**Table Tennis   
Shot Classification**

Dean Geckt, Adi Brucker

305766032,204748529

Contents

[Introduction: 2](#_Toc112602397)

[About the code: 2](#_Toc112602398)

[How to run with google Colab: 2](#_Toc112602399)

[Goals: 5](#_Toc112602400)

[Data: 5](#_Toc112602401)

[Architecture – high level: 6](#_Toc112602402)

[Architecture – Shot extractor: 7](#_Toc112602403)

[Media pipe pose 7](#_Toc112602404)

[Extraction: 7](#_Toc112602405)

[DL model: 9](#_Toc112602406)

[Training: 9](#_Toc112602407)

[Data augmentation: 9](#_Toc112602408)

[Feature selection: 10](#_Toc112602409)

[Results & Examinations: 10](#_Toc112602410)

# Introduction:

## About the code:

Train.py- this file reads the data, train the model and plot the result

Train.ipynb – notebook that should be opened by Google Colab, contains the code separated by logical sections.

Train\_colab.ipynb – same notebook with addition code to run on colab.

Internal folders:

utils – contain utils and common functions used by the notebook

visualization – utils that help us create gifs of the player pose movements and debug

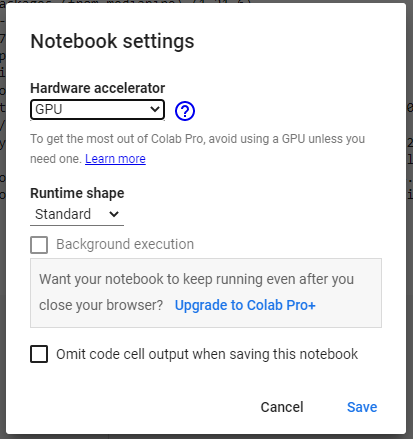
split\_data – raw data split to train, test and validation sets

shot\_extractor – image processing logic that apply on the raw data to extract the shots from the video.

raw\_data – contains the CSV - the output from the extractor

## How to run with google Colab:

1. Adding Project folder to google drive
2. Open train\_colab.ipynb via google colab
3. Change runtime to GPU



1. Configure google Colab environment by running:

Text

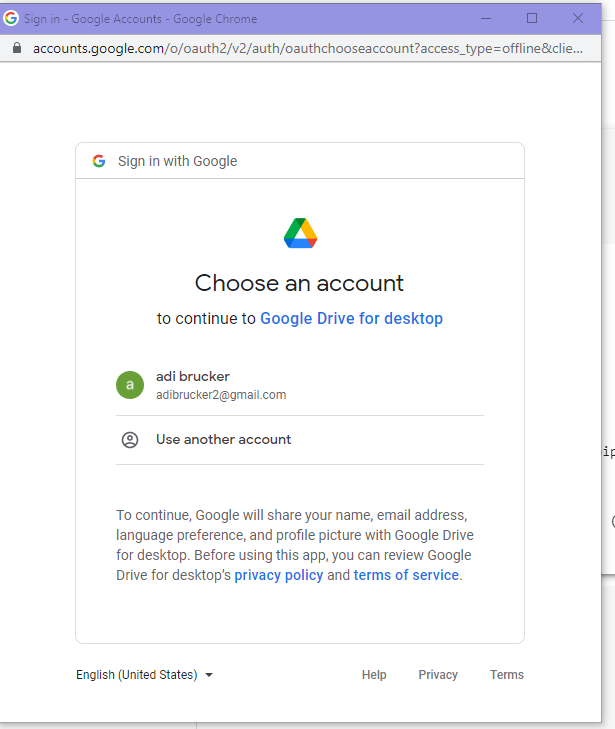
Description automatically generated

1. Approve connect to your Google drive

Graphical user interface, text, application

Description automatically generated

1. Follow the instruction



Graphical user interface, application

Description automatically generated

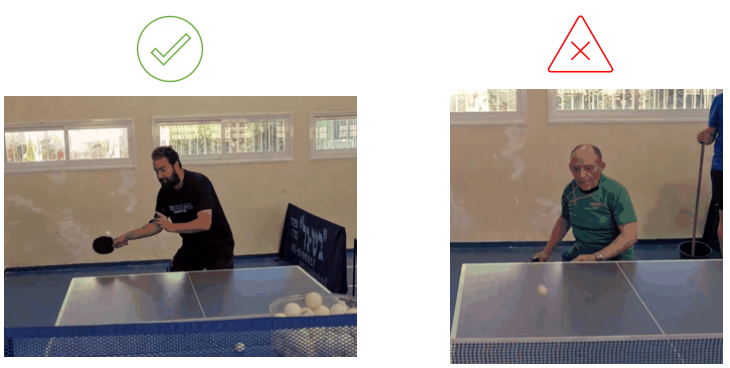
# Goals:

Main goal:

The main goal of this project is to classify and distinguish between **technically correct** table tennis shot to one that is not.

Application:

A real time mobile application used to improve player skill during practice.



# Data:

Raw data consists of table tennis **practice** **videos** (30 fps) of several players in Dean’s local club. (ages of 15-60). A practice video contains a **single** player training on a specific shot repetitively. Some parts of a video can contain “garbage” such as player picking up the ball, leaving the table to get it etc.

Shots were extracted from each video and were **labeled** manually by us.

We collected 2 types of shots: Forehand contra (fco) - 2301 shots, 162 good shots

Forehand top spin (fts) - 1371 shots, 268 good ones. Each shot is on average 15 frames.

The main difference between the two in term of technique is the fts shot is much faster then fco – there are less visible frames of the right hand moving. Also the fts shot usually start from the adjacent to the knee where as fco starts closer to the table and is more forward.

In our presentation we attached full videos that show this difference.

Each shot is a different data set thus we trained 2 models.

# Architecture – high level:

Video

Shot extractor

DL model

predictions

Graphical user interface

Description automatically generated

# Architecture – Shot extractor:

## Media pipe pose

Media-Pipe Pose is a ML solution for high-fidelity body pose tracking, inferring 33 3D landmarks and background segmentation mask on the whole body from RGB video frames

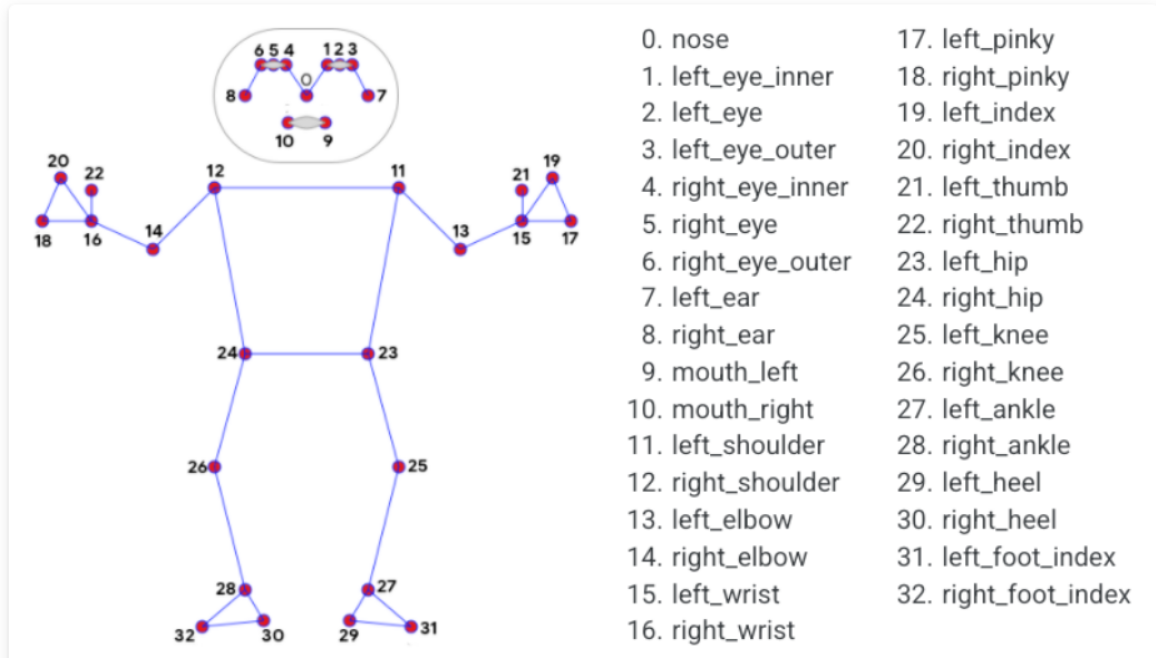
Link to Documentation:<https://google.github.io/mediapipe/solutions/pose.html>

The shot extractor is a **preprocess** component that extract shots from a video.

The Extractor uses Google's Media pipe pose – each frame is converted to landmarks.

We extract 25 landmarks per frame. Each landmark consists of 4 values - x, y, z and visibility, range from -1 to 1.

A shot is a sequence of up to 50 frames and In case of a short shot, we pad with 0’s.



In our project we use landmarks number 1-24. Those Landmark represent the upper part of the body since the lower part most of the time is invisible because of the table.

## Extraction:

We create a signal based on the landmarks (‘mxy’ - blue graph) ,This signal describes the **mobility** of the right hand.

In each frame we look at a window around it, There are 6 landmarks of the right hand, each has x and y coordinates. For each coordinate we average the diff between those values, in the window E.g. diff of landmark 16’s x = [0.048, 0.05, 0.052] - > [0.01, 0.02], E.g. the avg is 0.015.

We average the right-hand landmark’s previous averages to a single value, 6 averages for x and y per landmark.

To determine if a shot has started or finished, we use a dynamic threshold (purple line), The threshold is calculated each frame, just like the signal.

Threshold = P \* the avg of the signal so far.

Each player has a very different “shot profile” depends on his speed, reaction time, technique etc.

Dynamic threshold enabled the extraction for different inputs. At the end we have a list of shots and Landmarks are saved during the process.

Legend:

Green dots represent the start of a shot, and red dots represent the end of it.

The purple horizontal line is the dynamic threshold calculated at the end.

Chart

Description automatically generated

Chart

Description automatically generated

We can see 2 plots for two different players – we can see the first player is a bit faster then the other - the resting time between each shot is shorter.

# DL model:

The input from the previous step is a shot of shape [50, 100] - 50 frames (sequence), each frame has 100 values represent the 25 landmarks.

The model is a sequential model based on multi layers bidirectional LSTM with attention and Fully connected layer (hidden dimension -> 2 ) on top + log soft max

### Training:

We use Negative log likelihood loss, Adam optimizer, Step Lr scheduling and Early stopping.

# Data augmentation:

Our data set is very small and not balanced and therefore we act as follow:

* 1. Test & validation set created in advance and are balanced
  2. Train set is not balanced

We used data augmentation as part of our up-sampling technique

Pseudo code:

For each frame

For each right-hand landmark

X, y, z += Rand.Uniform[-0.01, 0.01]

\*Both frame, landmark and coordinates are chosen in random

Chart, line chart

Description automatically generatedChart, line chart

Description automatically generated

In the frame example above, we can see that in the augmented frame on the right, the right shoulder is a bit more down – person is more leaning, and hand is a bit more open

## Feature selection:

We have implemented a feature selection function which receives array of landmarks, for example [0,1,2,3,4,5,6,7,8,9,10,11] which represent the head of the player.

The function receives the landmark to remove and removes those features.

We experimented with all features vs. no head features and came up with the conclusion that all the features lead to better results in terms of accuracy.

# Results & Examinations:

We first split the data to train, test and validation set:

* Fts splitting: test + val – 75/75. Train – 118/953
* Fco splitting: test 50/50, val 32/32. Train – 80/2057

Experiments made during the development:

* + fixed test & validation set
  + fixed hyper parameters (64 hidden dim, 2layers etc.)

we have made many experiments, since we can't display them all we choose 5 which present the major improvement in our process:

* Each one of the tests made for each one of the shot types (‘fco’ and ‘fcs’)

1. Simple settings –without up sampling and attention layer.
2. simple up sampling – clone samples from minority class
3. Augmented up sampling – we use our data augmentation method
4. Augmented up sampling + adding the attention layer
5. Simple up sampling + attention layer

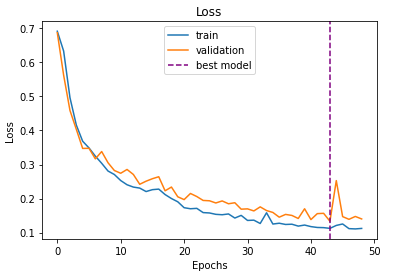
Summary of the 5 experiments represent in the following graph:

Best result received by the following hyper parameters:

Lstm with attention layer, all features selected – all landmarks were chosen, data augmentation up sampling used, and bidirectional.

* Batch size = 32
* Learning rate = 0.0002
* step size = 10
* gamma = 0.5
* hidden dimension = 64
* number of layers = 2

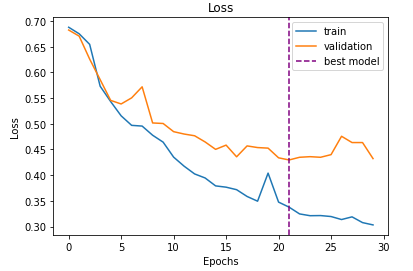
Chart, histogram

Description automatically generated

Chart

Description automatically generated

Chart, line chart

Description automatically generatedFts Results –

