# CIS II Coding Documentation

Gianluca Croso Felix Yu

March 2018

The code is separated into two different directories. There is *misc*, which contains scripts that do various data processing and data visualization. The other directory is *Models*, which contains scripts related to our Neural Networks, the training procedures, and the testing methods.

## 1 misc

Misc contains the following scripts, which each do these things:

### 1.1 create_lmdb_files.py

This script takes in video files, and outputs lmdb files that contain data that is compatible as input into C3D.
This script needs directory paths to the following three places:

- directory where the activity clips are located
- directory where the train/val split file is located
- directory where the lmdb files should be outputted to

These should be changed directly within the script.
It is assumed that the activity clips directory contains more subdirectories, each one corresponding to an activity, and these subdirectories are what contain the videos (in .avi form).
The lmdb files are written out into the output directory. In this directory, two subdirectories are formed, *train* and *val*, and each of these then contains more subdirectories, one for each activity. Within this subdirectory are the lmdb files.

To Run: `create_lmdb_files.py`

### 1.2 create_res _net_img_lmdb.py

Same as above, but the data is compatible with ResNet instead.

To Run: `python res_net_img_lmdb.py`

### 1.3 separate_phases.py

This script separates whole surgery videos into activity clips based on ground-truth annotations. 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 to

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, subdirectories are created, one corresponding to each activity, and clips are put in their respective subdirectories.

To Run: `separate_phases.py`

### 1.4 separate_phases.py

This is used to make sure the videos were written properly. If a video was not written properly or could not be properly loaded, delete the video.

- the directory where the activity clips are located.

These should be changed directly within the script. The text files will be outputted to the same directory.

To Run: `separate_phases.py`

### 1.5 split_train_test.py

This script outputs two text files, one containing file names of training clips, the other containing file names of validation clips. In order to do this, the script needs directory path to:

- the directory where the activity clips are located.

These should be changed directly within the script. The text files will be outputted to the same directory.

To Run: `split_train_test.py <percentage of training (as a decimal)>`

### 1.6 play_video.py

Plays a video of form .avi.

To Run: `play_video.py <path to video file>`

### 1.7 visualize_features.py

Given a 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: `visualize_features.py <path to numpy file>`

## 2 Models

### 2.1 Models.py

This script contains the classes for all the PyTorch network models we use in our other scrpits. This includes the following classes:

- C3D - the C3D model used to train with triplet loss, with normalized output features and the softmax layer removed from the end
- C3D_soft - the C3D model used to train with Cross Entropy Loss, with the softmax layer included

### 2.2 train_C3D.py

This script should be run to train C3D 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 lmdb datasets are located, as created by create_lmdb_files.py
- the directory where the weights should be outputed.

These should be changed directly within the script.

To Run: `train_C3D.py`

### 2.3 train_ResNet.py

This script should be run to train ResNet 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 lmdb datasets are located, as created by create_res _net_img_lmdb.py
- the directory where the weights should be outputed.

These should be changed directly within the script.

To Run: `train_ResNet.py`

## 2.4 train_C3D_crossEntropy.py

This script should be run to train C3D using tCross Entropy 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 lmdb datasets are located, as created by create_lmdb_files.py
- the directory where the weights should be outputed.

These should be changed directly within the script.

To Run: `train_C3D_crossEntropy.py`

## 2.5 train_ResNet_crossEntropy.py

This script should be run to train ResNet 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 lmdb datasets are located, as created by create_res _net_img_lmdb.py
- the directory where the weights should be outputed.

These should be changed directly within the script.

To Run: `train_ResNet_crossEntropy.py`

## 2.6 get_C3D_feats.py

Outputs a np matrix file contianing the features for 1000 segments, 100 from each activity, based on specific weights that are loaded for the C3D model. It requires a path to:

- the directory where the lmdb datasets are located, as created by create_lmdb_files.py
- the file where the weights are located.
- the output .npy file

These should be changed directly within the script.

To Run: `get_C3D_feats.py`

## 2.7 helpers.py

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

### 2.7.1 ActImageDataset class

This class extends the PyTorch Dataset class and serves as a dataset for a surgical activity, associated to a single lmdb file, specifically for use with models where the input is a single image (in our case, ResNet). It loads images in from lmdb and outputs them in the correct format to be used with pytorch, and can apply the standard transforms used with ResNet.

### 2.7.2 ActSegmentDataset class

This class extends the PyTorch Dataset class and serves as a dataset for a surgical activity, associated to a single lmdb file, specifically for use with models where the input is a 16-frame segment (in our case, C3D). It loads segments in from lmdb and outputs them in the correct format to be used with pytorch.

### 2.7.3 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.

### 2.7.4 get_val_indexes

Generates a validation dataset for cross entropy loss based training procedures. Inputs are a list num_clip_per_act containing the number of available images for each activity as well as an integer batch_size which is the number of images that should be returned in each activity. The latter might be changed by the function in order to ensure consistency. Returns a numpy matrix where each row represents an activity and each element within a row is the index for an example within the activity, as well as a second numpy matrix of the same dimension with the labels associated to those examples (the labels within a same row are always the same)

### 2.7.5 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.