Query By Video for Surgical Activities Documentation

Gianluca Silva Croso, Felix Yu

May 2018

This document is meant to provide directions on how to use the delivered code for this project.

1 Dependencies

Code requires python 3.6, along with the following packages:

- numpy
- matplotlib
- cv2
- skvideo
- sklearn
- pytorch version 0.2 or later
- torchvision version 0.1.9 or later

All packages installed with conda or pip install. Version numbers specified only for packages which basic conda/pip installation might give incompatible version. In my case, correct version of pytorch obtained using
conda install -c soumith pytorch

2 Data

This section explains how the code expects the data to be organized.

2.1 Phase labels

Each surgical phase or activity in cataract surgery was assigned a label from 1 to 10. The following table displays which phases are matched to which labels

ID	Activity Name
1	Side Incision
2	Main Incision
3	Capsulorhexis
4	Hydrodissection
5	Phacoemulsification
6	Cortical Removal
7	Lens Insertion
8	OVD Removal
9	Corneal Hydration
10	Suture Incision

2.2 Directory structure and naming conventions

There are multiple places in the scripts where data input/output directories are hard-coded. These need to be updated to the desired directories for whoever is running the code. To easily find where these paths are written, search for comments including the words "MODIFY IF NECESSARY". To make sure the only update to the code is the name of the directories, please use the following structure:

- For full surgical videos, data directories just contain all the videos within them. The last three characters of the video name (excluding the extension) should be a unique numerical identifier.
- Annotations directories contains all annotations text files within them. Naming follows vid_ZZZ.txt where ZZZ is the unique identifier of the corresponding video.
- for phase clips, data directories contain sub-directories with names ranging from "1/" to "10/" to represent each phase. Videos are just located inside those sub-directories. The code will automatically append the name of the sub-directories when applicable, so that the user only needs to provide the name of the main data directory. Phase clip are written with the following convention "pX_nY_vid_Z" where X is the phase number, Y is incremented if a particular phase appears more than once in a single video, and Z is the original surgical video identifier.
- for spatial feature or tool information npy matrices, the data directory should contain three sub-directories, "train/", "val/" and "test/". Within these, sub-directories from "1/" to "10/" to represent each surgical activities, containing the relevant .npy files. The naming convention for the tool .npy files should imitate the naming convention for the corresponding

phase clip spatial features .npy file, except with "_tools" at the end before the .npy extension.

- for final feature databases, each part of the data (train, val, test) gets one database "train.npy", "val.npy" and "test.npy" along with an accompanying .txt file that maps the database index to the specific phase clip.

2.3 Analysis Outputs

Final outputs include a confusion matrix (image) of the test set and a txt file that details which clip from the training set was the closest neighbor to each clip of the test set.

3 Code

The code for the final version of the pipeline is in the *Deliverables* directory. All the code in that directory is thoroughly documented, with every function including a description of what it does as well as the relevant inputs and outputs. We will also describe here what each file is and how to run it. The *scripts* directory contains several scripts for previous versions of the pipeline, along with architectures and data processing attempts that did not work. The latter is only included in case someone takes up this work in the future and wants to refer to what was already tried or use data processing tools that might be available there, but that code is not as thoroughly documented and we will not detail all of its files here. The relevant *Deliverables* directory is subdivided in the following three sub-directories:

- *squeeze_net* contains the scripts necessary to obtain spatial features using the squeezenet architecture.
- *RNN* contains the scripts necessary to train the Recurrent neural network using squeezenet outputs along with tool data, as well as to generate the databases containing the final spatio-temporal encodings of the multiple phase clips.
- *utils* contains scripts to split surgical videos into phase clips, convert data to optical flow, generate simulated tool labels, run the analysis on the test set and vizualize features.

Below we document all the files in these directories. Additional documentation for each particular function within the files is available in the code itself. Throughout the code, input and output directories are hard-coded. These can be easily found and modified, as they always follow comments defining what they represent and ending with the words "MODIFY IF NECESSARY"

3.1 squeeze_net

This directory represends the spatial feature extractor for our pipeline. The network itself was trained by Tae-Soo Kim, and the code in **cataract_data_utils.py** and **train_image_phase_balance.py** was provided by him. The obtained after training and used for classification were also provided in the directory in the form of the file **005_0.699.pkl**. We wrote a driver that will obtain spatial feature representations for clips using the squeeze net architecture and given weights

3.1.1 squeezenet_driver.py

This file is a driver to load squeezenet with pretrained weights and obtain feature matrix for each phase clip in data_dir, writing it out as an npy file to output_dir. Weights file should be given as first system argument when running the driver. For each clip, one frame will be considered per second of the clip. Features are outputed as npy files, one for each clip, using the directory structure specified in the Data section. It requires directory paths to:

- the directory where the phase clips are located
- the directory where the npy feature matrices should be outputed.

These can be updated within the code.
To Run: `python squeezenet_driver.py <weights_file>`

3.2 RNN

This directory includes the necessary code to extract temporal features for our pipeline, particularly through the implementation of an RNN. The input to the RNN are the feature matrices outputed by squeezenet along with corresponding tool annotation matrices for the relevant frames for each clip. Each clip outputs a temporal vector, ultimately saved in 3 npy databases, one for each of the train, val and test sets. The following files were written by us and are included:

3.2.1 Models.py

This script contains the class for the PyTorch network model we use as the temporal encoder. This is represented in the class GAP_RNN, which details the architecture of our recurrent neural network.

3.2.2 train_GAP_RNN.py

This script should be run to train the GAP_RNN using triplet-loss. It outputs the weights of the best trained model according to the validation procedure. It requires a directory path to:

- the directory where the npy feature matrices for each phase outputed by squeezenet_driver.py are located

- the directory where the npy tool annotation matrices for each phase, such as those outputed by tool_labels.py (simulated, see utils section), are located
- the directory where the weights should be outputed.

These should be changed directly within the script.

To Run: `python train_GAP_RNN.py`

3.2.3 temporal_encoder.py

Create database of spatio-temporal features based on either mean averaging or GAP_RNN model using spatial features outputed by squeezenet and tool features. Saves database as 3 files: train.npy, val.npy and test.npy Each npy file contains the feature vectors for all examples phase clips in the corresponding set. Should be run with either 0 or 1 as an argument to identify whether to use the GAP_RNN or simply take the mean of the multiple spatial feature vectors for a clip to obtain the final spatio-temporal feature vector for the clip. Requires the following directories:

- the directory where the npy feature matrices for each phase outputed by squeezenet_driver.py are located
- the directory where the npy tool annotation matrices for each phase, such as those outputed by tool_labels.py (simulated, see utils section), are located
- the path to the file containing the weights should the RNN be used.
- the directory where the output .npy databases should be written

These should be changed directly within the script.
To Run: `python temporal_encoder.py <which_model>`
which_model should be 0 to use mean, 1 to use RNN.

3.2.4 helpers.py

Contains multiple helper functions as well as the dataset classes used by the training procedures, which are:

- SpatialToolFeatDataset class:
  This class extends the PyTorch Dataset class and serves as a dataset for the spatial and tool features of a surgical activity clip. It loads spatial and tool features npy files with mathing names from the corresponding directories and outputs them concatenated in the correct format to be used with pytorch. Used for training the RNN.

- SpatialFeatDataset class:
  This class extends the PyTorch Dataset class and serves as a dataset for the spatial features only of a surgical activity clip. It loads spatial features npy files from the corresponding directory and outputs them in the correct format to be used with pytorch. Can be used for training the RNN in the absence of tool features.

- generate_val_exemplars:
  Generates a validation dataset for triplet-loss based training procedures. Inputs are a list num_clip_per_act containing the number of available segments for each activity as well as an integer num_exem_per_act which is the number of triplets that should be returned with an anchor in each activity. The latter might be changed by the function in order to ensure consistency. Returns a list of lists, where each item is a list of indexes for exemplars within each activity.

- select_triplets:
  Selects triplets for backpropagation in triplet-loss training. Specifically, tests all possible triplets given the input and chooses only triplets that are considered semi-hard, that is, the positive example is closer to the anchor than the negative, but the difference between the distances is less than alpha (a predefined constant margin). Inputs are a numpy matrix representing the features of multiple example segments (each row is a specific example), the aforementioned margin alpha (a float), and an integer num_segs_per_act detailing the number of segments that represents each activity in the matrix. Outputs three 1D numpy matrices, triplets_a, triplets_p and triplets_n, containing the indexes of the anchors, positives and negatives reprectively in the original feature matrix. Elements with the same index in these matrices form a single triplet.

3.3 utils

Utils contains the following scripts, which are used for multiple applications related to data processing or analysis:

3.3.1 separate_phases.py

This script separates whole surgery videos into activity clips based on ground-truth annotations of the form
start_frame end_frame phase
The script needs directory paths to:

- directory where the video clips are located
- directory where the video annotations are located
- directory where the activity clips should be outputted

These should be changed directly within the script.
It is assumed that the video clips are all located directly within the specified directory. The same goes for the video annotations. When the activity clips are outputted, sub-directories are created, one corresponding to each activity, and clips are put in their respective sub-directories.

To Run: `python separate_phases.py`

3.3.2 new_separate_phases.py

This script separates whole surgery videos into activity clips based on ground-truth annotations of the form
time legend phase
where legend is either start or end. These annotations are a result of newer annotation software when compared to the previous. The script needs directory paths to:

- directory where the video clips are located
- directory where the video annotations are located
- directory where the activity clips should be outputted

These should be changed directly within the script.
It is assumed that the video clips are all located directly within the specified directory. The same goes for the video annotations. When the activity clips are outputted, sub-directories are created, one corresponding to each activity, and clips are put in their respective sub-directories.

To Run: `python new_separate_phases.py`

3.3.3 convert_optical_flow.py

Converts phase clips into optical flow videos using cv2 dense optical flow implementation. Naming is kept exactly the same except in a different directory for output. The script needs directory paths to:

- directory where the regular activity clips are located
- directory where the optical flow activity clips should be outputted

These should be changed directly within the script.
To Run: `python convert_optical_flow.py`

3.3.4 tool_labels.py

Based on expected tool presence information for each phase as detailed by Dr. Vedula, generates simulated tool data. For each available spatial feature matrix, generates tool data for equivalent frames. There are 16 tools and the tool

feature vectors contain 16 elements, with a 1 if the equivalent tool is present and 0 otherwise. The tools are identified as follows:

Tool ID	Tool Name
1	Paracentesis blade
2	Keratome
3	Anterior chamber cannula
4	0.12 forceps
5	Calipers
6	Cystotome
7	Utrada forceps
8	Hydrodissection cannula
9	Phaco handpiece
10	I/A handpiece
11	IOL injector
12	Sinsky hook
13	Suture
14	Weckcell sponge
15	Syringe
16	Needle driver

For tool identifier i, the index in the vector is $i - 1$. If the spatial feature matrix has size NxF where N is the number of frames considered and F the feature vector size, the tool feature matrix will have size $Nx16$. Naming of output npy files will be identical to input except for "_tools" at the end before the ".npy" extension. Assumes pre-existing directory structures as detailed in Data for phase clips. The script needs directory paths to:

- directory where the spatial features are located
- directory where the tool features should be outputted

These should be changed directly within the script.
To Run: `python tool_labels.py`

3.3.5 visualize_features.py

Given a directory containing a "train.npy" numpy matrix containing feature encodings outputted from a model, where the rows represent inputs and the columns represent the features themselves, output a 2-D visual representation of the datapoints.

To Run: `python visualize_features.py <data_dir>`

3.3.6 analysis.py

Using train.npy as database, analyses accuracy of model in test database, producing accuracy statistics, printing precision and recall for each phase, and

saving a confusion matrix. Prediction is based on majority of num_neighbors nearest neighbors. If there is a tie, closest neighbor gets preference. Also outputs a text file that matches each phase clip from the test data with one most similar clip from the training data. Takes as arguments 0 or 1 to identify which kind of temporal model was used and the number of nearest neighbors to be considered for the analysis. The script needs directory paths to:

- directory where the npy databases are located
- directory where the confusion matrix and matchings file should be outputted

These should be changed directly within the script.
`python temporal_encoder.py <which_model> <num_neighbors>`
which_model should be 0 if mean was used, 1 if GAP_RNN was used without tools, 2 if GAP_RNN was used with tools. The value for num_neighbors is usually 1 (most similar video) but more could be used.

4 Sample Pipeline Run

Now we will describe a sample run of the complete pipeline, from separating the phase clips to obtaining the final analysis results.

- Depending on the kind of annotation available, decide whether separate_phases.py or new_separate_phases.py should be used. Update surgical video input and phase clip output directories in the script. Run the script.
- Update the input directory to squeezenet_driver.py to match the output of the previous step. Update the desired output directory for spatial features. Run the script giving as argument the file containing the squeezenet weights.
- If real tool annotations are available, format them in the same way described in the tool_labels.py section. Otherwise, update the input directory in tool_labels.py to the output of the previous step, and update the desired output directory for simulated tool information in the script. Run the script.
- Train the GAP_RNN model. Update the input directories for spatial and tool features as the output directories of the two previous steps. Update the output directory for the weights. Run the script.
- In temporal_encoder.py, update the input directory for spatial and tool features to be the output directories of the second and third step respectively. Update the path to the preferred weights file outputted in the previous step. Run the script using argument 1 as which_model.
- Optionally run visualize_features.py with the output of the previous step as an argument to visualize the feature distribution of the training dataset.

- Update the input directory in analysis.py to the output directory of temporal_encoder.py. Update the output directory for the confusion matrix and text file containing closest matchings. Run analysis.py with arguments 2 1 for which_model and num_neighbors respectively.