**---- DATA ----**

Full dataset available at: https://jacquard.liris.cnrs.fr/files/database/download.php

Log-in name: burak-mete

(I can provide the dataset if online download fails for some reason)

**Data Format:**

Since pyTorch dataloader type speeds up the training phase drastically, I have changed the data loading process to that mechanism

*pandas\_csv\_dataload.py* does the magic. Instead of gathering the whole data (image values) at once, I have gathered their paths, and all of their multiple labels at one .csv file.

A Dataloader class is overloaded to fetch the training data.

First column: Data path

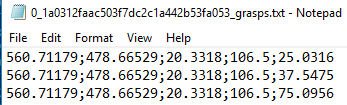
All of the others: Their corresponding labels (x, y, h, w, theta respectively)

x, y, h, w values are scaled down to 224x224, theta stays the same.

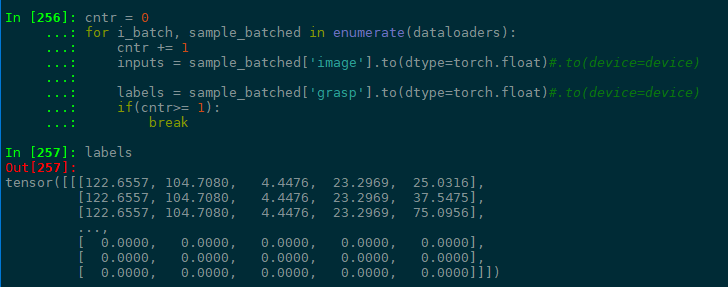
Number of labels are differing for each image, thus some of the columns are 0 for most of the images.

Number of columns are decided with a parameter coming from *the findMaxLabels() method at pandas\_csv\_dataload.py*

While selecting the appropriate label, the ones with 0 will be discarded.



**Fig 1- 3 labels for first image**



**Fig 2- Dataloader outcome for fetched labels, properly scaled down and 0’d out for non-existent labels**

Preprocessing steps are also overloaded:

[Rescale, ToTensor, Normalize] = {transforms the image to desired shape (224,224),

numpy to tensor,

normalize with a given mean and deviation parameters}

Now dataloader object has batched data with **['image']** having the data and **['grasp']** having the labels.

**num\_workers** parameter of dataloader constructor allows parallel programming, however windows sometimes fails to create pipeline, however Unix OS seems to work quite well.

In main.py, in order to start the training, there should be a valid .csv file which is an input to the dataloader object.