filTags documentation

This guideline contains the documentation of the python code for filTags, a method to tag the filters of a convolutional neural network (CNN).

Content

Code Structure

The main code consists of 4 steps:

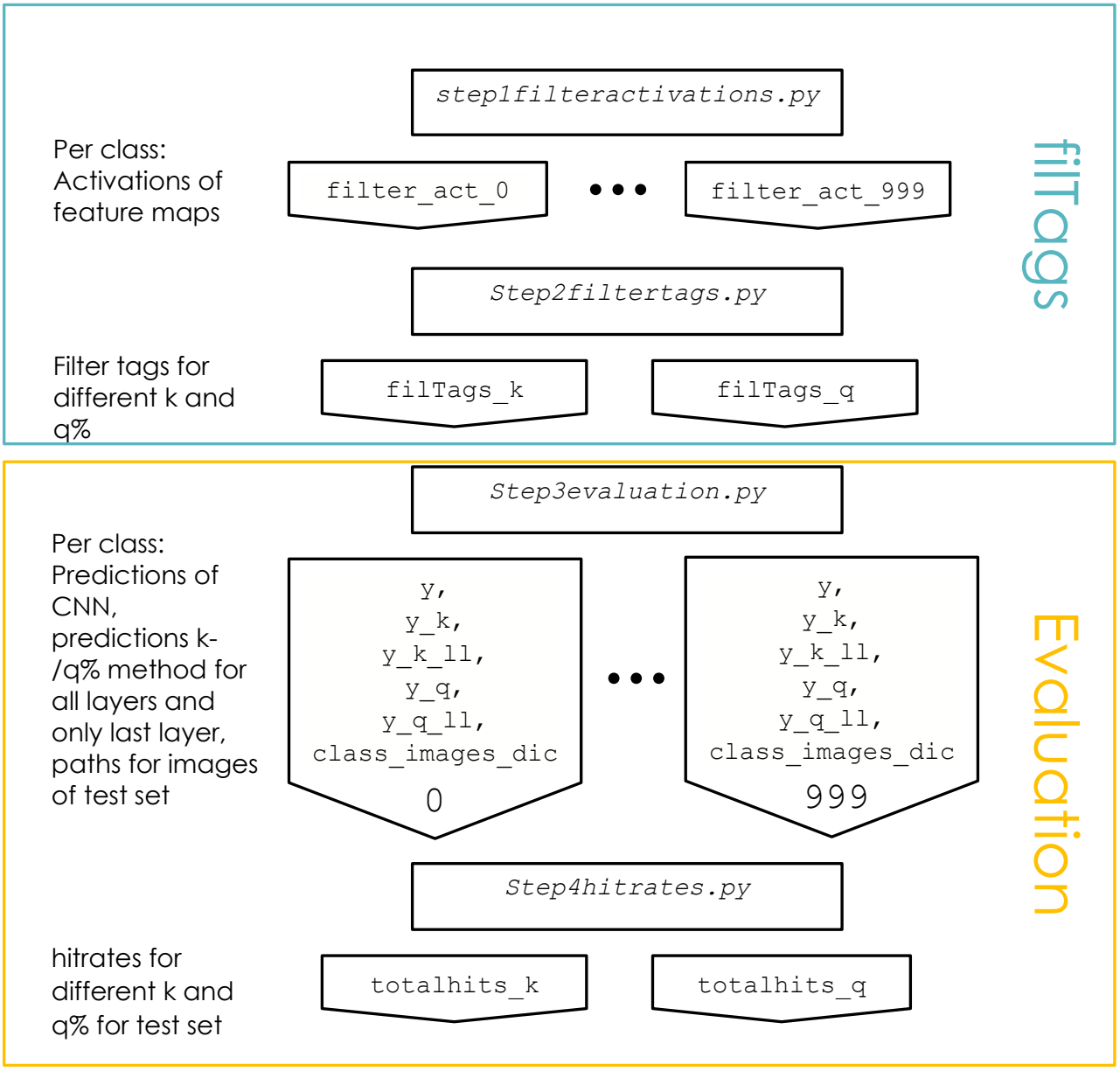
- Step1 Quantifying Filter Activations: *step1filteractivations.py*
- Step2 Tagging of Filters: *step2filtertags.py*
- Step3 Evaluation: *step3evaluation.py*
- Step4 Hitrates and Evaluation Results: *step4hitrates.py*

Additionally, there are 4 more scripts and one notebook for demonstration purposes:

- Single Image Evaluation: si-eval.py
- Evaluation Hitrates Plots: Hitrates-plots.py
- Visualization of filTags: filTagsvis.py
- Decoding of filTags: labdecode.py
- filTags Demonstration Notebook: filTagsDEMO

Codeflow

The main code pipeline is presented in the following figure.



The first two steps consider the approach to create tags for filters of a CNN. The latter two steps deal with its evaluation.

Step1 Quantifying Filter Activations: *step1filteractivations.py*

In the first step the activations for all images in the train set are computed. Further they are aggregated over each image class and feature map. This results in one “averaged activation” value for each feature map and class.

There are 6 inputs to specify:

- **network**: this parameter considers the used CNN. Possible options are networks that are supported by <keras.application>. These are "VGG16", "VGG19", "InceptionV3", "ResNet50", "ResNet101", "ResNet152", "ResNet50V2", "ResNet101V2", "ResNet152V2", "InceptionResNetV2".
- **save_dir**: sets path where to save the produced files
- **time_delta**: sets the deviation in time from UTC for the runID in the filenames.
- **images_dir**: specifies path to the images. This should be the same as in step3 to obtain the same allocation of training set and test set.
- **split**: sets the train-test-split, that determines how much of the data will be used for the creation of the filter tags or the evaluation. Here $(1 - \text{split}) \cdot 100\%$ of the data will be used for train (filTags) and $(\text{split}) \cdot 100\%$ for test (evaluation). This should be the same as in step3 to obtain the same allocation of training set and test set.
- **seed_rn**: specifies a random seed. This should be the same as in step3 to obtain the same allocation of training set and test set.

First the selected pretrained CNN is downloaded from Keras and the image data is accessed with ImageDataGenerator from Tensorflow. For each class, the corresponding images are fed to the CNN and the activations on each neuron in the convolutional layers are queried and aggregated. For this purpose, the activations are queried individually for each image. They are scaled for each layer into the interval 0 to 1 to reduce the influence of individual images. Next, they are aggregated per feature map by the mean value. Finally, the feature maps are aggregated by the mean value over the whole class, resulting in one “averaged activation” value for each feature map and class. These “filter activations” are stored in variable out, where <out[a][b]> returns the value for filter b of layer a.

The outputs are:

- pickle file ending with "-vars.pkl": contains the filter activations for each class with filter_act_i specifying the filter activations of class i.
- pickle file ending with "-vars-names.pkl": contains the variable names of the saved variables, these are filter_act_1, filter_act_2, …, filter_act_n with n being the number of classes.
- text file ending with "-vars-info.txt": contains information about the pickle files
- log file ending with "-info.log": log of the prints and runtime information
- log file ending with "-warnings.log": log of the warnings

Remark: In the original version the output files of step1 were abbreviated with CA, now with QFA

Step2 Tagging of Filters: *step2filtertags.py*

The second step assigns labels to the filters according to their value obtained from *step1filteractivations.py* and the model parameter k or q%.

The inputs are:

- **k_q**: values for model parameters k and q
- **activations_file**: path to "-vars.pkl" from step1 containing the activations
- **save_dir:** sets path where to save the produced files
- **time_delta**: sets the deviation in time from UTC for the runID in the filenames

The model parameters k and q select filters for the tagging according to their activations. For this, the k-highest or q%-highest filter activations per layer are selected and the corresponding filters are assigned the class label. This results in filter tags for the network as a function of k or q. Exemplary, for the choice of k=1 each class label is assigned to one filter of each layer. The filter tags are stored in the variables filTags_k and filTags_q, where <filTags_k[a][b][c]> denotes the tags for k=a for filter number c in convolutional layer number b.

The outputs are:

- pickle file ending with "-vars.pkl": contains the filter tags for the network depending on k (filTags_k) and q% (filTags_q)
- pickle file ending with "-vars-names.pkl": contains the variable names of the saved variables, these are filTags_k and filTags_q
- text file ending with "-vars-info.txt": contains information about the pickle files
- log file ending with "-info.log": log of the prints and runtime information
- log file ending with "-warnings.log": log of the warnings

Remark: In the original version the output files of step2 were abbreviated with LF, now with TF

Step3 Evaluation: *step3evaluation.py*

Based on the obtained filter tags of step2 predictions are made, that can be compared to the network prediction to assess the conformity of the tags.

The inputs are:

- **k_q**: values for model parameters k and q for evaluation
- **filTags_file**: path to "-vars.pkl" from step2 containing the filter tags
- **save_dir:** sets path where to save the produced files
- **images_dir**: specifies path to the images. This should be the same as in step3 to obtain the same allocation of training set and test set.
- **split**: sets the train-test-split, that determines how much of the data will be used for the creation of the filter tags or the evaluation. Here $(1 - \text{split}) \cdot 100\%$ of the data will be used for train (filTags) and $(\text{split}) \cdot 100\%$ for test (evaluation). This should be the same as in step1 to obtain the same allocation of training set and test set.
- **seed_rn**: specifies a random seed. This should be the same as in step1 to obtain the same allocation of training set and test set.
- **time_delta**: sets the deviation in time from UTC for the runID in the filenames

The images of the test set are fed to the CNN one after the other. For each, the network prediction is read first, and then further predictions are made using the activations and the filTags. This works as follows: an image is fed into the network and the most active filters for it

are determined using the activations. Then, e.g. for k=1, the most active filter of each layer is selected and for the corresponding tags (filTags[1]) of these filters the frequencies of the individual class labels are determined. The label with the highest number becomes the top prediction, the label with the second highest number becomes the second best prediction and so on. This is done for the specified k and q% in k_q. As a second way to determine the numbers, only the last layer is considered. This results in the network predictions (y), predictions of the filTags for k and q% based on layers (y_k and y_q) and predictions of the filTags for k and q% based the last layer (y_k_ll and y_q_ll), where <y_k[a]> are the predictions for k=a.

The outputs are:

- pickle file ending with "-varsX.pkl": contains the predictions and imagepaths for class X.
- pickle files ending with "-varsX-names.pkl": contains the variable names of the saved variables for class X, these are y, y_k, y_q, y_k_ll, y_q_ll and class_images_dict
- text file ending with "-vars-info.txt": contains information about the pickle files and variables:
    - y: prediction of the CNN
    - y_k: predictions depending on the choice of k
    - y_q: predictions depending on the choice of q
    - y_k_ll: predictions depending on the choice of k, only consideration of the last layer
    - y_q_ll: predictions depending on the choice of q, only consideration of the last layer
    - class_images_dict: file names of the test
- log file ending with "-info.log": log of the prints and runtime information
- log file ending with "-warnings.log": log of the warnings

Step4 Hitrates and Evaluation Results: *step4hitrates.py*

This code compares the predictions of the network to the predictions of the evaluation and computes hitrates. The parameter indicates how often the predictions match.

The inputs are:

- **eval_file**: path to "-vars.pkl" from step3 containing the predictions from the network and the evaluation
- **save_dir:** sets path where to save the produced files
- **m**: values for evaluation parameter m
- **k_q**: values for model parameters k and q for computing hitrates
- **classes_img**: image classes to evaluate
- **time_delta**: sets the deviation in time from UTC for the runID in the filenames

First the hitrates are computed for each class. They determine in how many cases the network prediction matches the top-m predictions of the evaluation. The results are printed in the text file ending with ”-results.txt”. Furthermore, the hit rates of the classes are summarized and one is calculated over all these classes. This results in hitrates over all classes depending on k (totalhit_k) and hitrates over all classes depending on q (totalhit_q)

The outputs are:

- pickle file ending with "-vars.pkl": contains hitrates depending on k and q
- pickle file ending with "-vars-names.pkl": contains the variable names of the saved variables, these are totalhit_k and totalhit_q

- text file ending with "-vars-info.txt": contains information about the pickle file
- text file ending with "-results.txt": contains value tables of the hitrates
- log file ending with "-info.log": log of the prints and runtime information
- log file ending with "-warnings.log": log of the warnings

Remark: In the original version the output files of step4 were abbreviated with PH, now with HR

Additional Code

Single Image Evaluation: *si-eval.py*

This script computes tag counts for the test images of one class. They are stored in a txt-file ending with "-results.txt". It can be executed after *step2filtertags.py.*

Decoding of filTags: *labdecode.py*

This script decodes the ImageNet wnids into human readable form e. g. “n02088364” to “beagle”. It can be executed after *step2filtertags.py.*

Visualization of filTags: *filTagsvis.py*

This script creates interactive plots of the filTags based on plotly. It can be executed after *step2filtertags.py.* It is recommended to decode the filTags first if necessary, with e. g. labeldecode.py for ImageNet. For networks other than VGG the number of presented layers should be restricted to reduce computational overhead.

Evaluation hitrates plots: *hitrates-plots.py*

This script produces plots for the hitrates as a function of m and k or q%. It can be executed after *step4hitrates.py.*

filTags Demonstration Notebook: *filTagsDEMO.ipynb*

This notebook is designed for execution on Google Colaboratory (https://colab.research.google.com/). It presents an introduction of filTags based on a small subset of CIFAR100. It treats *step1filteractivations.py, step2filtertags.py* and *filTagsvis.py.*