Disease detection from photos with small dataset with transfer learning

Romain Gautron
r.gautron@cgiar.org

March 8, 2019

We will work with keras library (tensorflow backend). In order to have reproducible results, include at the top of your code:

```
import numpy
numpy.random.seed(123)
from tensorflow import set_random_seed
set_random_seed(123)
```

We will need as well the `matplotlib` and `opencv-python` packages.

0 Overview

We will use transfer learning to learn how to discriminize photos of plant disease. To learn from very small dataset for that complicated task, we will use transfer learning.

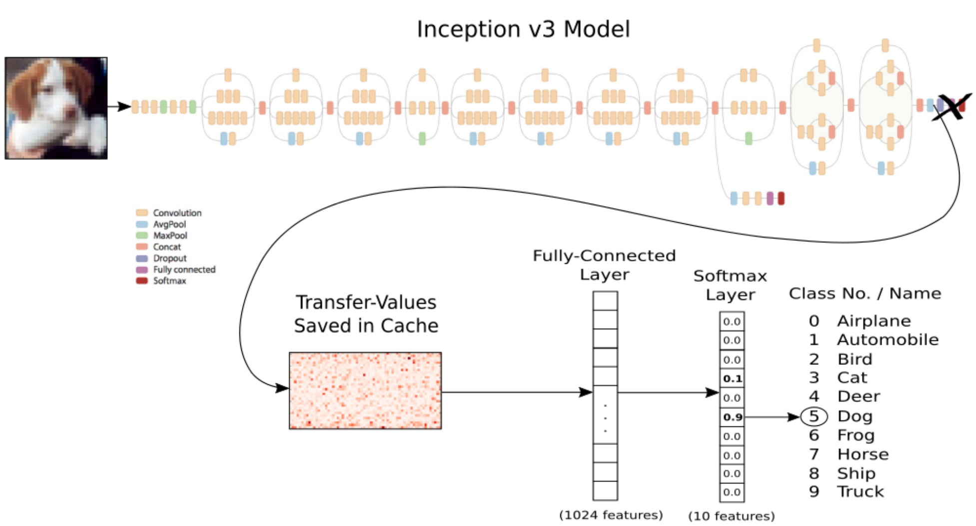

Figure 1: Principle (courtesy Hvass Labs)

1 Setup

1.1 Packages to load

Copy the `plotActivation.py` file from Github in your local folder and put at top following lines.

```
import matplotlib.pyplot as plt
import keras.backend as K
K.set_image_data_format('channels_last') # insures right format
from keras.models import Model, load_model
from keras.applications.xception import Xception
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import RMSprop
from keras.models import Sequential
from keras.layers import Dropout, Flatten, Dense, Input,\
BatchNormalization, MaxPooling2D, Conv2D
from keras.callbacks import EarlyStopping,\
ModelCheckpoint, ReduceLROnPlateau
from keras.models import Model
from sklearn.metrics import confusion_matrix
import plotActivation
```

Import as well your previous function for plotting keras model history.

1.2 Datasets

Save datasets from Github repository. You can photos find of leaves with two undetermined foliar diseases. Train and test split is already done.

1.3 Optimization

We will optimize the relevant loss but follow the accuracy as a metric. Optimizer will be `RMSprop` with default parameters unless specified.

1.4 Training

All fitting will be with 500 epochs and a batch size of 64.

1.5 Callbacks

You will use the following functions from `keras.callbacks`:

- `EarlyStopping()`
  - `min_delta=1e-4, patience=4, verbose=1`
- `ReduceLROnPlateau()`
  - `factor=.5,patience=4,verbose=1`
- `ModelCheckpoint()`
  - `save_best_only=True,verbose=1`

Each time you will monitor the loss validation : `monitor='val_loss'`.

2 Data loading

We will see together.

3 Small convolutional network

With **sequential** API, build a convolutional neural network with:

- a Conv2D layer with 16 filters of size 3x3, and ReLU activation
  input shape will be `(img_width, img_height, 3)`
- a batch normalization layer
- a MaxPooling2D layer of pooling layer of 2x2
- a Conv2D layer with 16 filters of size 3x3, and ReLU activation
- a batch normalization layer
- a MaxPooling2D layer of pooling layer of 2x2
- a Conv2D layer with 32 filters of size 3x3, and ReLU activation
- a batch normalization layer
- a dense layer of 32 neurons with linear rectified unit activation
- a batch normalization layer
- a fully connected layers of 32 neurons with ReLU activation
- a dropout layer with probability of 50%
- relevant output layer

Use a learning rate of $5 \cdot 10^{-5}$. Train it using `model.fit_generator()` method. If it is too heavy run it only for 10 epochs. Show learning curves, confusion matrix and accuracy on test set.

4 Troncated Xception model

Build a truncated model from `keras.applications.xception.Xception` at the `'avg_pool'` layer using functional API. We call it `bottleneckModel`.

5 Transfer values saving

For the training and validating sets, store on disk the output of all examples from the truncated model.

For this, you will simply assign to variables the predictions of the previously built model at section 4. As we used generator, use the method `model.fit_generator()`. You will save those predictions thanks to `numpy.save(open(path, 'wb'), varToSave)`.

Create vectors of corresponding labels (call for help).

6 Top model building

With **sequential** API, build a fully connected neural network (called `topModel`) with :

- a batch normalization layer with input shape `bottleneckModel.output_shape[1:]`
- a fully connected layer of 256 neurons with ReLU activation.
- a batch normalization layer
- a dropout layer with a probability of 0.7
- relevant output layer

Use a learning rate of 10^{-5}. Train it with the transfer valued we stored before. You can load a numpy file with `numpy.load(open(path, 'rb'))`. Show learning curves, confusion matrix and accuracy on test set. Once it is trained, you can load the best saved `topModel` by the `ModelCheckpoint` callback thank to `keras.models.load_model(path)`.

7 Full model assembling

We will now assemble our `bottleneckModel` and `topModel` keras models in a `fullModel` object. You can do it thanks to:

```
fullModel = Model(inputs=[bottleneckModel.input],\.
outputs=[topModel(bottleneckModel.output)])
```

Compute predictions of `fullModel` on the test generator. Give the matrix confusion and accuracy.

8 Function activation plotting

Choose on image from the c_34 class and save its path to testImagePath according to `testImagePath = r'path'`.

Call `plotActivation.plotActivationMap(fullModel,testImagePath,\`
`img_width,img_height,savingName=path)`.