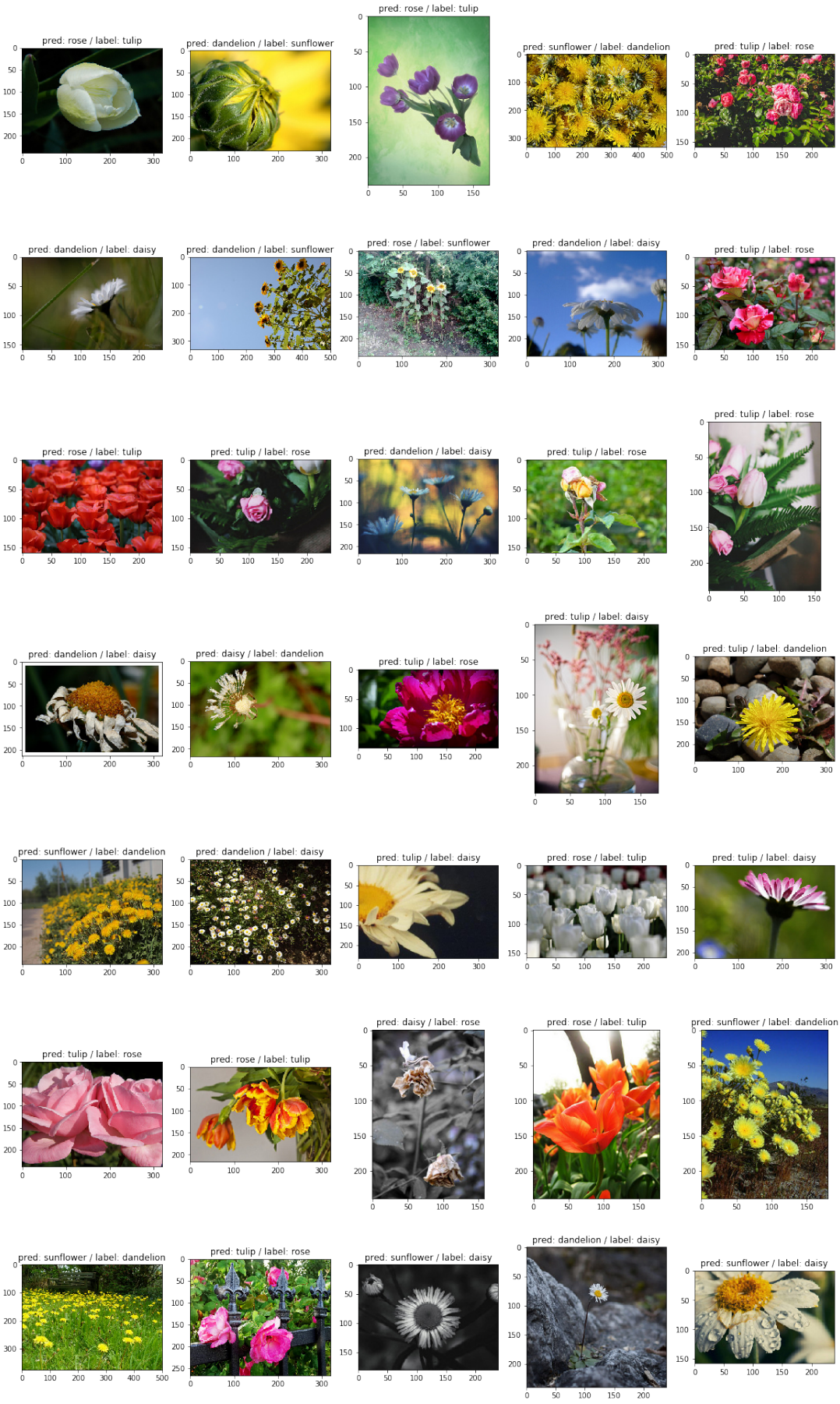

Figure 18: failed predictions in the test dataset

5.2 Reflection

The process used for this project can be summarized as follow:

1. Find a relevant/challenging problem to solve
2. find already existing work(s) in order to set the benchmark
3. Find a dataset suitable for answering this problem
4. Clean up and pre-process the dataset to make it training-ready
5. Test different base model for the CNN and select the best performing one
6. Train the final chosen CNN in the cloud
7. Evaluate the model and analyse the results

One particular difficult and tedious task has been the manual selection of samples to keep or remove from the dataset. This work is hardly automatable and had to be performed case by case, based on arbitrary rules and intuition.
I have been positively surprised by the differences in performance between the three based models tested. According to the references, VGG16 is the model among the three having the lowest accuracy on the ImageNet validation dataset [1]. However, this exercise comforted me in necessity to try different approaches.

5.3 Improvement

The main improvement for this project would be to increase the amount of flower classes. It would be particularly interesting to train the same CNN on the 100+ flowers types used by the BiCos-SVM model [6]. It would also worth trying other base models that have not been considered in this study. It would be beneficial to deeper analyse prediction failure cases in order to identify failures root cause and adapt accordingly the dataset or the model if feasible. Finally, in order to deploy the model in real life, it would be necessary to develop a mobile application able to directly process a picture taken by the device and return to the user a prediction.

References

[1] Keras, applications. `https://keras.io/applications/`.

[2] Scikit-learn, model evaluation, accuracy score. `http://scikit-learn.org/stable/modules/model_evaluation.html#accuracy-score`.

[3] A. Akashnain. Flowers are mesmerizing.

[4] Rob Di Pietro. A friendly introduction to cross entropy loss. `https://rdipietro.github.io/friendly-intro-to-cross-entropy-loss/`, May 2, 2016.

[5] Andrew ZISSERMAN Yuning CHAI, Victor LEMPITSKY. Bicos: A bi-level co-segmentation method for image classification. *ICCV*, 2011.

[6] Andrew ZISSERMAN Yuning CHAI, Victor LEMPITSKY. Flowers species recognition demo. `http://www.robots.ox.ac.uk/%7Evgg/research/flowers_demo/`, 2011.