

FACULTY OF ENGINEERING OF UNIVERSITY OF PORTO



Intelligent System for Live Skeleton-based Action Recognition

Luis Filipe Pinheiro Gonçalves Ribeiro

DISSERTATION

MASTERS IN ELECTROTECHNIC AND COMPUTERS ENGINEERING

Supervisor: Gil Manuel Gonçalves

Co-Supervisor: Liliana Antão

25 of February of 2021

Abstract

With the development of image classification and the increasing need for intelligent automation systems opens doors to new classification challenges such as human action recognition. With this challenge, more complete systems are being developed to increase the accuracy and modality representation of action samples. With this in mind, the purpose of our thesis is to create an intelligent system for live skeleton-based action classification.

To achieve this, extensive research was carried out on the current state-of-the-art approaches regarding skeleton-based models. The proposed method is a two-stream CNN-based model and includes attention mechanisms and Regularization layers such as Dropout 2D and Batch normalization. We used the RGB videos provided by the NTU-RGB+D 60 dataset, from which we extracted 2D skeleton poses. From these skeletons, we created geometric and temporal features decoupling the action samples into two streams. We propose an extensive preprocess phase to improve the dataset and solve various problems regarding the skeleton extraction method or the action length variability.

Furthermore, we performed live tests with the final model and concluded the best actions to be recognized, either when performed alone or in combination with other actions. Our final model was able to reach state-of-the-art performance, achieving 85.99% accuracy in cross-view and 82.81% in cross-subject NTU-RGB+D60 benchmarks.

Contents

1	Introduction	1
1.1	Context	2
1.2	Motivation	3
1.3	Problem Definition	3
1.4	Goals	4
1.5	Thesis Structure	5
2	Literature Review	7
2.1	Machine Learning	8
2.1.1	Artificial Neural Networks	9
2.1.2	Recurrent Neural Networks	10
2.1.3	Convolution Neural Networks	12
2.1.4	Others common mechanisms and layers	14
2.1.5	Evaluation Metrics	15
2.2	Related Work	18
2.2.1	Datasets	19
2.2.2	Machine Learning in Human Action Recognition	22
3	Action recognition Framework	33
3.1	First Approach	34
3.1.1	Dataset Preparation	34
3.1.2	Feature extraction	38
3.1.3	Action Recognition Model	42
3.2	Second Approach	43
3.2.1	Dataset Preparation	43
3.2.2	Feature Extraction	51
3.2.3	Action Recognition Model	51
4	Experiments and Results	55
4.1	Data Acquisition	56
4.2	Offline Action Recognition	58
4.2.1	First Approach	58
4.2.2	Second Approach	62
4.3	Live Action Recognition	75
4.3.1	Setup	75
4.3.2	First Tests	76
4.3.3	Combination Tests	79

5	Conclusions and Future work	83
5.1	Conclusions	84
5.2	Future Work	85
	References	87

Abbreviations and Symbols

Acc	Accuracy
AI	Artificial Intelligence
ANN	Artificial Neural Network
AVI	Audio Video Interleave
CDF	Cumulative Distribution Function
CNN	Convolution Neural Network
CV	Computer Vision
DL	Deep Learning
GPU	Graphics Processing Unit
HAR	Human Action Recognition
ID	Identification number
JJD	Joint to Joint Distance
JJO	Joint to Joint Orientation
JLD	Joint to Line Distance
JOD	Joint to Joint Orientation and Distance
eLU	Exponential Linear Unit
LLA	Line to Line Angle
LSTM	Long Short-Term Memory networks
ML	Machine Learning
OD	Object Detection
RNN	Recurrent Neural Network
PCA	Principal Component Analysis
PE	Pose Estimation
POS	Positional coordinates feature
ReLU	Rectified Linear Unit
SMOTE	Synthetic Minority Over-sampling Technique

List of Tables

2.1	Comparison between popular human action recognition datasets	19
2.2	Action IDs	22
2.3	Results comparison of Deep Learning models for Human Action Recognition . .	31
3.1	Feature dimensions	40
3.2	Feature Dimensions Table	53
4.1	Setup Heights and Distances	56
4.2	Performance test with different number of convolutions blocks	68
4.3	Convolution blocks outputs	68
4.4	Series of tests, regarding the last max pooling layer, the inclusion of an LSTM layer, an attention mechanism and a double convolution layer	69
4.5	Feature Training	70
4.6	Stream Convolution Blocks optimization	70
4.7	Last max pooling layer kernel size optimization	71
4.8	Features Comparison	71
4.9	Core CNN Optimization	72
4.10	Number of Convolution Blocks, second test	73
4.11	Dropout Inclusion Test and Number of Convolution Blocks corrected	73
4.12	Convolution layer hidden size optimization	74
4.13	Lack of Attention mechanism and different Evaluating Benchmark results	74
4.14	Models Performance	75
4.15	Setup Configuration Accuracy	76
4.16	Single Action Table	78
4.17	Comparison of F1-Score for Combo and single tests on the selected actions . . .	79
4.18	Combo Videos Frame Accuracy	80

List of Figures

2.1	Artificial Neuron Representation. Image inspired in [1]	9
2.2	Schematic diagram of a RNN. Figure adopted from [2]	11
2.3	Inside a RNN Cell. Figure adopted from [2]	11
2.4	Inside a LSTM Cell (Figure adopted from [2])	12
2.5	Convolution layer and Pooling layer example when applied to a 5x5 input	13
2.6	Learning Graph	15
2.7	Confusion Matrix example	16
2.8	Bars Metric	17
2.9	Scatter Metric	18
2.10	3D Weight Visualization	18
2.11	Illustration of the architecture of STA-LSTM. Figure adopted from [3]	23
2.12	Illustration of the architecture of VA-LSTM. Figure adopted from [4]	25
2.13	Illustration of the architecture of MS-AAGCN [5]	26
2.14	Illustrations of the architecture of DSTA-Net [6]	28
2.15	Illustration of the architecture of Part-Aware LSTM [7]	28
2.16	Illustration of the architecture of CNN+Motion+Trans [8]	30
3.1	Body Model 25	34
3.2	Raw data returned from OpenPose when applied to NTU-RGB+D dataset	35
3.3	Setup 1 Order	36
3.4	Zero Joint Statistics	37
3.5	Example of a sequence with zeros in joints, mid action	37
3.6	After Process Zero Joint Statistics	38
3.7	Line Set	39
3.8	Raw Database	41
3.9	Features Databases	42
3.10	Initial LSTM based model	43
3.11	Interpolation 3D. Action representation before(left) and after(right) of the interpolation	45
3.12	Two-Person Process	46
3.13	Lines Synthetic Generation	48
3.14	Skeleton Synthetic Generation	49
3.15	Full Subject Frame Distribution	50
3.16	Train-Test Split Subject Method	50
3.17	Initial CNN model illustration	52
3.18	Final model illustration	52
3.19	Our Model Blocks Illustration	53

4.1	Reading Algorithm	57
4.2	Own train test split method	59
4.3	First Learning Graph	59
4.4	Distance Test Table	61
4.5	0° Angle Distance Tests	61
4.6	45° Angle Distance Tests	62
4.7	Learning Graph of the initial LSTM model (left) and respective confusion matrix (right)	63
4.8	Frame samples per class distribution	63
4.9	Results with Weighted Loss Function Implementation	64
4.10	Results of the LSTM model after the dataset suffered Resampling Techniques, with a Learning Graph (on the left) and a confusion matrix (on the right)	64
4.11	2 and 4 class Training Learning Graph	65
4.12	2 and 4 class Training Confusion Matrix	66
4.13	Action 1 and 2 Training Results	67
4.14	CNN Training Results When fed with all 49 actions	67
4.15	Combined Learning Graph of tests presented in Table 4.9	73
4.16	Model Evolution Plot, LSTM(orange) and CNN(blue) based models accuracy chronologically and w.r.t the number of actions fed, either 2(triangle) or 49 (circle)	75
4.17	Room Setup	76
4.18	Single test confusion matrix	77
4.19	Action 15 JOD Featured Image example	82

Chapter 1

Introduction

1.1 Context

If machines could automatically interpret the activities people perform in everyday life, many tasks would be revolutionized. Given this, recognizing human activities (Human Activity Recognition - HAR) has been a significant research subject in the Computer Vision (CV) field and actively researched since the early 1980s. The ultimate goal of this research area is to create an automatic and intelligent system that can accurately perceive what type of activity is being performed by a person or group of people. Resembling the human vision system, the system created should generate a label after observing the human activity execution.

Human activities are inherently complex; however, it is possible to decompose in four different types: gestures, actions, interactions, and group activities. As the title suggests, this thesis focus on human actions for recognition. Therefore, recognizing gestures, interactions, and group activities is not covered under the scope of this thesis. Broadly each type of activity can be defined as follows:

- Gestures are a basic movement of a specific body part describing the meaningful motion. Usually, a gesture is very short and has low complexity. For example, "raising a thumb" and "shaking the head".
- Actions are activities executed by a single person that may be decomposed in multiple gestures arranged in a temporal order. Depending on the application, an action can be either defined as a single activity that is performed by a single person or as a combination of multiple atomic activities also executed by a single person. For example, "running", "up-and-go", "punching" and "drinking".
- Interactions are a type of activity performed by two actors (one human, another human, or object). For example, "hand-shaking," "hugging," or "using the computer".
- Group activities are the most complex type of activity, as it involves more than one person and allows for the combination of gestures and actions.

In HAR problems, several types of input data can be used. These data can be retrieved from different sensors like vision devices that collect RGB video frames or skeleton data, or even wearables like accelerometers that provide rotation and translation data. Contrary to this variety in input types, most HAR solutions make use of data-driven intelligent computer algorithms in the tandem of Machine Learning (ML) and Deep Learning (DL) models.

In the past, predicting and classifying human activities was mainly solved with traditional ML models, showing overall good results. Most recently, with the application of DL, these results have been significantly increased in accuracy by overcoming a lot of the constraints faced when using traditional ML methods. With DL, researchers have been able to achieve a better data-driven approach for learning differentiating features, resulting in better performances and scalability for more complex behavior recognition tasks. Given this, there is an increasing trend towards the implementation of DL algorithms in HAR. In this thesis, the challenges mentioned above will be tackled with the use of DL models.

1.2 Motivation

As mentioned above, HAR has been a overall hot topic in CV research. But in recent years, most of the research and industry efforts have been focused in one specific type of HAR: action recognition solutions, due to their use in a wide range of high-level applications, in real-world scenarios. This is a great motivating factor for this thesis, as the study and development of a solution in the context of action recognition can impact several different areas of application. Some examples of areas that could benefit from a flexible and reliable human action recognition system include, but are not limited to the following:

- **Automated Video Surveillance:** instead of the traditional security surveillance systems that require human supervision for video content analysis, an intelligent surveillance system that recognizes human activities can be highly beneficial. This includes the detection of suspicious activities, preventing dangerous situations.
- **Health Monitoring:** with action recognition one can create health monitoring and assisted living solutions to seamlessly track and understand humans within their own environment. For instance, the system can automatically recognize their actions and detect any discrepancy from their usual behaviors, such as falling down or tripping, calling medical assistance if needed.
- **Human-Robot Interaction:** by giving a robot the perception of what the human is doing, it can become more useful both in industrial setups and in domestic environments. With this information the planning of the robot's behavior can be adjusted accordingly, increasing efficiency of interaction or reducing stopping times.
- **Assisted Driving:** in this case human activity recognition can be used to alert the driver when dangerous actions are being performed while driving, for instance drinking or using the phone.

Inspired by this, motivation is provided to develop robust and cross-cutting solutions for human action recognition, that can be directly used or easily adapted to these different applications and their environments. By providing computer vision solutions with the ability to learn to automatically detect and predict actions in a similar manner to humans, computer vision as a whole is also empowered, as their solutions become much more intelligent and usable in different scopes.

1.3 Problem Definition

During the past decades, many solutions have been proposed for human action recognition. Despite the difference in requirements for each specific application, the type of input data mostly used is the same to all: articulated human joint coordinates, also known as skeleton data. Skeleton data have the distinctiveness of being more robust to environment noise when compared to accelerometer data while also providing direct knowledge and high-level features of human actions and their movements. Furthermore, skeletons are usually smaller in size when compared to RGB frames, allowing for models that are more simple and compact in complexity and hardware [9].

Given this, scientists usually opt for a skeleton-based solution in this field. Implementing these solutions can be a very complex problem due to many challenges like the dependency of human motion and shape on many factors, resulting in a large diversity of ways to perform the same action. Also, environmental settings like lighting conditions, camera settings, and fixed or dynamic camera positions can directly impact these types of solutions.

Although skeleton-based action recognition has been widely used and significantly improved over the last few years, developers are still gathering research to know what types of architectures might be best for this problem. There are already some skeleton-based solutions able to cover a limited set of movement types in some very confined environmental settings (fixed cameras and controlled lightning conditions), making their usage very problem-specific. Not only that, but most of the present recognition systems focus, mainly, are not ready for practical application. This last fact is primarily due to very accurate models that have very narrow and simple action sets, or are very slow in giving a response, as they focus on offline testing and performance. This area demands real-time classifications, where new studies should focus on extracting more information with less computational demand. Given this, there is a need for systems able to automatically detect complex human actions, analyzing human motion from skeleton information acquired from chosen robust sensors in real-time.

1.4 Goals

Given the problem defined above in this project, we aim to explore and develop an intelligent system that allows the automatic detection and prediction of human actions based on human skeleton data acquired from a vision camera in real-time. This intelligent system should be thought out as an upgrade for vision systems, providing its application with real-time information of the action being performed by the humans present in the monitored environment.

First, a general model using a dataset of pre-recorded videos should be developed. This general model should be view, subject, and illumination independent, as well as have a wide variance of actions and setups. The model should be capable of classifying single-person actions only. Using the model as a base, a real-time framework for action classification will be developed.

In this framework, the data will be captured and fed to the model, responding with an action prediction in real-time. A stereoscopic Camera (ZED) will be used to capture data, then applying OpenPose to the camera feed to extract skeleton positions. Afterward, the frameworks robustness and transversality will be explored by using different distances, lighting conditions and combining several actions as one complex activity for classification. Finally, the system's ability to predict the actions (correctly classify them before they end) will also be evaluated.

In sum, we have the following goals in this project:

1. Carry out extensive research of the current state-of-the-art DL approaches to skeleton-based human action recognition;
2. Create a model capable of accurately classify actions performed by one subject;

3. Develop a framework to incorporate data acquisition, feature extraction, and human action classification, working in real-time;
4. Explore the framework's robustness to different action combinations, camera setups, and lighting conditions;
5. Evaluate the model's ability to predict actions in relevant time correctly;

1.5 Thesis Structure

The rest of this document is divided into four more chapters.

In Chapter 2, the main goal is to give some background information regarding several ML and DL concepts relevant for this thesis, as well as some basic architectures commonly used in action recognition (Section 2.1), as well as present an overview of the current state-of-the-art in the scope of skeleton-based human action recognition (Section 2.2). In this overview, first, a presentation of data acquisition methods and most common datasets is provided as well as an overview of evaluation metrics used to compare models and then a summary of several relevant research works solving our problem is given.

In Chapter 3, the development stages of this thesis are presented. This Chapter is divided into two parts: the first regarding a more naive and initial approach (Section 3.1), and the second a more systematic approach (Section 3.2). In both sections, all development stages are explained, from data understanding and preparation, to feature extraction and model creation.

In Chapter 4, the various experiments performed and derived results are presented. These tests are divided in three sections: data acquisition (Section 4.1), offline tests (Section 4.2) and live tests (Section 4.3). In each one, a detailed discussion of their meaning and results is given.

Finally, in Chapter 5 the final remarks of the work done, the model limitations and the problems found are presented, as well as future work and possible ways to overcome some limitations.

Chapter 2

Literature Review

2.1 Machine Learning

As stated previously in Section 1.1, the majority of action recognition solutions make use of Machine Learning (ML) and Deep Learning (DL) models. ML refers to the automatic detection of meaningful patterns in data. In contrast to the typical applications where a script is followed to perform some task, ML tries to give the application the ability to learn, similarly to humans who acquire skills through some kind of learning. This learning is done through training the models on some sample data and based on the generalizations formulated from that data, the model will make decisions. Some of the popular ML algorithms for classification include Support Vector Machines and Artificial Neural Networks (ANNs). The goal is common to all and consists of finding particular relationships in the input data that helps the model to distinguish the output correctly. Today, ML powers many aspects of modern society, being used in applications ranging from object identification to speech recognition and select relevant results of online searches.

ML can be divided into three main groups based on their purpose: Supervised learning, Unsupervised learning, and Reinforcement learning, where this last one is not very relevant in the context of our problem. The most common form of ML is supervised learning. This form of learning is characterized by receiving labeled data, referred to as "training data", producing an output vector of scores for each label. The insight is that the true label should have the highest score from all labels. This is done by computing a loss function that measures the error between the output scores and the desired scores. The machine then modifies and adjusts its internal parameters (weights) to reduce this error. Unsupervised learning, on the other hand, feeds the model with unlabeled data, from where the model will try to find discriminative characteristic features in the unlabeled input data and tries to divide the dataset into groups. Some common examples are anomaly detection, where outliers from a distribution are detected, Association mining, where samples that occur more frequently are grouped, and clustering which groups the data accordingly to data similarities.

For decades, ML methods revealed great results derived from creative feature extraction processes that extracted insightful and well-represented information from raw data, which enabled the learning system to classify the input correctly. On the other hand, when it came to processing data in the raw form, these types of models were very limited. This is where DL, as a subset of ML, comes in. Nowadays, with DL, ML systems automatically find the features needed for classification directly from the raw data. DL algorithms can be regarded as an evolution of ML algorithms in complexity and computational power. This complexity of the model used should be held into account. Even if the model complexity depends on the nature of the training data, one should choose the lowest model complexity that allows solving the problem, as with high complexity comes a greater chance of overfitting. Overfitting is when the model becomes biased to the training set. The learning stops going in the direction of finding the decision boundaries that distinguish the classes for unseen data and start to fit the training data. This might be visible with growth in the loss, or the fall of accuracy on the test data after a peak of accuracy or an inverted peak of loss has occurred.

DL methods have several levels acquired by forming simple modules, where each transforms the representation of the data, starting with the raw inputs, into features in a higher abstract level. In classification tasks, these levels highlight important characteristics of the input and inhibit unimportant ones. The key factor of DL is that these layers are not developed by engineers but rather directly learned from data. DL has been successfully used in many applications, from image recognition and speech recognition to predicting the effects of DNA mutations and sentiment analysis. DL describes algorithms that equivalently examine data to how a human draws inferences. To accomplish this, DL applications use a layered structure of algorithms, i. e., ANNs.

With this brief introduction to ML, DL, and their properties, in the next subsections, key aspects relevant to their rise and evaluation will be explained in further detail, namely ANNs and two different types of ANN architectures used as DL models: Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), as well as other relevant mechanisms/layers, and evaluation metrics.

2.1.1 Artificial Neural Networks

A small introduction to Artificial Neural Networks (ANN) and an overview of its characteristics are briefly presented here. The lowest level of most ANNs is artificial neuron known as "perceptron"(Fig. 2.1).

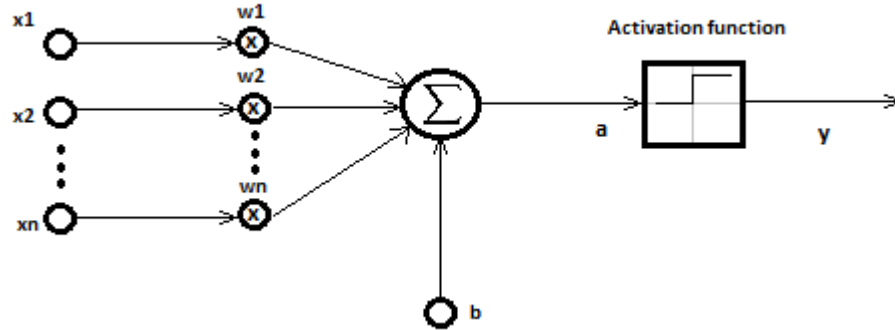


Figure 2.1: Artificial Neuron Representation. Image inspired in [1]

The neuron receives N inputs ($\mathbf{x}_i$) each multiplied by a weight ($\mathbf{w}_i$), which represents the significance of this input in this neuron, and sum all the products and a biased vector ($\mathbf{b}$), resulting in the activation $\mathbf{a}$ (Eq. 2.1), which is passed through an activation function to become the output ($\mathbf{y}$).

$$a = \sum_{i=1}^n x_i \times w_i + b \quad (2.1)$$

The activation functions are chosen accordingly to the nature of the data and are mostly used to comprehend the output values in a certain range. The most common ranges in classification problems are $[0,1]$ (Rectified linear activation function - Relu), and $[-1,1]$ (hyperbolic tangent - tahn). An interesting and commonly used activation function in multi-classification problems is

the softmax function, limiting the output to a range of 0 to 1 and the sum of all outputs for a given sample to 1, which gives an amicable understanding of the values.

The usual ANN is composed of interconnected layers of neurons. Having an **input layer** in the beginning, an **output layer** in the end, and middle layers called **hidden layers**. The input layers usually have a number of neurons equal to the number of different features in the dataset. In contrast, the hidden layers usually have a higher number of neurons so that they can extract more complex information. In the output layer, normally, the number of neurons is equal to the number of the classes in the dataset in order to give a value for each class and from there infer the classified action.

As the training data is fed through the ANN, a cost/loss function is used to calculate the error between the actual and the expected output. As explained previously for supervised learning, the goal of an ANN is to minimize this error and to do so, the weights and the biased values are changed accordingly during training. These changes are commanded by the optimization algorithm used and by the network parameters that define the several types of architectures. Moreover, the links between neurons can be bidirectional, and all of this leads to the vast amount of different ANN architectures.

In the next subsections, we will go through the two most popular ANN architectures in the topic of human action classification.

2.1.2 Recurrent Neural Networks

A Recurrent Neural Network (RNN) is a neural network architecture that includes cyclical connections, which enable learning temporal dynamics of sequential data. It is called recurrent because it keeps the final output and can be understood as a neural network with memory. This means that it keeps sequential information in contrast to the regular ANN. Most of its application resides in time series data, text, and audio predictions.

As seen in Fig. 2.2, the output h_0 is achieved from X_0 through the 'A' block. This block takes as input the machine input X and the previous state h_{t-1} , then a weighted element-wise sum is preformed and passed through a tahn activation layer. The weights, used in the sum, are the variables that will be optimizing during learning. Then h_0 is combined with the input X_1 to achieve h_1 . One of the disadvantages of these types of networks is that it may suffer from vanishing or exploding gradients for a long time sequence. This problem regards the weight modifications, as referred, on each training batch end, the weights along the layers will be updated, and this update difference is based on the gradients.

The gradients are a representation of the loss w.r.t. that weight, meaning that when we have gradients greater than 1 the update will increase, or less than 1, where the update will decrease. If the gradients keeps the same relation to 1 as before, the updates will continue to grow (explode) or to decrease (vanishing), to a certain point where the weights are being update in such a high or low scale that disables the model from learning correctly. Fig. 2.2 also shows that a RNN can be viewed as multiple copies of the same network, each passing a message to a successor. At the

beginning of the training, some parameters and variables are initialized with some values, as it is the case for the bias parameters and the initial step to form h_0 where no previous state exists.

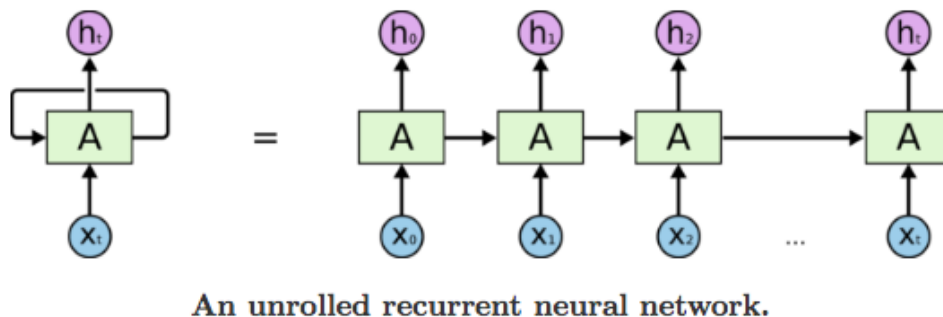


Figure 2.2: Schematic diagram of a RNN. Figure adopted from [2]

In Fig. 2.3 we see that a simple RNN cell concatenates the input with the final output, feeds it to an activation layer, a tanh activation, and then passes one copy of the output to the next cell and another copy as the cell output. These networks are beneficial for short-term dependencies but have some problems with long-term dependencies. As an example, as they are broadly used in text recognition, it would be easier to predict the word "water" in the sentence "The fishes swim in the ___" than it would be to predict the word "Italian" in the sentence "I went to Italy ... I learned how to speak ___" since the words related to water are closer to the word to fill than the "Italy" word is to the word to fill.

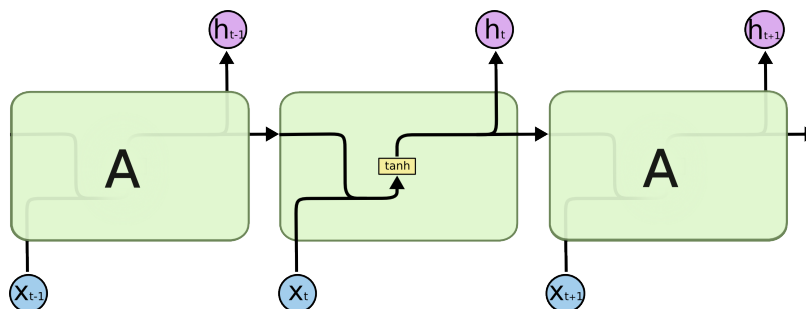


Figure 2.3: Inside a RNN Cell. Figure adopted from [2]

A very effective solution to the vanishing gradient problem is provided by Long Short-Term Memory networks (LSTMs). LSTMs are an evolved version of the RNN architecture, capable of learning long term dependencies. In Fig. 2.4 we can see that the 'A' block, instead of just having one gate, in LSTMs they have an Input gate, a Forget gate, and an Output gate. Besides that, instead of just having an output and input, this network also has a cell state shared by all cells. The purpose of the forget gate is to choose which information should be forgotten from the cell state. The Input gate has two parts. Similarly to the forget gate, it has a sigmoid function applied to the input to decide which values are to be updated and which are to be modified, and a second

part where a tanh function is used to rescale the information so that the data added is never higher or lower than 1 or -1, respectively. Both these parts are combined and added to the cell state.

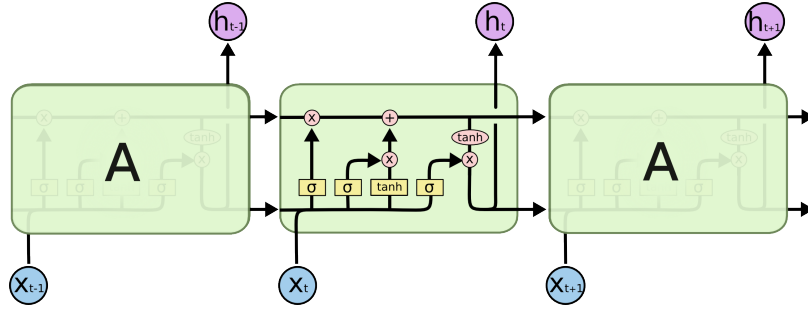


Figure 2.4: Inside a LSTM Cell (Figure adopted from [2])

In both figures (Fig. 2.3 and Fig. 2.4), we can see four operation blocks: $\otimes$, denoting the element-wise multiplication, $\oplus$ the element-wise addition, σ the sigmoid function which denotes the values into a range from 0 to 1 (for example in the forget gate, the sigmoid (σ) is responsible for specifying how much from nothing (0) to all (1) of the information is to keep in the cell state), and the tanh function, which translates the values into a range of -1 to 1 so that the values do not explode with all the additions.

In sum, the LSTMs have two gates to regularize the information in the cell state which operate on both the cell input $\mathbf{X}$ and the previous cell output h_{t-1} , and a third gate, which besides operating on the cell input and the previous cell output, also takes into account the cell state, to generate the cell output h_t . Due to the forget gate, some information might be kept longer, which would enable the ability to learn long-term dependencies. Some information might be forgotten sooner and replaced with new information, which would also allow the ability to understand short term dependencies.

2.1.3 Convolution Neural Networks

A Convolution Neural Network (CNN) is a feed-forward neural network, most commonly use in CV for recognizing objects in an image, in contrast to a RNN, which is suitable for sequence data. Although CNNs can also be applied to 1D data, we will focus on the 2D case since it is the most used in this area. Mainly, CNNs assume that the input data are images. A convolution layer applies several filters in the image in order to extract relevant features from it.

Some of these filters might learn how to detect horizontal edges, vertical edges, or even some color variance patterns. Then the output is fed to a fully connected layer to give the last output. Convolution layers are known for their ability to learn to detect several features, which might be more local or more global, depending on if the filter size is smaller or bigger, respectively. Under normal specifications, as to not use padding after the convolution, the convolution layers infer some minor dimension reduction, however the major one is usually obtained using a pooling layer. A popular pooling layer is the max pooling, which will extract the maximum value of a given window, which can be viewed as a zoom-out operation.

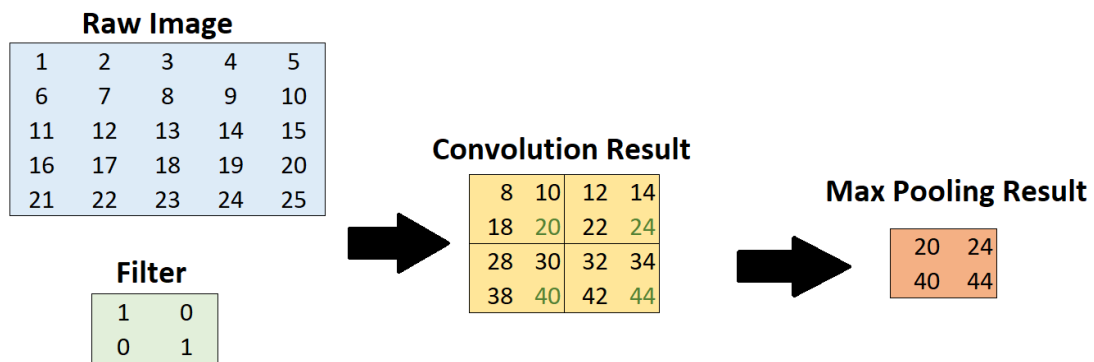


Figure 2.5: Convolution layer and Pooling layer example when applied to a 5x5 input

In Fig.2.5 we can see that a filter 2x2 was applied to an image 5x5, with 1x1 stride, resulting in a feature image 4x4, on top of which a Max Pooling layer was used, with a stride and pool size of 2x2, resulting in a 2x2 feature image.

The filters and pools are windows that will go through all the data extracting a single number as output per window, although with different extracting algorithms, and the strides define how many cells the window will move to the next convolution/pool, in the example, the convolution used stride 1x1, meaning that the convolution window will move in a maximum of one cell in the horizontal direction and one cell in the vertical after each convolution.

If we take a look at the evolution of the initial image to the final image shape, we see that the image suffered dimension reduction. This dimension reduction is mainly due to the stride size, for example if we have an 1D vector with length 6, and we use a 2 length window and count how many windows can we form accordingly to a certain stride, we can conclude that with stride = 1 we can have 5 windows, but if we use stride = 2, we only have 3 windows, while if with stride 1 but with filter = 3 we can have 4 windows. Due to this fact and due to by default the pooling layers having higher strides than the convolution layers, in CNN models the dimension reduction is usually obtained from the pooling layers, in comparison with the convolution ones.

In the pooling layer, a stride of 2x2 was used, and for each window to extract the maximum, the limits are identified as the divisions illustrated in the Convolution Result in Fig. 2.5, which since it has a 4x4 shape and the pool and stride shapes, in the pooling layer, are 2x2 result in 4 sub squares in the Convolution Result image, which when max pooled and put together, results in the 2x2 Max Pooling Result image. As we can see, the dimensional reduction is mainly from the max-pooling layer and depends on the pool, stride and input size.

In Fig.2.5 the convolution of only one filter is illustrated, but in a convolution layer, several filters might be used. These filters will learn to have different shapes, each one producing a different featured image. On top of each one, a max-pooling result might be applied. The smaller the filter size, the more local the feature extracted will be. On the other hand, the higher the filter size, the more global the feature extracted will be. Convolution layers can be built on top of each other, commonly interleaved with the pooling result. The higher on the CNN architecture, the more abstract will be the featured image, and for that reason, initially, local filters are used, and

more global filters are implemented in the end.

In the context of skeleton-based human action recognition, as the data used to represent the human pose is in its essence a graph, some solutions emerged combining graphs and CNNs, Graph Convolution Network(GCN). It works analogously to the CNN, but the input, instead of being a rectangular shape it might have any kind of shape. In fact, the GCN can be viewed as a generalization of a CNN, where the rectangular input of a CNN is a graph connected in a grid.

2.1.4 Others common mechanisms and layers

The attention mechanism appeared as an improvement over the neural machine translation system in natural language processing (NLP). Subsequently, this mechanism was then applied in other applications, including CV and speech processing. This mechanism can be summed up as a method that takes the input and learns to give attention to certain parts of that input. One common way to do so is to feed the input to a feed-forward network and, on top of that, a softmax layer. Finally, we apply an element-wise multiplication between the softmax output and the input, resulting in input with attention.

Typical applications of the attention mechanism predict the next word in a sentence, where the model needs to know to which word to give more attention. These cases will generally have an encoder and a decoder, and in between, there is the attention mechanism.

First, the encoded states, alongside the previous decoder state, are fed to a feed-forward, which will learn to identify which encoder states are more or less relevant depending on the values generated. These generated values are provided to a softmax layer, which will comprehend values from 0 to 1, useful to see the weight of each value generated, which are combined by an element-wise multiplication with the initial encoder states creating the context vector. After this, the context vector is concatenated with the last decoder output and fed to the decoder, which will predict the next word. The attention weights can be very useful to skeleton-based human action recognition to give and take the focus of some joints of the skeleton body.

A broadly used layer is the Fully-connected layer, which also is known as Dense layer. The fully-connected denomination comes from the fact that each input node is connected to all output nodes, where the output of each output node can be viewed as an weighted sum of all the input nodes. All the weights are different between different output nodes.

Dropout is also a very recruited layer which simulates a sparse activation from a given layer. In sum sets to 0 some units of the input, this process might be referred as dropping. A more advanced Dropout is Dropout 2D that instead of dropping units of the previous layer output it drops entire 2D feature maps. This type of layers have a parameter that indicates how many units or featured maps should be dropped, where this parameter is usually in a 0 to 1 scale. This layer is known to help prevent overfitting since it may introduces some noise due to the sparse activation mentioned.

Finally Batch Normalization layer is commonly used to regulate the model, in the sense that limits the range of the input in the output version. This normalization helps the model run faster, as it uses smaller numbers for calculations and helps regularize the model since it limits the values.

2.1.5 Evaluation Metrics

Evaluating ML models is an essential part of any project. The models that use the same dataset are comparable regarding the accuracy metric. Although there may be different accuracy metrics, multi-class classification problems usually resort to categorical accuracy or variation. It computes how often predictions match the expected labels. These expected labels might be represented as one-hot encoded vectors for a problem with few classes. In this one-hot encoded vector, a one is placed in the correct label index, and the rest of the vector is all zeros. This helps the computer make the calculations, but the memory demands grow with the number of classes to predict. On the other hand, the categorical sparse accuracy does the same computation. Still, instead of one-hot encoded vectors, an integer is passed, where the integer is the desired output index. In this topic a common categorical accuracy variance is Sparse Top K categorical accuracy, which tells the percentage of samples where the expected value is in the top k predictions.

The training is made in epochs. At the end of each epoch, the model is evaluated on test data and gives us feedback of the test and train accuracy and loss. Two plots are usually showed where in the X-axis the epochs are represented, and the values of the metrics in the Y-axis. One plot for each metric is typically the way to show these results since the values of accuracy and loss might have different natures values. To this type of graph we call it by Learning Graph or Learning Curves as illustrated in Fig. 2.6, where on the left side it's plotted the evolution of the accuracy metric along all epochs for both testing and training results, on the other side, similarly, it is plotted the loss value evolution through the epochs. We aim to increase test accuracy and reduce test loss. We also want the test results to go along with the training results as long as possible since the training accuracy usually tends to be higher and the loss value lower than the test correspondent metrics. In our Learning Graphs we will represent the test and train results with lines with orange and blue color respectively.

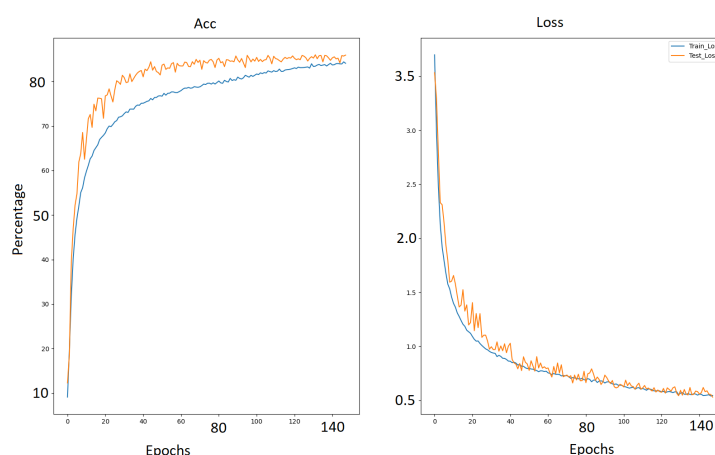


Figure 2.6: Learning Graph

A very common metric is a Confusion Matrix. It can show which classes were well classified, and where they were not, with which class it has been mostly mistaken for. This kind of infor-

mation is widely known as true and false positives, and analogously there are also true and false negatives. An example of a confusion matrix can be the Fig. 2.7, where a square grid image is shown. In this image both axis have all the classes to be predicted. Still, one is referent to the actual(expected) actions, and the other is referent to the predicted action. If we understand the cells, with coordinates of (predicted action, actual action), each cell tells us how many times, usually by color representation where the figure shall have a color bar, an action has been classified as some action, equal to the expected or not. From there, one can extract some information of the model, such as if the model is biased to one action, which can be identified as a big number of actual actions with the same predicted action, or one might analyze if the model can not distinguish between a group of actions, identified by high value cell symmetric to the diagonal, i.e., if for one expected action A and for a different predicted action B there is a high value cell, and if for the expected action B and for the predicted action A also have a high value cell, means that this two actions are badly distinguished, and frequently mistaken one by another. For these problems, usually, the answers rely on the balance of dataset, on an over padding side-effect or on the similarities between classes. The ideal confusion matrix would be a diagonal line filled with green, all surrounded by red squares.

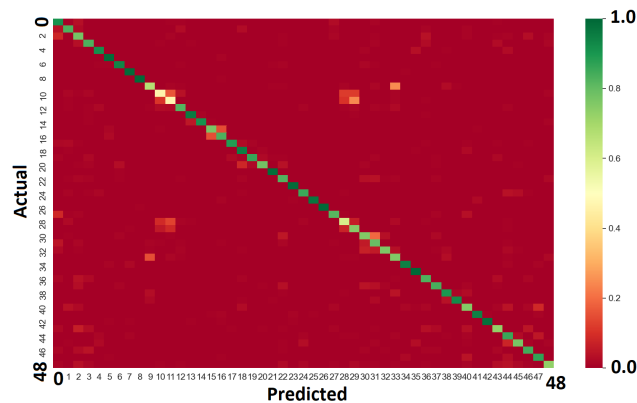


Figure 2.7: Confusion Matrix example

Regarding the same information, true and false positives and negatives, a table of metrics is also popular. In this table each row is a class and has four columns: 1) Precision, 2) Recall, 3) FN-score, 4) Support. Support is the number of samples per class, useful to detect unbalanced datasets. Precision usually measures how good the model is to classify an action when that same action is fed. On the other hand, Recall measures how good the model is in not classifying an action when that action is not fed. Finally, the FN-score is a weighted average of the Recall and Precision, and the N represents the coefficient that will configure the weights, if $N=1$, Recall, and Precision will have the same weight. This table of metrics values are easily acquired through `sklearn classification_report()` function by giving as arguments the expected output and the predicted output.

To understand how the model's confidence when classifying an action, we developed a simple

metric, where the last prediction for each action would have the weights, outputted by the softmax layer, plotted in a bar graph. The bar weights shall have different colors for the expected(blue), predicted(yellow), and the other irrelevant actions(red), and in case the expected action matching the predicted action, another color should be used(green). Although this metric not being a very reliable one, it would still give us some valuable information. An example can be found in Fig. 2.8, where ideally, in each axis, the respective x bin should have a green bar.

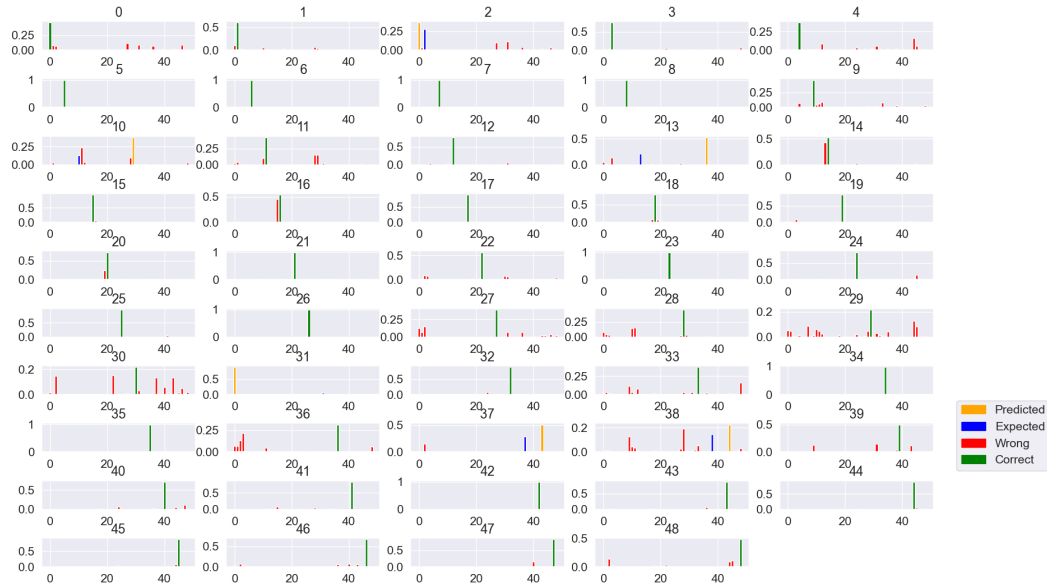


Figure 2.8: Bars Metric

Another simple metric we used was the scatter metric, which would plot a scatter plot where each point was defined by (sample number, predicted action). The color would depend on correctly classified(green) or wrongly classified(red) action. This kind of metric was mostly used initially due to a lack of knowledge in the programming language to implement the confusion matrix metric. Nevertheless, this metric might be useful for small datasets, not the case, but an example is illustrated in Fig. 2.9.

Finally, our last metric or analysis method we developed was the weight 3D visualization in one epoch. This technique aimed to develop our understanding of the weight manipulation inside the LSTM layers, trying to find out the reason for the unsatisfying performance. This weight visualization is composed of histograms plotted in each batch, generating a 3D image. There are a total of 12 grid positioned axes. The three columns are regarding the normal, recurrent, and the bias weights. On the other hand, the four rows are referent respectively to the Input, Forget, Cell, and Output gate.

Despite all the metrics used, in training, the model learns to minimize the loss function which in sum gives a value for the error between predicted and expected output. A common loss function used in this topic is the Sparse categorical cross-entropy, which returns a number that depends on the certainty and accuracy of the model, meaning that if class A was expected and class A was

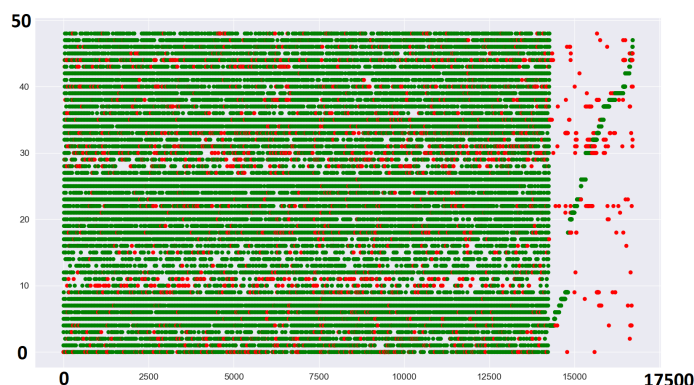


Figure 2.9: Scatter Metric

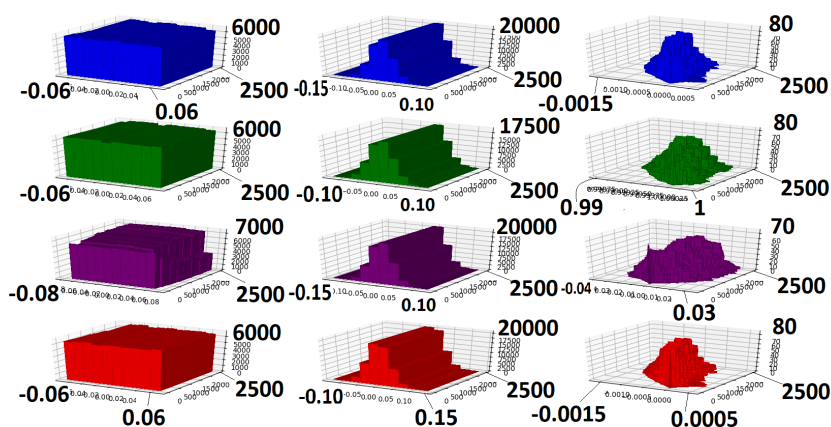


Figure 2.10: 3D Weight Visualization

predicted, our model is accurate. However, if the weights on class A are very close to class B, then although still accurate is not very certain. This loss function encodes the certainty and accuracy in a number that decreases with the increase of both characteristics.

2.2 Related Work

In this Section, there will be an overview of two essential aspects for solving the problem described in Section 1.3: choosing datasets for training and evaluating and models architectures to use. First, we will go over some of the datasets used in Human Action Recognition. The most accurate and exciting models will then be presented, taking notice of the type of dataset they used and the model architecture of each. All of the approaches are based on CNNs and LSTMs since they comprise the majority of the state-of-the-art in this field.

2.2.1 Datasets

An essential aspect of achieving human action recognition with DL is the dataset used. In a classification problem like HAR, a dataset is used for training and evaluating the specific model used, and the results obtained in each can perform the function of equally comparing competing approaches in the field. The best datasets need to have variability in their samples to not infer bias in the model. This variability can be video samples with different subjects and different angles and distances of the subject to the camera. They might also have video samples with occlusions or different lighting setups. All this variability works toward the robustness of the models trained with this kind of dataset.

However, with significant variability comes a large number of samples so that the model has enough time and data to learn the best decision boundaries and gain the robustness desired. Some datasets also have different modalities of their samples, meaning that they can supply RGB, depth, or Infra-Red videos. Some might even extract the skeletons directly and supply the dataset in skeleton sequence format. Some commonly used public datasets available to train and test action recognition methods are summed up in Table 2.1.

Dataset	Samples	Classes	Subjects	Views	Sensor	Modalities	Year
KTH [10]	600	6	25	1	RGB Camera	RGB	2005
Weizmann [11, 12]	90	10	9	1	RGB Camera	RGB	2005
IXMAS [13]	1148	12	7	5	RGB Cameras	RGB	2006
CAD-60 [14]	60	12	4	1	Kinect v1	RGB+D + 3DJoins	2011
RGBD-HuDaAct [15]	1189	13	30	1	Kinect v1	RGB+D	2011
MSR DailyActivity3D [16]	320	16	10	1	Kinect v1	RGB+D + 3DJoins	2012
DHG [17]	2800	14	20	2	Intel RealSense	D + 3DJoins	2017
SHREC [18]	2800	14	28	2	Intel RealSense	D + 3DJoins	2017
SBU Kinect Interaction [19]	300	8	7	1	Kinect v1	RGB+D + 3DJoins	2012
SYSU 3DHOI [20]	480	12	40	1	Kinect v1	RGB+D + 3DJoins	2017
NTU-60 RGB+D [7]	56 880	60	40	3	Kinect v2	RGB+D + IR + 3DJoins	2016
NTU-120 RGB+D [21]	114 480	120	106	3	Kinect v1	RGB+D + IR + 3DJoins	2019

Table 2.1: Comparison between popular human action recognition datasets

The KTH human motion dataset, mentioned by Roth *et al.* [10], contains six actions (walking, jogging, running, boxing, hand waving, and hand clapping) performed by 25 different actors. Four different scenarios are used regarding indoor and outdoor scenarios, and their RGB videos are provided. The Weizmann human action dataset, present by Gorelick *et al.* [11, 12] also contain RGB videos of ten different actions, performed by only nine subjects. It includes some actions outdoors like walking, raising arms, running, or jumping.

The IXMAS multi-view dataset, described by Weinland *et al.* [13], contains actions captured from five different viewpoints (five different Firewire cameras). A total of 11 subjects performed 14 actions (check watch, cross arms, scratch head, sit down, get up, turn around, walk, wave, punch, kick, point, pick up, throw overhead and throw from bottom-up), where some actions

are partly occluded. The actions are performed in an arbitrary direction concerning the camera setup. The camera views are fixed, with a static background and illumination settings. The RGBD-HuDaAct dataset, presented by Ni *et al.* [15], has a larger scale of samples and contains synchronized color depth video streams. Also, there are rich intra-class variations in the dataset, such as standing still, sitting, or exchanging the left and right hand. The background is limited to a single lab environment with a fixed camera. However, none of the datasets mentioned above provide the corresponding skeleton information.

The MSR Daily Activity 3D dataset, present by Tamou *et al.* [16] contains 16 daily activities recorded by ten different subjects. Specifically, two of the activities (sitting and standing) involve interactions between subjects and environmental settings such as a sofa. The RGB and depth channels are recorded separately and therefore not strictly synchronized. The videos are also captured with a single fixed-angle camera. The CAD-60 (Cornell activity dataset), introduced by Sung *et al.* [14] also provides RGB-D images and skeleton data (coordinates of joint positions and orientations) captured in five different natural scenes. The data set contains 12 categories of daily activities performed by four subjects, but the number of samples in each category is moderately imbalanced.

The SHREC 2017 dataset [18], contains 14 hand gestures performed whether with one finger or the whole hand, where each one is repeated 1 to 10 times by 28 participants resulting in 2800 sequences. Furthermore, all the participants were right-handed, and the sequences are labeled following their gesture, the number of fingers used, the performer, and the trial. Per each frame, it contains a depth image, 22 joints' 3D coordinates in the world space, and 2D coordinates in the depth image space. An Intel RealSense short-range depth camera was used to collect the dataset at a 30 fps capture rate. DHG was created and introduced in SHREC2017 - 3D Shape Retrieval Contest [17], being very similar to SHREC dataset. The main difference is that they only used 20 subjects, and each gesture was repeated a fixed number of five times.

The SBU Kinect Interaction, presented by Yun *et al.* [19], has eight action classes, performed by seven participants in 21 sets, resulting in approximately 300 samples. Most of the actions performed in this dataset involve interactions between two subjects. A color image, a depth map, and extracted skeletons are provided for each frame. In the SYSU dataset, described by Hu *et al.* [20] there are 12 different actions performed by 40 participants. The activities consist of a participant manipulating one of six other objects: phone, chair, bag, wallet, mop, and besom. This results in 480 video samples with different durations in the range from 1.9s to 21s. The video samples were captured using a Kinect camera, which enabled the capture of the RGB frames, depth sequence, and skeleton data.

Finally, the NTU-RGB+D Action dataset is was by far one of the most significant in terms of number of samples, classes, and subjects, which translates in a better fit for applying DL and possibly providing more robustness to a HAR model. Also, it is one of the most used datasets in literature, as well as one of the most significant in respect to action recognition (according to the definition provided in Section 1.1). Given all these aspects, NTU-RGD+D was the one selected for this thesis work, and therefore more details will be provided for this specific dataset and its

versions in the following subsection.

2.2.1.1 NTU-RGB+D Dataset

NTU-RGB+D Action Dataset can be divided in two different versions: NTU-RGB+D 60 [7], and NTU-RGB+D 120 [21]. Generally, this dataset collected four primary data modalities captured by Microsoft Kinect v2 sensors: depth maps, 3D joint information, RGB frames, and IR sequences. The Joint information consists of 3D coordinates of 25 body joints for tracked human bodies in the scene (Body25 model), using the Kinect Skeleton System. Alongside the 3D coordinate per joint, the corresponding pixels on RGB frames and depth maps are also provided in every frame. The depth maps are sequence of 2D depth values in millimeters with resolution of 512 x 424 achieved by a lossless compression of each individual frame. These resolution is the same for the infrared sequences in each frame. The RGB videos are recorded in the original resolution of 1920 x 1080.

NTU-RGB+D 60, described in [7], has 60 action classes in total, which are divided into three major groups: 40 daily actions (drinking, writing, reading, etc.), nine health-related actions (sneezing, chest pain, falling, etc.), and 11 mutual actions (punching, pushing, hugging, etc.). The complete set of actions, their IDs and names are presented in Table 2.2. To perform these actions, 40 subjects between the ages of 10 and 35, were used, each with one ID. Three cameras simultaneously captured three different horizontal views from the same action. 17 setups were defined, where in each one, the three cameras were located at different heights and distances from the subject. The three cameras differ in the horizontal angles consisting of -45° , 0° , and 45° . To each camera, it was attributed an ID number, and each camera ID corresponds to a horizontal angle. Each subject was asked to perform each action twice, once towards the left camera and once towards the right camera.

From the possible actions in the dataset (detailed in the Table 2.2) and their Action ID/ Name distribution it is notable that similar actions are brought together such as *Drink Water* and *Eat Meal*, *Nod Head/Bow* and *Shake Head*. This is helpful because if a model fails to recognize an action by a slight difference in action ID, it might be better than mistaking it with an action with a far away ID since close IDs entail similar actions.

The dataset also provides two evaluation benchmarks as train-test splits, Cross-Subject, and Cross-View. In the first benchmark the 40 subjects are split into training and testing groups. Each group consists of 20 subjects. For this benchmark, the number of training and testing samples are 40320 and 16560, respectively. In the second benchmark for testing, all the samples of one camera were selected, and the rest of the other cameras samples were taken for training. This division resulted in a training and testing set with 37920 and 18960 samples, respectively.

NTU-RGB+D 120 [21] is very similar to the NTU-RGB+D 60, but it has more classes, more subjects, and more setups with the camera's position at different heights and distance to the subjects. It has two benchmarks as well, Cross-Subject and Cross-Setup evaluation, where the first follows the same logic as in the NTU-RGB+D, and the Cross-Setup evaluation selects all the samples with even setup IDs for training and those with odd setup IDs for testing. The new actions

Action ID	Action Name	Action ID	Action Name
1	Drink water	31	Point to something
2	Eat meal	32	Taking selfie
3	Brush teeth	33	Check time (from watch)
4	Brush hair	34	Rub two hands
5	Drop	35	Nod head/bow
6	Pick up	36	Shake Head
7	Throw	37	Wipe face
8	Sit down	38	Salute
9	Stand up	39	Put palms together
10	Clapping	40	Cross hands in front
11	Reading	41	Sneeze/Cough
12	Writing	42	Staggering
13	Tear up paper	43	Falling down
14	Put on jacket	44	Headache
15	Take off jacket	45	Chest pain
16	Put on a shoe	46	Back pain
17	Take off a shoe	47	Neck pain
18	Put on glasses	48	nausea/vomiting
19	Take off glasses	49	Fan self
20	Put on a hat/cap	50	Punch/ slap
21	Take off a hat/cap	51	Kicking
22	Cheer up	52	Pushing
23	Hand waving	53	Pat on the back
24	Kicking Something	54	Point finger
25	Reach into pocket	55	Hugging
26	Hopping	56	Giving object
27	Jump up	57	Touch pocket
28	Phone call	58	Shaking hands
29	Play with phone/tablet	59	Walking towards
30	type on a keyboard	60	Walking apart

Table 2.2: Action IDs

added for this version (actions 61 to 120) were mainly focused on interaction with objects or other humans, so we considered not applicable to our scope of action recognition.

2.2.2 Machine Learning in Human Action Recognition

2.2.2.1 STA-LSTM

STA- LSTM is a model presented by Song *et al.* [3], where its name means (S)patial (T)emporal (A)ttention - LSTM due to the presence of spatial and temporal attention mechanisms applied with a main LSTM network in between. This model takes as input 3D coordinate joints representing the human pose acquired from the SBU and NTU-RGB+D 60 datasets. The model architecture, as mentioned, is divided into three groups: 1) Spatial Attention, 2) Main LSTM Network, and 3) Temporal Attention. Initially, they perform spatial attention to give more importance to some

skeleton joints than others. Then, it goes through a three-layer LSTM, and, before the output, a temporal attention is performed to highlight discriminative frames.

The output goes through the final activation, known as softmax. The Spacial and Temporal Attention modules contain a LSTM layer each, and the main LSTM network has three layers to avoid overfitting, as it might happen if they included more layers. Dropout was included, as well as Normalization layers in the Attention modules as seen in Fig. 2.11.

Both attention mechanisms are trained separately, where after tuned are incorporated with the main LSTM, which will primarily be trained only with learning in the main LSTM sub-network, disabling learning in the attention mechanisms, and then fine tune the final model with all sub-networks with enabled learning. Although they do not specify how the model is fed with regard to the number of people performing an action, from the "punching" action represented in Fig. 5 a) in Song *et al.* [3], one might draw the conclusion that both skeletons are fed to the network, where through the attentions mechanisms, some joints, from both skeletons, are attributed more significance as it can be visualized as the red circles in the mentioned figure. Furthermore live testing is not preformed, but due to the tests offline they conclude that the attention mechanisms proved to be helpful to the model training resulting in an increase of performance of the model when compared to the results obtained by a model composed either by only the main LSTM network, by the main LSTM network and spatial attention mechanism or by main LSTM network and temporal attention mechanism.

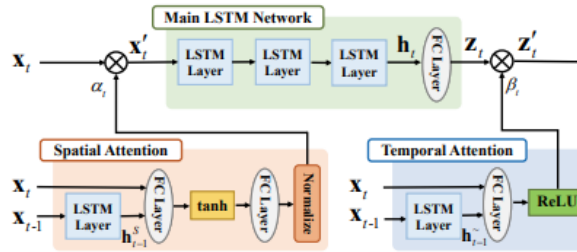


Figure 2.11: Illustration of the architecture of STA-LSTM. Figure adopted from [3]

2.2.2.2 ST-LSTM

The ST-LSTM model is presented by Zhang *et al.* [22], where the ST stands for (S)patial (T)emporal. Despite they not naming the model specifically, their approach uses ST-LSTMs. each neuron of each ST-LSTM layer corresponded to a joint. The neuron states are transformed in a tree structure based on skeletal connections, and each neuron has two hidden units, one for the previous frame and the other for the previous joint.

As the input, this model explored the geometric features instead of just taking the 3D coordinate joints. They tested some straight forward features such as joint to joint distance or orientation, and went to more complex features based on a created set of lines and planes. Each line formed by the connection of two specific joints under certain rules, for instance, one of the rules was the

lines connecting the extremities joint, head, hands, and feet. As for the planes they were defined by 3 joints in the 3D space, where only 5 planes were considered, the torso plane, arms and legs. The feature extraction resulted in a total of 8 features, including the Joint Coordinate (JC) feature.

They tested the model on SBU, NTU-RGB+D 60 and 120, UT-Kinect, and Berkley MHAD datasets. The architecture used was a 3 ST-LSTM layer network. They concluded that feeding the model with all the 8 features didn't outperform the best feature results, this being the Joint to Line Distance (JLD). Regarding the JC feature, JLD, Joint to Joint Distance (JJD) and Line to Line Angle(LLA), proved to be more discriminative features, since they all had higher accuracies than JC. Furthermore, although all of them suffer from overfitting, due to this discriminative difference, JC suffer the less overfitting since the optimization gets rather difficult.

They also conclude that a subset of the JLD can also achieve comparing results, and that JLD needs less training samples to optimize.

Despite this conclusions they don't perform live testing, neither specify how the input is processed regarding the number of persons.

2.2.2.3 VA-LSTM

VA-LSTM was presented by Zhang *et al.* [4], where the VA stands for (V)iew (A)daptation sub-network, and it tries to tackle the problem of view invariability. They try to make the model learn how to transform the data into a front view skeleton and from there achieve the action classification. They take as input the 3D coordinate joints acquired from SBU, SYSU, and NTU-RGB+D 60 datasets.

The architecture of the model sums up in two different groups: 1) View Adaptation Subnetwork, and 2) Main LSTM Network. The first consists of two branches, one for the rotation and another for the translation composed by an LSTM Layer and a Fully Connected layer. Later on, this will modify the initial skeleton to achieve a better point of view for action recognition, such as a more front directed and centered point of view. The Main LSTM Network Consists of three LSTM layers followed by the usual Fully connected layer and softmax activation to produce the desired label as shown in Fig. 2.12.

They conclude that using rotation and translation modules the model accuracy would rise in comparison to using none or only one of them. Moreover they conclude that frame level pre-processing is inferior to sequence pre-processing, the difference between both can be understood, with the translation module for example, as to where to translate the skeleton in order to be centered, where in the frame level, the skeleton is translated to the center of the axis, whereas in the sequence level the skeleton is translated to the position in the initial frame. The last conclusion is logic, since the sequence level pre-processing keeps information between frames, such as movement, where the frame level one loses that kind of information.

Despite this conclusions, the model doesn't perform any live tests nor specifies how the input is processed with regards to multi-person performed actions.

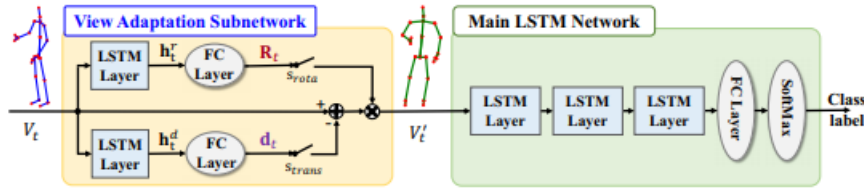


Figure 2.12: Illustration of the architecture of VA-LSTM. Figure adopted from [4]

2.2.2.4 MS-AAGCN

This model presented in Shi *et al.* [5] composes one of the top best models up to this day, where the name stands for (M)ulti-(S)treem (A)ttention-enhanced (A)daptive (G)raph (C)onvolutional neural (N)etwork. We say composes because the MS-AAGCN operates on skeleton data, but it was incorporated to work simultaneously with a RGB image-based model. To the result of this fusion, it was attributed the name of RNX3D101+MS-AAGCN-C.

To acquire the skeleton data they used both the skeleton modality provided by NTU-RGB+60 dataset and the skeletons extracted by the OpenPose toolbox from the Kinect dataset.

Since we will only work with skeleton data, we will focus this model explanation on the MS-AAGCN part. They take as input the 3D coordinate joints, from where they extract the bones' information and bones and joints' motion. These features are separated into four streams and fed to the model, hence the MS in the name (Multi-Stream). The architecture consists of a series of "basic blocks" containing adaptive graph convolution layers. They try to find the best graph topology to represent the data in a given layer for a given sample, based on a gate mechanism between two graphs, one learned for each layer, and one to each sample. The weights of both graphs, in the global graph calculation, change between layers, where in the deepest layers, the sample dependent graph will have higher weights.

The solution for the highly variant graph in the beginning of the training is to initialize the weights used to generate the sample dependent graph and its weight in the global graph calculations to 0.

The basic blocks also contain a have spatial, temporal, and channel attention blocks, where the channel is the feature space. For regularization, they incorporate Batch Normalization and ReLU activation layers and a residual connection as seen in Fig. 2.13 on the left. In this figure, *Convs* represent the spacial adaptive graph convolution layer, the *ConvT* represents the Temporal adaptive graph convolution layer, and the STC represents the Spatial, Temporal, and Channel attention. The global architecture is a nine basic blocks stacks one after the other followed by a Global pooling and a softmax activation layer, and it is represented in Fig. 2.13 on the right.

Regarding the process applied to multi-person action samples from the NTU-RGB+60 dataset, they fed a concatenated version of the skeletons from both persons, where the second person skeleton is zero padded depending on the number of people detected. Moreover to set every video sample to the same length the padded each video by replicating the actions until they achieve the maximum action length of 300.

From the tests they conclude that the inclusion of both sample and layer dependent graphs outperform the models with none or just one of them. Furthermore, to test the best solution for the highly variant graph in initial epochs, they compared results with the solution mentioned earlier and the solution where the typical human based skeleton topology was included in the weighted sum calculation for the global graph, where the weights for sample and layer dependent graphs were initialized with 0, from where they concluded the non-inclusiveness of the human based skeleton topology solution performed best. Moreover they also concluded that the inclusion of the gate mechanism to control the weights of both graphs improved accuracy.

Tests were made regarding the Spatial, Temporal and Channel Attention, whether they were used or not, and if so whether they were alone or combined, where the attention mechanisms were tested in a common Spatial-Temporal Graph Convolution Neural Network(ST-GCN). For the combining strategy they tested two methods, the Adding all the attentions or by concatenating them. Which they concluded that combined attention helped improve training and that the concatenation method slightly outperformed the adding strategy. Besides the last conclusion they perform the combining tests using the AGCN proposed, which improved accuracy in relation to a ST-GCN.

Moreover they tested the four streams, mentioned above, with respect to the standalone use or the combining tests of them. From where they concluded that both the combination of joint and bones spatial information (JB) and the all stream combination(or Multi Stream - MS), performed better than when each stream was fed individually, where the MS combination, outperforms the JB combination.

Despite these conclusions this model also wasn't live tested.

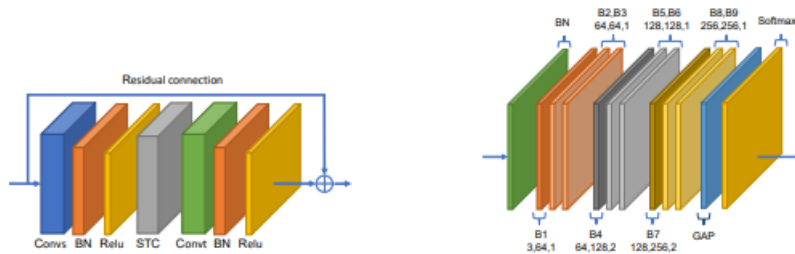


Figure 2.13: Illustration of the architecture of MS-AAGCN [5]

2.2.2.5 DSTA-NET

DSTA-Net was presented by Shi *et al.* [6], where the name stands for (D)ecoupled (S)patial-(T)emporal (A)ttention (NET)work, and it's also in the top best models in this topic. They take the skeleton data of each samples represented as a matrix $N \times T \times C$ where N is number of joints, T the number of frames and C the channels, being the x , y and z coordinates. They decouple the video samples skeleton sequence into four streams, the spatial temporal stream, spatial stream and the slow and fast rate temporal stream. The Spatial information can be understood as the distances of each joint in different frames, and the temporal streams can be understood as the movement of

the same joint in different frames, meanwhile the spatial temporal stream represents the raw joint coordinates.

The low and fast rates are related to the frame distance from which the movement is acquired. A low rate would have a higher frame distance than a fast rate. Then they feed each stream to L stacked attention blocks that in the end are passed through a global average pooling, a fully connected layer and a softmax activation layer, as illustrated on the left in the Fig. 2.14. The attention block regarding the spatial stream has a Position encoded and Spatial Global Regularization blocks, to keep track of the joints and to force the model to learn more general attention for different samples as presented in Fig. 2.14 on the right. In the same figure, the strategy chosen for decoupling the outputs of the positional encoding and forward them to the attention mechanism was the best strategy out of 3 tested ones. The first strategy consists in having an Attention map resultant from the application of a softmax activation over the dot multiplications of both decoupled encodings frame by frame, having a different Attention map for each frame. The second strategy, is similar, but an attention map is calculated for every 2 joints in all frames, resulting in an high computation complexity and meaning that the inter and intra-frame relations were taken into account simultaneously. Finally the last strategy consisted in a compromise between the previous ones, where for every frame an attention map will be calculated but all the attention maps for that frame are averaged and shared.

The model is tested with SHREC, DHG and NTU-RGB+60 datasets.

They compare the need for position encoding for the skeleton data with respect to the space and time dimension. For this comparison they test the no use of positional encoding (PE), or the use of either (U)nified and (D)ecoupled PE, where in the former each joint across all time is has a sequential encoding, and in the latter the encoding is decoupled for either time and space dimensions, where in the space one each joint in the same frame have sequential encoding and the same joint in different frames has the same encoding, whereas in the time dimension, reversely and analogously, all joints in the same frame have the same encoding, and joints in different frames have a sequential encoding. From these tests was concluded that the DPE outperforms the UPE. Furthermore, using DPE, they study the use of (S)patial or/and (T)emporal (G)lobal (R)egularization module, where the temporal one, is analogous to the spacial in the sense that forces the model to learn more general attention for different frames of a video sample. Relying on the tests effectuated, they concluded that the temporal regulation would prejudice the model, where, on the hand, the spatial one would improve accuracy. Moreover they compare the 3 decoupling strategies mentioned above, where the third achieved best accuracy.

They also study the use of each of the streams, mentioned above, alone and combined, where the result show that the spatial temporal stream alone achieves best accuracy than the other streams alone, but combined achieves an even greater accuracy. Furthermore, comparing the class accuracy differences between streams, they also concluded that for actions with fast changes in time the fast rate temporal stream achieved higher accuracies, where in the slow changing actions, the slow rate temporal stream achieved higher performances, comparing the temporal streams. Regarding the performance difference between spatial stream and slow rate temporal stream, the actions which

are more related to the shape changes are better recognize by the spatial stream, and the actions more related to the position changes are better classified by the slow temporal rate

Despite these conclusions this model wasn't live tested nor details about the process applied to video samples with actions preformed by two persons.

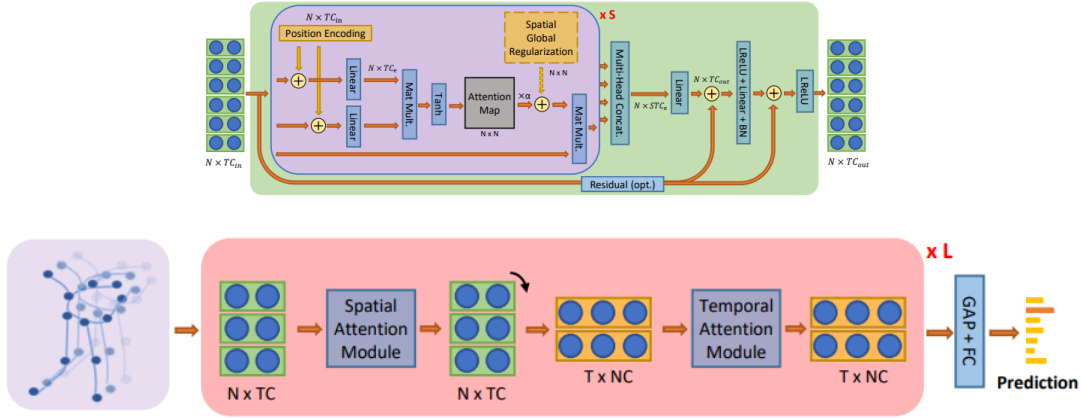


Figure 2.14: Illustrations of the architecture of DSTA-Net [6]

2.2.2.6 P-LSTM

This model is presented in the Shahroudy *et al.* [7] takes as input the 3D coordinate joints of a human skeleton, acquired from the NTU-RGB+D 60 dataset. The architecture is two Part-Aware LSTM layers. The Part Aware LSTMs are proposed layers in the paper, where each neuron of the LSTM is specific to one body part, they have the individual gates with exception of the output gate which they all share. We can see the Part-Aware LSTM illustrated in the Fig. 2.15.

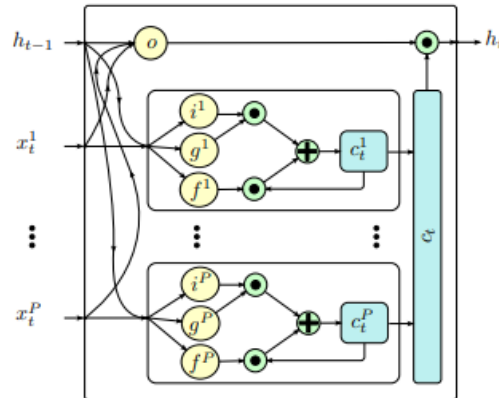


Figure 2.15: Illustration of the architecture of Part-Aware LSTM [7]

In Fig. 2.15 the g and i gates correspond to the input gate in Fig. 2.4 explained in the Subsection 2.1.2. At a frame t , P body parts are present in the frame, where each part is represented with

a x_t^p . There are five body parts identified : torso, two hands, and two legs, each part contains the correspondent joint coordinates.

To feed the model they applied some pre-processed techniques such as centering and rotating the skeleton data, when 2 bodys were recognized both were transformed with regard to the main actor's skeleton, which the identification of the main actors is done by analysing the 3D body motion of each, the one with the highest motion is identified as the main actor.

Furthermore, each video sample, in each training iteration, was evenly divided into 8 temporal segments, from where a frame will be randomly selected from each, to build the model recurrent learning models input.

They compare state-of-the-art models with publicly available implementation, along side with RNN, LSTM and the proposed model all of three with 2 variations, each either with 1 layer or with 2. The results showed that the proposed model out performed any other test, both with 1 and 2 Part-aware LSTM layers, the latter with higher performance.

Despite the conclusions, no live tests were performed, nor more detailed information, besides the one mentioned above, about the multi-person actions process was provided.

2.2.2.7 CNN+Motion+Trans

This model is the one closest to our final model, and it is presented in [8], where the nomenclature comes from the main model architecture, CNN, the temporal feature they extract from the action frame sequences, Motion, and for the inclusion of an attention mechanism denominated by transformer. They take as input the 3D coordinate joints and extract the motion between two consecutive frames, resorting to the NTU-RGB+D 60 dataset. They implement a variant of an attention mechanism called a skeleton transformer, using a fully connected layer with no bias.

To address the action length variability problem, they applied a resize operation to the image representing the video samples. On the other hand, to address the multi-person actions they perform a maxout operation after the last convolution layer, previously sending through the same layers, up to this point, both person skeletons. In the maxout operation the elements of both feature maps are element-wise max pooled, avoiding the zero pad strategy adopted in other papers such as Shi *et al.* [5].

Since the model is fed in a two-stream input, this skeleton transformer is applied to the skeleton joint stream and motion stream. The output of each is passed through a convolution network. Both outputs are concatenated and forward to another convolution net and finally by a fully connected layer and a softmax activation as seen in Fig. 2.16 on the top. This network, until the last fully connected layer, composes the backbone network, and its purpose is to form an image with lots of channels, where these channels are features extracted by the convolutions. To this image, a few windows are extracted using a window proposed network (WPN), which learns how to choose what are the best windows to extract. The extracted windows are then resized to the same size and fed to another convolution network and then to a fully connected layer and a softmax activation function as can be perceived from Fig. 2.16 on the bottom.

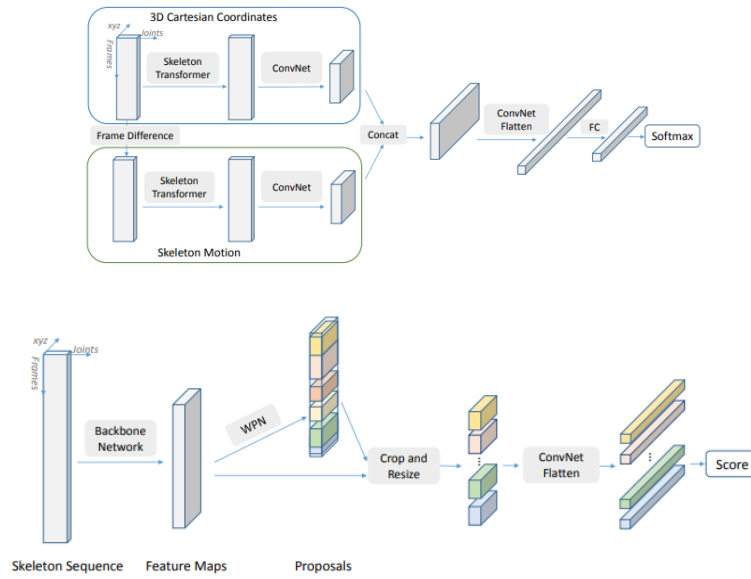


Figure 2.16: Illustration of the architecture of CNN+Motion+Trans [8]

They test the main CNN Network fed with only positional information (without the motion) and without attention, with Attention, fed only with motion features, and all together with both positional and motion information as well as the attention mechanism. From where the results showed that the model fed only with Motion and the model fed with positional with attention mechanism, performed better than only with positional information, having the model fed with motion without attention a slight distinction in the Cross-View NTU-RGB+60 benchmark. On the other hand the inclusion of the attention mechanism and the two streams input achieved even higher accuracies.

Comparing with some state-of-the-art models using wither LSTM or CNN, also showed great promising in accuracy terms, leading them to conclude that CNN can also learn temporal dependencies, in this case even better than LSTM networks.

2.2.2.8 Comparing Results

In Table 2.3 the results for accuracy (in percentage), for each model are presented. Since NTU-RGB+D datasets have predefined splits (Cross-Subject (CS) and Cross-View (CV)), the results for each split's accuracy are presented ($CSAcc$ and $CVAcc$). The other datasets accuracy is presented as Acc . The values were retrieved from the papers that introduced each method. In cases where the paper does not present results for a particular dataset, the value is omitted.

Regarding the results shown, we can notice an improvement through the years in accuracy values for common datasets. Along side the chronological evolution, we can also see that in the beginning LSTM based models were predominant. Then, the incorporation of attention mechanisms contributed to an improvement in the model's accuracy. More recently, CNN and GCN are

Name	Model Name	Dataset	Acc	CS Acc	CV Acc	Year
Shahroudy <i>et al.</i> [7]	P-LSTM	NTU-RGB+D 60	-	62.93%	70.27%	2016
Song <i>et al.</i> [3]	STA-LSTM	SBU NTU-RGB+D 60	91.50% -	- 73.40%	- 81.20%	2016
Zhang <i>et al.</i> [22]	ST-LSTM	SBU UT-Kinect Berkeley MHAD NTU-RGB+D 60	99.02% 95.96% 100% -	- - - 70.26%	- - - 82.39%	2017
Zhang <i>et al.</i> [4]	VA-LSTM	SBU SYSU NTU-RGB-60	97.2% 77.5% -	- - 79.40%	- - 87.60%	2017
Li <i>et al.</i> [8]	CNN+Motion+Trans	NTU-RGB+D 60	-	83.20%	89.30%	2017
Shi <i>et al.</i> [5]	MS-AAGCN	Kinetic	61%	-	-	2019
		NTU-RGB+D 60	-	90.00%	96.10%	
	RNX3D101+MS-AAGCN-C	Kinetic	-	-	-	
		NTU-RGB+D 60	-	96.20%	99.00%	
Shi <i>et al.</i> [6]	DSTA-NET	DHG	93.8%	-	-	2020
		SHREC	97%	-	-	
		NTU-RGB+D 60	-	91.50%	96.40%	
		NTU-RGB+D 120	-	86.60%	89.00%	

Table 2.3: Results comparison of Deep Learning models for Human Action Recognition

being introduced and are capable of significant growth in performance concerning the LSTM models. Besides that, for NTU-RGB+D it can be inferred that the Cross View train-test split achieves higher scores than the Cross Subject in all papers. This table will later on be used to locate our developments within state-of-the-art.

Chapter 3

Action recognition Framework

3.1 First Approach

In this section we will present the details of our first approach. First we will go over the analysis and processes applied to the dataset, in Subsection 3.1.1. Afterwards we will explain the type of features that we extract in Subsection 3.1.2. And finally we will go over the model creation and analysis performed to classify the actions on the features selected (Subsection 3.1.3). This approach revealed several limitations and problems (shown later in the Experiments and Results Chapter 4).

3.1.1 Dataset Preparation

As previously mentioned, we decided that the best dataset to use was NTU-RGB+D 60, due to its popularity in papers in the same area, to the dimension, and the variability in the dataset. Having 40 subjects and three different points of view, we would expect the model to be subject and view-invariant. More details on the dataset characteristics can be found in Subsubsection 2.2.1.1.

In NTU-RGB+D 60, both the RGB frames and the skeleton sequence are already available. However, since the main goal is to develop a real-time action recognition system, we will need to use a camera feed as a live input. A camera video frame, by itself, can not estimate poses. So, analogously to the process that will be performed to extract the skeleton pose from the camera, we will apply the same approach to the dataset for better results. For that, we rely on the RGB frame dataset modality and use an algorithm to extract the skeleton poses.

The software we employed was OpenPose [23], an open-source API that takes as input video formatted data and extracts skeleton poses, able to save the skeletons in a lightweight data-interchange format, JSON (JavaScript Object Notation). The default body model is the Body Model25, and its structure and joint organization can be seen in Fig. 3.1. It is important to notice that the joints numbered between 0 and 15 have most of the skeleton's positional information. The following 10 (from 16-25) are more useful to the skeleton's orientation information.

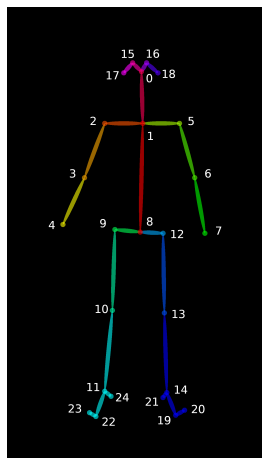


Figure 3.1: Body Model 25

The dataset videos are encoded in Audio Video Interleave (AVI) format, and they are fed to OpenPose, each one separately. For this, a *powershell* script is used that also supplies the desired folder's location to save the JSON files, where we use a different subdirectory for every video sample. The JSONs generated by OpenPose have the information encapsulated as a dictionary. This dictionary can be understood as a data structure of pairs key and value, where the keys are a string and the value correspond to its content. The content can be almost any type of data from single values, to a list, to another dictionary. The joints' positional information can be found in the *'people'* key, where the content is a list of each person detected in the scene.

Meanwhile, the person's information is stored in another dictionary, where the joints positional information is found in the *'pose_keypoints_2d'* key. In this key the content is a serialized array of joint's position and confidence of recognition. OpenPose is capable of recognizing 3D and 2D coordinates depending on the nature of the input. Since only 2D is achievable from both the dataset's RGB frames and the RGB frames captured by the live feed camera, we opted for 2D. This allowed using videos recorded with any camera, allowing for testing the model with home videos and commercial webcams, if needed. With this compromise, the final joint positional information is an array of 75 values per frame, grouped in sets of three per joint. Each group has the *X* and *Y* coordinates of the joints in the frame, along with its confidence with values from 0 to 1.

With this in mind, we found some limitations on the OpenPose software, such as when a person's joint is not visible, OpenPose attributes to that joint the previous known coordinates and confidence of 0. On the other hand, if a person has never appeared in the camera, then the info shown is an empty list. The pose estimation software performance is also dependable on the number of people it aims to recognize and can receive as a parameter the maximum number of people. Since in the dataset, the maximum number of people performing an action is 2, we send that information to OpenPose to increase its performance.

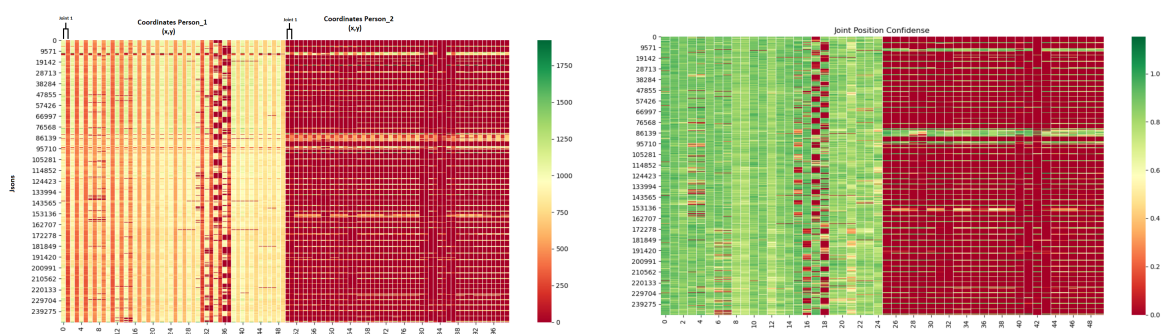


Figure 3.2: Raw data returned from OpenPose when applied to NTU-RGB+D dataset

The raw data returned from OpenPose regarding the first setup, with setup ID 1, can be visualized in Fig. 3.2. In both subfigures, the positional information from two persons is presented, dividing the X-axis in half. The right part of the figures is regarding the second person, which when an action is performed only by one person, the data referent to the second person is set to zeros. The left figure represents the *X* and *Y* distribution, and the right one represents each joint's

confidence. Both Y-axis represent the frame samples or the JSON files present in setup 1.

Two images are presented instead of one due to the different range of values of each variable, where X , Y and Confidence ranges are $[0,1920[$, $[0,1080[$ and $[0,1]$, respectively. The color difference in the left figure represented as an interleaving of yellow and red colors are due to the difference in X and Y ranges. The red columns are mostly Y values, and the yellow ones the X values of each joint. In sum, each joint coordinate is represented in two consecutive X -bins in the image. Bins 0 and 1 are referent to joint 0, bins 2 and 3 to joint 1, and so on. Similarly, in the right image, we have for each X -bin a value of confidence on a joint position, where bin 0 is referent to Joint 0, bin 1 is to 1, and so on.

With that in mind, we can see in both figures that around Joint 15 to Joint 18, OpenPose does not perform well and misrecognizes the actual position of the joint. To tackle this problem, we decided to only focus on the first 15 joints because we believe those joints have the highest ability to hold information about the actions performed and provide enough information so that the rest of the joints can be ignored. The small vertical red sequences that might be found in the right figure means that a joint that is not recognized since the beginning, from a certain point till the end of the action. It can also occur if a joint gets occluded from a certain moment, and the 0 coordinates are attributed to that joint. It is also notable the amount of data referent to the second person being 0. Having many zeros in a dataset might be harmful because they do not bring any new information to the model.

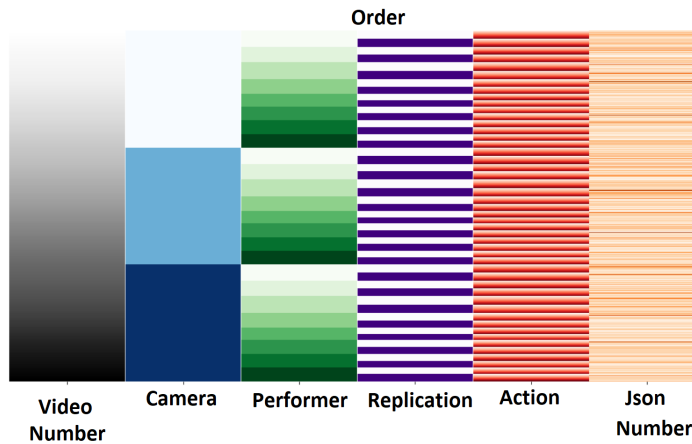


Figure 3.3: Setup 1 Order

The dataset's order of organization is represented in the Fig 3.3, where the Setup 1 samples organization is showed as an example. The nomenclature of the video samples is composed of 6 fields: 1) (S)etup, 2) (C)amera, 3) (P)erformer, 4) (R)eplication, and 5) (A)ction. The number of actions, replications, and cameras is constant through all performers, having respectively the values 60, 2, and 3, but different setups might have different performers. There are a total of 40 subjects, but only eight are included in the first setup. In Fig. 3.3 all the field distribution for setup 1 are shown, with the inclusion of a gradient color bar representing the number of videos in that setup.

To better understand the zero distribution along the dataset, we show the histogram in Fig. 3.4. The orange color represents the zeros in the middle of the action. By the height of the orange bar, one can conclude that there is at least one moment in time, before and after, that the joint is recognized and does not have 0 coordinates. On the other hand, the yellow color represents the joints with 0 coordinates from the beginning of a video until some moment in the video. Analogously the blue color represents the joints with 0 coordinates from some point till the end of the video. The Green color represents the number of joints that are never identified since the beginning and until the video's end. Finally, the grey bar represents the total number of zeros.

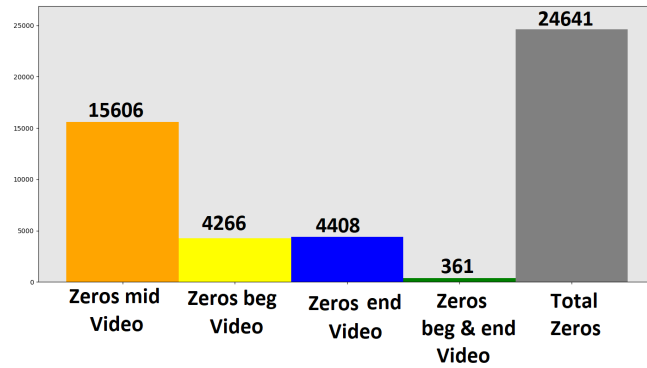


Figure 3.4: Zero Joint Statistics

Facing this histogram, we notice that most of the zeros are in the middle of the actions. An example of this is represented in Fig. 3.5. The image sequence is referent to the action "Pick up", where when the subject is leaning down to pick up, the Joint from the hand is occluded, and its position is attributed to zero. To solve this problem, we propose a window average method to fill the missing joint positions.

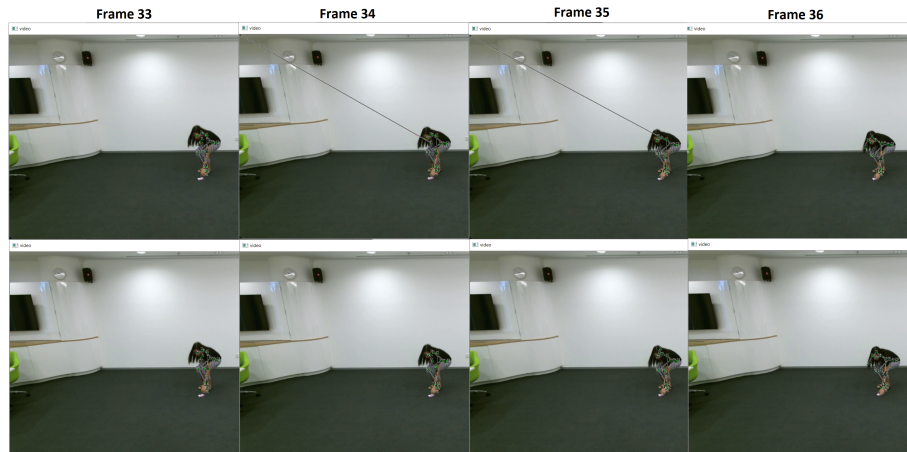


Figure 3.5: Example of a sequence with zeros in joints, mid action

For the first 0 coordinate found in the middle of the action, we take the previous 10 values of the same joint's coordinate and average it to achieve the new coordinate's value. In cases where there are less than 10 values, the maximum possible window size will be assumed and averaged.

Although it's not optimal precise, we believe that this simple process will decrease the error of the Joint location in comparison to a zero valued joint coordinate. The results for this process can be seen in the changes in the histogram from Fig. 3.4 to Fig. 3.6

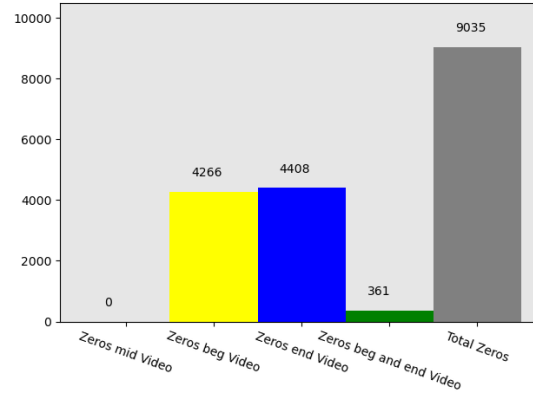


Figure 3.6: After Process Zero Joint Statistics

Like mentioned before we only aim for action recognition, so only actions performed by one person will be used (categories "Daily Actions" and "Medical Conditions" from NTU-RGB+D), which represents 5/6 of the entire dataset. It is important to notice that some of the video samples where the action is performed only by one person, might contain another person in background and are non relevant for the action.

3.1.2 Feature extraction

In this stage of the developments, relevant information (features) were extracted from the dataset created. We choose to use geometric features used in Zhang *et al.* [22]. A feeding sample, should be understood as the sample format of each element of the batch being fed, where the batch is an agglomerate of feeding samples that will be fed to a model, useful for memory management since it allows to not bring the entire dataset into memory. The feeding sample should have shape [Window Size, Feature size], and it should be a window of frames with the geometric features in each frame, having in mind that the video samples are sequential data. This means that we cannot change the order of the JSON files in a video sample, but we can shuffle the order of when each video is fed without changing the order of the video itself, meaning that we can either feed the model first with video sample 1 and then with video sample 2, or in reverse order, similarly we also can feed the model with the first window of a video sample and then the second window of the same, or in the reverse order, but inside a window we cannot change the order of each frame. In a window the first frame will always be older than the following frames, and in the same window, only frames of the same video sample are present.

We chose to use a time window to have size 11, where it would include the present frame and the last ten, but the time window would only start on the 11th JSON of the action. Each time window would be a sample to the model, and to each one a label is associated. With this operation

the number of samples per action would be slightly reduced but the size of each sample increase significantly, in comparison to the initial frame samples.

We will go through a detailed explanation of the feature extraction performed in Zhang et al. [22] and how did we apply a similar feature extraction in our dataset. The extraction operates on two sets: a joint set, in our case, all the first 15 joints, and a line set. The line set is achieved by selecting in groups of pairs of joints, where the groups are created accordingly to one rule each:

1. Every line composed by two directly connected joints.
2. Every line composed by any of the joints in the extremities, head, hands and feet, and the torso
3. Every line composed between any two joints in the extremities

This line selection can be observed in Fig. 3.7, where on the top four figures, a joint numbered skeleton is showed in the background and the selected lines on top of it. The numbered skeleton is missing on the bottom five figures, and only the joints and selected lines are shown. On both top and bottom, the first image represents the skeleton with no lines selected; the second image, the first group of lines; the third image represents the second group, and the fourth the third group of lines. In the fifth bottom image, a gathering of all groups is presented, noting that the line corresponding to the neck present in groups 1 and 2 is counted as only being one line, resulting in 28 lines.

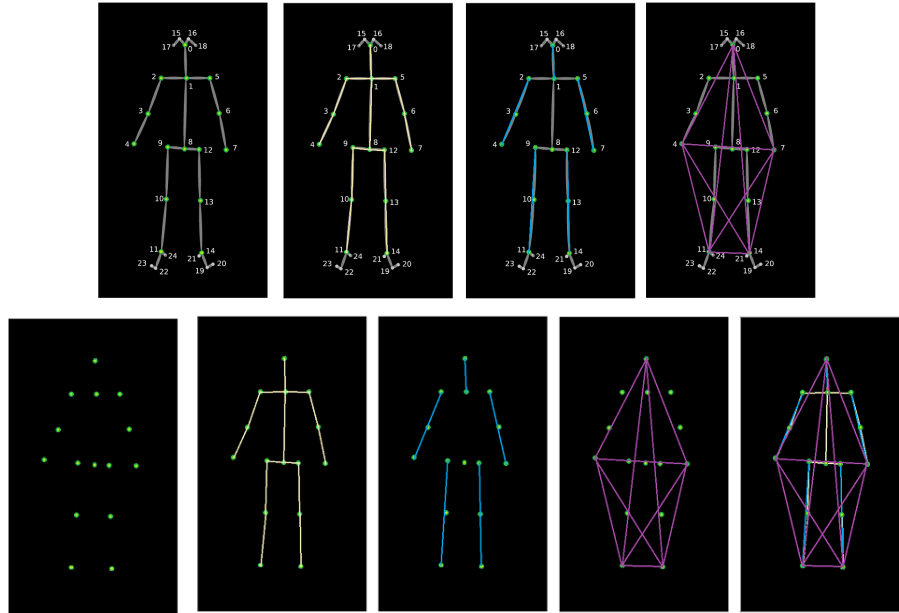


Figure 3.7: Line Set

With this in mind, we proceed to present Table 3.1 to demonstrated the feature size and formula based on the two sets of joints and lines. The *POS* feature name is referred to the skeleton joint's position, where the skeleton has 15 joints, each with two coordinates, resulting in a feature size

of 30. Similarly, the *Joint to Joint Orientation (JJO)* will have two values for each pair of Joints, resulting in 420 values. Each value is normalized, corresponding to x and y coordinates, and the vector's norm constituted by both values is one. Likewise, the norm of those values unnormalized is the distance between two joints, which forms the feature *Joint to Joint Distance (JJD)* with 210 features. On the other hand, the *Joint to line distance (JLD)* has a distinct formula to achieve the number 364, where L_i is the total number of lines where the joint i is not present. For each of those lines, the smallest distance is computed. Meanwhile, the *Line to Line Angle (LLA)* feature has a similar formula to *JJD*, where the feature size is one value for each pair of lines.

Feature Name	Formula	Feature Size
POS	15×2	30
JJO	$(15 \times 14) \times 2$	420
JJD	15×14	210
JLD	$\sum_{i=0}^{15} L_i$	364
LLA	28×27	756

Table 3.1: Feature dimensions

$$distance(P_1, P_2, P_0) = \frac{|(x_2 - x_1)(y_1 - y_0) - (y_2 - y_1)(x_1 - x_0)|}{\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}} \quad (3.1)$$

To compute *JJO* and *JJD*, as briefly mentioned above, we will get the distance vector between two joints and extract the unitary vector (*JJO*) and the norm of the distance vector (*JJD*). On the other hand *JLD*, for every joint and every line not containing the said joint, we apply the formula to get the distance between a point and a line (Eq. 3.1). A line is represented by two points, where P_0 is the joint, and P_1 and P_2 are the joints forming the line. Each joint P has x and y coordinates.

This brought some problems, such as if all three joints are zero or if P_1 and P_2 have the same location, the denominator will be zero, which may lead to a result of *NaN*. To tackle this, we made the following changes:

1. If all three joints are zero, the distance value should be zero.
2. If both line joints are 0, but the single joint is not, the distance shall give the joint distance between the single joint and one of the line joints, which are both zero.
3. If both line joints are the same, the distance value will get the joint distance between the single joint and one of the line joints since they are the same.

LLA on the other hand is computed by calculating the arc-cosine of the dot product between the transposed orientation of the joints in the first line, with the orientation of joints in the second line, which would result in a value from 0 to π in radians. In sum, we want to have a data storage with all features, inclusively the initial 2D coordinate joint skeletons (*POS*). The implementation of this plan will be detailed till the end of this subsection.

After generating the JSONs, the data was gathered. Given the number of JSON files and the nature of the data sequence, it would be useful to have some metadata for each JSON. This

metadata was composed of video number, JSON number, and action label, which was later added to the person number field. The video number is a way to save memory instead of saving all the attributes (the setup, the camera ID, the subject ID...). On the other hand, the label would be the action ID but shifted by one such that the action ID 1 would correspond to label 0. Finally, the person number is the number of persons recognized in the JSON frame. In sum the metadata can be understood as a descriptive information about the frames content.

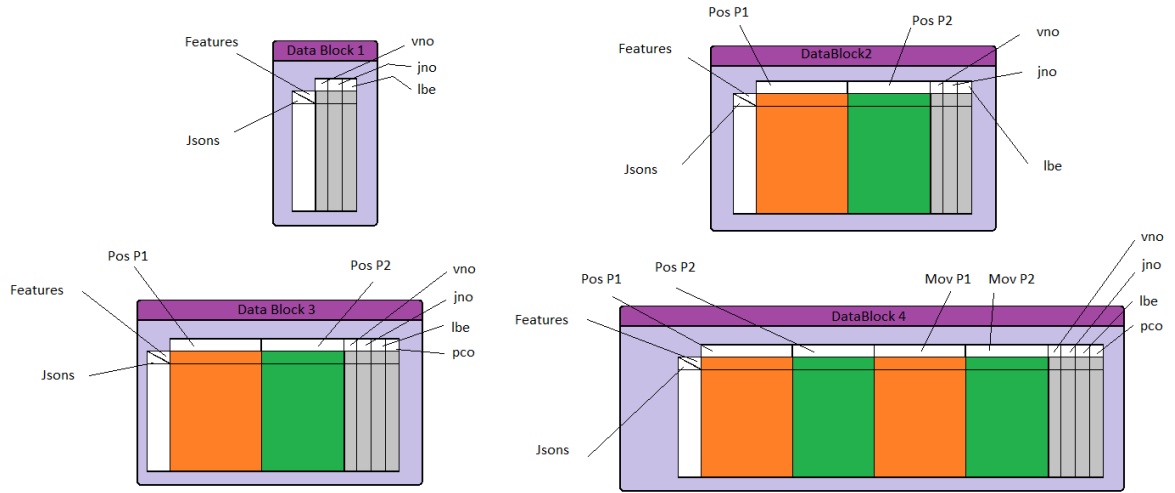


Figure 3.8: Raw Database

This metadata is organized as a table where each row is a JSON, and each column its metadata field. Afterwards, we want to append the *POS* features. Due to its simple implementation we also extracted the movement of both persons in two consecutive frames of the same action, meaning that if we take the Fig. 3.2 as an example, and we wanted to extract the movement mentioned above, and, for explanation purposes, each video samples had length of 3 frames, we were going to subtract each line, since it represents the jsons and therefore the frames, from the previous line starting in the second of each video sample. This extraction would result in 2 lines instead of the initial 3, therefore, to achieve the same line number in an action we would add a line full with 0s and place it in the first position of every video sample. If we think that each line only has one value, and take the following video sample example under such circumstances, represented by three lines with values [1,2,4], and we extracted the movement between two consecutive frames we would get a movement video sample representation with values [1,2], where from adding a 0 line in the beginning, we get the new movement video sample represented by [0,1,2], which can be understood as an element wise correspondence to each frame in the initial video sample, tho 0 would be the movement of the first frame with value 1 in the video sample example, the 1 in the movement video would be referent to the second frame with value 2 in the initial video sample and so on. We can see the file's evolution in Fig. 3.8, where *vno*, *jno*, *lbe*, and *pco* are respectively video number, JSON number, label, and person counter (or number). In Data Block 1, we can have a pointer of features pointing to nothing to stress that this block is referent to just the metadata and still does not have any features. On the other hand, *PosP1* and *PosP2* are the

POS features of both person 1 and 2. *MOV* P1 and P2 are the movement between frames of person 1 and 2, respectively.

Although we only use one person, the process is ready for adapting to other applications where multiple persons may be needed. Then from this file, we aimed at creating one for each geometric feature. Due to the similar achieving method of *JJO* and *JJD*, they were joined in a single feature called *JJOD*, further on abbreviated as *JOD*. We only extracted the geometric features from *POSP1*. If needed, the same could be applied to *POS* P2 and *MOV* P1 and P2, whose database format can be seen in Fig. 3.9. In the bottom left corner we see our final data block composed by the concatenation of all the features.

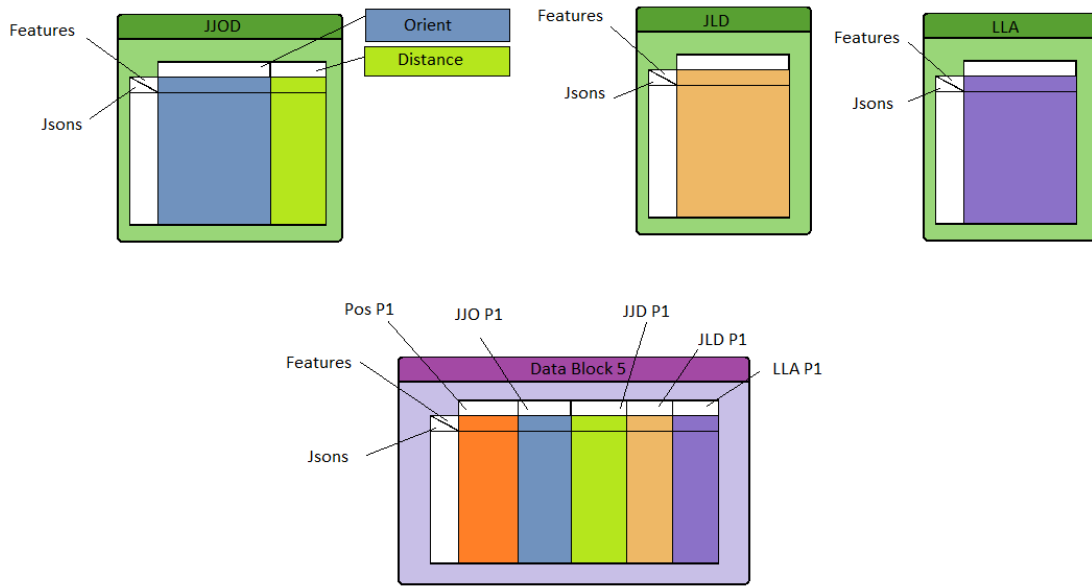


Figure 3.9: Features Databases

We will now proceed to a detailed walk-through on the model used for action recognition.

3.1.3 Action Recognition Model

In this project we first used a three layer LSTM sequential model with a Dropout layer and two Dense layers. LSTMs are able to learn temporal dependencies, which is great facing our geometric features as input. For implementation the Keras library was used. The first two LSTM layers return sequences, where the output will still have a time window with the feature sequences per sample, but with its values changed by the weights of each LSTM layer. This way the next layer will still be able to learn temporal dependencies. On the other hand, the last LSTM layer will aggregate the time window and return a feature sequence for the input sample. The Dropout layer will randomly drop 50% of the feature sequence, and the two dense layers will aggregate the feature sequences into a list with length equal to the total number of labels. This list will have values from 0 to 1, which can be interpreted as the model's confidence to label an input sample with a certain label, keeping in mind that index 0 of that array corresponds to action ID 1, and index 1 to action ID 2

and so on, further on the paper we will differentiate the action IDs format by denominate them 0 based and 1 based action IDs.

An Adam Optimizer was used because it is known to work well with big datasets, and it is broadly used in this area. As loss function, the sparse categorical co-entropy function was chosen. Similarly, the metrics used to compute accuracy, the Sparse Top K Categorical Accuracy, was also defined in Keras.

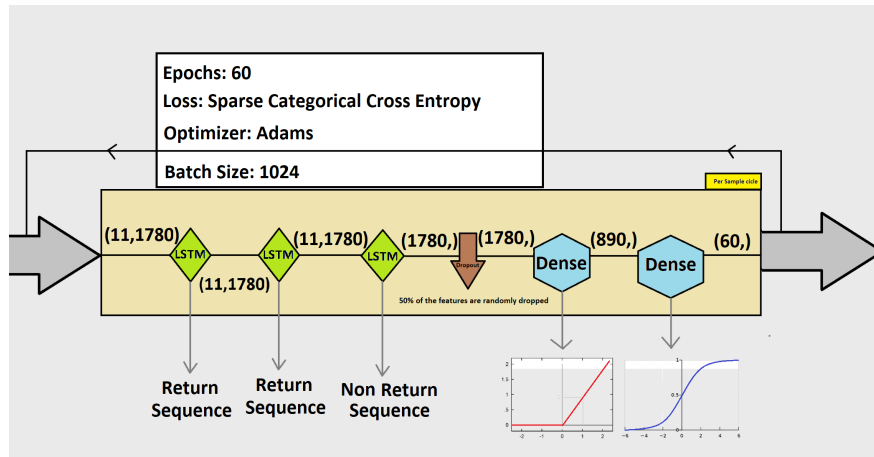


Figure 3.10: Initial LSTM based model

The model can be easily visualized in Fig. 3.10, where the numbers above the arrows mean the data shape being transformed through the layers. A softmax final activation is used as the last Dense layer, whereas a ReLU activation is used in the one before. A batch size of 1024 was used, and the model was trained for 60 epochs. The LSTM layers used are optimized to run on GPU in order to fasten the process. To the same goal, the batch size was maxed as a power of 2, as it is usually configured, to the maximum allowed by the RAM used. Maxing the batch size improves the significantly the time of training since less batches are needed to train. As one can see, this model is rather simple and lightweight.

3.2 Second Approach

In this second approach, we will explain how we implemented improvements in our dataset and feature extraction process to surpass the problems from the first approach (detailed in the later Section 4.2.1). We will also describe what issues were faced as well as the solutions implemented. Finally, the more complex model used in this second approach will be presented.

3.2.1 Dataset Preparation

The process of creating the dataset is very similar to the one presented in Section 3.1. With all the video JSONs created, given the problems found with the first approach (later detailed in Subsection 4.2.1) we started systematically re-studying its content. First, to easily access the joint

information in the JSONs, we choose to save the relevant information in a *pytable* (improving memory management). When reading the JSONs, at first, we choose only to read one person's data, and when two or more were available, we selected the first person in the JSONs 'people' array. Then we saved the values referent to x, y of the first 15 joints, resulting in an array of 30 floats, in the *pytables*, together with the video sample's name.

We save the x and y value from the first 15 joints from each frame and ignore the confidence value. This x and y are coordinates in pixels from a 1920 x 1080 window. The origin is located in the top left corner of the window, so the y coordinate will need processing if we need to set the origin to the bottom left corner when an empty JSON appeared. Meaning that if it had no persons recognized, a *None* filled array with the same shape as if a person was recognized (with all coordinates equal to *None*) was created and saved. We used *matplotlib.pyplot* and *cv2* python libraries to show the skeleton in the *pytable* on top of the video recorded. To this end, we needed to change the origin view of the skeleton to the bottom left corner. This feature enabled us to find and visualize errors needing fixing, and visualize the same data after processing.

With a CNN based model, we started to think to feed the video samples as a featured image representation with shape [Height, Width, Channels], where height would be the video sample length and the width and channels would depend on the feature, as described in Subsection 3.2.3. This kind of video sample representation in a similar form as image we call them featured images. The most common use of CNN is with fixed sized image feeding, meaning all the images we would feed would have to have same height, the other parameters where already reassured to be fixed because it would be extracted the same type of feature for all video samples. In order for all the images we feed having the same height, the video samples would have to have same length and a common way around this is with a padding technique. We chose to pad the video samples with zeros since the zeros had some meaning, as in the initial dataset if the person left the frame would later be translated as a frame with a skeleton full of zeros. We tried to implement a very simple CNN inspired by Strobel Max [24], as described in Subsection 3.2.3 and fed it with POS featured images acquired from the video samples and finally the results were positive, as described in Subsection 4.2.2.

The NTU-RGB+D 60, although a very complete dataset, has some missing skeletons, where the recommended approach is to ignore these video samples. In a similar approach we ignored the video samples where from begging to end of the video at least one joint of the 15 selected is never recognized. The number of videos to ignore resulted in 50, where their distribution was not meaningful enough to debilitate the dataset balance from this operation.

ML and DL models work with numbers. Our dataset values either are a number or '*None*.' So, we eliminated the presence of '*None*' values in our database by replacing every '*None*' with a zero. This resulted in the conclusion that now the database is composed of either a pair of coordinates in the space from a recognized joint, or it was misrecognized, and it is zero. Due to this, the dataset now has a higher number of zeros present, resulting in a lot of data with no meaning to the action, which would debilitate the model in the future. We choose to use linear interpolation to address this problem. To apply it, we needed a past value and a future value, then regarding the distance

of a point between the past and future known value, interpolate the value of that new point. If the past value is 1 and the future value is 3, and the past value is in $t=0$, and the future value is in $t=2$, we can interpolate the value in $t=1$ to be 2. In this manner, the interpolation was applied to every joint equal to zero.

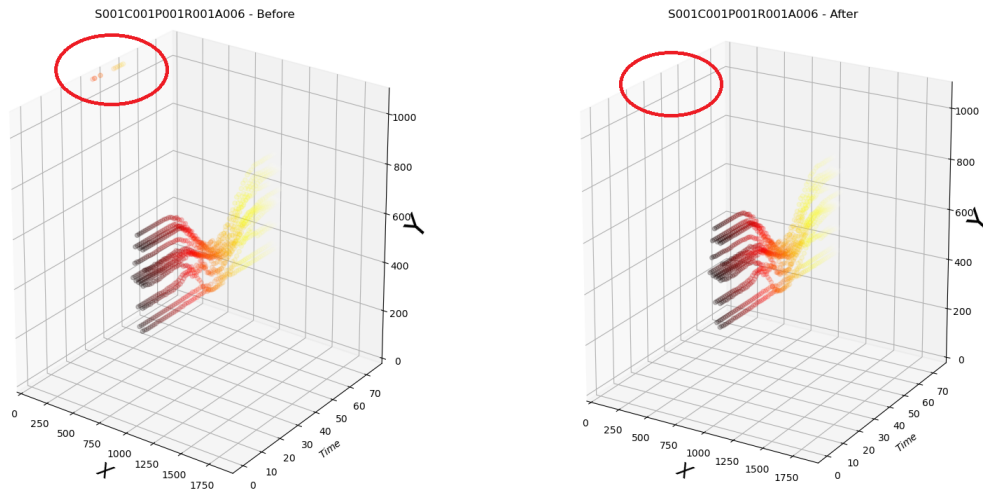


Figure 3.11: Interpolation 3D. Action representation before(left) and after(right) of the interpolation

Although this type of interpolation might have some limitations with high rate variances in the original lines, this method is effective for steady lines and small x distances. We can apply this concept in 3D lines instead of 2D, and if we understand a line in the 3D space representing the movement of a joint in the 2D space, where time would be the third dimension, we are performing a Joint position correction method to eliminate problems like the one presented in 3.5. To visualize this more interestingly, we not only plotted the 3D line of the joints with zero but the full skeleton itself so that we can see the new results in a user-friendly way in Fig. 3.11. We can see a few joints dislocated from the top left corner's actions, which corresponds to the 0 value coordinates in the database. This method is more reliable than the one applied in the First Approach (Section 3.1).

With this refinement allowed us to conclude that the dataset contains either joints that are correctly recognized and not zero, or the joints have a sequence from zero from the start until some point, or from some point till the end of the video sample. However, we notice that not every joint different from zero were correctly recognized. They were correctly located, but they might belong to another person other than the one we were tracking. The JSONs have a key that entails a person's ID. Tracking the person ID revealed not reliable enough to select which person we want to focus on during that video sample because in some cases, the person ID would switch from one to another. We also noticed that in most videos, the main person is more horizontally centered than the person in the background.

Given this information, we applied an unsupervised learning method to divide the samples into two groups, keeping in mind that each sample is a person's skeleton. Ideally, each group should be related to each of the persons. For simplification purposes, we first made an average sum of all the skeleton coordinates resulting in a 1D array per video sample, being each value a scalar representation of the skeleton. For 1D data, a usual method to use is the *Jenks Natural Breaks*. But as we felt it would be more accurate to use a center of mass representation in 2D instead of 1D, in that case, the best method to use would be *K-means* method defined in a python library *sklearn*. With the cluster *K-means* method as an expected reliable option to discriminate skeleton frames into two clusters, depending on its skeleton position, we proceeded to encode the skeleton position to a center of mass representation. This representation was achieved by a weighted sum heavier in the hip and hip neighbors joints. This way, we ensure that the k-means method would receive one 2D coordinate point per frame in order to group the points in two cluster centers. Afterward, both are compared, where the most horizontally centered would be selected as the true cluster center. All the frames whose center of mass is closer to that center are frames with the correct person classified, and the others are not.

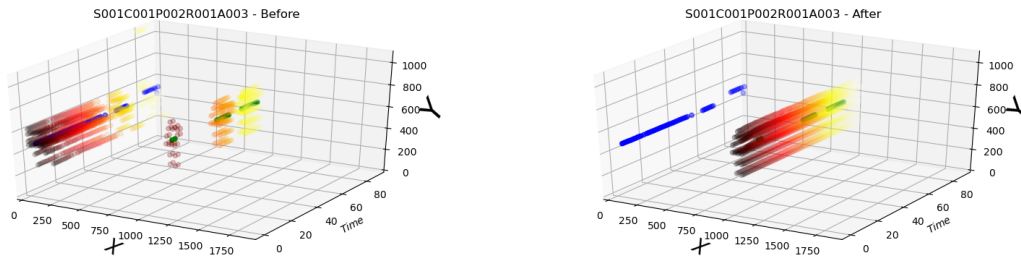


Figure 3.12: Two-Person Process

The next process is constituted by replacing the JSONs where the false skeleton was recognized, and instead of reading the first person in the people array, we read the second. Similarly to the 3D interpolation (Fig. 3.11), this Two-Person Process results can be visualized in Fig. 3.12 where the false and true center of masses per frame is represented with blue and green colors, respectively. In the image on the left, we can see how our dataset had the information stored, and on the right, we can see the differences after this process.

With this second issue solved, we notice yet another problem: the database is imbalanced. Although the number of video samples is the same for each class, the video samples are transformed to JSONs, where the number of JSONs per video sample differs from one to another, even in videos from the same class. To solve the imbalanced database problem is commonly used over-sampling or under-sampling technique. The first is applied to the under-sampled classes, generating more samples of a class so that it isn't an under-sampled class anymore. Opposite to this technique, the Under Sampling is applied to the oversampled classes, where some classes are ignored, resulting in a database more balanced, where if applied in our case, would result in a dataset more frame sample balanced, however becoming video sample imbalanced, due to the video sample length

variability mentioned earlier.

Either one of the two options is too drastic for the database in use since we would need to ignore a lot of video samples from one class or to generate a lot of video samples to achieve an equal enough number of frame samples between classes. Ignoring many video samples from a class might result in too few samples and would debilitate the model concerning the class in question. On the other hand, generating many video samples from a class might result in a bias introduced by generated video samples. We compromised in a middle position, where we put our goal on the average number of class JSON samples. This way, some classes were applied an oversampling technique, and others were used an under-sampling method. We accept that the number of class frame samples will not match in all classes, but the standard deviation will decrease significantly, resulting in a more balanced database.

Opposing the under-sampling technique simplicity, where only ignoring a video sample is required, the oversampling technique is more complex. For that reason, we broke it down in a few steps. First, we wanted to test simple generation techniques. Instead of applying a video sample skeleton sequence generation right away, we converted the skeleton sequence to a center of mass representation sequence. It can be viewed as a line in a 3D space where one axis is the time of an action, and the others are the X and Y space, the x and y coordinates of each dot in the line would represent the center of mass of the skeleton in a moment in time. Then, suppose we apply the same change in video sample representation to all the videos in an under-sampled class. In that case, we have a group of around 900 lines, each one representing a video sample of the class.

One commonly used oversampling approach is a *Synthetic Minority Over-sampling Technique* (SMOTE) strategy. Where it takes an under-sampled class sample, plot lines between all of them, and randomly choose a point in the line to be the new synthetically generated sample of that class. This ensures similarity between the synthetic and original samples of the same class. However, it does not add new information to the model since the new samples are formed from the previously existing ones. Similar to a SMOTE approach, from those 900 lines, we want to generate a new one. The algorithm followed was divided into three steps: 1) Choose one random line, 2) Choose the most similar line to the randomly selected one, and 3) Create a line between the two selected lines. Instead of choosing a random position between two samples, we always choose to be the middle point between both selected lines, keeping in mind that each dot in the line corresponds to one video class frame.

Opposing the simple implementation of the first step, to the randomly selected an video sample out of the 900, the most similar line selection method is more complex. RV Coefficients are a generalization of Pearson coefficients, which is applied in similar situations for 1D data, known as being used for multidimensional data like we are using. They both represent a certain degree of correlation between two data and, therefore, can identify the most similar line to the randomly selected line. We used RV coefficients, which higher the output more similar the two lines would be, keeping in mind that the output ranges from 0 to 1. We choose to perform under and oversample techniques each in groups of three, one per camera set. So, if a video sampled were to be ignored, all camera views of that video were ignored, and analogously when generating a video sample,

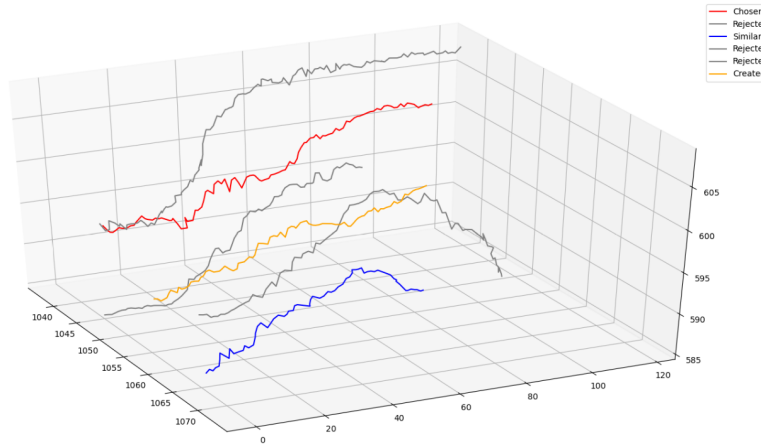


Figure 3.13: Lines Synthetic Generation

three were generated instead, one per camera view.

After choosing the line with the highest RV Coefficients and having the two lines selected, we start to generate the new line. The new line is generated point by point through time, being the new point the middle point between the edges belonging to the two selected lines at a given moment in time. This can be perceived if we imagine that in the Fig. 3.13 where the center of mass video sample representation is presented and axis nomenclature follows the one in Fig. 3.12, we extract the values for the blue, red and yellow lines in the time = 0. The values extracted if represented in a 2D plot, one would notice that the yellow point is in the exact middle point between the red and blue ones. The generation continues until the shortest line ends. As we want to keep the average length of class video samples, we aim for the newly generated lines to have lengths equal to the two selected lines mean lengths. If the existing lines chosen have lengths 2 and 4, the new line would have length 3. For that purpose, the longest line variations are mirrored in the new line until the same has reached its end. For simplification and illustration purposes, in Fig. 3.13 we show this oversampling technique applied to lines, but instead of showing the 900 lines, we show a total of 5 lines. The red line is the randomly selected line. The blue is the most similar line to the randomly selected one, the gray lines are random lines that were not the most similar, and the yellow line is the synthetically generated one. This complete process was then applies using the 15 lines referent to each joint in a skeleton (the two selected actions are the same for the generation of all 15 lines).

One set of 15 generated lines was understood as a skeleton sequence generated sample, and it would be included in the *pytable*. Still, they needed some kind of nomenclature, so we followed the one used in the original video samples, attributing the setup and replication values that did not exist in the previous dataset. First, we attributed the class ID number to the action, and then per each generated video, a new subject was created with a unique ID. As for the camera, since the oversampling process is done in groups of three (one per view), the camera ID was automatically attributed. We needed this nomenclature because to train the model, the train-test splits referenced in the NTU-RGB+D paper will be used: cross-view, and a cross-subject, so then the machine will

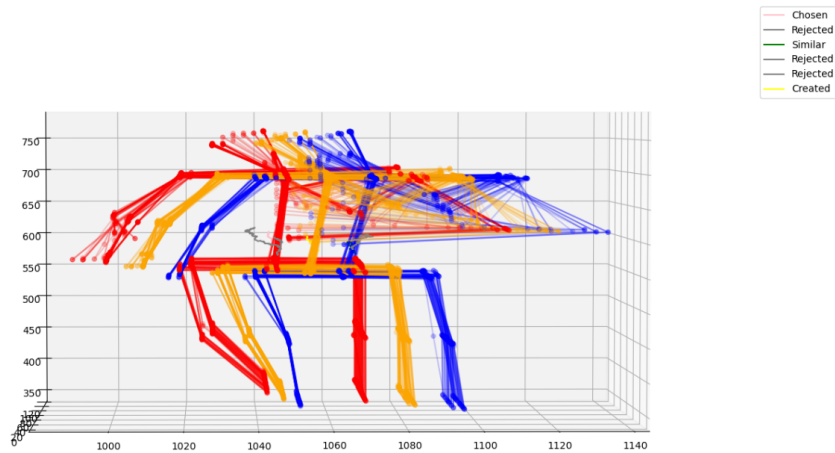


Figure 3.14: Skeleton Synthetic Generation

make decisions based on the action ID classified and the expected ID. Similarly to Fig. 3.13, we show Fig. 3.14, where the color meaning is the same as before, but the randomly selected line, the most similar line to the latter, and the generated line are expanded into full skeletons, with similar variations. At this stage our dataset can be considered balanced with small standard deviation of samples per class, with irrelevant bias introduced by the balance process.

When we were feeding the model, we fed it with data in time windows. Doing this means that there can not be one window with a section from a class and another part from another class, which implicates that the number of windows we can extract from a video sample depends on the length of that video sample. We know that the video samples' length changes meaningfully between classes, so the number of windows per class would be the number of samples per class fed to the model, which would mean the database was still unbalanced since different classes have difference lengths hence different number of samples per class was fed to a model, resulting in an unbalance dataset. The most common method is to pad the class videos to counter this obstacle and achieve a constant video length among all video samples, meaning that the number of windows per class would be constant. We zero-padded all videos till they reach the maximum video length, believing the model would learn to ignore the zeros in actions. Although now having more video samples of the shortest actions, there is also a significant number of zeros, which can be understood as the shortest actions have less valuable information per sample. Still, this lack of useful information is fought with the high number of samples.

After applying the resampling techniques described above we adjusted the training and test sets to incorporate the newly generated video samples. For the Cross-View benchmark, an easy inclusion was performed since there was an equal number of camera views in the new video samples. It was identified in the nomenclature adopted. On the other hand, for the Cross-Subject benchmark, a more detailed analysis was required. As the number of frames per subject in previously existing subjects is much higher than for the new subjects, as conferred in Fig. 3.15, where a clear difference between the number of frames per subjects before and after

representing the disparity between samples from existing subjects and the generated ones. For every class where oversampling was applied, a subject with ID depending on the new video ID was created, i.e., if one action needs to be oversampled by 10 videos, there will be 10 subjects created, one for each video, with an ID of 40, the existing subjects, plus the new video number, resulting in a set of subjects in a range from 41 to 50. If after that class another class needs to be oversampled by 9 videos, 9 subjects shall be created with IDs in range from 41 to 49. This algorithm to attribute a subject to a newly generated video, can be represented, w.r.t frame distribution per subjects, in both Figs. 3.15 and 3.16.

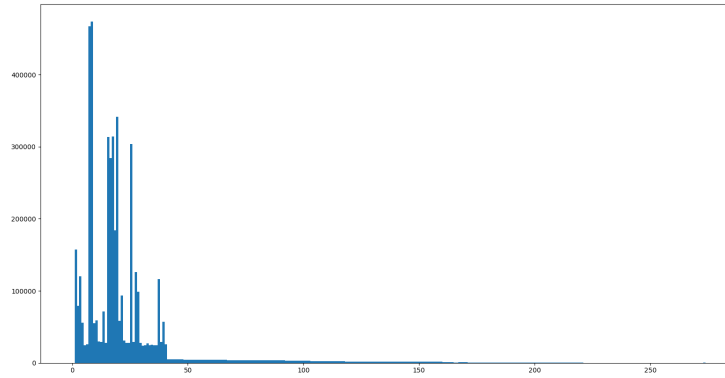


Figure 3.15: Full Subject Frame Distribution

With that in mind, we show in Fig. 3.16 only the new subjects distribution, where we performed a Cumulative Distribution Function(CDF) and scaled it to fit the same image as the histogram distribution of samples per subjects. From the CDF, we extracted the subject where the CDF achieved a 50% value. From that subject backward, the samples were included in the test split, and from that subject onwards to the train split. Since, as verified in Fig. 3.15, the number of frames from the generated samples is very small in comparison to the existing ones, an additional 50% of the total number of frames from the generated samples in each split, train, and test, shouldn't prejudice the model in a meaningful way.

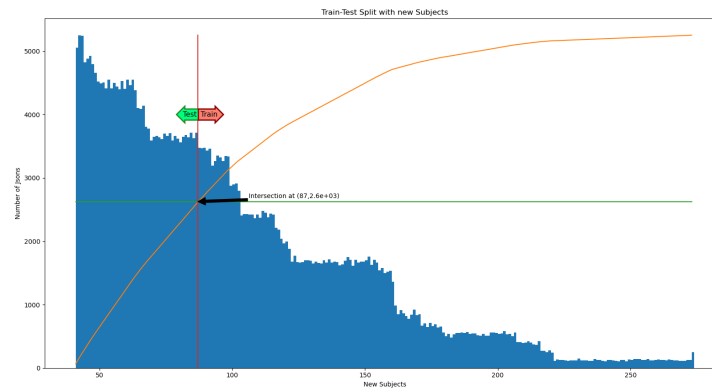


Figure 3.16: Train-Test Split Subject Method

3.2.2 Feature Extraction

In this second approach, our feature extraction always was very similar to the one presented in the first approach (Subsection 3.1.2). Basically, we created geometric features like the ones presented by Zhang *et al.* [22]: *JJO* - "Joint to Joint Orientation", *JJD* - "Joint to Joint Distance", *JLD* - "Joint to Line Distance" and *LLD* - "Line to Line Distance", being the Lines as a custom selection of specific pairs of joints as explained in the first approach in Section 3.1.

Due to the abundance of papers that use a temporal feature such as the movement of the joints, subtracting a future value to a past value achieving the movement of the joint in that time interval, usually being one frame. Given this, we also extracted this type of feature. But instead of just extracting from the *POS* feature, analogously, we also extract the movement of the *JOD* feature. In these cases, it is always essential to notice that if we have a video sample with 100 frames and extract the movement of each joint between two consecutive frames, it would result in a video with 99 frames with information about the joint motion. This difference of dimension length might be significant when testing with and without movement, where the input layer might depend on this number of frames.

One technique we use to solve this difference is passing one frame with zeros and incorporating it in the movement video sequence, where 1 in 100 frames is not a meaningful amount of zeros to be included. Other techniques that could be adopted would be image resize as suggested in Fig. 2.5, which would also be helpful as an alternative to the raw videos pad to achieve an equal number of frames per class, as it will be discussed in the Future Work Section 5.2.

3.2.3 Action Recognition Model

Given that the LSTM-based model presented in the the First Approach (Section 3.1) was not able to successfully learn to recognize actions (more details on this in later Subsection 4.2.1), we chose to change change model, to a CNN-based one. Although CNNs are usually applied to an image, our dataset video samples could also be easily represented as images. The width and channel dimensions would depend on the type of feature. To feed in batches we developed feeder generators. Opposed to the First Approach the train and test split extraction are made inside the generators, instead of previously saving files with the train and test splits, this reduced the disk memory demand a lot.

Our model was analyze regarding Sparse Top k categorical accuracy with $k = 1$, and Sparse categorical cross-entropy loss function with a detailed explanation in Subsection 2.1.5

The architecture of our initial CNN model can be visualized in Fig. 3.17. The input and output grid icon, doesn't resemble the shapes of the actual input and output, the input grid icon means that the input is an image, and the output icon means that the output is a 1D array. In this model the input testes was the *POS* feature. Furthermore, the initial architecture is formed by a sequence of 2 Convolution Blocks of type I and a Classification Block. On the other hand our final model architecture can be found in Fig. 3.18 and is formed by two branches one for spatial features (*JOD*) and another for the temporal feature extracted from the spatial feature (*JOD Motion*). Each branch

is composed by a decouple attention mechanism followed by a convolution block of type II. Both branches are then concatenated, and passed through a Dense layer to keep the initial channel shape as in both featured images, and are then fed to the Main CNN. The output of the main CNN is passed through the Classification Block to produce the desired output. All of these blocks that form each model are explicit in Fig. 3.19 and will be explained below.

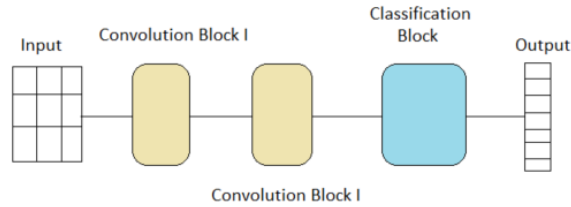


Figure 3.17: Initial CNN model illustration

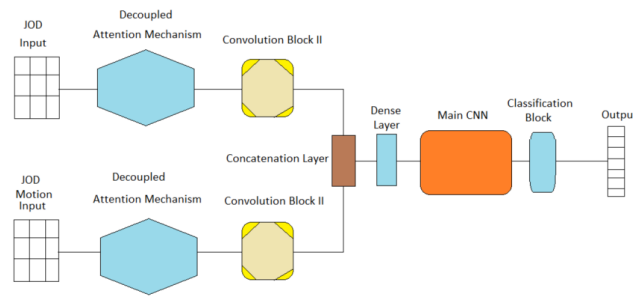


Figure 3.18: Final model illustration

The Convolution Block of type I has is composed by a sequence of 2D Convolution layer, ReLU Activation layer and 2D Max Pooling Layer. After some tests this convolution block evolved to incorporate Dropout 2D and a Batch Normalization Layer, achieving the denomination of Convolution Block II. The Classification Block has the usual Flat, Dense and Softmax activation sequence to produce the output in the form of a 1D array. The Decoupled Attention Mechanism is composed by a Permutation Layer(PL), Dense Layer and another Permutation Layer(PL), where the PLs are to assure that the Dense Layer occurs on the width dimension and not the channel dimension. In the case of POS feature, PLs ensure that the attention is performed on the joint dimension instead of their coordinates, similarly in JOD feature, PLs ensure that the attention is performed on the joint pairs dimension. The attention is performed by the Dense layer, which will produce an output with same shape as input, but with values modified by the weights of the dense layer. Finally the main CNN is composed by 1 Convolution Block of type III followed by 3 Convolution Blocks of type II, where the former is equal to a Convolution Block of type II, but with 2 consecutive 2D Convolution Layers.

To feed the model, we need first to understand the shape of the input we want to give to our model. As mention before, a CNN expects input as an image, i.e., in the format ((H)eight x

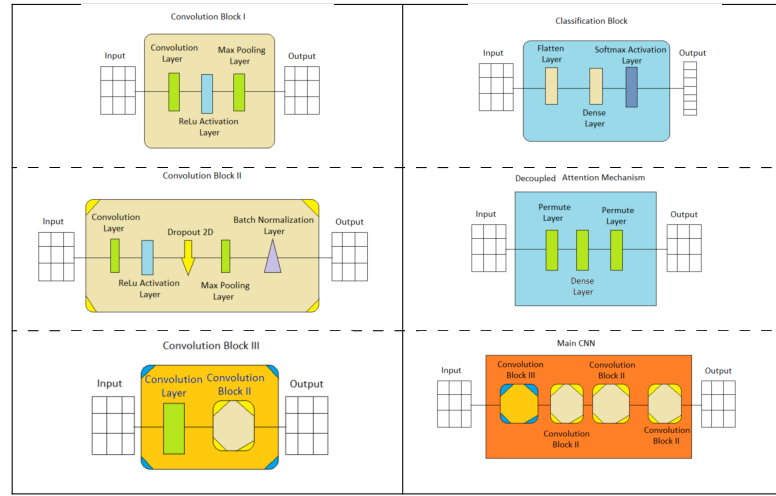


Figure 3.19: Our Model Blocks Illustration

(W)idth x (C)hannels). The height dimension was represented by the video sample length, which is constant in all features due to the padding process, which resulted in a constant video sample length of 300 frames. The width and channels might depend on the features. For instance, the raw features would have a width of 15 correspondent to the total number of skeleton joints and have a channel dimension equal to 2 since there are two coordinates for each joint.

Feature Name	Height	Width	Channels
Raw	300	15	2
JOD	300	210	3
JLD	300	364	1
LLA	300	28	27

Table 3.2: Feature Dimensions Table

On the other hand, *JOD* would have a width of 210 joint pairs, each with three values associated, the orientation in x , orientation in y , and distance. Meanwhile, the *JLD* feature is difficult to place a pivot dimension since different joints might have a different number of lines in which the distance was computed. Therefore we conclude that one possible representation might be with a width of 364 pairs of joints and non-inclusive lines with one channel having the value of the pair's distance. Finally, the feature *LLA* would have a width equal to 28 as it is the total number of lines and 27 channels, one per each of any other line whose angle was computed. A summary of these features dimensions can be found in Table 3.2

Notably, the *JOD* feature has a shape very similar to a usual image. It is not so rectangular as the raw or *LLA* features, where the width differs significantly from the height. Not only that but *JOD* also has three channels just like an RGB image, and since the CNN are proved to work well on RGB image, we felt pretty confident with this feature, and due to the time left, it was unknown if the best option would be to test all the features.

Chapter 4

Experiments and Results

4.1 Data Acquisition

Our dataset was achieved in the processes already explained in Subsections 3.1.1 and 3.2.1, where the RGB videos from the NTU-RGB+D 60 are passed through the OpenPose software, which would extract the skeleton data. But for the purpose of comparing the quality of our generated dataset with the one already provided by the skeleton data modality of NTU-RGB+D, this skeleton data was downloaded. This process also served to verify if our LSTM model’s unsatisfactory results were derived from the dataset or model itself.

Finally, for testing with the live camera feed, a wrapper was created which calls and feeds OpenPose with the camera’s data, getting the skeleton data as a return. We implemented a function to save a JSON with the skeleton in the same way OpenPose saves its JSONs. Since the NTU-RGB+D 60 dataset has different setups and each setup has a different height and distance, we show the Table 4.1. This table is based on the one presented by Shahroudy *et al.* [7], but with the accuracy of the model when fed only with samples from the same setup, to choose a height and a distance for our testing setup.

Setup	Height	Distance	Accuracy
1	1.7	3.5	86.23
2	1.7	2.5	87.39
3	1.4	2.5	89.55
4	1.2	3	84.62
5	1.2	3	84.60
6	0.8	3.5	86.59
7	0.5	4.5	83.89
8	1.4	3	80.93
9	0.8	2	90.02
10	1.8	3	85.01
11	1.9	3	82.52
12	2	3	83.69
13	2.1	3	84.14
14	2.2	3	83.99
15	2.3	3.5	86.82
16	2.7	3.5	84.21
17	2.5	3	81.65
18	1.7	3.1	89.28

Table 4.1: Setup Heights and Distances

The 18 setup is referent to the synthetically generated video samples created from two videos. Therefore we attributed height and distance as the mean of all setups. Since setup 3 and 9 had the highest accuracies, our setup will have a height between 0.8 and 1.4, and a distance between 2 and 2.5. We might conclude that setup 18 is the third best. Still, we have to consider that all the other setups are class balanced, where the synthetically generated videos are class imbalanced. Some classes have more synthetic videos than others, and for that reason, we did not incorporate this setup in choosing the height and distance values, as it was done with setup 3 and 9.

During tests, the computer had three processes running simultaneously, the Camera-OpenPose process, the Reader, and the Evaluation process. The Cam-OpenPose one is responsible for taking the live camera images, feeding them to OpenPose, and writing the JSONs with the skeleton pose recognized in each frame. On the other hand, the Reader process is responsible for reading the JSONs, extract features and send the model input to the Evaluation one. Finally, The Evaluation process is responsible for feeding the model and analyze the response.

The Reader reads the JSONs, and iteratively creates a bidimensional array Width x Channels with the raw data, where the width is the number of joints and the channels both axis coordinates, per JSON. Then it processes the raw data array, switching the *none*s with zeros, and applying the same zero process used in building the database, as mentioned in Subsection 3.2.1. Finally, it extracts the geometric features, resulting in a 2D array where the Width x Channels shape will depend on the feature. Still, in the same iteration, each 2D feature array will be included in a 3D array Height x Width x Channels where width and channels depend on the feature we are testing and height is set to 300, as it is the length of all video samples after padding.

Initially, the 3D array is all set to zeros, and, considering that a 3D array can be understood as an array of 2D arrays where the first (referent to the oldest JSON) is at the top. The last (referent to the most recent JSON) at the bottom, in each iteration, a 2D array, is inserted in one of the positions of the 3D array. In the first, it will be in the first position, the second iteration in the second position, and so on until all the slots in the 3D array have been changed. From that point on, the 3D array will suffer a roll-up operation where the first 2D array will be the last, the last will be the second last and the second will be now the first 2D array. Then the last 2D array will be replaced by the 2D array created in the due iteration. After updating the 3D array in the iteration, the same will be forwarded to the Evaluation process.

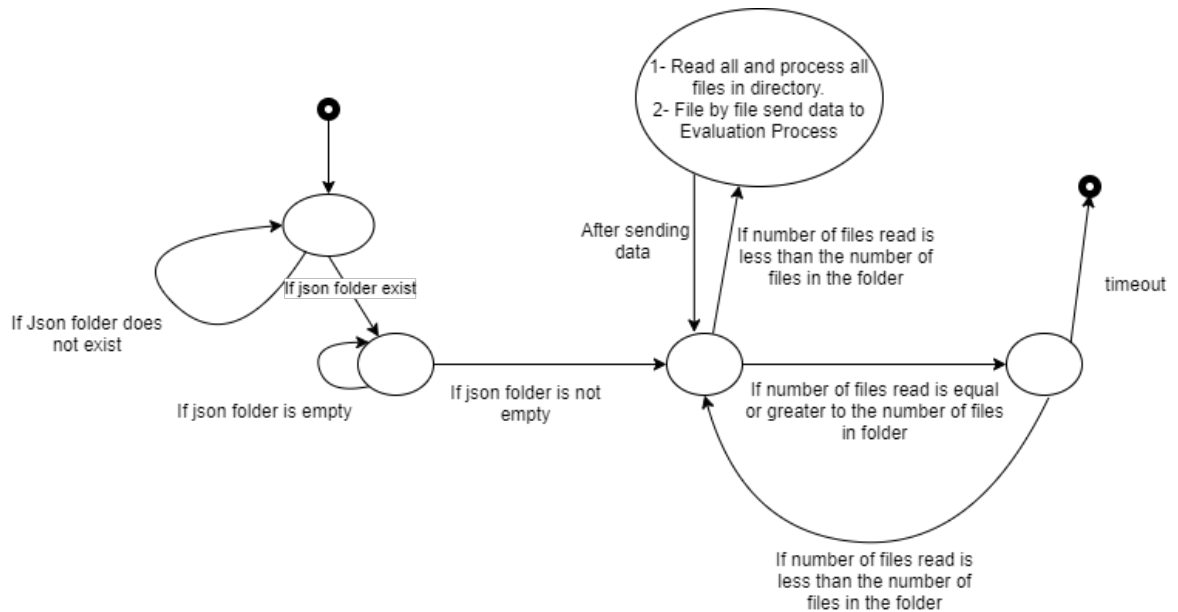


Figure 4.1: Reading Algorithm

The Evaluation process takes the featured image with the 3D shape: Height x Width x Channels, and feeds it to the model, saving the result in an array. If provided and necessary, it can show a visualization of the metrics relating to the actual class and predicted. These metrics might be the confusion matrix, the scatter, or the bar plot. The way we found to assure real-time classification is using different processes for reading and classifying JSONs. The time spent on recognizing pose and saving to a JSON can be considered real-time, and most of the time comes from OpenPose computation, which we cannot improve.

Another step that might take some meaningful time is in how we read the JSONs into memory. The process can be visualized in Fig. 4.1. In this process, the most time-expensive part is when we read and process all the existing files in the folder. However, the JSON reading frequency can be a lot higher than the writing JSONs frequency. The writing frequency is 30 frames per second, meaning that every second one JSON is written. On the other hand, the reading JSONs frequency can read many more than 30 files in one second. Due to this, and based on the writing JSON frequency still be considered real-time writing. One can assume that the reading of the same JSONs is also real-time. The last but obstacle is the model classification frequency when fed full-speed from the Reading process, which is still far higher than the writing JSON frequency. From the same line of thought as before, the classification can also be considered real-time.

4.2 Offline Action Recognition

This section is divided into two approaches, such as in Chapter 3. In the first, we will go over our tests and experiments in chronological order of development, concluding the subsection with a few remarks to keep in mind for the next approach. In the second, we will go through a detailed explanation of the steps taken since the end of the first approach, including the tests after applying the remarks made in the previous approach, as well as the model optimization, wherein the end we present a table to compare our final model with state-of-the-art models.

4.2.1 First Approach

In First Approach, after the process detailed in Subsection 3.1.1 with the model described in Subsection 3.1.3, we concluded that feeding all the features extracted concatenated was not the best option as it had a lot of resource and training time demand. For that reason, we splat the features each for its own machine, changing the input size to the respective feature size, and we tested only with setup 1 to alleviate the resources being used.

Although Shahroudy *et al.* [7] recommend benchmarks for training and test splits, we tried one of our own, which consisted of dividing in a 47 to 1 proportion train-test split. The video samples were agglomerated into groups of 60 each, resulting in 48 groups of 60 actions. In Fig. 4.2, where on the column dimension we have the action IDs and on the row dimension are represented the group IDs. On the other hand, the green squares represent the test samples, and the blue ones represent the train samples. For each group, we take one action with the same ID as the group, but

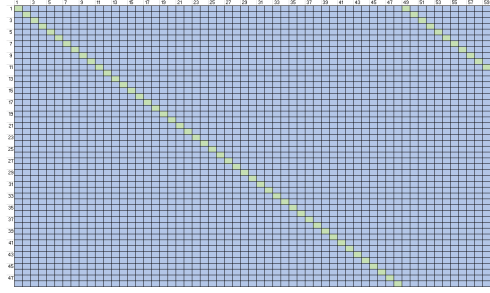


Figure 4.2: Own train test split method

due to the difference between the number of groups and the number of actions, the groups with id 1 to twelve have two test samples.

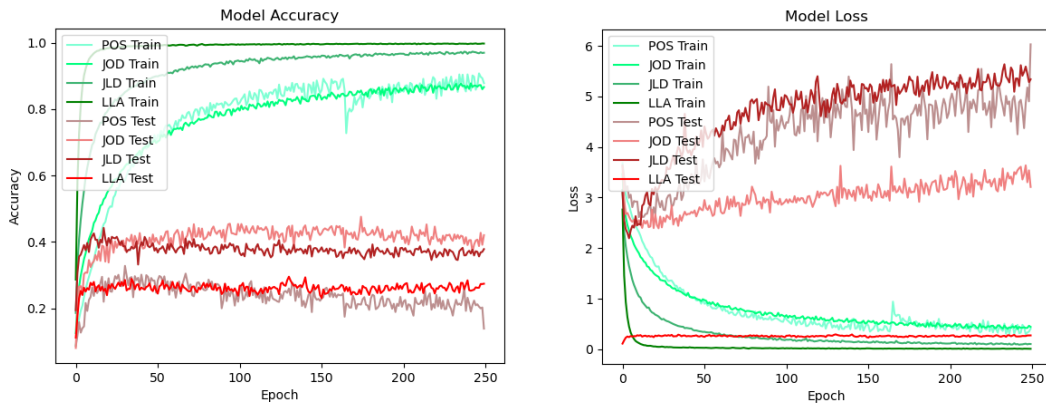


Figure 4.3: First Learning Graph

The results under this split can be visualized in Fig. 4.3. Regarding training results, the *POS* and *JOD* machine show similar accuracy and loss responses, represented with the lightest green color. However, they were surpassed by *LLA*, which quickly achieved excellent values in both metrics, leaving *JLD* in second place in both metrics performance. On the other hand, regarding the test results, the *LLA* and *POS* accuracies are very similar, represented as the darkest and semi-lightest red color, respectively, but still lower than *JLD* and *JOD*, where the later reaches a maximum of approximately 45 % accuracy.

The *LLA* losses are very low and almost constant around 0.3, where the other three seem to be increasing with the epochs, being the *JLD* losses the greatest reaching 5.5. This later loss behavior can be understood as overfitting, and the model starts to learn towards the training data, becoming biased, which is not desirable. Conversely, the losses of *LLA* initially grow and stagnate, which is the opposite of what might be expected. This behavior, based on the fact that this model's training accuracy also quickly rises and stagnates, can also be understood as an overfitting problem since epoch one, which means the model cannot extract information that allows it to classify unseen samples correctly.

With Shahroudy *et al.* [7] in mind, where they also describe the distance and height of each setup, considering the four models we had trained with setup one, we decided to test them with homemade videos with different distances and analyze in which distances the models performs better. For this distance test, we tested all our four models with the video samples of the same action, "Drinking Water," with distances in a range from 2.5 and 4 meters, each video sample with an increment in a distance of 25 cm. In Fig. 4.4, the results can be evaluated. Each sub-image on the Y-axis has the possible action IDs of 0 to 59. On the X-axis, the video length is represented. On the other hand, the color scale from 0 to 1 is the softmax output and can be understood as the model's confidence predicting a sample as an action in a given frame. It can be noted that most of the frames where the model has high confidence in the correct action occur in the middle of the action. Furthermore, it seems that the *POS* based model, almost regardless of the distance, has some inclination to classify the sample as actions around 14 or 21, instead of the expected 0.

Similarly, *JOD* based models seem also to have some dependency towards the action 14 and 21. This dependency tends to fade off as we increase distance. This model also shows a great concentration of confidence in the low action IDs, from middle action on forward. Meanwhile, the *JLD* based model shows a bit of dependency towards action 9 and 21. But the most significant dependency would be action around ID 45, more present in the middle and end of the sample, oppositely to the other dependencies mentioned. Finally, the *LLA* model seems to have more consecutive correct confidences, increasing its length with distance until 3.25m. Regarding its dependencies, one might conclude that the action ID around 21 is a source of the model's dependencies, based on the confidence sequences in this action ID.

To better analyze the action ID dependencies, we must understand what each of those actions is in its nature. Taking in mind that the IDs representation are 0 based and that in Table 2.2 are 1 based, meaning each one starts with different numbers. Besides that, also bearing in mind that close action IDs might correspond to similar actions (as referred at the end of Subsection 3.1.1), we proceed to a brief analysis of the action ID dependencies mentioned above, using the 0 based id numeration. The evaluating action "Drink Water" is composed of three stages, the starting position holding the cup, the second stage rising the cup towards the face, and the third stage of drinking the water resulting in an almost static position for a very brief window of time.

On the other hand, actions around action 21 are "Take off hat/cap", "Cheer up," and "Hand Waving", all of them have the first two stages of drinking water, so that might justify the common dependency around action ID 22. Meanwhile, the actions around 14 and comprehend "Put on a jacket", "Take off jacket" and "Put on shoe", which are some of the longest actions in the dataset, which might mean that models with this dependency are biased to actions with greater length. Finally, the actions around ID 9 and 45 are "Sit down" and "Stand Up", and "Headache" and "Chest Pain", and only occur in the *JLD* model, which learned based on the distances from joints to lines, which one might conclude that these distances are very similar in the action 0 in the beginning, for actions around 9, and in the end for actions around 45.

To make conclusions about which distance performed better and compare it to the expected distance from the paper mentioned, we created Figs. 4.5 and 4.6. The results are based in the

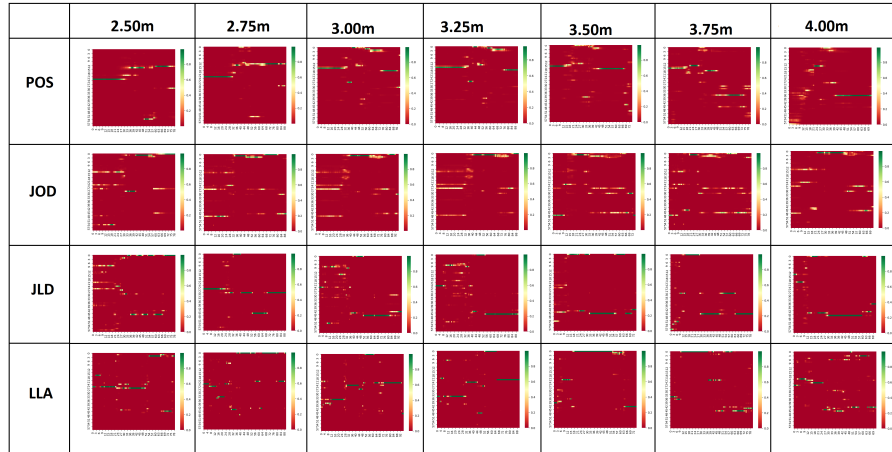
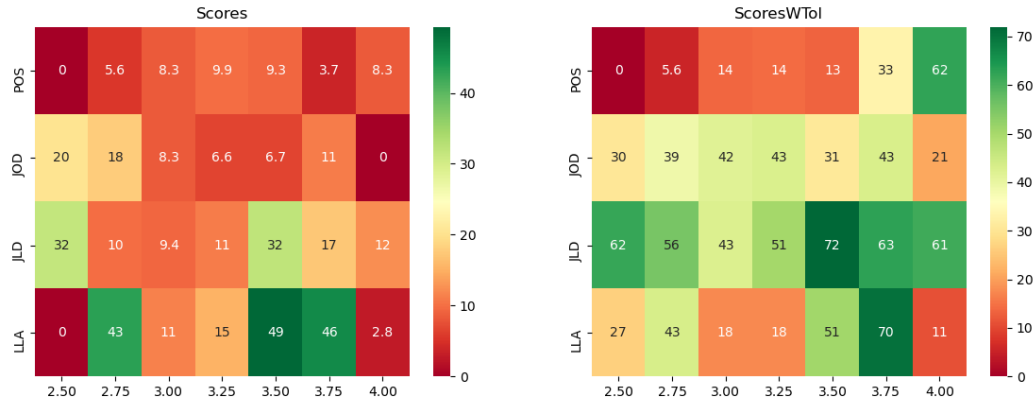


Figure 4.4: Distance Test Table

previous Fig. 4.4 where the videos were recorded with a 0° angle, meaning we faced the camera and based on similar tests with a 45° angle. Both Figs. 4.5 and 4.6 have a pair of sub-figures each. The left ones are the score values without toleration and the right where scores took a certain tolerance into account. The scores can be seen as a percentage of correctly classified samples per total number of samples. On the other side, the tolerance accepts as correctly classified samples if the confidence of the true, correct action is in the top 3 actions.

Figure 4.5: 0° Angle Distance Tests

From Fig. 4.5 we can conclude that the best distance should be between 3.50 and 3.75, based mostly on the results achieved by *JLD* and *LLA*. On the other hand, in Fig. 4.6 we can see either with and without tolerance, the *JLD* seems to perform better in the 2.5m distance, while *LLA* keeps the best performance at 3.5 and 3.75. Although the models suffer from overfitting, and their test accuracies being below 50%, we can draw some conclusions from them, such as *JLD* seeming to be more view-dependent than *LLA* since when we changed the angle, the results in different distances also changed for *JLD*. According to Shahroudy [7] the distance for setup 1 is 3.5m as inferred from Fig. 4.5 and explained above, from which we might conclude that the models *JLD*

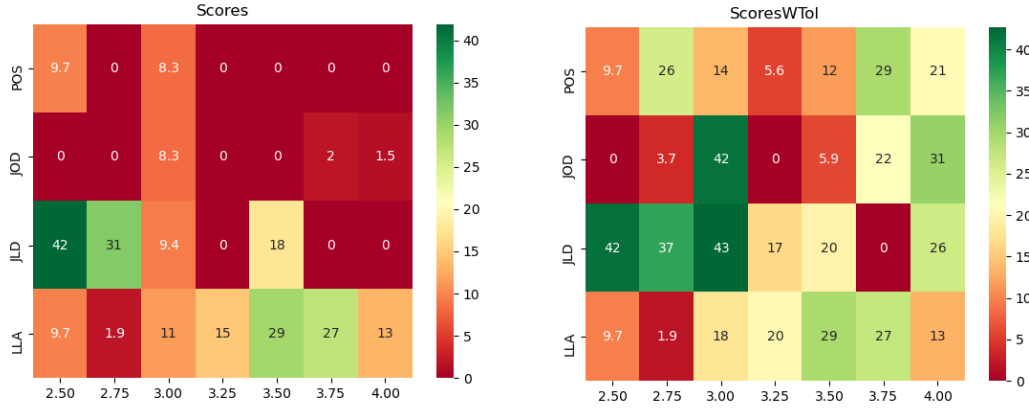


Figure 4.6: 45° Angle Distance Tests

and *LLA* were biased to the correct distance. In further testing, the distances should be based on the setup information provided.

This concludes the First Approach tests. The aspects to be kept in mind for future work is to include all the 17 setups in training and testing, change the train-test split to one of the two recommended in [7], and, as mentioned above, define the testing setup accordingly to the setup information presented.

4.2.2 Second Approach

After the First Approach, we proceed with the tests, starting by including the 17 setups of the dataset and performing all the data preparation methods mentioned in Subsection 3.2.1. Alongside this remark implementation, we also changed the train-test split to the Cross-Subject benchmark mentioned before, focusing on the actions performed only by one subject (1 to 49). All the following tests were performed, first to improve and solve the problems with the LSTM-based model, and then to evaluate the second approach CNN-based model.

4.2.2.1 LSTM-based Model

When we tested the 3 LSTM model with the POS features, the results, represented in Fig. 4.7 were very unsatisfactory. We thought it might have been due to the dataset being imbalanced as described in subsection 3.2.1. In the confusion matrix images where the color bar is not in the 0 to 1 range, they are not normalized by the class support in the dataset, where class support is the number of samples per class, nevertheless it's still possible to extract conclusions that lead to the identification of dataset imbalance as described below.

The accuracy of both testing and training sets rose to around 4%, along with the mostly stagnating loss around value 4. On the other side of Fig. 4.7 we can see a confusion matrix showing that the model predicted the same action for every actual one. The action predicted was always the 13 action ID, which corresponds to action 14 in Fig. 4.8 whose total frame number is high above

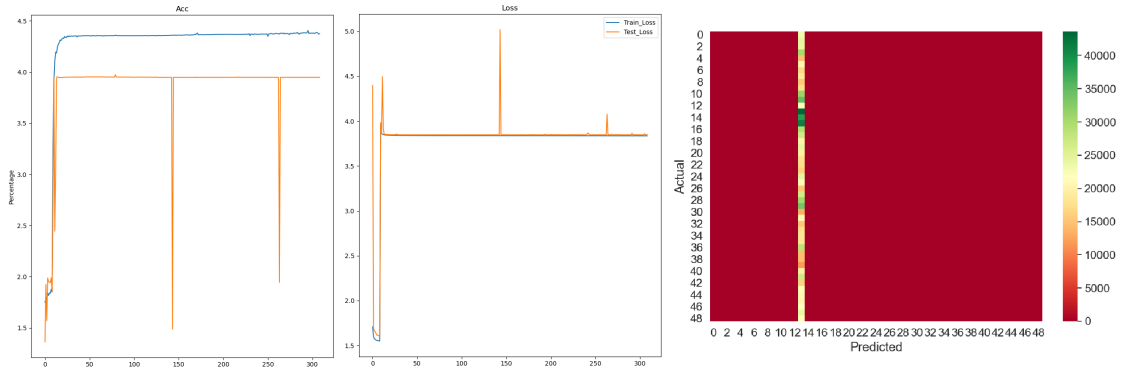


Figure 4.7: Learning Graph of the initial LSTM model (left) and respective confusion matrix (right)

any other action. Since the number of videos sample per class, at this point, was the same, one can conclude that the model is biased for high length actions, specifically to the action "Put on a jacket" accordingly to Table 2.2.

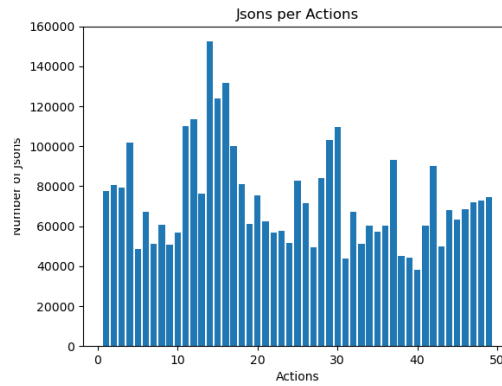


Figure 4.8: Frame samples per class distribution

To fight this problem, we tried two approaches. The first was to implement a loss function that considered the dataset imbalance, attributing certain class weights based on the class support into the loss value computation. The second consisted in applying the resampling techniques mentioned in Subsection 3.2.1. First, we tried to implement the weighted loss function mentioned. The results can be visualized in Fig. 4.9. They can be described as an improvement, regarding the maximum accuracy achieved values around 6% for training and 3% for testing. The loss did not change significantly, the highest difference was noticed in training loss from 4 to 5.

The confusing matrix on the left, when compared to Fig. 4.7, represents a new action being recognized. This connects with the learning graph accuracy plot slope between epochs 20 to 40, can be perceived as the model "unlocking" the new action 15 in a 0 based action ID, which based on Fig. 4.8 is the second class with most samples. One could argue that all the actions might be unlocked with optimized weights, allowing the model to learn effectively and show better results. However, due to the still unsatisfiable accuracies of 6% and 3%, we opted to implement

the resampling techniques.

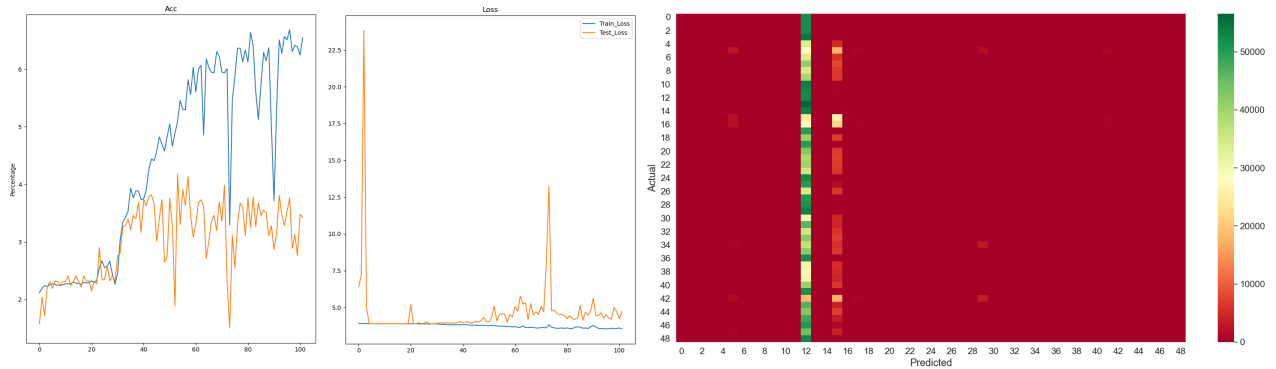


Figure 4.9: Results with Weighted Loss Function Implementation

With the resampling techniques applied and the train-test split updated, we proceeded to train the model with the new database, where the results are presented in Fig. 4.10. It was trained with 60 epochs, and no sign of the slope mentioned earlier was found. Besides that, the accuracy from train and test achieved a value of around 2.5%, and the loss value stabilizes at 4 for both sets. We can see that the confusion matrix is very similar to the one in Fig. 4.7, except for being more evenly green. This is due to the confusion matrix plots not being normalized concerning the actual actions being fed. Therefore the color represents the number of samples classified as the predicted action when an actual action was provided. Since the new dataset suffered more sample-balanced data than the previous one, the color difference between both plots is expected.

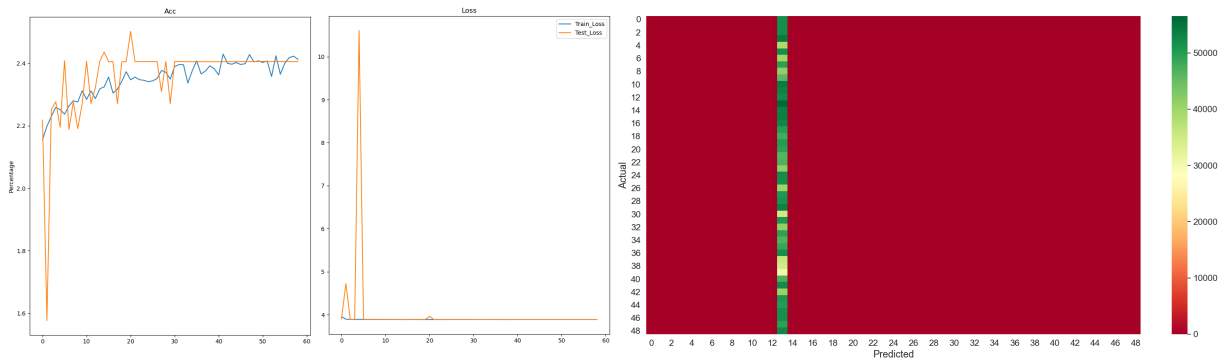


Figure 4.10: Results of the LSTM model after the dataset suffered Resampling Techniques, with a Learning Graph (on the left) and a confusion matrix (on the right)

From these results, we conclude that the main problem was not the database's balance. Therefore we decided to keep the new database since it solved the sample unbalance present in the previous one and focus on improving the model to explore where the problem should be if not in the database's balance.

To further study and improve our LSTM-based model, we performed various tests. First, we tried to alternate the last activation function from softmax to sigmoid, but the results remained the same. Since softmax is recommended to use in a multi-class classification and the sigmoid just

for a bi-class classification, we chose to stand with softmax since we have to classify 49 actions and not only 2. We tried reducing the LSTM hidden size from 256 to 64. However, from epoch 15 onward to the final 30th epoch of training, the test accuracy started to drop from around 2% to 1.5%. Oppositely the loss value of both test and accuracy remained stagnated at 4, so we kept the LSTM hidden size to 256.

Since we weren't achieving any valuable improvements, we started a more "diagnostical" approach, where we tested the model by training it with only one class. We randomly selected two different actions to test separately: action 1 and action 5. Both learn completely to classify the actual action when only the same actual action was provided in training. Similarly, we made two more tests, one where only actions with IDs 1 and 2 were fed during training, and another with only actions with IDs from 1 to 4. The first achieved an accuracy of 50%, and the second achieved around 25%. Alongside this, the losses in both models had a peak in the early epochs (reaching 2.5). From there, they both descended till the end, reaching 2.25 and 2, for the models fed with two and four actions, respectively.

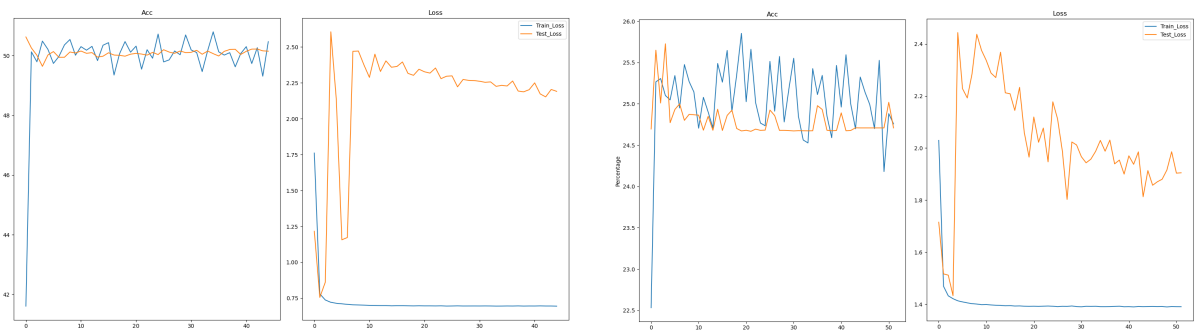


Figure 4.11: 2 and 4 class Training Learning Graph

An interesting characteristic was that although the training accuracy was always close to the test accuracy, the training loss had an ideal loss behavior, although not achieving a good loss value. This is shown in Fig. 4.11 having the models fed with two and four actions on the left and right, respectively. The confusion matrix plots can be visualized in Fig. 4.12, where on the left we see the model trained only with 1 and 2 classes, and in the right the one with classes from 1 to 4. Similarly, as before, all the actual actions were being classified as the same action. Since we can evaluate the type of problem only by feeding two actions where a lot of time is saved during training, we proceeded with the tests only using actions 1 and 2 for training until we find a solution.

The next tests consisted of replacing ReLU with eLU in the LSTM activations function. However very similar ReLU cannot return negative values, but the eLU can (small negative numbers). For big values, eLU returns higher values than ReLU. The test was made feeding only actions 1 and 2 during training and testing. We noticed that the accuracy remained around 50%, and the loss value reduced from around 2.5 to 0.75. It seemed that the test loss was now accompanying the training, instead of getting stuck in a higher values. With this observation, we kept eLU as the activation function for all LSTM layers, although the accuracy remaining at 50%.

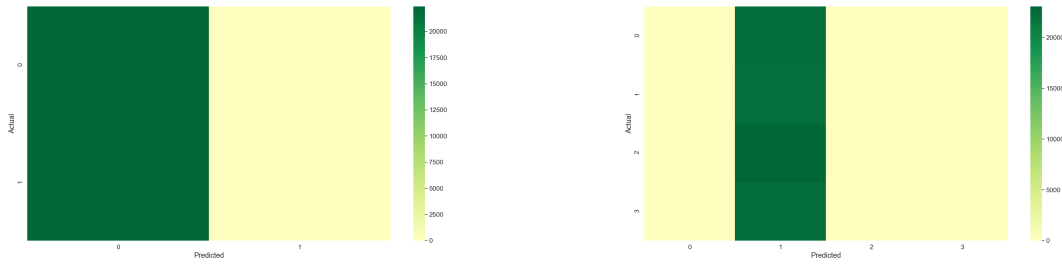


Figure 4.12: 2 and 4 class Training Confusion Matrix

Similarly as before, we tried to change the LSTM hidden size, but instead of decreasing it in half, we increased it to double (512). The results remained the same, but the initial point of the test loss function as it started already as around 0.75 e it was mostly kept around that value. We decided to keep a hidden size of 256 since the results did not show improvements in changing it to 512, so to save the computer resources, a lower number of hidden size was chosen.

To find out if our initial batch size was too big, we tried reducing it to a fourth of what it was, resulting in a batch size of 64. The main difference between the 2 model behaviors was that the loss value of the model with the smallest batch size was a bit higher. With this, we concluded that a batch size 256 was better than 64. A batch size of 512 was too big for the computer resources, so we compromised with what was possible with the computer resources. We tried to train the model with 1, 2, and 3 LSTM layers, and the results did not have many differences. The one with the lowest loss value was the model with 2 LSTMs, so we kept that architecture since it also was a middle point in the trade-off regarding the temporal demand in training and the number of LSTM layers in the model.

In previous models, we were applying Batch Normalization after all LSTM layers, and only a Dropout after the last LSTM layer and before it's Batch Normalization layer. So we tried to change the configuration experimenting only with one or two Dropout, one Dropout, and one Batch Normalization after the last LSTM and finally one Dropout and one Batch Normalization, after each LSTM layer. The results remained the same, with accuracy around 50% and reaching around 0.75 in loss value. We kept the last configured architecture consisting of one Dropout and one1 Batch Normalization layers after the 2 LSTM layers. Since we could not solve the problem with the model by changing its parameters or architecture, and with no useful hint on what to change, we decided to change entirely the model nature.

4.2.2.2 CNN-based Model

As explained in Subsection 3.2.3, we decided to use an CNN-based mode, inspired on the work of Li *et al.* [8] and Strobel Max [24]. The first test was made only by feeding actions 1 and 2. The results are shown in Fig. 4.13, where it can be noticed that the test accuracy reached 85% around the 90th epoch, from where it started to suffer slight overfitting. Where the test accuracy started decreasing, the test loss value started slowly increasing. On the other hand, in the confusion

matrix, we could see that it had started to learn to classify different actions as different predicted actions, which was the goal we couldn't achieve with the LSTM based model.

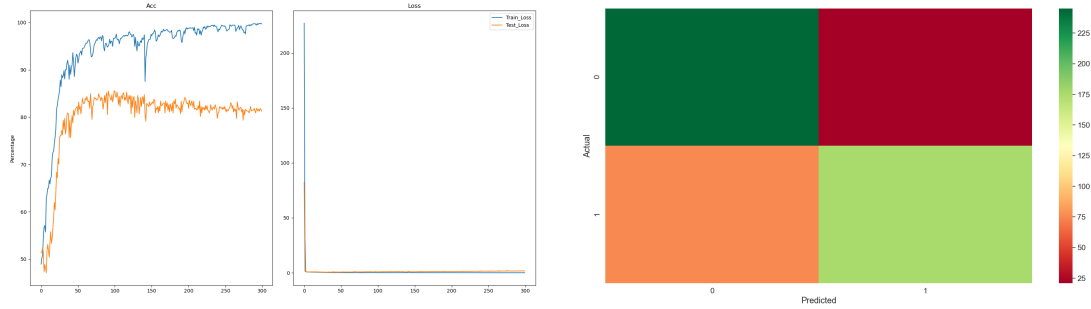


Figure 4.13: Action 1 and 2 Training Results

Motivated by the improvement in the results, we trained the model with all the actions performed by one person, where accordingly to Fig. 4.14, the model reached an accuracy of 15% on epoch 150. On the other hand, the loss value started increasing almost since the beginning, indicating it would still keep growing. In the confusion matrix, we were now starting to see a diagonal line very blurred with the areas around them. We understood this as the beginning of a better model. It needed to be studied and improved since it's architecture was a straightforward one. In sum, we had a considerable improvement in the accuracy value, but the loss behavior was worse compared with the LSTM model results.

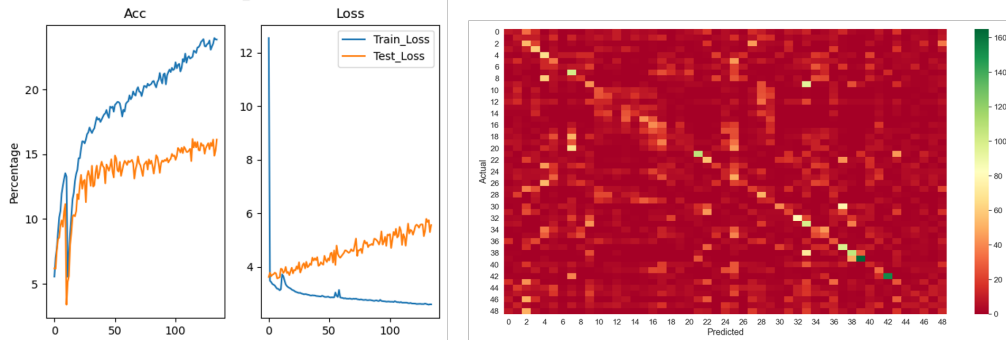


Figure 4.14: CNN Training Results When fed with all 49 actions

To solve the increase in loss, we decided to test small variances in the model architecture defined initially (Subsection 3.2.3). The variations tested first were the number of convolution blocks, where the results summary is presented in Table 4.2. A higher number of CNN seemed to increase the test accuracy of the model. Besides the information present in the table, all the test models suffered from increasing loss value from approximately epoch 20 onward, leaving the four convolution blocks model with lower final loss value than the others. Due to this improvement in performance for the model with four convolution blocks, we chose to proceed with the tests with this configuration.

The next steps consisted of training with different configurations of layer output shapes, where at first all convolution blocks had the same hidden size but the last one, which we decreased to half.

Number of Convolution blocks	stabilized accuracy	stabilized epoch
2	15%	50
3	57%	50
4	61%	40

Table 4.2: Performance test with different number of convolutions blocks

We experimented with always decreasing hidden size, and all equal hidden size. The results are presented in the Table 4.3. The similarity between the normal and similar sized output is expected, based on the initial architecture, only the last convolution layer hidden size was halved. On the other hand, besides the decreasing hidden size configuration, it achieves less test accuracy values. It behaves in a very deformed way from what would be expected from a Learning Graph. Due to these factors, we moved on using the equal hidden size configuration.

Output mode	Accuracy
Original	61%
Decreasing	58 %
Equal	62%

Table 4.3: Convolution blocks outputs

From there, we proceeded to a series of tests, the first consisting of changing the last max-pooling layer output for the width dimension not to be 1, as it was before. The second test implemented an LSTM layer after the last convolution block to see what change it might bring to the model. The third consisted of implementing a simple Attention Mechanism, applied by one dense or a fully connected layer at the beginning of the model. The final test was adding the previously mentioned Attention Mechanism with a Double Convolution Layer in the first convolution block. A Double Convolution Layer refers to two consecutive convolutional layers.

The results are summed up in Table 4.4, where we can notice that with the integration of an LSTM layer, the model's performance drops drastically. On the other hand, including a double convolution layer improves performance in the model using our attention mechanism. Although the table shows the implementation of the attention mechanism degrades the model performance, we kept it believing that eventually, with the model development and architecture debugging, it would improve rather than degrading, as there are many papers using attention mechanisms to improve the accuracy as mentioned in in Subsection 2.2.2.

Inspired by the architecture of the model proposed by Strobel Max [24], where it received one input, and from that, the movement of that input was extracted resulting in two data streams: the Positional input and the Movement input, we decide to work towards this architecture. So first we tested the geometric features and the temporal features alone, then we made our own movement extraction. Finally, we performed the concatenation of both streams on the model. For this test, we performed two concatenation variations, where we will use the POS features as an explanation example, where its shape is a featured image representation with shape [Frame Length, Joint Number, Joint coordinates] = [300,15,2], keeping in mind that the coordinate dimension can also

Architecture	Accuracy
Original	62.47%
Reshaped	59.41%
with LSTM	6.46%
with Attention Mechanism	55.85%
with Attention and Double Conv. Layer	61.18%

Table 4.4: Series of tests, regarding the last max pooling layer, the inclusion of an LSTM layer, an attention mechanism and a double convolution layer

be referred as channel dimension, and the Frame Length dimension as the temporal dimension. From this Pos feature the movement will be extracted and will result in a same sized featured image. The first variation consisted in the concatenation along the temporal axis, where after the concatenation the featured image would have shape [600, 15,2]. For the second concatenation variation, the concatenation will be performed in the coordinate, or channel, dimension, resulting in a shape [300,15,4]. Keeping in mind that in the first tests with themovement being fed as a single stream no padding was applied, and that resulted in a 299 first dimension size in the featured image, oppositely in the following tests a zero padding operation on the bottom of the image, was preformed to keep the dimension as 300. A summary of these test results is represented in Table 4.5.

We notice that *JOD* and *JODMOV*(ement) features perform better than the *POS* and *POSMotion* features in the first block. The table entry test *POSMOV* is relative to when 0 padding was applied to the movement featured image, which achieved a disappointing value of 39.47% for accuracy. Then, although no satisfying results were obtained from the complete HCN model replication, we were still motivated to combine both geometric and temporal features and create a featured image that would be fed to a base Convolution network.

To do so we tried on the *JOD* features, which achieved a 70% accuracy. However, it still was below the accuracies of *JOD* alone and *JODMotion* without feature padding. Furthermore, this test was made with a concatenation of both streams along the height axis, which lead to a more intuitive concatenation along the channel axis. The last change showed to perform better than all other tests, and for that reason, we kept the model architecture with the concatenation of both streams along the channel axis.

4.2.2.3 Parameter optimization

We started by optimizing the convolution layers kernel, where the kernel refers to the filter's shape that a convolution layer will try to recognize in the input image. As we are feeding two input streams (two featured images), one stream from the geometric feature, the *JOD* Feature since the initial featured figure is more square than the other features and it has three channels just as RGB images as mentioned in Subsection 3.2.3 and since the *JOD* featured performed best in Table 4.5. The fact that lead to the conclusion of what feature is more square is the comparison between the Height/Width ratios from each featured image achieved by each geometric feature, where the

Name	Input Image shape	Accuracy
POS	300x15x2	61.18%
JOD	300x210x3	73.42%
POS MOV	299x15x2	61.68%
JOD MOV	299x15x2	76.72%
POS MOV	300x15x2	39.47%
HCN Replication	300x15x2 , 300x15x2	5.11%
JOD + JOD MOTION / Concat Height	300x210x3 , 300x210x3	70.37%
Concat Channels	300x210x3 , 300x210x3	78.13%

Table 4.5: Feature Training

closest the ratio is to 1 more square the featured image will be. The second featured image is a temporal feature: the movement from one sample to another in two consecutive frames (or heights in terms of featured image dimensions) and 0 padded in the end. This temporal featured image is extracted from the geometric feature previously obtained. We started to test different configurations for the kernel size in the convolution layers, applied directly on the two different streams before being concatenated. Both kernel configurations are the same in both convolution layers.

The configuration of the kernel sizes in all other convolution layers is constant through all tests in this test set, and the only difference from our last best model, besides the kernels we are changing, is the last max-pooling layer pool shape, which was 3v1 and now 2v1, resulting in less severe dimensional reduction. The kernels we were changing had an initial shape of 1v1. The test set results can be observed in Table 4.6.

kernel size	epochs trained	accuracy	output shape
1v1	86	76.68%	9v50v8
1v3	79	73.76%	9v50v8
1v5	26	75.46%	9v49v8
3v1	121	74.09%	9v50v8
5v1	36	73.77%	9v50v8

Table 4.6: Stream Convolution Blocks optimization

This table shows that the increase of the first dimension results in a slight drop in the test accuracy. Besides that, we can't extract a conclusion from increasing the second dimension in the filter shape. Therefore we stuck with 1v1 shape in these convolution layers since it had better accuracy. One can also conclude that the effect of increasing a dimension in the filter shape, resulting in a lower dimension in output shape due to the dimension reduction happening in convolutional layers and max-pooling layers, is more notable in the second dimension than in the first.

This second dimension sensibility is because the featured image is longer in the first dimension than in the second. Besides choosing the 1v1 shape in the first convolution layers, we tried to change the last max-pooling layer pool size back to 3v1. Where the accuracy rose from 76.68% to 76.94%, as seen in Table 4.7. Where, since the "Original" model and the "first 1v1 with last

3v1" have the same architecture and were fed in the same way, we didn't know why we couldn't achieve the 78.13% in the second one. We assumed it was due to the randomness intrinsic to the model that made the 78.13% an outlier from what should have been. Since the difference is small, the theory is plausible.

Model	Accuracy
Original	78.13%
first 1v1 with last 2v1	76.68%
first 1v1 with last 3v1	76.94%

Table 4.7: Last max pooling layer kernel size optimization

With the first convolution layers already optimized, we decided to test again with more variability, different geometric and temporal features alone, and combined to the model and compare the results. When feeding in standalone features, the samples don't go through the decoupled branches layers, as they go straight to the Main CNN, based on the Fig. 3.18 discussed in Subsection 3.2.3. The results are visible in Table 4.8. We didn't perform the combination test between *JLD* and *JLD* Motion because the two features alone achieved very low accuracy. The combined features' accuracy is expected to be significantly lower compared to the other features. From the rest of the table, we can conclude that *JOD* based features perform better than *POS*, temporal features from geometric features perform better than just the geometric features regarding *POS* and *JOD* features. Finally, the combination of both geometric and temporal features leads to higher accuracies.

Input	Input shape	Output shape	Accuracy
POS	300x15x2	6x1x8	27.89%
JOD	300x210x3	6x50x8	65.25%
JLD	300x364x1	5x58x32	9.39%
POS Motion	300x15x2	6x1x8	59.73%
JOD Motion	300x210x3	6x50x8	69.64%
JLD Motion	300x364x1	5x58x32	4.12%
POS + Motion	300x15x2 , 300x15x2	6x1x8	28.64%
JOD + Motion	300x210x3 , 300x210x3	6x50x8	74.40%

Table 4.8: Features Comparison

This test reinforces our conclusion mentioned before regarding the 78% model being an outlier from the model's possible accuracy with the same architecture. This reinforcement is because the architecture of JOD+Motion is the same as the ones used previously as "Original" and "first 1v1 with last 3v1" in Table 4.7, where we can reinforce the above point, based on our three samples, about our model average accuracy being 76.49% with a 1.55% as a standard deviation. Based on the best accuracy, we chose to use JOD feature combined with its motion to be the model's input.

After all those tests, one thing that still needed optimization was the Core Convolutional Network's kernel sizes. Reminding that the structure of that core is a sequence of Convolutional(C), Activation(A), and Max Pooling(MP) layers, the sequence can be illustrated, from left to right, as

the following: C-C-A-MP — C-A-MP — C-A-MP — C-A-MP — F-D

Where the three consecutive hyphens are just for illustration of the model structure as blocks CAMP or CCAMP. We performed six ordering configurations of the kernel and pool sizes for both convolution and max-pooling layers. Table 4.9 we have one column for each convolution and max-pooling layer in the core CNN numbered accordingly to the model's position. The smaller numbers are closer to the input, and the other two comparative columns are the best test accuracy of the model and the epoch on which it was achieved.

Name	C1	C2	MP1	C3	MP2	C4	MP3	C5	MP4	Acc	Epoch
Order1	5v2	5v2	3v2	4v2	2v2	2v2	2v1	5v1	3v1	74.40%	24
Order2	1v1	2v2	2v2	4v4	3v3	5v5	3v3	5v5	3v3	2.22%	1
Order3	2v2	2v2	2v2	4v4	1v3	5v5	1v3	5v5	2v3	27.92%	7
Order4	2v2	2v2	2v2	4v4	3v3	5v5	4v4	5v3	2v1	49.94%	5
Order5	2v2	2v2	2v2	4v4	3v3	5v5	3v2	5v3	2v1	66.05%	8
Order6	2v2	2v2	2v1	4v4	3v1	5v5	3v2	5v3	2v3	77.33%	20

Table 4.9: Core CNN Optimization

Besides Table 4.9 information about kernel shapes configuration, maximum accuracy, and the epoch on which it happened, we also present the combined Learning Graph with the learning behavior of each of the models through its epochs in Fig. 4.15. At first, we might strange the low epoch on which some models achieved maximum accuracy (Table 4.9), but when looking at its behavior and compare it with the other models, in Fig. 4.15, we see that there is an optimal curve line which rises fast in the beginning and stabilizes in the end, regarding the accuracy plot.

On the other hand, we expect the loss plot to be a line that decreases drastically and then starts to stabilize. If we look at the small length lines, we can see that their accuracy would stabilize in a very low value, and the expected loss behavior with that accuracy evolution is expressed in the loss plot. Due to this prediction of bad performance, some models were stopped earlier than others. Paying attention to the loss plot we can see that after epoch 20 the best accuracy models loss seems to begin to slowly rise, meaning it might be overfitting from that epoch on.

We choose the architecture of Order6 since it had the best results and went on to experiment the inclusion of Batch Normalization. We made study about how many Convolution blocks should have the Core Convolution Network which is illustrated in Table 4.10. The goal of this test was to be sure that the previously selected four convolution blocks were still the best option. Furthermore now we compare it to a higher and a lower number of blocks. From this table we were able to conclude that the incorporation of batch normalization increased accuracy. Furthermore we concluded that four and five convolution blocks are better than three convolution blocks, and five convolution blocks is slightly better than four convolution blocks. Since all the optimizations were made regarding four blocks, and the difference between number of blocks was very small, we choose to keep the four convolution blocks.

Finally we tried two different types of dropout, both with different positioning in the architecture, in relation to each Max Pooling layer. The 1D Dropout is performed after Max Pooling layer,

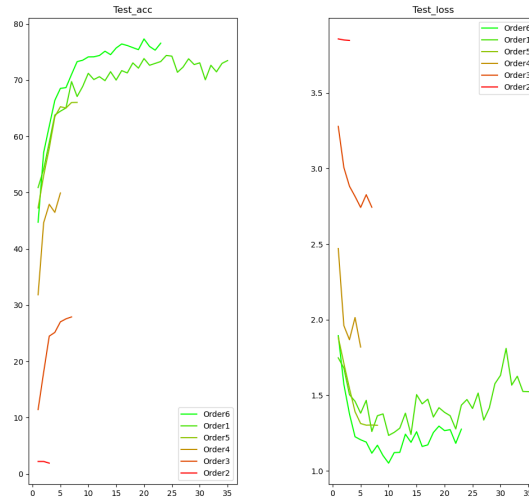


Figure 4.15: Combined Learning Graph of tests presented in Table 4.9

N° Conv	Accuracy	Epoch on max	Total Epochs
3	78.95%	5	16
4	79.38%	13	19
5	79.74%	14	18

Table 4.10: Number of Convolution Blocks, second test

whereas 2D Dropout is placed after the Max Pooling layer. The results are present in Table 4.11, where the tests with different convolution blocks were remade. From the top part of this table we may conclude that Dropout2D outperforms the other options, and the 1D Dropout, represented as "Dropout" in the table, decreases accuracy. Furthermore in the bottom of the same table, one might conclude that four Convolution blocks are the best option to go however very close to five and 3 blocks (this latter the one with less accuracy).

Type	Acc	Epoch on Max	Total Epochs
noDropout	77.14%	18	21
Dropout	41.77%	57	128
Dropout2D	82.05%	78	83
Dropout2D 3Conv	80.56%	61	61
Dropout2D 4Conv	81.93%	76	83
Dropout2D 5Conv	81.60%	113	115

Table 4.11: Dropout Inclusion Test and Number of Convolution Blocks corrected

Next, we wanted to explore the hidden nodes' variability. In previous models, we were using 16 as a hidden node seed. This hidden node is a parameter of the convolution layers, which specifies the desired output channel size of a given convolution layer's featured output image.

The test results can be found in Table 4.12, where we conclude that the hidden size 32 performs slightly better than the other two tests performed regarding the max accuracy value. However, the loss value is higher, so we kept the 16 seed since it performed better than the 8 seed with a minimal difference in loss value.

Hidden size seed	Accuracy	Loss	Epoch on Max	Total Epochs
8	81.01%	0.71	117	134
16	82.41%	0.73	122	159
32	82.81%	1.00	155	131

Table 4.12: Convolution layer hidden size optimization

To conclude the model we performed two last tests: one without attention mechanism, and one with attention mechanism and the view train-test split from Shahroudy *et al.* [7], opposed to the subject train-test split we've been using. The test results in Table 4.13 helped to conclude that the lack of attention mechanism prejudices the model, as suspected in the analysis of Table 4.4 in Subsubsection 4.2.2.2. Not only that but also allows to conclude that as expected from other papers in Table 2.3 the model performs better on the Cross-View benchmark. After these tests, all the following tests were performed with the cross-view benchmark.

Name	Acc
no Attention	74.97%
View Split	85.99%

Table 4.13: Lack of Attention mechanism and different Evaluating Benchmark results

To summarize the tests we present Fig. 4.16, where the orange color resembles the LSTM based model and the blue color the CNN based model. On the other hand, the circle markers denote models being fed with 49 actions, and the triangle markers the ones fed with two actions. The Y-axis and X-axis resemble the model's test accuracy and the chronological enumeration of our model benchmarks. This chronological evolution can be viewed as a first struggle in the LSTM based model as discussed in Subsubsection 4.2.2.1 and visualized as the first four orange markers, followed by a more "diagnostical" approach by testing with only two classes, where we decided to start using CNN, illustrated as the markers between the X-axis range from 4 to 7. With a significant improvement in the CNN model, we proceed to optimize it, as discussed in Subsubsection 4.2.2.2, first with no significant improvements, noted as the markers between 9 and 12 X-axis coordinates, until we included the batch normalization and entered a more steady improving phase, ending in the model Cross View model.

To conclude this section we present Table 4.14, based on Table 2.3. This table allows us to compare our approach to the state-of-the-art accuracy values. We position ourselves in the top 4 of the presented values with respect to Cross-Subject benchmark, and in the bottom 4 of the Cross-View benchmark, keeping in mind that Shi *et al.* [5] present two models, where the best also relies on RGB information.

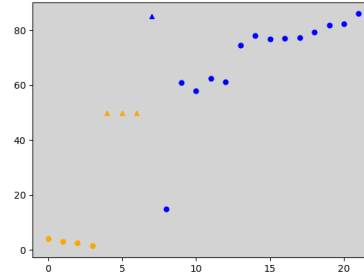


Figure 4.16: Model Evolution Plot, LSTM(orange) and CNN(blue) based models accuracy chronologically and w.r.t the number of actions fed, either 2(triangle) or 49 (circle)

Name	Model Description	CS Acc	CV Acc	Year
Sharoudy et al. [7]	P-LSTM	62.93%	70.27%	2016
Zhang et al. [22]	ST-LSTM	70.26%	82.39%	2017
Song et al. [3]	STA-LSTM	73.40%	81.20%	2016
Zhang et al. [4]	VA-LSTM	79.40%	87.60%	2017
BiA-CNN (ours)	BiA-CNN	82.81 %	85.99 %	2021
Li et al. [8]	CNN+Motion+Trans	83.20%	89.30%	2017
Shi et al. [5]	MS-AAGCN	90.00%	96.20%	2019
Shi et al. [6]	DSTA-NET	91.50%	96.40%	2020
Shi et al. [5]	RNX3D101+MS-AAGCN-C	96.10%	99.00%	2019

Table 4.14: Models Performance

4.3 Live Action Recognition

4.3.1 Setup

With the training and offline tests completed, the model needed to be deployed and the framework tested in a live manner, directly from the camera feed. To start these tests, first, we needed to fix a height, a distance, and an angle for the camera setup. The first two are fixed in each setup of the dataset, accordingly to Shahrudy *et al.* [7], so we tested the model with each setup to see which would have higher accuracy. The results are presented in Table 4.15

Setup 18 represents all the synthetically generated samples, which has data generated from all the other setups. Given this, its height and distance were set as an average of all the other different setups' values. We extracted the two best from the accuracy achieved in each setup, ignoring the third, as it had height and distance values attributed by us. The two best were setup 3 and 9, with accuracies of 89.55% and 90.02%, respectively. In conclusion, we wanted our setup to have a height from 0.8 to 1.4 and a distance to subject between 2.5 and 3.5m. With the room's resources, we managed to achieve a height of 1m and a distance of 2.5m as illustrated in Fig. 4.17.

To choose the camera angle, we looked to the cross-view split from the dataset, which trains with camera views 2 and 3, and evaluates camera 1 (0° camera). In the dataset, the subjects were asked to repeat each action two times, one facing camera 2, the other facing camera 3. This means that to achieve better results, when choosing camera 1 with a 0° angle, one must perform the live

Setup	Height	Distance	Accuracy
1	1.7	3.5	86.23%
2	1.7	2.5	87.39%
3	1.4	2.5	89.55%
4	1.2	3	84.62%
5	1.2	3	84.60%
6	0.8	3.5	86.59%
7	0.5	4.5	83.89%
8	1.4	3	80.93%
9	0.8	2	90.02%
10	1.8	3	85.01%
11	1.9	3	82.52%
12	2.0	3	83.69%
13	2.1	3	84.14%
14	2.2	3	83.99%
15	2.3	3.5	86.82%
16	2.7	3.5	84.21%
17	2.5	3	81.65%
18	1.7	3.1	89.28%

Table 4.15: Setup Configuration Accuracy

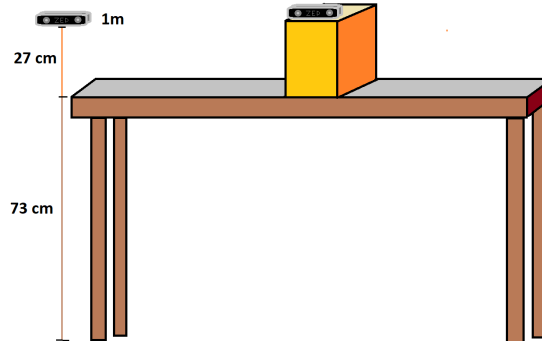


Figure 4.17: Room Setup

feed actions facing deviated 45° degrees from the existing camera, for any side. The room had three illumination sets, no light, half lamps on, and full light, where for testing, we chose full light. During tests, no body parts were occluded except for the action 'cheers', 'jump' or 'drop' where some parts or the full-body might not be captured in the camera range. Objects were used to perform some of the actions to recreate the motions for that specific activity correctly. This concluded our setup for the live tests, allowing us to initialize the experiments.

4.3.2 First Tests

We started by testing the model with live recorded videos for all 49 actions. For each class, five videos were recorded, achieving the results illustrated in Fig. 4.18 and summed up in Table 4.16

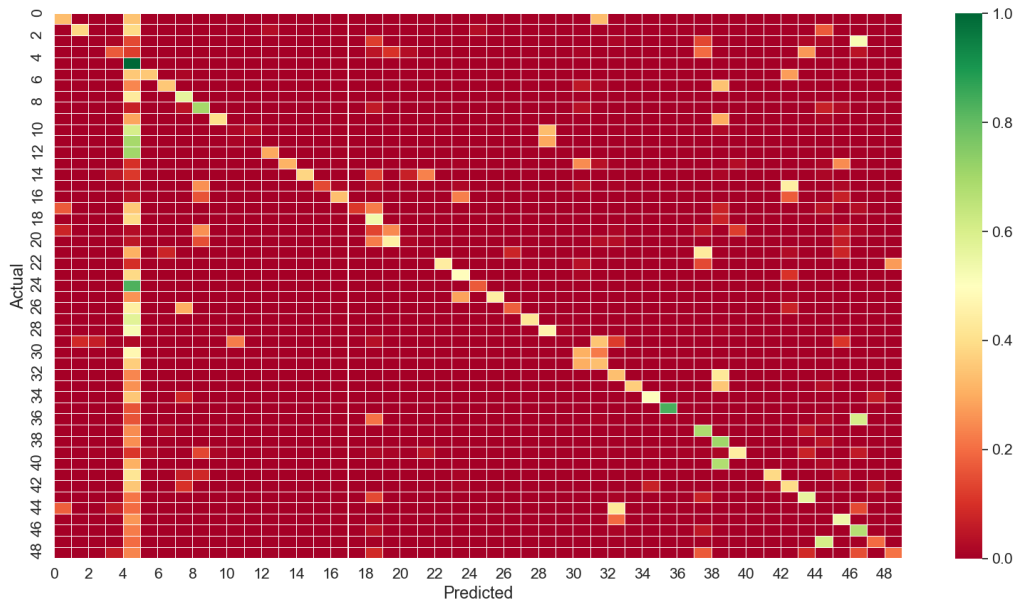


Figure 4.18: Single test confusion matrix

From Fig. 4.18 we can see that some actions weren't recognized from almost any input sample. This can be understood by observing the red lines in the image. For example, from vertical lines in predicted action 2 or 11, we can see that they are most likely classified as another. In the case of action 2, it is misunderstood as action 4 or 46, visualized by spotting non-red rectangles, in the line of actual action 2. Placing names in the ID numbers, the actions of "Brushing Teeth" is mislabeled as the action "Drop" or the action "Touching neck," where the latter is an expectable error, since both actions are performed similarly (both start by rising the hand towards the head, followed by performing some rotating hand movements).

On the other hand, we can see that the model is more biased towards action 4, "Drop," in the sense that for every action fed, there is a significant chance that action 4 will be classified. This conclusion is extracted from the non-red vertical line in the mentioned confusion matrix.

Similarly to action 2 and 46 misclassifications, we can also visualize a more close confusion between actions 19 ("Put on hat/cap") and 20 ("Take off hat/cap"), and actions 30 ("Pointing to something with a finger") and 31 ("Taking a selfie"). This kind of confusion can be tolerated since the first two actions are very similar since they are opposite, almost symmetric actions. Usually, opposite actions tend to be similar, such as putting and taking off glasses. If we inverted one in time to a human, the result would be the opposite action. For that reason, these confusions might be acceptable. It is also clear that the confusion between actions is not always symmetric. For instance, if action A is very misunderstood as action B, that does not mean that action B will be misinterpreted as action A. That lack of symmetry can be understood as the lack of a horizontal line in action 4, so it would be symmetric to the vertical one.

Regarding the general misclassification of action 4, "Drop", it might be due to the constant skeleton position when dropping something. The action consists of opening the hand and letting

Name	F1-Score	Frame Score	Confidence
A036_Shake head	93.70%	88.94%	91.20%
A009_Standing up (from sitting position)	70.51%	73.32%	2.97
A023_Hand waving	62.12%	49.54%	0.30&
A040_Cross hands in front (say stop)	60.82	64.50%	1.51%
A026_Hopping (one foot jumping)	60.77%	44.79%	-1.80%
A035_Nod head/bow	60.31%	50.60%	-1.37%
A028_Phone call	59.68%	42.53%	-1.90%
A010_Clapping	57.14%	40.0%	-2%
A042_Staggering	55.29%	38.84%	-1.09%
A015_Take off jacket	55.17%	38.10%	-2.06%
A044_Rub two hands together	53.33%	56.12%	-1.29%
A004_Brushing hair	35.84%	17.48%	-0.30%
A005_Drop	28.57%	100%	-1.88%
A021_Take off a hat/cap	2.56%	1.49%	25.90%
A011_Reading	3.70%	1.41%	-0.06%
A037_Wipe face	0.00%	0.00%	-1.30%
A022_Cheer up	0.00%	0.00%	-1.78%
A012_Writing	0.00%	0.00%	7.54 %
A041_Sneeze cough	0.00%	0.00%	1.41%
A030_Typing on a keyboard	0.00	0.00%	2.87%
A045_Chest pain	0.00%	0.00%	0.16%
A003_Brushing teeth	0.00%	0.00%	-0.51%

Table 4.16: Single Action Table

something drop, or someone carrying something and let it drop. Given this, the skeleton position would almost not change, except for opening hands or small movement of the forearms. However, these slight movements might not be recognized since the joints are represented as 2D coordinates. The skeleton recognition could not have granularity enough to represent these motions reliably. So the dropping action probably, to the model, resembles the subject doing nothing notion. In almost all actions, there are parts where the skeleton remains still, most commonly in the beginning/end of action, and we think this contributes a lot to the misclassification of this action.

In Table 4.16 we show the top 11, 2 middle, and bottom 9 actions in regard to their F1-score, where we can also see their Frame Score and a Confidence score. While the F1-score metric was already explained in Subsection 2.1.5, the frame score is the percentage of samples from a given class that were classified correctly, or in a more practical way, the percentage of JSON files correctly classified for a given class. On the other hand, the confidence score is more complex to achieve, and it was created with the goal of differentiating the frames with 0% in the other scores.

To calculate the confidence score, for every JSON file (sample), the softmax output was analyzed. The average of all actions' weights was extracted (except for the expected action) and later subtracted to the predicted action's weight, resulting in a number from -1/48 to 1. This max value of 1 means that the model is highly confident that the classified class is correct. In the opposite limit, -1/48 means the model is highly confident that that isn't the desired action.

For the high confidence scenario, the model would attribute a 100% weight to the correct class and 0 for the others. The confidence values would be $100 - 0 = 100\%$. In the opposite scenario, where the model weight to the correct class is 0, the value that will be subtracted to these is the rest of the actions' average. Since we know that the total of the remaining weights has to be 1 and that there are 48 actions besides the correct one, it results in a minimum limit of $-1/48$.

The Confidence, Frame, and F1 -score values were calculated for every video sample and then averaged accordingly to each class. We can see that the "Drop" action has an F1-score of 28.57%. However, when we feed the "Drop" action, the model is excellent at classifying it as the action it is. When feeding other actions, it will sometimes classify the majority of actions as "drop." This behavior leads to a F1-score of 28.57%. On the other hand, the "Nod Head/Bow" action has a 60.31% F1-score, and in the confusion matrix, we can see that this action is basically just misclassified as the drop action but is correctly classified more times than misclassified when the action is fed, as it can be concluded from the Frame score.

4.3.3 Combination Tests

We then performed tests for the combination of actions to perceive if our framework behaved well in more complex scenarios. For that reason, we choose three from the top 4 best actions, one action from the 0% frame score, one from the bottom where the frame score wasn't 0%, and three special actions including "Stand Up", "Drop", and "Brushing hair". The "Stand Up" action was considered special due to the fact that it couldn't be performed as a middle or end action. If we start the activity standing up, we can't stand up. On the other hand, the "Drop" as mentioned earlier, is unique due to the model's bias.

Finally, the less special one is Brushing Hair, which it was strange to see so far below in Table 4.16 from "Brushing teeth", since they are such similar actions. The rest of the actions can be found in Table 4.17 where is also presented the single F1-score as in Table 4.16, and the combo F1-score, where the score is computed based on the results of the combo videos. This table can be understood as an analysis of the behavior of the combo videos' individual actions.

Name	F1-score single	F1-score combo
Shake head	93.70%	16.83%
Standing up (from sitting position)	70.51%	82.17%
Hand waving	62.12%	14.72%
Cross hands in front (say stop)	60.82%	26.87%
Brushing hair	35.84%	5.13%
Drop	28.57%	5.23%
Reading	3.70%	0.00%
Brushing teeth	0.00%	2.48%

Table 4.17: Comparison of F1-Score for Combo and single tests on the selected actions

F1-score is obtained from two arrays, one having the predicted label and another for the expected label. At first, we were not able to spot the exact frame an action should be started to be

recognized as the next. For that reason, and to automate the process for all combo-videos, we estimated the average action duration based on the combo-videos duration and the number of actions included in the same videos. Then from each estimated action's duration in a combo video, a list of the same length with the correct label was created according to the combo-video name. For instance, if the estimated duration was around 60 frames, it would mean that if a combo video had actions A, B, and C performed chronologically, there was a total length of 190 frames. The expected label array would include 60 positions full with A label, followed by another 60 with B label. To the rest of the frames, the C label was going to be attributed, resulting in 70 positions in the expected label array.

From Table 4.17 one can also draw some conclusions. For instance "Reading" is less recognized when in combination with others actions. In fact, as default for almost all the other actions, the F1-score decreased, with exceptions to "Brushing Teeth" and "Standing Up". "Brushing Teeth" only could rise in F1-score or stagnate at the previous value. Therefore the rise of the score to 2.48% isn't surprising, but on the other hand, "Standing Up" had a significant increase in F1-score. As another point of view, in Table 4.18 we show the Frame accuracies of each combo video regarding the two arrays described earlier, one with the expected labels and one with the predicted ones.

Name	Frame Accuracy
Stand-Cross	83.13%
Stand-Drop	66.0%
Stand-Drop-Cross	48.94%
Stand-Cross-Drop	46.56%
Cross-Drop	38.17%
Drop-Cross	18.25%
Cross-Shake-Wave	18.72%
Shake-Wave-Cross	15.71%
Cross-Wave-Shake	14.12%
Shake-Cross-Wave	11.84%
Wave-Shake-Cross	5.81%
Wave-Cross-Shake	5.66%
BTeeth-BHair-Read	14.37%
BTeeth-BHair	3.68%
BTeeth-Read-BHair	1.81%
BHair-BTeeth-Read	0.0%
Read-BTeeth	0.0%
Read-BTeeth-BHair	0.0%
Read-BHair	0.0%
BHair-BTeeth	0.0%
BTeeth-Read	0.0%
BHair-Read	0.0%
BHair-Read-BTeeth	0.0%
Read-BHair-BTeeth	0.0%

Table 4.18: Combo Videos Frame Accuracy

We separated the last table into three parts. The top part is regarding the tests with some special cases such as "Standing Up" and "Drop" actions. The "Standing Up" action must be performed first because the "Drop" and "Cross hands" actions are always performed up on two feet, and the "Standing Up" action starts sitting down. Since, in combination tests, the "Standing Up" is always the first action, the model would recognize well that first part of the combined action, due to the incremental feeding algorithm where the first action, of a combined action, at the last frame of that single action will produce an input, for the model, very similar to the input produced in the standalone version, where the "Standing Up" was classified as second best class, in F1-score. Furthermore the other actions in combination with "Standing Up" include "Cross hands in front" and "Drop" actions, where the latter achieves 100% Frame Score and the former a not so notorious but still high Frame score of 64%. In this model, "Drop" action is known as being the most wrongly classified action, but the model is effective to classify this action when fed. Due to this, the combination actions with Drop action incorporated would have higher Frame accuracy than the others, just as seen in the table. From all the F1-score combo for the "Cross hands in front" action, is the second best action. Due to the "Standing Up" action order restrictions and high F1-Score single and Frame Score, to the "Drop" action bias, and due to the "Cross hands in front" action achieving high F1-Score single and combo and Frame Score, all the combinations, including "Standing Up" action resulted in a high Frame Score.

The middle part of the table contains tests composed of the three best actions selected, besides "Standing Up". We can conclude that the "Cross hands in front" action works well as a starting action, although it achieved the individual lowest score from the three. On the other hand, as expected "Shake Head" action performs well as the starting action. On the other side, the "Wave hand" action imposes lower Frame Accuracy when starting the combination action. If we take a look at the lower part of the table, we see the combination actions composed by the two "bad" actions and one surprisingly better, but still not good enough F1-score.

The actions where at least one sample was correctly classified occurred when the sequence started with the "Brushing-Teeth" action, which is surprising since it had the worse score in the F1-score table. We think that the "Brushing Hair" works best as a second action in the action combination when the first action is "Brushing Teeth" in contrast to the second action being "Read". Since the "Brushing Hair" is more similar to the "Brushing Teeth", it might give an idea of unstoping brushing for a long time, which would lead to a classification of one brushing activities.

Accordingly to the algorithm, described in Section 4.1 for reading the JSONs, we start to build the action by putting the oldest action on the top of the figure, and from JSON to JSON we modify each row of the figure until we reach the figure's limit. From that point on, the figure will suffer a roll up operation and the bottom line will be updated per each next JSON.

An example of an action in a featured image can be visualized in Fig. 4.19, where the black represents the data with zeros and the colors an RGB interpretation of each of the values for joint distance and orientation. If we think of this process for three actions instead of one as represented, at a certain point, the image will be filled with color. If the action being fed at that moment is a very short action, its dataset featured image has a significant number of zeros. This featured

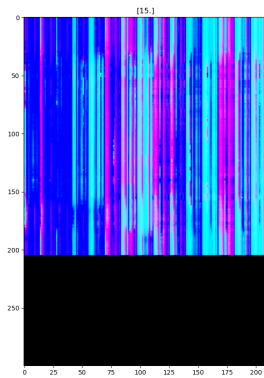


Figure 4.19: Action 15 JOD Featured Image example

sample, as it's filled with color, doesn't have any zeros. It will be challenging for the model to classify that action correctly. For that reason, we conclude that the model is sensitive to the video length.

Chapter 5

Conclusions and Future work

5.1 Conclusions

In this thesis, the problem of developing an intelligent system for live skeleton-based action recognition was addressed. This system implements real-time incorporation of OpenPose with a DL model for action classification. The outcomes of our first LSTM-based model were very poor, not surpassing the 5% test accuracy. For that reason, a lot of time searching for the problem was invested, leading us to adopt a CNN-based approach. Our final CNN-based model developed and achieved state-of-the-art performances, with 82.81% and 85.99% accuracies for the Cross-Subject and Cross-View NTU-RGB+D dataset benchmarks, respectively.

In searching for the problem with our LSTM-based model, various issues in the original dataset were found, from wrongly recognized joints or entire persons, to the dataset's imbalance concerning the frame samples. To solve this, we implemented the K-means method and resampling techniques relying on SMOTE like approaches and RV Coefficients, allowing us to improve the original dataset, generating a more clean and complete one. We were also able to conclude that the best-performing feature tested was the *JOD* feature, which, combined with its frame movement, improved accuracy. Besides that, we proved the usefulness of including Batch Normalization, Dropout 2D, and the decoupled attention mechanism.

Extensive research is also presented regarding the current state of DL on the topic of action recognition, focusing mainly on LSTM and CNNs networks. This research allowed a better understanding of how this topic is usually approached and some of its applications. Some of the best performing approaches were reviewed and compared, which led to more informed development of the proposed system. However, some works lacked implementation details, which lead us to a more creative approach to solving our problems.

From live testing, we concluded that some actions were better recognized than others, perhaps due to the movement features, where actions with big motions have more chance of being well recognized. On the other hand, actions with small movements were frequently misrecognized, with special attention to the "Drop" action, which the model seems biased towards due to the skeleton's almost static position, resembling either the initial or final frames of other actions. Moreover, some actions, for example, "Shake head", might be well recognized alone but worse when in combination with others, when comparing with some actions, such as "Cross hands in front(say stop)", that performed worse alone, but better when combined with others.

One of the most significant limitations is the dependency on the video length for classification. One of the minor limitations is the required folder to write the JSON files from the Openpose, as they are created for an indefinite or very long period of time, which for a long time of exposure will result in a big chunk of files. The model could have more robustness tests, but due to Covid-19, the restrictions that came with it made it much more difficult for the testing component.

5.2 Future Work

As the most significant limitation is the sensibility for video length, as future work we could implement a Spatial Pyramid Pooling as described in [25], allowing a variable-sized input to a constant variable input. This way, the mentioned sensibility would probably decrease. Another option would be applying a frame select mechanism such as a Region Proposal Network as WPN used by Li *et al.* [8]. Assuming that the video length limitation is surpassed by one of the methods described above, another aspects that could be relevant to explore would be interaction recognition (actions performed by two or more persons). For instance, that could be achieved by creating a featured image of the same shape as the ones in both streams, a mechanism to concatenate undefined streams into one, could be adopted such as the Maxout strategy in Li *et al.* [8].

Furthermore, LSTMs are used for word prediction in a text of any language. With this in mind to enable the model to not only detect and recognize action but to predict them, a LSTM layer could be applied to the output of this model, where this prediction model would receive the labels of the actions classified and, from there learn temporal dependencies to predict the next action based on the ones before. Furthermore, we could add an intermediate model to group the last N actions into 1 Activity ID, which would serve as input for the LSTM Network. Besides this reducing the variability on the input also might be a kind of a time selection solution similar to the WPN mentioned earlier.

References

- [1] Fernando Martins. Solar radiation forecast using artificial neural networks. Reproduced as Figure 2 in https://www.researchgate.net/figure/S-YMBOLIC-REPRESENTATION-OF-AN-ARTIFICIAL-NEURON-AND-ITS-PARAMETERS_fig2_235933074, 2012.
- [2] Christopher Olah. Understanding lstm networks, 2015. URL: <http://colah.github.io/posts/2015-08-Understanding-LSTMs>.
- [3] Sijie Song, Cuiling Lan, Junliang Xing, Wenjun Zeng, and Jiaying Liu. An end-to-end spatio-temporal attention model for human action recognition from skeleton data. In *AAAI*, 2017.
- [4] Pengfei Zhang, Cuiling Lan, Junliang Xing, Wenjun Zeng, J. Xue, and N. Zheng. View adaptive recurrent neural networks for high performance human action recognition from skeleton data. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 2136–2145, 2017.
- [5] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with multi-stream adaptive graph convolutional networks. *ArXiv*, abs/1912.06971, 2019.
- [6] Lei Shi, Yi-Fan Zhang, Jian Cheng, and H. Lu. Decoupled spatial-temporal attention network for skeleton-based action recognition. *ArXiv*, abs/2007.03263, 2020.
- [7] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. NTU RGB+D: A large scale dataset for 3d human activity analysis. *CoRR*, abs/1604.02808, 2016. URL: <http://arxiv.org/abs/1604.02808>, [arXiv:1604.02808](https://arxiv.org/abs/1604.02808).
- [8] Chao Li, Qiaoyong Zhong, Di Xie, and Shiliang Pu. Skeleton-based action recognition with convolutional neural networks. *CoRR*, abs/1704.07595, 2017. URL: <http://arxiv.org/abs/1704.07595>, [arXiv:1704.07595](https://arxiv.org/abs/1704.07595).
- [9] Chao Li, Qiaoyong Zhong, Di Xie, and Shiliang Pu. Co-occurrence feature learning from skeleton data for action recognition and detection with hierarchical aggregation. *arXiv preprint arXiv:1804.06055*, 2018.
- [10] Peter M. Roth, Thomas Mauthner, Inayatullah Khan, and Horst Bischof. Efficient human action recognition by cascaded linear classification. pages 546 – 553, 11 2009. doi:10.1109/ICCVW.2009.5457655.
- [11] Moshe Blank, Lena Gorelick, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. In *The Tenth IEEE International Conference on Computer Vision (ICCV’05)*, pages 1395–1402, 2005.
- [12] Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. *Transactions on Pattern Analysis and Machine Intelligence*, 29(12):2247–2253, December 2007.

- [13] Daniel Weinland, Remi Ronfard, and Edmond Boyer. Free viewpoint action recognition using motion history volumes. *Computer Vision and Image Understanding*, 104(2):249 – 257, 2006. Special Issue on Modeling People: Vision-based understanding of a person’s shape, appearance, movement and behaviour. URL: <http://www.sciencedirect.com/science/article/pii/S1077314206001081>, doi:<https://doi.org/10.1016/j.cviu.2006.07.013>.
- [14] Jaeyong Sung, Colin Ponce, Bart Selman, and Ashutosh Saxena. Human activity detection from RGBD images. *CoRR*, abs/1107.0169, 2011. URL: <http://arxiv.org/abs/1107.0169>, arXiv:1107.0169.
- [15] B. Ni, Gang Wang, and P. Moulin. Rgbd-hudaact: A color-depth video database for human daily activity recognition. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 1147–1153, 2011. doi:10.1109/ICCVW.2011.6130379.
- [16] Abdelouahid Ben Tamou, Lahoucine Ballihi, and Driss ABOUTAJDINE. Automatic learning of articulated skeletons based on mean of 3d joints for efficient action recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 31, 09 2016. doi:10.1142/S0218001417500082.
- [17] Quentin De Smedt, Hazem Wannous, Jean-Philippe Vandeborre, J. Guerry, B. Le Saux, and D. Filiat. 3D Hand Gesture Recognition Using a Depth and Skeletal Dataset. In Ioannis Pratikakis, Florent Dupont, and Maks Ovsjanikov, editors, *Eurographics Workshop on 3D Object Retrieval*, pages 33–38. The Eurographics Association, 2017. doi:10.2312/3dor.20171049.
- [18] Bart Iver Van Blokland. Shrec 2017 large scale 3d shape retrieval dataset (mirror), 2020. URL: <https://data.mendeley.com/datasets/ysh8p862v2>, doi:10.17632/YSH8P862V2.
- [19] Kiwon Yun, Jean Honorio, Debaleena Chattopadhyay, Tamara Berg, and Dimitris Samaras. Two-person interaction detection using body-pose features and multiple instance learning. pages 28–35, 06 2012. doi:10.1109/CVPRW.2012.6239234.
- [20] JF Hu, WS Zheng, J Lai, and J Zhang. Jointly learning heterogeneous features for rgb-d activity recognition. *IEEE transactions on pattern analysis and machine intelligence*, 39(11):2186–2200, 2017.
- [21] Jun Liu, Amir Shahroudy, Mauricio Lisboa Perez, Gang Wang, Ling-Yu Duan, and Alex Kot Chichung. Ntu rgb+ d 120: A large-scale benchmark for 3d human activity understanding. *IEEE transactions on pattern analysis and machine intelligence*, 2019.
- [22] S. Zhang, X. Liu, and J. Xiao. On geometric features for skeleton-based action recognition using multilayer lstm networks. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 148–157, 2017.
- [23] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.
- [24] Max Strobel. Hcn-prototypeloss-pytorch. <https://github.com/maxstrobel/HCN-PrototypeLoss-PyTorch>, 2018.
- [25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. *CoRR*, abs/1406.4729, 2014. URL: <http://arxiv.org/abs/1406.4729>, arXiv:1406.4729.