University of Pireaus

Master Degree (M.Sc.) in Informatics

Thesis

# Video classification with Recurrent Neural Networks
# -
# Ταξινόμηση βίντεο με Αναδρομικά Νευρωνικά Δίκτυα

Supervisor: Assistant Professor Aggelos Pikrakis

Student Name: Leoniditsa Chondromatidou

Father name: George

Student Number: ΜΠΠΛ14093

Delivery Date: 30-09-2019

**Examination Committee**

-

**Τριμελής Επιτροπή**

| | | |
|---|---|---|
| *Assistant Professor Aggelos Pikrakis* | *Professor Themistoklis Panagiotopoulos* | *Associate Professor Dimitrios Apostolou* |
| - | - | - |
| *Επίκουρος Καθηγητής Άγγελος Πικράκης* | *Καθηγητής Θεμιστοκλής Παναγιωτόπουλος* | *Αναπληρωτής Καθηγητή Δημήτριος Αποστόλου* |

# Contents

## Abstract

The present project aims to conduct video classification by training a network of stacked LSTM cells to recognize the sport being conducted in a subset of *Sports-1M Dataset*. The contribution of this project is that unlike traditional methods on video classification, that feed frame-images to the network, it attempts to use Carnegie Mellon's OpenPose pose-estimation library, to extract human poses from a predefined number of frames and use them as input features to the network. This effort intends to help the network identify and learn movement patterns from each sport. The main challenge of this undertaking was that *Sports-1M Dataset* is a machine generated dataset, that contains user-produced videos and therefore is susceptible to noise. The latter comes from possible unrelated videos mistakenly selected by YouTube's annotation system or the users not focusing on the sport carried out, but instead zooming randomly into the crowd, the face of a player, zooming out on the empty field etc. Apart from common difficulties unconstrained videos introduce, such as varied illumination, scale, camera motion, viewpoints etc., this dataset also varies substantially in duration and resolution. The approach, followed to counter the aforementioned challenges, was to define a fixed window of 30 frames for each video (2 frames per second - aka 15 seconds of video), with the selection beginning after 30% of video's run time, in order to increase the probability of encountering the sport in action. Furthermore, to control the quantity and quality of the people selected from each frame, the people were filtered through an *index of interest*, which quantifies how big, complete and central each person is, in relation to insignificant ones in the frame and use that as a rule to pick the 2 most *interesting*. Finally, after hyperparameter investigation, the network was able to produce 89% accuracy, for 5 sport-classes and 73% for 10 sport-classes. This was achieved through a network of stacked LSTM cells, of 64 and 32 units in depth respectively, with L1,L2 regularizers applied at each layer, followed by a densely connected Neural Network with the same amount of units, as the sport-classes.

## Περίληψη

Το παρόν έργο έχει ως στόχο να διεξάγει ταξινόμηση βίντεο, εκπαιδεύοντας ένα δίκτυο αποτελούμενο από στοιβαγμένες stacked LSTM μονάδες, έτσι ώστε να αναγνωρίζει το άθλημα που εμπεριέχεται σε ένα υποσύνολο των βίντεο του Sports-1M Dataset. Η συνεισφορά αυτής της εργασίας είναι ότι σε αντίθεση με τις παραδοσιακές μεθόδους ταξινόμησης βίντεο, όπου τροφοδοτούμε το δίκτυο με καρέ-εικόνες, εδώ χρησιμοποιούμε την βιβλιοθήκη του Carnegie Mellon OpenPose, με σκοπό να αποσπάσουμε τα ανατομικά σημεία των ανθρώπων (στάση σώματος) από έναν προκαθορισμένο αριθμό διαδοχικών καρέ, ως μεταβλητές εισόδου του δικτύου. Η κυρια πρόκληση αυτού του εγχειρήματος είναι ότι το Sports-1M Dataset έχει δημιουργηθεί με αυτοματοποιημένο τρόπο και η λήψη των βίντεο έχει γίνει από απλούς χρήστες, με αποτέλεσμα αυτά να περιέχουν θόρυβο. Ο θόρυβος προκύπτει είτε από μη σχετικά βίντεο που έχουν επιλεχθεί λανθασμένα από το σύστημα παρατηρήσεων του YouTube, είτε από χρήστες που δεν επικεντρώνονται στο άθλημα που λαμβάνει χώρα και σε ανυποπτο χρόνο κάνουν ζουμ στο κοινό, πρόσωπο του αθλητή, περιβάλλον χώρο κλπ. Ταυτόχρονα το παρόν σύνολο δεδομένων, εκτός από τις καθιερωμένες δυσκολίες που παρουσιάζει ως σύνολο ελεύθερων βίντεο, όπως η έκθεση σε διαφορετικούς φωτισμούς, κλίμακες, μη σταθερή κάμερα, σκοπιές κλπ. έχει και μεγάλες αποκλίσεις ως προς την διάρκεια και ανάλυση των βίντεο. Η προσέγγιση που ακολουθήθηκε, για την επίλυση των παραπάνω, ήταν για κάθε βίντεο να οριστεί ένα σταθερό παράθυρο τριάντα (30) καρέ προς εξαγωγή (2 καρέ ανά δευτερόλεπτο - δηλαδή 15 δευτερόλεπτα βίντεο συνολικά), ξεκινώντας από το 30% του βίντεο, με σκοπό να αυξηθεί η πιθανότητα να πετύχει το άθλημα όταν αυτό εκτελείται. Επιπρόσθετα, για να τεθεί υπο έλεγχο η ποσότητα και ποιότητα των ανθρώπων (ανατομικών σημείων) που βρέθηκαν στο κάθε καρέ, οι άνθρωποι φιλτράρονται από έναν δείκτη ¨ενδιαφέροντος¨, που ποσοτικοποιεί το πόσο μεγάλοι σε μέγεθος, ολόκληροι και κεντραρισμένοι είναι σε σχέση με κάθε άλλο άτομο στο καρέ. Το παραπάνω χρησιμοποιείται ως κανόνας για να επιλέξει τους δύο (2) πιο ενδιαφέροντες ανθρώπους. Κλείνοντας, ύστερα από αναζήτηση των παραμέτρων εκείνων που θα βελτιστοποιούν το εν λόγω δίκτυο, ηταν σε θέση να παράξει αποτέλεσμα ακρίβειας (accuracy) 89% , με 5 κλάσεις και 73% με 10 κλάσεις. Το δίκτυο αυτό απαρτιζόταν από 2 στοιβασμένες LSTM μονάδες, με βάθος 64 και 32 κρυμμένων μονάδων αντίστοιχα και ρυθμιστές (regularizers) L1,L2 σε κάθε επίπεδο. Τέλος, η κατηγοριοποίηση ολοκληρώθηκε από ένα πυκνά συνδεδεμένο νευρωνικό δίκτυο με τόσες μονάδες όσε ήταν και οι αντίστοιχες κατηγορίες αθλημάτων.

# Chapter 1

# Introduction

A video is a series of sequential-in-time images (frames). Video classification is the task of labeling a video with its assigned label either on the frame-level or for the whole video [75]. Video makes an interesting classification problem because apart from spatial information (captured on the 2D frame level), it adds temporal information encoded in the sequence of frames, revealing the relations between its frames. The data whose points are dependent on other points are called *sequential data* [23] and different Machine Learning (ML) methods were developed in order to capture those relationships. Recurrent Neural Networks (RNNs) is one of those methods derived from the *connectionist* ML school of thought and is an alteration of the conventional neural network (NN). It consists of a network of neuron-units (cells/blocks) that a weight to each connection between the units is assigned, representing how strong/weak the connection is. Then these weights are adjusted by the *backpropagation algorithm* in proportion to the gradient error, so that the intended output matches the network output. The main idea behind RNNs is to make the same computations for all the data points in the sequence, adding each time to the datapoint currently being processed, all the preceding frames' outputs [101]. In that way it is possible to create a persistent-memory, where each datapoint takes into account all past datapoints of the sequence, the same way a person understands the meaning of a word based on the comprehension of the preceding words in a sentence.

The achilles heel of RNNs though is the *Vanishing-Exploding Gradients*. This problem occurs when the weights turn out to be very small or very large preventing the network to learn high temporal distance relationships between data [101]. To overcome this problem, new RNN variations were introduced. One of which was *Long Short Term Memory* network (LSTM). The latter consists of three (3) gates: *Input*, *Output* and *Forget*, rensposible for writing, reading and resetting the unit, using an analog (sigmoid) and not binary activation function, which keeps the gradients steep enough and permits the network to remember very long sequences [101].

The aim of this project is to carry out video classification utilizing an LSTM network with the sequential data to be the human poses extracted from a predefined number of frames per video. The pose estimation of each frame is accomplished via the use of OpenPose library [14], developed by Carnegie Mellon University. The video dataset selected is the Sports-1M Dataset [42], which consists of realistic, user produced videos, publicly available on the internet (videos "in the wild"). This dataset provides the opportunity of testing the classification on real world data, but it comes at a cost of added complexity due to the varied illumination, scale, camera motion, viewpoints etc. This effort attempts to identify the patterns of movements that correspond to specific sports and belongs to *action recognition* problems.

# Chapter 2

# Related Work

The domain of *action-recognition* and *action-detection* come a long way with years of research. The terms *action recognition* and *action detection* sometimes are mistakenly used interchangeably. Soo and Wildes [55] explain that *action recognition* refers to the act of classifying an action in a video and *action detection* concerns locating actions of interest in space and/or time. The present project is about *action recognition.* On the latter, different taxonomies have been presented [57], that attempt to define what is a *movement, action, activity* etc. and what's the hierarchy between them. Moeslund et al. [57] uses a hierarchy that consists of *action/motor primitives, actions* and *activities.* The *motor primitive* is any joint movement, an *action* is a set of *motor primitives* and an *activity* is a set of actions. In the present project we use action as sport-action. Poppe [68] organizes the approaches of action recognition domain under *image representation* and *action classification*, since they are common parts of all action recognition approaches in literature.

## Types of features

*Image representation* refers to the process of extracting useful information from raw video data (feature extraction) and representing them (encoding) in a form that is suitable for classification. Poppe [68] divided the different approaches of representation into two categories: *global (top-down)* and *local representations (bottom-up).* The first, were attempting to encode the visual observation as a whole, by localizing the person first through background subtraction or tracking. Common descriptors extracted from that category are *silhouettes* or *optical flow*, for when background subtraction is not performed. The second, were attempting to encode the visual observation as a collection of local descriptors or patches. The latter were divided by whether they were derived via *dense* or *sparse* sampling.

Soo and Wildes [55] make a similar distinction between *dense* and *sparse* sampling. They address *dense sampling methods* as the methods that divide videos into either rectilinear patches or more irregular supervoxels via techniques like *mean shift* [44], *streaming hierarchical supervoxel method* [97] and *SLIC* [13]. Similarly they address *sparse sampling methods* too, as the methods that sparsely extract interest points (*spatiotemporal interest points - STIPs*) or regions via a regular grid. This last group of methods is further broken down between the methods that extend an already-known 2D detector into 3D in videos (like *Harris* to *Harris 3D* [38, 49] and *Hessian's* matrix in 3D [4]), those that use a different detector per dimension (spatial and temporal) (like *cuboid detector* [24]) and *tracking-based detectors* that use *good-features to track* criterion [76] (like *dense trajectory features* [30, 91]).

On top of the previous disctinction mentioned, Soo and Wildes [55] presented another taxonomy of the same methods based on the type of *feature descriptors* each produces. They distinguish between the *General primitive features*, that the features derived from raw input videos don't require any additional processing and the *Specialised primitive features* that do.

*General primitive features* are divided into three main sub-categories: *filter, flow* and *convolutional neural networks (CNN)-based.* The *filter-based* methods try to represent an object's shape and appearance by the local intensity gradients and edge direction (*gradient-based*), or by using the dimension of local orientation and scale (*spatiotemporal oriented bandpass-based*). Some of the most known *gradient-based* techniques are *Histogram of oriented gradients - HOG* [21, 50], HOG3D [46], cuboid descriptor [24], scale-invariant feature transform (SIFT) [27] etc. Respectively, some known *spatiotemporal oriented bandpass-based* techniques are *3D Gabor filters* [15] and *Gaussian derivate filter* [39]. The *flow-based* descriptors try to represent motion information. Some of the most known techniques are *Kanade-Lucas-Tomasi - KLT* [65], *Histogram of Optical Flow - HOF* [38], *Motion Boundary Histogram - MBH* [30]. Lastly, *CNN-based* methods depend on a neural network and its architecture based either in spatio-temportal convolutions, recurrent neural networks or two-stream architectures.

*Specialised primitive features* are divided between *silhouette-based* and *tracking-based.* *Silhouette-based* filters extract silhouettes which are either converted into an 1D signal via *R transform* [83, 93], converted into binary images called *motion energy images - MEI* or scalar images called *motion history images - MHI* that are then described using *Hu moments* [36], or stacked to form *spacetime volumes - STV* [11, 99]. Lastly *tracking-based* methods attempt to track the trajectory of the entire actor to segment him from the background, or by tracking body parts. On the latter, Poppe [68] stress that joint locations are difficult to derive but they constitute rich representations.

Recent work is using a mixture of filter and flow-based algorithms together and are directed towards CNN algorithms [8, 40, 43, 77, 77, 87, 92, 100]. Additionally, since Poppe's survey there's been progress on pose-estimation problems and approaches have managed to succesfully locate anatomical human keypoints - joints with higher accuracy [14, 29]. In this project, we use as feature extraction technique the OpenPose library of [14], that uses two-stream CNNs and through *heatmaps* and *affinity fields* is able recover very accurately the human joints of the video actors. A big part of action-recognition with pose features was using mainly 3D skeleton features which were restrictive to situations where such 3D data could be acquired and used [37, 52, 89, 96]. Later, more 2D pose features with motion information were introduced [17, 18], with the most similar to the current project to be [16], where the same OpenPose library is used to locate human joints, colorize them based on the relative time of the frame from which the joints were acquired and aggregate them to represent the clip-level pose motion as features to a shallow CNN. The present approach is passing as an input to an LSTM network, the (x, y) points of each joint as provided by [14] [102], with no further motion information and tested on raw videos *in the wild* provided by the Sports-1M Dataset [42].

### Types of Classifiers

Classification is when a training dataset is used to predict the pre-defined classes of a test dataset. Soo and Wildes [55] break down the classifiers between *Deterministic*, where the class of a test data input is predicted without taking into account the probability distribution between classes of the training data and *Probabilitistic*, where is not. *Deterministic classifiers* are divided further in to *lazy* and *eager* learners. To the *lazy learners* belong the classifiers that postpone data processing until they receive a request to classify, like *K-nearest neighbor -KNN.* To the *eager learners* belong the classifiers that generalize the data on the learning process even before is attempted to classify the test data, with models like *Support Vector Machines - SVM*, *Adaptive Boosting - AdaBoost* and *Artificial neural networks - ANNs & CNNs.* Respectively *Probabilistic classifiers* are divided further to *General classifiers*, and *Temporal state-space models.* The latter use temporal information of features, while *genetal classifiers* don't but rather use probabilities to map the features to their action class . Known general classificiers are *Naive Bayes* and *Relevance vector machines - RVMs* and for *Temporal state-space models* are *Hidden Markov Model(HMM)* and *dynamic Bayeasian network - DBN.*

In the present project LSTMs are used as a classifier. LSTMs have been used in action-recognition literature in [53, 94, 95, 100], to model longer temporal dependencies of the actions.

# Chapter 3

# Long short-term memory - LSTM

Long Short Term Memory (LSTM) networks are an extension of Recurrent Neural Networks (RNN) initially created for Machine Learning problems that involve sequential data. LSTM was proposed as a solution to the problem of *Vanishing-Exploding Gradients* by extending the memory of vanilla RNNs and learning longer sequences. In order to understand LSTM is important to explore where it comes from and how it differentiates from plain RNNs.

## LSTM on the Artificial Intelligence map

Artificial Intelligence (AI), Machine Learning (ML), Artificial Neural Networks(ANN or NN) and Deep Learning (DNN) are some of today's most popular terms often confused and mistakenly used interchangeably. The reality is that AI is the field of science and engineering that encompasses the rest of the terms aiming to understand and most importantly to build intelligent entities [71]. The idea of what constitutes intelligence though can vary substantially depending on the scientific discipline trying to answer that question and therefore all these other terms are just different approaches to accomplish AI.

The different theories of intelligence can be grouped into 4 main categories, each of which claims that a program is intelligent if is [71]:

- **Acting Humanly:** A program can imitate human intelligence in such degree that a human can't tell the difference (Empirical Science-Turing Test)
- **Thinking Humanly:** The input-output behavior of a program matches the corresponding human behavior according to the known human mind theories and this acts as evidence that the program mechanisms operate like humans and therefore is intelligent (Interdisciplinary Cognitive Science and psychology)
- **Thinking Rationally (laws of thought):** The program proves to have logic in the Aristotlean way of "right thinking", where given the correct premises it yields the correct conclusions following *syllogisms* (patterns for argument structures)(Logicist tradition)
- **Acting Rationally:** A program acts in a way to achieve the best outcome or when there is uncertaintly the best expected outcome. This does not focus exclusively on correct inferences as the *laws of thought*, but minds it as a part of possible mechanisms to achieve rationality.

The different scientific fields and their distinct perception of intelligence gave rise to different schools of thought on how a computer can learn more efficiently. These different ways to achieve *intelligence* are studied through the field of ML, which is a subfield of AI and is defined as the study of algorithms that improve automatically through experience [56].

The main 5 school of thought in ML emerged are [25]:

- **Symbolists:** They believe that intelligence can be represented as symbols that can be manipulated the same way mathematical equations do. Their main technique is *inverse deduction*, which seeks what knowledge is missing in order to make a deduction and generalize. This school of thought is coined as G.O.F.A.I. (Good Old Fashioned AI) [32] and is connected to *laws of reasoning* mentioned above. Modern application of this school is expert systems.

- **Connectionists:** They believe that the brain is the most efficient learning tool in nature and it should be reversed engineered. Their approaches are inspired by neuroscience and the prefrontal cortex paradigm, which learns by firing and creating connections among its neurons. The problem connectionists are trying to solve is to figure out which connections are to blame for the errors and adjust them accordingly. Their main technique is *backpropagation*, which compares the system's output with the desired one and updates the connections of the artificial layers in a way to minimize the error. Modern algorithms of this school are Neural Networks and Deep Architectures.
- **Evolutionaries:** They believe that the most intelligent mechanism in nature is natural selection, through which everything was created. They focus primarily on learning structures and not only adjusting parameters (like backpropagation). Their approaches are inspired by evolutionary studies and their main technique is *genetic programming*, where programs are encoded as a set of genes that can be modified using the *genetic algorithm.*
- **Bayesians:** They believe that everything is uncertain and even learned knowledge is a form of uncertain inference. The problem Bayesians attempt to solve is to deal with noise, incomplete and often contradicting information. Their approach is inspired by *probabilistic inference*, *derivatives* and *Bayes Theory.* A modern algorithm of this school is Naive Bayes.
- **Analogizers:** They believe that intelligence lies on recognizing similarities between situations. Modern algorithm of this school is Support Vector Machines, which distinguish what is important to remember in order to make the correct prediction.

Via this categorization, it is well understood that AI is the field that aims to build intelligent entities, ML is a set of algorithms that are trying to fulfil AI and ANN, DNN are approaches of ML belonging to connectionist school of thought. The ANN have received great attention the past years, although its algorithms were introduced in the public (see TV show headline in 1960s in Figure 3.1 [48]) back in 1958 (birth of neural networks with *Perceptron*) and 1986 (*backpropagation*) [33].

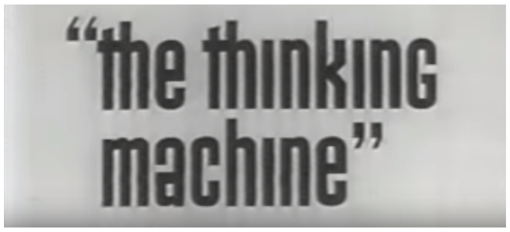


Figure 3.1: TV show headline about Perceptron in 1960s - repost from [48]

This rise of Artificial Neural Networks was caused by the increasing computational power and accumulation of data that could actually realise the potential of those algorithms. LSTM belongs to the connectionist school [51] of thought and it is a progression of Recurrent Neural Networks, which are an alteration of Neural Networks. LSTM follow the same principles as NNs and is critical to understand how they work and how LSTM networks are different.

## How Neural Networks work

The idea of the Artificial Neural Networks starts with the idea of the *perceptron*, which was introduced in 1960s by psychologist Rosenblatt [48]. *Perceptron* was his attempt to mathematically model how a single brain cell operates. The network of brain cells consists of millions of neurons that transmit information via electro-chemical signaling and form connections (synaptic connections) depening on which neurons are *inhibited* or *fired*. The same way, the *perceptron* receives several inputs, it performs a weighted summation and a binary classifier determines whether the value suffice for the cell/neuron to *fire* -1 or not (*inhibited*) - 0.
The perceptron was actually a *binary classifier*, a function that mapped its inputs (a real valued vector) to an output (1 or 0). One perceptron though could not represent more complex problems on its own and

for this reason a *multilayer perceptron* or *neural network* was created. A *multilayer perceptron* is simply a network with more than one layer of perceptrons. The more the layers, it resembles the solving ability of the human brain (prefrontal cortex) and is often referred to as Deep Learning.

Each layer is a function that serves as an input to the next layer. The more we move forward the levels of layers the more the information is abstracting as seen in figure Figure 3.2 [31]. Consequently, the *neural network* is a chain of composite functions [63] and its main purpose is to find the right weights to which the inputs would result in the desired output (*optimal feature representation*).

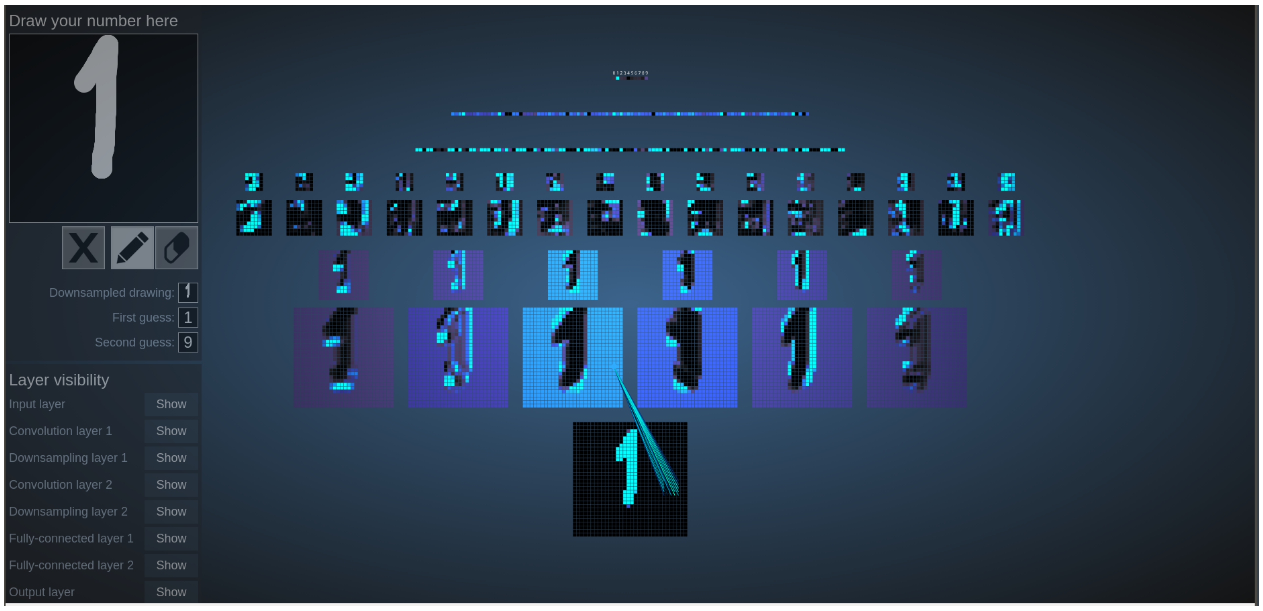


Figure 3.2: Visualized feature abstraction per level of layer of a hand written digit-derived from Convolutional Neural Networks - implemented by Ryerson University [31]

The way the neural networks manages to optimize the composite function is via *feed-forward* process and *backpropagation*. The first is the process of moving forward the information from input nodes through the levels of layers until the output node, without any cycles or loops and the second consists of the process of moving backwards (from output to input), in order to update the weights in a way to improve the next prediction. More Specifically:

**Feed Forward Process:**

The feed-forward neural network initially sets a fixed number of neurons/cells/nodes/units to represent the input values, then the desired amount of layers (also known as *hidden layers*) and finally the desired number of output nodes. Each input node is connected to all other nodes of the following layers and each connection has a random weight assigned to it. Each input node represents an independent variable, a feature we think it will contribute to the solution of the problem

On the first step of the training process each node of the first hidden layer calculates a weighted sum of the input nodes, adds a predefined bias and if the produced value exceeds the required threshold of the activation function we have assigned, it *fires* and *activates*, otherwise to remain inactive [88]. When this process has been completed for all the hidden layers, the network outputs an estimated value. This value will be compared to the value it should have predicted (the actual value) and generate the difference between them (the error) via the *cost function*. Finally, through backpropagation the weights are adjusted in a way that improves future predictions. Ultimately, the goal of the network is to *minimize* the cost function.

**Backpropagation:**

Intuitively speaking, *backpropagation algorithm* is trying through trial and error to teach the network what the value of each weight should be in order to output the right value. For each input (*iteration*) the network makes a prediction and this prediction is compared to the correct value (or else *ground truth)* to find how far the estimation was from the truth. This distance from the truth is also known as the *error* and is calculated via the *cost function*, that reflects how wrong the network is. The goal is to *minimize the cost function* and therefore decrease the distance from the ground truth. The way backpropagation minimizes this error is by the use of optimization algorithm *Gradient Descent*, which technically speaking searches for the local or global minima of a function. Intuitively speaking, gradient descent points the direction (or *gradient*) the weights in the network should take so that error drop down to zero, or else to *converge*. At each iteration the weight's value is adjusted in a way and direction to reach zero error. How fast the network will converge is defined by the *learning rate*, which adjusts how big the steps towards zero error would be.

Mathematically speaking, the network is actually a function of multiple or nested functions (composite function), where each layer represents a simple function whose inputs are a weight vector and the outputs of the previous layer [60]. So the *backpropagation algorithm* is accounted to optimize a differentiable function. To achieve that, it is required to figure out the partial derivative of the error in respect to each individual weight in the network [22]. The way backpropagation achieves this is via the mathematical formula of *Chain Rule* (see figure Figure 3.3 [10] ). Chain rule is applied for all the possible paths in the networks, to find the gradients of each weight in respect to the output, in order to update them [70].

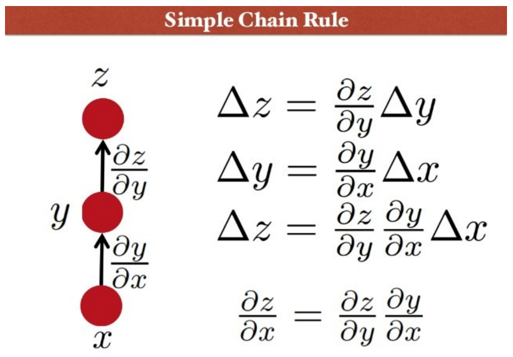

Simple Chain Rule
$z$
$y$
$x$
$\frac{\partial z}{\partial y}$
$\frac{\partial y}{\partial x}$
$\Delta z = \frac{\partial z}{\partial y} \Delta y$
$\Delta y = \frac{\partial y}{\partial x} \Delta x$
$\Delta z = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x} \Delta x$
$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \frac{\partial y}{\partial x}$

Figure 3.3: Chain Rule - repost from [67]

The update of the weights is performed via *Gradient Descent* mentioned before, which given the network's output calculates the error of each neuron in the hidden layers moving backwards layer by layer. It finds how much each neuron in a hidden layer is responsible for the output's error by summing up the products between the errors of the neurons in the next layer and the weights of the connections to those neurons multiplied by the derivative of the activation function [10]. These errors are used to calculate the variation (delta) of the weights as a result of the current input pattern and ideal outputs [70]. At the end of each iteration the weights' values are adjusted to fit the accumulated Deltas multiplied by the learning rate [70] (see formula below Figure 3.4 [70] ) and this is how the network learns.

$$w_{ij} = w_{ij} + ( \Delta w_{ij}\_Final * LearningRate )$$

Figure 3.4: Formula that computes the new weights

## How Recurrent Neural Networks are different to Neural Networks

Multilayer Feed-Forward Artificial Neural Networks concept is one of the most widely used tools in Machine Learning, but its limitation is that it cannot handle sequence data. These are the data whose points are dependent on other points [85] such as audio, text, video, time series, financial data, DNA etc. The reason ANNs can't handle that type of data is because they have no memory and therefore their architecture doesn't allow them to remember past outputs. The signals were designed to travel only one way, from input to output without the output affecting the layer that produced it. This inability to remember past events though, disregards all of the temporal information of sequence data [85]. Human beings don't think from scratch, but they rather associate past events to recent information. So they have *persistent memory*. To imitate that, RNNs were introduced, or else *Feedback Artificial Neural Networks* that allowed signals to travel in both directions, giving a feedback to the node of its previous outputs [85]. As Christopher Olah defines it [64]: *a recurrent neural network is multiple copies of the same network, each passing a message to a successor.*

In reality, the recurrent cell is the same as the standard feedforward cell but it only adds the idea of loops to the architecture. The recurrent cell has an internal memory, in a sense that outputs of the network are conditional on the recent context in the input sequence and it's output is taken into account by the next input. It achieves that by feeding the previous states back to the network. In that way the RNNs take advantage the ordered nature of sequence data.

By the same token, back propagation is conducted the same way as in conventional feedforward networks with only a small difference, since the parameters are shared for each instance. The calculated gradients depend not only on the processed input but on all the previous ones too [101]. In that way the input sequence is considered to be a single element of the training set. So the error gradient for an input sequence is actually the sum of error gradients at each time in the sequence, that is also called **the Backpropagation Through Time** (BPTT) [101]. Unfortunately though, since the layers and time steps in RNNs are related to each other through multiplication, derivatives are sensitive to a known problem called *Vanishing* or *Exploding Gradients*. The weights' gradients can start to become very small or very large through the numerous multiplications for computers to learn. When sigmoid is applied to data multiple times it becomes flat with almost no slope. The answer to this problem was *Long Short-Term Memory* (LSTM) network

## How LSTM Networks are different to Recurrent Neural Networks

Solving the problem of Vanishing Gradients requires an architecture that allows to selectively forget states and information that were not important. LSTM cell is a solution to this problem proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber in 1997 [101]. It is based on its ancestors (perceptron) but in reality its architecture is far more sophisticated.

The main idea behind the LSTM unit is that there is a separate *memory* or else the *cell state* [51], that is being altered and updated by three (3) distinct *gates* that regulate what is being kept and what is discarded from the current cell state. These gates help to avoid possible weight conflicts and separate them from the main cell state - memory [35]. The unit takes as an input the *current input, previous output* and *previous memory-cell state* and generates a *new output* and a *new/altered memory-cell state* for the next LSTM unit and sequence input to process [98] (see figure Figure 3.5 [98]).

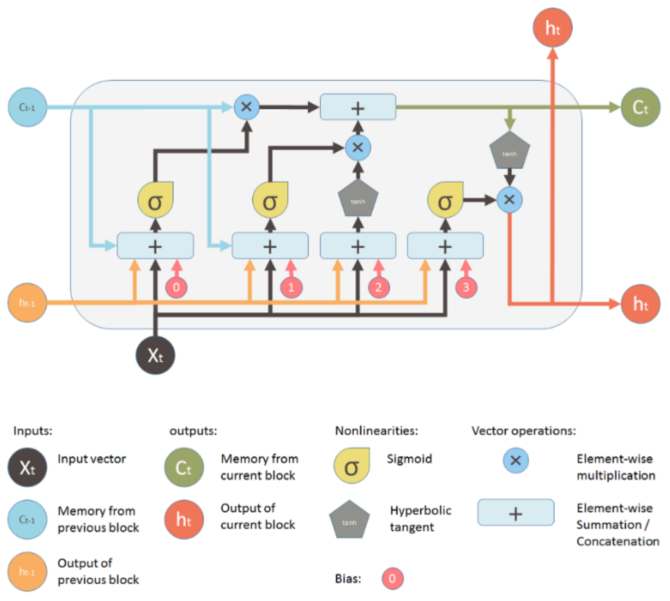


Figure 3.5: LSTM unit - repost from [98]

More specifically the function of each gate is:

**Forget Gate Layer:**

This gate evaluates the previous cell state and decides what information will be discarded from the old cell state. The decision is made by a *sigmoid* function and outputs a number between 0 (discard completely the old cell state) and 1 (keep entirely the old cell state) for each number in the old *cell state*. This is an one layer Neural Network with a sigmoid activation function (see below [64] ).

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

**New Memory Gate Layer:**

This gate consists of two separate steps and evaluates which information will be stored in the cell state and updates them. The first step is called the **input gate** and is responsible to select which values of the old cell state - memory will be updated. This is achieved through an one layer Neural Network with a sigmoid activation function. The step is responsible to create new candidate values of the cell state through a different Neural Network layer and the information of what should be stored in the cell state are ready (see below [64]).

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

The new cell state is updated by multiplying the old cell state, with the output of the FG, in order to keep the amount of the old cell state it was decided and then add the amount of information it was decided to be stored via the *New Memory Gate* (see below [64]).

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

**Output Gate Layer:**

This is the last layer of the LSTM unit and is responsible to form the new output. The previous output and sequence input are filtered through an one layer Neural Network with a sigmoid activation function in order to decide which parts of the cell state will be selected and the output is multiplied by the new cell state after being rescaled between the -1 and 1 via a tahn activation function (see below [64]).

$$o_t = \sigma\left(W_o \; [h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

# Chapter 4

# Pose Estimation

## Pose Estimation Problem

Pose estimation is the problem of localizing anatomical keypoints ("parts") or human joints [14] [86] in order to recover a representative layout of body parts from image features [61]. The problem poses challenges on both hardware and software level. On hardware level, different viewpoints of the same pose can create different appearances and often special equipment is required such as 3D depth camera, infrared camera, multiple cameras or specialized equipment such as Kinect or Motion Capture [73]. On software level, some of the basic challenges are the unknown number of people that appear on the frame and could potentially increase the runtime complexity and false positives, along with partially occluded limbs that make part detection trivial.

Moreover, until recently, there was a shortage of quality datasets that included people with their body parts annotated, making the problem even more difficult to resolve. This gap was breached by the arrival of *COCO dataset* [1], *MPII dataset* [6] and *VGG dataset* [2]. Each one of those introduced their own way to represent the human pose, with different body landmarks. More specifically, as seen in Figure 4.1 [1], the COCO dataset uses 18 points (17 human joint points and 1 for the background), while MPII uses 15 (14 human joint points and 1 for the background). These datasets and Coco's Keypoint Detection Task [2], invited more people to approach the pose estimation problem.

There are two main approaches to pose estimation problem: a *top-down* and *bottom up approach* [61] [14]. The *top-down approach* aims to localize the objects-humans first and then estimate their joints-limbs individually. Although this approach seems intuitive, it conceals the danger of overall failure in case the person is not detected or detected incorrectly ( ex. two people being classified as one person within a bounding box - very likely when close interactions take part). In addition, the computational time is highly correlated to the number of people in the frame. On the other hand, *bottom-up approach* is done on pixel-level (taking into account all pixels of the frame) and aims to identify body-parts first. The pixel-level analysis is important since it forms a global context of the frame (missing from previous approach), so that it is easier to distinguish between people. After the body parts are detected, they are combined into its human-pose representation.

In this project, the method used for pose estimation is based on Carnegie Mellon University's paper *"Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields"* and belongs to bottom-up approaches. The paper has released the implementation of their method through the creation of a C++ library called *OpenPose*.

[1]The COCO body landmarks were retreived by the official github of paper mentioned in Ubid., cit. 1(15) url: "https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/output.md". The MPII's body landmarks were drawn based on the MPII guidelines for showcasing purposes

[2]COCO Keypoint Detection Task requires localization of person keypoints in challenging uncontrolled conditions, url: http://cocodataset.org/#keypoints-2017

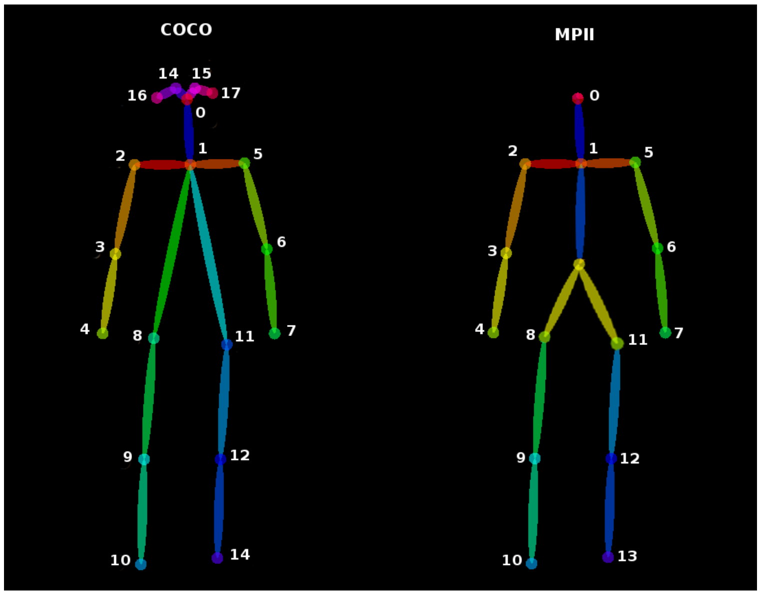


Figure 4.1: Body Landmarks of COCO and MPII datasets

## OpenPose

OpenPose [14] is a C++ library using OpenCV and Caffe [41] developed by Carnegie Mellon University using COCO and MPII datasets. It includes three sets of pre-trained models ( for body, hands and faces' pose estimation). Each set has several models depending on the dataset they have been trained on COCO or MPII [80]. OpenPose was the winner of 2016 MSCOCO Keypoints Challenge, 2016 ECCV Best Demo Award, and 2017 CVPR Oral paper. Available implementations other than C++ are Tensorflow (using CPU), Pytorch, Chainer, MXnet, MatConvnet and CNTK. In June 2018, it was released a Python API for OpenPose [19]. In this project the Tensorflow implementation [45] (using CPU), along with OpenPose's Python API are compared and used.

As stated in the official github README.md file, OpenPose implements *bottom-up approach for realtime multi-person pose estimation, without using any person detector.* In a nutshell, the strategy the paper follows is to detect the position of each of the body parts for every person appearing in a frame and represent human pose as a graph of parts. The detection of the body parts and limbs is done through a 2 branch neural network, a new idea introduced called *Part Affinity Fields (PAF).* Moreover, the connection of the limbs to a human pose model is represented as a k-partite graph assignment problem. This is not a new idea and is dating back to Pictorial Structures (PSs), introduced by Fishler and Elschlager [86]. The challenge of this technique though, is that finding the right part connections that forms a full human through a fully connected graph is an NP-hard problem and it could take hours depending on the number of people present in the frame. OpenPose approached this problem in a greedy manner [3], by decomposing the fully-connected graph into a set of bipartite graphs.

The paper includes two models, one trained on Multi Person Dataset (MPII) and one on the COCO dataset. So one can choose the desired body-landmarks output to be like MPII, COCO or BODY_25. The last is a 25 body parts representation (24 joint points and 1 for backgroun) which consists of foot landmarks added to COCO body points, as seen in Figure 4.2 [14].

[3]A Greedy strategy completes a task of size n by incrementally solving the problem in steps. At each step, a Greedy algorithm will make the best local decision it can given the available information, typically reducing the size of the problem being solved by one. Once all n steps are completed, the algorithm returns the computed solution. [34]

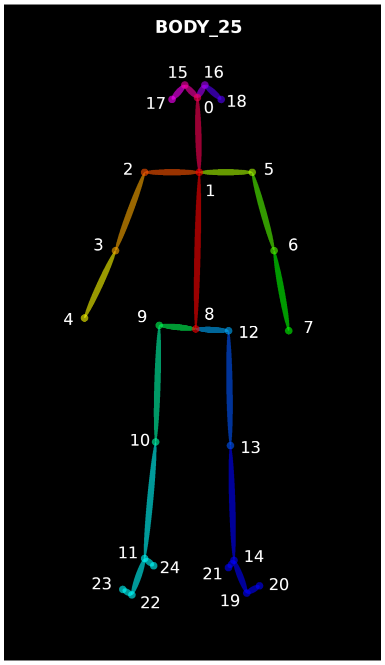


Figure 4.2: Body Landmarks of BODY_25 - repost from [14]

The output is a 25-dimensional array, where each sub-array represents a body-part position in the pixel coordinate system [4]. It is important to examine each step of the OpenPose pipeline and understand all the stages the image-frame undertakes before the result described above is reached.

## Pipeline

The overall Open Pose pipeline is summed up in the diagram below Figure 4.3 [14]:

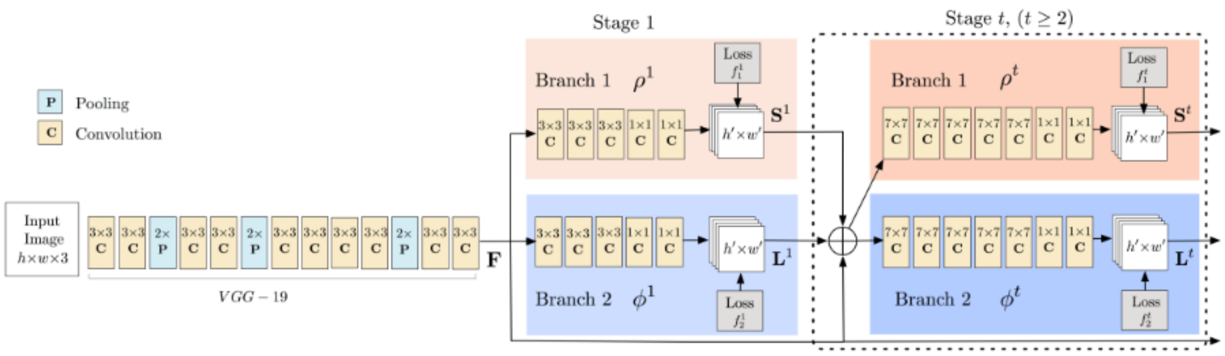


Figure 4.3: OpenPose Architecture - repost from [14]

### Feature Extraction:

The first step of the pipeline is the feature extraction from the input image-frame. This is accomplished via a Convolutional Neural Network (CNN) structured by VGG-19 architecture. There are different architectures a CNN can be structured, but VGG-19 was preferred. The latter is the winner of ImageNet Challenge 2014 and one of the most commonly used architectures. It comprises of a 19 layer depth network that uses very

[4]A digital image is made up of rows and columns of pixels. A pixel in such an image can be specified by saying which column and which row contains it. In terms of coordinates, a pixel can be identified by a pair of integers giving the column number and the row number. For example, the pixel with coordinates (3,5) would lie in column number 3 and row number 5. Conventionally, columns are numbered from left to right, starting with zero. Most graphics systems, including the ones we will study in this chapter, number rows from top to bottom, starting from zero. Some, including OpenGL, number the rows from bottom to top instead. [26]

small (3x3) convolution filters along with aggregating techniques such as pooling [78]. The features are then fed into a two-branch multistage CNN for body part and limb detection.

**Body part and limb detection:**

The features from the VGG-19 Convolutional network are fed into a two branch multi stage CNN, for further body part and limb prediction. To achieve that, a special feedforward network is used as feature extractor that is trying to simultaneously calculate *heatmaps* and *affinity fields*, that are used to detect body joints and associate them to form body parts. More specifically:

- **Heatmap**: [81] (beige color- Branch1) is a detection confidence map, aka a matrix that stores the confidence the network has that each pixel belongs to a certain body joint . If we use the BODY_25 body landmark representation, 25 (+1 for background) heatmaps would be produced, each for every body joint and indexed as mentioned earlier in the chapter in Figure 4.2. So, if there were 2 people in a picture, the heatmap for "head" would had to include very high probabilities for at least 2 pixels, while the heatmap for "elbows" would need to include atleast 4 pixels, since 2 people have 4 elbows in total.
- **Part Affinity Fields**: [81] (blue color - Branch2) is a set of 2D vector fields that encode the location and orientation of the limbs (body parts) . For each part there is a *Part Affinity Field (PAF)* in the *x direction* and one in the *y direction*. There are 50 PAFs for each pair.

At the end of this part of the pipeline, for one picture, the network would have produced 25 heatmaps and 50 (25*2) Part Affinity Maps.

**Non Maximum Suppression:**

After we have the confidence map for each body part, the algorithm *Non Maximum Suppression (NMS)* is applied to turn the pixel probabilities into specific points. The NMS algorithm steps [81] are:

- Take the first pixel of heatmap of x body part
- Surround the pixel with a filter (pixel window) of size 5 and keep the maximum value
- Use the max value to replace the center pixel value
- Slide filter one pixel and repeat in entire heatmap
- Compare original heatmap and new heatmap. The pixels that have the same value in both are the peaks and final points *Suppress* the other pixels by setting their value to 0.

After this process we end up with body part points.

**Bipartite graph:**

In this part, all candidate body parts find the right pair to connect and form a limb. This is solved as a regular assignment problem from graph theory. Since the assignment problem in fully connected graph is an NP-hard problem OpenPose approaches it in a greedy manner, by decomposing the fully-connected graph into a set of bipartite graphs. Each body part is considered a vertix and each pair candidate is considered an edge. All the sets of body parts will be connected with one another and form a complete bipartite graph.

**Line Integral:**

Line Integrals extend the concept of simple integrals by doing what integrals do (only applicable in 2D) but into 3 Dimensions [12]. Integrals help us combine-integrate quantities when multiplication can't, such as changing numbers or vectors [7]. In our case, we have vectors in *Part Affinity Fields* that capture the position and direction from one body part to other, forming possible limbs. With the line integral we can measure their effect, by giving each connection a score that will be saved as a weight between the body parts detected and represented in the bipartite graph.

The candidate limb formed by connection of certain pair of points is examined on whether is aligned with the corresponding PAF and if so, it is considered a true limb. By the end of this process, we would have gathered all possible body part connections broken down into pairs, with its equivalent scores. This will allow us to solve the assignment problem, in order to create the final full body model. A simplified version of the problem is seen in Figure 4.4 [14] below:

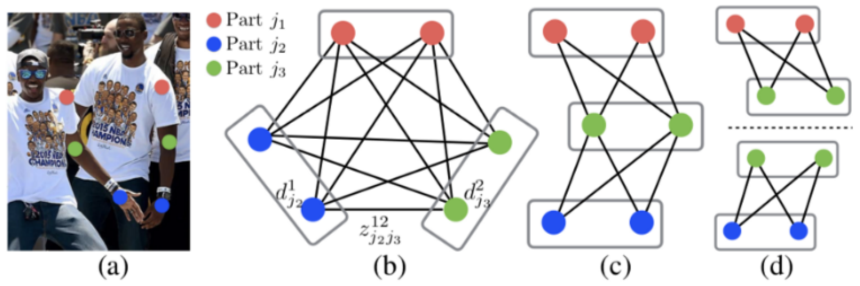


Figure 6. Graph matching. (a) Original image with part detections (b) $K$-partite graph (c) Tree structure (d) A set of bipartite graphs

Figure 4.4: repost from [14]

In the end, this graph forms a matrix and is solved by the connections with the higher scores and others are dropped. By the end of this process we have the body parts and their correct connections between them. It is only left to merge them together.

**Merging:**

The last step of this pipeline is merging, so that we form the final skeleton. The way this is achieved, a naive assumption is made that each part is a Human. So its human has an index, and a coordinate $x$ and $y$, represented into tuples. If there are 2 tuples with the same part index it means that they belong to the same person, so these tuples are merges and the human was represented ceases to exist. This process repeats until there are no more tuple parts.
Finally, the pipeline can be summed up in Figure 4.5 below:

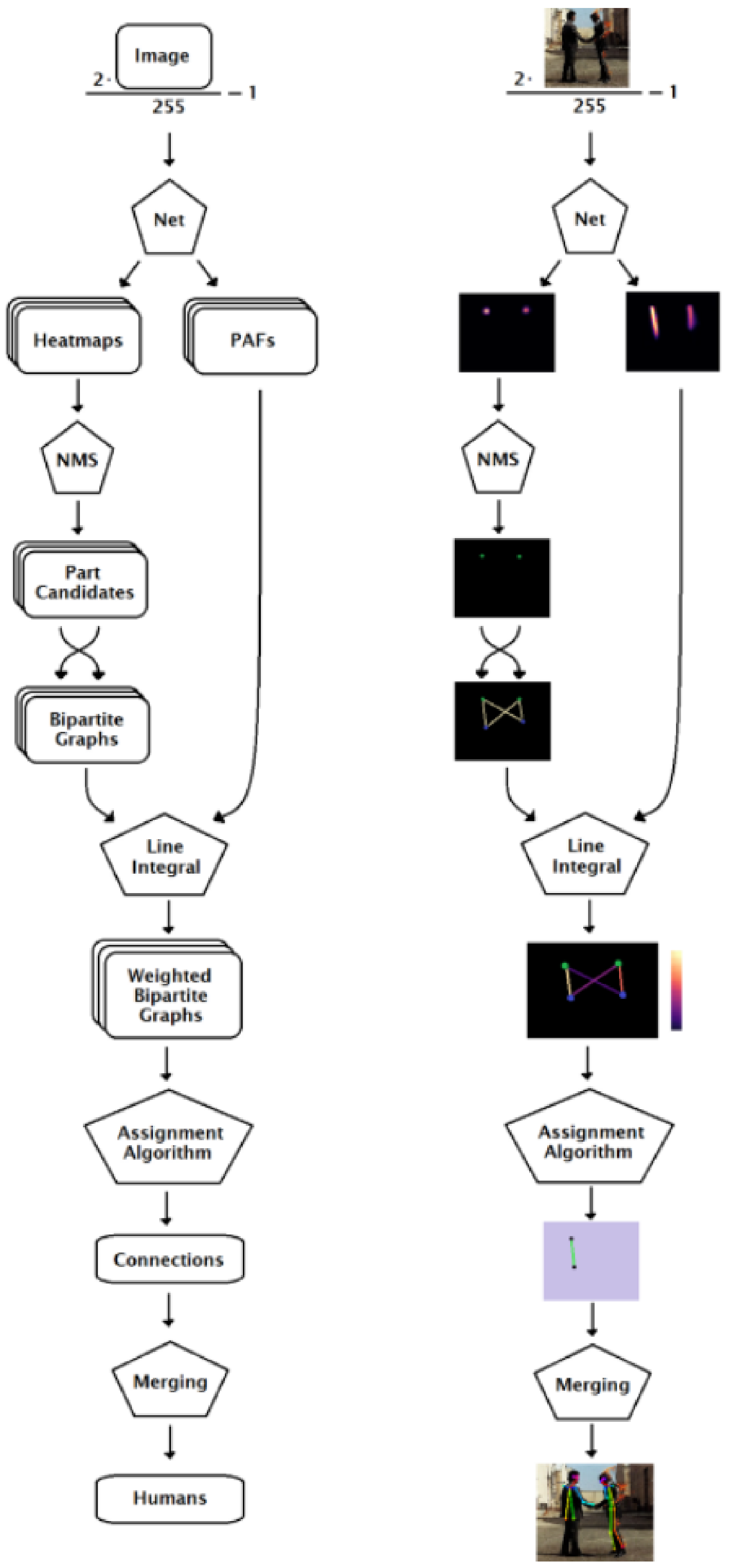
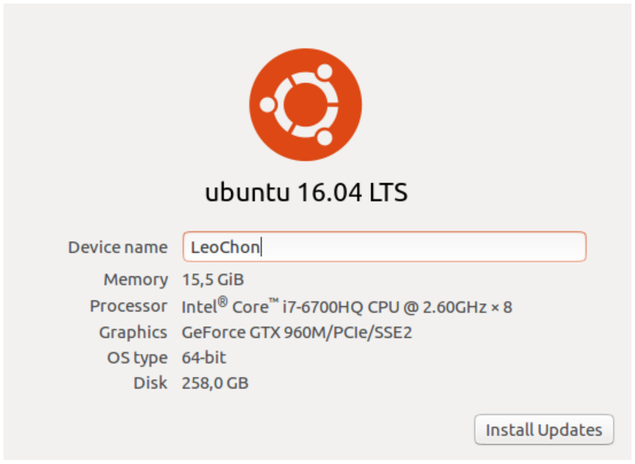


Figure 4.5: OpenPose Pipeline - repost from [81]

## Inference Time

The inference time of a model can vary depending on the hardware the processes are run on, along with the architecture and efficiency of the model itself. On the hardware part, this project was ran on Lenovo Y700 laptop, with Ubuntu 16.04 Operating System. The specifications of it can be seen in Figure 4.6, Figure 4.7 below:


ubuntu 16.04 LTS

| | |
|---|---|
| Device name | LeoChon |
| Memory | 15,5 GiB |
| Processor | Intel® Core™ i7-6700HQ CPU @ 2.60GHz × 8 |
| Graphics | GeForce GTX 960M/PCIe/SSE2 |
| OS type | 64-bit |
| Disk | 258,0 GB |

Figure 4.6: Lenovo Y700 system specifications

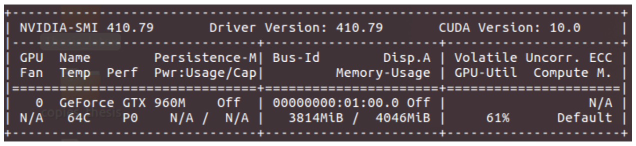

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 410.79       Driver Version: 410.79       CUDA Version: 10.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  GeForce GTX 960M    Off  | 00000000:01:00.0 Off |                  N/A |
| N/A   64C    P0    N/A /  N/A |   3814MiB /  4046MiB |     61%      Default |
+-------------------------------+----------------------+----------------------+
```

Figure 4.7: System GPU information

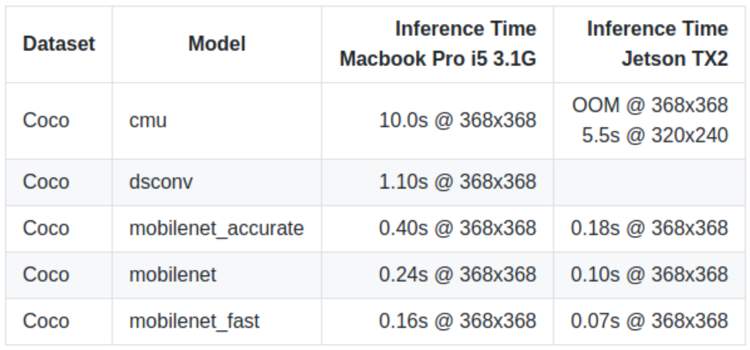
On the model part, *Tensorflow implementation* of OpenPose that is run on CPU and the original *OpenPose Python API* that uses GPU, were selected as candidate pose-estimation models.

### Tensorflow implementation of OpenPose:

It provides multiple variations of the OpenPose's VGG pretrained network models mentioned in the original paper. More specifically:

- CMU: the original paper's model, with weights converted in Caffe format to use in tensorflow
- DSCONV: a trained model using transfer learning that uses the same architecture as CMU. The author notes that speed and accuracy are unsatisfactory
- MOBILNET: these models are a product of a network architecture that on the feature-extraction part of the pipeline uses 12 convolutional layers instead of 19 in the original paper.
  - mobilnet
  - mobilnet_fast
  - mobilnet_accurat

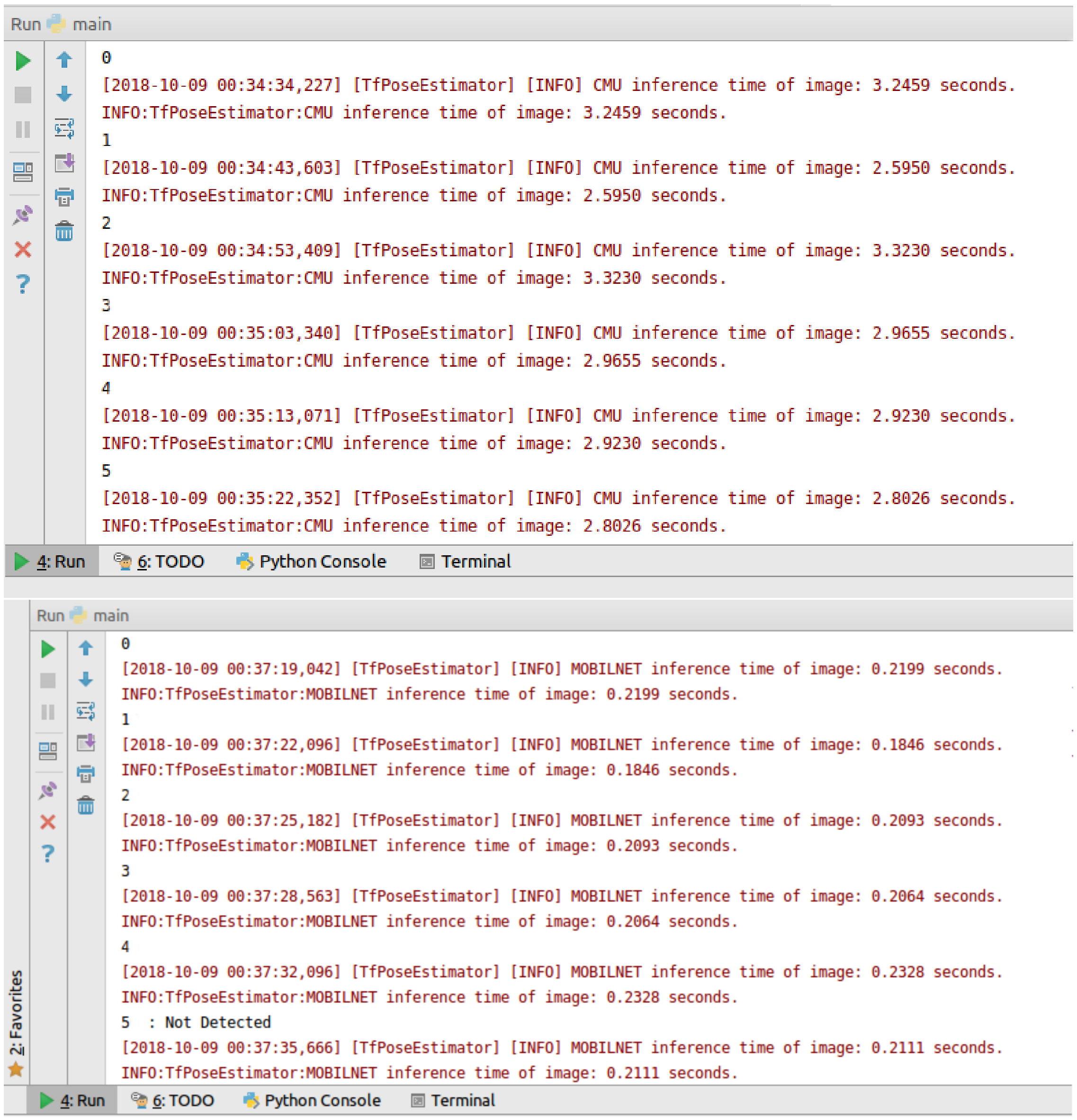
According to the github documentation, the time inference for each model is (Figure 4.8):



| Dataset | Model | Inference Time<br>Macbook Pro i5 3.1G | Inference Time<br>Jetson TX2 |
|---|---|---|---|
| Coco | cmu | 10.0s @ 368x368 | OOM @ 368x368<br>5.5s @ 320x240 |
| Coco | dsconv | 1.10s @ 368x368 | |
| Coco | mobilenet_accurate | 0.40s @ 368x368 | 0.18s @ 368x368 |
| Coco | mobilenet | 0.24s @ 368x368 | 0.10s @ 368x368 |
| Coco | mobilenet_fast | 0.16s @ 368x368 | 0.07s @ 368x368 |

Figure 4.8: Time Inference Benchmark for models in TensorFlow OpenPose version

The models run for this project was CMU and MOBILNET, with CMU's mean time inference to be 2.98 seconds per frame and MOBILNET's mean time inference 0.21 second per frame. Indicatively, the run time logger of both models can be seen in Figure 4.9



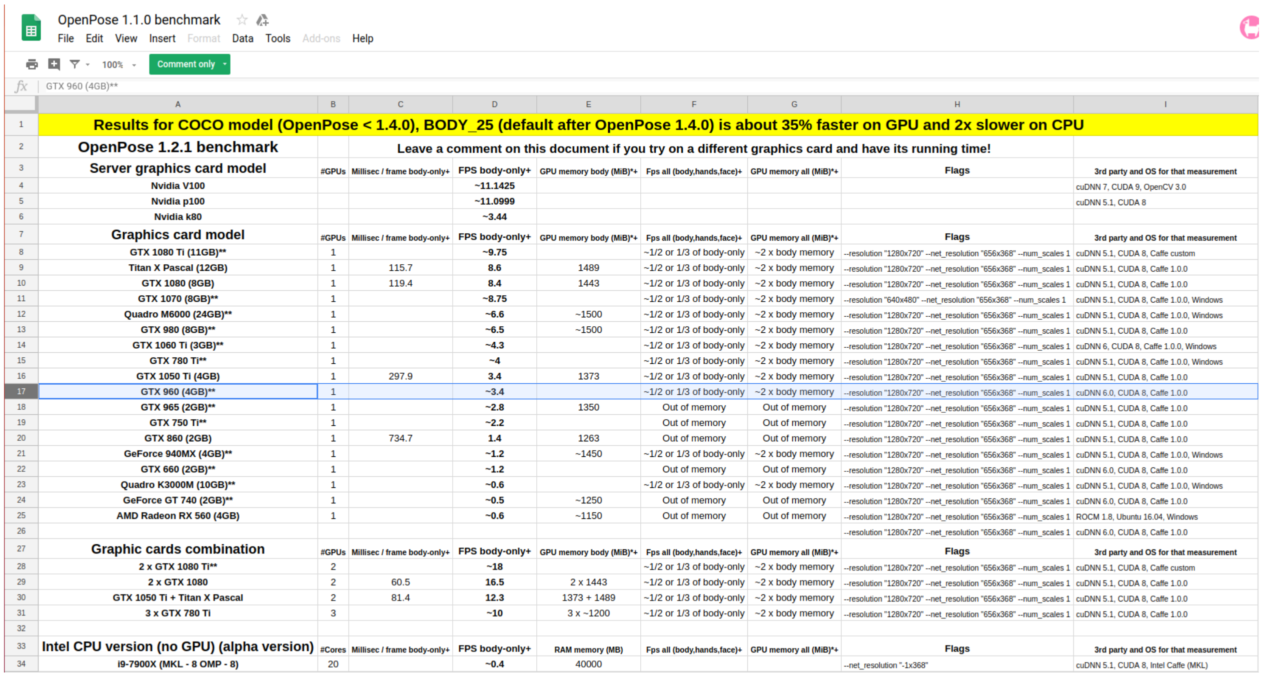
```
0
[2018-10-09 00:34:34,227] [TfPoseEstimator] [INFO] CMU inference time of image: 3.2459 seconds.
INFO:TfPoseEstimator:CMU inference time of image: 3.2459 seconds.
1
[2018-10-09 00:34:43,603] [TfPoseEstimator] [INFO] CMU inference time of image: 2.5950 seconds.
INFO:TfPoseEstimator:CMU inference time of image: 2.5950 seconds.
2
[2018-10-09 00:34:53,409] [TfPoseEstimator] [INFO] CMU inference time of image: 3.3230 seconds.
INFO:TfPoseEstimator:CMU inference time of image: 3.3230 seconds.
3
[2018-10-09 00:35:03,340] [TfPoseEstimator] [INFO] CMU inference time of image: 2.9655 seconds.
INFO:TfPoseEstimator:CMU inference time of image: 2.9655 seconds.
4
[2018-10-09 00:35:13,071] [TfPoseEstimator] [INFO] CMU inference time of image: 2.9230 seconds.
INFO:TfPoseEstimator:CMU inference time of image: 2.9230 seconds.
5
[2018-10-09 00:35:22,352] [TfPoseEstimator] [INFO] CMU inference time of image: 2.8026 seconds.
INFO:TfPoseEstimator:CMU inference time of image: 2.8026 seconds.
```

```
0
[2018-10-09 00:37:19,042] [TfPoseEstimator] [INFO] MOBILNET inference time of image: 0.2199 seconds.
INFO:TfPoseEstimator:MOBILNET inference time of image: 0.2199 seconds.
1
[2018-10-09 00:37:22,096] [TfPoseEstimator] [INFO] MOBILNET inference time of image: 0.1846 seconds.
INFO:TfPoseEstimator:MOBILNET inference time of image: 0.1846 seconds.
2
[2018-10-09 00:37:25,182] [TfPoseEstimator] [INFO] MOBILNET inference time of image: 0.2093 seconds.
INFO:TfPoseEstimator:MOBILNET inference time of image: 0.2093 seconds.
3
[2018-10-09 00:37:28,563] [TfPoseEstimator] [INFO] MOBILNET inference time of image: 0.2064 seconds.
INFO:TfPoseEstimator:MOBILNET inference time of image: 0.2064 seconds.
4
[2018-10-09 00:37:32,096] [TfPoseEstimator] [INFO] MOBILNET inference time of image: 0.2328 seconds.
INFO:TfPoseEstimator:MOBILNET inference time of image: 0.2328 seconds.
5  : Not Detected
[2018-10-09 00:37:35,666] [TfPoseEstimator] [INFO] MOBILNET inference time of image: 0.2111 seconds.
INFO:TfPoseEstimator:MOBILNET inference time of image: 0.2111 seconds.
```

Figure 4.9: Run time Logger of Inference Time for CMU (up) and MOBILNET (down)

**Original OpenPose Python API:**

It uses the architecture and pretrained model of the original paper. According to its documentation, depending on the infrastructure the network speed changes. Indicatively, more detailed information can be seen in the OpenPose Benchmark for each GPU model in Figure 4.10 [20] below:



| | A | B | C | D | E | F | G | H | I |
|---|---|---|---|---|---|---|---|---|---|
| 1 | Results for COCO model (OpenPose < 1.4.0), BODY_25 (default after OpenPose 1.4.0) is about 35% faster on GPU and 2x slower on CPU | | | | | | | | |
| 2 | OpenPose 1.2.1 benchmark | | Leave a comment on this document if you try on a different graphics card and have its running time! | | | | | | |
| 3 | Server graphics card model | #GPUs | Millisec / frame body-only+ | FPS body-only+ | GPU memory body (MiB)*+ | Fps all (body,hands,face)+ | GPU memory all (MiB)*+ | Flags | 3rd party and OS for that measurement |
| 4 | Nvidia V100 | | | ~11.1425 | | | | | cuDNN 7, CUDA 9, OpenCV 3.0 |
| 5 | Nvidia p100 | | | ~11.0999 | | | | | cuDNN 5.1, CUDA 8 |
| 6 | Nvidia k80 | | | ~3.44 | | | | | |
| 7 | Graphics card model | #GPUs | Millisec / frame body-only+ | FPS body-only+ | GPU memory body (MiB)*+ | Fps all (body,hands,face)+ | GPU memory all (MiB)*+ | Flags | 3rd party and OS for that measurement |
| 8 | GTX 1080 Ti (11GB)** | 1 | | ~9.75 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe custom |
| 9 | Titan X Pascal (12GB) | 1 | 115.7 | 8.6 | 1489 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 10 | GTX 1080 (8GB) | 1 | 119.4 | 8.4 | 1443 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 11 | GTX 1070 (8GB)** | 1 | | ~8.75 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "640x480" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0, Windows |
| 12 | Quadro M6000 (24GB)** | 1 | | ~6.6 | ~1500 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0, Windows |
| 13 | GTX 980 (8GB)** | 1 | | ~6.5 | ~1500 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 14 | GTX 1060 Ti (3GB)** | 1 | | ~4.3 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 6, CUDA 8, Caffe 1.0.0, Windows |
| 15 | GTX 780 Ti** | 1 | | ~4 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0, Windows |
| 16 | GTX 1050 Ti (4GB) | 1 | 297.9 | 3.4 | 1373 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 17 | GTX 960 (4GB)** | 1 | | ~3.4 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 6.0, CUDA 8, Caffe 1.0.0 |
| 18 | GTX 965 (2GB)** | 1 | | ~2.8 | 1350 | Out of memory | Out of memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 19 | GTX 750 Ti** | 1 | | ~2.2 | | Out of memory | Out of memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 20 | GTX 860 (2GB) | 1 | 734.7 | 1.4 | 1263 | Out of memory | Out of memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 21 | GeForce 940MX (4GB)** | 1 | | ~1.2 | ~1450 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0, Windows |
| 22 | GTX 660 (2GB)** | 1 | | ~1.2 | | Out of memory | Out of memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 6.0, CUDA 8, Caffe 1.0.0 |
| 23 | Quadro K3000M (10GB)** | 1 | | ~0.6 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0, Windows |
| 24 | GeForce GT 740 (2GB)** | 1 | | ~0.5 | ~1250 | Out of memory | Out of memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 6.0, CUDA 8, Caffe 1.0.0 |
| 25 | AMD Radeon RX 560 (4GB) | 1 | | ~0.6 | ~1150 | Out of memory | Out of memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | ROCM 1.8, Ubuntu 16.04, Windows |
| 26 | | | | | | | | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 6.0, CUDA 8, Caffe 1.0.0 |
| 27 | Graphic cards combination | #GPUs | Millisec / frame body-only+ | FPS body-only+ | GPU memory body (MiB)*+ | Fps all (body,hands,face)+ | GPU memory all (MiB)*+ | Flags | 3rd party and OS for that measurement |
| 28 | 2 x GTX 1080 Ti** | 2 | | ~18 | | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe custom |
| 29 | 2 x GTX 1080 | 2 | 60.5 | 16.5 | 2 x 1443 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 30 | GTX 1050 Ti + Titan X Pascal | 2 | 81.4 | 12.3 | 1373 + 1489 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 31 | 3 x GTX 780 Ti | 3 | | ~10 | 3 x ~1200 | ~1/2 or 1/3 of body-only | ~2 x body memory | --resolution "1280x720" --net_resolution "656x368" --num_scales 1 | cuDNN 5.1, CUDA 8, Caffe 1.0.0 |
| 32 | | | | | | | | | |
| 33 | Intel CPU version (no GPU) (alpha version) | #Cores | Millisec / frame body-only+ | FPS body-only+ | RAM memory (MB) | Fps all (body,hands,face)+ | GPU memory all (MiB)*+ | Flags | 3rd party and OS for that measurement |
| 34 | i9-7900X (MKL - 8 OMP - 8) | 20 | | ~0.4 | 40000 | | | --net_resolution "-1x368" | cuDNN 5.1, CUDA 8, Intel Caffe (MKL) |

Figure 4.10: Time Inference Benchmark for original OpenPose model per GPU

Taken into account OpenPose's GPU Benchmark, the laptop used for the present project could produce pose estimates for almost 3.4 frames per second.

# Chapter 5

# Dataset

## Datasets for Video Classification

The problem of video classification is studied with the help of video data. The datasets have different characteristics, depending on the problem they were created to solve. More specifically, video datasets can vary in size, quality, degree of constrain and label type [75]. Consequently, the videos can be rather short or long in length, have different resolutions, frames per second and be temporally segmented depending on the action performed or left unconstrained with no processing("in the wild" [72] [82]). The last, has the advantage of testing models on realistic, user-produced videos publicly available. However, it comes at a cost of added complexity created by large variations in illumination, scale, camera motion, viewpoints etc. Among the publicly available "in the wild" video datasets, that are large enough for deep learning architectures and don't require techniques such as transfer learning, are:

- **UCF101 - Action Recognition Data Set** [82]:
  UCF stands for University of Central Florida and is a collection of 13.320 Youtube videos from 101 categories, grouped in 25 groups, where each group consists of 4-7 videos of an action. The categories are divided into 5 types:
  - Human-Object Interaction
  - Body-Motion Only
  - Human-Human Interaction
  - Playing Musical Instruments
  - Sports

  As shown in Figure 5.1 [82], the videos' length vary. The dataset consists of unprocessed Youtube Videos. The only processed element is the labels, since the video content is known.

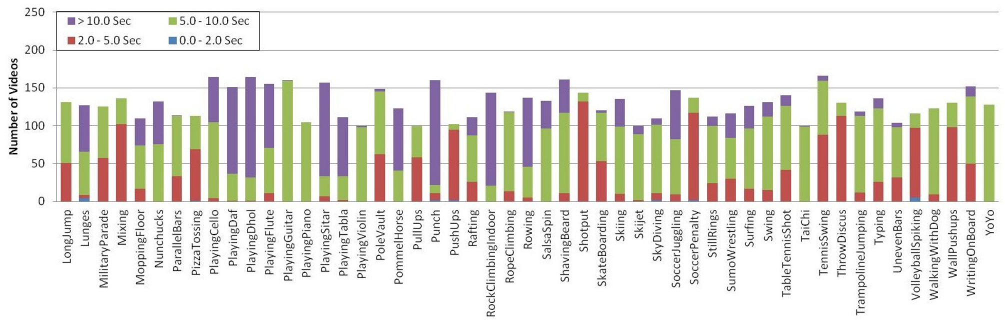


Figure 5.1: UCF101 dataset videos' duration distribution - repost from [82]

- **UCF50 - Action Recognition Data Set** [82]:
  This is the same as UCF101 consisting of 50 categories instead of 101. The categories are grouped into 25 groups where each group consists of 4 action clips.

- **YouTube-8M**: [3]
  The dataset was created by Video Understanding group of Google AI Perception and consists of 6.1 million video Youtube URLs and 3.862 class labels[1]. It is machine-generated by YouTube video annotation system in addition to human-based filtering on metadata and query click signals to ensure the quality of label-video matching [43]. The dataset is accompanied with its pretrained model.

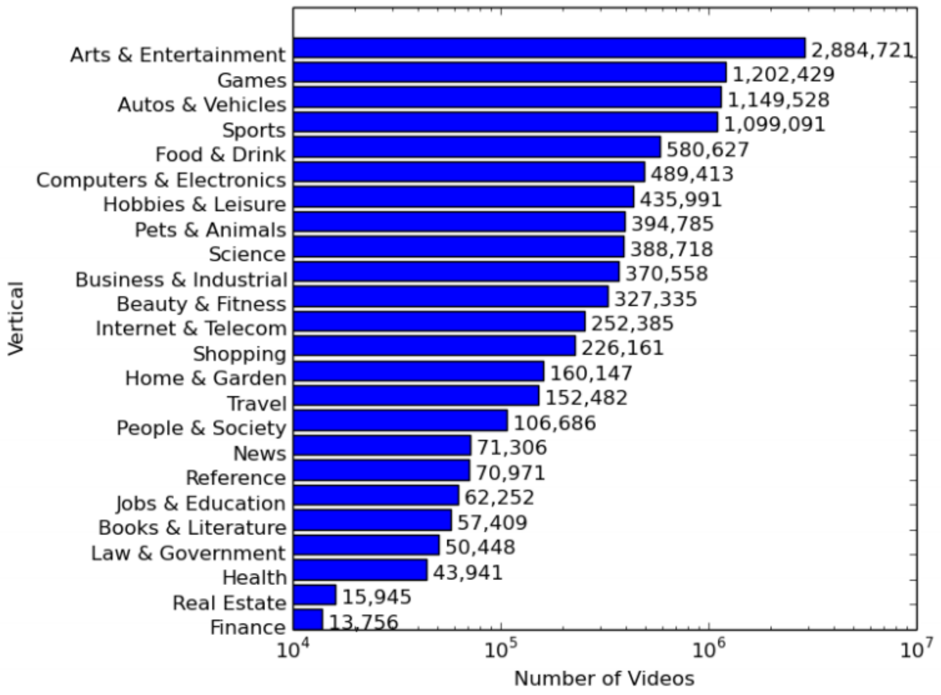


Figure 5.2: YouTube-8M Dataset histogram showing the number of videos per category. Each video may be annotated with more than one class - repost from [3]

- **Sports-1M Dataset** [43]:
  This is a subset of the Youtube-8M dataset (see above), focusing on sports. It consists of 1.113.158 videos with 487 sport label classes. The dataset is accompanied with its pretrained model. Since the dataset is focused in sports is a good candidate for pose estimation analysis.

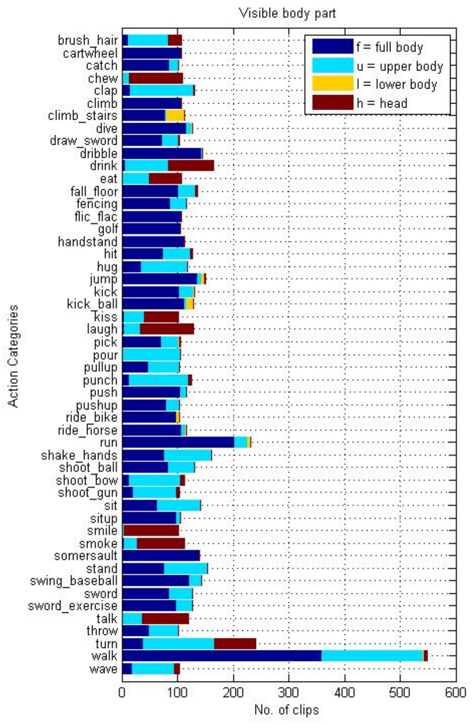
- **HMDB51** [47]:
  HMDB stands for Human Motion database. It was created by Brown University Research Group, consisting of 6.849 videos divided into 51 action categories, each containing a minimum of 101 clips. It was collected by movies and public databases such as Prelinger archive, Youtube and Google videos. The human motions included concern action that incporporate both face and body. As shown in Figure 5.3 the body parts included in the video are identified, making it useful to select the appropriate data for human pose estimation analysis.

- **Hollywood2** [54]:
  The dataset was created by IRISA (Institut de recherche en informatique et systèmes aléatoires) the joint computer science research center of CNRS, University of Rennes 1, ENS Rennes, INSA Rennes and Inria located in Brittany, France. It was composed of video clips from 69 movies and consists of 12 human actions' classes and 10 classes of scenes, distributed over 3.669 videos [2]. See below in Figure 5.4 for more details on the actions and scene information.

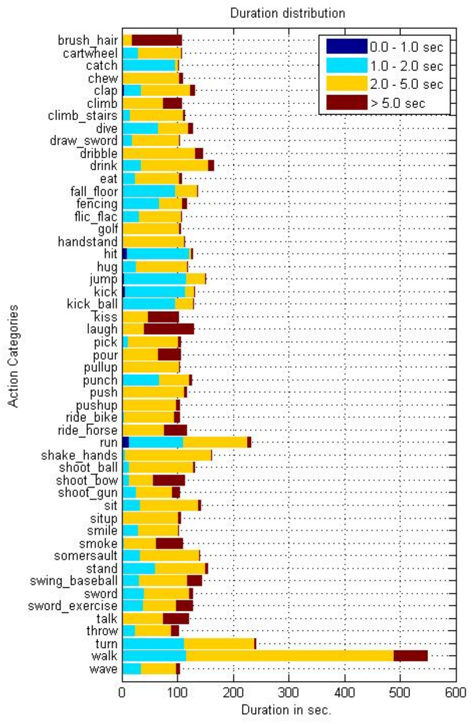
For cases where dataset size is not important, there are more available video-datasets. Indicatively:

[1]The amount of data has changed from the release of the dataset's official paper. Data taken from updated dataset site: https://research.google.com/youtube8m/

[2]Data taken from the official and updated dataset site: http://www.di.ens.fr/ laptev/actions/hollywood2/

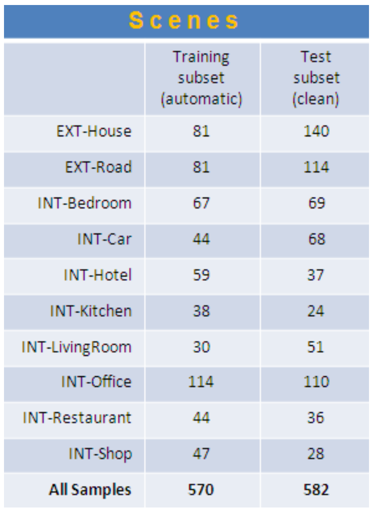


(a) Distribution of body part per category

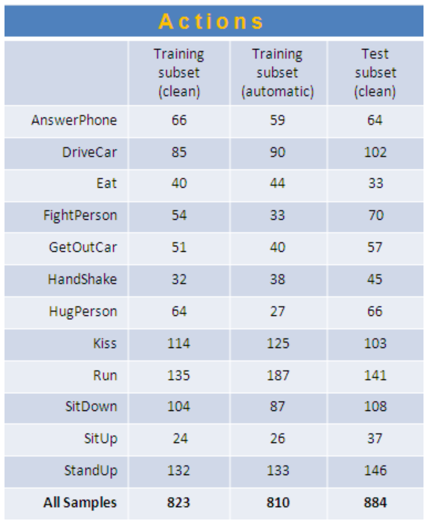


(b) Duration Distribution per time threshold

Figure 5.3: Indicative statistics of HMDB51 database - repost from [47]



| Scenes | | |
|---|---|---|
| | Training subset (automatic) | Test subset (clean) |
| EXT-House | 81 | 140 |
| EXT-Road | 81 | 114 |
| INT-Bedroom | 67 | 69 |
| INT-Car | 44 | 68 |
| INT-Hotel | 59 | 37 |
| INT-Kitchen | 38 | 24 |
| INT-LivingRoom | 30 | 51 |
| INT-Office | 114 | 110 |
| INT-Restaurant | 44 | 36 |
| INT-Shop | 47 | 28 |
| **All Samples** | **570** | **582** |



| Actions | | | |
|---|---|---|---|
| | Training subset (clean) | Training subset (automatic) | Test subset (clean) |
| AnswerPhone | 66 | 59 | 64 |
| DriveCar | 85 | 90 | 102 |
| Eat | 40 | 44 | 33 |
| FightPerson | 54 | 33 | 70 |
| GetOutCar | 51 | 40 | 57 |
| HandShake | 32 | 38 | 45 |
| HugPerson | 64 | 27 | 66 |
| Kiss | 114 | 125 | 103 |
| Run | 135 | 187 | 141 |
| SitDown | 104 | 87 | 108 |
| SitUp | 24 | 26 | 37 |
| StandUp | 132 | 133 | 146 |
| **All Samples** | **823** | **810** | **884** |

Figure 5.4: Action and Scene categories of the Hollywood2 dataset - repost from [54]

- **KTH Dataset** [74]:
  It was created by KTH Royal Institute of Technology, containing 600 artificial videos (data was staged by actors and is not realistic). It consists of 6 types of actions and 100 clips per category.
- **Weizmann Institute Dataset** [11]:
  It was created by Weizmann Institute and consists of 10 action categories and 9 clips each.

## Dataset Selection

The current project aims to use anatomical points of people in frames as features to a sequential deep learning architecture, such as this of LSTM (Long short-term memory). Consquently, the desired video dataset should contain full human bodies, performing different actions. The pose-estimation module includes additional models for head and upper body pose estimation. Consequently, a dataset where the videos' human body parts are annotated and known(like HMDB51 dataset), could be taken into account so that the appropriate model is used for the pose estimation process. However, since in realistic unconstrained videos the human body parts included in the video are not known in advance, this idea was abandoned.
In place of this, an unconstrained dataset with category actions that theoretically utilize full human body was preferred, aiming to result in a realistic classification model. For this reason, the Sports-1M Dataset was chosen.

The Sports-1M Dataset is a subset of the Youtube-8M dataset focusing on sports. It consists of 1.113.158 videos with 487 sport label classes. The positive aspects of this dataset are:

- **Size**:
  Each sport's category has more than 1000 videos each.
- **Topic**:
  The dataset's topic is sports, where sports incorporate, most often than not full body utilization, making it suitable for human pose estimation.

The challenges of this dataset are:

- **"In the wild" obstacles**:
  The unconstrained videos add extra complexity due to large variations in illumination, scale, camera motion, viewpoints etc.
- **Machine-generated dataset**:
  Since the dataset is machine-generated by YouTube video annotation system, there is a high margin of error during the selection of the videos' urls. It is possible that a lot of videos include in their titles sports not because they are executing them, but rather talk about them. Moreover, with a random selection of videos to preview, a lot were found to be unrelated to the sport assigned to, in other cases the camera motion was so high that no clear image could be formed whatsoever and in worst cases the video was not available to preview. For this reason, it is difficult to determine the percentage the dataset's error is contributing to the overall model performance's error.

### Exploratory Analysis of Dataset

Sports-1M Dataset contains more than 1 million videos with 487 sport label classes. For the purposes of this project, it was decided that the use of all the sports' labels would not bring any additional value for the extra processing time, computational power and memory usage that would require. As a result, 5 labels were selected, that fulfilled the criteria:

1. **the sport category must include more than 1000 video clips**:
   The higher the videos assigned to the label, the higher the probability of actually acquiring 1000 videos as an end result. This is important for future data loss from unavailable videos listed in the dataset, and from post processing.
2. **the sport must require full of half-full body to be executed**:
   This is imperative since the purpose of the project is to analyse the poses of people in frames as they progress in time.

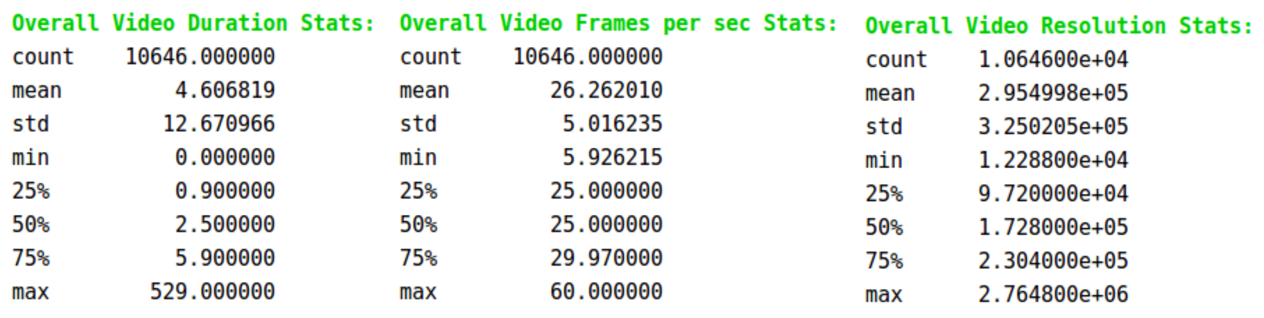
The sports labels' selected are:

- Bowling
- Olympic Weightlifting
- Squash
- Table Tennis
- Wing Chun

The video urls were downloaded and both statistical and graphical exploratory data analysis was applied, to create a better understanding of the characteristics of the dataset. The indicators measured were **duration**, **frames per second (fps)** and **resolution**.

### Within Differences

The within differences examine the vertical differences of the overall dataset as a unit and not in-between its categories. To investigate the within differences both quantitative and graphical representations of the data were analysed.



```
Overall Video Duration Stats:
count    10646.000000
mean         4.606819
std         12.670966
min          0.000000
25%          0.900000
50%          2.500000
75%          5.900000
max        529.000000
```

```
Overall Video Frames per sec Stats:
count    10646.000000
mean        26.262010
std          5.016235
min          5.926215
25%         25.000000
50%         25.000000
75%         29.970000
max         60.000000
```

```
Overall Video Resolution Stats:
count    1.064600e+04
mean     2.954998e+05
std      3.250205e+05
min      1.228800e+04
25%      9.720000e+04
50%      1.728000e+05
75%      2.304000e+05
max      2.764800e+06
```

Figure 5.5: Quantitative within-differences of subset selected from Sports-1M dataset

According to Figure 5.5 and Figure 5.6, we can observe:

- 10.646 videos were downloaded from the 12.979 urls initially selected
- As seen in Cumulative distribution in Figure 5.6, close to 90% of videos' duration is around 5 minutes
- Videos' duration:
  - the average video in all categories is approximately 5 minutes long
  - the range of videos' duration is between 0 seconds to 8.8 hours long (529 minutes)
  - 25% of videos' duration is almost 1 minute
  - 50% of videos' duration is 2.5 minutes
  - 75% of videos' duration is 5.9 minutes
  - most videos' duration (68% of data) appear unintuitively to be 12.67 minutes longer or shorter than the mean duration (1 STD) biased by the dispersion of data. The dispersion of dataset's duration expressed in Coefficient of Variation (CV) is approximately 2.7 units (std/mean)
- Videos' Frames per Second(fps):
  - the average fps are 27
  - the range is between 6 to 60 fps
  - 25%, 50% and 75% of videos have from 25fps to 30fps
  - most videos' fps (68% of data) are within range of 21 to 31 fps (5 fps 1 STD). Its Coefficient of Variation is 0.2 units

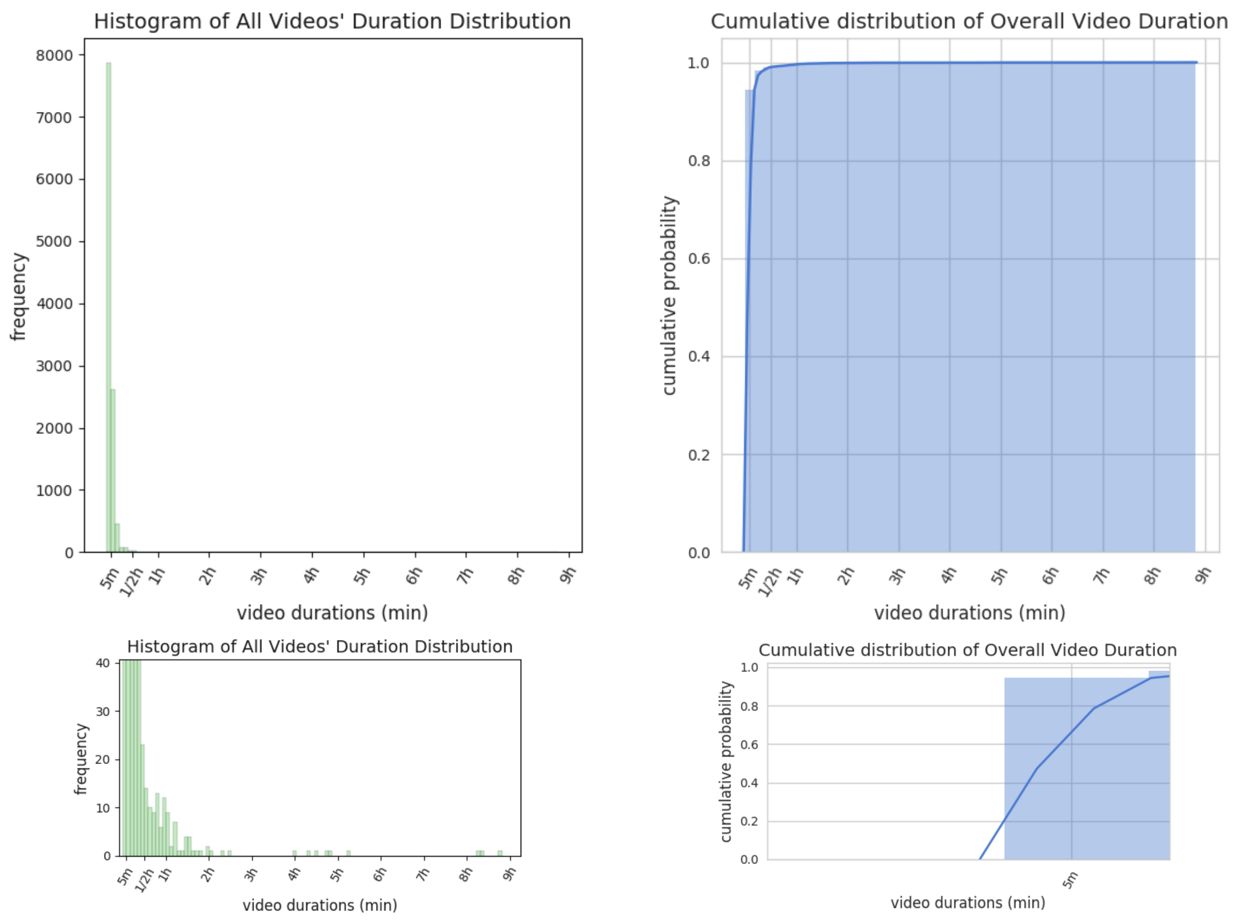


Figure 5.6: Graphical within-differences of subset selected from Sports-1M dataset. Normal and Zoomed In Histogram and Cumulative Distribution

- Videos' Resolution:
  - the average resolution is 295.499 overall pixels, aka a little less than 480p (720 x 480)
  - the range is between 12.288 (aka a bit more than 120 x 90) and 2.764.800 (1920 x 1440)
  - 25% of videos' resolution 97.200 pixels (a little less than 240p: 426 $\times$ 240)
  - 50% of videos' resolution 172.800 pixels (close to 360p: 380 x 360)
  - 75% of videos' resolution is 230.400 pixels (aka 480p)
  - most videos' resolution (68% of data) are going as high as 620.518 pixels (close to 720p: 1280 x 720) (1 STD)

The big picture of dataset's within-differences is normal, since the mean and 50%, 75% percentiles for all indicators (duration, fps, resolution) don't vary substantially. A few outliers are noticed (min and max values are very different to the mean and percentiles), but they are not enough in number to distort the overall dataset's statistics.

### In-Between Differences

The in-between differences refer to horizontal differences that seeks to find the differences between its distinct categories. To investigate that both quantitative and graphical exploratory analysis was conducted.

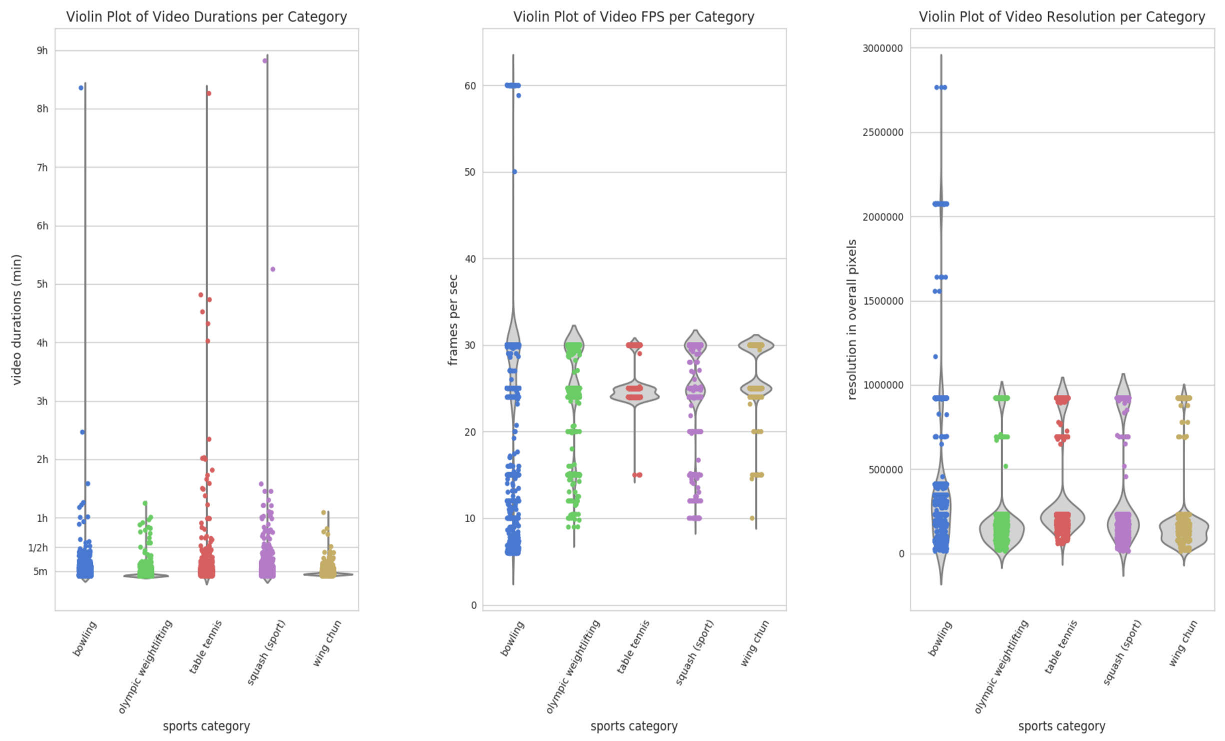


Figure 5.7: Graphical in-between-differences of dataset's sports categories - Violin Plot

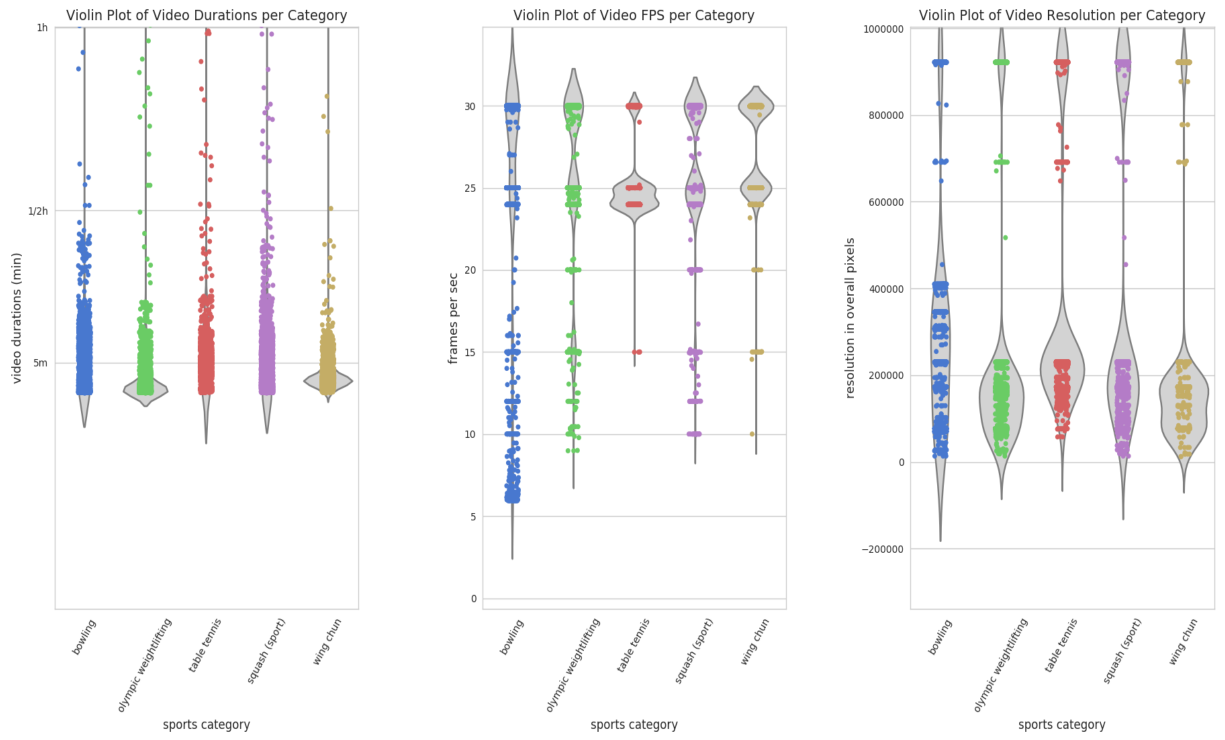


Figure 5.8: Zoomed In: Graphical in-between-differences of dataset's sports categories - Violin Plot

As seen in Figure 5.7, *duration* appears to have around 9 major outliers in *bowling*, *squash* and *table tennis* categories. Similarly, *frames per second* have a small amount of outliers concentrated around *bowling* category. Lastly, *resolution's* major outliers appear mainly in *bowling* category with the rest sharing the same outliers among all categories, creating uniformity.

Additionally, by zooming into violin plots in Figure 5.8, we can observe that the shapes of mainly *squash* - *table tennis*,*bowling* and *olympic weightlifting* - *wing chun* seem to be alike, indicating similarities in dispersion of videos' duration. In the same way, *olympic weightlifting* - *squash*, *bowling* and *table tennis* - *wing chun* appear to share similar distributions of their videos' frames per second. Finally, *olympic weightlifting* - *table tennis* - *squash* -*wing chun* show similarities on their videos' resolution distribution.

As verified in Figure 5.9 the overall differences don't appear to be extreme, but rather mild.

Moving on to quantitative analysis in Figure 5.10 we can make the following observations:

- the number of videos per category is close to 2000 with *table tennis* being the smallest in number (2046 videos)
- *duration* per category varies, with 68% of videos be within the range of 2 to 7 minutes. *Table Tennis* is the category with the longest videos in average(mean duration almost 7 minutes) and *olympic weightlifting* the category with the shortest videos in average ( mean duration 2.1 mins)
- we confirm that *squash*, *table tennis* and *bowling* are the categories with the major outliers (8.8h, 8.25h and 8.35 long videos accordingly), which affects their *means* and *stds*. These are also the most variant with Coefficient of Variance of 3.6, 3, 2.3 units, compared to 1.2, 2, 1 of *bowling*, *olympic weightlifting* and *wing chun* accordingly
- *frames per second (fps)* shows small differentiation with mean values be within the range of 25 to 27 fps. Almost all categories have max 30fps with the exception of *bowling* (max: 60)
- *resolution* varies too, with average videos of categories exist within the range of less than 360p to more than 480p

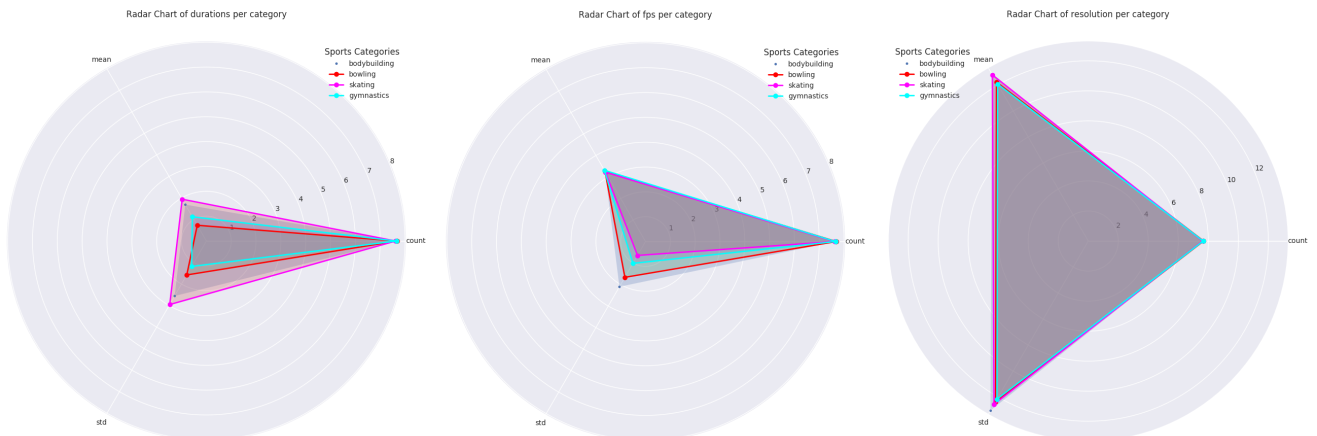
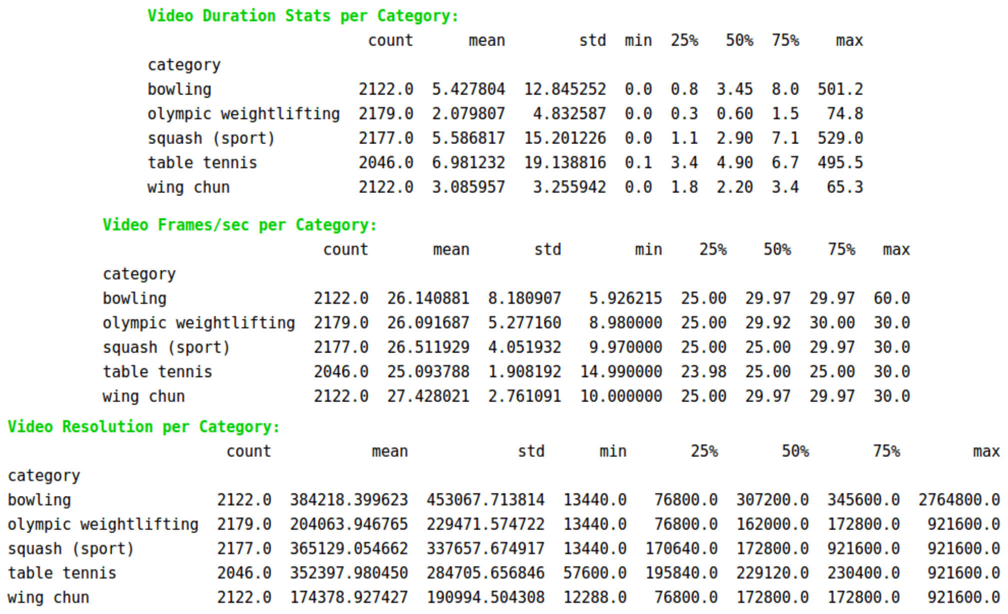


Figure 5.9: Radar Chart of duration, fps and resolution per category



```
Video Duration Stats per Category:
                       count      mean        std  min  25%   50%  75%    max
category
bowling               2122.0  5.427804  12.845252  0.0  0.8  3.45  8.0  501.2
olympic weightlifting 2179.0  2.079807   4.832587  0.0  0.3  0.60  1.5   74.8
squash (sport)        2177.0  5.586817  15.201226  0.0  1.1  2.90  7.1  529.0
table tennis          2046.0  6.981232  19.138816  0.1  3.4  4.90  6.7  495.5
wing chun             2122.0  3.085957   3.255942  0.0  1.8  2.20  3.4   65.3
```

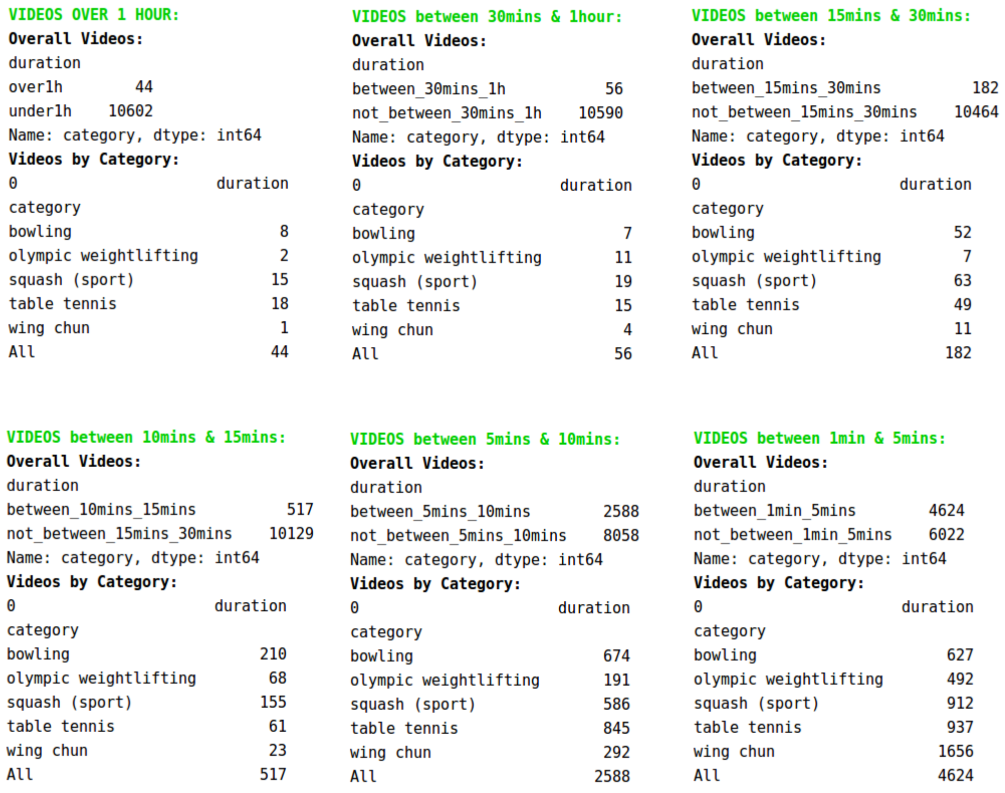
```
Video Frames/sec per Category:
                       count       mean       std        min    25%    50%    75%   max
category
bowling               2122.0  26.140881  8.180907   5.926215  25.00  29.97  29.97  60.0
olympic weightlifting 2179.0  26.091687  5.277160   8.980000  25.00  29.92  30.00  30.0
squash (sport)        2177.0  26.511929  4.051932   9.970000  25.00  25.00  29.97  30.0
table tennis          2046.0  25.093788  1.908192  14.990000  23.98  25.00  25.00  30.0
wing chun             2122.0  27.428021  2.761091  10.000000  25.00  29.97  29.97  30.0
```

```
Video Resolution per Category:
                       count           mean            std      min       25%       50%       75%        max
category
bowling               2122.0  384218.399623  453067.713814  13440.0   76800.0  307200.0  345600.0  2764800.0
olympic weightlifting 2179.0  204063.946765  229471.574722  13440.0   76800.0  162000.0  172800.0   921600.0
squash (sport)        2177.0  365129.054662  337657.674917  13440.0  170640.0  172800.0  921600.0   921600.0
table tennis          2046.0  352397.980450  284705.656846  57600.0  195840.0  229120.0  230400.0   921600.0
wing chun             2122.0  174378.927427  190994.504308  12288.0   76800.0  172800.0  172800.0   921600.0
```

Figure 5.10: Quantitative in-between-differences of dataset's categories

Lastly, the exact amount of *duration* outliers was investigated, by counting the number of videos per category that fell into the different duration ranges defined (longer than 1h, 1h-30mins, 30mins-15mins, 15mins-10mins, 10mins-5 mins, 5 mins-1min).

As seen in Figure 5.11 there are only 44 videos over 1 hour and 56 between 30minutes and 1 hour. Most videos appear between 0 to 10 minutes (7.729 videos : adding ranges of 0-1min, 1-5mins and 5-10minutes). Once again we confirm that the biggest outliers appear in categories *squash*, *table tennis* and *bowling*.



```
VIDEOS OVER 1 HOUR:
Overall Videos:
duration
over1h       44
under1h   10602
Name: category, dtype: int64
Videos by Category:
0                        duration
category
bowling                         8
olympic weightlifting           2
squash (sport)                 15
table tennis                   18
wing chun                       1
All                            44
```

```
VIDEOS between 30mins & 1hour:
Overall Videos:
duration
between_30mins_1h           56
not_between_30mins_1h    10590
Name: category, dtype: int64
Videos by Category:
0                        duration
category
bowling                         7
olympic weightlifting          11
squash (sport)                 19
table tennis                   15
wing chun                       4
All                            56
```

```
VIDEOS between 15mins & 30mins:
Overall Videos:
duration
between_15mins_30mins          182
not_between_15mins_30mins    10464
Name: category, dtype: int64
Videos by Category:
0                        duration
category
bowling                        52
olympic weightlifting           7
squash (sport)                 63
table tennis                   49
wing chun                      11
All                           182
```

```
VIDEOS between 10mins & 15mins:
Overall Videos:
duration
between_10mins_15mins          517
not_between_15mins_30mins    10129
Name: category, dtype: int64
Videos by Category:
0                        duration
category
bowling                       210
olympic weightlifting          68
squash (sport)                155
table tennis                   61
wing chun                      23
All                           517
```

```
VIDEOS between 5mins & 10mins:
Overall Videos:
duration
between_5mins_10mins        2588
not_between_5mins_10mins    8058
Name: category, dtype: int64
Videos by Category:
0                        duration
category
bowling                       674
olympic weightlifting         191
squash (sport)                586
table tennis                  845
wing chun                     292
All                          2588
```

```
VIDEOS between 1min & 5mins:
Overall Videos:
duration
between_1min_5mins        4624
not_between_1min_5mins    6022
Name: category, dtype: int64
Videos by Category:
0                        duration
category
bowling                       627
olympic weightlifting         492
squash (sport)                912
table tennis                  937
wing chun                    1656
All                          4624
```

Figure 5.11: Videos found per category per duration ranges

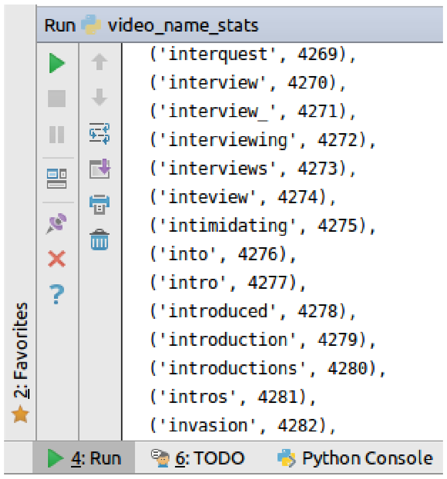
**Conlusion**

In conclusion, the selected subset of Sports-1M Dataset does not include videos of fixed *duration* and *resolution* but appears to have quite similar *fps*. On the *duration* variance, is important to decide how to fix *duration* at a certain time window, so as to extract the same amount of frames from all videos and use this to define the shape of the LSTM network's input. On the *resolution* variance, is necessary to understand how is affecting the ability of candidate models to pose estimate accurately, in order to discard videos that produce low quality pose results. All these findings will unravel the constraints of the system to be built. Finally, due to a lot of inherent errors residing in the dataset (since it is a machine-generated dataset and not a manually selected one), some extra basic natural language preprocessing is required, to spot possible non-sport videos.

## Dataset Cleaning

As mentioned previously in the chapter, the dataset is highly likely to include a lot of videos that are not related directly to the label attached to. This is because it is a machine generated dataset and was not manually selected by humans. For this reason, in order to minimize the error as much as possible, a very basic Natural Language Processing (NLP) technique was utilized.

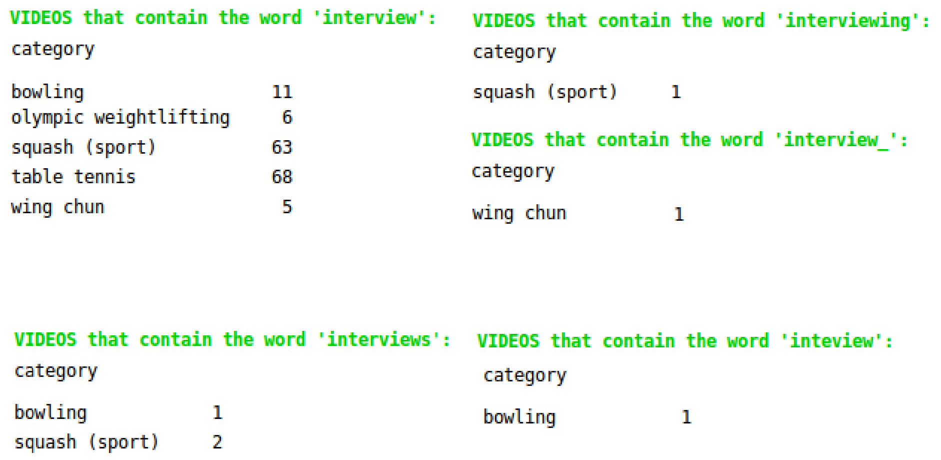
More specifically, all the video titles were represented as a *bag of words*. The *bag of words*(BoW) creates a vocabulary of unique tokens [69] of a given text. The goal was to create a vocabulary of unique words that we would be able to manually examine. The produced *bag of words* was a 8.582 long list of unique words, a fragment of which can be seen in Figure 5.12.



```
('interquest', 4269),
('interview', 4270),
('interview_', 4271),
('interviewing', 4272),
('interviews', 4273),
('inteview', 4274),
('intimidating', 4275),
('into', 4276),
('intro', 4277),
('introduced', 4278),
('introduction', 4279),
('introductions', 4280),
('intros', 4281),
('invasion', 4282),
```

Figure 5.12: Bag of Words representation of Videos' title

From the list of unique words found in the dataset, there was a word and its variations, that could imply that the content of those videos are not likely to actually perform a sport, but rather talk about it. The words are: *interview, interviewing, interviews, inteview, interview_* .



```
VIDEOS that contain the word 'interview':
category

bowling                 11
olympic weightlifting    6
squash (sport)          63
table tennis            68
wing chun                5
```

```
VIDEOS that contain the word 'interviewing':
category

squash (sport)     1
```

```
VIDEOS that contain the word 'interview_':
category

wing chun          1
```

```
VIDEOS that contain the word 'interviews':
category

bowling          1
squash (sport)   2
```

```
VIDEOS that contain the word 'inteview':
category

bowling          1
```

Figure 5.13: Number of videos per category found to have the word 'interview', 'interviewing', 'interviews', 'inteview' and 'interview_'

As seen in Figure 5.13, 139 videos had the word *interview*, 2 *interviewing*, 3 *interviews*,1 *inteview* and 1 *interview_*. Those videos will not be included in the training process.

# Chapter 6

# System Constraints

In subsection 5.2.1, exploratory analysis was applied on the dataset to understand its main characteristics. The dataset was examined through the prism of three (3) variables: *duration, fps and resolution*, which they were found to vary substantially from video to video. The dataset is to be used to extract human poses from each frame of the video and then use that collection as features to an LSTM model, in order to recognize the sport label the video belongs to. To achieve high accuracy, is important to understand how much video duration is ideal to obtain good classification results and how much resolution is necessary to produce quality pose-estimates. In this section is evaluated how the dataset's variability in duration and resolution affects the overall system and set its constraints.

## Resolution

The dataset's videos were found to have varied resolutions, ranging from less than 120x90p up to 1920x1440p. Those videos are used to extract the peoples poses from each frame using a pose-estimation model of choice and is important that the model functions properly irrespective of the video's resolution. In this section, is examined how the predictions of each pose-estimation model are affected depending on the resolution. The candidate models (analysed in chapter 4) are: *mobilenet-thin*, *tensorflow cmu* and *cmu Python Wrapper*. Indicatively three (3) videos were selected from the dataset, of different difficulty. The level of difficulty is defined by how many people and objects are in the video. More specifically, the levels are:

- **High Difficulty:** video multiple people
- **Medium Difficulty:** video with few people and objects interfering
- **Low Difficulty:** video with one person

Each video selected had a resolution as high as 1920x1080 and downsized to four (4) different resolutions. The resolutions available for each video are:

- 256 x 144 p (144p)
- 352 x 240 p (240p - SD)
- 640 x 480 p (480p)
- 1280 x 720 p (720p) (Half HD)
- 1920 x 1080 p (1080p) (Full HD)

In Figure 6.1, Figure 6.2 and Figure 6.3 are presented the model comparisons per resolution and different level of frame difficulty.

In High-Difficulty Frame (Figure 6.1), there are 10, small in size people to be detected. *Mobilnet model* appears to be the weakest in detecting people, not making any detections until 480p (1 of 10 people) and with not satisfying results even in 1080p (3 of 10 people). Better results seem to show the *cmu model implemented using tensorflow* with some mild detection in 240p (1/2 of 10 people), with more sufficient detections after 480p (2 of 10 people) and not great differences from that point after. Finally, the best results are granted to *cmu model of the original Caffe implementation with Python Wrapper*, that detects people even in 140p (2 of 10 people), with more detections than any other model at 240p (4 of 10 people) and increasing ever after making almost no difference after 720p (9 of 10 people).

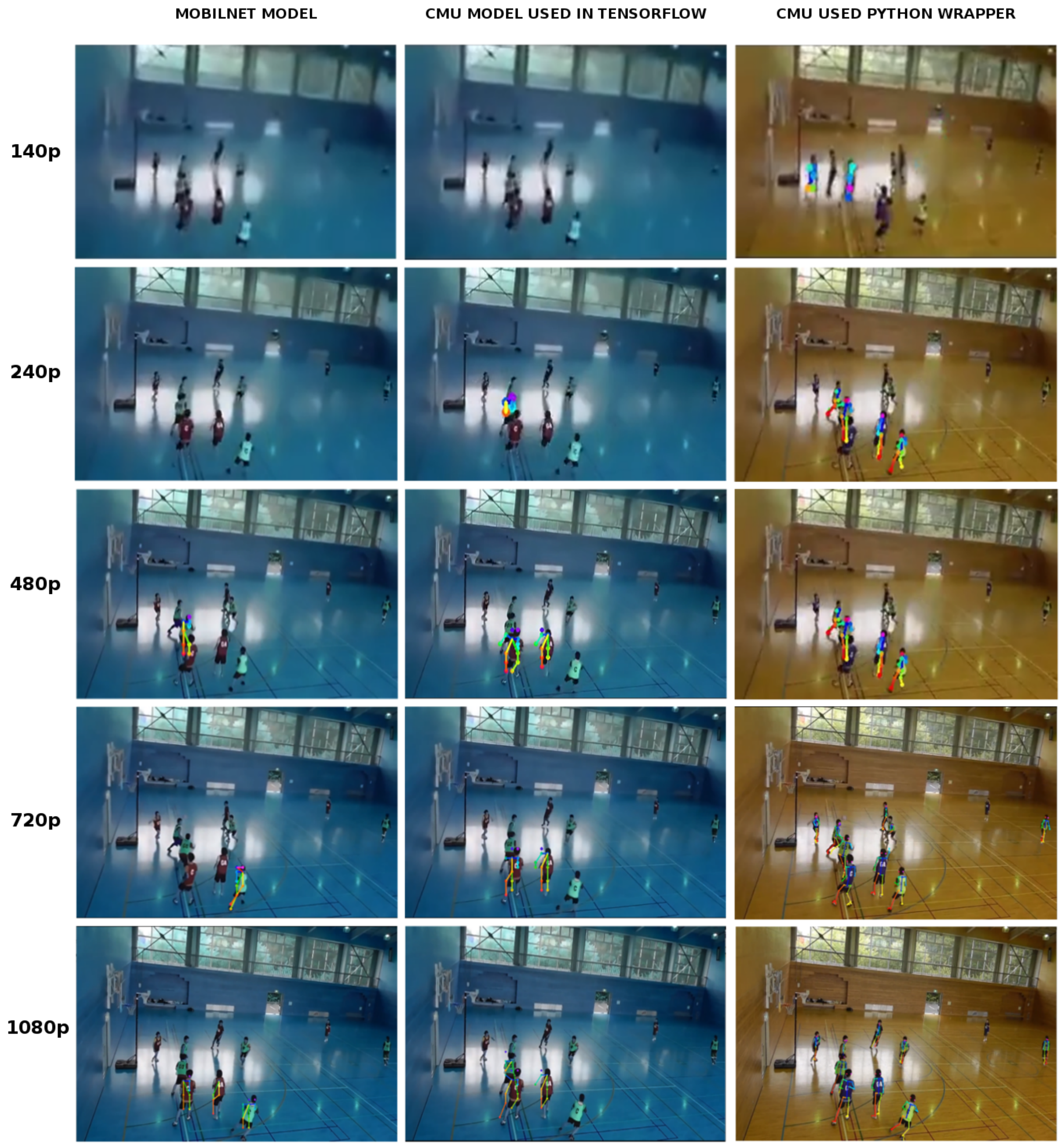


Figure 6.1: Model Comparison per Resolution on frame with multiple People - High Difficulty

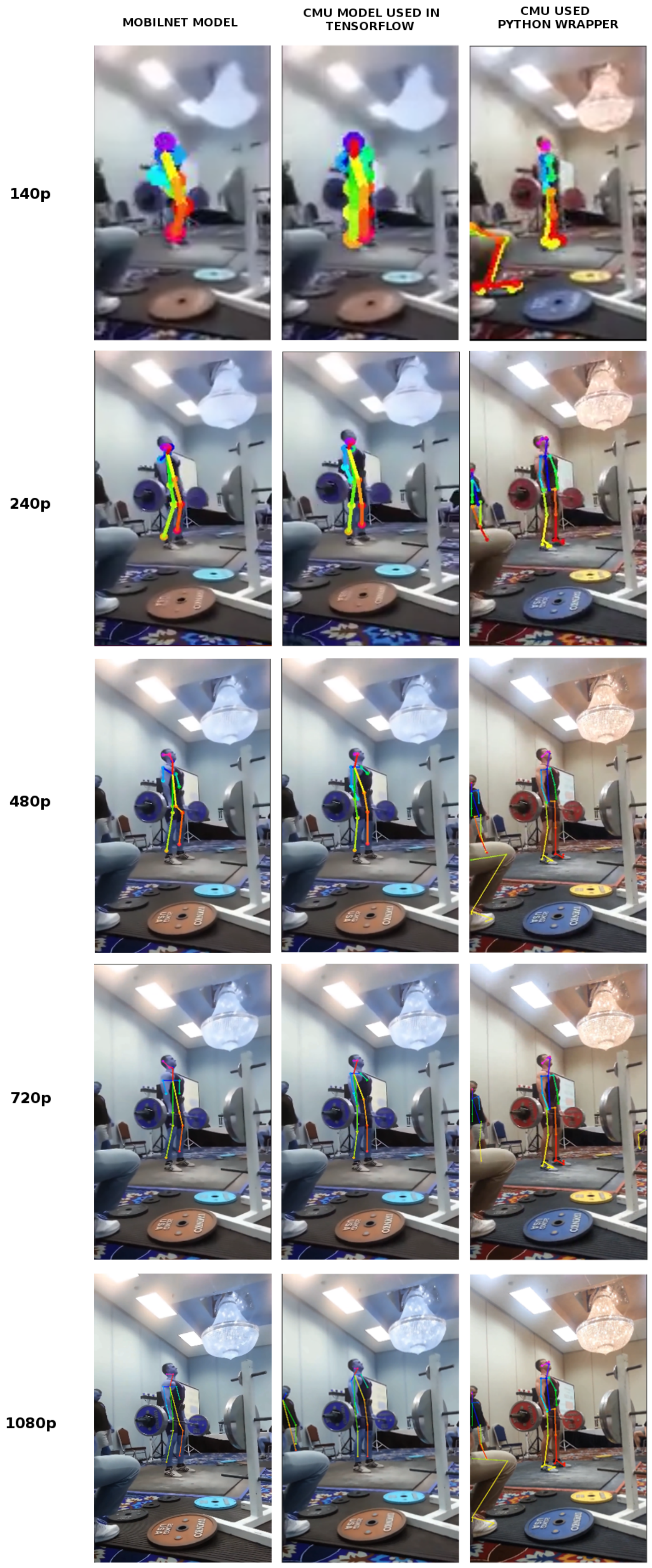


Figure 6.2: Model Comparison per Resolution on frame with few people and objects interfering - Medium Difficulty

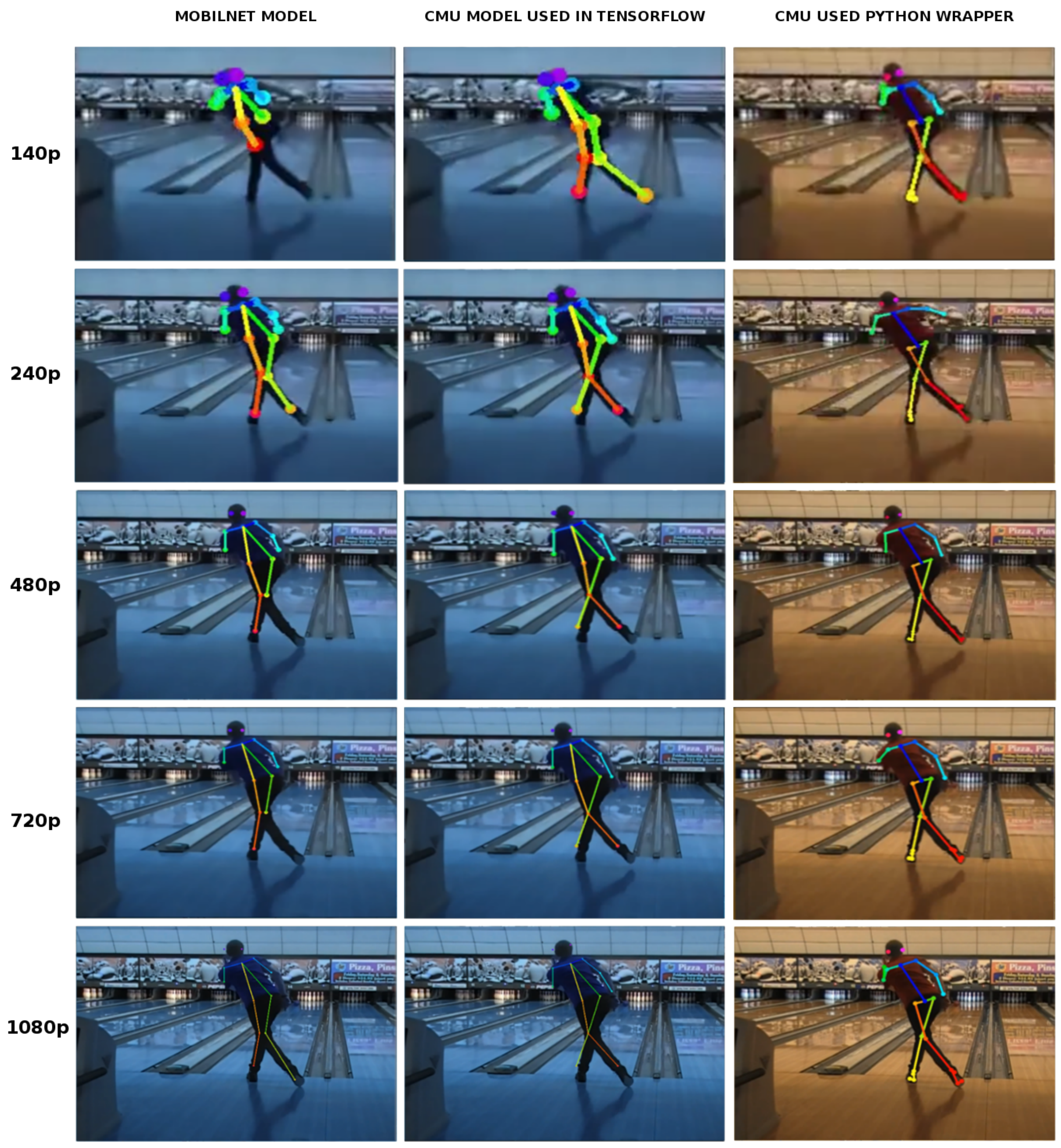


Figure 6.3: Model Comparison per Resolution on frame with one person - Low Difficulty

In Medium-Difficulty Frame (Figure 6.2), there are 2 people and 1 person's leg, with the central person holding a barbell, which could possibly confuse the detection. All three (3) models detect accurately the central person, which is also the main subject, but *cmu Python Wrapper's* model detects both people in the frame and the person's leg from 140p, while the second person is detected only by *cmu model with tensorflow* in 1080p.

Lastly in Low-Difficulty Frame (Figure 6.3), there is only one person. All three (3) models detect accurately the person, which is also the main subject, but *cmu Python Wrapper's* model detects both people in the frame and the person's leg from 140p, while the second person is detected only by *cmu model with tensorflow* in 1080p.

In conclusion, the best model appears to be *cmu Python Wrapper*, which in higher difficulty frames it appears to have the best results after the resolution of 720p.

## Video Duration

As seen in subsection 5.2.1, the videos of the dataset have durations that are within the range of a few seconds up to 8.8 hours long. If a brute-force approach was followed and only 10 seconds were used from each video, aka approximately 270 frames ( based on the foundings in subsection 5.2.1 indicating that the average video has 27 frames per second ), with OpenPose Python API cmu model being the fastest and most accurate, with 1 second per pose extraction (as seen in subsection 4.2.2), it would require 35 days before we had the pose features. Additionally, during the random selection of 10 second video fraction, it would be unknown whether the video contained people conducting the actual sport or noise, such as zooming into the audience, on player's face or just showing the room during break time. Ideally, we could double our sample to increase the probability of actually capturing the sport, but that comes at a cost of doubling the processing time. As is well understood, it is a multivariate problem and the decisions on how the features are extracted would affect directly the accuracy of the classification model. For these reasons, it was attempted to ensure quality video features for the least amount of necessary time, by applying minimal filtering on the frames before selection and restrict the frames into a specific number for each video.

### Quantity of frames selected:

Intuitively, it seems computationally expensive and reduntant to use all frames within the second fraction of the video, since the changes in human motion doesn't change as fast. For this reason, it was decided first to select two (2) frames per second, in order to capture the possible pose changes that may happen in a second, without increasing too much the processing time.

In order to gain more control over the processing time and due to the high variability of videos' durations, it was also important to determine the maximum frames to be selected from each video. This approach was trying to prevent the excessive load of processing time on the longer videos. As seen in subsection 5.2.1, there are 44 videos between 30 minutes and 1 hour and if an one (1) hour video would take two (2) hours to process (at a 2 fps rate), then for the 56 videos alone would require approximately 4.5 days. Additionally, this approach allowed the utilization of the whole dataset, even the 44 videos that are over 1 hour, that in other case would might be wiser to discard.

The window of frames selected for each frame was set to 30, which corresponds to 15 seconds of video time and approximately 4 days of processing time for the entire dataset. A smaller window to this would risk to not capture the sport in action, so it was preferred to increase the number of frames and the probabilities accordingly.

### Quality of frames selected:

The approach of the 30 frames window was thought as a way to increase the probabilities of capturing the sport in action, but at the same time it also increased the probability of those frames containing noise. For that reason it was attempted to filter each frame before selection and those that were lower than the determined threshold of *frame interest* to be discarded while the rest to be added in the list of selected frames.

**Index of Interest:** In a controlled environment where the people conducting the sport were isolated and centered towards the camera, the sport motion is sure that it would be captured successfully, as there are no unrelated information interrupting it. On the other hand, on videos with unconstrained environments the behavior of the person recording the video is uncertain. More specifically, the video can contain zooms into the face, hands etc. of the person who conducts the sport, or shift the focus into the reaction of the crowd and even focusing on the empty field, course, track etc. The pose estimation model does not distinguish between people of interest and unrelated noise people, so it will extract the poses of all people in the frame.

Feeding all the people of the frame into the classification model, would introduce a lot of noise, increase its computational complexity and processing time and decrease the accuracy of the model. To avoid that, the frames were filtered and only up to two (2) most relevant people were kept as features for the classification model. This will not necessarily prevent the model to save poses from frames with crowd but it will make it harder and will ensure that when it does, it selects the least amount possible. Finally, it will make sure that frames without people, or a few not important people, along with zooms into a person's face or body part would be discarded.

The technique used to achieve that was the *Index of Interest*, which aimed to keep only the most *interesting* people of the frame, as can be seen in Figure 6.7. What makes a person *interesting* is how **complete**, **central** and **big** is in relation to the rest of the people in the frame. These are represented by the indexes of completeness, size and centrality that are calculated for each person in the frame.

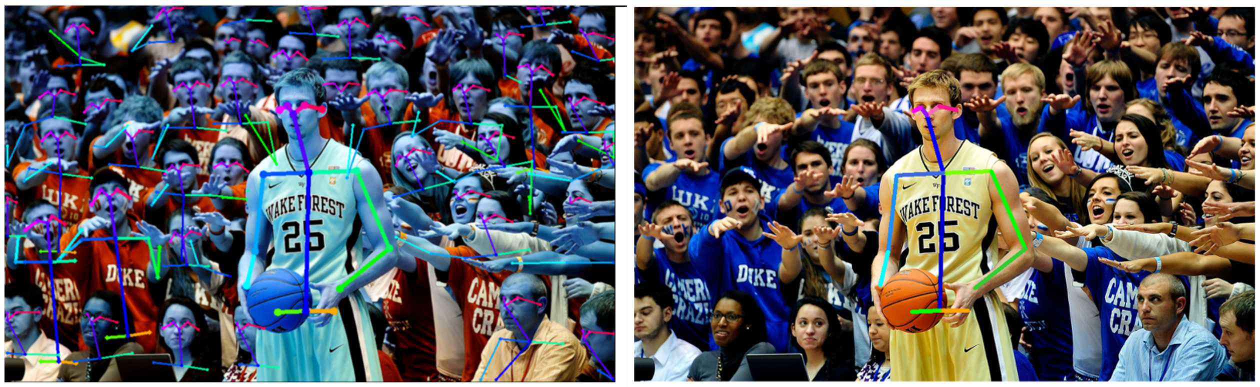


Figure 6.4: Before and After applying Index of Interest on Frame

The **completeness** of each person is calculated based on the body landmarks of BODY_25. Each body point is assigned with a weight depending on the importance of the joint capturing the motion of a person. All weights sum up to 1, with 1 meaning that the person is 100% complete and 0 as 0% complete. As seen in Figure 6.5, points like nose, ears and feet were considered less important and were assigned the lowest weights. On the other hand points like shoulders, arms, forearms, pelvis and thighs had higher weights. Finally, if the person doesn't have a torso (aka the points of the neck and the pelvis) then it is immediately considered an incomplete person and therefore is assigned a 0 completeness score.

The **size** of each person was calculated by comparing the person's torso to the torso of the biggest person that could hypothetically appear on the frame. In order to determine the biggest possible person, a picture of a person in a relaxed pose was taken and pose estimated. After acquiring the body parts of the person, they were summed up and then divided by each individual part, in order to find the ratios of each in relation to the overall body part summation. Then the torso's ratio was used as a rule to extract the same proportion from the diagonal line of the frame (starting from one corner of the frame to the other) - which was serving as the biggest possible distance existing in the current frame - and estimate the biggest possible person's torso. Finally, the person's size was expressed as the ratio of the person's torso to the biggest possible person's torso.

The **centrality** of each person was calculated by how many of its body parts were located in the center of the frame. More specifically, two centers were created, the outer and the inner. For the outer center the frame is divided into 6 pieces in both x and y axis and a rectangle is drawn from 1 to 5 in both axes creating the center and excluding the padding left around it. Lastly for the inner center, the rectangle is drawn from 2 to 4 in both axes again, serving as a stricter center. Based on those two centers, if the person's body joint is located inside the inner center, is multiplied by a coefficient to increase the importance of those body joints. Lastly, the inner and outer center's body joints are summed with the final score signifying the centrality of the person. As seen in Figure 6.6, the people that are further away from the outer and inner center are not taken into account (only the torso of the people was drawn on the figures).

By the end of these calculcations, each person would be represented as an array of their completeness, size and centrality indexes. These indexes will determine who is selected as the most interesting. More specifically,

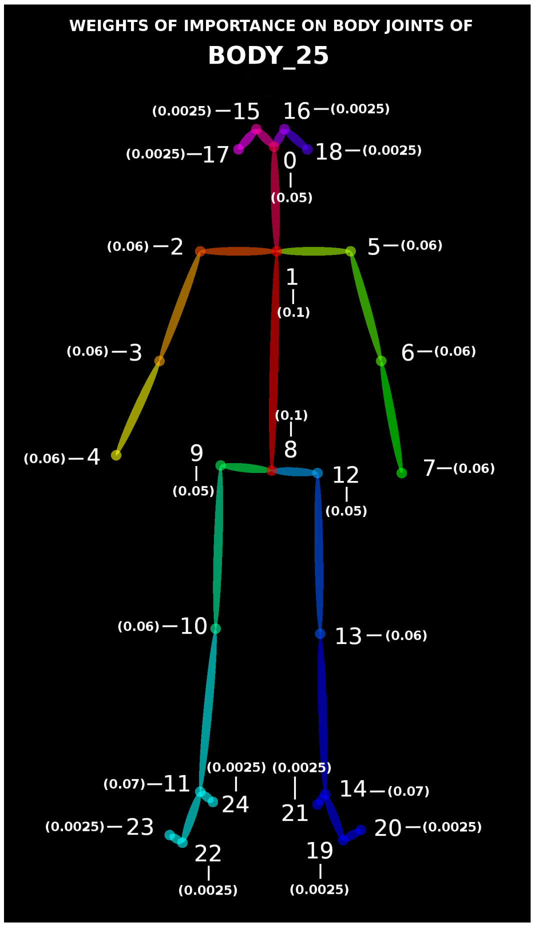


Figure 6.5: Weights assigned to each body joint of BODY_25's body landmarks model

the control flow will first find people that their level of completeness is greater than 0.7 (aka 70% complete). Then, from the pool of the complete people found, only the people with level of centrality of 1 (aka 100% central) are selected. Then, central peoples' deviation is calculated along with the deviation of each person's size. The deviation of the top 2 biggest and most central people is set as a threshold of minimum deviation a person should have before is considered adequately big and central. At the final stage, the people who can meet both the deviation threshold of size and centrality are selected from the pool of complete people. The maximum number of *interesting* people that can be selected is either 1 or 2.

**Sequences of Frames:** The examination of each frame before selection with the *index of interest* would filter out noise and avoid possible unrelated frames, but at the expense of additional time. More specifically, in terms of processing time, the worst case scenario is that a large video would not contain the sport at all and it would have to go through the entire video without finding any *interesting* frame, adding up empty time. Additionally, in terms of quality of frames, the worst case scenario is to go through an entire long video and attain 30 sporadically selected frames, in which case the goal of capturing the motion in poses would fail. For this reason it was decided that only the 30 sequential frames of each video would be kept for processing, risking to lose the videos that don't suffice.

The dataset was processed and from each video was attempted to acquire 30 frames, depending on their *index of interest*. As seen in Figure 6.7 from the 10.646 overall videos of the dataset, 10.327 were found to have at least 1 *interesting* frame. From those 10.327 videos, 8.572 were found to have 30 *interesting* frames and 414 to have less than 10 as seen in Figure 6.8. The fact that not all videos found to have *interesting* frames, indicated and was manually confirmed, that probably the remaining 319 videos didn't include people conducting a sport.

Lastly, the 10.327 *interesting* videos, were investigated further to see how many of those had sequential

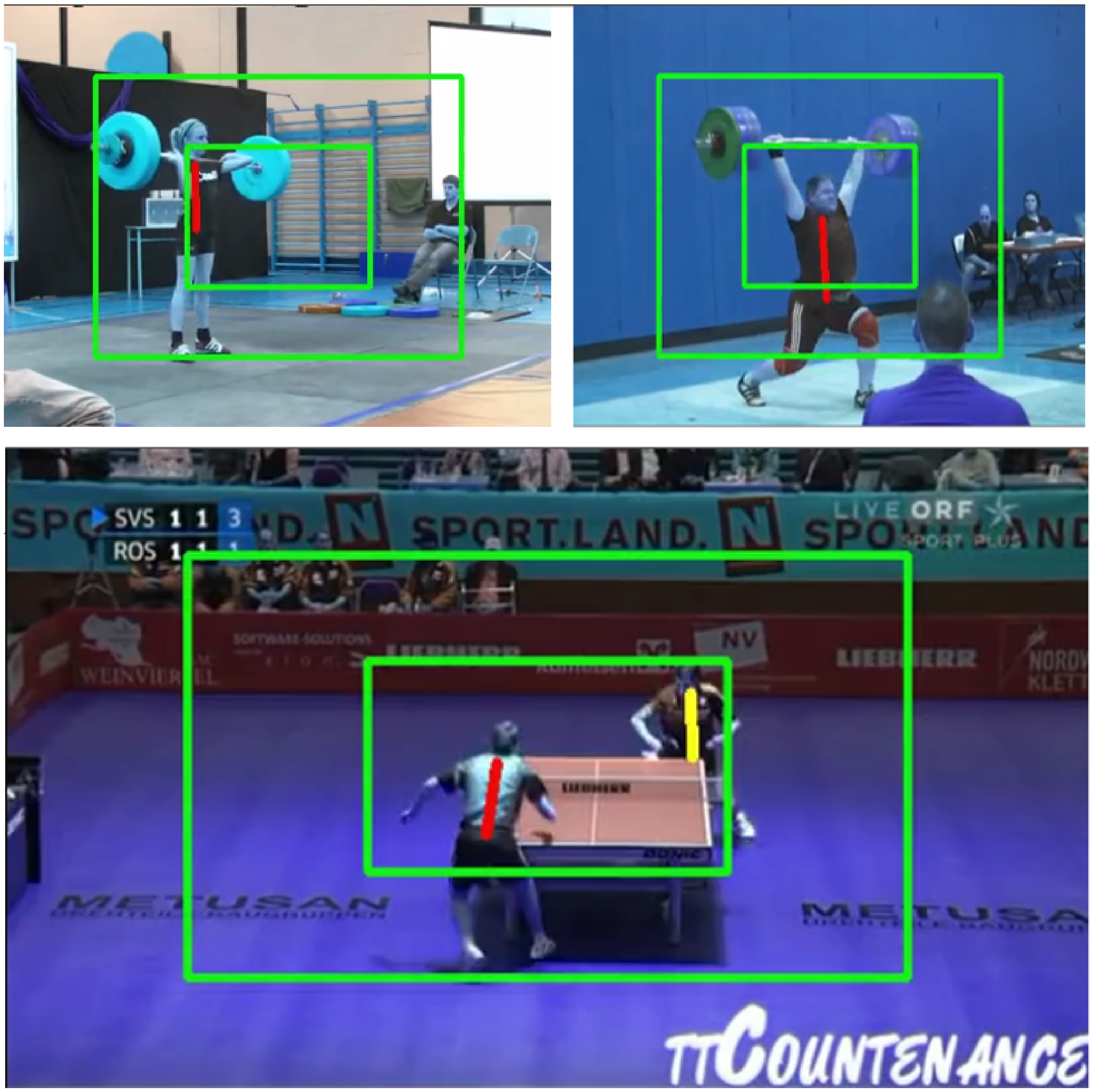
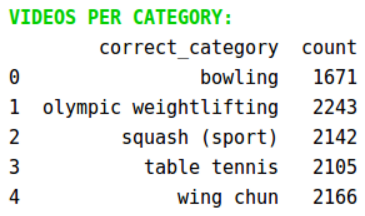


Figure 6.6: Results from the Centrality Measurement

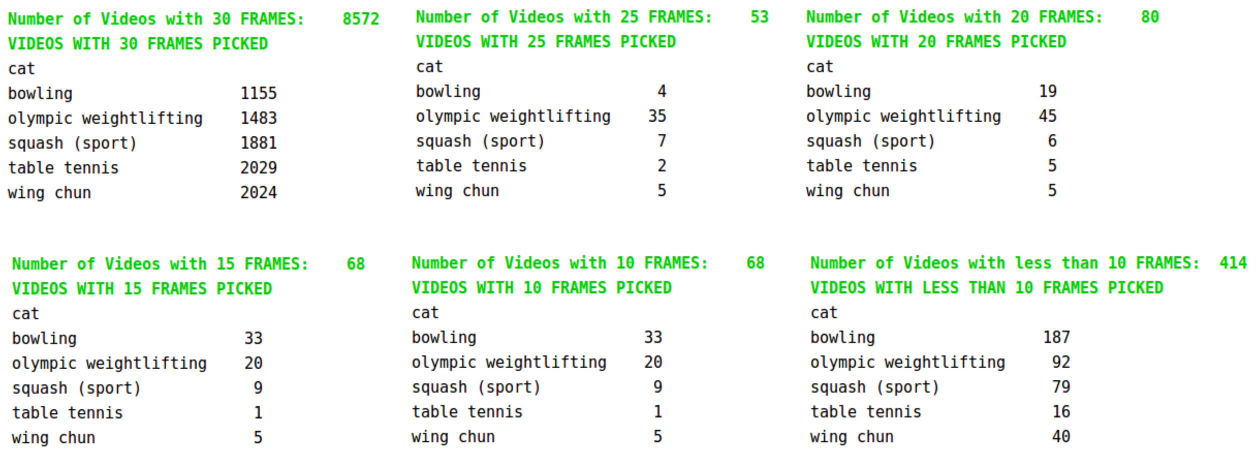
frames, which would increase the probability that the sport's motion was captured. As seen in Figure 6.9, 9.126 videos were found to have 10 sequential frames (seq.fr.) and with those relatively enough videos maintained up until the 15 seq.fr. with 7665 frames. After the 15 seq.fr. it appears in Figure 6.10 that the number of videos declines further with 5.974 videos found to have 20 seq.fr., 4.763 with 25seq.fr. and finally 3.836 with 30 seq.fr.

As important the number of videos found to have sequential frames is, it is not the only criteria to decide on the configuration of the features fed into the LSTM classification model. It is equally important to observe the way those numbers of videos are distributed per category. It is apparent that in 10 and 11 seq.fr the catogories seems quite balanced and after 12 seq.fr the *bowling* category starts to have less than 1000 observations and goes all the way down to 140 for 30 seq.fr. while the rest categories seem to join but with much slower pace.



```
VIDEOS PER CATEGORY:
        correct_category  count
0                bowling   1671
1   olympic weightlifting   2243
2          squash (sport)   2142
3            table tennis   2105
4               wing chun   2166
```

Figure 6.7: Videos per category that were found to have at least 1 frame interesting



```
Number of Videos with 30 FRAMES:    8572
VIDEOS WITH 30 FRAMES PICKED
cat
bowling                    1155
olympic weightlifting      1483
squash (sport)             1881
table tennis               2029
wing chun                  2024
```

```
Number of Videos with 25 FRAMES:    53
VIDEOS WITH 25 FRAMES PICKED
cat
bowling                     4
olympic weightlifting      35
squash (sport)              7
table tennis                2
wing chun                   5
```

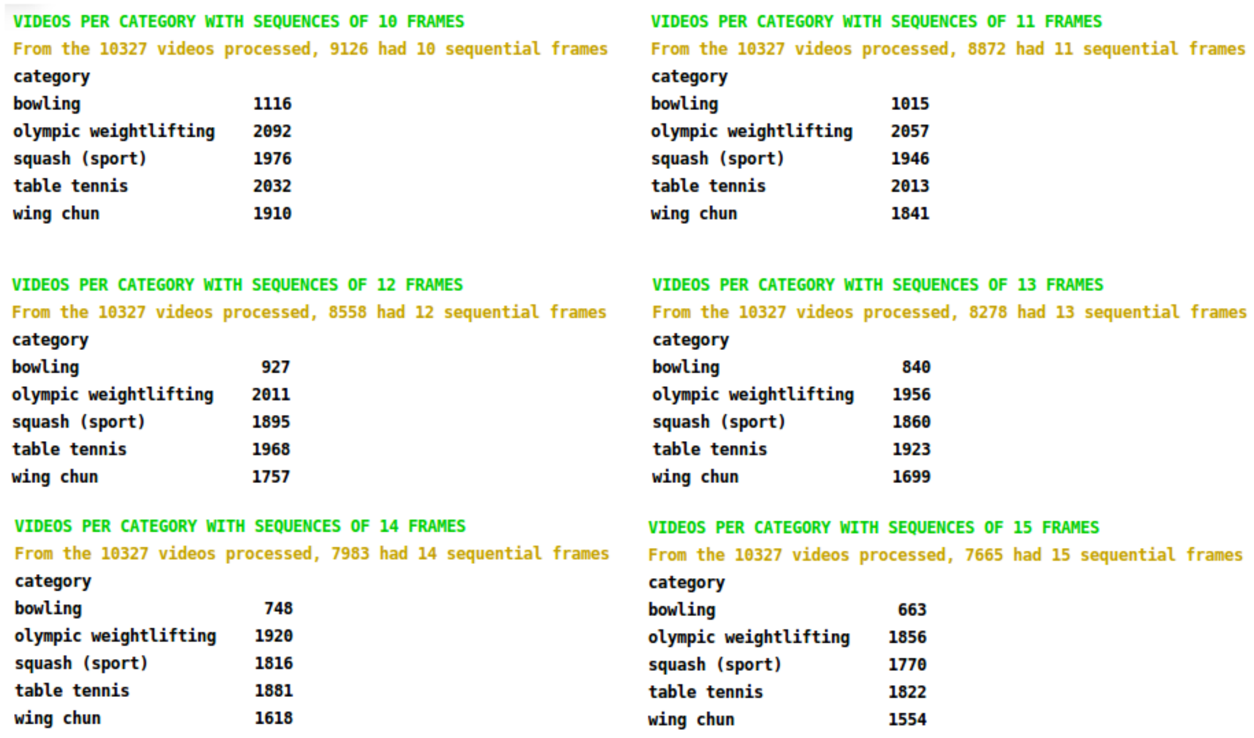
```
Number of Videos with 20 FRAMES:    80
VIDEOS WITH 20 FRAMES PICKED
cat
bowling                    19
olympic weightlifting      45
squash (sport)              6
table tennis                5
wing chun                   5
```

```
Number of Videos with 15 FRAMES:    68
VIDEOS WITH 15 FRAMES PICKED
cat
bowling                    33
olympic weightlifting      20
squash (sport)              9
table tennis                1
wing chun                   5
```

```
Number of Videos with 10 FRAMES:    68
VIDEOS WITH 10 FRAMES PICKED
cat
bowling                    33
olympic weightlifting      20
squash (sport)              9
table tennis                1
wing chun                   5
```

```
Number of Videos with less than 10 FRAMES:  414
VIDEOS WITH LESS THAN 10 FRAMES PICKED
cat
bowling                    187
olympic weightlifting       92
squash (sport)              79
table tennis                16
wing chun                   40
```

Figure 6.8: Videos per category that were found to have at least 30, 25, 20, 15, 10 and less than 10 interesting frames



```
VIDEOS PER CATEGORY WITH SEQUENCES OF 10 FRAMES
From the 10327 videos processed, 9126 had 10 sequential frames
category
bowling                  1116
olympic weightlifting    2092
squash (sport)           1976
table tennis             2032
wing chun                1910
```

```
VIDEOS PER CATEGORY WITH SEQUENCES OF 11 FRAMES
From the 10327 videos processed, 8872 had 11 sequential frames
category
bowling                  1015
olympic weightlifting    2057
squash (sport)           1946
table tennis             2013
wing chun                1841
```

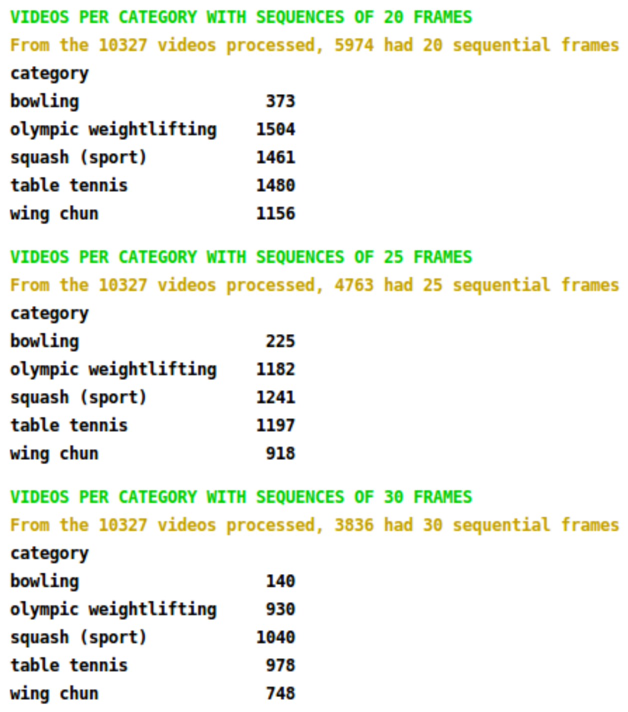
```
VIDEOS PER CATEGORY WITH SEQUENCES OF 12 FRAMES
From the 10327 videos processed, 8558 had 12 sequential frames
category
bowling                   927
olympic weightlifting    2011
squash (sport)           1895
table tennis             1968
wing chun                1757
```

```
VIDEOS PER CATEGORY WITH SEQUENCES OF 13 FRAMES
From the 10327 videos processed, 8278 had 13 sequential frames
category
bowling                   840
olympic weightlifting    1956
squash (sport)           1860
table tennis             1923
wing chun                1699
```

```
VIDEOS PER CATEGORY WITH SEQUENCES OF 14 FRAMES
From the 10327 videos processed, 7983 had 14 sequential frames
category
bowling                   748
olympic weightlifting    1920
squash (sport)           1816
table tennis             1881
wing chun                1618
```

```
VIDEOS PER CATEGORY WITH SEQUENCES OF 15 FRAMES
From the 10327 videos processed, 7665 had 15 sequential frames
category
bowling                   663
olympic weightlifting    1856
squash (sport)           1770
table tennis             1822
wing chun                1554
```

Figure 6.9: Videos per category that contain sequences of 10, 11, 12, 13, 14 and 15 frames



```
VIDEOS PER CATEGORY WITH SEQUENCES OF 20 FRAMES
From the 10327 videos processed, 5974 had 20 sequential frames
category
bowling                  373
olympic weightlifting   1504
squash (sport)          1461
table tennis            1480
wing chun               1156

VIDEOS PER CATEGORY WITH SEQUENCES OF 25 FRAMES
From the 10327 videos processed, 4763 had 25 sequential frames
category
bowling                  225
olympic weightlifting   1182
squash (sport)          1241
table tennis            1197
wing chun                918

VIDEOS PER CATEGORY WITH SEQUENCES OF 30 FRAMES
From the 10327 videos processed, 3836 had 30 sequential frames
category
bowling                  140
olympic weightlifting    930
squash (sport)          1040
table tennis             978
wing chun                748
```

Figure 6.10: Videos per category that contain sequences of 20, 25 and 30 frames

**Distance between the Frames:** Another important matter, that had to be examined, was the number of frames each frame would have to be away from the next in the sequence of the 30 selected frames per video. Hypothetically speaking, the closer the frames are to one another, the more probable would be that the video segment selected had captured the sport motion. For this reason the accuracy of the model was tested for 1, 10 and 20 frames away (10 seconds).

# Chapter 7

# Results

To create an accurate model it was important to investigate the optimal hyperparameter values that would work at the benefit of the model. The highest accuracy achieved was **89%** through a network of stacked LSTM layers (for visual description of the model and features see Figure 7.2, Figure 7.3 and Figure 7.4). More specifically, the model consisted of 2 LSTM layers, with 64 and 32 units of depth respectively and L1 (Lasso Regression), L2 (Ridge Regression) regularization techniques applied on each layer. The last layer was a densely connected Neural Network with 5 units, same as the number of sport-classes. The optimization algorithm used to update the network weights was *Adaptive Moment Estimation (adam)*, with *categorical cross entropy* as the loss function. The network received as input batches of 32 sequences of 30x100 datapoints per video, rescales to be within the range of -1 and 1. These points represented the human poses (x,y joint coordinates - 25 pairs/50 points per person) of 30 frames, where at least 1 or 2 *interesting* people were found (see Figure 7.1). These 30 frames were selected in a way to not be more than 20 frames away (10 seconds) from one another based on the way it was ordered on the video.

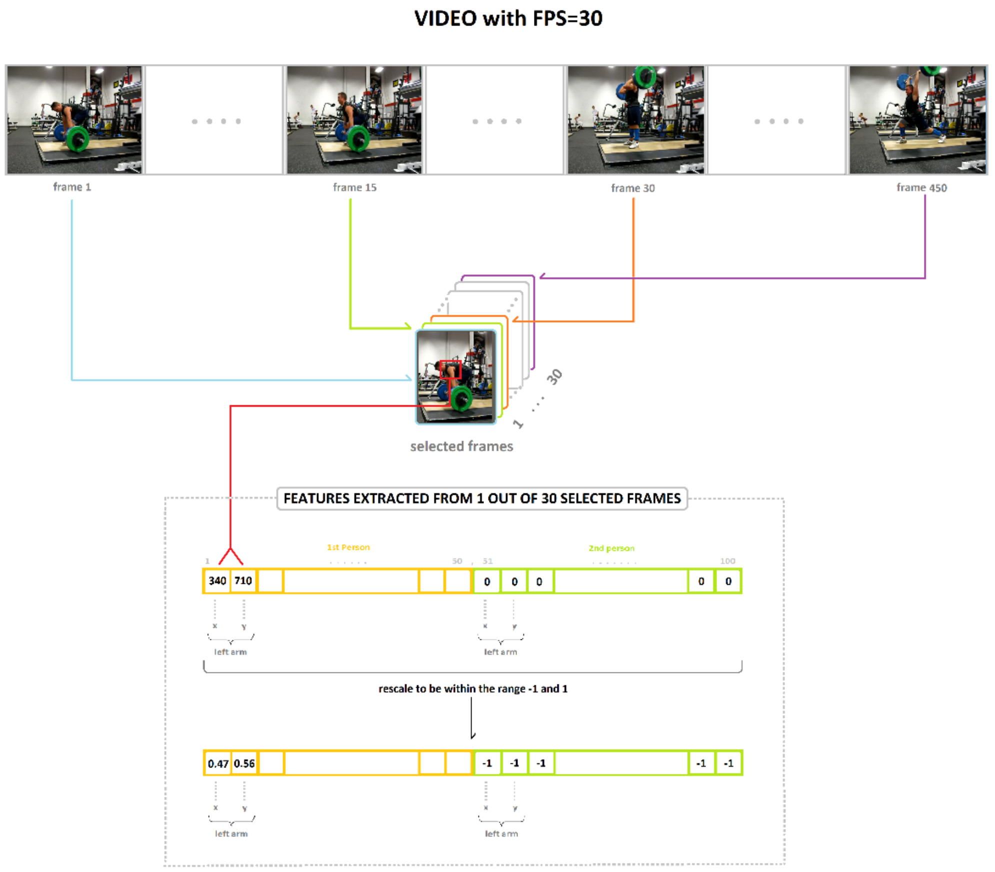


Figure 7.1: Feature Extraction from videos

## Hyperparameter Tuning

When defining a new model it is important to determine those *hyperparameters* that will optimize the *parameters* of our model. More specifically, values that are external to the inner workings of the model and cannot be estimated from data, such as the number of *hidden layers, hidden units, regularization techniques* e.t.c. are considered to be *hyperparameters*, because ultimately they control the actual *parameters* of the model, such as *model weights, bias* that are internal to the model and are estimated from the learning process of the given data [58] [66]. In order to tune the hyperparameters, different experiments were conducted concerning the:

1. **Format of the features**
2. **Regularization Techniques**
3. **Network Architectures**
4. **Number of frames to represent a video**
5. **Distance between frames that represent the video**
6. **Dataset Augmentation**

The metrics used to measure the efficiency of the model are **accuracy, precision, recall, f1-score** along with their **confusion matrices**. The initial experiments were conducted with a network architecture that consisted of a single LSTM cell with 2048 units depth layer, followed by a regular densely-connected NN with 5 nodes, the same number as the sports' classes. The network was tested on different hyperparameters to understand how it behaves. After getting a grasp of what favors the model, different architectures were examined. The aforementioned variables are analysed below:

**Format of features:** The training features were transformed and rescaled to be within different ranges, in order to understand how they affect the accuracy of the model. A network that consisted of a single LSTM cell with 2048 units depth layer, followed by a regular densely-connected NN with 5 nodes, as the number of sport classes, was set up and trained with different formats each time, to understand how it affects the accuracy:

- *Rescaled data to be within the range of 0 and 1* by dividing each data point with the maximum value.

$$z = (max(X) - x)$$

- *Normalized data to be within the range of 0 and 1* & between *-1 to 1* with the equation below:

$$\frac{x_i - \min(\boldsymbol{x})}{\max(\boldsymbol{x}) - \min(\boldsymbol{x})}$$

- *Standardized data* to have a distribution with mean value of 0 and standard deviation of 1, using this equation:

$$z = (x - u)/s$$

$\mathbf{u}$ = mean of training samples and $\mathbf{s}$ = standard deviation

The accuracy of each transformation was 63%, 54%, 72% and 64% respectively. Consequently, the feature transformation selected was the rescale of data within the range of -1 and 1.

**Regularization Techniques:** A substantial difference between *validation* and *test* accuracy was observed, indicating overfitting. The latter happens when during training the model takes into account too many parameters to describe the data and although it captures the structure of the training data, it cannot generalize well into new data. This way the model ends up becoming too complex and tailored for the training data's needs. To combat overfitting, different regularization techniques are used across literature [9, 62] that either penalize the loss function, in order to select the most important parameters (*L1 & L2 norm* [5, 59], or randomly droping nodes and their connections accordingly in order to form a simpler, less noisy and generalizable model (*dropout*) [84, 90].

Again a network that consisted of a single LSTM cell with 2048 units depth layer network , followed by a regular densely-connected NN with 5 nodes, as the number of sport classes, was trained with the different regularization techniques, to understand how it affects accuracy:

- *L1 - Lasso Regression* values between 0.1
- *L2 - Ridge Regression* values between 0.1
- *Dropout* values of 0.5
- *L1=0.1 & L2=0.1*
- *L1=0.1 & L2=0.1 & Dropout=0.5*

The accuracy of each regularization technique was 78%, 76%, 65%, 79% and 64% respectively. Consequently, the regulization technique-combination selected was the L1 & L2.

**Network Architecture:** Deep architectures are not yet well studied, documented and tested, so a trial-and-error approach is often adopted. To test the different network architectures, we used as features the x,y coordinates of the anatomical joints of the two most interesting people in 30 selected frames, that could have maximum 20 frames (10 seconds) distance between them. The features were formatted to be within the range -1 and 1 and L1,L2 regularization techniques were used, as selected previously. The first set of network architectures consisted of an LSTM cell with different, proportionally increasing, unit depths, followed by a regular densely-connected NN of 5 nodes, as the number of sport classes. Finally, LSTM cells of proportionally decreasing depths were stacked in a descending order, as an attempt to achieve dimensionality reduction on our way to the last layer that conducts the classification. Mode specifically, the architectures examined were:

- *LSTM cell with 32 units depths*
- *LSTM cell with 64 units depths*
- *LSTM cell with 128 units depths*
- *LSTM cell with 256 units depths*
- *LSTM cell with 512 units depths*
- *LSTM cell with 1024 units depths*
- *LSTM cell with 2048 units depths*
- *stacked LSTM network of LSTM cells of 2048-1024-512-256-128-64-32 units depth*
- *stacked LSTM network of LSTM cells of 1024-512-256-128-64-32 units depth*
- *stacked LSTM network of LSTM cells of 512-256-128-64-32 units depth*
- *stacked LSTM network of LSTM cells of 256-128-64-32 units depth*
- *stacked LSTM network of LSTM cells of 128-64-32 units depth*
- *stacked LSTM network of LSTM cells of 64-32 units depth*

| Architecture | Accuracy |
|---|---|
| LSTM(32) + FCNN(5) | 87.2% |
| LSTM(64) + FCNN(5) | 86.9% |
| LSTM(128) + FCNN(5) | 87.5% |
| LSTM(256) + FCNN(5) | 88.7% |
| LSTM(512) + FCNN(5) | 87.8% |
| LSTM(1024) + FCNN(5) | 87.9% |
| LSTM(2048) + FCNN(5) | 87.7% |
| LSTM(2048) + ... + LSTM(32) + FCNN(5) | 88% |
| LSTM(1024) + ... + LSTM(32) + FCNN(5) | 88.1% |
| LSTM(512) + ... + LSTM(32) + FCNN(5) | 88.1% |
| LSTM(256) + ... + LSTM(32) + FCNN(5) | 88.4% |
| LSTM(128) + ... + LSTM(32) + FCNN(5) | 88.6% |
| LSTM(64) + LSTM(32) + FCNN(5) | 89% |

The accuracy for each architecture was:

The best architecture selected consisted of an LSTM cell of 64 units depth, an LSTM cell of 32 units depth and a regular densely-connected NN of 5 nodes, as the number of sport classes (see Figure 7.2).

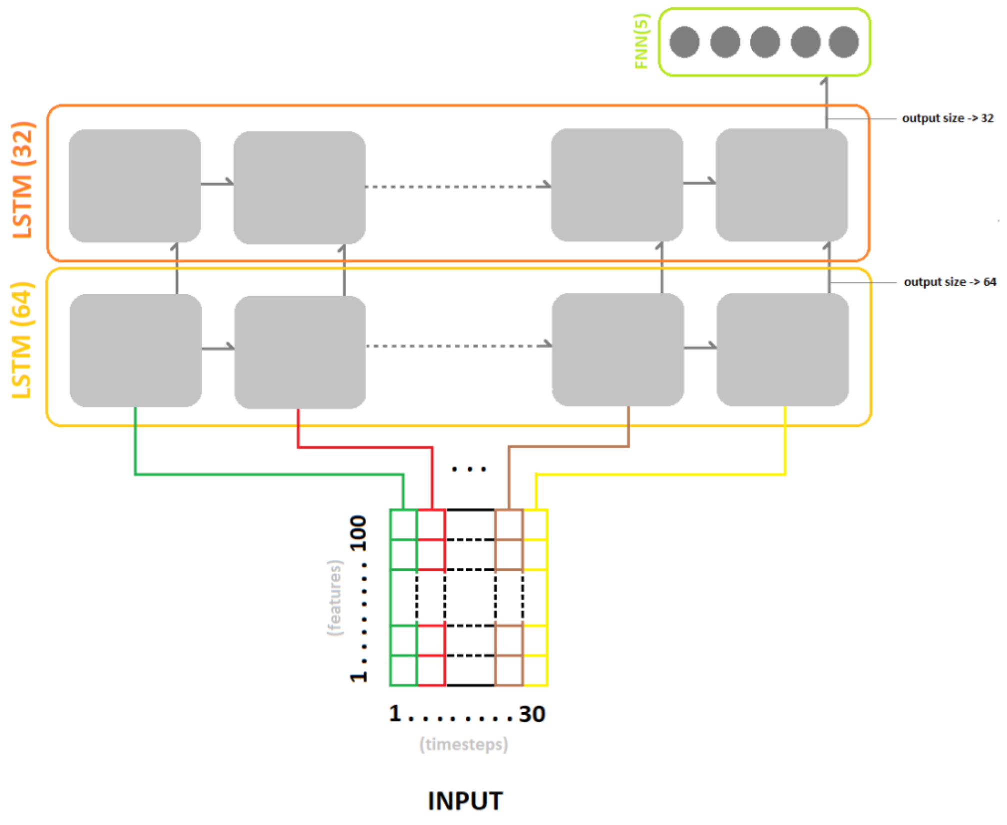


Figure 7.2: Model Architecture

**Distance between frames:** Previously, different network architectures were tested on features that consisted of coordinates to each anatomical joint of the 2 most interesting people found in 30 frames selected and were not more than 20 frames (10 seconds) away from one another. Different distances were examined:

- *1 second away - 2 frames*
- *10 seconds away - 2 frames*
- *20 seconds away - 40 frames*
- *30 seconds away - 60 frames*

The accuracy for each distance was 87.9%, 89%, 88.3% , 88% respectively. The distance selected was 10 seconds away or 20 frames.

**Number of frames to represent a video:** Until now the network was tested on features extracted from 30 frames. Different lengths of frames were examined:

- *10 frames* with 12928 train, 4032 test and 3232 validation videos
- *20 Frames* with 3072 train, 928 test and 736 validation videos
- *30 Frames* with 2432 train, 768 test and 608 validation videos

The accuracy for each length was 55.5%, 86.4% and 89%. The 30 frames were confirmed to be the best [1]. The feature shape can be seen in Figure 7.3.

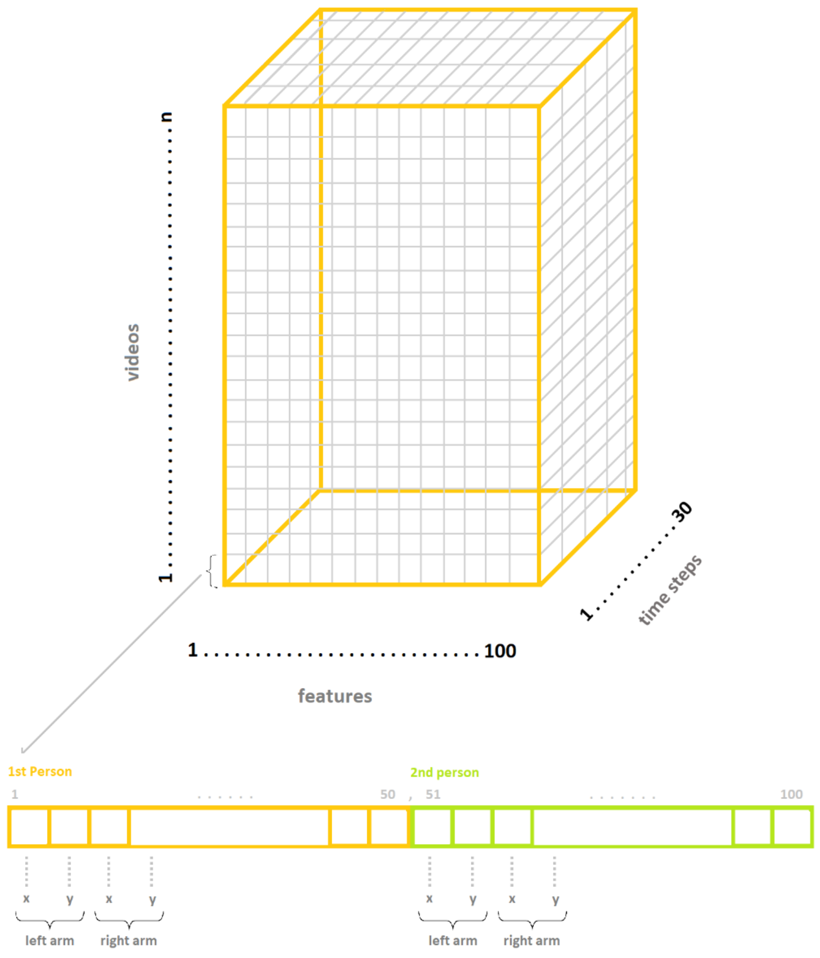


Figure 7.3: Model Architecture

[1]The reason the number of videos doesn't sum up to the video results in subsection 6.2.2 is because in a 30-frame sequence there might exist three (3) 10-frame sequences, or two (2) 15-frame sequences. Also in order to make the number of videos divisible to the batch size, some videos had to be discarded.

**Dataset Augmentation:** Due to imbalanced dataset, there was an attempt to increase the samples of the underepresented sport-classes with data-augmentation. The dataset was shifted *up*, *down*, *left* and *right* by:

- 1, 5, 10 and 15 pixels
- 10, 20, 30, 40 and 50 pixels

Unfortunately, it did not increase the accuracy of the model.

The final model's architecture can be inspected in detail in Figure 7.4 below:

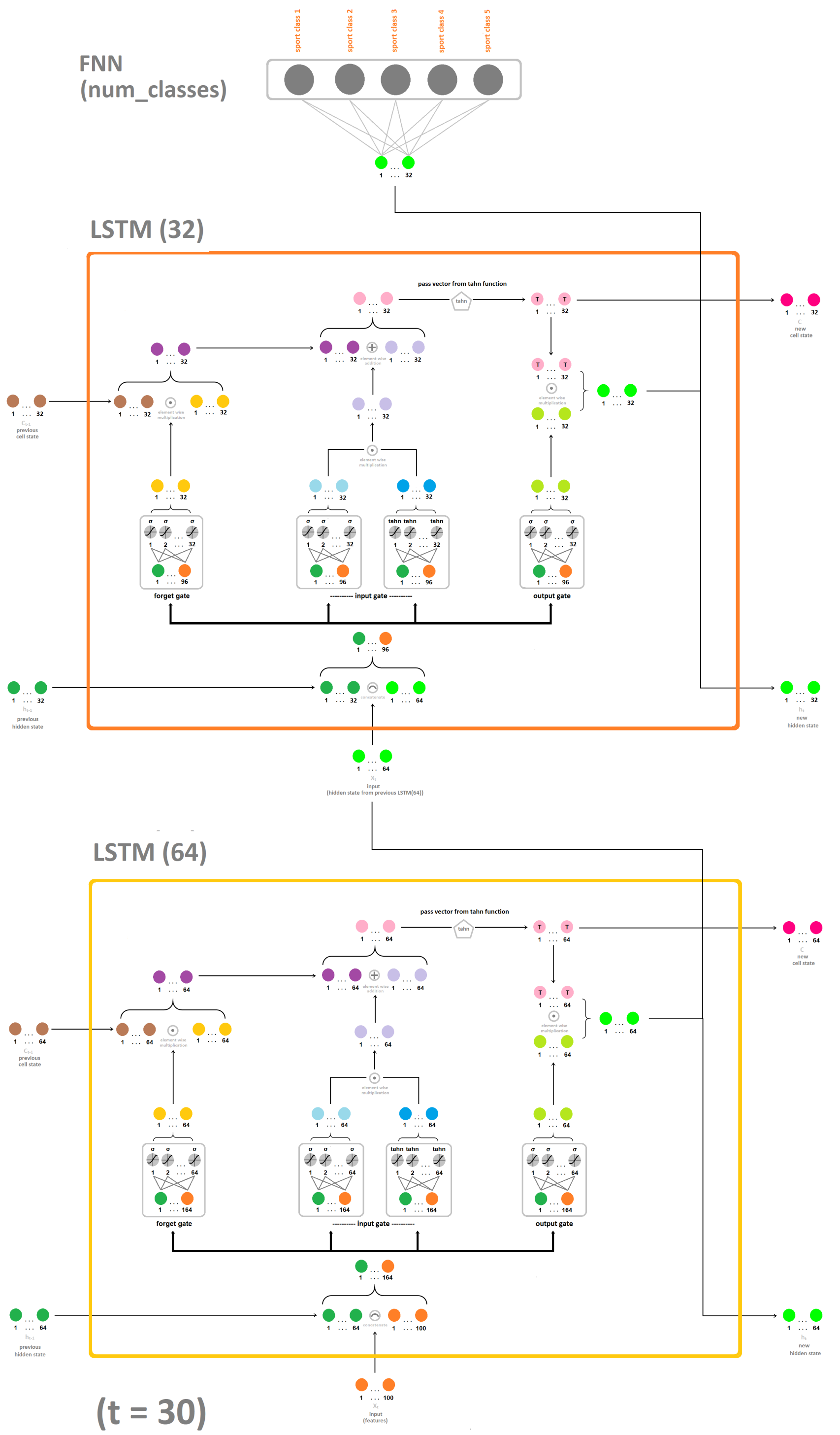


Figure 7.4: Detailed Model Architecture

In conclusion, the different tests conducted indicate that the number of sequential frames affect the accuracy of the model. It was observed that the higher the number of sequential frames, the higher the accuracy. Possibly an increase on the frame level by selecting more than 2 frames per second or on the video level by adding more than 30 sequential frames, can affect the accuracy in a similar manner. Additionally, as seen in subsection 6.2.2, the number of videos were not balanced for 30 sequential frames, with *bowling* appearing to be highly reluctant, with only 140 videos. This could possibly has affected model's accuracy.

## Model Consistency

The model was able to achieve accuracy as high as **89%**, but it was tested only on 5 from the 487 sport categories of Sport-1M dataset. To investigate the model's consistency to accurate predictions, 5 more sports-categories were added to the dataset. The sport-categories added were:

- **wrestling** with 1378 videos
- **golf** with 668 videos
- **skateboarding** with 426 videos
- **marathon** with 1036 videos
- **fencing** with 2108 videos

The model's accuracy dropped from **89%** to **73%**, see comparison in **??** below:

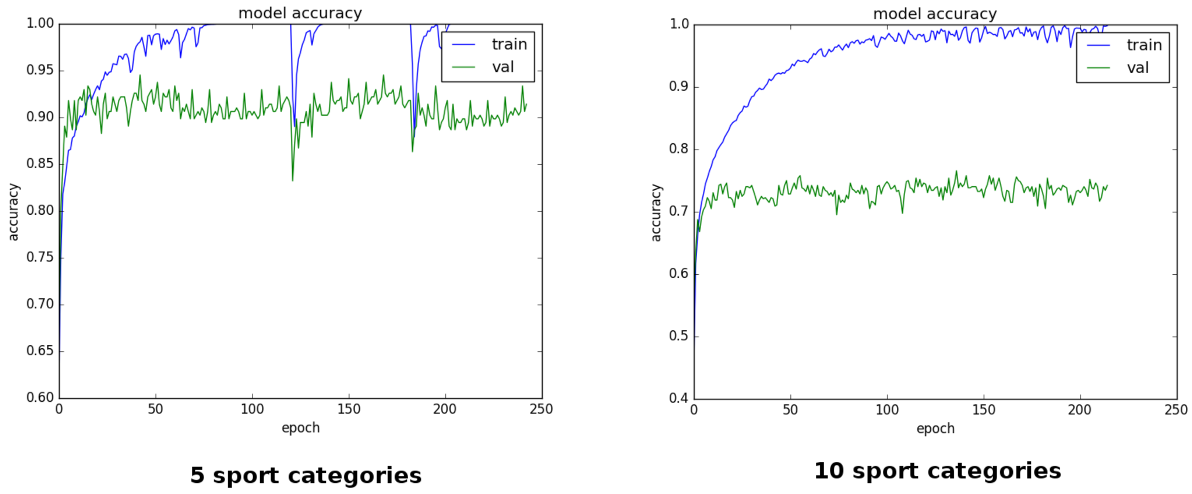


Figure 7.5: Model Accuracy with 5 and 10 sport categories

## Study of Errors

The model achieved an accuracy of **89%** for 5 sport classes and **73%** for 10 sport classes. Although high accuracy is an indicator of good performance, it can be misleading, especially for datasets with large class imbalances. The predictive power of a model derives from its ability to classify correctly, while is avoiding **Type I errors**, or false-positive (FP) prediction. This can become more clear when the classification task concerns a medical prediction, where ones life depends on it. Other metrics that can shed more light on the predictive ability of the model is:

- **Precision** is the ratio of the correctly classified samples of a class $x$ to the total number of samples that were classified as class $x$ (True Positive (TP) / TP + False Positive (FN)). Another way to see precision is as the fraction of correctly classified samples of a class. The question it answers is: *How many of the samples my model classified as class $x$, were actually class $x$?* High precision indicates a low FP rate. So high precision would mean that from 12 videos that were classified as sport $x$, the 10 were actually sport $x$.

- **Recall** is the ratio of the correctly classified samples of class $x$ to the total number of samples that actually belong at class $x$ (TP/TP+FN). The question it answers is:*How many of the samples that actually belong in class $x$ were identified/recalled?* So high recall would mean that from 10 videos of sport $x$, my system identifies the 9.

- **F1-score** is the harmonic mean of precision and recall.

$$F1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

These metrics are presented in Table 7.1 and Table 7.2, for 5-class and 10-class model, respectively. On both tables, it can be observed that the sport-classes with the most samples, achieved higher *precision, recall* and *f1-score*. For facilitation, the sport-categories are enumerated below, in descending order, by their number of samples:

1. fencing
2. table tennis
3. wing chun
4. squash
5. olympic weightlifting
6. wrestling
7. marathon
8. bowling
9. golf
10. skateboarding

Table 7.1: Mean Precision - Recall - F1 score of 5 sport classes

| Sport | precision | recall | f1 score |
|---|---|---|---|
| wing chun | 0.924 | 0.924 | 0.922 |
| squash | 0.864 | 0.852 | 0.858 |
| table tennis | 0.92 | 0.924 | 0.92 |
| olympic weightlifting | 0.89 | 0.884 | 0.882 |
| bowling | 0.638 | 0.646 | 0.638 |

Table 7.2: Mean Precision - Recall - F1 score of 10 sport classes

| Sport | precision | recall | f1 score |
|---|---|---|---|
| golf | 0.492 | 0.452 | 0.464 |
| fencing | 0.688 | 0.66 | 0.672 |
| wrestling | 0.684 | 0.712 | 0.694 |
| skateboarding | 0.646 | 0.672 | 0.656 |
| marathon | 0.508 | 0.516 | 0.51 |
| wing chun | 0.894 | 0.858 | 0.872 |
| squash | 0.692 | 0.716 | 0.7 |
| table tennis | 0.856 | 0.834 | 0.844 |
| olympic weightlifting | 0.798 | 0.796 | 0.794 |
| bowling | 0.558 | 0.63 | 0.59 |

To examine, in greater detail, the model's misclassifications, it was important to inspect the *confusion matrices* of both the 5-class and 10-class models. As observed in Table 7.3 and Table 7.4, the model was confusing *squash* to *table tennis*, mainly because they are both the same sport, played mainly on different courts. So they both incorporate the same movement patterns. Similar confusion can be observed on other sports, like *golf* and *bowling*, because they are both performed standing and utilize their upper half of the body to throw or kick the ball. Additionally, *marathon running* is sometimes confused with *fencing, squash* and *bowling*, because move pattern of running resembles a lot or can be a subset of other movements incorporated in other sports.

Table 7.3: Mean Confusion Matrix of 5 sport classes

| Sport | wing chun | squash | table tennis | olympic weightlifting | bowling | total |
|---|---|---|---|---|---|---|
| wing chun | 1462 | 35 | 13 | 39 | 31 | 1580 |
| squash | 34 | 1394 | 91 | 55 | 61 | 1635 |
| table tennis | 12 | 84 | 1703 | 15 | 31 | 1845 |
| olympic weightlifting | 45 | 54 | 17 | 1109 | 30 | 1255 |
| bowling | 33 | 47 | 32 | 32 | 261 | 405 |

Table 7.4: Mean Confusion Matrix of 10 sport classes

| Sport | golf | fencing | wrestling | skateboarding | marathon | wing chun | squash | table tennis | olympic weightlifting | bowling | total |
|---|---|---|---|---|---|---|---|---|---|---|---|
| golf | 282 | 47 | 30 | 21 | 39 | 17 | 61 | 18 | 47 | 63 | 625 |
| fencing | 42 | 1387 | 121 | 23 | 112 | 30 | 181 | 125 | 32 | 52 | 2105 |
| wrestling | 12 | 88 | 979 | 22 | 68 | 51 | 37 | 24 | 47 | 47 | 1375 |
| skateboarding | 17 | 16 | 23 | 285 | 19 | 0 | 34 | 4 | 16 | 11 | 425 |
| marathon | 35 | 107 | 67 | 17 | 535 | 30 | 100 | 20 | 42 | 82 | 1035 |
| wing chun | 31 | 34 | 52 | 8 | 59 | 1672 | 23 | 9 | 26 | 36 | 1950 |
| squash | 46 | 142 | 35 | 28 | 86 | 11 | 1299 | 63 | 49 | 56 | 1815 |
| table tennis | 15 | 138 | 33 | 8 | 26 | 7 | 62 | 1668 | 10 | 33 | 2000 |
| olympic weightlifting | 34 | 28 | 42 | 18 | 50 | 34 | 42 | 7 | 1143 | 42 | 1440 |
| bowling | 38 | 25 | 55 | 12 | 66 | 21 | 42 | 19 | 28 | 524 | 830 |

## Future Improvements

For future improvements it would be attempted to:

1. increase the number of frames selected per second (from 2 to three 3)
2. increase the time of video sample more than 15 seconds, or decrease it to less than 15 seconds, to capture the videos that are less than 15 seconds in length and decrease the dataset's class imbalance.
3. try Cyclical Learning Rate technique, developed by Leslie Smith [79]
4. convert each x,y coordinates of human joints into a value that would represent its distance from the pelvis of the person. In that way, the feature array of each person would not represent [x,y] coordinates in a 2D plane, but more meaningful values. This approach would aim to map how the distance of each body part change, as the movement on the video progresses
5. try other LSTM variants such as Peephold Connections, Full Gradient etc., although it is suggested that none of them significantly improves performance [28]

Chapter 8

# Conclusion

In this project, it was presented a way to conduct video classification by using human poses, represented as [x,y] joint coordinates, as features in a network of stacked LSTM cells. Previous work on video classification, has involved the utilization of frames as images for features to the network. Here is shown that extracted poses from video frames via Carnegie Mellon's OpenPose library can produce an accuracy of 89% when trained on 5 sport classes and 73% on 10 sport classes.

The main challenges of this project involved that the selected Sports-1M Dataset consists of realistic, user produced videos, publicly available on the internet. That means that it could contain user-noise in the form of varied illumination, scale, camera motion, viewpoint etc. along with content-noise in the form of focusing in unrelated topic to the one of interest, like capturing the crowd, zooming into players' face or zooming out on the surrounding environment etc. Additionally, the dataset was found to contain completely unrelated videos to the topic of interest because it is machine-generated. Lastly, the rest videos didn't have a fixed duration and resolution adding to the complexity and processing time of the problem.

The challenges were addressed by determining a fixed window of frames, that would be selected from each video (30 frames), with a specific amount of frames per second (2 fps), starting after the 30% of the video's duration. Additionally, to filter out as much as possible the aforementioned noise, there was constructed an index of interest with the goal to keep the most interesting or relevant people. The latter, were the people who were the biggest, most complete and central, compared to the rest of the frame.

The different experiments conducted in order to define the best hyper-paraneters of the network, indicated that a higher number of frames, either by selecting more than 2 frames per second, or by addind more than 30 frames, affects the accuracy of the model. Additionally, among the regularization methods, L1 and L2 seemed to perform best with LSTM units compared to Dropout.

This is a promising approach to other problems of the same domain, because it suggests an alternative way of achieving video classification and action recognition.

# Bibliography

[1] Coco-common object in context dataset. url:http://cocodataset.org/#keypoints-2018. Accessed: 2019-01-15.

[2] Vgg human pose estimation datasets. http://www.robots.ox.ac.uk/~vgg/data/pose_evaluation/. Accessed: 2019-01-15.

[3] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *CoRR*, abs/1609.08675, 2016.

[4] D.A. Clausi A.H. Shabani and J.S. Zelek. Salient feature detectors for human action recognition. 2012.

[5] Habtamu Alemu, Wei Wu, and Junhong Zhao. Feedforward neural networks with a hidden layer regularization method. *Symmetry*, 10:525, 10 2018.

[6] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[7] Kalid Azad. A calculus analogy: Integrals as multiplication. https://betterexplained.com/articles/a-calculus-analogy-integrals-as-multiplication/. Accessed: 2019-01-15.

[8] Moez Baccouche, Franck Mamalet, Christian Wolf, Christophe Garcia, and Atilla Baskurt. Sequential deep learning for human action recognition. 11 2011.

[9] Antonio Valerio Miceli Barone, Barry Haddow, Ulrich Germann, and Rico Sennrich. Regularization techniques for fine-tuning in neural machine translation. *CoRR*, abs/1707.09920, 2017.

[10] Yoshua Bengio. Deep supervised learning of representations. https://portal.klewel.com/watch/webcast/deep-learning-tools-and-methods-workshop/talk/1, July 2016. Idiap Research Institute Workshop. Accessed: 2019-01-15.

[11] M Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. Action as space-time shapes. In *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, volume 29, pages 1395– 1402 Vol. 2, 11 2005.

[12] Brilliant.org. Line integral. url:https://brilliant.org/wiki/line-integral/. Accessed: 2019-01-15.

[13] A. Pinz C. Feichtenhofer and R.P. Wildes. Dynamically Encoded Actions. Dynamically encoded actions based on spacetime saliency. page 2755–2764, 2015.

[14] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[15] Olivier Chomat and James Crowley. Probabilistic recognition of activity using local appearance. volume 2, pages –109 Vol. 2, 02 1999.

[16] Vasileios Choutas, Philippe Weinzaepfel, Jérôme Revaud, and Cordelia Schmid. Potion: Pose motion representation for action recognition. 2018.

[17] Guilhem Chéron, Ivan Laptev, and Cordelia Schmid. P-cnn: Pose-based cnn features for action recognition. 06 2015.

[18] Guilhem Chéron, Ivan Laptev, and Cordelia Schmid. P-cnn: Pose-based cnn features for action recognition. 06 2015.

[19] CMU-Perceptual-Computing-Lab. Github repo of openpose python api: openpose. https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/modules/python_module.md. Accessed: 2019-01-15.

[20] CMU-Perceptual-Computing-Lab. Openpose benchmark for gpus. https://github.com/CMU-Perceptual-Computing-Lab/openpose/blob/master/doc/faq.md#speed-up-memory-reduction-and-benchmark. Accessed: 2019-01-15.

[21] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, 2, 06 2005.

[22] Hadelin de Ponteves, Kirill Eremenko, and Charles Elkan. Online course: Deep learning and computer vision a-z$^{\mathrm{TM}}$: Opencv, ssd & gans. https://www.udemy.com/computer-vision-a-z/. Accessed: 2019-01-15.

[23] Thomas G. Dietterich. Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer-Verlag, 2002.

[24] P Dollar, V Rabaud, Garrison Cottrell, and Serge Belongie. Behavior recognition via sparse spatio-temporal features. pages 65– 72, 11 2005.

[25] Pedro Domingos. *The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World*. Basic Books, 2015.

[26] David J. Eck. *Introduction to Computer Graphics*. Hobart and William Smith Colleges, January 2018.

[27] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110, 11 2004.

[28] Klaus Greff, Rupesh Kumar Srivastava, Jan Koutník, Bas R. Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. 10 2017.

[29] Rıza Alp Güler, Natalia Neverova, and Iasonas Kokkinos. Densepose: Dense human pose estimation in the wild. 02 2018.

[30] C. Schmid H. Wang, A. Klaser and C.L. Liu. Action recognition by dense trajectories. 2011.

[31] Adam Harley. Ryerson university computer science department's interactive node-link vizualization of convolutional neural networks. http://scs.ryerson.ca/~aharley/vis/conv/flat.html, 2015. Accessed: 2019-01-15.

[32] John Haugeland. *Artificial Intelligence: The Very Idea*. A Bradford Book, 1989.

[33] Simon Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall, 2 edition, July 1998.

[34] George T. Heineman, Gary Pollice, and Stanley Selkow. *Algorithms in a Nutshell- A practical guide*. O'Reilly Media, April 2016.

[35] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[36] Ming-Kuei Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8(2):179–187, February 1962.

[37] Mohamed Hussein, Marwan Torki, Mohammad Gowayyed, and Motaz El Saban. Human action recognition using a temporal hierarchy of covariance descriptors on 3d joint locations. 08 2013.

[38] C. Schmid I. Laptev, M. Marszalek and B. Rozenfeld. Learning realistic human actions from movies. 2008.

[39] Hueihan Jhuang, Thomas Serre, Lior Wolf, and Tomaso Poggio. A biologically inspired system for action recognition. volume Vol. 1, pages 1–8, 01 2007.

[40] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, Jan 2013.

[41] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[42] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732, June 2014.

[43] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *CVPR*, 2014.

[44] Yan Ke, Rahul Sukthankar, and Martial Hebert. Event detection in crowded videos. pages 1–8, 01 2007.

[45] Ildoo Kim. Deep pose estimation implemented using tensorflow with custom architectures for fast inference repo: tf-pose-estimation. https://github.com/ildoonet/tf-pose-estimation. Accessed: 2019-01-15.

[46] Alexander Kläser, Marcin Marszalek, and Cordelia Schmid. A spatio-temporal descriptor based on 3d-gradients. 09 2008.

[47] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: A large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563, Nov 2011.

[48] Andrey Kurenkov. A 'brief' history of neural nets and deep learning. http://www.andreykurenkov.com/writing/ai/a-brief-history-of-neural-nets-and-deep-learning/, December 2015. Accessed: 2019-01-15.

[49] I Laptev and Tony Lindeberg. Space-time interest points. volume 64, pages 432–439 vol.1, 11 2003.

[50] Svetlana Lazebnik, Cordelia Schmid, and J Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. volume 2, pages 2169 – 2178, 02 2006.

[51] Zachary Chase Lipton. A critical review of recurrent neural networks for sequence learning. *CoRR*, abs/1506.00019, 2015.

[52] Fengjun Lv and Ramakant Nevatia. Recognition and segmentation of 3-d human action using hmm and multi-class adaboost. pages 359–372, 05 2006.

[53] S. Ma, L. Sigal, and S. Sclaroff. Learning activity progression in lstms for activity detection and early detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1942–1950, June 2016.

[54] M Marszalek, Ivan Laptev, and Cordelia Schmid. Actions in context. In *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2929 – 2936, 07 2009.

[55] Soo Min Kang and Richard Wildes. Review of action recognition and detection methods. pages 43–77, 10 2016.

[56] T.M. Mitchell. *Machine Learning*. McGraw-Hill International Editions. McGraw-Hill, 1997.

[57] Thomas Moeslund, Adrian Hilton, and Volker Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding*, 104:90–126, 11 2006.

[58] Andrew Ng, Kian Katanforoosh, and Younes Bensouda Mourri. Deep learning specialization. https://www.coursera.org/lecture/neural-networks-deep-learning/parameters-vs-hyperparameters-TBvb5. Accessed: 2019-01-10.

[59] Andrew Y. Ng. Feature selection, l1 vs. l2 regularization, and rotational invariance. In *ICML '04*, 2004.

[60] Michael A. Nielsen. *Neural Networks and Deep Learning.* Determination Press, October 2015.

[61] Sergey Nikolenko. Neuronuggets: Understanding human poses in real-time. https://medium.com/neuromation-io-blog/neuronuggets-understanding-human-poses-in-real-time-b73cb74b3818, April 2018. Accessed: 2019-01-15.

[62] Ismoilov Nusrat and Sung-Bong Jang. A comparison of regularization techniques in deep neural networks. *Symmetry*, 10:648, 11 2018.

[63] Christopher Olah. Neural networks, types, and functional programming. http://colah.github.io/posts/2015-09-NN-Types-FP/, September 2015. Accessed: 2019-01-15.

[64] Christopher Olah. Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/, August 2015. Accessed: 2019-01-15.

[65] Kevin Patrick Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning.* PhD thesis, 01 2002.

[66] Sayak Paul. Hyperparameter optimization in machine learning models. https://www.datacamp.com/community/tutorials/parameter-optimization-machine-learning-models, 08 2018. Accessed: 2019-01-10.

[67] Roelof Pieters. Deep learning for nlp: An introduction to neural word embeddings, lecture note, kth royal institute of technology. 12 2014.

[68] Ronald Poppe. A survey on vision-based human action recognition. *Image and Vision Computing*, 2010.

[69] Sebastian Raschka. *Python Machine Learning.* Packt Publishing, September 2015.

[70] Siraj Raval. Backpropagation in 5 minutes. https://www.youtube.com/watch?v=q555kfIFUCM, 2017. Accessed: 2019-01-15.

[71] S.J. Russell, S.J. Russell, P. Norvig, and E. Davis. *Artificial Intelligence: A Modern Approach.* Prentice Hall series in artificial intelligence. Prentice Hall, 2010.

[72] S. M. Safdarnejad, X. Liu, L. Udpa, B. Andrus, J. Wood, and D. Craven. Sports videos in the wild (svw): A video dataset for sports analysis. In *2015 11th IEEE International Conference and Workshops on Automatic Face and Gesture Recognition (FG)*, volume 1, pages 1–7, May 2015.

[73] Samim. Human pose detection. mining body language from videos. https://medium.com/@samim/human-pose-detection-51268e95ddc2, May 2017. Accessed: 2019-01-15.

[74] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local svm approach. In *Proceedings - International Conference on Pattern Recognition*, volume 3, pages 32 – 36 Vol.3, 09 2004.

[75] R. Shanmugamani. *Deep Learning for Computer Vision: Expert Techniques to Train Advanced Neural Networks Using TensorFlow and Keras.* Packt Publishing, January 2018.

[76] J. Shi and C. Tomasi. Good features to trac. 1994.

[77] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. *Advances in Neural Information Processing Systems*, 1, 06 2014.

[78] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[79] Leslie N. Smith. Cyclical learning rates for training neural networks. 04 2017.

[80] Ale Solano. Human pose estimation using openpose with tensorflow (part 1). url:https://arvrjourney.com/human-pose-estimation-using-openpose-with-tensorflow-part-1-7dd4ca5c8027, October 2017. Accessed: 2019-01-15.

[81] Ale Solano. Human pose estimation using openpose with tensorflow (part 2). https://arvrjourney.com/human-pose-estimation-using-openpose-with-tensorflow-part-2-e78ab9104fc8, November 2017. Accessed: 2019-01-15.

[82] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. UCF101: A dataset of 101 human actions classes from videos in the wild. *CoRR*, abs/1212.0402, 2012. official site:.

[83] Richard Souvenir and Justin Babbs. Learning the viewpoint manifold for action recognition. In *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 06 2008.

[84] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[85] Christos Stergiou and Dimitrios Siganos. Neural networks. https://www.doc.ic.ac.uk/~nd/surprise_96/journal/vol4/cs11/report.html#Feed\discretionary{-}{}{}forward%20networks, September 2015.

[86] Alexander Toshev and Christian Szegedy. Deeppose: Human pose estimation via deep neural networks. *CoRR*, abs/1312.4659, 2013.

[87] Du Tran, Jamie Ray, Zheng Shou, Shih-Fu Chang, and Manohar Paluri. Convnet architecture search for spatiotemporal feature learning. 08 2017.

[88] Avinash Sharma V. Understanding activation functions in neural networks. https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0, 2017. Accessed: 2019-01-15.

[89] Raviteja Vemulapalli, Felipe Arrate, and Rama Chellappa. Human action recognition by representing 3d human skeletons as points in a lie group. 06 2014.

[90] Stefan Wager, Sida Wang, and Percy Liang. Dropout training as adaptive regularization. *Advances in Neural Information Processing Systems*, 07 2013.

[91] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. pages 3551–3558, 12 2013.

[92] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. *ECCV*, 08 2016.

[93] Ying Wang, Kaiqi Huang, and Tieniu Tan. Human activity recognition based on r transform. In *Proceedings / CVPR, IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 06 2007.

[94] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. 04 2015.

[95] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. 04 2015.

[96] L. Xia, C. Chen, and J. K. Aggarwal. View invariant human action recognition using histograms of 3d joints. In *2012 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pages 20–27, June 2012.

[97] Chenliang Xu, Richard F. Doell, Stephen Hanson, Catherine Hanson, and Jason Corso. A study of actor and action semantic retention in video supervoxel segmentation. *International Journal of Semantic Computing*, 07, 11 2013.

[98] Shi Yan. Understanding lstm and its diagrams. https://medium.com/mlreview/understanding-lstm-and-its-diagrams-37e2f46f1714, March 2016. Accessed: 2019-01-15.

[99] Alper Yilmaz and Mubarak Shahb. A differential geometric approach to representing the human actions. In *Computer Vision and Image Understanding (CVIU)*, volume 109, pages 335–351, 03 2008.

[100] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. *Cornell Univ. Lab.*, 03 2015.

[101] Giancarlo Zaccone, Rezaul Karim, and Ahmed Menshawy. *Deep Learning with TensorFlow: Explore neural networks with Python.* Packt Publishing, April 2017.

[102] Wentao Zhu, Cuiling Lan, Junliang Xing, Wenjun Zeng, Yanghao Li, Li Shen, and Xiaohui Xie. Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. *CoRR*, abs/1603.07772, 2016.