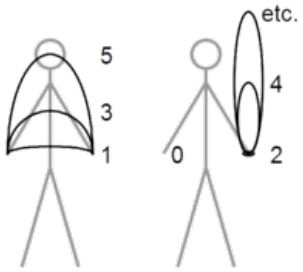
Project Recognizing Siteswap juggling tricks

link to colab code: https://colab.research.google.com/drive/1rGkIaU7v11UP7Mq_rlde4B9QJm0Gw6Ou

Idea

The siteswaps trick in juggling means repeating the sequence of throws with both hands by turns.
Each throw is labelled with a number between 0 – 9. A sequence of throws of a particular trick makes a nice show (e.g. *51 trick*).

Siteswaps notation



I would like to analyze single throws to understand the initial sequence. It requires the following steps:

- labelling data,
- recognizing single throws,
- merging all throws from one video to take the most probable sequence (consulting that not all sequences are possible).

Now I managed to do two first steps, but as the results seem to be better than established, probably the third step is still possible.

I use the database from https://sites.google.com/view/jugglingdataset/juggling-tricks?authuser=0.
For each second of the video, there are 30 frames with coordinates of all balls from the trick. Part 1
I managed to analyze and label 19 videos and I received 1107 samples of single throws.

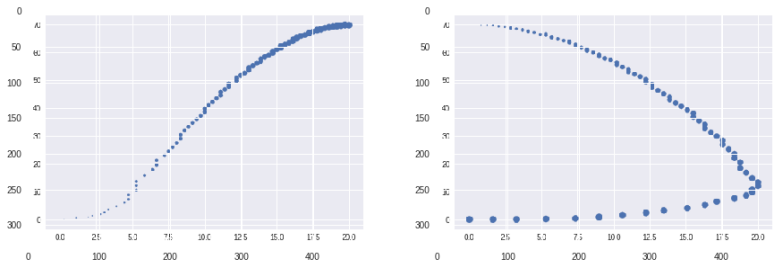
The approach

As discussed during the laboratory I divided videos into single ball throws (represented as a ball trace in time on the screen).
To separate the next throws of one ball, I choose a moment when a ball starts going up.
It is not equivalent to the moment when it is thrown up (and this one was hard to find), because for some time the juggler holds the ball speeding it up. Part 2 in code
I track the direction in which the ball goes and when it starts going up, I cut the throw.

Trick 3 in 2 throws

This algorithm gave me almost 100 of half-throws – two trace fragments which together mean one real throw, but cut in the half (with a ball at the top):

Half-throw



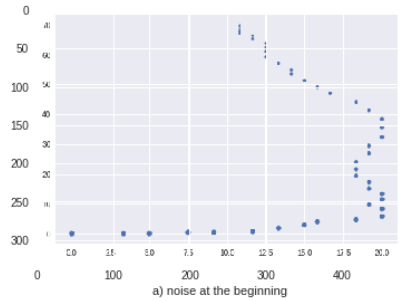
To deal with it I add a constraint: I could cut the video only if the end of the throw is on the similar height (on the screen) as the beginning of the throw.

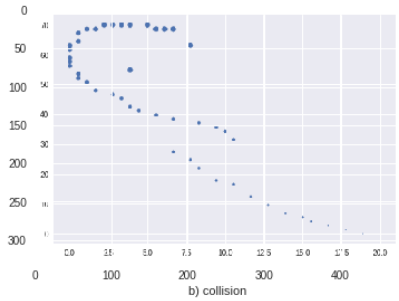
Labelling

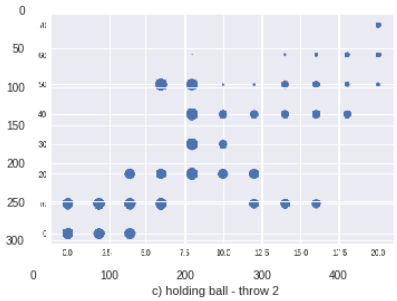
For each single throw trace, I had to translate it to siteswap notation's number. Part 3 in code
For each video, I displayed all single throws. Next, I performed the following steps:
- to distinguish noise at the beginning/end of the video from the real trick (in some videos jugglers start from having balls in hand) – see figure (a);
- to recognize what is the throw sequence for the ball (e.g. *trick 3* consists only of *throws 3*, but *trick 64* means juggling 3 balls in one hand with *throws 6* in one hand and juggling 2 balls in second hand with *throws 4);*
- to decide which traces I can label as single throws (check if the algorithm cuts a video correctly);
- to recognize unusual situations (ball collisions – see figure (b), smooth switching between tricks).

I couldn't label *throw 2* which in notation means „holding a ball without tossing". As juggler never holds the ball without moving it and the algorithm tries to divide his subtle movements I got unstructured traces – see figure (c).

Unlabelled traces


a) noise at the beginning

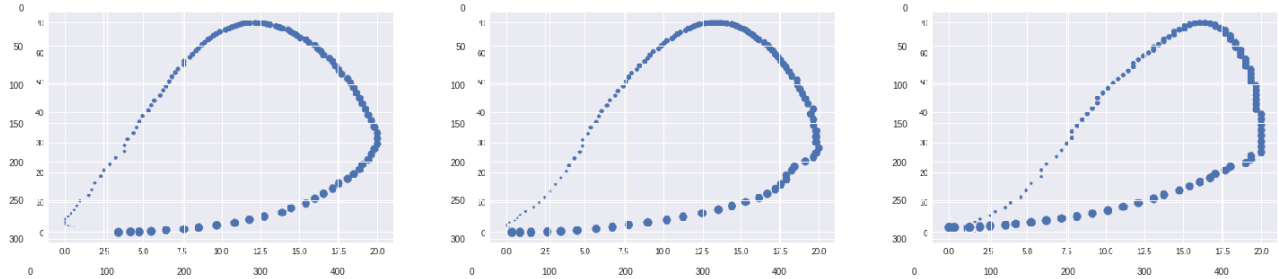

b) collision


c) holding ball - throw 2

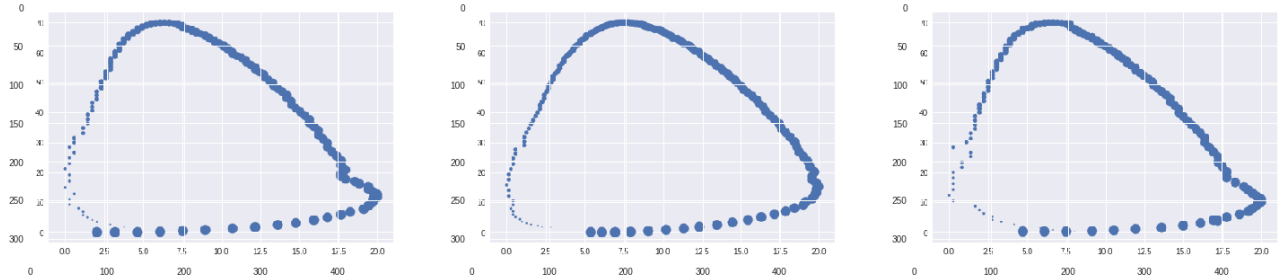
Differences between classified groups

It is easy to decide if the throw is in a group *1-3-5-7* or in *4-6* as the first group puts the ball to another hand. Then the next throw starts from the other side of the picture.
The hardest part is to distinguish throws from a group *3-5-7* or *4-6* as on normalized pictures I don't see the heights of the throws. To deal with it I used the knowledge about the video. E.g. if I knew that the ball trace consists of *throws 4* and *throws 6* I was comparing the trace with the video – checking the time when the ball was caught by hand, paying attention with which hand it was caught. Also, as a higher throw goes up faster and is also faster carried by hand, I was comparing different traces with the density of points in trace and the angles when a ball changes direction. And I was trying to find the most probable sequences for all balls for the whole trick.

Exemplary pictures of throw 4 All exemplary pictures: Part 4 in code



Exemplary pictures of throw 6



Neural network
To convert my pictures into a network input I normalize them to the 20x70 image. Each pixel in the image means the density of this point in the ball trace. During this step, I also add labels to every single picture (which have a little bit different format I recorded during labelling). Part 5 in code

As the output, I expect the throw number from siteswap notation.
For a given set of tricks, I divide data in ratio 64:16:20 for training:validiation:test databases.

Different architectures I compared by taking the average of 7 test errors for my database.

I started from building a simple regular densely-connected sequential neural network from keras library. I used softmax activation for the output to take the most probable number. Part 6 in code

Adding hidden layers or enlarging layer sizes was decreasing test error and maximal error values. The best result I achieved was 8 %, but as checked on a smaller group of throws (e.g. tricks [3,4] only), the network was overlearning, so I decided to use a one-hidden-layer network with around 11% which works much better on smaller sets, too. Part 7 in code (Testing the network on a group of throws means that I take to the database only throws from this group.)

Changing inner parameters like activation functions didn't improve the network.

The best regular neural network I built was:

```
model = Sequential()
model.add(Dense(64, input_dim=matrix_size[0]*matrix_size[1], activation='relu'))
model.add(Dense(32, activation='relu"))
model.add(Dense(num_classes, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(X_train, Y_train, validation_split=0.2, epochs=10, batch_size=50, verbose=verbose)
```

Secondly, I tried to use a convolutional neural network for this problem.
As these networks are bigger and work slower, I decided to take an average of 3 samples.

I started from converting the best regular network into convolutional one by changing Dense layers into Conv2D layers from keras. Adding one Dropout layer (p = 0.20) to it turned out to be the best network I built – with circa 5% test error for the group with all throw types.

```
model = Sequential()

model.add(Conv2D(64, kernel_size=3, activation='relu',input_shape=(matrix_size[0], matrix_size[1], 1)))
model.add(Dropout(0.20))
model.add(Conv2D(32, kernel_size=3, activation='relu'))
model.add(Flatten())
model.add(Dense(8, activation='softmax'))

model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['accuracy'])

model.fit(X_train, Y_train, validation_split=0.2, epochs=10, batch_size=50, verbose=verbose)
```

Results

Applying the best networks I built for different groups of throws I got:

Results for some throw groups for regular neural network

Group	1 3	1 4	1 6	4 6	3 4	3 5	5 7	4 5 6 7	1 2 3 4 5 6 7
Mean Test Error	2.87 %	0.64 %	3.03 %	8.48 %	3.66 %	4.29 %	7.27 %	10.30 %	11.26 %

Results for some throw groups for convolutional neural network

Group	1 3	1 4	1 6	4 6	3 4	3 5	5 7	4 5 6 7	1 2 3 4 5 6 7
Mean Test Error	0.00%	0.00 %	1.52 %	6.67 %	1.83 %	5.81 %	4.55 %	7.68 %	8.86 %

Looking at smaller groups of throw numbers:

- in both – small and big groups – convolutional neural network is mostly better than regular neural network,
- the network can recognize *throw 1* with less than 2% error when compared to other groups,
- throw 6 is hard to distinguish as it has small dataset,
- for very similar cases *3-5*, *4-6, 5-7 or 3-4* it works on circa 4% error.

Because datasets are small, results vary more because random division into training:validiation:test puts very few images of one throw in any group.

Next steps

To improve results, I could:

- get more data – label more videos,
- repeat labelling – it is possible that I made some mistakes, I was unsure sometimes,
- apply data augmentation and use cross-validation.

After that maybe the next step of understanding the whole sequence would be also possible.