Fraunhofer IPK

Technische Universität Berlin
Electrical Engineering and Computer Science
Department of Telecommunication Systems

Machine Learning based Human Action Recognition using pose estimation and object detection in Human-Robot Collaborative Tasks

Kyra Kerz

Matriculation Number: 329289

Supervisor: Prof. Dr.-Ing. Jörg Krüger
Prof. Dr.-Ing. Jens Lambrecht
Advisor: Dr. Kevin Haninger

Berlin, August 18, 2023

Master's thesis statement of originality

I hereby declare that the thesis

Machine Learning based Human Action Recognition using pose estimation and object detection in Human-Robot Collaborative Assembly Tasks

submitted is my own, unaided work, completed without any unpermitted external help. Only the sources and resources listed were used.

The independent and unaided completion of the thesis is affirmed by affidavit:

Berlin, August 18, 2023

Name

Abstract

Human-robot collaboration in industrial settings aims to support and simplify assembly tasks carried out by a human operator. A collaborating robot has to act accordingly in order to achieve a joint goal, which requires being able to recognize actions of different operators for a given task. The detection of those actions needs to be tailored to the constraints of the industrial settings, which desire low costs, data efficiency, high performance quality and an easy set up.

To this purpose this thesis presents a light weight human-action-recognition (HAR) framework for industrial assembly tasks, which uses just one RGB-D camera for data recording, and supervised machine learning models for the action classification. In order for the models to be able to do real time classification, they have to be trained before hand with data sets of two assembly task setups, which are recorded during execution. RGB and depth frames are used for the extraction of selected features, which involves the human pose with skeleton tracking and work pieces present in the task through object detection. Four different types of deep learning networks are created and trained, in order to find the configurations of features, which are yielding the highest classification performance.

The results are showing that the HAR framework is able detect actions in terms of fulfilling constraints of industrial settings. Training the different models on pre-selected features revealed the importance of temporal dependencies. However spatial information in form of the human motions are crucial, which the comparison of both task setups shows. Furthermore the object detection did increase the performance more in comparison to the depths values, especially when human motion is locally restricted.

Zusammenfassung

Die Mensch-Roboter-Kollaboration in der Industrie zielt auf die Unterstützung und Vereinfachung von Montageaufgaben, die von einem menschlichen Bediener durchgeführt werden. Ein kollaborierender Roboter muss entsprechend handeln, um ein gemeinsames Ziel zu erreichen, was voraussetzt, dass er in der Lage ist, die Handlungen verschiedener Bediener für eine bestimmte Aufgabe zu erkennen. Die Erkennung dieser Handlungen muss auf die Randbedingungen der Industrie zugeschnitten sein, die niedrige Kosten, Dateneffizienz, hohe Leistungsqualität und eine einfache Anwendung wünscht.
Zu diesem Zweck wird in dieser Arbeit ein praktisches System zur Erkennung menschlicher Handlungen (HAR System) für industrielle Montageaufgaben vorgestellt, das nur eine RGB-D Kamera für die Datenaufnahme und überwachte maschinelle Lernmodelle für die Handlungsklassifizierung verwendet. Damit die Modelle in der Lage sind, Echtzeit-Klassifikationen durchzuführen, müssen sie zuvor mit Datensätzen von zwei Montageaufgaben trainiert werden, die während der Ausführung aufgezeichnet werden. RGB- und Tiefenbilder werden für die Extraktion ausgewählter Merkmale verwendet, die die menschliche Pose mit Skelettverfolgung und die in der Aufgabe vorhandenen Werkstücke durch Objekterkennung umfassen. Vier verschiedene Arten von neuronalen Netzen werden trainiert, um die Konfigurationen von Merkmalen zu finden, die die höchste Klassifizierungsleistung erbringen.
Die Ergebnisse zeigen, dass das HAR System in der Lage ist, Handlungen zu erkennen, die den Anforderungen der industriellen Umgebung entsprechen. Das Training der verschiedenen Modelle auf zuvor ausgewählten Merkmalen zeigte die Bedeutung zeitlicher Abhängigkeiten. Allerdings sind räumliche Informationen in Form der menschlichen Bewegungen von entscheidender Bedeutung, wie der Vergleich der beiden Aufgabensetups zeigt. Darüber hinaus hat die Objekterkennung die Leistung im Vergleich zu den Tiefenwerten erhöht, insbesondere wenn die menschliche Bewegung lokal begrenzt ist.

Contents

List of Figures

List of Tables

1 Introduction

Robotics encompasses a wide field of applications and deployments of autonomous systems that are increasing in numbers nowadays, due to optimization of energy and resource usage or reducing carbon emissions [19], as well as alleviating labor shortage and reducing physically straining work [3]. Those reasons are especially concerns for industrial and manufacturing work settings. This leads to a rising amount of robotic systems, which are not just performing tasks locally separated from humans, but instead are migrating into fields that require collaborative effort between humans and robots like assembly tasks [1]. However successful collaboration between several actors, whether human or machine, relies on various conditions. According to Wang et al. [80] being able to recognize intentions of their co-actors like common goals and corresponding executed actions are crucial. While those abilities are natural to humans, machines have to be taught to recognize intentions and actions, especially in the context of human-robot-collaboration (HRC).
Human action recognition (HAR) is a field that aims to understand and classify human behaviour into actions. According to Aggarwal and Ryoo [2] human actions are composed by sets of gestures like raising an arm or stretching a leg, which are in temporal order and are building purposeful movements like running or waving. Different action classes are created by different sets of gestures, which leads to a certain interclass variance. But individual humans tend to execute the same action class with some variability in speed or style, which leads to intraclass variance and variations of similar gesture sets corresponding to the same action class[55]. Therefore various attributes like kinetic motions, facial mimic, gaze, environment, interactions with objects or other humans can be analysed in order to gain an understanding of their actions and intentions. A sequence of actions may merge into new higher level actions or activities e.g cooking, which involves actions like cutting, stirring or peeling, which adds also a dependency on temporal behaviour. Different data modalities like audio, infrared, point clouds, RGB and skeleton can be used to encode those attributes.

In order to make those information usable for a machine, classifications or predictions have to be made based on those data modality inputs. Deep learning is a very popular approach for this purpose, especially for supervised classification tasks which rely on labelling each fragment of the incoming data stream. However supervised classification approaches require prior knowledge during the labelling process in order to apply the correct classes to each data fragment. Depending on the collaboration task, domain knowledge is necessary. HRC for industrial assembly tasks has to consider area related specifications in terms of workflows, working environment, involved participants, changing collaborating operators and more, in order to reliably recognize actions despite high inter- and intraclass variance. Simultaneously HRC is subject to practical constraints like costs, uncomplicated integration and usage, as well as safety concerns. Therefore this thesis presents the implementation of a light weight HAR-framework for industrial assembly tasks for human-robot-collaboration, which leverages features extracted from RGB-D recordings of two different assembly task set ups as data input for four deep learning action classification architectures, in order to gain insight of essential action attributes which are increasing the HRC performance.

1.1 Motivation

The majority of the currently deployed robotic systems in industrial settings are limited to tasks that are required to be repetitive and predictable in order to be solved [3], thus being controlled by pre-generated rigid codes. Those tasks include at most the coexistence of human and robot[1], where there is no direct contact between both. However joining their forces and aiming for HRC, allows to combine the flexibility and adaptability of humans with the precision, strength and durability of robots, achieving a higher efficiency and productivity for manufacturing procedures. However in a collaborative setting the robot is required to understand the actions and intentions of the partaking operator in order to achieve a joint goal. This means controlling the robot with pre-generated rigid codes is not feasible, since in addition, task durations are varying in execution style and speed when collaborating with different operators. One challenge in this regard is the inability of today's robot systems to perceive and reason about the world. Computer vision techniques like object detection and pose estimation have emerged to provide the robot with

semantic information. In order to predict those actions, spatial and temporal dependencies of the object detection and pose estimation are used to train several machine learning algorithms and help to generalize in a semi structured environment. Thus the capabilities of training sample efficient machine learning algorithms based on visual input for human action recognition are explored in this thesis.

1.2 Research Question

Ensuring a safe and efficient framework for collaboration between robots and humans in an manufacturing setting relies on several criteria. One is the ability to anticipate motivation and goal of the collaborators regarding the task in order lend their support accordingly. Secondly the framework has to be low cost, intuitive to use and robust against cluttered workshop environments and changing collaborators. Additionally features for action classification have to be extracted based on two different perspectives. It has to be determined how an action is recognized and characterized by humans in the industrial context on one hand, which includes action attributes, involved tools, movements and the surrounding environment. On top of that it has to be examined how those features can be extracted by image processing approaches in regards to sensing modalities, spatial and temporal information and more, in order to provide them as input to classification algorithms.
This leads to the core question of this thesis: Which and how many features are necessary in order to identify and classify an action in the domain of industrial robot-human collaboration tasks with machine learning approaches?

1.3 State of the Art

There are already several state-of-the-art approaches for tackling human action recognition, that are involving the usage of machine learning techniques. In order to acquire the sufficient features and input data, most of them rely either on wearable accelerometers and inertial measurement units (IMU) sensors or use computer vision based methods. Both approaches are mainly used for detecting singular, isolated actions. This is done by capturing the body movements involved in the execution of an action and in case of computer vision based methods, also other factors like environment are involved. Those features are

then fed into neural networks like deep neural networks (DNN), convolutional neural networks (CNN) or long-short term (LSTM) neural networks, which are classifying the actions accordingly to the monitored collaborative task. However using wearable components for measuring movements are impractical for monitoring the movements of collaborators in joint industrial assembly tasks, as putting the sensors on and off is extra effort for the collaborators and can lead to errors in measurements by wrongful application. Furthermore similar movements involved in different actions might lead to a lot of uncertainty in detecting a correct action. Therefore vision based methods have some advantages over wearable based approaches. First putting the cameras into place is a one time effort and after that the monitoring setup doesn't change. Secondly vision based approaches allow to gather more information that involves the environment or used tools for the action. However using all full frames as input for neural networks has a high computational complexity and relies on the use of expensive hardware like strong GPUs, which are expensive right now given the delivery shortages due to Covid 19. Furthermore many current approaches rely on several cameras observing the scene from different angles, which makes the setup very complex and expensive. And finally another limitation of those approaches is the focus on singular actions instead of a whole action sequences or activities, where one movement transitions into another. This is especially true for industrial assembly tasks where there are no public data sets available for training.

1.4 Contributions and Validation

Based on the motivation and the research question this thesis aims to present a human-action-recognition (HAR) framework for facilitating robot-human-collaboration for industrial assembly tasks. Furthermore the developed framework aims to overcome the challenges and shortcomings of the current approaches. Therefore the contributions of this work are the following:

- **Developing lightweight HAR framework:** In order to minimize the costs and hardware overhead, just one, low priced depth vision sensor is used - the Kinect v2 with the embedding in Linux with an open source driver setup for depth cameras.
- **Setting up two different assembly tasks for monitoring:** In order

to examine which features are important for recognizing actions in an industrial domain, two different industrial assembly task setups with different tools and different working steps are prepared. Even though the workspace and fixed camera position are the same, different level of occlusion and view of the work tools are occurring due to the structure between both tasks, as well as spatial separation of individual actions, the human movements that are required and more.

- **Creating a domain appropriate data set:** As there are no public available data sets of industrial manufacturing tasks that feature a whole assembly pipeline, a custom data set for feature extraction and training the used machine learning algorithms has to be recorded. This data sets includes two different assembly tasks. The first scenario involves the construction of an elevator cabin hanger and the second scenario of a control cabinet door.
- **Decreasing computational complexity and increasing data efficiency with optimized feature extraction:** To this purpose skeleton tracking is used to monitor the movements of the collaborators or more precisely the 3D spatial information of their joints. Additional information is gained by using object detection. The combination of skeleton and object features makes for a low dimensional feature vector as input for the neural networks, which lowers computational costs while optimizing the classification accuracy. Reduction of data recordings due to less clutter.
- **Comparing classification performances of different deep neural network architectures:** The classification success relies on the ability of the model to recognize the correct actions depending on the provided features. However the interpretability of those features varies with the chosen network architecture. To this purpose four different models are implemented. The feed forward neural network provides insights about spatial information, a 1S-LSTM provides temporal information, a 3S-LSTM for analysing the influence of 2D, depth and object data individually and a TS-LSTM for analyzing different action duration and transitions between actions.

In order to validate those contributions, metrics for analyzing classification success are used. Those are the confusion matrix for classification accuracy

of the different classes, probability distributions for identifying classification uncertainties, heat maps for visualizing the important features for the classification success and finally the receiver operating characteristic (ROC) curve is shown in order to get information about the diagnostic abilities of the models.

1.5 Structure of this Thesis

In this thesis a human-action recognition (HAR) framework for classifying the execution of an action by an operator for a certain assembly task is presented. An overview of the HAR-framework structure is displayed in figure 1.1. The main components are the recording of the data set, the feature extraction, the training of the machine learning models and finally the evaluation of the classification capabilities of the trained models. The basis of the data sets is given by the two assembly task setups that require collaboration effort between two operators. Both setups are recorded by a RGB-D camera. Next feature extraction of the recordings is done and includes the tracking of the human pose as well as the working pieces. Those features are used as input for different classification algorithms, which are trained as supervised learning algorithms and are able to leverage spatial and temporal dependencies. Finally the classification performance the models is evaluated and the feature importance regarding the assembly task is analyzed. Finally In total the the-

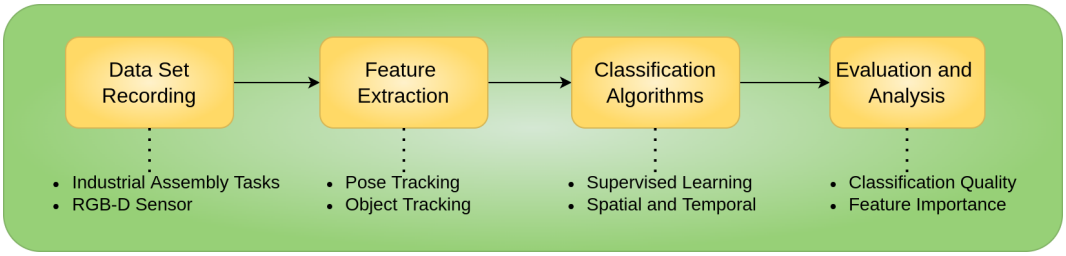


Figure 1.1: HAR framework structure

sis consists of seven chapters. The first chapter 1 is giving the introduction into the thesis, which encapsulates the motivation for the chosen topic, the research question, a short overview of the state of the art contributions and validation as well as the structure of the thesis. In the second chapter 2 the

related work is examined. Established methods and technologies are introduced, the strengths and limitations are analyzed and the contribution of this thesis are set in relation. The third chapter 3 deals with the involved methods for designing the HAR framework. This includes the problem statement, the proposed solution and the workflow of the proposed solution and its analysis. The background of the implemented solution and algorithms are explained in detail in chapter 4. The next chapter 5 highlights the validation methods for the proposed architecture in order to determine feasibility, benefits and disadvantages of the chosen approach. For this purpose the setup of both assembly tasks, the set up for recording the data sets, the software for the feature extraction, the machine model architecture and pre-processing of the data and the evaluation methods are described. Furthermore in chapter 6 the details of the executed experiments are described and properties of the proposed solution are tested with ablation, performance and robustness tests, as well as compared with other approaches. Followed by the conclusion in chapter 7, which includes a summary of what was shown in the work and a description of how it relates to the introduction. Finally future work is discussed.

2 Literature Review

In this section, previous research on human action recognition is examined. There are a variety of approaches due to usage of different sensors for the recording of human motions and different methods for the actual recognition of actions out of those motions. However, in this thesis the focus lies in the usage of spatio-temporal information gathered by joint coordinates of the human skeleton. Therefore the presented related work in this section is divided by feature extraction methods, used sensors, domain of the recorded data and isolation-level of the actions.
Approaches for feature extraction methods can be grouped into two categories, which are human action recognition based on feature maps and human action recognition based on skeletons [62]. Another distinction is made by the deployed sensors for recording, which are mostly based on using cameras to record the scenes or wearing sensors like inertial measurement units (IMU), accelerometers or gyroscopes according to [30]. Another distinction is made by analyzing single actions compared to analyzing sequences of actions or activities, which means also an analysis of the transitions between different actions. In the second case classifying actions which are transitioning into other actions is more complex than considering isolated actions [51]. And finally there is a distinction made by the context in which the actions take place, classifying day to day activities in contrast to activities involved in manufacturing assembly tasks, which is mirrored in the freely available data set for human action recognition and barely involved manufacturing task processes [53].

2.1 Sensors for Human Action Recognition

Human activity recognition is studied in various environments and use cases. Hussain et al. are presenting a survey, that divides human action recognition approaches mainly into vision based approaches and wearable sensor based approaches [30]. The vision based approaches are encompassing action monitor-

ing via RGB or RGB-D cameras, event cameras, LiDAR or infrared cameras. Wearable sensor based approaches are including sensors like accelerometer, gyroscope, and magnetometer. Tao et al. are proposing an attention-based approach using multiple wearable IMUs for health monitoring for several sport related actions [75]. The most discriminative features of the recorded IMU data within the frequency domain are extracted with convolutional neural networks. Furthermore an attention-based fusion mechanism is applied in order to detect the importance of the sensors at different body locations. A multi-layer neural network is used for classification of the actions.
A similar approach for human action recognition that is based on wearable IMU sensors, is presented by Huang et al. [29]. They are using a CNN which is capable of cross-channel communication, meaning all channels in the same layer can interact with each other. This allows to discard redundant information and to capture more discriminative features of the sensor input. Another approach is presented by Golestani and Moghaddam [25]. They are proposing a wireless, magnetic induction-based system, which represents the motion of human body parts. The participants are wearing transmitter coils on various body parts and a receiver as belt. During physical motions the variations in the magnetic inducted signals are transmitted from transmitter to the receiver and are recorded. The set of magnetic induction signals, which are observed at a certain time by the coils and are used as input for a recurrent neural network. Also the work of Chen et al. [11] incorporates wearable sensors and analyzes sensor readings from accelerometers and gyroscopes. However on top of that they also rely on location information recorded by wearable RGB cameras which recordings are used for object detection. This allows the classification done by LSTM networks to rule out certain activities given the information about certain objects in the scene. Using the local information improves the overall accuracy compared to using only skeleton information. However using wearable sensors is critical in an industrial manufacturing setting, which would take a lot of time for the operators to put them on. Furthermore the analyzed scenes are only containing single actions and not whole assembly tasks, which leaves out the difficulty of classification for transitioning from one action to another.
A vision based approach that uses a RGB-D sensor, is presented by Imran et al. [31]. The recordings of daily action performed by participants are used for generating motion history images (MHI) at first, which represent the temporal

information about the action. After that the depth data is rotated in 3D point clouds and three depth motion maps (DMM) are generated over the entire depth sequence corresponding to the front, side and top projection views. A four channel Convolutional Neural Network is trained, where the first channel is for classifying MHIs and the remaining three for the front, side and top view generated from depth data respectively.
Liu et al. [51] are proposing a distributed RGB-D camera network for human action recognition in order to handle the occlusion problem effectively, which exists in a single-view image setup. They are extracting joint coordinates from participants doing isolated movements as actions. The skeleton data is then fused with a information weighted consensus filter and finally used for action classification with a support vector machine. However setting up several RGB-D cameras is costly, require extensive sensor fusion and calibration, as well as sufficient space for the installation, which be challenging in an industrial context.
Huang et al. are using event based cameras for action recognition [28]. They are using a time stamp encoding 2D network, which encodes spatial-temporal images with polarity information of the event data as input and outputs the action label. Finally the information is given into a spiking neural network (SNN), which is used for classification. However event cameras are expensive and relying solely on the local changes in brightness. In this context, Roitberg et al. [65] introduce a human robot collaboration framework in an industrial setting using computer vision. It takes RGB-D recordings of several human operators performing actions related to the robot and converting them into a skeleton with joint coordinates. Those are then used in order to train random forests and a hidden markov model that are able to detect the performed actions. However in their approach they are using two RGB-D sensors, relying on sensor fusion. In this case they were just recording actions like waving or pointing. In assembly scenarios normally people don't necessarily make one specific action and than go idle. In most cases there are sequences of actions that transition from one to another and are also not intentional like waving or point, but more natural actions belonging to the assembly task.

2.2 Video Feature Extraction for human action recognition

Action recognition usually consists of two steps - first the feature extraction step and second the classification step. There are many approaches that try to capitalize on the spatio-temporal dependencies of video frames. A very common approach is the usage of convolutional neural networks (CNN) given the video frames as input [5]. However CNNs used for images are only able to extract features from the spatial dimension and foregoing temporal information [48]. In the work of Ji et al. [35] they are extending regular 2D CNNs to 3D CNNs by performing 3D convolutions of stacked frames. This allows to extract motion information of adjacent frames. However training CNNs end-to-end, especially 3D, is computational very expensive with rising cost depending on the number of available frames and their size. Furthermore cluttered background or unnecessary information provided by the frame content can lead to subpar real-world applicability [5]. In the work of Chaaraoui et al. actions of the MSR-Action3D data set will be classified [9]. For this purpose a RGB-D device was used in order to extract 3D coordinates of the human skeletons of the recordings. Then evolutionary feature subset selection is applied while taking the topological structure of the skeleton into account. This reduces the dimensionality of the input data and increases the classification success. However using only the skeleton coordinates and kinetic motions ignores additional information about tools, other actors or environments that are important for recognizing actions. The work of Shinde et al. is also focused on a pre-selection of features for reducing the dimensionality of the data [69]. But instead of using evolving joint coordinates, they are focusing on key components like involved objects or interacting humans in singular frames. Then Yolo v4 is used to train their model based on recorded scenarios in an office environment. However their input data set is based on large objects that are embedded in the environment like doors or white boards instead of tools that are carried around and assembled. Furthermore most actions don't share a similar environment, which provides additional information, compared to an assembly task that is done in the same lab for the whole duration.
The work of Das et al. also explores the combination of 3D skeleton movements and feature extraction of RGB images to capture interactions with the environment [81]. The 3D coordinates are which are provided for classification

by a Deep-Temporal LSTM network. The RGB frames are cropped at the location that provides the most discriminative information and finally classified with a support vector machine. Their work is validated on the public data sets CAD60, MSRDailyActivity3D and NTU-RGB+D. All those data sets are involving daily actions. Yu et al. are presenting a similar approach that combines the sequential attributes of skeleton geometry with features of RGB images [81]. To this purpose regions of interest are extracted from the RGB frames, based on the joint location of the skeleton inside the frames. They are validating their results on the NTU-RGB+D and Northwestern-UCLA Multiview data sets, which capture daily actions like sitting down or throwing an object. The skeleton coordinates are provided as input for a GCN and the extracted regions of interest are then fed into a CNN. The loss functions of both networks are calculated individually and the joint loss used as classification metric. However those approaches are again relying on RGB as input for real time classification, which adds higher computation time, even though the regions of interests are smaller in pixel size compared to the original frames. Also those approaches require direct interaction with the environment of the human, since only regions around their skeleton are extracted as features. However in this approach occlusion becomes dependent on the body movement and location of the participant, since the person might stand in line of sight between camera and objects to interact with or small objects might get occluded during holding or using an object. In an industrial context this would place a higher responsibility on the worker to move accordingly to avoid occlusion and bad angle towards the recording device, instead of simply concentrating on their assembly task.

2.3 Actions and Activities

So far most research work has been conducted on data sets like UCF101 [74] or HMDB51 [43], which are featuring 101 and 51 clips of isolated, everyday actions respectively, like sitting down, applying lipstick or waving, instead of whole activities [46]. Therefore most systems for recognizing human actions ignored posture transitions in the past, because incidence of posture transitions is lower and the duration is shorter than other basic physical activities. However, analyzing single actions does not adapt well to the dynamic nature of human activities. There are some publicly available data sets which are

also containing activities and not just clips with actions like ActivityNet [21], Charades [71] or MPII Cooking [64]. Chen et al. [10] investigated activity recognition with postural transition awareness. For this purpose they used an accelerometer and a gyroscope of an inertial measurement unit in order to track physical motion of humans. The tracked motions are consisting of sequences of the three actions standing, sitting and lying down for which the transition between all of the possible order of actions per sequence are analyzed. However in this scenario three very distinctive actions in repetitive order where chosen as sequence. This doesn't match the complexity of manufacturing tasks, which involves more actions which are also more granular.
Another approach is presented by Zahng et al. [83]. They are using the RGB-D sensor Kinect v2 for extracting skeletal data. Those coordinate sequences are then divided into pose feature segments and motion feature segments. A maximal entropy markov model for the classification, given the likelihood probabilities which are provided by a key pose matching process. This ensures that start and end points of human actions don't have to be known in advance for real time applicability. However this method alone relies simply on the kinetic movements present in the recordings and omitting environmental information that might give insight in recognizing the actions.

2.4 Industrial Assembly Tasks

There are a lot of freely available data sets for human action recognition to find online. However, existing human action recognition data sets mostly contain common life activities like running, drinking or applying make-up, which leaves a huge lack of data sets in the domain of industrial assembly tasks [46]. A very recently published data set in this area is created by Iodice et al. [33]. They recorded 30 categories of industrial-like actions in 2940 clips with 98 frames for each action. However even though the chosen actions are related to industrial tasks, they don't feature a complete assembly task sequences, but rather isolated actions. Zhang et al. [82] are featuring the robot collaborative assembly of a car engine. They are using RGB-D recordings of this assembly task to extract skeleton joint coordinates and feed them into a recurrent neural network. The goal is to predict the movement trajectory of the human. However this approach focuses mainly on the recognizing of a handover situation

between human and robot. Also they are ignoring task related information like tool usage.
One publicly available data set for human action recognition is the InHard (Industrial Human Action Recognition Dataset) [16]. It features 14 different industrial related assembly activities, containing 72 different actions carried out in a lab environment by 16 participants. The data is gathered with an IMU for collecting 3D joints locations and RGB video streams. However so far it lacks extended evaluation, which Manns et al. are tackling in their research [4]. They are using a combined CNN and LSTM based training approach to evaluate the InHard dataset and compare it with a custom created dataset recorded featuring industrial assembly tasks in their lab environment. Furthermore they are using spatial video data and IMU-based 3D skeleton coordinates separately as training data. Additionally they added a pose detection overlay to the video frames in order to generate RGB-based skeleton data as third input modality. However while they achieved high results for their IMU based classification, their video based classification yields worse results due to inefficient recording angles and occlusion being present in their setups. Also training on a publicly available data set may not cover the range of actions needed for certain assembly tasks and doesn't take into account the specifics that custom created data sets are providing for a tailored task.

3 Human Action Recognition

In this chapter the problems of human action recognition in the context of industrial assembly tasks are outlined and the proposed solution presented. First a formal statement of the problem is given in section 3.1, which includes a description of challenges and advantages given the specific domain. Secondly the human action recognition (HAR) framework as proposed solution and own contributions are presented in section 3.2, as well as an overview of the established methods that are built upon. A more in depth explanation of the used methods and algorithms are presented in the chapter 4 and 5. Finally in section 3.3 the presented solution is described on application level and outlined as complete workflow, showing how each part of the solution fits in together. This also includes the modules for the analysis of the HAR framework.

3.1 Formal Statement of the Problem

In this thesis, a human action recognition (HAR) method for real time human-robot collaboration in an industrial context is presented. This combines two different areas, that both come with their own specifications. The first area is human-robot collaboration in general. In order for two agents to be able to collaborate, they must be able to recognize each others intentions [9]. This however requires knowledge about currently executed actions, which in itself have to be detected and recognized. But what defines as an action? Aggarwal and Ryoo [2] describing human actions as composed sets of gestures like raising an arm or stretching a leg, which are in temporal order and are building purposeful movements like running or waving. Additionally Kragic et al. [42] describing that an action can be recognized by a wide range of features like kinetic states, environment, number of participants or involved objects in the scene. Given this information an agent is able to make inferences about the intention and state of the other agent and therefore able to collaborate accordingly. The second area outlines the domain specific specifi-

cations for the human-robot collaboration for industrial assembly tasks. This includes not just being able to recognize intentions of the participants involved in those assembly tasks, but also comes with requirement for the usability of the HAR-framework. The industrial context puts already several constraints onto the aforementioned requirements for recognizing actions related to assembly tasks, but does also come with its own set of challenges. Collaboration between humans and robots can't be programmed as static set of commands, since different operators may vary in execution style and time for the same assembly task and therefore needs a framework that provides robust generalization capabilities. Furthermore assembly tasks are rarely made up by one isolated actions, but rather a concatenation of actions which are transitioning from one to another - which is creating a whole activity [79]. This adds the challenge of recognizing when an single action starts and when it ends. Typical approaches for human action recognition don't cover this challenge, since publicly available HAR data sets, like the UCI HAR data set [18] or the UCF101 [74] are just featuring isolated actions and not activities. However the specific domain presents prior knowledge about objects, order of actions, participants and environment for a more concise feature selection, which is important to assure real time applicability for human-robot collaboration in industrial settings. Also in the field of industrial assembly tasks barely any free data sets are existing, especially for whole activities. This means the data sets for the HAR-framework have to be custom created. Also the HAR-framework has to be flexible, fairly cheap and easy to utilize, constricting the perception of the actions to camera based recording setups. Furthermore state of the art approaches are using all the recorded frames as input for CNN networks. However, this leads to a very high computational effort, as well as the possibility of learning wrong features for classification. Especially since the use case doesn't provide large data sets for the training, which is a requirement for the industrial context in regards to costs, data and time efficiency. This leads to the core question of this thesis: Which and how many features are necessary in order to identify an action from video recordings in the domain of industrial robot-human collaboration assembly tasks.

3.2 Proposed Solution

In this section the proposed solution is presented as human action recognition framework (HAR). The core idea lies in deploying supervised action classification based on deep neural network models. As seen in the the section that outlines the formal statement of the problem in 3.1, several aspects for recognizing actions, as well as domain specific challenges have to be considered. Since the HAR-framework is supposed to be applicable for human robot collaboration purposes for industrial assembly tasks it has to work in the given domain specific task, has to have be real time and generalization capabilities, has to use easy accessible hardware, flexible and is intuitively to use for workers. On top of that a high classification quality of the actions has to be ensured, which has to be considered for the feature selection. In order to combine all of those requirements, the proposed HAR-framework consists of four major parts:

1. Creation of a domain appropriate data set
2. Feature selection
3. Classification of actions with deep neural networks
4. Performance Analysis

Since there are no freely available data sets featuring industrial assembly tasks as activities, those have to be created at first. To this purpose, two different assembly tasks are setup, which are carried out by different participants. The executions of the tasks are recorded with an RGB-D camera, which allows to record RGB frames, as well as depth frames for feature selection.
The second part part focuses on the feature selection process. Since using the whole frame as data input is too expensive and might lead to incorrect feature selection for the classification, a pre-selection of features is done. Movement, involved objects and participants, or environment are providing important information for action recognition. However in the case of industrial assembly tasks, the environment can be neglected, since the lab environment doesn't change at whole, but can be cluttered. The information about the participants can be abstracted, since clothing and optics isn't relevant in this regard. In this case the movements of the participant, as well as the involved tools are selected as important features for identifying an action. To this purpose the

movements of the participants are tracked by the image coordinates of their joints with Openpose. Those coordinates are extended with a depth pixel values, because RGB as well as depth frames are available. Furthermore the objects are extracted with Yolo v4, which just needs a subset of the recorded images for object detection.
The third part covers the classification of the actions, given the previously selected features. Four different types of deep neural networks, one feed forward network and three variations of recurrent neural networks, are implemented, that leveraging the influence of certain features for increasing the classification capabilities by focusing on spatial and temporal dependencies.
Finally the performance analysis is carried out. Classification accuracy, discriminability of actions and challenges of different task setups, as well as feature importance are examined.
This leads to the own contributions that are made in this thesis:

- Developing a lightweight HAR framework: In order to minimize the costs and hardware overhead, just one, low priced depth vision sensor is used - the Kinect v2 with the embedding in Linux with an open source driver setup for depth cameras.

- Setting up two different assembly tasks for monitoring: In order to examine which features are important for recognizing actions in an industrial domain, two different industrial assembly task setups with different tools and different working steps are prepared. Even though the workspace and fixed camera position are the same, different level of occlusion and view of the work tools are occurring due to the structure between both tasks.

- Creating a domain appropriate data set: As there are no public available data sets of industrial manufacturing tasks that features a whole assembly pipeline, a custom data set for feature extraction and training the used machine learning algorithms has to be recorded. This data sets includes two different assembly tasks with pre-defined actions and predefined workpieces. The first scenario involves the construction of an elevator cabin hanger and the second scenario of a control cabinet door.

- Decreasing computational complexity with optimized feature extraction: To this purpose skeleton tracking is used to monitor the movements of the

collaborators or more precisely the 2D spatial information of their joints and the depth frames. Additional information is gained by using object detection. The combination of skeleton and object features makes for a low dimensional feature vector as input for the neural networks, which lowers computational costs while optimizing the classification accuracy. Reducing of data recordings due to less clutter.

- Analysing and identifying which of the extracted features of the assembly tasks are the most suited for human action recognition: To this purpose different variants of feed forward and recurrent neural networks are implemented in order to examine highly influential attributes and feature importance in regards to both assembly setups.

3.3 Complete Workflow

The proposed algorithm consists of four main parts. Those are the recording of the data set, the pre-selection of appropriate features, the classification via machine learning and the evaluation of the model performances. An overview of the detailed workflow of the HAR framework is observable in image 3.1. First the data sets are created. Two assembly tasks are chosen and designed in a way that several assembly steps are carried out, with hand overs of the needed tools in between. Also both tasks are varying in recording angle and movement of the constructors. For the recording of the data sets of the two assembly tasks the Kinect v2 is used. However the Kinect v2 is initially designed for Windows devices, which means that an open source driver like Libfreenect2 has to be deployed. This allows the camera to receive 1920x1080 RGB frames, as well as 512x424 depth frames. Those frames are used for the next step, which is the feature selection.
For human action recognition in this scenario, the movement of the participants, as well as objects related to the assembly tasks. To this purpose the open source software packages Openpose and Yolo v4 are implemented. Openpose uses the RGB recording in order to extract the skeletons of of the participants, which in return leads to 2D joint coordinates as features. Yolo v4 on the other hand is using the RGB frames as input for object detection, which in return provides probabilities of the occurrences of the tools for each frame. The depth frames on the other hand are used in order to provide the joint coordinates with a third dimension. To this purpose OpenCV is important,

which stores the recordings of the RGB frames as well as the Depth frames into OpenCV Mats. This is also important, since both types of frames have a different resolution and resize operations are needed in order to match the depth information accordingly to the 2D joint coordinates.
After the features are extracted, they are used as input for several, different machine learning models, in order to be able to classify and recognize actions. To this purpose feed forward neural networks, single stream LSTM networks [27], three stream LSTM networks [50] and temporal segment LSTM networks [47] are implemented. The different architecture of those for models, allows deeper analysis of the spatial and temporal attributes that are involved in the recognition of actions. The trained models are then used on unseen recordings for the analysis of their classification capabilities.
Finally the performance of the trained models are then evaluated with four different methods. The first is the confusion matrix, which provides information about the classification accuracy for each class, as well as false positives and false negatives classifications for each class. The second methods are probability density graphs. This allows a visualized classification probability for each class and each frame. Furthermore they serve as a visual aid in order to show the class estimations for the transition period between following classes. After that the mulit-class receiver operating characteristic curve gives insight about how difficult a certain action is classifiable for the different models. Finally heat maps are created in order to visualize the most important features as subset of the input features, which have the most weight for correct classification.

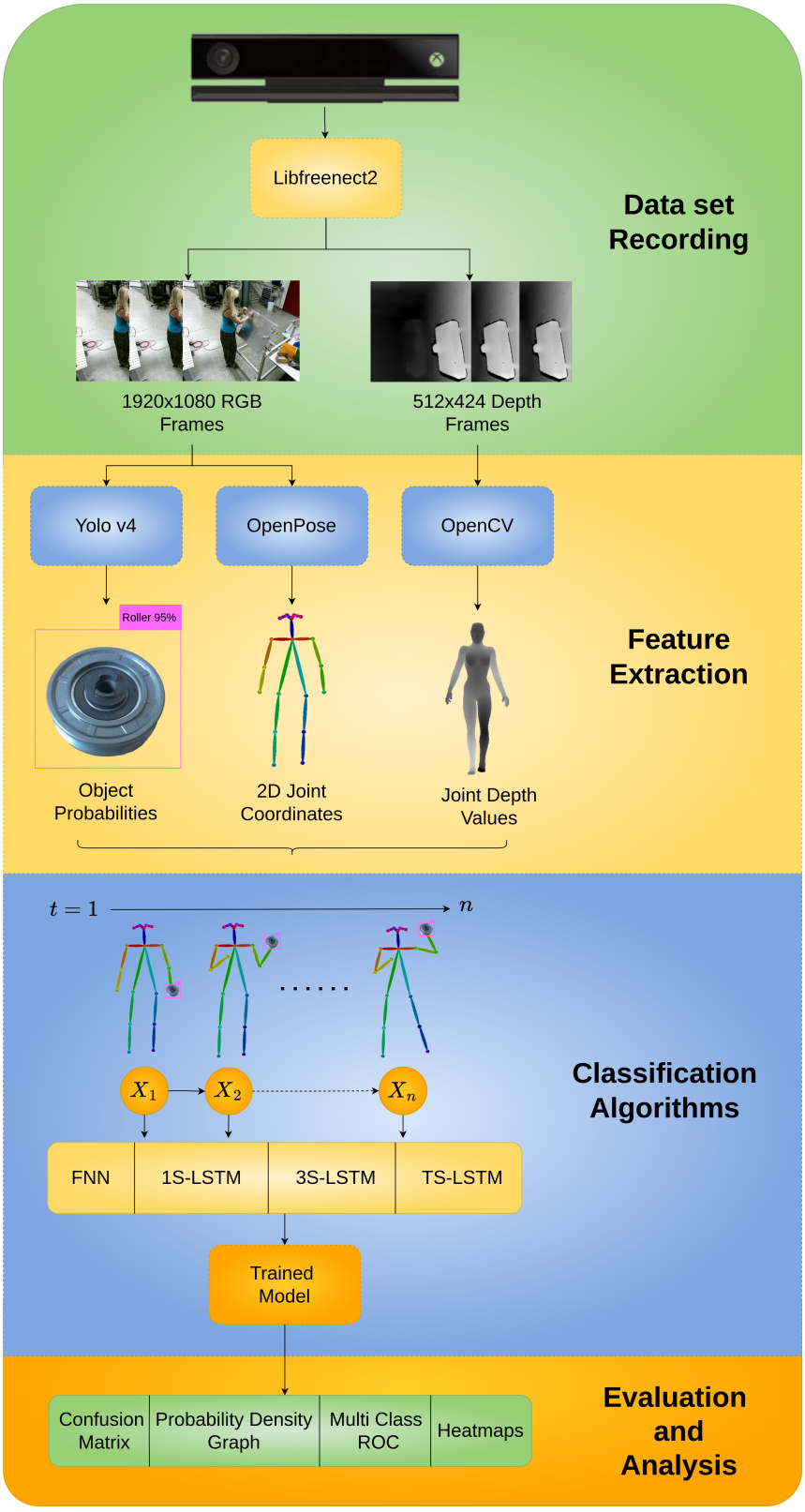


Figure 3.1: Overview of the complete HAR workflow, showing a detailed approach of the four main components

4 Background

The HAR framework encapsulates several components that are build by deep neural networks. After the recording of the data sets for both assembly tasks, the feature extraction is the next step. The object detection and processing of the 2D skeleton joint coordinates are relying both on convolutional neural networks (CNN). The machine learning models that use those features as input for the classification tasks are feed forward neural networks (FNN) and long short-term memory (LSTM) based neural networks. In this section the theoretical background of those networks is presented in order to provide a foundation for architecture and tuning choices later on.

4.1 Artificial Neural Networks

Artificial Neural networks (ANN) or often just called neural network (NN) are a subset of machine learning algorithms. They are inspired by the way of how the human brain works. Information is passed from neurons connected to other neurons connected by edges and only firing when a certain threshold is passed. Therefore ANN are networks that are build by artificial neurons. They are used as algorithms to approximate functions which are able to determine the correlation between the features and the target transforming the data set according to a layer of neurons. A representation of a typical, basic neural network is illustrated in figure 4.1.

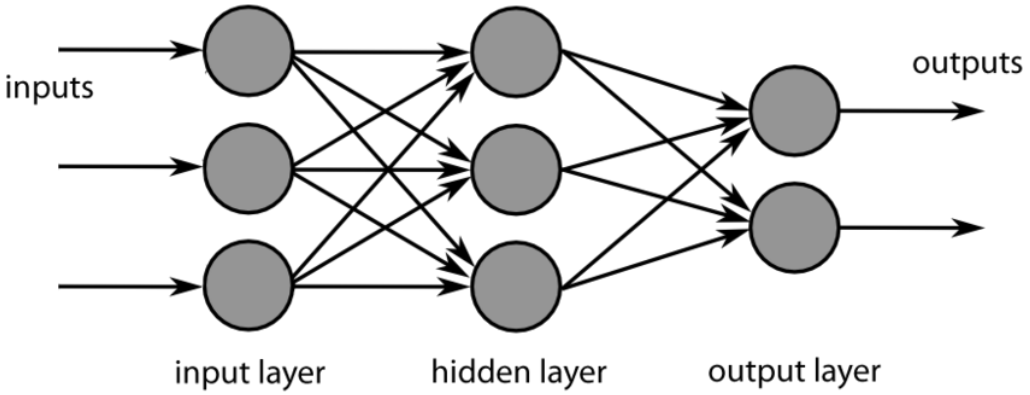


Figure 4.1: Model of a three layer feed forward neural network [13]

Most commonly they consist of three or more layers. One input layer, one output layer and several hidden layers in between. The input layer is a vector with the length equal to the number of input variables of the data set. The output layer consists of artificial neurons that describes the target variable, for instance the different classes in a multi-classification problem. A network with no hidden layer is called single-layer perceptron, as it consists just of the output layer filled with one neuron. The perceptron is not able to learn linearly separable patterns and is just able to compute binary classifications. For multi classification the number of nodes in the output layer has to be increased accordingly to the number of desired classes. However in order for the network to be able to learn non-linearly separable patterns, one or more hidden layers have to be introduced. This was proven by George Cybenko in 1989 and coined as the universal approximation theorem, which states that a neural network with at one hidden layer can approximate any continuous function for inputs with a certain accuracy [52]. Networks with one or more hidden layers are called multilayer perceptron (MLP) or deep neural networks (DNN) and are a subset of ANN. Adding more hidden layers, as well as the number of neurons in the hidden layers are tunable hyperparameters, which determine the accuracy of the function and successively of the classification.

4.1.1 Perceptron

A perceptron, artificial neuron or simply neuron is the elementary unit in an ANN. A schematic of the perceptron is displayed in image 4.2. As already

stated the basic perceptron is just one node and therefore just one layer, which creates the output $\hat{y}$. It gets a data vector $X = (x_1, ..., x_n)$ with $n \in \mathbb{N}$ as input and each variable is multiplied with a corresponding weight $W = (w_1, ..., w_n)$ with $n \in \mathbb{N}$. After that one bias value b per layer is added. The result of the summation is given to a activation function σ that applies the non-linearity to the output.

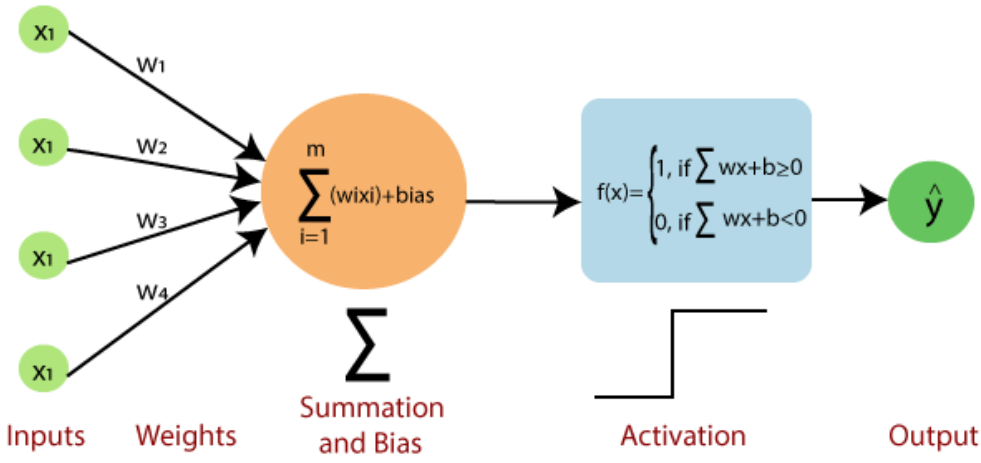


Figure 4.2: Structure of a perceptron [34]

Therefore the formal statement for the output of one perceptron is as follows:

$$\hat{y} = \sigma(b \sum_{i=1}^{n} w_i \cdot x_i + b) \tag{4.1}$$

Adding more perceptrons together creates a network with more layers and nodes, which are all described by a superposition of function 4.1. However each layer has different values for its weights, as well as a different bias. Also the activation functions are the same across all neurons in the hidden layers, while different activation functions might be used in the output layer depending on the classification or prediction task at hand. There are several different activation functions, with useful attributes depending on the given data set and classification task. Four popular activation functions are displayed in 4.3.

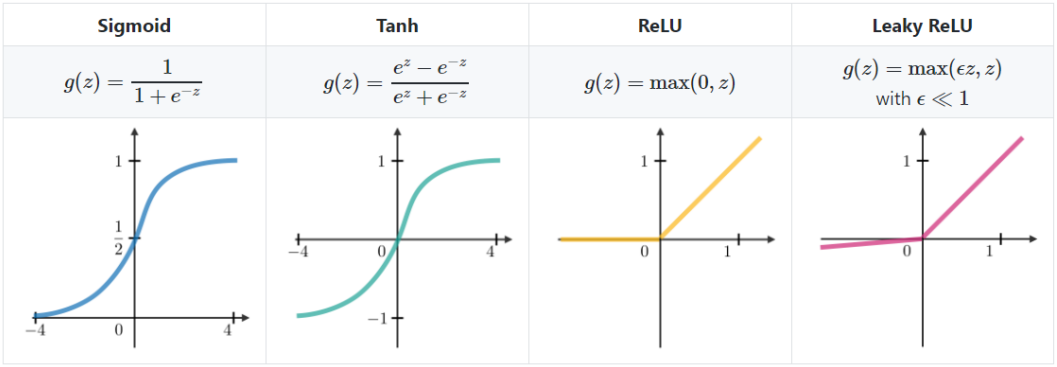


Figure 4.3: Curves and functional equations of sigmoid, hyperbolic tangent, ReLU and leaky ReLU as common activation functions for deep learning models [77]

The first is the sigmoid function and takes a real-valued number and outputs a value in the range between zero and one. This means large negative numbers become 0 and large positive numbers become 1. It is mostly used for binary classification tasks.
The hyperbolic tangent (tanh) function $tanh(x) = 2\sigma(2x) - 1$ transforms a real-valued number to the range $[-1, 1]$. Its output is zero-centered.
The Rectified Linear Unit (ReLU) has become very popular in the last few years. It computes the function $f(x) = max(0, x)$, which means that the ReLU function creates zero values for negative numbers and lets everything unchanged pass the threshold above zero. This avoids the vanishing gradient problem and gives better computation performance.
The softmax activation function is mostly used at the last layer of the neural network which calculates the probabilities distribution of the event over $n \in \mathbb{N}$ different events. This allows the function to handle multiple classes.

4.1.2 Training a Neural Network

As stated a neural network is used to approximate a function that is able to represent a correlation between the given input data and the desired output. In order to find the best fitting function, a neural network computes two steps: Forward propagation and backward propagation . First the input data is fed into the network and is forward propagated. This means an initial set of weights and biases will create an output depending on the given input. For

a classification problem an error function is used in order to compare the calculated output with the desired output and the goal is to minimized the error between both. Therefore calculated error is used for the back-propagation step.

Forward Propagation

Forward propagation indicates how a result is evaluated given an input vector $X = (x_1, ..., x_n)$ with $n \in \mathbb{N}$. The forward propagation will be explained on the neural network seen in image 4.4. The calculations of the output of a single neuron given inputs, weights and a bias is already explained in the perceptron subsection.

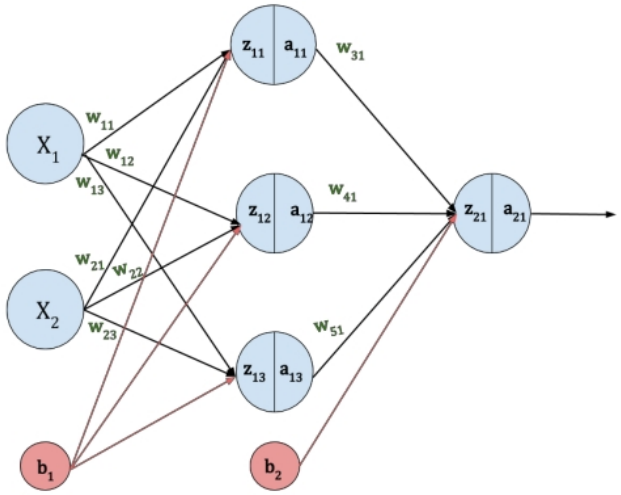


Figure 4.4: Neural network showing weights and bias connections during forward propagation [36]

For each neuron in a layer the output is computed as $a_i^1 = \sigma(w_{i1}x_1 + (w_{i2}x_1 + (w_{i3}x_1 + b_i)))$ and forward propagated as input for the next layer. This again follows the same rule $a_i^2 = \sigma(w_{i1}x_1 + (w_{i2}x_1 + (w_{i3}x_1 + b_i)))$, and yields the output $\hat{y}$

Error Function

After the computing of the forward pass the resulting output $\hat{y}$ and the desired output y are compared. This is done with an error function, also often times called loss or cost function and gives us a metric about how far resulting and desired output diverge from each other. The error function $\mathbb{L}(\hat{y}^{(i)}, y^{(i)})$ calculates the error for each sample i in a training set with $m \in \mathbb{N}$ samples, which leads to the total error depending on the weight matrices $W = [W^{(1)}, ..., W^{(L)}]$ for each layer $W^{(j)}$ with $j \in (1, ..., L)$:

$$J(W) = \frac{1}{m} \sum_{i=1}^{m} \mathbb{L}(\hat{y}^{(i)}, y^{(i)}) \tag{4.2}$$

There are several different option for calculating the error function. The first is the mean squared error (MSE). It is mainly used for regression problems and outputs a real-valued number.

$$J(W) = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}^{(i)} - y^{(i)})^2 \tag{4.3}$$

Cross-Entropy-Loss is used for classification tasks. The output is a probability value between zero and one. It can be applied to binary and multiclass classification. The binary cross entropy loss is:

$$J(W) = -\frac{1}{m} \sum_{i=1}^{m} (y^{(i)} log(\hat{y}^{(i)} + (1 - y^{(i)}) log(1 - \hat{y}^{(i)}) \tag{4.4}$$

For more than two classes a separate loss for each class label per training example is calculated and the result summarized, with S indicating the number of classes.

$$J(W) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{i=1}^{S} (y_c^{(i)} log(\hat{y}_c^{(i)} + (1 - y_c^{(i)}) log(1 - \hat{y}_c^{(i)}) \tag{4.5}$$

In order to prevent overfitting when dealing with a data set with a large number of features regularization techniques are used. Two famous regularization techniques are the L1 regularization or Lasso regression and L2 regularization or ridge regression. Both are adding a penalty term to the error function. L1:

$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$

L2: $\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$ The key difference between these techniques is that Lasso shrinks the less important feature's coefficient to zero thus, removing some feature altogether. So, this works well for feature selection in case we have a huge number of features

Backward Propagation

After calculating the error, the next goal is to determine weights and biases that are minimizing the loss function:

$$W^* = \underset{W}{argmin} J(W) \tag{4.6}$$

This is done by the so called backpropagation. The core concept is to update weights and biases in order to decrease the loss by taking the derivative of the loss function with respect to all parameters. This is done by applying the chain rule recursively to all biases and weights. An example of this can be observed in image 4.5. The output of the network is z and L is the loss function, while the inputs are x and y. Starting point is the derivative of L over z with $\frac{\partial L}{\partial z}$. Since the calculation of z is depended on both the input parameters x and y, the chain rule hast to be applied to each parameter individually, as can be observed in the equations depicted next to the inputs in image 4.5.

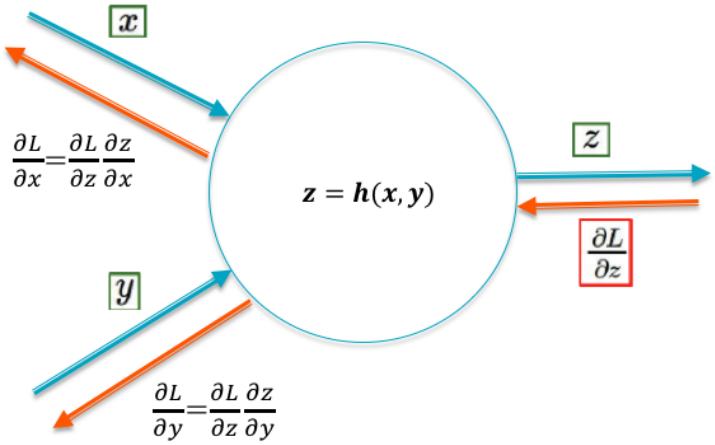


Figure 4.5: Influence of a neuron towards the loss function depending on its input weights and bias [78]

This allows the computation of the gradient descent algorithm, which is used to calculate the updates of the parameters. First the weights are randomly initialized. Secondly for all parameters the gradient $\frac{\partial J(W)}{\partial W}$ is computed. After that, the weights are updated with a learning factor η, which controls how many weights are updated during each training step, and with the calculation of $W = W - \eta\frac{\partial J(W)}{\partial W}$. Those steps are repeated until convergence of the parameter values is reached. However calculating the gradient creates several problems. The loss function has to be continous and differentiable, furthermore the calculation of all gradients is costly and also the vanishing gradient can lead to information loss. In order to reduce the complexity while calculating there are different variations of the gradient decent algorithm. One is the stochastic gradient descent (SGD). For SGD just a single example is randomly chosen and the weights updated accordingly. This speeds up the calculations, but its very noisy. Therefore an improved version of this is using a batch of several examples which leads to less noise.

4.1.3 Hyperparameter

Hyperparameter determine the quality of the trained neural network. This includes tuning a model in a way, that overfitting and underfitting are avoided. Overfitting means that the model fits exactly against the training data and

therefore prevents the model from generalizing well on unseen data. It can be spotted, if the model has a low error rate and a high error rate on the test data. Underfitting on the other hand means that the model isn't able to represent the correlation between the input and output variables, which is indicated by a high training error and a high test error. There are several hyperparameters in a network that are tunable in order to prevent overfitting or underfitting. The Hyperparameter are dependent on the chosen network architecture and the training process. Common parameter for the model architecture are the number of layers, the number of nodes in hidden layer and the loss function. More parameters are set for the actual training process, which are the number of epochs, the batch size and the learning rate. Furthermore different layer types require further parameter, e.g long short-term memory cells (LSTM) require windowsize and stepsize for their input data. The common hyperparameter are described in the following, layer specific parameter are outlined in their corresponding sections.

- Number of layers: One hidden layer is already enough to represent an arbitrary non linear function, according to to the universal approximation theorem [52]. But deeper networks ensure, that the network is able to learn and not just represent the function, which means a better generalization. Unfortunately there is no perfect answer for the best number of layers [57]. Making the network infinite deep isn't an option either. This can lead to overfitting, especially for smaller data sets and at some point the gradient will vanish, even though the LSTM units will prolong it. Therefore the number of layers is dependent on the number of nodes in the input layer.

- Number of nodes: In this case the number of nodes is dependent on the input dimension for the input layer. For the hidden layer

- Loss function: In this scenario the categorical crossentropy is chosen as loss function, which is chosen multi-class classification tasks by quantifiying the difference between the probability distributions for the predicted classes.

- Number of epochs: The number of epochs determines how many rounds the architecture is trained on the data set. Ideally it should be enough that the loss is converging near to zero. However is training loss still

decreasing, while the validation loss is increasing again, overfitting is likely to occur.

- Batch size: In this case a mini-batch approach is chosen. This means that the batch size is the number of samples from the training set, that is used to estimate the error gradient before the model weights are updated. A too big batch size may lead to worse generalization, too few samples may lead to a worse accuracy [7], since the influence of batch size is inverse proportional to the number of epochs. For computational purposes the batch size is chosen to match the power of the CPU and is therefore often a number by the power of two [38]. In this case a batch size in the range from 2^2 to 2^{10} is chosen for grid search. For LSTM based networks the batch size correlates strongly with the window size and is in general bigger than for normal feed forward networks.

- Learning rate: The learning rate controls how fast the model will adapt the weights of the given tasks and is a parameter of the optimizer. Lower learning rates indicate, that smaller changes are made to the weights each update. Thus more epochs are required and it is easier to get stuck in a local mimimum. Bigger rates on the other hand can result in too fast convergence and therefore to suboptimal solutions [57]. The learning rate is normally found by grid search, starting with bigger values and decreasing slowly.

- Optimizer: Optimizers are used to change the learning rate and weights of the network, in order to reduces the loss. The adaptive moment estimation optimizer (ADAM) is used for the training process. Compared to other optimizers it computes different learning rates for different parameters and it is chosen, because it works generally well without depending on hyperparameter tuning so much [40].

The tuning of those hyperparameter is done in two steps. First the ranges and values are chosen by research. In the second step the parameters are updated by the grid search method of SciKitLearn. Those steps are chosen, because grid search is a valid method for hyperparameter optimization, but the computation can get very expensive, if the search space is chosen to large [12].

4.2 Recurrent Neural Networks

In order to display temporal dependencies in sequential data, ordinary feed forward networks (FFN) aren't sufficient. This is the task of recurrent neural networks (RNN), which have an internal state to memorise temporal behaviour [68]. This means they remember information of previous inputs. Time series data based tasks like speech or handwriting recognition is made applicable by RNN. This is possible, because in comparison to FFN, RNN possesses loops. Additionally to forward connections, the neurons are possessing feedback connections to the previous layer or to themselves. There are different kinds of feedback connections, which are partitioned as follows. The first is the direct feedback. In this case the output of a neuron is used as another input and connection to itself. The second is the indirect feedback where the neuron is connect to the neurons of the previous layer. The third is the lateral feedback where each neuron is connected to all other neurons of the same layer.w The fourth is the full feedback where each neuron is connected to every other neuron. Figure 4.6 shows the architecture of a RNN. The variable x describes the input neuron and o the output neuron. h is the hidden layer that acts as the memory of the network. It is calculated on the current input and the previous time step of the hidden state. The output layer of the RNN is described by o. The connections between input layer and hidden layer, hidden layer to itself and hidden layer to output layer are respectively parameterized by the weight matrices U, V and W. This means the hidden state and the output are calculated as follows [56]:

$$h_t = f_h(Ux_t + Wh_{t-1} + b_h)$$

$$o_t = f_o(Vh_t + b_o)$$

In this case b_s is the bias and f_s a non-linear transformation function like tanh or softmax with $s \in \{h, o\}$.
The right side shows an unfolded version of a RNN into a full network which takes all the time steps from x_{t-1} to x_{t+1} of the sequence into account.

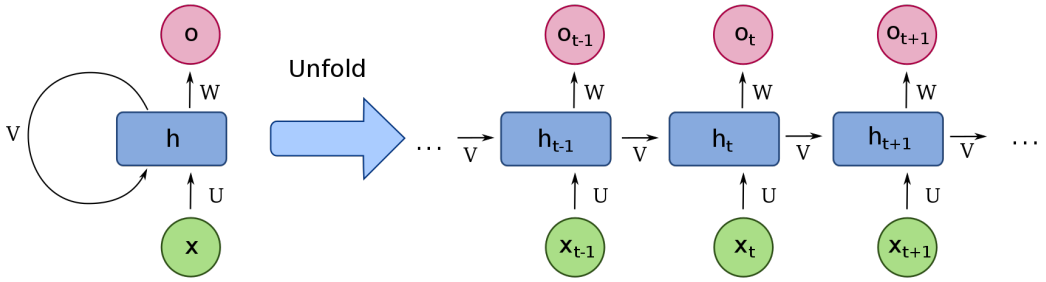


Figure 4.6: Unfolded RNN over $t + 1$ timesteps [14]

Similar to normal neural networks the parameters are trained with the help of back propagation through time (BPTT). As a result the loss will be minimized by the gradient descent method. Thereby the derivative of each activation function will be calculated iteratively. Because of that in each step the loss will be multiplied with a scaling factor for each layer in the network. If the value of this factor is between zero and one, the loss will become smaller and smaller and vanish eventually. Therefore this will lead to ineffective calculations of the weights. On the other hand if the values of the factor are bigger than one, gradient would eventually explode [63].
This means that the recalculation of the parameters of the hidden layers near the input layer have less impact on the output. Furthermore information that was passed early on in the sequence will be forgotten faster than information that will be passed at the end of the sequence, because of the back propagation. Therefore an improved version of the RNN is needed to prevent the problem of the vanishing/exploding gradient [27].

4.3 Long Short-Term Memory

Long short-term memory networks (LSTM) is a direct feedback based recurrent neural network, that is tackeling the problem of the vanishing gradient. The most basic version was proposed by Sepp Hochreiter [27]. It is able to store it's hidden state throughout time due to a memory cell in the hidden layer, an input gate and an output gate. Additionally Klaus Greff et al. [26] introduced the forget gate and shortly later the peephole connections into the architecture

[24] by Schmidhuber et al. This means that the forget gate, the input gate and output gate are connected to the present cell state C_{t-1}, while the output is connected to the updated cell state C_t. LSTM are based on a direct feedback. It possesses the capability to add or remove information to the cell state due to the functionality of the gates. This means that LSTM are like RNN chained one after another, but in contrast they are able to regulate the information that will pass on to the next state. Figure 4.7 shows the structure of an LSTM module.

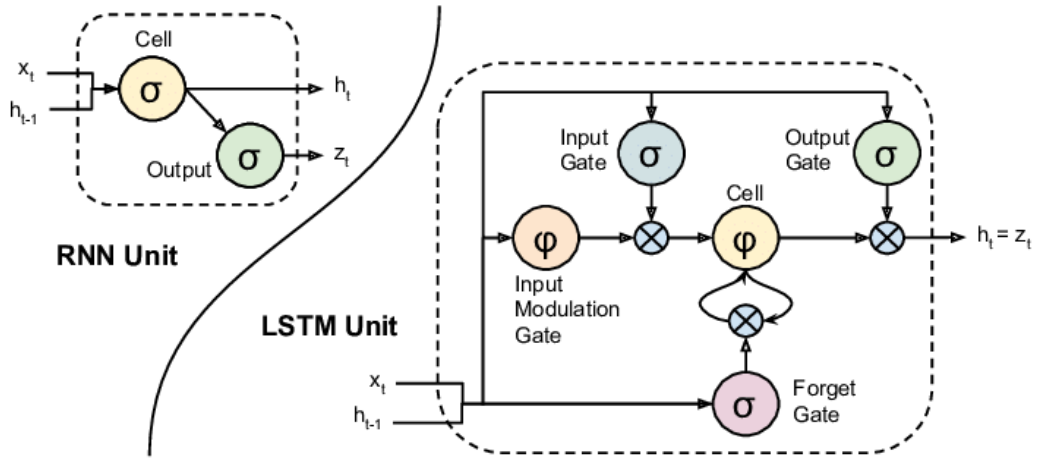


Figure 4.7: Overview of the structure of a RNN Unit and LSTM Unit in comparison [17]

x_t describes the input, h_{t-1} the previous hidden state and h_t the new hidden state. The hidden state is holding the information of earlier transmitted inputs and is used for predictions. On entering the cell, the sum of h_t and x_t named s_t is calculated. This result is used as input for the gates and impacts their behaviour. Furthermore it also used to update c_t.
[26] describes the components of the cell as follows with W_h being the weight matrix and b_h the bias vector with $h \in \{o, i, f, C\}$.
The forget gate regulates what information gets discarded from the cell state. Even though the basic form of the LSTM can prevent the vanishing gradient problem like normal RNN it is still possible that the LSTM can suffer from the exploding gradient.That's why the forget gate was added. There s_t is passed through a sigmoid function, which returns values between zero and one. Is the

result close to one c_{t-1} will be kept, is it close to zero it will be deleted.

$$f_t = \sigma(W_f \cdot (s_t, h_{t-1}) + b_f)$$

The purpose of the input gate is updating the cell state. Again s_t is pushed into a sigmoid function. Values near one decides which values are being updated, while values near zero are being discarded. This is multiplied with the result of s_t and shoved into a tanh function.The output of the tanh function lies between 1 and -1. This will help to regulate the network to prevent exploding values.

$$i_t = \sigma(W_i \cdot (s_t, h_{t-1}) + b_i)$$

$$\hat{C}_t = tanh(W_C \cdot s_t + b_C)$$

In the next step the old cell State c_{t-1} is updated to C_t. To this end c_{t-1} is multiplied with f_t to forget the information that was rendered uselsess. The result is added with the convolution of i_t and C_t, which decides how much the update is scale by s_t.

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \hat{C}_t$$

As depicted in figure 4.7 the cell state has a connection back to the forget gate. This is the key to preventing the vanishing gradient. This loop is called constant error carousel (CEC) and thus activity circulates in the CEC as long as the forget gate remains open. Just as the input gate learns what to store in the memory block, the forget gate learns for how long to retain the information, and once it is outdated to erase it by resetting the cell state to zero. This prevents the cell state from growing to infinity and allows the memory cell to store new incoming data, without overwriting the information from previous cells. The output gate will model the next hidden state h_t based on the cell state C_t. To prevent the cell state from blowing up, it will be transformed by a *tanh* function. This result is multiplied with the output of the *sigmoid*, which transformed s_t. This leads to the new hidden state h_t, which contains information on previous inputs.

$$o_t = \sigma(W_o \cdot (s_t, h_{t-1}) + b_o)$$

$$h_t = o_t \cdot tanh(C_t)$$

Both the new cell state c_t and the new hidden state h_t is carried over to the

next time step. The last components are the peephole connections. When the output gate is closed, none of the gates have access to the CECs they control. The resulting lack of essential information may interfere network performance. The peephole connections allow the other gates to take a look at the cell state and therefore the gates gain access to the CEC [24].

$$f_t = \sigma(W_f \cdot (s_t, C_{t-1}, h_{t-1}) + b_f)$$
$$i_t = \sigma(W_i \cdot (s_t, C_{t-1}, h_{t-1}) + b_i)$$
$$o_t = \sigma(W_o \cdot (s_t, C_t, h_{t-1}) + b_o)$$

Therefore the ability to handle vanishing and exploding gradients enabled by the CEC, makes LSTM a very strong candidate for handling time dependent data. It is able to handle short term, as well as long term dependencies.

4.4 Convolutional Neural Networks

Convolutional Neural Networks (ConvNet/CNN) are one of the backbones of computer vision. They are able to take images as input and is used mainly for image classification tasks. Since the input is an image, the input vector would have a high dimension resulting by the image shape width x height x depth. Regular feed forward neural networks aren't able to handle variations like translation, rotation and illumination in images. Furthermore sing fully connected layers like in regular NN, would lead to a very height number of weights for the gradient decent algorithm. In order to avoid that, the spatial properties of the input image are exploited. This means each pixel (with the shape 1x1x3 for x and y coordinate and 3 for the channel number) has common attributes as its neighbor ones. This allows the usage of kernels over a local area, which reduces the amount of weights. A standard architecture for CNN is seen in image 4.8. It consists of several layers, that are typical for the CNN architecture like

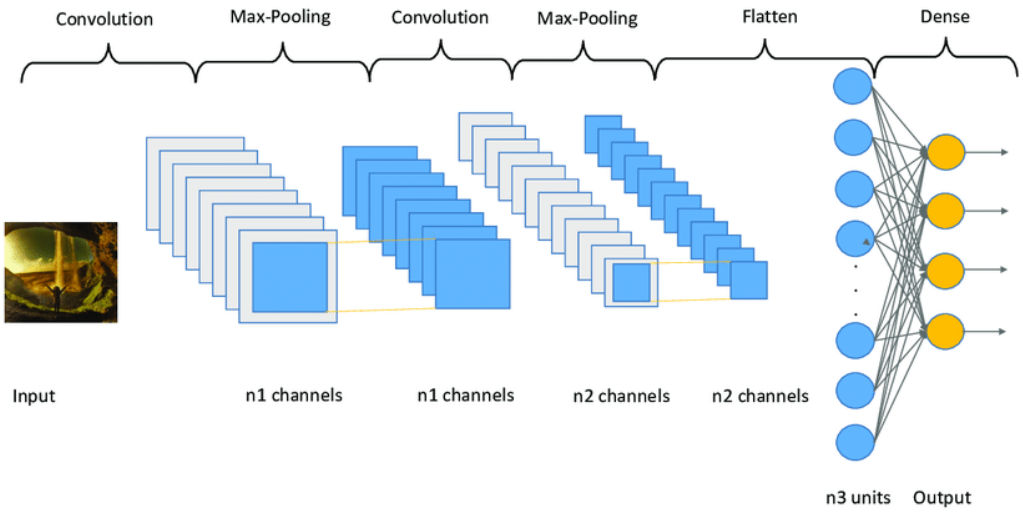


Figure 4.8: Structure of a Convolutional Neural Network [23]

convolutional layer, Pooling layer and a fully connected layer. Convolutional layers are the building blocks of CNNs are filters or kernels. Kernels are used to extract the relevant features from the input using the convolution operation with a part of the receptive field of the image. Convolving an image with a kernel results in a feature map. The size of a kernel is called stride and can vary, but is always a lot smaller than the input image. Making the kernel a lot smaller than the input image leads to sparse interaction. Because of that it is possible to store fewer parameter with shared weights for the layers, while gaining the same information density, which reduces computational complexity. Furthermore due to the kernel sliding over every pixel, we have an input of size W x W x D and Dout number of kernels with a spatial size of F with stride S and amount of padding P, then the size of output volume can be determined by the following formula:

$$W_{out} = \frac{W - F + 2P}{S} + 1$$

Since the convolution operation is a linear operation, a non-linearity layer is placed right behind it featrung sigmoid, tanh or ReLU functions. The pooling layer is another methods for reducing the computation complexity. It sum-

marizes the outputs of the convolutional layers, which are features that are found in a location in the feature map. There are numerous pooling functions, like L2 norm w=or weighted average, but the most popular is max pooling. Max pooling returns the maximum value of a neighborhood slice. The output volume of the pooling layer is

$$W_{out} = \frac{W - F}{S} + 1$$

After reducing the number of parameters with the help of convolutional and pooling layers, a fully connected layer is applied. It helps a to calculated the representation between the input and output.

4.5 Object Detection

Object detection is a computer vision task in order to find object in a digital images marked by a bounding box, as well as classifying those objects [66]. It is a sub task of object recognition. Object recognition can be subdivided into image classification, object localization, object detection and object segmentation. According to the annual ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [85] those three categories can be defined as follows:

- Image Classification: This predicts the the object categories, which are present in a given image. The input of the algorithm is a digital image and the output is a list of class labels.
- Object Localization: This kind of algorithms are detecting the locations and scales of objects in an image and indicating those with a bounding box. Given a digital image with several objects, it delivers bounding box as outputs, which are define by the center point, width and height.
- Object Detection: This approach is combing both of the aforementioned methods. The goal is to identify the types of objects in a given image, as well as the location of those objects, marked with a bounding box. Therefore the output of a object detection algorithm delivers a list of label, as well as a list of corresponding bounding boxes.

There are several techniques for object detection like aggregate channel features (ACF) SVM classification using histograms of oriented gradient (HOG)

features or the Viola-Jones algorithm for human face or upper body detection. However a lot of popular techniques are based on deep learning models, which are mainly build by convolutional neural networks. There are two predominantly architectures used as state of the art for object detection - the single stage detector and the two stage detector. Am overview of both architectures can be seen in image 4.9. Both architectures are build by a back-bone, neck and prediction head, while the two stage-detector also contains a region-based prediction head. First the backbone is given an image as input and is using a convolutional neural network in order to extract its features. In the neck

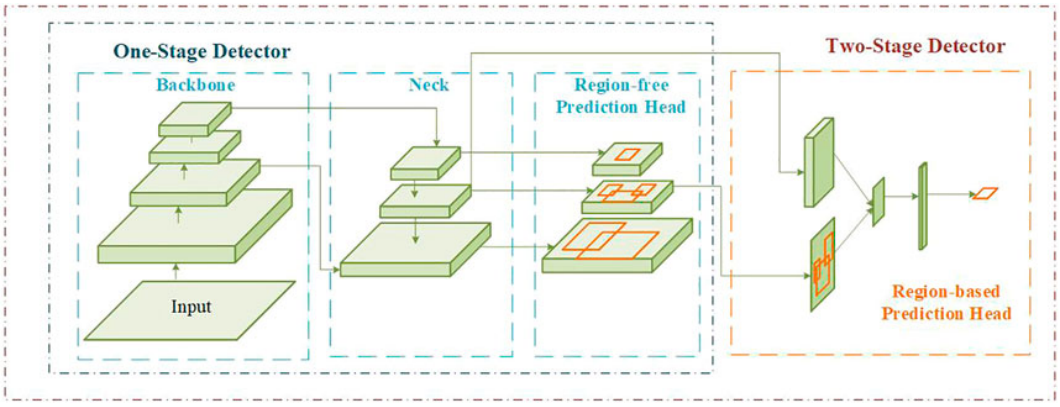


Figure 4.9: Two and One Stage Detector [84]

4.5.1 Two Stage Networks

Often used variants of the two stage networks are based on Region-Based Convolutional Neural Network (R-CNN) and its variants. In the first stage sub sets of the image are chosen as region proposals where an object might be present. In the second stage those region proposals are used for the object classification. Module 1: Region Proposal. Generate and extract category independent region proposals, e.g. candidate bounding boxes. Module 2: Feature Extractor. Extract feature from each candidate region, e.g. using a deep convolutional neural network. Module 3: Classifier. Classify features as one of the known class, e.g. linear SVM classifier model. YOLO Model Family.
While the accuracy of two stage networks is very high, the feature extraction of each region leads to a high computational complexity with a very high computation time.

4.5.2 One Stage Networks

In order to provide real time applicability, one stage networks are widely used, especially with the You Only Look Once algorithm (YOLO). In this case no region proposals are generated. Instead bounding-boxes and object classification are done directly. Instead of using region proposals, the image is partitioned by a grid structure.
Based on object detection is object segmentation, where an object is not marked by a bounding box, but rather by the prediction of specific pixels belonging to the object.

5 Validation

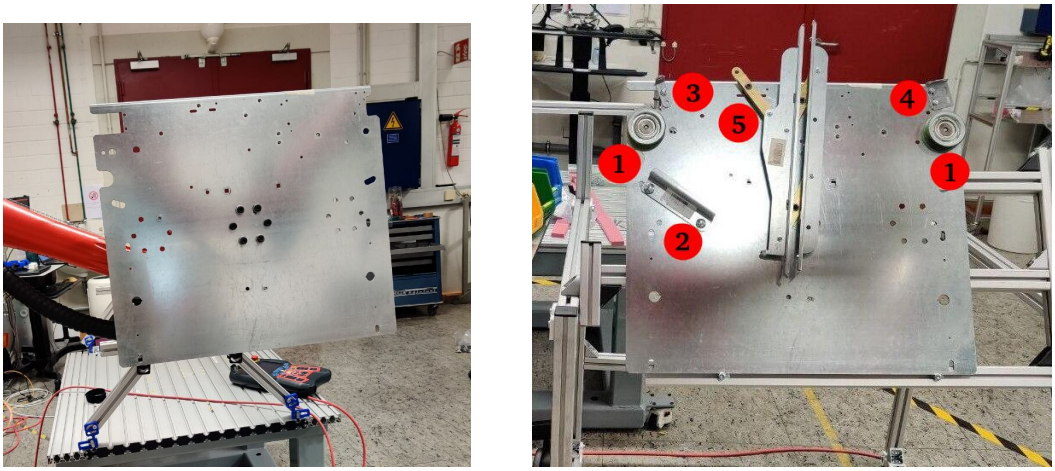
In this chapter the test setup as well as the used hardware and software for the human action recognition (HAR) framework are presented. Since the goal is to recognize human actions during the execution of industrial assembly tasks, first a suitable data set has to be created which later on can be used for the action classification. Therefore the very first step encapsulates the design and set up of two feasible industrial assembly task scenarios involving several action stages and the hand over of various work pieces. To achieve that, the following steps of the HAR framework are divided into two major implementations. The first is a C++ application with the main purpose of recording the data set, which encompasses an interface for registration, pre-processing and storage of the color and depth streams of a RGB-D camera. On top of this the feature extraction is implemented, which includes retrieving skeleton poses and object availability. The Kinect v2 is used as sensor with libfreenect2 as driver for Ubuntu 18.04 and serves as interface for further processing of the RGB and depth streams in Openpose and YoloV4.
The second module is a Python implementation for the human action recognition pipeline. This encompasses the pre-processing and the labelling of the extracted features for a supervised learning approach, which is realised with four different networks, which are feed forward neural network, a single stream LSTM a three stream LSTM and a temporal sliding LSTM. Finally the methods for analysing the classification performance of the models, as well as the feature importance for human action recognition are described. For the implementation Python 3.7 is used as programming language and Keras with Tensorflow as deep learning framework. For the training a GPU server with Geforce RTX30 with 12 GB GDDR5 memory, 4vCPUs and 15GB RAM is used. All the code, the scripts and the results are available in a public repository in my GitHub profile [37].

5.1 Assembly Tasks

In order to enable the HAR framework to recognize different industrial assembly tasks, it has to be trained and evaluated on appropriate data sets. Two different scenarios, which are involving their own unique assembly steps and work pieces are chosen in order to be able to detect and analyze challenges regarding human action recognition. Each scenario consists of an assembly activity, which is subdivided into several actions. The actions on the other hand are indicated by the installation of various work pieces or movements of the operator related to the tasks. Both set ups share similarities in getting handed over work pieces, screwing on work pieces and sharing the same laboratory as environment.
The first scenario includes the assembly of an elevator cabin hanger and the second setup consists of the assembly of a control cabinet door.

5.1.1 Elevator Cabin Hanger

The first setup involves the assembly of an elevator cabin hanger. The initial state of the task is seen in image 5.19a on the left side. In the beginning the empty hanger plate, which is measuring 100cm x 100cm, is placed on a metal frame 140cm above ground making it therefore easily reachable for assembly operations while the operator is standing straight in front of the hanger. The final state of the fully assembled elevator cabin hanger is seen in 5.19b on the right side.
The Kinect v2 is placed above the working space, in order to minimize occlusion, recording the execution of the task from a side angle.
Image 5.3 is showing all twelve stages involved in the completion of the elevator cabin hanger task, which can be subdivided into eight actions. Five of those actions are requiring work pieces, which are listed in 5.2 in the order they are occurring throughout the task execution. The work pieces are two rollers, a labyrinth hanger, a bumper bracket, a shunt bracket and an anti-opening device, which are also tracked by the object detection implementation.
In the beginning of the assembly task the operator stands relaxed and with arms down in front of the hangar plate, which indicates the starting position and marks the first action. After that the operator turns around and receives the first work piece. Handing over work pieces is classified as a second action.



(a) Empty hanger plate (b) Assembled hanger plate

Figure 5.1: Frontal view of the workspace for the elevator cabin hanger task for the starting configuration a) and fully assembled after completing the task execution b), which shows an enumeration of the involved work pieces. The corresponding images for the work pieces can be seen in figure 5.2.

1. Roller 2. Labyrinth Hanger 3. Shunt Bracket

4. Bumper Bracket 5. Anti-Opening Device

Figure 5.2: All five work pieces that are installed on the elevator cabin hangar plate.

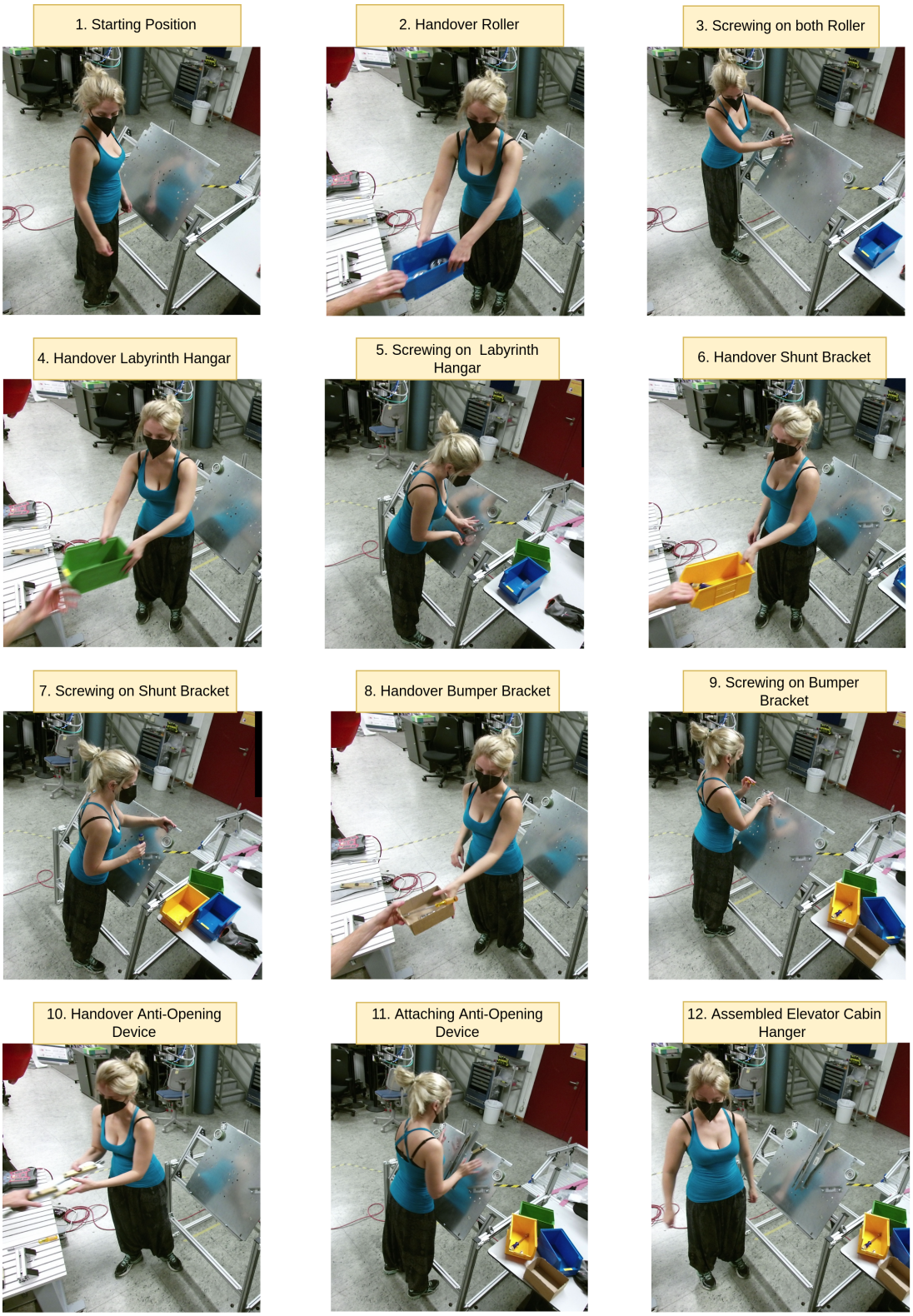


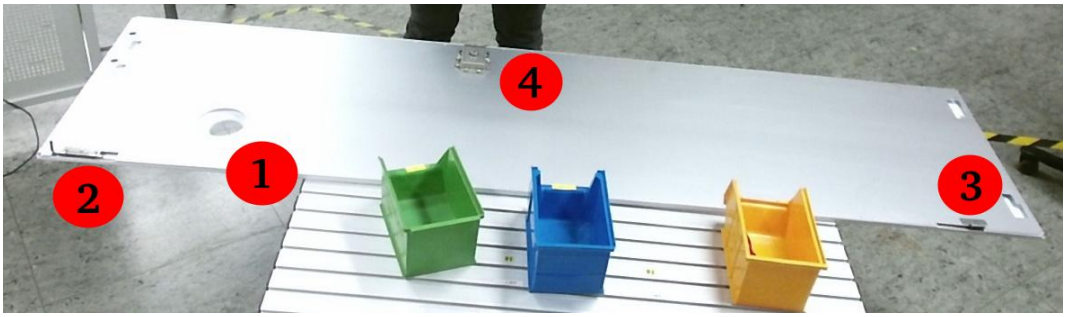
Figure 5.3: Elevator cabin hanger task assembly steps

The third action involves the two rollers that are screwed on to the hanger. Each of them is placed on the upper side of the hanger, left and right respectively. Next the installation of the hangar labyrinth follows as fourth action. It has to be screwed in the right part of the middle section of the hanger. After that, above the location of the right roller, the shunt bracket is screwed on with two M8x20 screws as fifth action. As sixth action the bumper bracket is also screwed on with two M8x20 screws above the left roller. Next the anti-opening device is plugged in the middle of the hangar as seventh action. After completing the assembly, the operator faces towards the camera with arms down as final position, which marks the eight and last action. In summary the eight distinct actions are the starting position, handover of tools, screwing on both rollers, screwing on the labyrinth hanger, screwing on the bumper bracket, screwing on the shunt bracket, plugging in of the anti-opening device and the ending position.

5.1.2 Control Cabinet Door

The second setup is the control cabinet door scenario. The initial state of the task is seen in image 5.4a. In the beginning the white door plate, which is measuring 200cm x 80cm, is placed on a table placed 100cm above ground thus allowing a top view on the operation space with few occlusion. The final state of the fully assembled control cabinet door is seen in 5.4b.
The task is divided into seven different actions that have to be performed by the operator. The Kinect v2 is placed above the working space, in order to minimize occlusion, recording the execution of the task from the front. Image 5.6 is showing all ten stages involved in the completion of the control cabinet door task, which can be subdivided into seven actions.
Four of those actions are requiring work pieces, which are listed in 5.5 in the order they are occurring throughout the task execution. The work pieces are a white ring, a pin hinge, a bottom hing and a locker, which are also tracked by the object detection implementation. In the following the directions of left and right are used from the view point of the operator towards the workplace. In the beginning of the assembly task the operator stands relaxed and with arms down in front of the table on which the backside of control cabinet door lies. This indicates the starting position and marks the first action. After that the operator turns and goes a little to his left in order to receive the first work piece. Handing over work pieces is classified as second action. The third action

(a)



(b)

Figure 5.4: Frontal view of the workspace for the control cabinet door task for the starting configuration a) and fully assembled after completing the task execution b), which shows an enumeration of the involved work pieces. The corresponding images for the work pieces can be seen in figure 5.5.

1. White Ring

2. Pin Hinge

3. Bottom Hinge

4. Lock

Figure 5.5: All four work pieces that are installed on the control cabinet door plate.

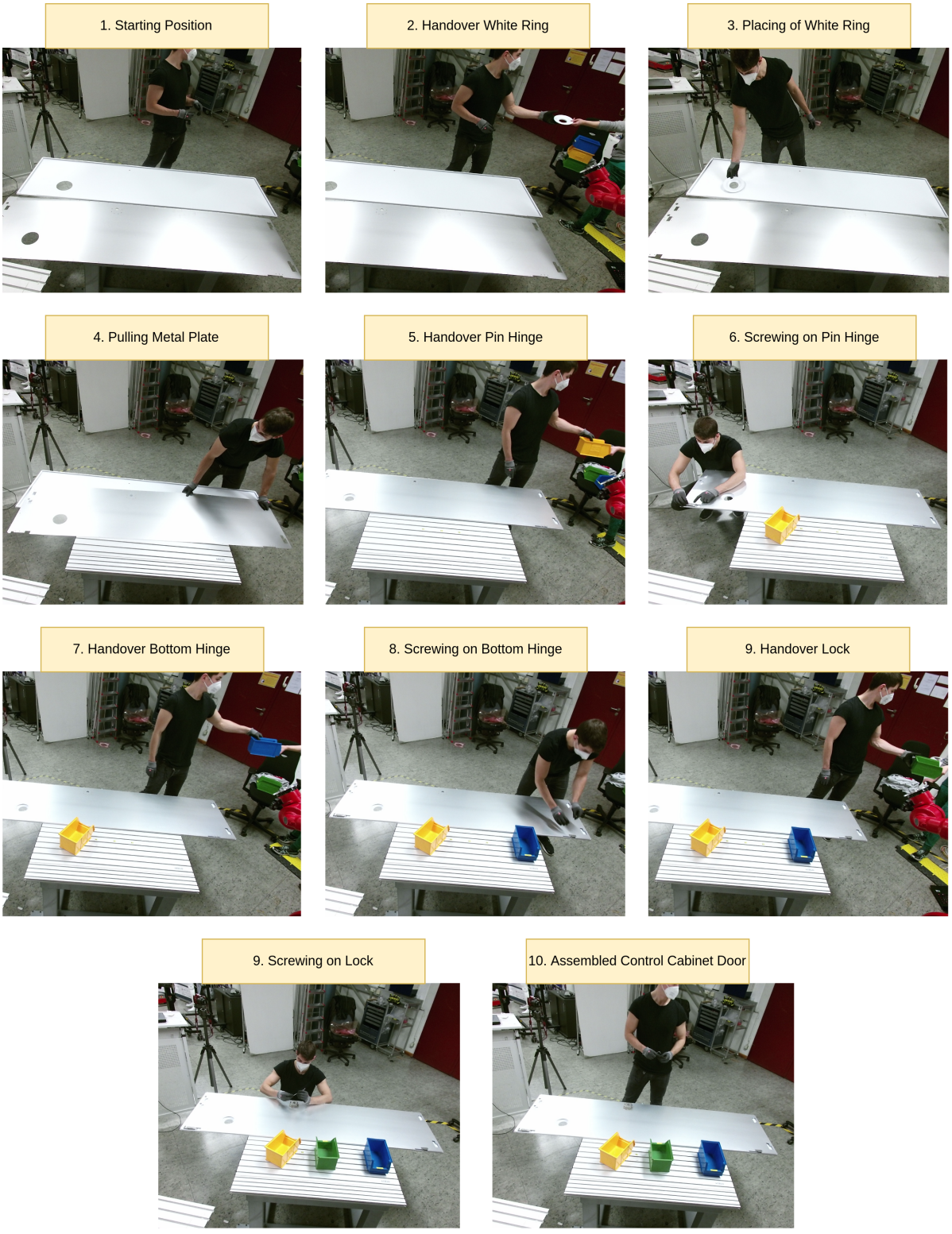


Figure 5.6: Control cabinet door task assembly steps

involves the the white ring, which the operator places in the cut-out segment of the back plate. Next the the metal plate that lies in front of the back plate, is pulled over the back plate and aligned on the corresponding edges as fourth action. Next the fifth action involved the installation of the pin hinge on the right top edge of the control cabinet door. After that the bottom hinge is screwed on the top left edge and marks action six. The seventh and final action involves the installation of the lock in the bottom middle of the control cabinet door. In summary the seven distinct actions are the starting position, handover of tools, placing of the white ring, pulling of the metal plate, screwing on the pin hinge, screwing on the bottom hinge and screwing on the lock.

5.2 Recording Setup

The hardware setup for the HAR-framework consists of a recording device for RGB and depth frames - the Kinect v2 and a portable notebook. For the recording of the assembly tasks a C++ application was developed, that allows the registration of RGB and depth channels separately. This is done with libfreenect2 as driver for Ubuntu 18.04. The RGB frames are saved as AVI files with the help of the OpenCV library, that is also used for saving the depth frames into CV Mats.

5.2.1 Kinect v2

The Kinect v2 is a RGB-D sensor, which combines a RGB camera and an infrared projector [54]. It was created by Microsoft Corporation and released in 2014. It was originally used for its motion-sensing capabilities for the X-Box One gaming console. The users were able to control the software with their motions instead of a gamepad. An illustration of the Kinect v2 can be seen in image 5.7. It comes with a RGB camera, infrared camera, infrared emitter and a multi array mic. The color camera has a resolution of 1920x1080 and the depth camera of 512x424. Compared to other RGB-D sensors, which are mainly base on structured light (SL) feedback, the Kinect v2 utilizes the time-of-flight (ToF) approach [39].
The illustration 5.8 explains the mechanism of the ToF approach. In a continuous-wave ToF camera, light from an amplitude modulated light source is back scattered by objects over a distance d in the camera's field of view, and the phase

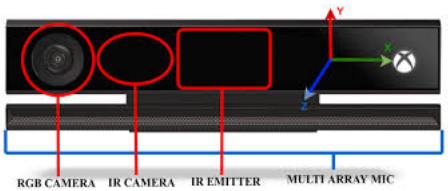


Figure 5.7: Kinect v2 [76]

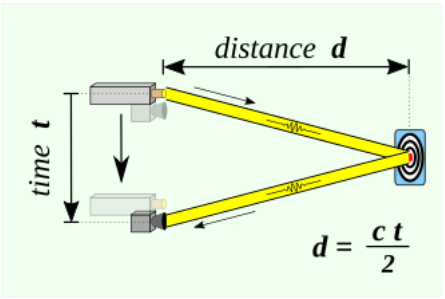


Figure 5.8: Time of flight [15]

delay, depicted as time t, of the amplitude envelope is measured between the emitted and reflected light. This phase difference is calculated by $d = \frac{ct}{2}$ into a distance value for each pixel in the imaging array.

5.2.2 Libfreenect2

Libfreenect2 [20] is an open source driver software for the Kinect v2, which makes the camera usable with the used version of Ubuntu 18.04 and other Ubuntu distributions. It is written in C++ and was first published in 2015. It provides functions for RGB image transfer, IR and depth image transfer as well as registration of RGB and depth images. This allows to receive a stream of 1920x1080 RGB frames, as well as a stream of 512x424 depth frames.

5.2.3 OpenCV

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision and machine learning [59]. It was originally developed by Intel in 2006 and supports cross-platform development for C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS. It provides functions and data structures for computer vision, like the Mat class, which is used to store the values of an image as an n-dimensional array. Those Mats are allowing fast processing of images like resizing, cropping, stacking and many more. Additionally it allows to create video recordings out of RGB frames. For the the HAR-framework the bytes of the incoming RGB and depth frames are stored in Mats, while the RGB Mats are additionally converted to an .avi video recording.

5.3 Feature Extraction

The next part of the HAR framework involves the feature extraction of the saved recordings. Instead of simply feeding all frames into suitable machine learning models, a pre-selection of features is done. This ensures a lower computational time and is less dependent on hardware requirements. Furthermore focusing on features that are specific to certain actions is serving the classification quality.
To this purposes an application for feature extraction is developed, that lies on top of the recording application, which is described in section 5.2 and is also written in C++. It extracts the skeleton joint coordinates of the assembly operators and the tools used for the assembly tasks out of the recordings for each frame. In order to extract the kinetic movements Openpose is presented for the tracking and extraction of the skeleton joint coordinates. The object detection involves the image labelling software Roboflow and in the following YoloV4 as network implementation.

5.3.1 Openpose

Openpose is an open-source library for real time multi-person 2D human pose estimation and was developed by researchers at the Carnegie Mellon University [8]. It is originally written in C++ and Caffe. The RGB frames of the recorded assembly task are used as input data and it outputs a skeleton based visualization of the poses of the the featured humans and provide a set of 2D locations for all joints corresponding to a skeleton. Openpose allows to display the skeleton in different formats. In this thesis the COCO format is used. It stores the 2D coordinates of 18 joints. Image 5.9 shows an overview of which joints of the human body are extracted.
The main idea behind Openpose consists of finding the locations of joint candidates of the input frame and encoding the association between those joints, in order to form limbs out of matching joint candidates, which are ultimately assembled to the skeleton. To this purpose the Openpose architecture contains two major parts, which are a multi-stage CNN and a weighted bipartite matching approach.
An illustration of the multi-stage CNN is shown in image 5.10. First the multi-stage CNN receives a feature map $\mathbf{F}$ as input, which is generated by preprocessing the images of size $w \times h$ with a channel size of three by a VGG19

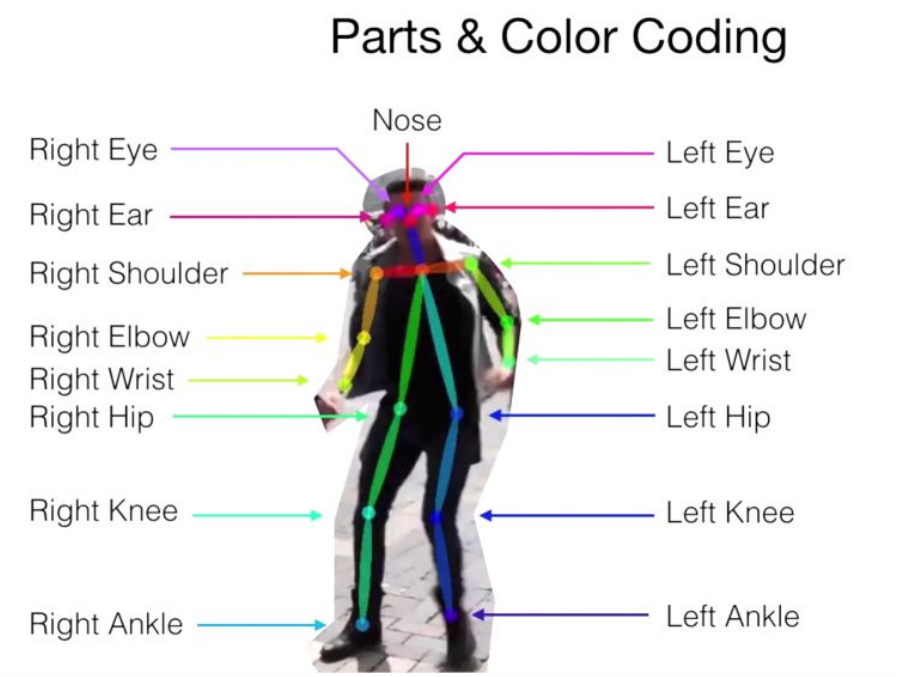


Figure 5.9: Openpose skeleton representation in COCO format [22]

CNN [73]. Depicted in blue and orange are the two stages of the network, which are repeated for several iterations. The first stage in blue creates a set of part affinity fields (PAF). The PAF are encoding the degree of association from location and orientation between pairs of detected joints as set of 2D vector fields $L = (L_1, L_2, ..., L_C)$ with $L_c \in \mathbb{R}^{w \times h \times 2}, c \in 1...C$. This means there are C possible joint combinations and for each is a PAF encoded in a matrix of the dimension $w \times h$ with a 2D vector for each entry.
After the prediction of the PAF, the believes about the joint locations of each image are predicted as confidence maps in the second stage, which is depicted in orange. The confidence maps $S = (S_1, S_2, ..., S_J)$ with $L_j \in \mathbb{R}^{w \times h}, j \in 1...J$ are created for each of the J joints and are matrices with the dimension $w \times h$ and each entry being filled with a scalar to indicate the confidence about the location of the joint for this pixel. The first iteration of each stage the initial PAF and confidence map are calculated

$$L^t = \Phi^t(F), t = 1$$
$$S^t = \phi^t(F, L^{T_P}), \forall t = T_P$$

with Φ and ϕ referring to the multi-stage CNN fir inference at the given iteration stage t and T_P and T_C referring to the number of iterations of the stages involving the PAF and confidence map respectively. First the PAF is produced and after T_P iterations for refinement the confidence map is calculated and refined for another T_C iterations.

$$L^t = \Phi^t(F, L^{t-1}), \forall 2 \leq t \leq T_P$$
$$S^t = \phi^t(F, L^{T_P},), \forall T_P < t \leq T_P + T_C$$

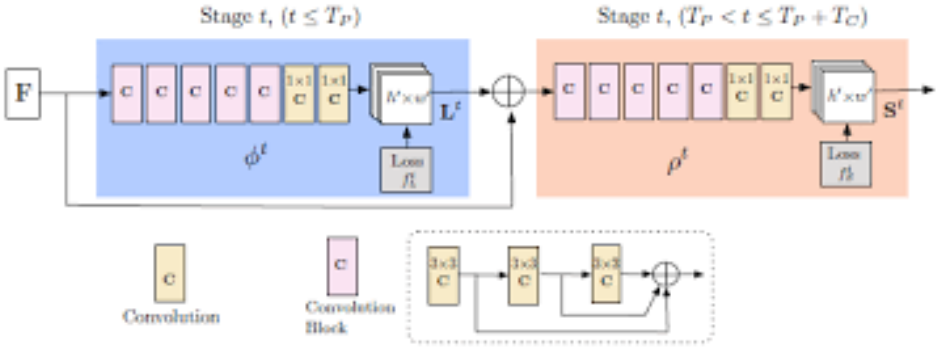


Figure 5.10: Architecture of multi-stage CNN [8]

At the end of each stage the L2 loss between the estimated predictions and the ground truth maps and fields is calculated, in order to monitor the vanishing gradient.

$$f_L^{t_i} = \sum_{c=1}^{C} \sum_{p} W(p) \cdot ||L_C^{t_i}(p) - L_C^*(p)||_2^2$$
$$f_S^{t_k} = \sum_{j=1}^{J} \sum_{p} W(p) \cdot ||S_j^{t_k}(p) - S_j^*(p)||_2^2$$

In this case L_c^* is the ground truth part affinity fields, S_j^* is the ground truth part confidence map, and W is a binary mask with W(p) = 0 when the annotation is missing at the pixel p. The ground truth for the confidence map S_J^* is generated by 2D joint annotations for the corresponding locations in the image. The ground truth L_C^* of the PAF is also generated by the annotated

2D joint locations, however for each pair of joints, it is calculated if a pixel is associated with a pair and this pixel get a unit vector of one pair pointing from one involved joint to the other. For all other pixel this vector is zero valued. The overall objective is given by:

$$f = \sum_{T_P}^{t=1} f_{\mathbf{L}}^t + \sum_{T_P+T_C}^{t=T_P+1} f_{\mathbf{S}}^t$$

However the network outputs several proposals for body locations for each confidence map. In order to get the limbs for the final skeleton the proposed body part detections have to be matched correctly with each other, which draws information from the PAF. A measurement of a PAF L_c corresponding to a pair of joint locations is given by

$$E = \int_{u=1}^{u=0} L_c(p(u)) \cdot \frac{d_{j2} - d_{j1}}{||d_{j2} - d_{j1}||_2} du$$

In this case $p(u) = (1-u)d_{j1} + ud_{j2}$ and interpolates the position of the joint locations d_{j1} and d_{j2}. This has to be done for all possible candidates of joint pairs and results in a maximum weight matching problem, with the nodes of the graph are the candidate body parts, and the edges are the candidate limbs formed from these body parts, with the weight of each edge being its score. Given the constraints of using a minimum spanning tree for the skeleton and not the whole graph and splitting the matching into bipartite submatchings the following optimization problem has to be solved:

$$\underset{Z}{max} E = \sum_{c=1}^{C} \underset{Z_c}{max} E_c$$

E_c is the overall weight of the matching from limb type c, Z_c is the subset of Z for joint pair c. The hungarian algorithm is used for solving this linear optimization problem in polynomial time [44]. Finally the skeleton is assembled to a full-body pose by connecting all limbs, which are sharing the same joints. This results in a set of 2D locations for the 18 joints as output for each person existing in the frame.

5.3.2 Depth Values

Openpose provides skeletons in COCO format with 18 key points as 2D coordinates. In order to retrieve depth information for the joint coordinates, the 2D skeleton and the depth frames have to be fused. The 2D skeleton is retrieved from 1920x1080 sized frames, while the depth frames have a resolution of 512x424. Therefore OpenCV is used to upscale the depth frames which are stored in OpenCV Mats to 1920x1080 and match the 2D joint coordinates with the corresponding coordinates in the depth frame and retrieving the local depth pixel values. This increases the number of features by 18 depth values per frame.

5.3.3 Roboflow

Roboflow is an online open-source tool for image labelling [32]. Hand selected frames of the videos are uploaded and the objects corresponding to the tools used in both assembly task, which are depicted in table 5.2 and 5.5 are outlined with bounding boxes and marked with their corresponding label. 1000 frames per assembly task are chosen, since this is the freely available capacity. Furthermore the data set is augmented with greyscale, hue ($\pm 83°$), blur (0.75px) and noise (5%). The annotations are translated into the Pascal VOC format for the following object detection pipeline.

5.3.4 Yolo v4

For the detection of the work pieces involved in the assembly tasks, a Yolo v4 network implementation based on Darknet is used. Darknet is an open source framework for neural networks, which is written C and CUDA [60]. It takes the recorded RGB images as input, as well as the bounding box marker in Pascal VOC format for the training process. During inference for each frame Darknet outputs bounding boxes as 2D center coordinates and width and height values, as well as a probability of occurrence for each working piece. The architecture of the embedded network is based on Yolo v4 and is a deep-learning approach for object detection and making predictions of bounding boxes and class probabilities labels with real time capability [6]. Object detection can be divided into two sub tasks - detecting possible object regions and classifying the image in those regions into object classes.

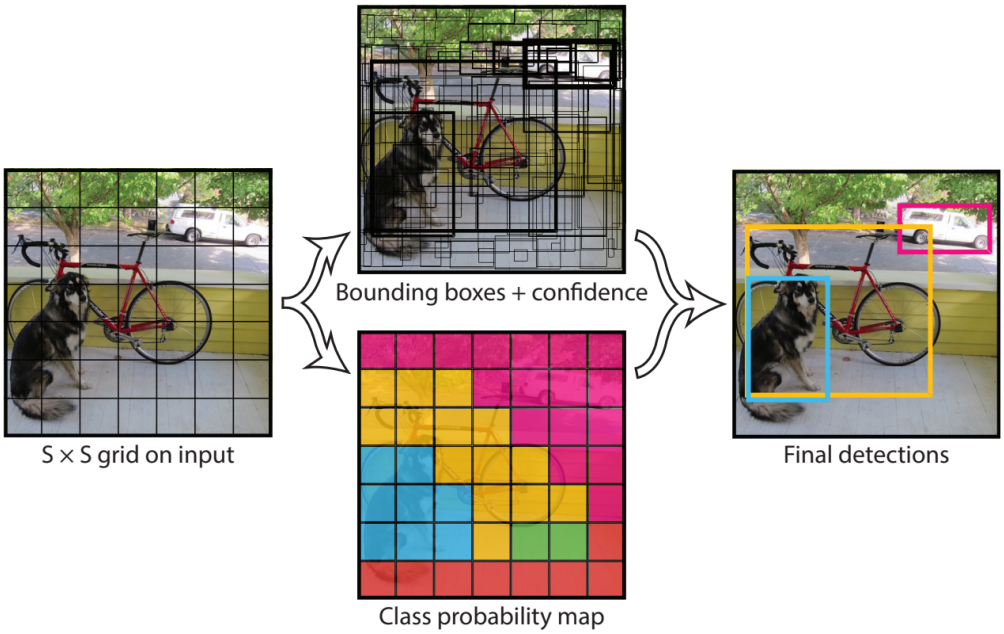


Figure 5.11: Yolo v4 pipeline from input grid to calculating the final detections [61]

The core concept behind Yolo v4 is shown in illustration 5.11. First an image is split into several cells using a $S \times S$ grid. Each cell is able to predict up to several bounding boxes, along with the object label and probability of the object being present in the cell. The combination of bounding box candidates and class probability map leads to one bounding box with the highest confidence per object as output. This approach lowers the computation cost, because detection and recognition are handled by cells from the image. Furthermore Yolo v4 is a one stage-detector, which is depicted in image 5.12. Two-stage detectors decouple the task of object localization and classification for each bounding box, while one-stage detectors are making the predictions for object localization and classification at the same time. Both architectures are described in detail in section 4.5.

The architecture of the one stage detection part consists of input, backbone, neck and the head. The backbone is for feature extraction, which is typically provided by pre-trained weights. In this case a CSPDarknet53 architecture is used as backbone, which is trained on ImageNet [67]. In the neck the combination of different feature maps of different stages of the backbone happens, which leads to feature aggregation. SPP (Spatial pyramid pooling) and PAN

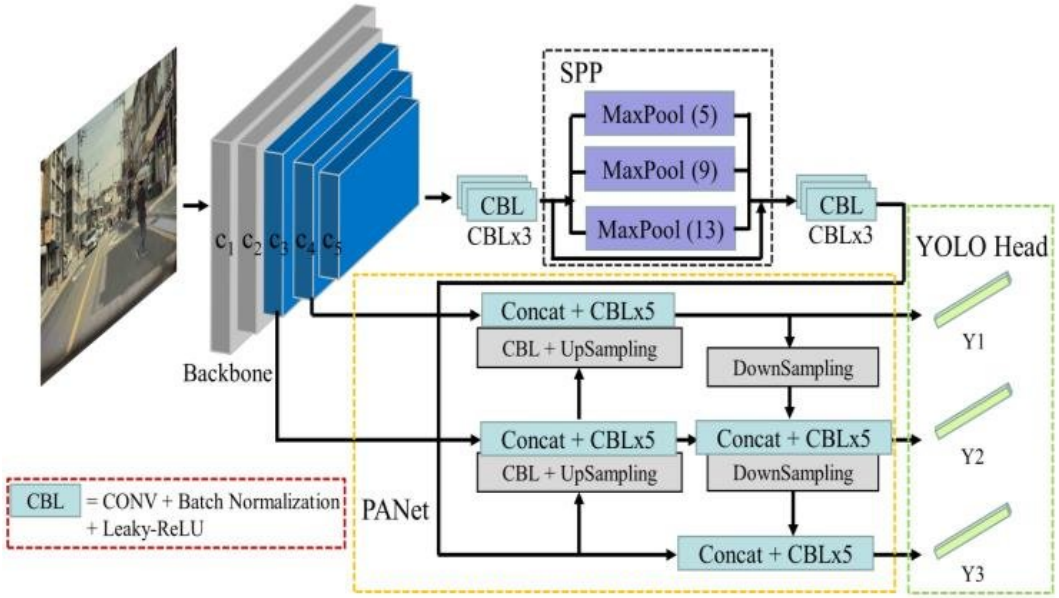


Figure 5.12: Overview of used Yolo v4 one stage detector architecture [49]

(Path Aggregation Network) are used in this instance. The SPP layer will allow to generate fixed size features whatever the size of our feature maps. In order to generate this, it uses pooling layers like max pooling, which is applied to sliding kernel kernels with different sizes. The hereby created feature maps are concatenated as output, which is a fixed in length, regardless of the input size.

PaNet has introduced an architecture that allows better propagation of layer information from bottom to top or top to bottom. Its purpose is to preserve spatial information. To this purpose the current layer and information from a previous layer is added together to form a new vector. In the YoloV4 implementation, a modified version is used where the new vector is created by concatenating the input and the vector from a previous layer.

The head is responsible for the dense prediction and is based on YoloV3. It detects the bounding box coordinates (x,y,w,h) with a corresponding confidence score for a class. The included network architecture is the Darknet-53 and is composed of 53 convolutional and residual layers, with average pooling, fully connected, and softmax for the last three layers.

In order to improve the performance of YoloV4, bag of freebies (BoF) for increasing the generalization ability of the and bag of specials (BoS) for increasing detection accuracy are introduced. BoF is a data augmentation technique,

which includes image manipulation like rotating, blurring, changing contrast, random erasing etc. BoS on the other hand are including attention mechanisms, increment of the receptive field and post-processing like non-maximum elimination.

5.4 Classification Process

In this section the classification process of the actions involved in the assembly tasks are outlined. After the feature selection, four models of different machine learning models are trained on the extracted data. First the involved frameworks and software Keras and the sliding windowing function are described and later on the architecture of the implemented machine learning models. The used models are a feed forward neural network, a single stream LSTM a three stream LSTM and a temporal sliding LSTM. All networks are trained supervised, which requires labels. Therefore the features of each frame are labelled by hand to the corresponding action class which is executed by the operator in the current frame. Depending on the task their are either 58 or 59 features present for each frame, 18 for the 2D joint coordinates, 18 for their corresponding depths values and four or five object probabilities respectively. Each model type uses the same baseline architecture for both tasks, for comparability of the performance evaluation later on. Only the hyper parameter are adjusted depending on the assembly task, which are presented for the training in the Experiments chapter 6.

Keras

All models are implemented with the Keras framework, which is an open-source library for implementing artificial neural networks with a Python API, which is acting as an interface for the general machine learning libraries Theano or TensorFlow [70]. The main feature are the neural-network building blocks. Those include typical attributes known from neural networks like activation functions, layers, neurons, objectives and optimizers. Furthermore different kind of neural networks are support like CNN, RNN and LSTM, as well as regularization methods like dropout and pooling layers and batch normalization, which makes the models highly customizable. Additionally it is possible to generate models in two different ways - sequential pattern and functional

API. In case of the sequential pattern the model is created by adding layers sequentially. This only requires to specify the input shape in the first layer and the parameters for each layer. The functional API on the other hand is used to created more complex model that can have an inception module or multiple inputs and outputs.

Sliding Windows

The LSTM based models are constructed with the help of the predefined sequential model by Keras [70]. For each Layer of the neural network Keras provides different architectures, which have a range of tuneable parameters. In this case LSTM cells are used as neurons. The LSTM layers expecting a time sequence as input or more precisely a tensor with three dimensions. According to the data set, two dimensions are already given by the sample size and the number of features. The third dimension is created by a sliding window approach, which will model the features of the data set into a time sequence. The samples are copied into windows of a chosen size and a chosen step width, which both are two hyper parameter. An illustration of the changed input sized can be see in image 5.13, where the window size is chosen as three and the step size is one.
And thus leads to an input tensor in the shape of:

$$\textit{number of samples} \times \textit{number of windows} \times \textit{number of features}$$

In algorithm 1 the pseudo-code for transforming the input with the sliding window approach is shown. As input the features like 2D-joint coordinates, depth coordinates and object probabilities per frame are expected. First a window size for a time period is chosen. The size is dependent on the amount of frames per action. However different actions vary in execution duration. For instance handovers are executed a lot faster compared to the installation of certain tools. Therefore a small window size may result in actions being broken up in several windows and reducing information about temporal dependencies over the whole activity. If the window size is to big, several actions may be encapsulated in one window which reduces information about shot-term dependencies. Furthermore a step size has to be set. The step size indicates the number of frames the windows is moved. In this case one was chosen with each step counting for each frame, with an overlap depending on the

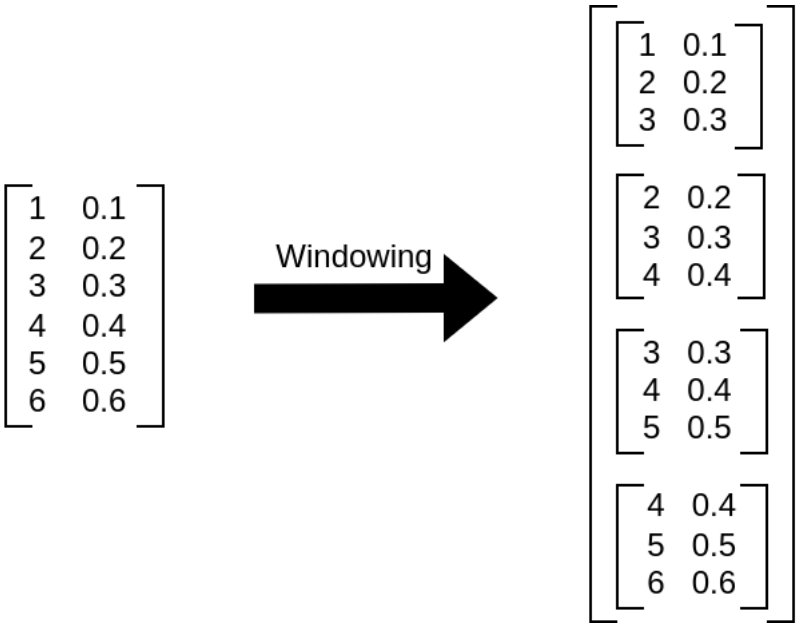


Figure 5.13: Example of windowed input tensor

total amount of frames. A bigger step size is resulting in more coarse window overlapping and thus in less precision. After processing the data into the tensor input format, they are plugged into the LSTM based models.

5.4.1 Feed Forward Neural Network

The feed forward neural network (FNN) allows a classification without focus on the temporal dependencies, while relying more on spatial attributes of both assembly setups. The architecture of the neural network model can be seen in illustration 5.14. On top the input of the feed forward network is represented by a sequence of the concatenation of 2D and depth coordinates, as well as object probabilities, which are extracted of each video frame. The setup of the model consists of three dense layers, with the first one being the input layer. The input layer consists of neurons equal to the amount of features belonging to the data set. As activation function the Rectified Linear Unit (ReLU) is chosen. The second layer consists of neurons equal to half of the features in order to slowly filter the important features, as well as a ReLU activation function. The output layer has as number of neurons the amount of classes that are required to be recognised. In order to gain a probability vector as output the output layer is implemented with a softmax activation function.

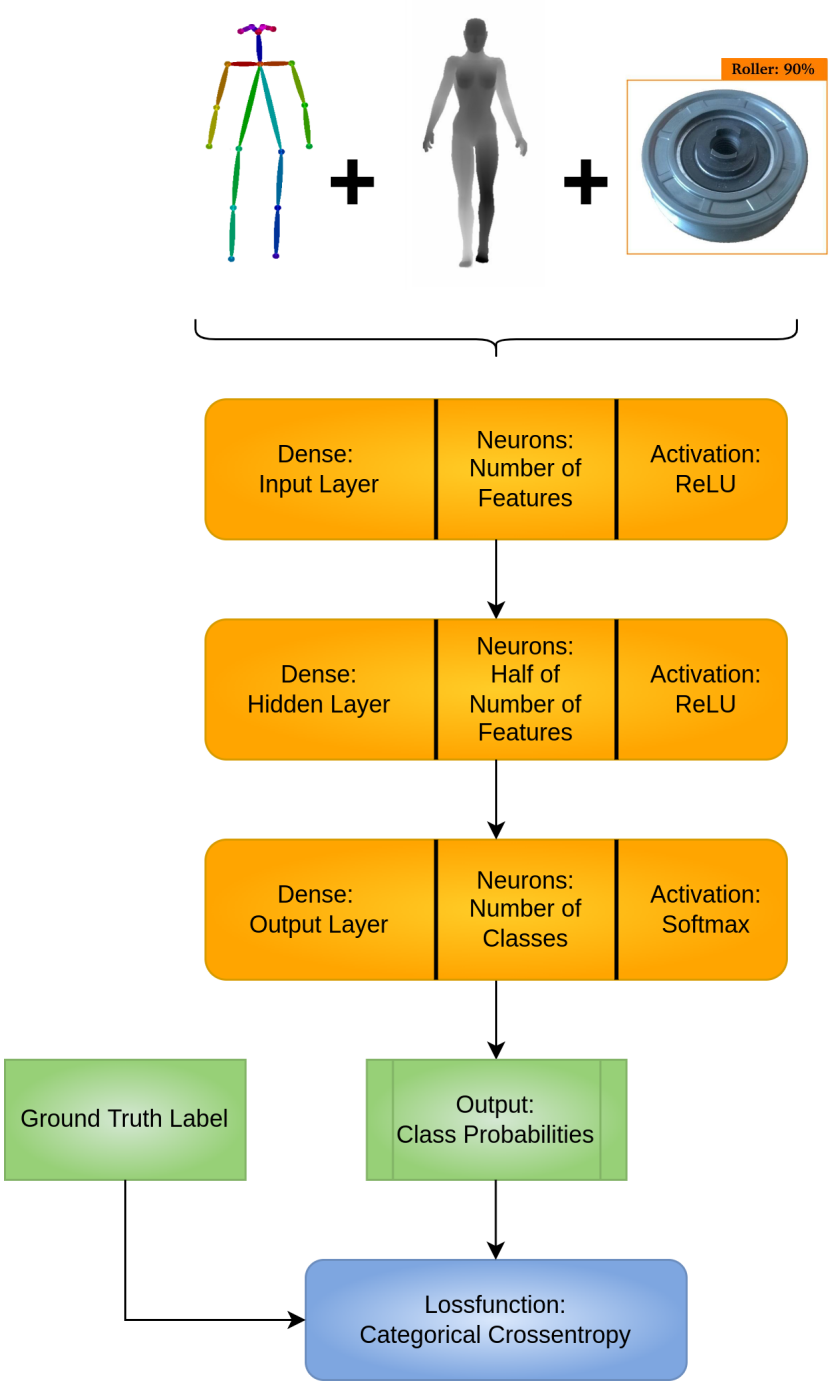


Figure 5.14: Architecture of the feed forward neural network model

```
Data: Features
Result: Windowed Data for LSTM
initialize windowsize;
initialize stepsize;
initialize tensor_output_list;
for i < len(features) - windowsize - 1 do
    Gather past records upto the lookback period initialize
     templist;
    for i < len(features) - windowsize - 1 do
        templist.append(features[i + j + stepsize])
    end
    tensor_output_list.append(templist);
end
Return tensor_output_list;
```

Algorithm 1: Windowing of Features

For the back propagation step the categorical cross-entropy is chosen as loss function, it is a classification approach for multiple action and the predictions are compared to the ground truth label.

Pre-processing the data

For the feed forward neural network the data has to be shaped in two dimensional tensor in order to fit into the input layer. The features per frame are concatenated for each recorded scenario, as a two dimensional tensor with the shape of 58 or 59 features per frames respectively, which are stored locally as numpy array. Since temporal dependencies aren't focused with the first model, no sliding window algorithm has to be applied. The labels are stored in a numpy array as well, however it is a one dimensional vector with one label per frame. Since human action recognition is a classification of categorical variables, one hot encoding is used with the Keras function *to_categorical* to represent each class as a binary vector.

5.4.2 Single Stream LSTM

Single stream LSTM (1S-LSTM) are able to capture temporal dependencies, additional to spatial dependencies. The design of the 1S-LSTM is presented in image 5.15 and consists of a LSTM layer, a dropout layer and a dense layer.

It is build with the sequential model of Keras. The input of the 1S-LSTM is represented by a sequence of concatenated 2D coordinates, depth value and object probabilities extracted of each video frame. However the involved LSTM is the input layer and requires the data to be shaped as three dimensional tensor as input and therefore the first step is a pre-processing including the windowing function. The details of the pre-processing are explained in more detail further below in 5.4.2. Th The number of neurons for the LSTM input layer is equal to the number of features belonging to the data set. Since each LSTM layer requires are three dimensional tensor as input. As activation function is the hyperbolic tangent (tanh) chosen, since its second derivative can sustain for a long range before going to zero during back propagation. The dropout layer prevents LSTM based model to overfit, while randomly setting the input units to zero. The percentage of units that are set to zero is another hyper parameter. After the dropout layer, a dense layer is applied. The number of actions to be classified specifies the number of neurons for this layer. Also a softmax activation function is applied in order for the output to be a vector with class probabilities.

Pre-processing the data

For the 1S-LSTM the data has to be shaped in three dimensional tensor in order to fit into the LSTM input layer. The features per frame are concatenated for each recorded scenario, as a two dimensional tensor with the shape of the amount of features per frames, depending on the recorded task. Those are stored locally as numpy array. Since temporal dependencies are focused with this model, the sliding window algorithm has to be applied to the input data. The labels are stored in a numpy array as well, which is vector with one label per window. Since each window contains several frames, the corresponding label is calculated by the class label which occurs most frequently during the for all the frames encapsulated by the window. Since Human Action Recognition is a classification of categorical variables, one hot encoding is used with the Keras function *to_categorical* to represent each class as a binary vector.

5.4.3 Three Stream LSTM

The three stream LSTM (3S-LSTM) is able to capture temporal and spatial dependencies for classification purposes. However the additional purpose of

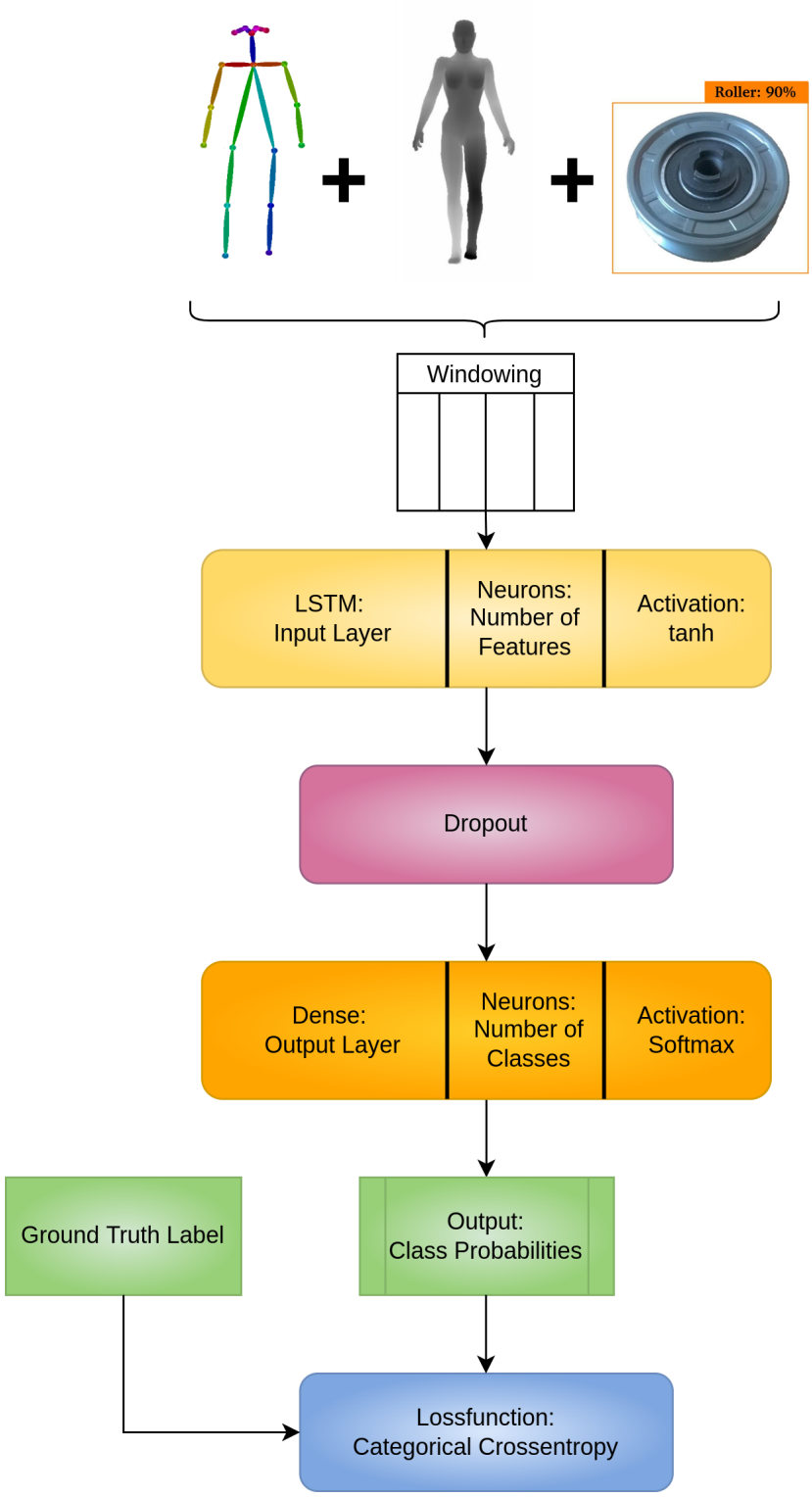


Figure 5.15: Architecture of the single stream LSTM model

this network lies in allowing underrepresented features to be more influential for the classification performance, e.g the tracked skeleton provides 36 features per frame, the detect objects just four for the control cabinet door assembly task. The idea and network architecture are loosely based on the work of Yang et al. [50], who are also dividing the training of different feature modalities to three separate models and concatenating the outputs of those models for further processing. The design of the 3S-LSTM is presented in image 5.16. It is build with the functional API of Keras. It consists of three separate 1S-LSTM models, which are setup similar to the 1S-LSTM model from section 5.4.2. However their outputs are concatenated before being handed to a additional dense layer. Each 1S-LSTM model is build identical, but the input features differ from each other which results in a different amount of neurons for each input layer. In this case is trained on the 2D joint coordinates and gets 36 input features. The second is trained on the depth values and gets 18 input features. The third is trained on the object probabilities and the number of features depends on the number of objects tracked in a given assembly task. After each 1S-LSTM outputs a class probability vector, those are concatenated with an Add layer, provided by Keras. The concatenated probability vector is then processed by another dense layer, providing the final class probabilities.

Pre-processing the data

The pre-processing of the data for the 3S-LSTM network, is similar to the 1S-LSTM network which is described in section 5.4.1. However dimensions of the windowed input tensors differ for each of the three involved 1S-LSTM networks and are depending on the amount of input features. The window size is a hyper parameter, but are equal for all three 1S-LSTM networks, since the window size depends on the amount of frames corresponding to an action and not the amount of input features per frame.

5.4.4 Temporal Segment LSTM

The temporal segment LSTM (TS-LSTM) is used for classification, while being able to leverage short and long term temporal dependencies. The architecture of the TS-LSTM is presented in image 5.17. In this scenario four 1S-LSTM networks with different window sizes for the input data are the building blocks for the TS-LSTM, however all features are concatenated and being used as in-

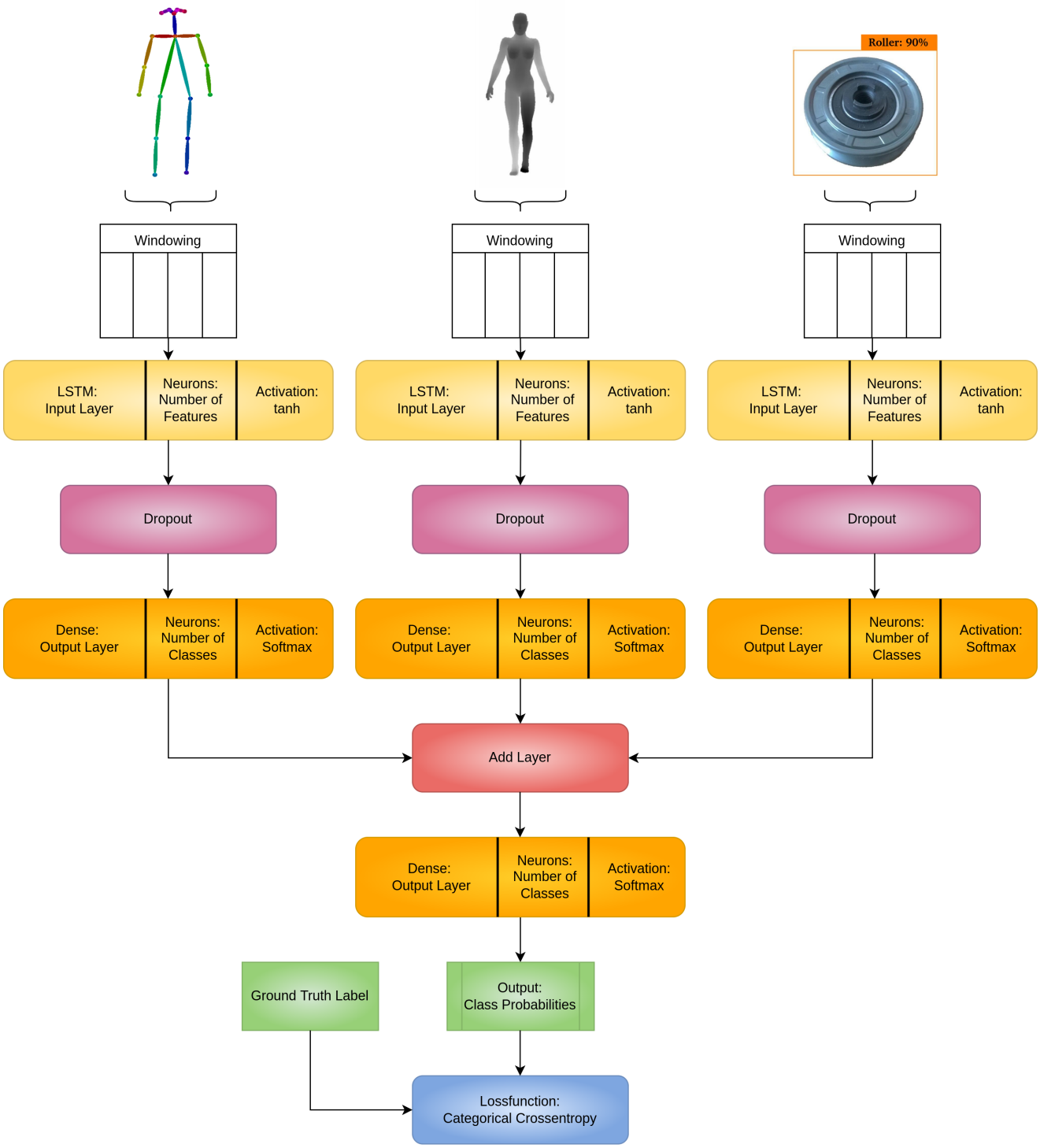


Figure 5.16: Architecture of the three stream LSTM model

put. The different window sizes allow to exploit the duration for the execution time of different actions, as well as the short and long term dependencies in a sequence of actions. The idea and network structure are based on the work of Chen et al. [47], who are training an ensemble of four different LSTM based models, with their input data consisting of different window length It is build with the functional API of Keras. It consists of an ensemble of four 1S-LSTM models, which are build similar to the 1S-LSTM presented in section 5.4.2. The main difference is the shape of the input data. While all features are getting concatenated to one feature vector for each input layer, the dimension of the input tensor changes with the window size for each input layer. The number of neurons for each layer is equal to the amount of features depending on the data set. Finally the predictions of each 1S-LSTM are averaged as final result.

Pre-processing the data

For the TS-LSTM the data has to be shaped in three dimensional tensor in order to fit into each of the 1S-LSTM input layers. Since temporal dependencies are focused with this model, the sliding window algorithm has to be applied to the input data. In this case each model gets assigned its own window size, starting small and increasing a lot. However, since the windows are overlapping, bigger window size are leading to an input tensor with more entries compared to tensors build by smaller window sizes. In order to avoid conflicts for taking the average for the outputs of each model, zero padding is applied in order to achieve the same amount of entries for each input tensor.

5.5 Evaluation Methods

This section highlights the used methods for evaluating the results. Since the human action recognition framework is based on the classification of multiple actions, a metric for evaluating the quality of the classification is needed. For this purpose the confusion matrix is used. Furthermore the feature importance is analyzed, since it allows to draw conclusions about on attributes to focus for feature extraction for identifying human actions. Finally a multi class receiver operating characteristic (ROC) curve is implemented, in order to analyse which actions are easier for the models to classify than others.

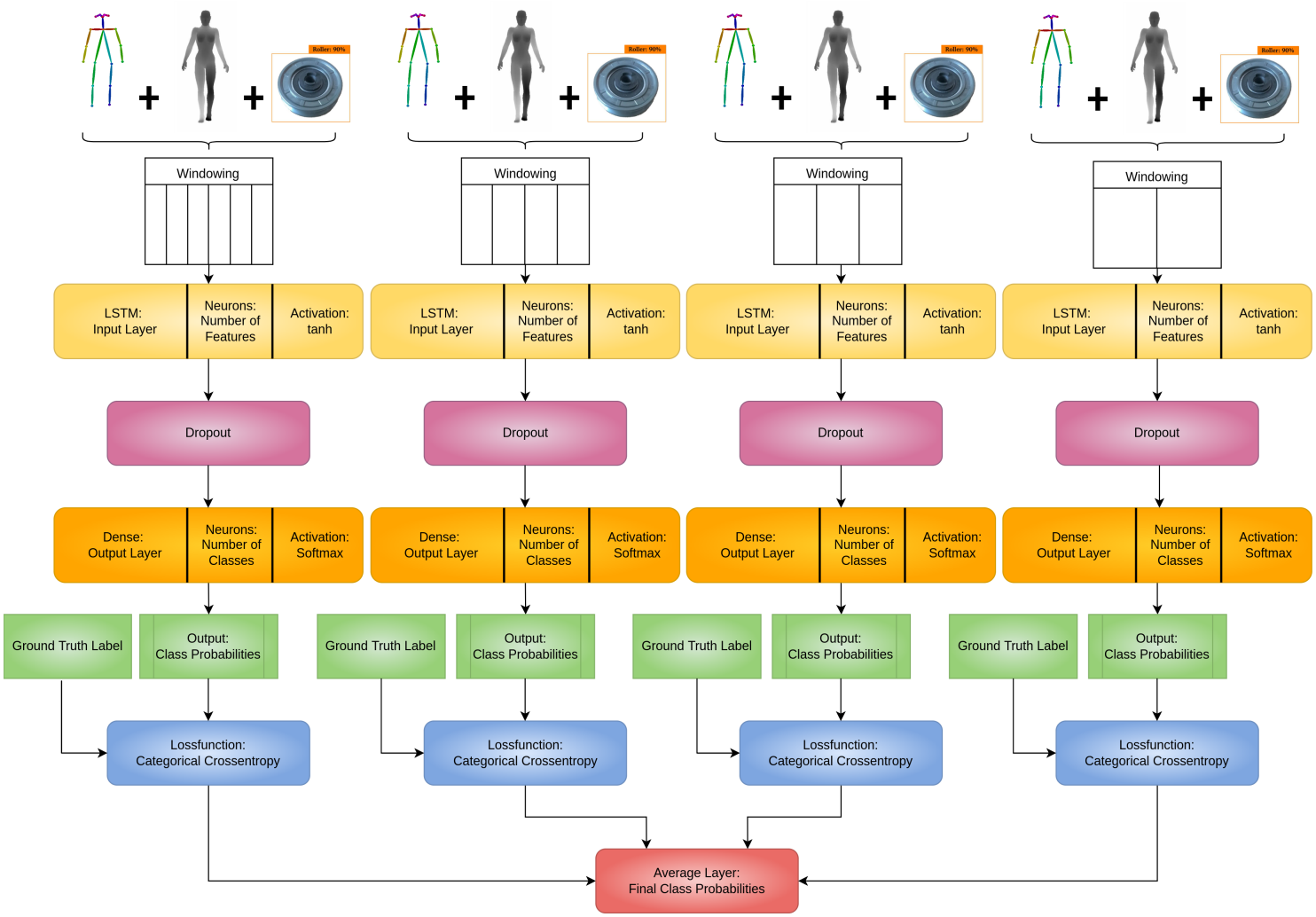


Figure 5.17: Architecture of temporal segment LSTM model

5.5.1 Confusion Matrix

The proposed models are multi-class classification models. Therefore a confusion matrix is one tool used to evaluate the results and to validate the architecture. The four deep learning classifiers are trained supervised, and therefore class labels are applied for for each frame by hand. Therefore it is possible to compare the predicted actions of the assembly tasks against the actual actions. In figure 5.18 the confusion matrix is depicted. It shows how the predicted and actual classifications are related to each other. If prediction and actual label are matching, it is called a true positive (TP). The true negative (TN) describes, if a not predicted class is not matched to the actual label. The false positive (FP) indicates a positive prediction for the wrong class label. Two other metrics are the true positive rate (TPR) and the false positive rate (FPR). The TPR indicates which amount of all predicted classes got correctly classified by the classifier, while the FPR which amount of all predicted classes got wrongly classified. Since the prediction performance is equally important for all classes, the accuracy is calculated as sum of TP and FP and set in ration to the total amount of predictions.

5.5.2 Multiclass Receiver operating characteristic

Another evaluation tool is the receiver operating characteristic or ROC curve. It gives a measurement for the classification quality, depending on the classification thresholds. Usually the ROC curve is used for binary classification, which is indicated in image 5.19a. The true positive rate (TPR) is plotted against the false positive rate (FPR). The Ideally a TPR of one and a FPR of zero is desired. Since this outcome is very unlikely for real applications, a steeper curve is aimed for with maximizing the area under the curve (AUC) and therefore finding a threshold that increases the TPR, while minimizing the FPR. The bisector indicates random guessing.
However the HAR framework analyses multi class problems, and therefore the ROC curve has to be modified to a multiclass ROC curve. An example of a multiclass ROC curve can be seen in figure 5.19b. In this case for each class one curve is plotted against the others resulting in a binary representation. As the trained models are making predictions of the class probability for each frame, the threshold is the variation of those probabilities given a specific class. This allows to visualize about how well each of the different classes are predicted

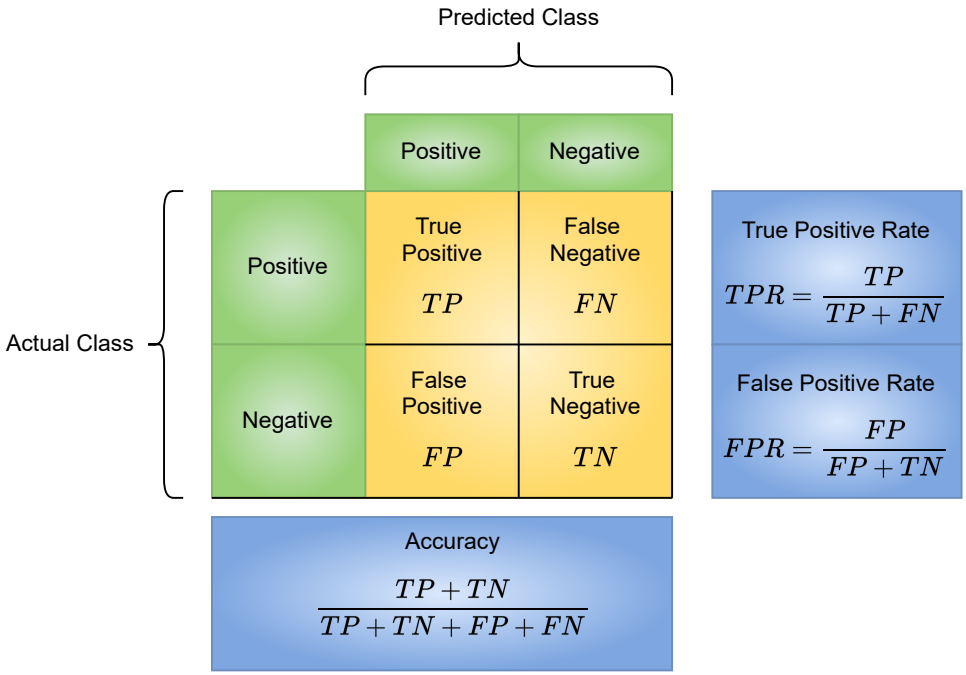
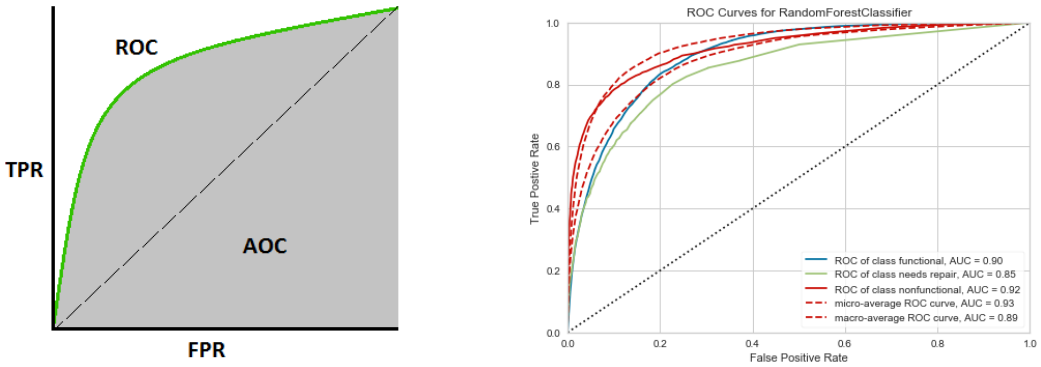


Figure 5.18: Confusion Matrix

and which classes are difficult for the models to detect. To this purpose the Sklearn metrics library [58] for the calculation of the ROC curve is used, as well as the pre-processing library label binarizer. This is important because for multi-label classification the output has to be binarized. Furthermore a macro-average ROC curve and a micro-average ROC curve can be calculated. The macro average is the average of all present ROC curves, which means it doesn't take class imbalance into account and is heavily influenced by the largest class. However since some actions are occurring in way less frames then other, the micro average as is the weighted average displaying the influence of smaller classes with more precision [45].

5.5.3 Heat Maps

In this thesis several features are used for the HAR framework. Those are the two coordinates of the human joints with a corresponding depth value and the work pieces which are used in the experiments. While the number of tools ranges from four to five, 58 or 59 features respectively per frame are provided.



(a) Binary class ROC curve (b) Examples for multi class ROC curve

Figure 5.19: ROC curves for binary and multi class evaluation

Since the focus of the analysis for the action recognition lies in the spatial and temporal properties of the features, it is beneficial to evaluate the importance of each feature during the recordings. For this purpose heat maps are presented as tool in order to evaluate the importance of each feature. To this purpose the ELI5 (Explain like I'm 5) and its permutation importance library are used, which is applicable to neural network architectures [41]. It works for global interpretation over all recorded frames corresponding to a certain assembly task. This library allows to train each of the presented models (FNN, 1S-LSTM, 3S-LSTM and TS-LSTM) several times, while leaving out one feature at a time. It outputs a list with the average weight of every feature and its standard deviation sorted by the amplitude of the most important feature. Subsequent to this step, the values of the list are used for the visualisation of the feature importance. Depending on the weight list for an assembly task, a recorded frame with a fully assembled work plate is chosen as representative image. The corresponding Openpose skeleton is applied on top of the image, as well as the bounding boxes provided by Darknet. Now a Multivariate-Gaussian distribution is used as visual aid for interpreting the feature importance. As seen in equation (5.1) the vector $\mathbf{f}$ indicates the features, μ the mean and Σ the covariance matrix.

$$p(\mathbf{f}|\mu, \Sigma) = \frac{1}{\sqrt{(2\pi)^d |\Sigma|}} e^{-\frac{1}{2}(\mathbf{f}-\mu)^T \Sigma^{-1} (\mathbf{f}-\mu)} \tag{5.1}$$

The location of the gaussian distribution is defined by the amplitudes of the x,

y and z value of a joint and the entries of covariance matrix is given by their corresponding standard deviations. The resulting kernels are placed on the respective joint positions in top of an image. Similar is the gaussian distribution applied to the probabilities of the detected objects, which are equally placed in the scene. Each feature is marked by a red circle with, indicating a greater feature importance for more prominent circle radii.

6 Experiments

In this chapter the practical experiments are described and results presented. The purpose is to test and validate the proposed HAR-framework with two different assembly tasks. In this context every aspect of the pipeline is analyzed and the experiments will be described for each task individually. The details of the implementation are outlined in depth in chapter 5. However specific parameters and modalities for the experiments in regards to the data sets and models are part of this chapter. First the recording of each data set is presented. After that, the data sets are applied to a feed forward Neural Network, single stream LSTM (1S-LSTM), three stream LSTM (3S-LSTM) and temporal segment LSTM (TS-LSTM) in order to train them for multi-classification of the actions. To this purpose, the hyper parameters used for training are presented, as well as the performance of the training process. The trained models are then evaluated based on their classification quality. This includes the evaluation of the metrics presented in 5.5, like confusion matrix, heat map and multiclass receiver operating characteristic curve (ROC). Also an ablation study is done by selecting different feature categories as training input. Finally the results of the experiments of each task are compared towards each other, allowing conclusions about the classification effort and faced obstacles for the presented actions.

6.1 Dataset: Elevator Cabin Hanger

The data set of the elevator cabin hanger features the assembly task setup described in section 5.3 and will be additionally referred to as *ECH task* in the following sections. This data set consists of RGB-D video recordings of the elevator cabin hanger assembly task, which are recorded with the Kinect v2 with a frame rate of 13 fps. It features the recordings of two participants, one female and one male, which are performing the assembly task each ten times. This results in 19894 recorded frames over 20 recordings. Image 6.1 shows

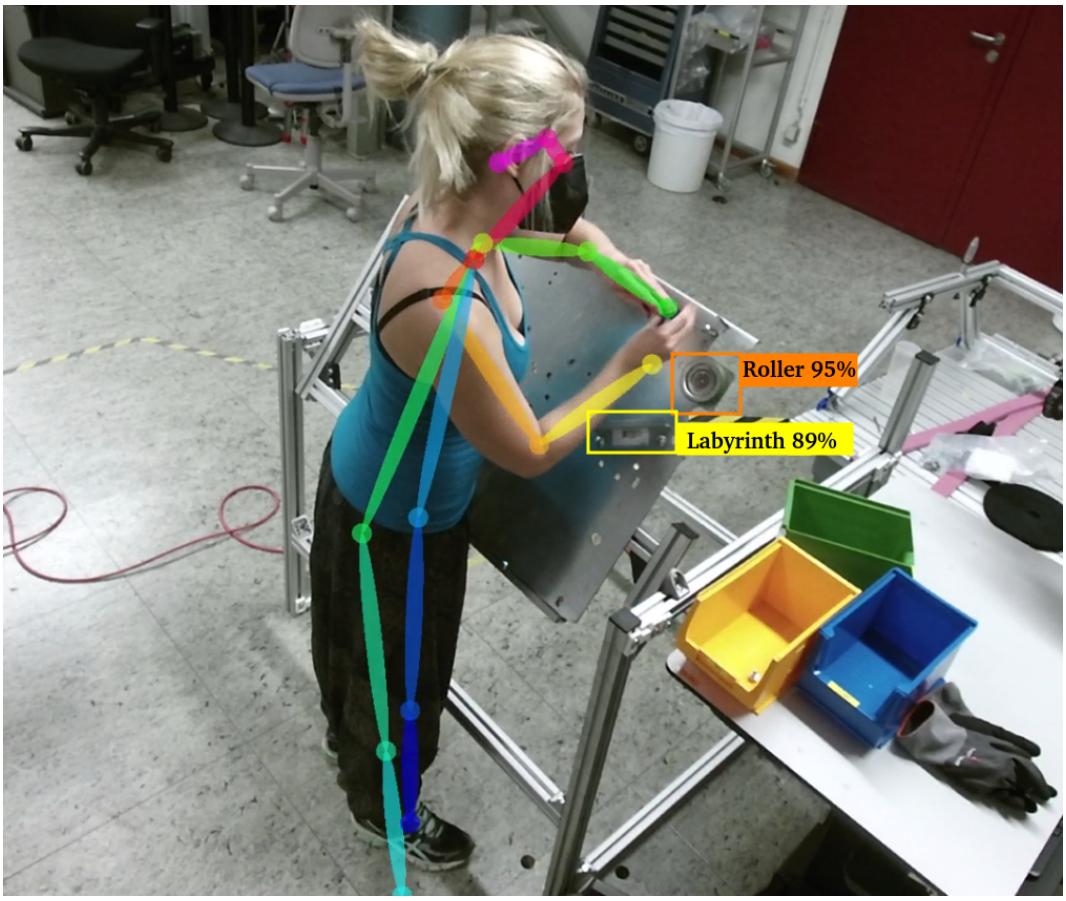


Figure 6.1: Skeleton and object tracking of the operator and object detection during the execution of the elevator hangar assembly task

the real time application of the skeleton tracking and object detection, while the operator is executing the assembly task. For the first assembly task five work pieces, as well as the skeleton of the operator and their depth values are tracked. The threshold for accepting the detection of the work pieces in the videos is set to 85%, since some work pieces like the shunt bracket are small and due to the reflecting, metallic surfaces of all involved work pieces, they are sometimes hard to detect for the object detection. The data set is labelled, with each frame getting assigned one action involved in the assembly task as label, which are described in detail in 5.3. There are overall eight classes - the starting position, handing over of work pieces, screwing on the two rollers, shunt bracket attachment, bumper bracket attachment, the installation of the anti-opening device and the final position after task completion. Furthermore the executed actions will be referred by a number corresponding to their

chronological appearance instead by a description for convenience. Table 6.1 is showing the translation between action explanation and their numeration, e.g Action 2 is corresponding to the hand over action.

Action 1	Starting position in front of empty hangar plate
Action 2	Handing over work pieces
Action 3	Screwing on of both rollers
Action 4	Screwing on of hangar labyrinth
Action 5	Screwing on of shunt bracket
Action 6	Screwing on of bumper bracket
Action 7	Installation of Anti-Opening Device
Action 8	Final position in front of assembled hangar plate

Table 6.1: Enumeration of actions and their corresponding execution steps for the ECH task

This means that overall each frame has 59 features in total for the first assembly task. Each of the used models trained for this setup, are using 15332 frames for training, 3225 frames for validation and 1347 unseen frames for testing purposes. Since the temporal dependencies are important, the frames aren't chosen randomly for the validation and test set. Instead the frames of the validation and test set are belonging to two and one whole recordings respectively, in order to get access to all stages of the assembly task. Image 6.2 shows for each involved action how many frames are labeled as this action class. In red is the mean $A = \frac{1}{n}\sum_{i=1}^{n} a_i$ of the frames involved in the training process depicted, as well as the standard deviation $SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$ as error bar to show amount of variability. It is observable that some actions like the starting position are featured in fewer frames compared to the attachment of the roller. In case of the handover the total amount of frames during the task is shown, however this is the only action that happens more than once with four occurrences in total. This means the action itself involves just one fourth of the depicted frame number. The same holds true for the distribution of frames involved in the model performance testing, which is depicted in blue. Overall the test set involves a little less frames per action than the mean over the training set. This is due to the recording of a full assembly activity is

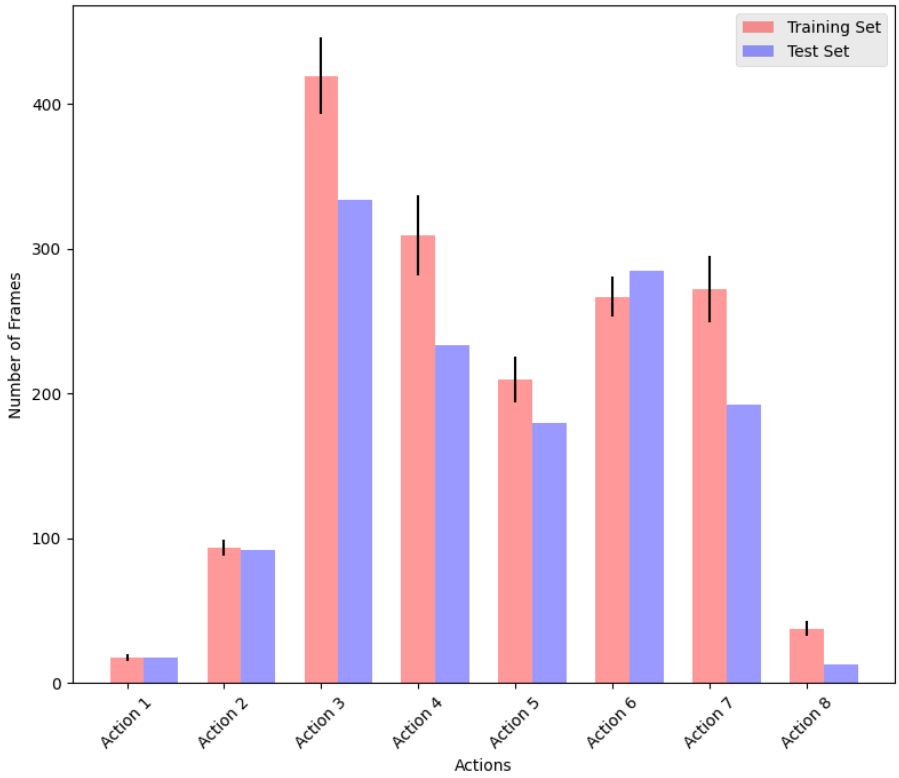
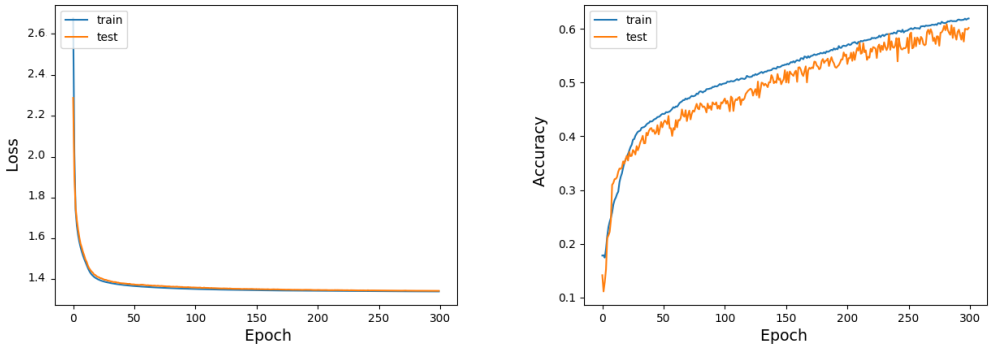


Figure 6.2: Frame distribution for each action of ECH task

chosen randomly out of all recordings as unseen test set, and the execution time per action is varying for each assembly round for each operator.
In the following the experiments for each model are presented. First the training schedule and hyper parameters are described. Secondly the results are shown and analyzed with the classification probability distribution, the confusion matrix and the multiclass receiver operating characteristics (ROC) curve.

6.1.1 Forwarad Neural Network

The first experiment is conducted with a FNN, without taking the temporal component into account. Instead the focus lies on spatial information. The model architecture consists of three dense layers. The dimension of the input tensor is 15332×59 and the output is a vector of probabilities for the eight classes for each frame. The architecture is lined out in detail in section 5.4.1.



(a) Training loss (b) Training accuracy

Figure 6.3: Training for FNN for ECH task

For the training process the data set is shuffled, since the network doesn't take temporal dependencies into consideration.

6.1.1.1 Training

The hyperparameters of the training schedule can be observed in table 6.2. A batch size of 32, the ADAM optimizer and a learning rate of $1 \cdot 10^{-3}$ are chosen via grid search.

Batch Size	Learning Rate	Optimizer	Epochs
32	$1 \cdot 10^{-3}$	ADAM	300

Table 6.2: Hyperparameter for the training of FNN for ECH task

The network is trained for 300 epochs, since further epochs are leading to over fitting of the model. The progress of the model accuracy and model loss can be seen in image 6.3a and image 6.3b respectively. The training loss, as well as the validation loss are dropping very fast and are approximating to zero after the first 30 epochs. The final training loss reaches a value of 1.34 with the validation loss being higher around 1.37. The training accuracy of the model reaches a value over 45% after 50 epochs and converges to an accuracy of 66.86% at the end of the training. The validation loss follows closely, with a final accuracy of 65.95%.

6.1.1.2 Results

Testing the model on unseen data is yielding the following results. Out of 1347 test frames, 885 are classified as true positive values. This is a ratio of 65.70% classification accuracy. Image 6.4 shows a visualization of class probabilities per frame in comparison to the actual label, which is presented as black line. The corresponding colors for each of the eight classes can be seen in the legend. The first Action is depicted in blue and marks the starting position. It can be seen that there is not a huge overlap between the actual label and prediction of Action 1. This is further underlined by the confusion matrix presented in image 6.5. It shows the ratio between actual label and prediction of each class. For Action 1 the TPR is 27.78% and a huge chunk of the FPR is predicted as Action 3 and Action 4 and no other false positives for the other classes.
Action 2, presented in orange, corresponds to the handover action. It can be observed, that this action has fewer frame occurrences with around 20 frames per action and is the only action where the frames aren't all connected in one sequence. However 68.48% of all predictions for Action 2 are true, even with this class imbalance. Around 5-7% of the frames belonging to Action 2 are predicted as classes. This can be explained with the evolving transition phase between the actions, making it harder to mark the end and beginning of an action. This is underlined by observing declining accuracy in the beginning and the end of a handover phase.

Action 3 is shown in green and represents the placement of the two roller. This is the state containing the most frames for this assembly task and is predicted with a TPR of 82.04%. False predictions are made for the following three classes with around 3% to 6%. This action execution spans 388 frames and it is is observable in figure 6.4 that after 150 frames the probability for each frame raises from 55% to almost 100%. This is also the approximate frame number where the second roller is found by the object detection, raising the probability of this class even further.
In red is Action 4 depicted, which translates to placing the labyrinth hanger on to the hangar plate. This action has a TPR of 58.37% and has a huge FPR for Action 5. Since Action 4 and Action 5 are executed on two positions that are very close to each other, it is harder for the network to only rely on spatial information. Action 5 is depicted in violet with an accuracy of 30.00 %. This class is the hardest to predict, since 26.67% of all frames are classified as class

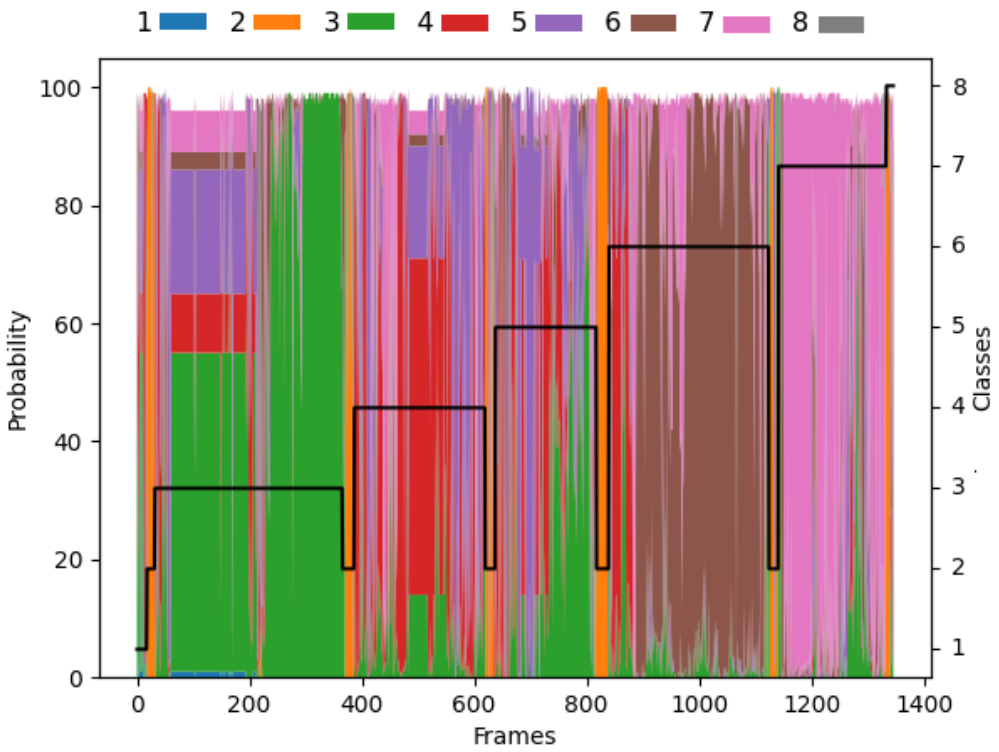


Figure 6.4: Accumulated class probabilities for FNN over frames for ECH task

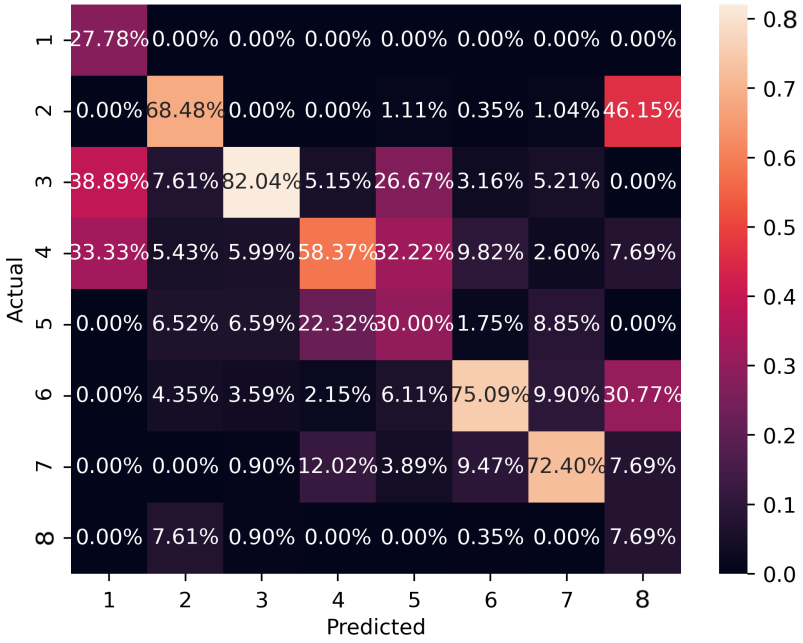


Figure 6.5: Confusion matrix of FNN for ECH task

two and 32.22% as class three. In comparison to Action 4, there is a lot of occlusion occurring through this assembly stage and it has the least amount of frames of the assembly states with just 168 frames. Action 6 and Action 7 on the other hand are among the most accurate classified with a classification accuracy of 75.09% and 72.40% respectively. Both of those classes have the operator work on different and unique locations on the hangar plate. Action 6 is taking place on the left upper corner, while Action 7 is taking place in the middle and both classes don't have to deal with much occlusion. Finally Action 8, which is the ending pose, is presented in grey. This action is very short with under ten frames and with 7.69% classification accuracy not able for the network to recognize.

The classification success differs among the classes. In illustration 6.6 the multi class Roc curve is featured, as well as its AUC value by analyzing the false positive rate against the true positive rate. All classes are reaching an AUC over 0.68 and six of those even over 0.84, which indicates that the FNN model is indeed capable of classifying the actions. However Action 1, Action 2 and Action 3 have values over 0.9, which indicates that those classes contain

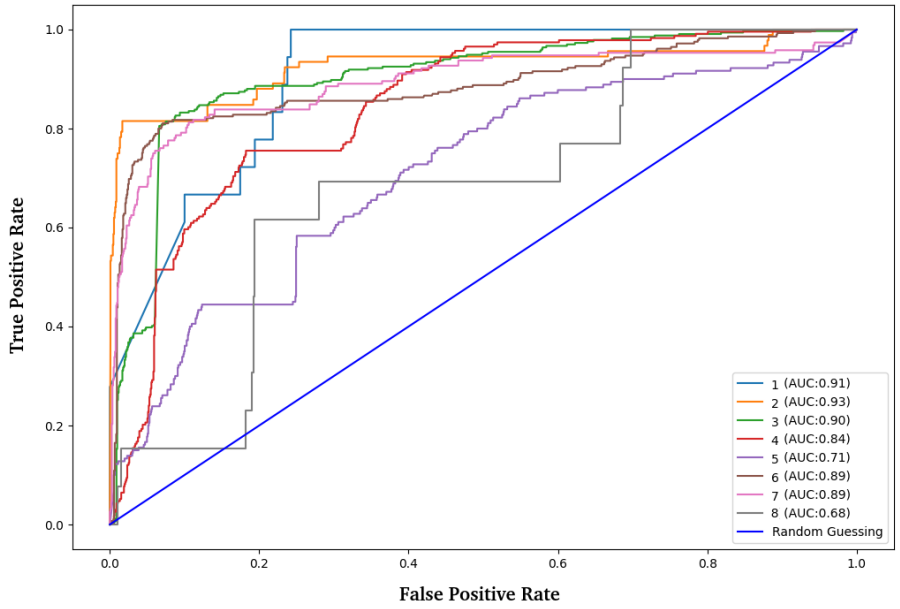


Figure 6.6: ROC curve of FNN for ECH task

some features that are making it easier for the classification than the other actions. Action 1 contains fewer frames, but the distinctive movement makes it easier to detect. Likewise the ROC curve for those three classes is starting very steep and reaching a true positive rate of 1 without having more than 0.1 of false positives. Action 5 on the other hand reaches a TPR over 0.9, with a FPR over 0.7 and has a AUC of 0.71. This indicates that Action 5 is harder for the model to clearly separate compared to the other classes, which is also indicated by the low accuracy success of 0.3. The hardest class to classify is class seven which is always predicted with a FPR offset, before predicting the correct labels. Since class seven is the shortest class with just less than 15 frames on average, and occurs just in the end, only the location of the operator and its posing is taken into account.

6.1.2 Single Stream LSTM

The second experiment is conducted with a 1S-LSTM network. Now the temporal components as well as the spatial components are involved for the classification. Since the input data needs to be three dimensional, the window size

determines the third dimension. Therefore the shape of the input tensor is $15332 \times 59 \times 3$ and the output is a vector of probabilities for the eight classes for each frame. The model architecture, as well as the pre-processing of the input data is lined out in detail in section 5.4.2.

6.1.2.1 Training

The hyperparameters for the training process can be observed in table 6.3. A batch size of 512, dropout of 0.15 and window size of 3 are chosen through grid search. The same applies to the chosen ADAM optimizer with a learning rate of $1 \cdot 10^{-3}$, $1 \cdot 10^{5}$ decay steps and a decay rate of 0.9.

Batch Size	**Learning Rate**	**Decay Steps**	**Decay Rate**
512	$1 \cdot 10^{-3}$	$1 \cdot 10^{5}$	0.9

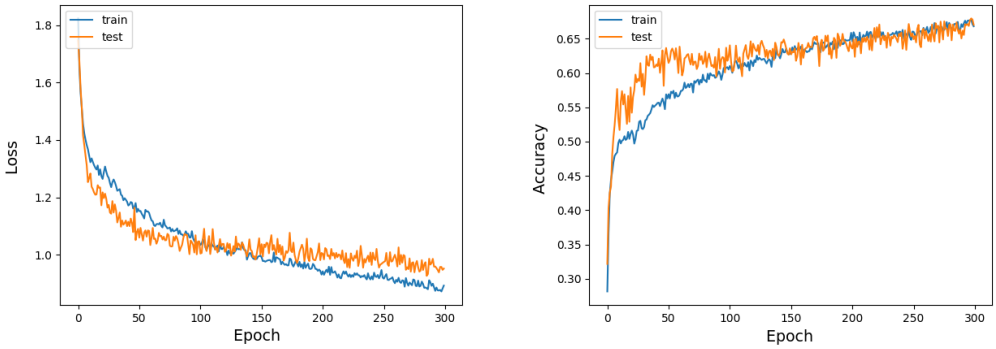
Optimizer	**Epochs**	**Dropout**	**Window Size**
ADAM	300	0.15	3

Table 6.3: Hyperparameters for LSTM Network and ECH task

The network is trained for 300 epochs, since further epochs are leading to over fitting of the model. The progress of the model accuracy and model loss can be seen in image 6.7a and image 6.7b respectively. The training loss, as well as the validation loss are taking some time to decrease with around 100 epochs in order to fall under 1. The training stops at epoch 300 in order to prevent over fitting since the validation loss converges around 0.91 while the training loss is still decreasing. The final training loss reaches a value of 0.85. The training accuracy of the model reaches a value over 60% after 150 epochs and converges to an accuracy of 71.89% at the end of the training. The validation accuracy converges around 71.55%.

6.1.2.2 Results

Testing the model on unseen data is yielding the following results. Out of 1347 test frames, 960 are classified as true positive values. This is a ratio of 71.26% classification accuracy. Image 6.8 shows a visualization of class probabilities per frame in comparison to the actual label, which is presented as black line.



(a) Training Loss (b) Training Accuracy

Figure 6.7: Training for 1S-LSTM for ECH task

The corresponding colors for each class can be seen in the legend. Action 1 is depicted in blue and marks the starting position, which reaches a classification accuracy of 33%, as depicted in the confusion matrix 6.9, which shows the ratio between the actual label and prediction of each class. A huge chunk of the FPR is predicted as Action 3 and Action 7 with 33.33% and 16.67 % respectively. Since the starting position contains on average less than 10 frames among all recordings compared to the other actions, it is harder to classify as it also features no present tools and just a few similar joint configurations with the person standing in front of the plate.
Action 2, presented in orange, corresponds to the handover action. It can be observed, that the handover action has fewer frame occurrences with around 20 frames per hand over and is the only action where the frames aren't all connected in one sequence. However 84.78% of all predictions for Action 2 are true, even with this class imbalance. The false positives are distributed among the other classes with 1%-5%. This can be explained with the distinctive movement of the handover, with reaching out an arm to the side of the scene. However it is also observable in image 6.8, that the false positives are occurring at the start or ending of a handover, showing the transition between actions is harder to classify. Action 3 reaches an accuracy of 84.73%. The class probability distribution shows similar behaviour like the trained Neural Network 6.1.1 with the accuracy shooting from 70% towards almost 100% after 200 frames, with the initial accuracy of the stage being higher by 25%. Action 4, which indicates the screwing on of the hanger labyrinth, reaches an accuracy value of 59.66%. A huge portion of the misclassification happens

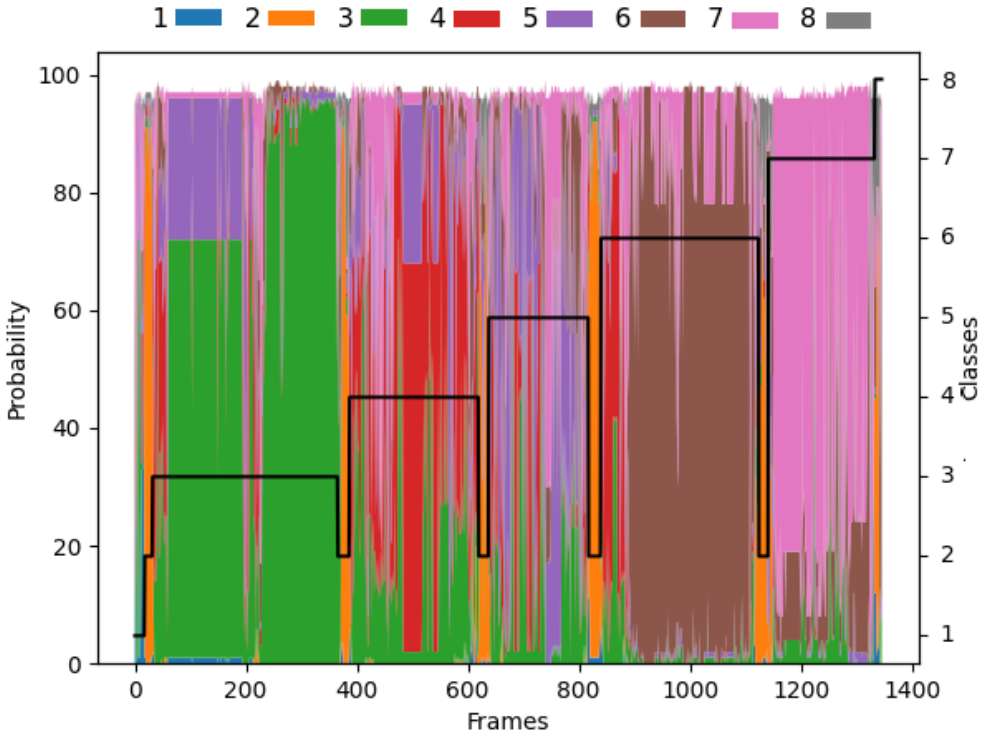


Figure 6.8: Accumulated class probabilities for 1S-LSTM over frames for ECH task

for Action 2 with 22.75%. This probably due to the hanger labyrinth being heavily occluded during the installment, where just the roller are present and the location during execution is similar to the one for the first roller. With 38.89% Action 5 is again one of the classes that has the second worst classification accuracy. However compared to the results of the FNN, the FPR is lower for the other classes. This is probably due to similar reasons happening to Action 4, but the bumper brackets shape is very small compared to the hanger labyrinth, which leads to a more robust object detection. Action 6, which marks the installation of the shunt bracket, reaches an accuracy of 78.60%. There is a bigger mismatch with Action 4 with 12.63% compared to the other classes, even though both actions don't share many similarities in location and subsequent to each other. Action seven reaches an accuracy of 82.29% for the plugging in of the anti-opening device. Finally Action 8 is confused with Action 2 by a value of 61.54%, marking it with an accuracy of 7.69% again to the most difficult action to recognize.

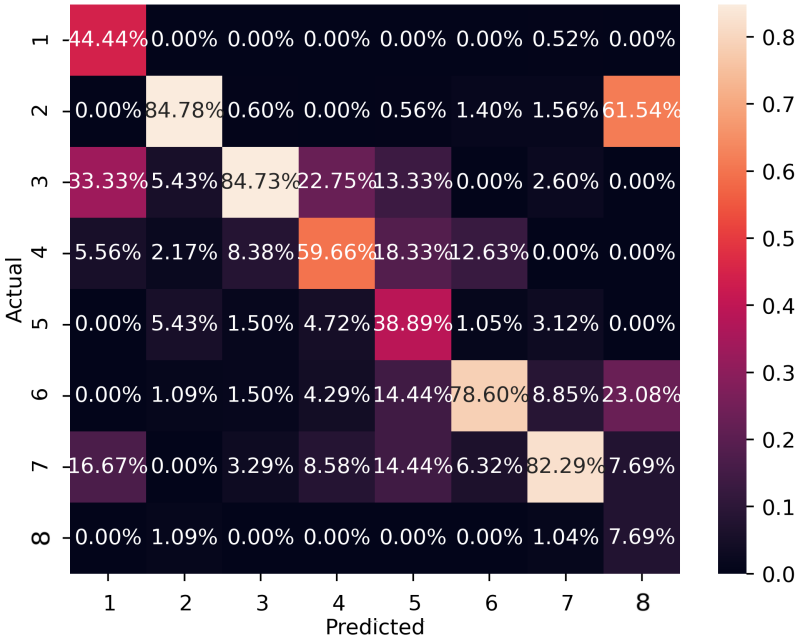


Figure 6.9: Confusion matrix for 1S-LSTM for ECH task

The classification success differs among the actions. In illustration 6.10 the multi class ROC curve is featured, as well as its AUC value by analyzing the false positive rate against the true positive rate. All classes are reaching an AUC over 0.9. Action 2, Action 3 and Action 7 are quite easy for the 1S-LSTM to classify with an AUC over 0.94, which corresponds to the high accuracies in the confusion matrix. Action 4 and Action 5 are more difficult to classify. Both curves start quite shallow and Action 5 reaches a TPR of 0.8 only for a FPR over 0.3, while the other classes are reaching the same TPR sooner. Action 1 and Action 8 are reaching high AUC values, however their accuracies are the lowest. This due to small amount of involved frames compared to the other actions and due to their prediction probabilities being close to the prediction probabilities of the wrongly classified actions.

6.1.3 Three Stream LSTM

The third experiment for this task is conducted with a 3S-LSTM network. Compared to a regular 1S-LSTM, three 1S-LSTM networks are implemented.

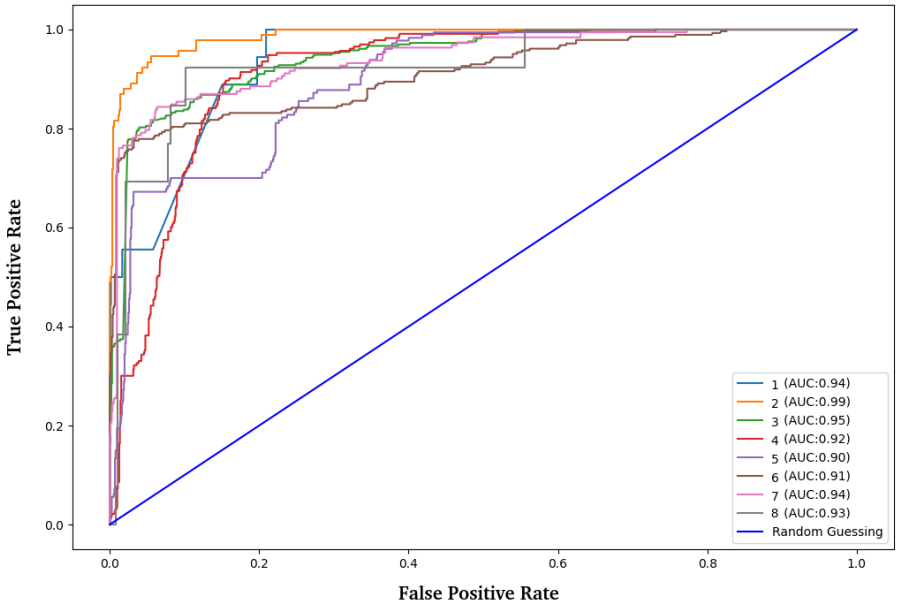


Figure 6.10: ROC curve of 1S-LSTM for ECH task

Each network is trained on different features of the data set, meaning one LSTM is trained on 2D joint coordinates, the second LSTM on the depth coordinates and the third LSTM on the object probabilities. Since the input data still needs to be three dimensional for each network, the window function is applied to the inputs. Therefore the shapes of the inputs tensors are $15332 \times 36 \times 3$, $15332 \times 18 \times 3$ and $15332 \times 5 \times 3$ respectively. The outputs of each network are then concatenated and are serving as input for a final dense layer, which provides the array with class prediction probabilities. The model architecture as well as the pre-processing of the input data is lined out in detail in section 5.4.3.

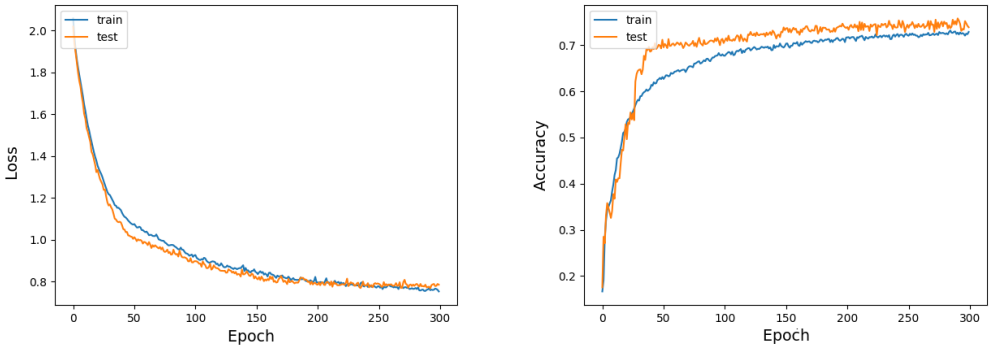
6.1.3.1 Training

The hyperparameters used for the training process can be observed in 6.4. A window size of three, dropout of 0.1 and batch size of 1024 are chosen through grid search. The same applies to the chosen ADAM optimizer with a learning rate of $1 \cdot 10^{-3}$, $1 \cdot 10^{5}$ decay steps and a decay rate of 0.9.
The network is trained for 300 epochs, since further epochs are leading to

Batch Size	Learning Rate	Decay Steps	Decay Rate
512	$1 \cdot 10^{-3}$	$1 \cdot 10^{5}$	0.9

Optimizer	Epochs	Dropout	Window Size
ADAM	300	0.15	3

Table 6.4: Hyperparameters for 3S-LSTM Network and ECH task



(a) Training Loss (b) Training Accuracy

Figure 6.11: Training for 3S-LSTM for ECH task

overfitting of the model. The progress of the model accuracy and model loss can be seen in image 6.11a and image 6.11b respectively. The training loss, as well as the validation loss are taking some time to decrease fast and converge to a value around 0.79. The training and validation accuracy are converging around 0.76. The training accuracy of the model reaches a value over 60% after 50 epochs and converges to an accuracy of 71.89% at the end of the training. The validation accuracy converges around 72.55%.

6.1.3.2 Results

The model is tested on 1347 unseen frames. Out of those frames 1018 are classified correctly, which leads to a classification accuracy of 75.68%. Image 6.12 shows the class probabilities for each frame in comparison to the ground truth. Action 1 is depicted in blue and is the starting position. It has an accuracy of 33.33%, which can be observed in its confusion matrix, which describes the ratio predicted and actual label and is depicted in image 6.13. Action 2 and Action 3 are containing the majority of wrongly classified frames, with 33.33% and 16.67% respectively. Action 1 corresponds to the hand over

action, which is depicted in orange. It reaches an accuracy of 79.32%. False classifications for the other classes are below 10% and the probability density distribution shows uncertainties during the transitions of the hand over action with the other actions. Action 3 is shown in green and marks the attachment of the rollers. This action is classified with an accuracy of 88.02%, which is the second highest value for all classified actions. Similar to the 1S-LSTM the accuracy rises sharply to 100 % after the attachment of the second roller, around frame 210. However, the accuracy starts out lower at 60% compared to the results for the 1S-LSTM with 75%. The FPR for the other classes ranges below 7%. The attachment of the hangar labyrinth is colored in red as Action 4. It reaches an accuracy of 75.11%. Due to occlusion of the work piece during its attachment, the prediction probability for the class drops to almost zero around frame 450 and rises again around frame 500, when the object detection module catches it again. However, Action 7 is predicted predominantly during the drop with a FPR of 21.03% overall. Action 5 has the lowest TPR with 32.78%. The corresponding task is the attachment of the shunt bracket onto the hangar, which is depicted in purple. The FPR is higher and lies around 37.22 % for Action 4, which is especially observable in 6.12 by the beginning of this task stage. The prediction probability for Action 5 is oscillating a lot, since the object detection has a hard time to detect the shunt bracket due to its small size. The attachment of the bumper bracket as Action 6 reaches a TPR of 81.40% and is shown in brown. The biggest FPR is found in Action 4 with 12.63%, and it is observable that this is due to the beginning of this stage, when the bumper bracket isn't detected yet. After frame 850 it is no more occluded and for the rest of the stage the probability stays at almost 100%. The last action that involves a work piece, which is anti-opening device, is Action 7 and shown in pink. It reaches a TPR of 87.50% and a FPR for all other classes below 10%. It has a high prediction probability through out all involved frames, showing that this class prediction is very robust. Class seven is the final positioning, marking the finish of the elevator task and is shown in grey. It reaches the highest TPR of 100% as only class.

The ROC curve in image 6.14 shows that all classes are reaching an AUC of more than 0.92, except for Action 5. It is again the most difficult to classify with an AUC of 0.85, which matches with the results shown in the probability density distribution 6.12 and confusion matrix 6.13. Action 2 and Action 3 on the other hand are the easiest to classify for the 3S-LSTM with an AUC of 0.98

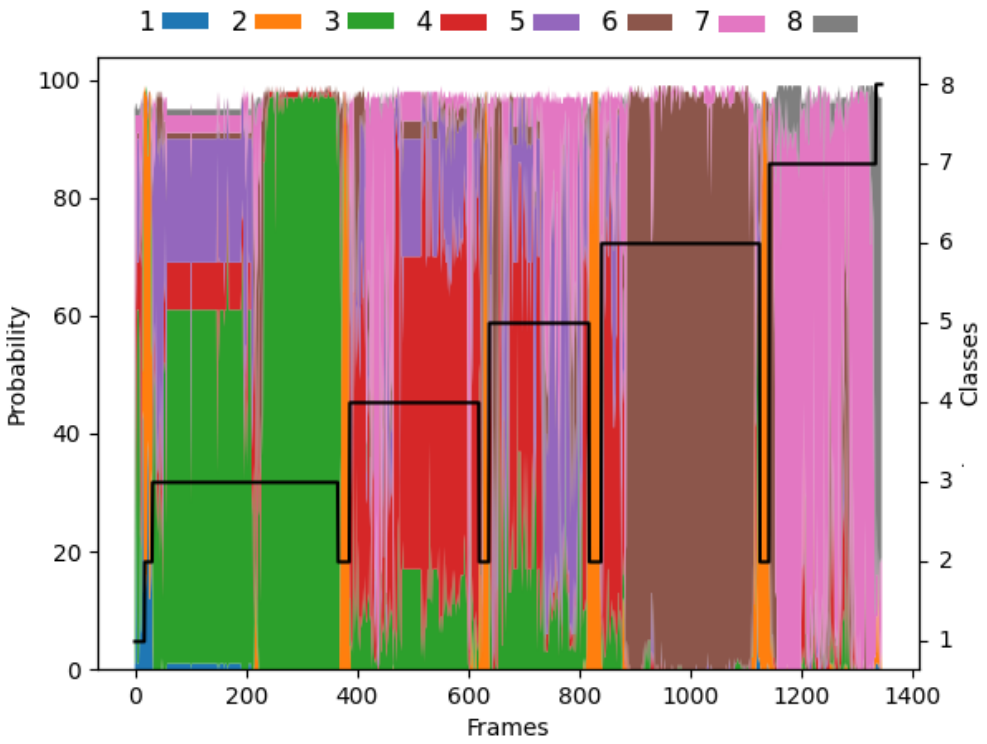


Figure 6.12: Accumulated class probabilities for 3S-LSTM over frames for ECH task

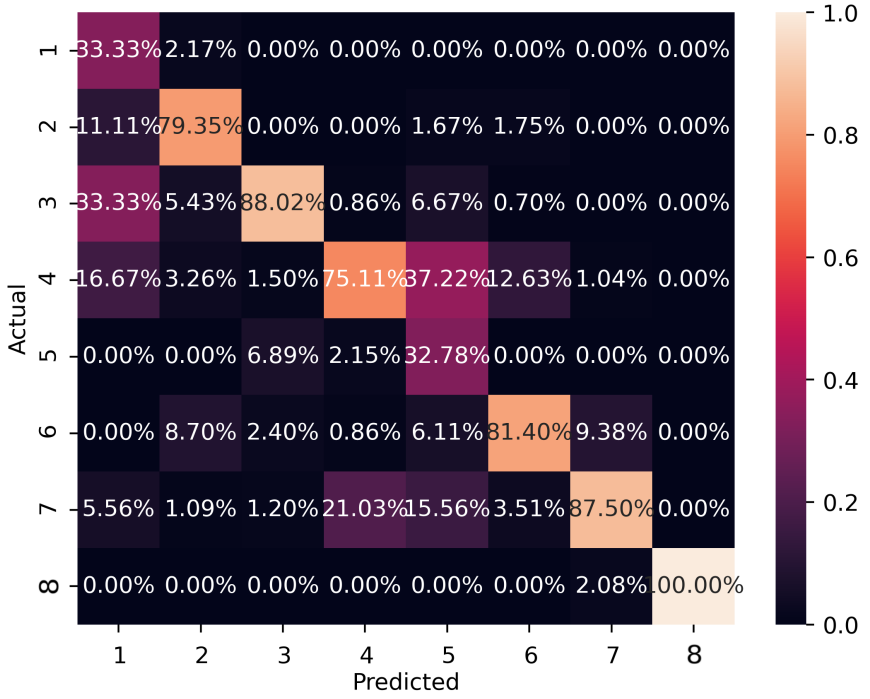


Figure 6.13: Confusion matrix for 3S-LSTM for ECH task

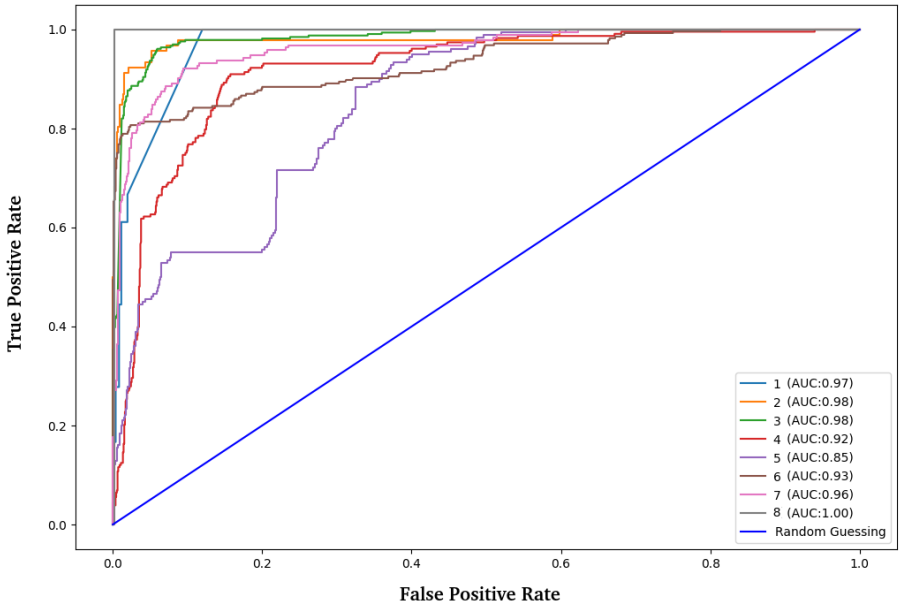


Figure 6.14: ROC curve of 3S-LSTM for ECH task

respectively. Action 4 produces a slow rising ROC curve that reaches an AUC of 0.92 and it is seen that it takes a FPR of 0.15 until reaching a TPR over 0.8. A similar behaviour of this is seen for Action 6. In both cases the probability density distribution showed low prediction values for both classes during the early stages of their executions. The results show that for the 3S-LSTM model it is even more observable that the second half of each action contains a higher density of correctly classified frames compared to the results of the neural network 6.1.1 and 6.1.2. In most cases the used tools are small metal plates, that are just fully observable after getting picked out of their containers and being attached to the hangar plate, without having the occlusion due to hands or body of the constructor. Also training three networks, one for each feature category, allows smaller feature categories, like the detected objects with five values compared to the 36 of the skeleton, to be more influential in the classification process. This is especially true for Action 4 and Action 5, which have an increased accuracy of 75.11% and 39.89% respectively. It is also the only model that is able to recognize Action 8 correctly with an accuracy of 100% with an AUC of 1.

6.1.4 Temporal Segment LSTM

The fourth experiment is conducted with a Temporal Segment LSTM (TS-LSTM) network. This model consists of four separately trained 1S-LSTM networks. Each network is getting the same input features, but the dimensions of the input tensors are varying due to different window sizes. Each network delivers a vector with probabilities for each class for each frame as output. All models of the ensemble are predicting on unseen data individually and the average of their predictions is chosen as the final result. This is done in order to analyse the influence on short and long-term temporal components of different window sizes. However training of the network revealed that to big window sizes would lead to drastically worse performances. Therefore the shapes of the input tensors are $15332 \times 59 \times 3$, $15332 \times 59 \times 7$, $15332 \times 59 \times 12$ and $15332 \times 59 \times 15$. The model architecture as well as the pre-processing of the input data is lined out in detail in section 5.4.4.

6.1.4.1 Training

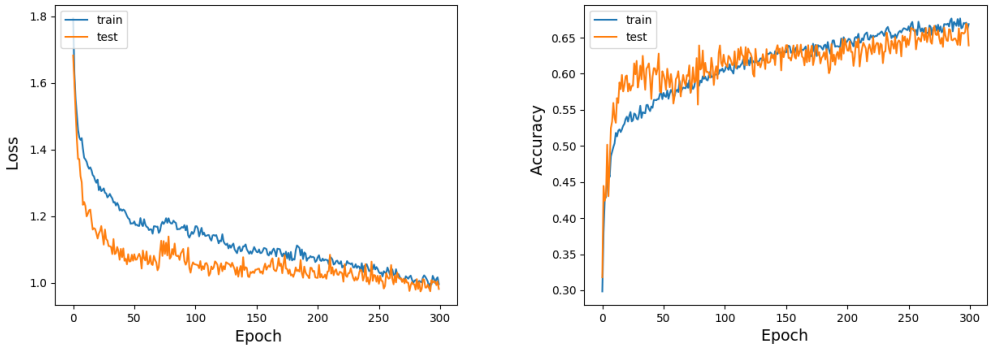
The hyperparameters can be observed in table 6.5. A dropout of 0.1 and batch size of 1024 are chosen through grid search. The same applies to the chosen ADAM optimizer with a learning rate of $1 \cdot 10^{-3}$, $1 \cdot 10^{5}$ decay steps and a decay rate of 0.9. The window sizes are chosen to 3,7,12 and 15 respectively.

Batch Size	**Learning Rate**	**Decay Steps**	**Decay Rate**
512	$1 \cdot 10^{-3}$	$1 \cdot 10^{5}$	0.9

Optimizer	**Epochs**	**Dropout**	**Window Size**
ADAM	300	0.15	[3, 7, 12, 15]

Table 6.5: Hyperparameters for TS-LSTM network and ECH task

The network is trained for 300 epochs, since further epochs are leading to over fitting of the model. The progress of the accumulated model accuracy and model loss of all four trained models can be seen in image 6.15a and image 6.15b respectively. The training loss, as well as the validation loss are taking some time to decrease with around 200 epochs in order to fall under 1. The training stops at epoch 300 in order to prevent over fitting since the validation



(a) Training Loss (b) Training Accuracy

Figure 6.15: Training for TS-LSTM for ECH task

loss converges around 0.91 while the training loss is still decreasing after that. The final training loss reaches a value of 0.85. The training accuracy of the model reaches a value over 60% after 100 epochs and converges to an accuracy of 65.39% at the end of the training. The validation accuracy converges around 63.55%.

6.1.4.2 Results

The TS-LSTM model is tested on 1347 unseen frames. Out of those frames 930 are classified correctly, which leads to a classification accuracy of 69.04%. Image 6.16 shows the class probabilities for each frame in comparison to the ground truth. The confusion matrix for this experiment is seen in 6.16, which gives an overview of the ratio between the actual and predicted class label in percentage based on the corresponding amount of frames per class. The starting position as Action 1, which is shown in blue, achieves a TPR of 5.56%. Action 3 and Action 7 are predicted the most instead with FPRs of 33.33% and 50.00% respectively. The handover is depicted in orange as Action 2 and reaches a TPR of 90.22%. The FPR is below 7% for the other classes. The installation of the rollers as Action 3 is shown in green. This action reaches a classification accuracy of 86.23%. Similar to the results of the 1S-LSTM, the prediction probability for this action starts at 60% and jumps to around 90% during the second half of this action. The FPR of the other classes are below 7% in this case. Action 4 corresponds to the attachment of the hangar labyrinth and is shown in red. A TPR of 59.66% is achieved. Action 3 and

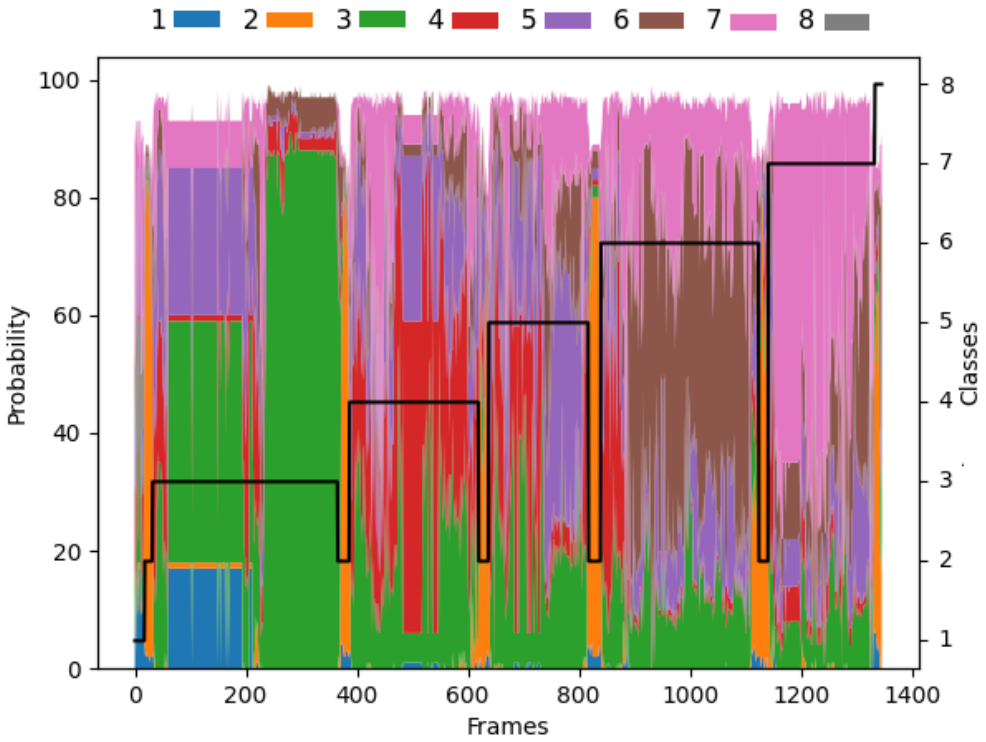


Figure 6.16: Accumulated class probabilities for TS-LSTM over frames for ECH task

Action 7 are predicted mainly for the wrongly classified frames with 15.45% and 18.88% respectively. The installation of the shunt bracket relates to Action 5 and is depicted in purple. It reaches a TPR value of 37.22%. Action 3 and Action 4 are are predicted mainly for the wrongly classified frames with 22.78% and 26.67% respectively. The attachment of the bumper bracket as Action 6 is depicted in brown. It reaches an TPR of 72.63% with FPRs of 10.53% and 9.47% for Action 6 and Action 7 respectively. Compared to the 3S-LSTM the class accuracy rises also after frame 850, however it doesn't reach a probability close to 100% and the prediction probabilities for class five during this stage are oscillating a lot. In pink is Action 7 presented, which corresponds to the installation of the anti-opening device. It reaches a classification accuracy of 76.04% and a FPR of 13.54% for class five. Action 8 as the finishing pose is depicted in grey and is not classified at all with 0%. Action 2 and Action 6 are predicted during that stage with an FPR of 69.23% and 30.77% respectively.

The ROC curve in image 6.18 shows that all classes are reaching an AUC of

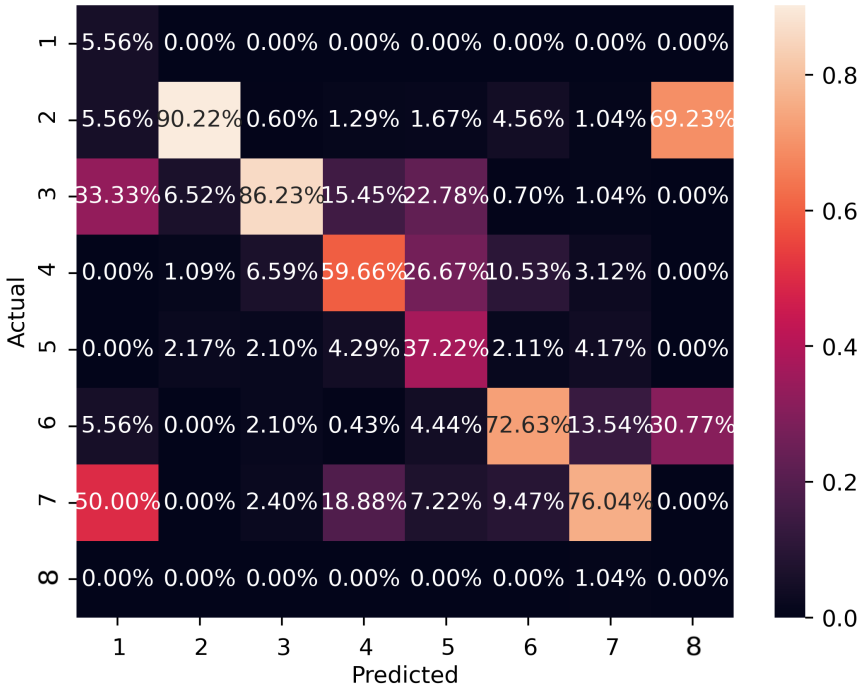


Figure 6.17: Confusion matrix for TS-LSTM for ECH task

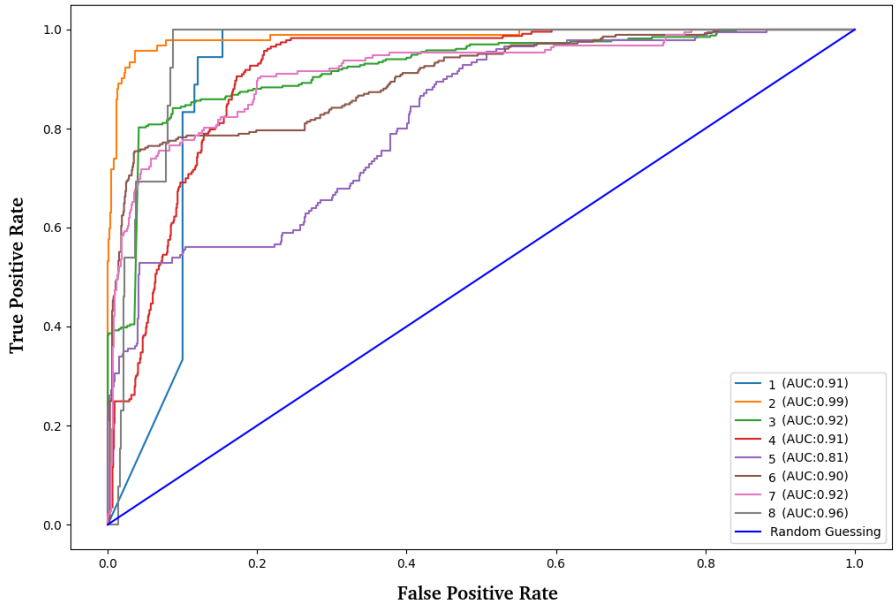


Figure 6.18: ROC curve of TS-LSTM for ECH task

more than 0.90. Compared to the ROC curves for the other models, several classes have a shallow slope or are even starting with an offset. It shows that the model is capable of identifying the classes, but it struggles more. Action 5 is again the most difficult action to classify with an AUC of 0.81, which matches with the results shown in the probability density distribution 6.12 and confusion matrix 6.13. Action 2 other hand is the easiest to classify for the TS-LSTM with an AUC of 0.99. Action 6 produces a slow rising ROC curve that reaches an AUC of 0.90 and it is seen that it takes a FPR of 0.55 until reaching a TPR over 0.9. A similar behaviour of is see for Action 5. In both cases the probability density distribution showed low prediction values for both classes during the early stages of their executions. Action 2, Action 3 and Action 7 are also rising slow compare to their equivalent ROC curves of the other models and are reaching an AUC of 0.92, 0.91 and 0.92 respectively. Overall the performance of this model shows a lower classification accuracy for each class and more oscillation for the prediction probabilities. Compared to the other models the variation of the window sizes doesn't enhance the classification capability for this experiment setup. This is probably due to the different action durations, where actions involving more frames are getting

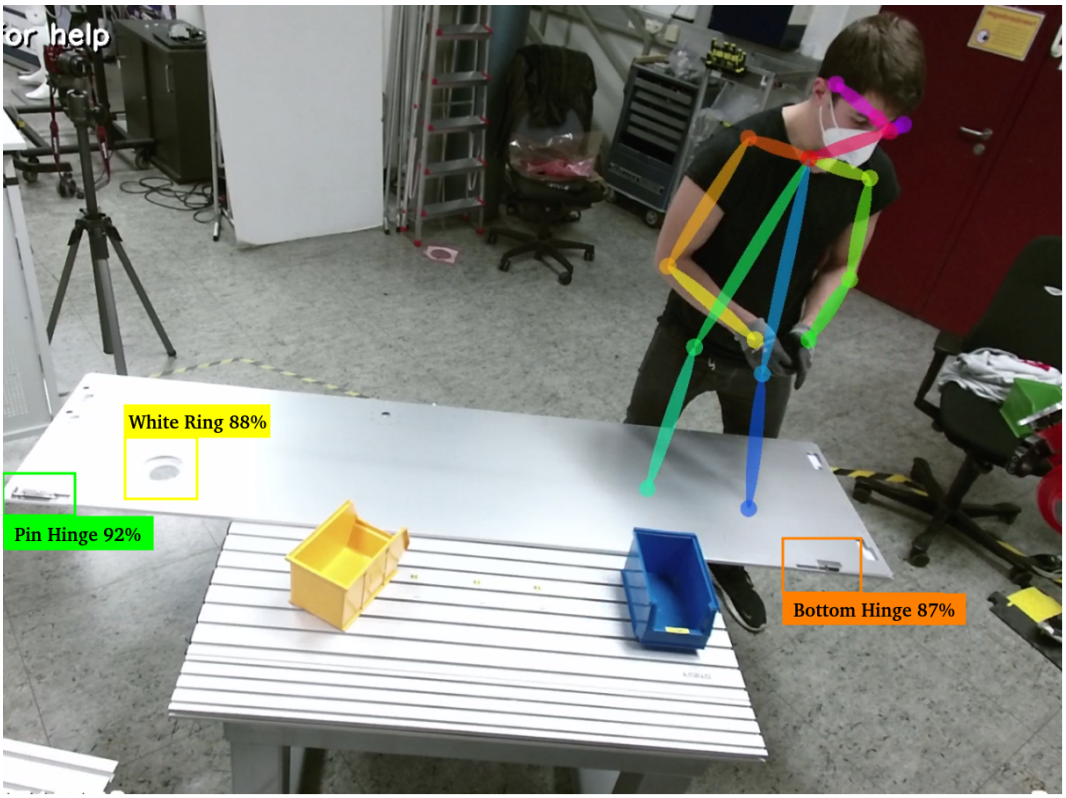


Figure 6.19: Skeleton and object tracking of the operator and object detection during the execution of the control cabinet door assembly task

more weight for bigger window sizes.

6.2 Data Set: Control Cabinet Door

The data set of the control cabinet door features the assembly task setup described in section 5.6 in detail and will be additionally referred to as *CCD task* in the following sections. This data set consists of RGB-D video recordings of the control cabinet door assembly task recorded with the Kinect v2 with a frame rate of 13 fps. The data set is created by recording two participants, both male, which are performing the assembly task seven times each. The whole test set contains 16745 frames, distributed among 14 recorded assembly scenarios. Image 6.19 shows the real time application of the skeleton tracking and object detection, while the operator is executing the assembly task.
For the second assembly task four work pieces are tracked, as well the skeleton of the operator and their depth values. The threshold for accepting the

detecting the work pieces in the videos are set to 95%, since the work pieces are easy to detected for this assembly setup. The data set is labelled, with each frame getting assigned one action involved in the assembly task as label, which are described in detail in 5.6. There are overall seven classes - the starting position, handover, white ring placement ring, metal plate adjustment, pin hinge installation, bottom hinge installation and lock installation. Furthermore the executed actions will be referred by a number corresponding to their chronological appearance instead by a description for convenience. Table 6.6 is showing the translation between action explanation and their numeration, e.g Action 2 is corresponding to the hand over action.

Action 1	Starting position in front of empty door plate
Action 2	Handing over work pieces
Action 3	Placing of white ring
Action 4	Pulling metal frame over door plate
Action 5	Screwing on of pin hinge
Action 6	Screwing on of bottom hinge
Action 7	Installation of locker

Table 6.6: Enumeration of actions and their corresponding execution steps for the CCD task

This means that overall each frame has 58 features for the second assembly task. Each of the used models trained for this setup, are using 12321 frames for training, 2575 frames for validation and 1849 unseen frames for testing purposes. Since the temporal dependencies are important, the frames aren't chosen randomly. Instead the frames of the validation and test set are belonging to two and one whole recordings respectively, in order to get access to all steps and actions of the assembly task. Image 6.20 shows the amount of recorded frames, each action of the assembly task is assigned to. In red is the mean $A = \frac{1}{n}\sum_{i=1}^{n} a_i$ of the frames involved in the training process depicted, as well as the standard deviation $SD = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$ as error bar to show amount of variability. It is observable that some actions like the starting position or placement of the white ring are featured in fewer frames compared to the attachment of the locker e.g. In case of the handover the total amount of

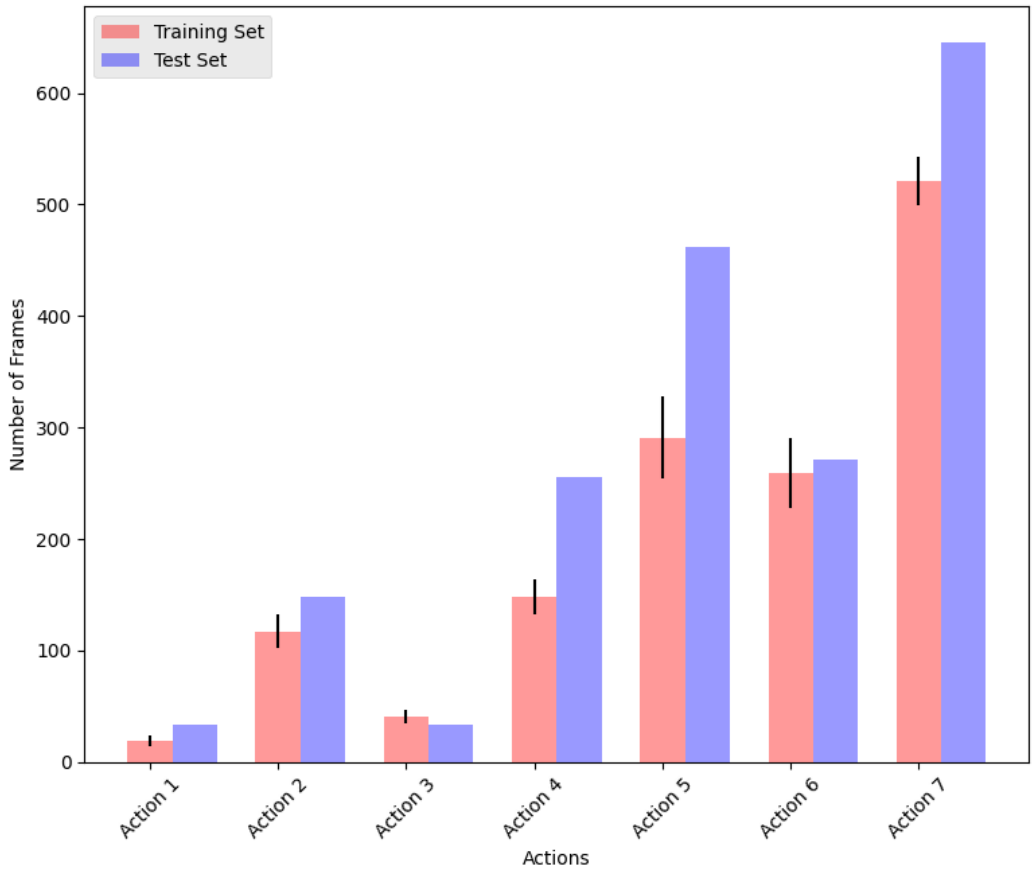


Figure 6.20: Frame distribution for each action of CCD task

frames during the task is shown, however this is the only action that happens more than once with four occurrences in total. This means the action itself involves just one fourth of the depicted frame number. The same holds true for the distribution of frames involved in the model performance testing, which is depicted in blue. Overall the test set involves more frames per action than the mean over the training set. This is due to the test set being chosen randomly , before the remaining data is used for the training, and features recordings where the participants aren't as experienced with the execution yet. This results into more frames, compared to later recordings, where the operators are assembling faster, which leads to fewer frames.
In the following the experiments for each model are presented. First the training schedule and hyperparameters are described. Secondly the results are

shown and analyzed with the classification probability distribution, the confusion matrix and the multiclass receiver operating characteristics (ROC) curve.

6.2.1 Forward Neural Network

The first experiment is conducted with the FNN for focusing spatial spatial dependencies over temporal dependencies. The model architecture as well as the pre-processing of the input data is lined out in detail in section 5.4.1. The dimension of the input tensor for training is 14896×58 and the output is a vector of probabilities for the seven classes for each frame. For the training process the data set is shuffled, since the network doesn't take temporal dependencies into consideration.

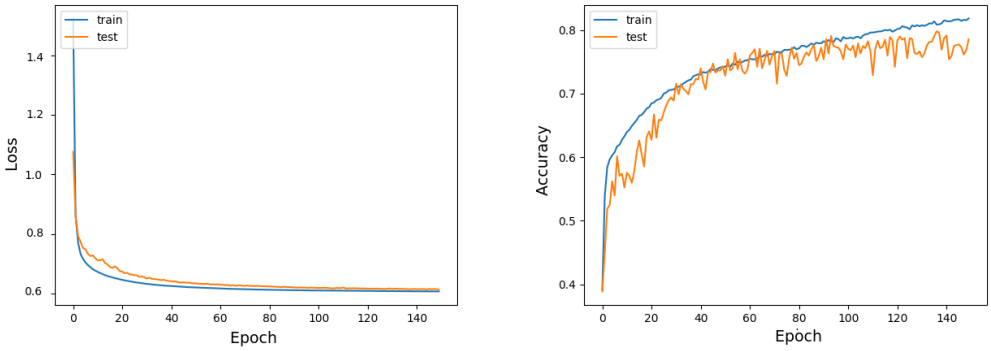
6.2.1.1 Training

The hyperparameters of the training schedule can be observed in table 6.7. A batch size of 128, the ADAM optimizer and a learning rate of $1 \cdot 10^{-3}$ are chosen via grid search.

Batch Size	**Learning Rate**	**Optimizer**	**Epochs**
128	$1 \cdot 10^{-3}$	ADAM	150

Table 6.7: Hyperparameter for the training of FNN for CCS task

The network is trained for 150 epochs, since further epochs are leading to over fitting of the model. The progress of the model accuracy and model loss can be seen in image 6.21a and image 6.21b respectively. The training loss, as well as the validation loss are dropping very fast to under 0.6 within the first 40 epochs. The final training loss reaches a value of 0.54, with the validation loss being higher around 0.56. The training accuracy of the model reaches a value over 70% after 40 epochs and converges to an error of 81.06% at the end of the training. The validation loss takes up to 50 epochs until it reaches an accuracy over 70%. After that it increases, but it oscillates around the value of the training accuracy and stays at around 79%.



(a) Training loss (b) Training accuracy

Figure 6.21: Training for FNN for CCD task

6.2.1.2 Results

Testing the model on unseen data is yielding the following results. Out of 1849 test frames, 1462 are classified as true positive values. This is a ratio of 79.06% classification accuracy. Image 6.22 shows a visualization of class probabilities per frame in comparison to the actual label, which is presented as black line. The corresponding colors for each class can be seen in the legend. Action 1 is depicted in blue and marks the starting position. It can be see that there is an overlap between the actual label and prediction of Action 1. This is further underlined by the confusion matrix presented in image 6.23. It shows the ratio between actual label and prediction of each class. For Action 1 the TPR is 30.30% and a huge chunk of the FPR is predicted as Action 7 with 63.64 %, which corresponds to the screwing on of the lock and almost no other false positives for the other classes.
Action 2, presented in orange, corresponds to the handover action. It can be observed, that the handover action has fewer frame occurrences with around 15 frames per action and is the only action where the frames aren't all connected in one sequence. However still 76.35% of all predictions for Action 1 are true, even with this class imbalance. The false positives are distributed quite evenly among the other classes with 3%-6%, which can be contributed to the handover being connected to each action and therefore due to classification uncertainties while transitions are occurring. It is also observable in image 6.22, that more false classifications are occurring at the end of an action towards a handover, than from handover towards next action.

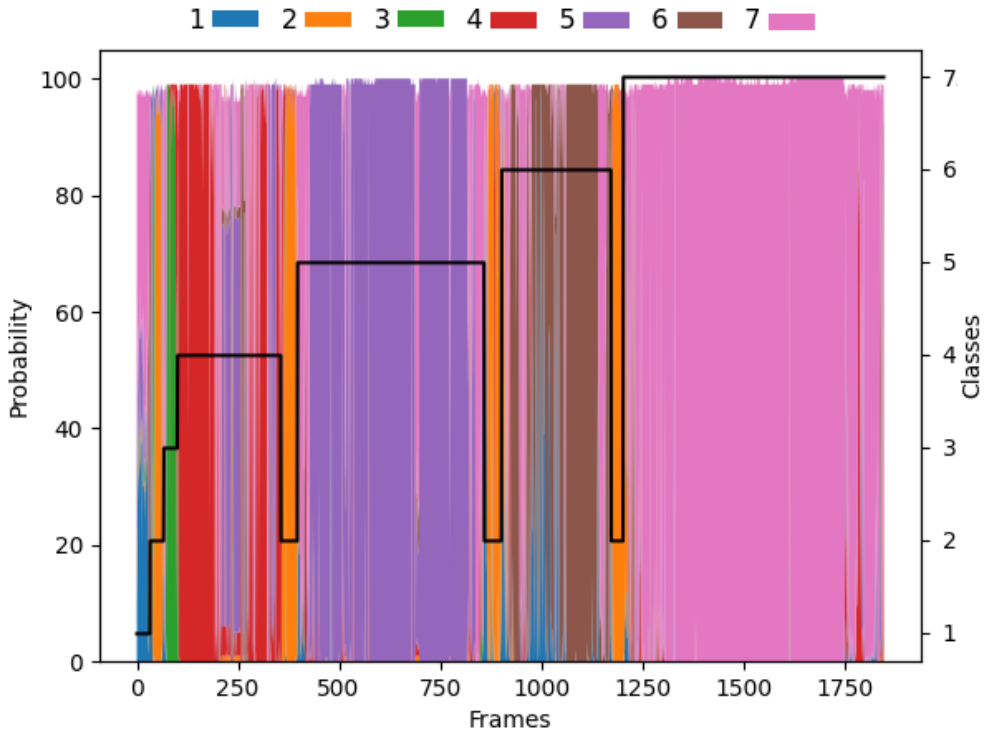


Figure 6.22: Accumulated class probabilities for FNN over frames for CCD task

Action 3 is shown in green and represents the placement of the white ring. This is also a very short class with 25 frames and is predicted with a TPR of 73.53%. False predictions are made for class six with 11.76% and one and three with 8.82% respectively. This class prediction also shows the errors during the transitions with its neighbor actions, since Action 3 and Action 4 are evolving into each other without a handover in between. Action 7 might be influenced by straight standing positioning in the middle of the scene.
In red is Action 4 depicted, which translates to the pulling of the metal plate on top of the cabinet door frame. This action has the lowest TPR with 60.94%. It is the only action that is not started by a handover action. However no FPR is found from Action 3, showing that the transition between those two actions is fairly smooth. However as can be seen in illustration 6.22 after almost 100 frames classification probabilities of over 90%, in the middle of the action, the accuracy drops sharply to under 20%. Action 7 is predicted during that time,

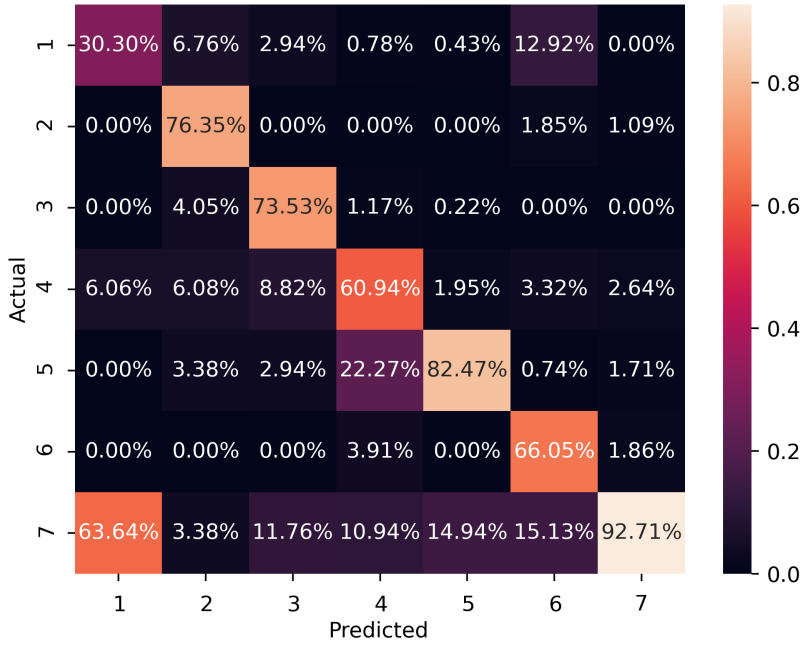


Figure 6.23: Confusion matrix for FNN for CCD task

with a FPR of 10.94%. This is the time period after the pulling of the metal plate and during its adjustment to make sure that the metal plate and cabinet door are aligned as close as possible. While the pulling of the plate is similar among all recordings, the alignment varies a lot for each recording. This leads to different adjustments of the border placement and locations the worker has to move around the plate, making it harder to generalize those movements well.

Action 5 is the screwing action of item a in the left corner and presented in violet. This action has the second highest TPR with 82.47% and is also the second largest with over 500 frames. A lot of false classification is happening towards the end of the action, before the handover. Also the wrong predictions are counting mostly towards Action 7 with 14.94%, while the worker is moving back from the corner position towards the middle of the plate.

In brown is the screwing of the bottom hinge in the right corner depicted, which corresponds to Action 6. The TPR lies at 66.05%. In this case Action 7 and Action 1 are sharing the amount of falsely predicted class five labels, with around 12.92% and 15.13% respectively. Again near the end of the action to-

wards the hand over the error increases. However in this class false predictions are happening throughout the action, with no other error clusters, except the ending.
Action 7 is depicted in pink with a TPR of 92.71% and has barely any false predicted frames, which are evenly distributed among the other classes. It is by far the class with the most frames involved, over 600, which can be seen in 6.22. Therefore it is reasonable that all other class predictions are finding an amount of their predictions marked as Action 7. In comparison Action 2 with 3.38%, Action 5 with 10.94% and Action 6 with 10.94% have the lowest amount of false predictions towards Action 7. This might be due to very distinct movement patterns like the hand over, or bending forward, as well as distinct positioning at the corners of the table for Action 5 and Action 6.
The classification success differs from among the classes. An illustration 6.24 the multi class Roc curve is featured, as well as its AUC value by analyzing the false positive rate against the true positive rate. All classes are reaching an area over 0.89 which indicates that the model is indeed capable of classifying each action. However Action 2, Action 3 and Action 6 have values over 98%, which indicates that those classes contain some features are making it easier for the classification than the other classes. Likewise the ROC-curve for those three classes is stating very steep and reaching a true positive rate of 100% without having more than 10% of false positives. Action 4 on the other hand reaches over 90% TPR, with a FPR over 50%. This indicates that Action 4 is harder for the model to clearly separate compared to the other classes. Similar hard to classify is Action 1 which is always predicted with a FPR offset, before predicting the correct labels. Since Action 1 is the shortest class with just 10 frames on average, and occurs just in the beginning, only the location of the worker and its posing is taken into account, which does overlap with poses involved in other actions.

6.2.2 Single Stream LSTM

The second experiment is conducted with a 1S-LSTM network. Now the temporal components as well as the spatial components are involved for the classification process. Since the input data needs to be three dimensional, the window size determines the third dimension. Therefore the shape of the input tensor is $14896 \times 58 \times 12$ and the output is a vector of probabilities for the eight classes for each frame. The model architecture, as well as the pre-processing

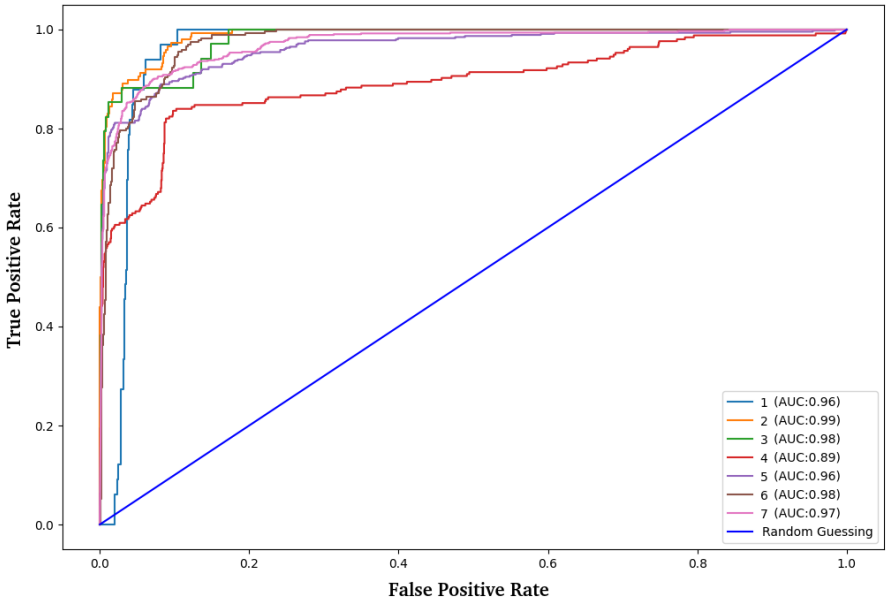


Figure 6.24: ROC curve of TS-LSTM for CCD task

of the input data is lined out in detail in section 5.4.2.

6.2.2.1 Training

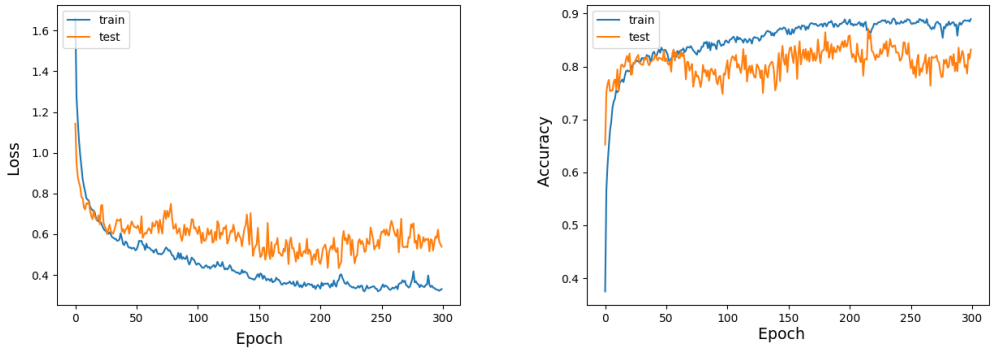
The hyperparameters for the training process can be observed in table 6.8. A batch size of 512, dropout of 0.1 and window size of 12 are chosen through grid search. The same applies to the chosen ADAM optimizer with a learning rate of $1 \cdot 10^{-3}$, $1 \cdot 10^{5}$ decay steps and a decay rate of 0.9.

Batch Size	Learning Rate	Decay Steps	Decay Rate
512	$1 \cdot 10^{-3}$	$1 \cdot 10^{5}$	0.9

Optimizer	Epochs	Dropout	Window Size
ADAM	300	0.1	12

Table 6.8: Hyperparameters for 1S-LSTM Network and CCD task

The network is trained for 300 epochs, since further epochs are leading to over



(a) Training loss (b) Training Accuracy

Figure 6.25: Training for 1S-LSTM for CCD task

fitting of the model. The progress of the model accuracy and model loss can be seen in image 6.25a and image 6.25b respectively. The training loss, as well as the validation loss are dropping very fast to under 0.6 after the first 40 epochs. The final training loss reaches a value of 0.42, with the validation loss being higher around 0.57. The training accuracy of the model reaches a value over 70% after 17 epochs and converges to an error of 84.06% at the end of the training. The validation accuracy takes up to 25 epochs until it reaches an accuracy over 70%. After that it increases, but it oscillates around the value of the training accuracy and stays at around 80.92%.

6.2.2.2 Results

Testing the model on unseen data is yielding the following results. Out of 1849 test frames, 1548 are classified as true positive values. This is a ratio of 80.54% classification accuracy. Image 6.26 shows a visualization of class probabilities per frame in comparison to the actual label, which is presented as black line. The corresponding colors for each class can be seen in the legend. Action 1 is depicted in blue and marks the starting position. The confusion matrix presented in image 6.27 shows the ratio between actual label and prediction of each class. For Action 1 the TPR is 3.03% and a almost all of the FPR is predicted as Action 7, which corresponds to the installation of the lock.
Action 2, presented in orange, corresponds to the handover action. The model predicts 82.43% of all frames correctly. 18.24% of the false positives can be found in Action 7 again.

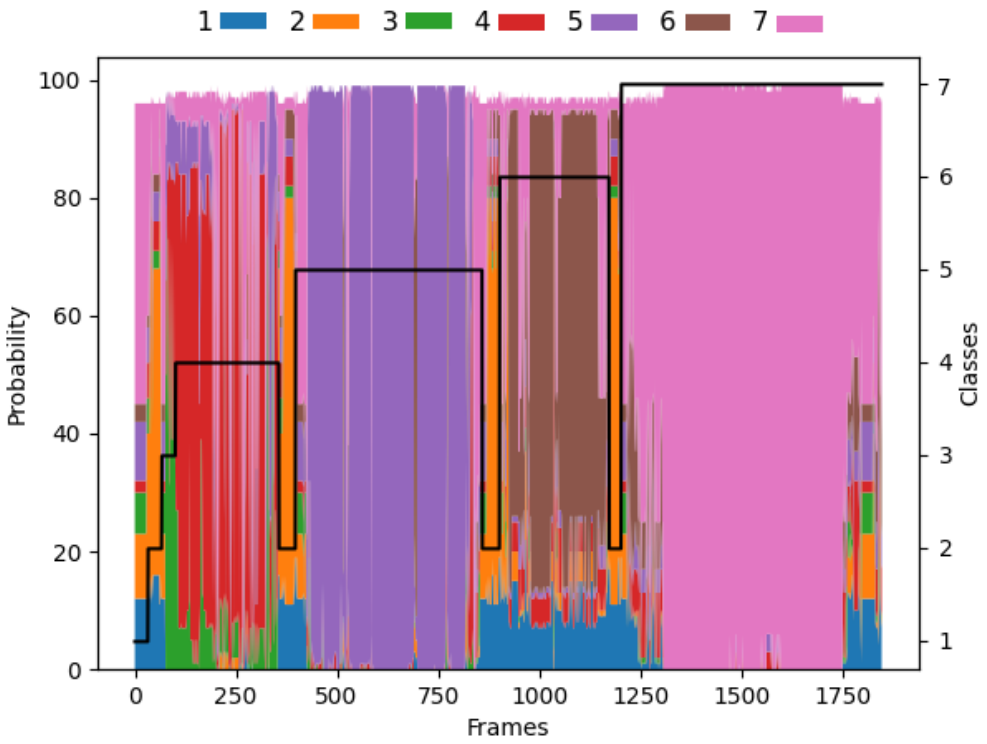


Figure 6.26: Accumulated class probabilities for 1S-LSTM over frames for CCD task

Action 3 two is shown in green and represents the placement of the white ring. This is also a very short class with 25 frames and is predicted with a TPR of 32.35%. False predictions are made towards Action 4 with 41.18% and Action 7 with 26.47% respectively. This class prediction also shows the errors during the transitions from placing the white ring to the pulling of the metal plate.
In red is the Action 4 depicted, which translates to the pulling of the metal plate on top of the cabinet door frame. This action has a FPR of 74.61%. Wrongly predicted frames of this action are predicted as Action 7 with 16.80%. However for Action 3 a FPR of just 3.12% . This shows that the model confuses the action of placing the white ring with its subsequent action, but pulling and adjusting the metal plate is not confused with its predecessor action. Even though the prediction probability doesn't drop to 20% for several frames like for the FNN, image 6.26 shows, that the prediction probabilities for Action 4 are still oscillating a lot towards the end of this action. Again Action 7 is predicted during that time, with a FPR of 16.80%. This and the higher accuracy of 10% for Action 4 for the LSTM compared to the FNN could be traced towards the temporal information, which are now taken into account and not just the locations of the skeleton poses.

Action 5 is the screwing on of the pin hinge in the left corner and presented in violet. This action has the second highest TPR with 82.68% and is also the second largest with over 500 frames. A lot of false classification is happening towards the end of the action, before the handover. Also the wrong predictions are counting mostly towards Action 7 with 12.12%, while the worker is moving towards and back from the corner position to the middle of the plate in the beginning and end of the action.
In brown is the installation of the bottom hinge in the right corner depicted as Action 6. The TPR lies at 75.65%. In this case Action 7 features 17.34% of the falsely predicted label. Again near the end of the action towards the hand over the error increases. However in for this action false predictions are happening throughout the action, with the biggest error clusters occurring towards the ending.
Action 7 is depicted in pink with a TPR of 98.76% and has barely any false predicted frames, which are evenly distributed among the other classes. It is by far the class with the most frames involved, over 600, which can be seen in 6.26. Therefore it is reasonable that all other class predictions are finding an amount of their predictions marked as class six. On top of that every other action also

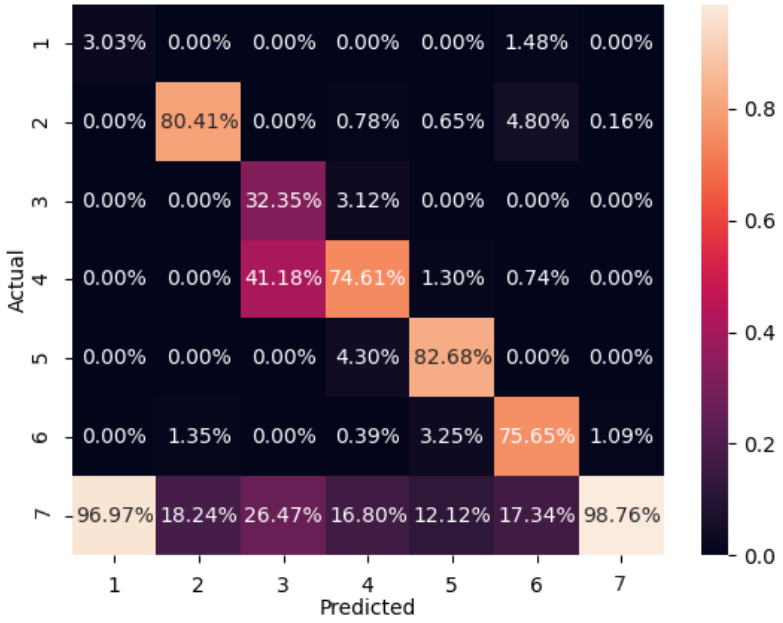


Figure 6.27: Confusion matrix for 1S-LSTM for CCD task

involves the operator to pass the middle of the plate while executing the other action, but this location occurs many more frames compared to other classes, which leads to wrong predictions for them. Also up from Action 4, the later executed actions are all achieve a higher prediction accuracy for the LSTM compared to the FNN, showing the importance of the temporal component.
The classification success differs from among the classes. In illustration 6.32 the multi class ROC curve is featured, as well as its AUC value by analyzing the false positive rate against the true positive rate. All classes are reaching an area over 0.87, which indicates that the model is indeed capable of classifying each action. However Action 2, Action 6 and Action7 are achieving values over 0.98, which indicates that those classes contain some features are making it easier for the classification than the other classes. Likewise the ROC curve for those three classes is starting very steep and reaching a true positive rate of 100% without having more than 10% of false positives. Action 3 stagnates at 0.7 until it jump to 1 for a FPR at 0.2. This indicates that Action 3 is harder for the model to clearly separate compared to the other classes. This is also reflected in the split of predictions between Action 3 and Action 4. Similarly difficult to classify is Action 1, which is even predicted with a bigger FPR

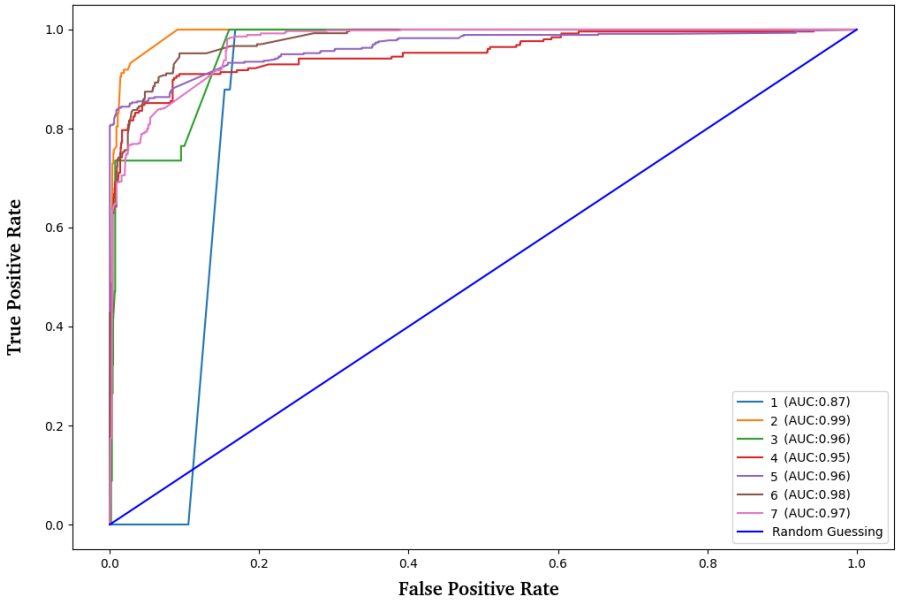


Figure 6.28: ROC curve of 1S-LSTM for CCD task

offset of 0.15 compared to the results of the ROC for the FNN, which can be observed in 6.32, before predicting the correct labels. Since Action 1 is the shortest class with just 10 frames on average, and occurs just in the beginning, only the location of the worker and its posing is taken into account, which targets the predictions heavily to Action 7.

6.2.3 Three Stream LSTM

The third experiment for this task is conducted with a 3S-LSTM network. Compared to a regular 1S-LSTM, three 1S-LSTM networks are implemented. Each network is trained on different features of the data set, meaning one LSTM is trained on 2D joint coordinates, the second LSTM on the depth coordinates and the third LSTM on the object probabilities. Since the input data still needs to be three dimensional for each network, the window function is applied to the inputs. Therefore the shapes of the inputs tensors are $14896 \times 36 \times 12$, $14896 \times 18 \times 12$ and $14896 \times 4 \times 12$ respectively. The outputs of each network are then concatenated and are serving as input for a final dense layer, which provides the array with seven class prediction probabilities. The model

architecture as well as the pre-processing of the input data is lined out in detail in section 5.4.3.

6.2.3.1 Training

The hyperparameters used for the training process can be observed in 6.9. A window size of three, dropout of 0.1 and batch size of 1024 are chosen through grid search. The same applies to the chosen ADAM optimizer with a learning rate of $1 \cdot 10^{-3}$, $1 \cdot 10^{5}$ decay steps and a decay rate of 0.9.

Batch Size	Learning Rate	Decay Steps	Decay Rate
512	$1 \cdot 10^{-3}$	$1 \cdot 10^{5}$	0.9

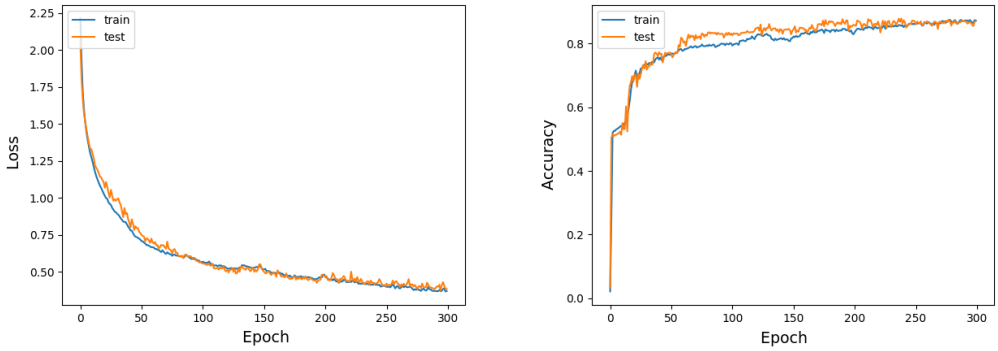
Optimizer	Epochs	Dropout	Window Size
ADAM	300	0.1	12

Table 6.9: Hyperparameters for 3S-LSTM Network and CCD task

The network is trained for 300 epochs, since further epochs are leading to over fitting of the model. The progress of the model accuracy and model loss can be seen in image 6.29a and image 6.29b respectively. The training loss and the validation loss are proceeding closely together. They are dropping fast under 0.5 after 100 epochs. The final training loss reaches a value of 0.47, with the validation loss being higher around 0.49. The training and validation accuracy are also evolving closely together. After 100 epoch they are both are reaching values over 0.8. The training accuracy converges at 89.23%. The final value of the validation accuracy reaches around 88.94%.

6.2.3.2 Results

Testing the model on unseen data is yielding the following results. Out of 1849 test frames, 1626 are classified as true positive values. This is a ratio of 88.56% classification accuracy. Image 6.30 shows a visualization of class probabilities per frame in comparison to the actual label, which is presented as black line. The corresponding colors for each class can be seen in the legend. The first class is depicted in blue and marks Action 1. The confusion matrix is presented in image 6.31 and shows the ratio between actual label and prediction of each



(a) Training loss (b) Training Accuracy

Figure 6.29: Training for 3S-LSTM for CCD task

class. For Action 1 the TPR is 3.03% and a huge amount of the FPR are predicted as Action 2 and Action 3, which corresponds to the handover phase and the placing of the white ring. This might be for the pose and location of the operator during Action 1, but another similarity lies in the sparse presence of detected objects for the first work piece, the white ring, isn't featured in many frames yet.
Action 2 is presented in orange. It can be observed, that the handover action has fewer frame occurrences with around 25 frames per action and is the only action where the frames aren't all connected in one sequence. However this stage contains a TPR of 74.32%. A small amount of the FPR can be found predominantly in class three, four and five with 10.14%, 6.76% and 6.76% respectively. This can be explained with the distinctive movement of the handover for the high accuracy and the FPR during the transition phases, which can also be observed in 6.30 towards the end or beginning of the other actions.

Action 3 is shown in green and represents the placement of the white ring. It has a TPR of 58.82% and has the second lowest accuracy among all classes. Similar to the 1S-LSTM the false predictions are made for Action 4 with 41.18%, which are occurring mostly towards the end of Action 3.
Action 4 is depicted in red, which translates to the pulling of the metal plate on top of the cabinet door frame. This action has a TPR of 95.31% and is therefore the best predicted class by this model. Falsely classified frames are mostly belonging to the handover phase with 3.12%, which occurs only at the end of Action 4.

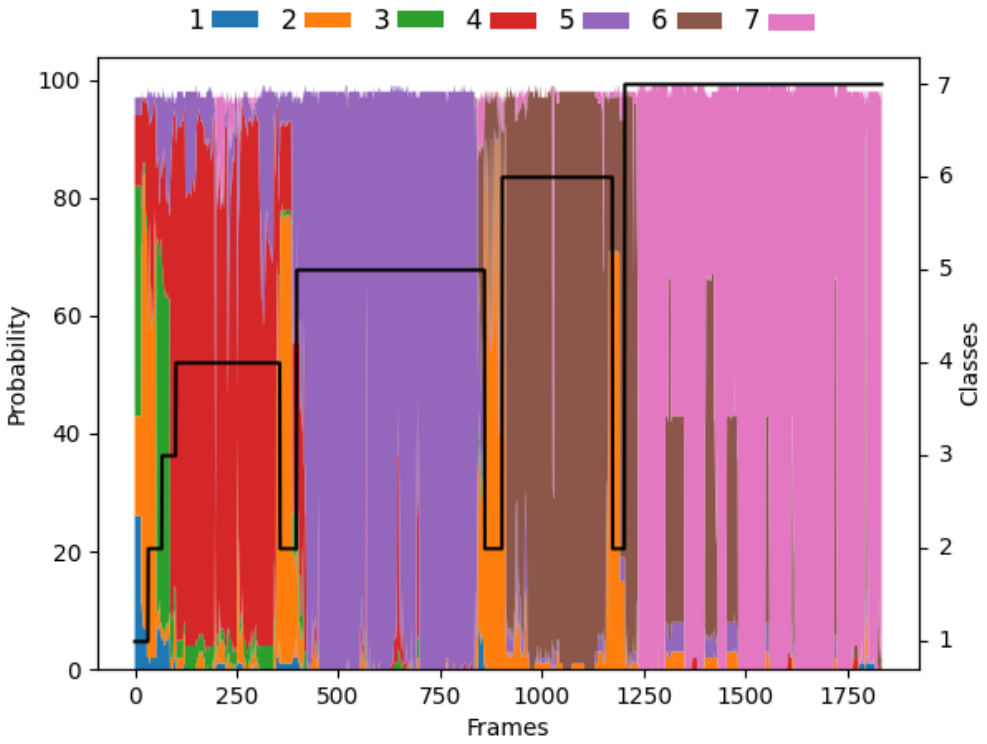


Figure 6.30: Accumulated class probabilities for 3S-LSTM over frames for CCD task

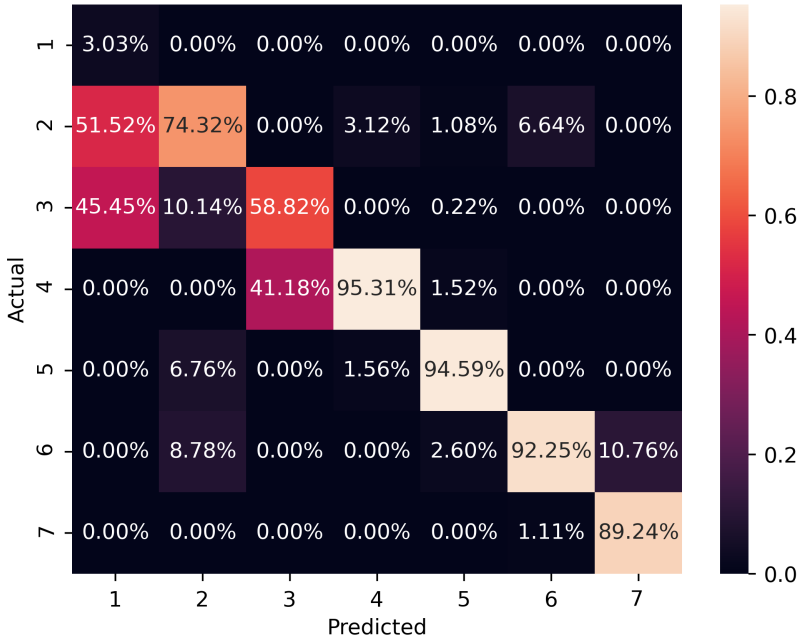


Figure 6.31: Confusion matrix of 3S-LSTM for CCD task

Action 5 represents the screwing on of the pin hinge in the left corner and is shown in violet. This action has the second highest TPR with 94.59% and is also the second largest with over 500 frames. A lot of false classification is happening towards the end of the action, before the handover. Also the wrong predictions are counting mostly towards Action 6 with 2.60%, while the worker is moving back from the corner position towards the middle of the plate.
Action 6 is depicted in brown and corresponds to the installation of the bottom hinge in the right corner. The TPR lies at 92.25%. In this case 6.64% of the false predictions are targeted to Action 2. Again near the end of the action towards the hand over the error increases. However in this class false predictions are happening throughout the action, with no other error clusters, except the ending.
Action 7 is depicted in pink with a TPR of 89.24% and has barely any false predicted frames, which are evenly distributed among the other classes. It is by far the class with the most frames involved, over 600, which can be seen in figure 6.30. Almost no frames of other actions are predicted as Action 7, which is a huge difference compared to the values of the confusion matrix **??**

for the 1S-LSTM results. This could be due to the objects gaining more influence, because of the 3S-LSTM model structure, since classifications towards the final action shouldn't occur, as Action 6 features all work pieces that were installed before. The only exception is the lock, which is occluded by the operators hands in the beginning of the action, which explains the FPR of 10.76% towards Action 6, where also just the other three work pieces occur. This relationship between Action 6 and Action 7 doesn't show up in the confusion matrix 6.31 for the 1S-LSTM results, where Action 6 has a FPR of 1.09% towards Action 7. Furthermore it is observable that the prediction probabilities shown in figure 6.30 are oscillating a lot less compared to the FNN or the 1S-LSTM, showing the robustness of the prediction probabilities for the 3S-LSTM model.
The classification success differs among the classes. In illustration **??** the multi class ROC curve is featured, as well as its AUC value by analyzing the false positive rate against the true positive rate. All classes are reaching an area over 0.97, which indicates that the model is has a strong classification capability. Also the curves for all classes are starting very steep and are reaching TPR values over 0.85 for FPR values less than 0.1 . However class one, five and two have values over 98%, which indicates that those classes contain some features are making it easier for the classification than the other classes. Likewise the ROC-curve for those three classes is stating very steep and reaching a true positive rate of 1 without having more than 0.1 of false positives. Class three on the other hand reaches over 0.9 TPR, with a FPR over 0.5. This indicates that class three is harder for the model to clearly separate compared to the other classes. Similar hard to classify is class zero which is always predicted with a FPR offset, before predicting the correct labels. Since class zero is the shortest class with just 10 frames on average, and occurs just in the beginning, only the location of the worker and its posing is taken into account.

6.2.4 Temporal Segment LSTM

The fourth experiment is conducted with a Temporal Segment LSTM network. This model consists of four separately trained LSTM neural networks. Each network is getting the same input features, but the dimensions of the input tensors are varying due to different window sizes. Each network delivers a vector with probabilities for each class for each frame as output. All models of the ensemble are predicting on unseen data individually and the average of

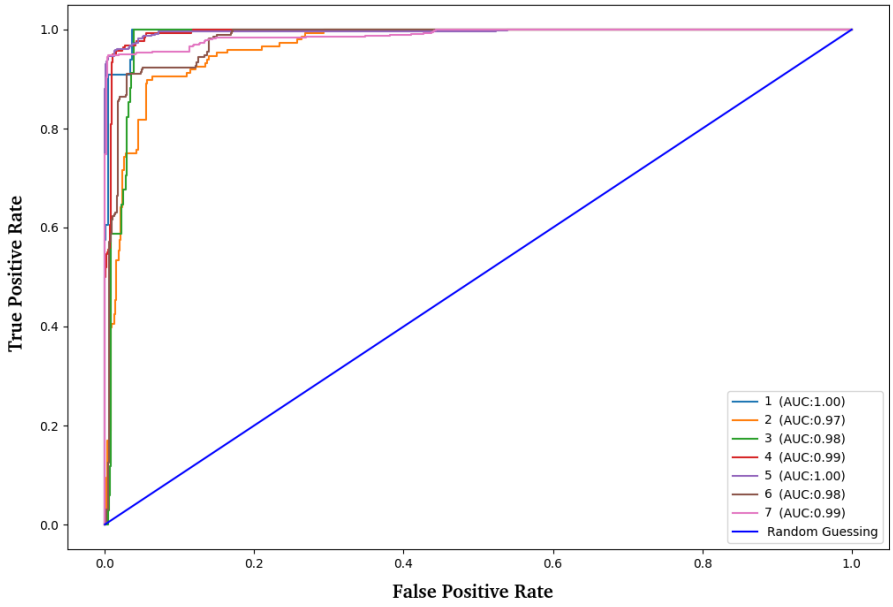


Figure 6.32: ROC curve of 3S-LSTM for CCD task

their predictions is chosen as the final result. The model architecture as well as the pre-processing of the input data is lined out in detail in section 5.4.4.

6.2.4.1 Training

The fourth experiment is conducted with a Temporal Segment LSTM network. This model consists of four separately trained 1S-LSTM networks. Each network is getting the same input features, but the dimensions of the input tensors are varying due to different window sizes. Each network delivers a vector with probabilities for each class for each frame as output. All models of the ensemble are predicting on unseen data individually and the average of their predictions is chosen as the final result. This is done in order to analyse the influence on short and long-term temporal components of different window sizes. However training of the network revealed that to big window sizes would lead to drastically worse performances. Therefore the shapes of the input tensors are $14896 \times 58 \times 7$, $14896 \times 58 \times 12$, $14896 \times 58 \times 20$ and $15332 \times 59 \times 25$. The model architecture as well as the pre-processing of the input data is lined out in detail in section 5.4.4.

6.2.4.2 Training

The hyperparameters can be observed in table 6.10. A dropout of 0.1 and batch size of 1024 are chosen through grid search. The same applies to the chosen ADAM optimizer with a learning rate of $1 \cdot 10^{-3}$, $1 \cdot 10^{5}$ decay steps and a decay rate of 0.9. The window sizes are chosen to 7, 12, 20 and 25 respectively. For the window size are higher values chosen compared to the ECH task, since the training performance isn't decreasing for higher values for the CCD task.

Batch Size	**Learning Rate**	**Decay Steps**	**Decay Rate**
512	$1 \cdot 10^{-3}$	$1 \cdot 10^{5}$	0.9

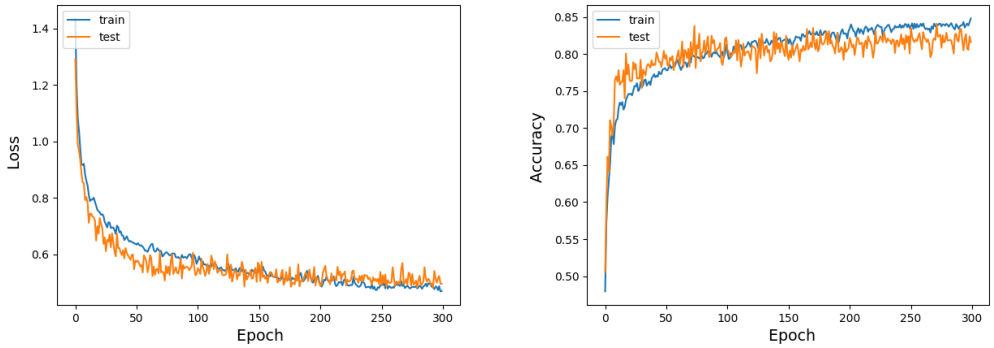
Optimizer	**Epochs**	**Dropout**	**Window Size**
ADAM	300	0.15	[7, 12, 20, 25]

Table 6.10: Hyperparameters for TS-LSTM network for CCD task

The network is trained for 300 epochs, since further epochs are leading to over fitting of the model. The progress of the accumulated model accuracy and model loss of all four trained models can be seen in image 6.33a and image 6.33b respectively. The training loss, as well as the validation loss are taking some time to decrease with around 100 epochs in order to fall under 0.1. The training stops at epoch 300 in order to prevent over fitting since the validation loss converges around 0.57 while the final training loss reaches a value of 0.55. The training accuracy of the model reaches a value over 70.5% after 35 epochs and converges to an training error of 84.86% at the end of the training. The validation loss shows a similar behaviour and converges to an final accuracy of 82.15% while the strong initial oscillating behaviour is ebbing down.

6.2.4.3 Results

Testing the model on unseen data is yielding the following results. Out of 1849 test frames, 1564 are classified as true positive values. This is a ratio of 84.58% classification accuracy. Image 6.34 shows a visualization of class probabilities per frame in comparison to the actual label, which is presented as black line. The corresponding colors for each class can be seen in the legend. Action 1



(a) Training loss (b) Training accuracy

Figure 6.33: Training for TS-LSTM for CCD task

is depicted in blue and is predicted with 3.03%, which can be observed in the confusion matrix 6.35. It shows the ratio between actual label and prediction of each class. A huge chunk of the FPR is predicted as Action 7 with 84.85%, which corresponds to the installation of the lock and a FPR of 12.12% is predicted fo the hand over action. Both classes are featuring the operators pose centered in the scene while encompassing way more frames than Action 1.
Action 2, presented in orange, corresponds to the handover action. 90.54% of all predictions for class one are true, which is the highest prediction value for Action 2 over all models, with an improvement over 10%. and 9.46% of the false predictions can be found in Action 7. This high accuracy can be explained with the distinctive movement of the handover, with reaching out an arm to the side of the scene. Furthermore the person is standing up straight compared to other actions. However while investigating 6.34, it is also observable that the false positives are occurring at the start or ending of a handover, showing the transition between actions is harder to classify. It is also observable that more false classifications are occurring at the end of an action towards a handover than from handover towards next action.

Action 3 is shown in green and represents the placement of the white ring. It is predicted with a TPR of 61.76%. False predictions are made for Action 7 with 23.53% and Action 4 with 14.71% respectively. This class prediction also shows the errors during the transitions with its subsequent action. Again, the wrong predictions towards Action 7 might be influenced by straight standing positioning in the middle of the scene.

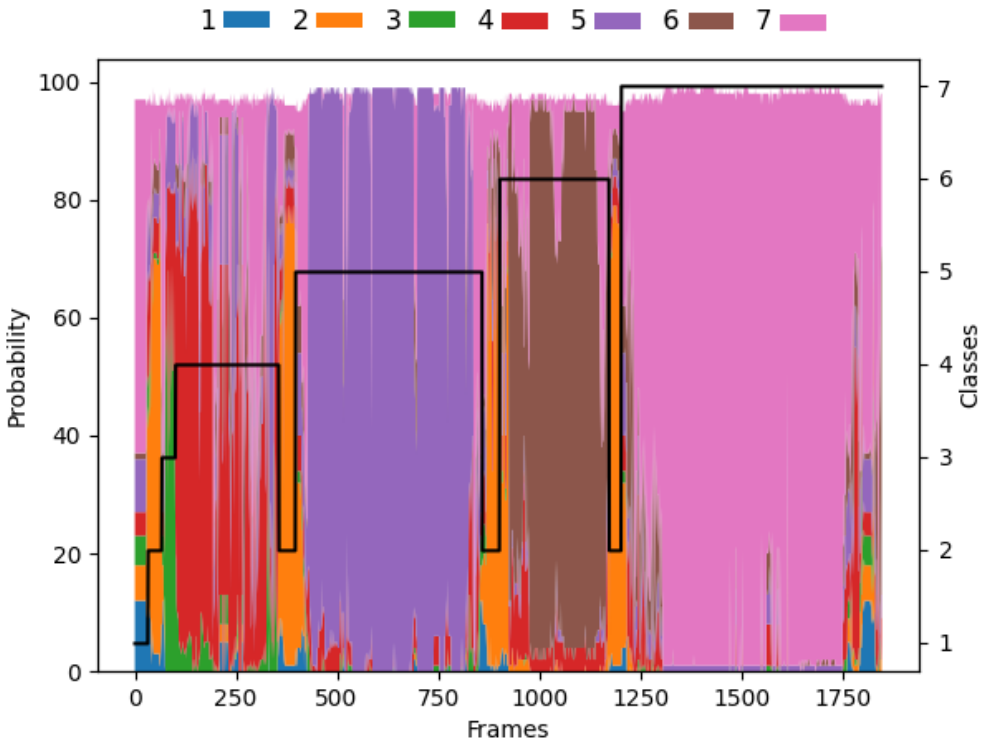


Figure 6.34: Accumulated class probabilities for 3S-LSTM over frames for CCD task

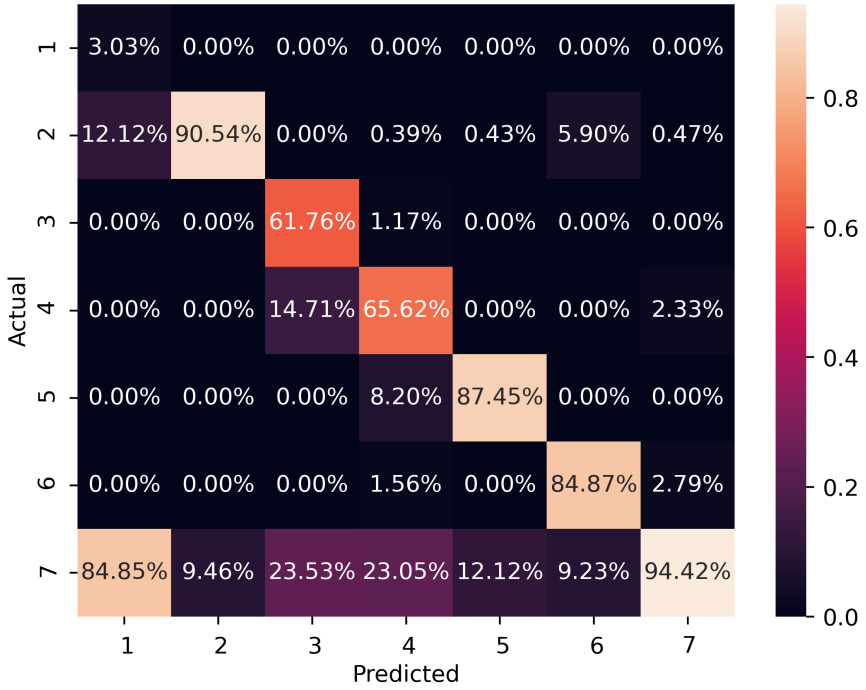


Figure 6.35: Confusion matrix of TS-LSTM for CCD task

In red is Action 4 depicted, which translates to the pulling of the metal plate on top of the cabinet door frame. This action has a TPR of 65.62%. It is the only action that is not started by a handover action. However very few FPR are found in class two with 1.17%, showing that the transition between those two actions is fairly smooth. However as can be seen in illustration 6.34 after the initial high classification accuracy of around 80% drops to below 40% in the end, which is a similar behaviour that can be observed in the results of the other models for the CCD task.

Action 5 relates to the installation of the pin hinge in the left corner and is marked in violet. This action has the third highest TPR with 87.45% and is also the second largest with over 500 frames. A lot of false classifications are happening towards the end of the action, before the handover. Also the wrong predictions are counting mostly towards Action 7 with a FPR of 12.12%, while the operator is moving back from the corner position towards the middle of the plate. The accuracy is stable throughout this stage, only frames close to the hand over phases are losing accuracy.

Action 6 is shown in brown, which corresponds to the installation of the bottom hinge in the right corner. The TPR lies at 84.87%. Again in this Action 7 is encapsulating falsely predicted labels, with around 9.23%. Also near the end of the action towards the hand over the classification accuracy decreases. Action 7 is depicted in pink with a TPR of 94.42%. This class encompasses the most amount of correctly classified frames and has barely any false predicted frames. For the TS-LSTM the majority of wrongly predicted label, where classified as Action 7, which is similar to the results shown in the confusion matrix 6.27 for 1S-LSTM and choosing different window sizes doesn't change that behaviour. However Action 2 and Action 3 do have higher classification accuracies compared to the 1S-LSTM and the 3S-LSTM, which indicates that smaller classes do profit from different window sizes.
The classification capability of the model differs among the classes. In illustration 6.36 the multi class ROC curve is featured, as well as its AUC value by analyzing the false positive rate against the true positive rate. All classes are reaching an area over 0.97%, which indicates that the model is indeed capable of classifying each action. All curve are starting with a steep slope and are reaching a TPR of 0.8 for a FPR pf 0.1 or less. Again Action 3 has the most shallow slope out of all curves and the curve for Action 4 has the highest FPR with over 0.6, when it reaches the the TPR of one. This again shows the confusion of both actions during the classification process of the model.

6.3 Ablation Study

Now the influence of different sensor inputs on the classification quality of each model architecture is tested. Since there are three different feature sources - 2D joint coordinates, depth values and object probabilities - being present in the frame, it is also feasible to check, which source contributes in which capacity to the classification quality. For each task and model the following configurations are tested: solely the 2D joint skeleton data, the skeleton data in combination with the depth values as 3D data, the 2D skeleton data in combination with the object probabilities and all three feature modalities combined. Since the models are trained on small data sets, the 2D joint coordinates are tested isolated. They offer 36 features, compared to the 18 features of the depth value and the four or five features of the detected objects respectively.
The testing accuracy for each model, trained for each data set, can be seen in

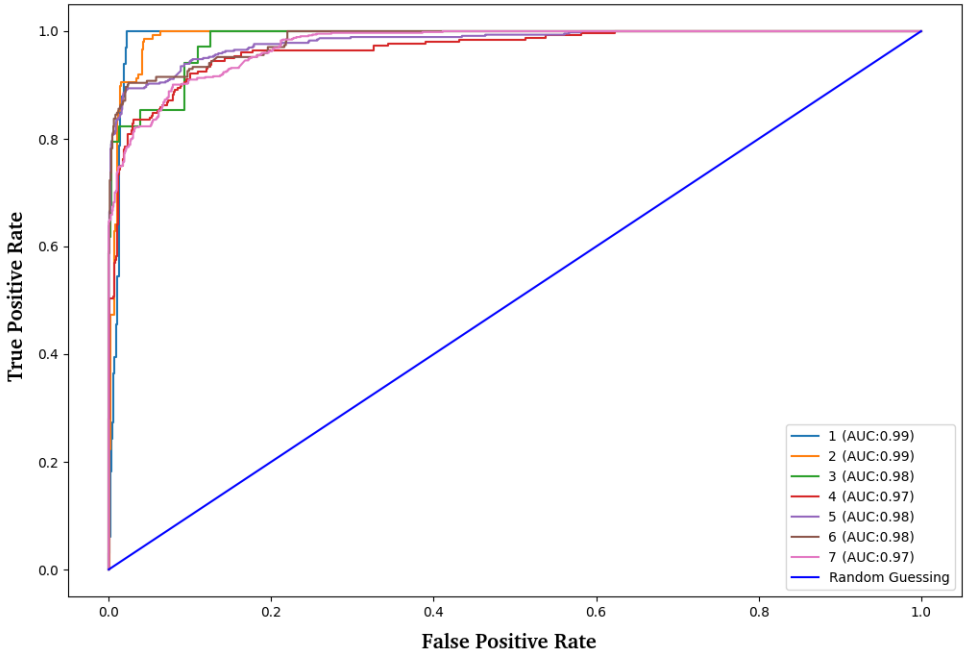


Figure 6.36: ROC curve of 3S-LSTM for CCD task

the table 6.11 and table 6.12 respectively. For each task the same corresponding tests set are used. All models are trained with the aforementioned input configurations, except for the 3S-LSTM, since the 2D input alone is the same for a 1S-LSTM and 3S-LSTM.

Elevator Cabin Hangar				
	2D	**3D**	**2D+Objects**	**3D+Objects**
Neural Network	47.51%	55.60%	57.98%	65.70%
1S-LSTM	53.31%	56.27%	68.59%	71.26%
3S-LSTM	-	64.23%	66.94%	75.68%
TS-LSTM	60.13%	62.73%	66.81%	69.04%

Table 6.11: Classification accuracy for the elevator cabin hangar assembly task for each trained model and each feature configuration

For the ECH task it is overall observable that more features are leading to more testing accuracy by 14% on average across all models from 2D coordinates to 3D coordinates with object probabilities. Also adding object probabilities to the 2D skeleton joints leads to a performance improvement of 2-3% over adding a third dimension to the joint coordinates. In general the neural network performs worse than the LSTM based models, with a difference of almost 10% for the 3S-LSTM with all features. With a value of 75.68% it achieves the highest prediction accuracy while trained on all features. This shows the importance of the ability to recognize temporal dependencies for assembly tasks. This is also underlined by the influence of the detected objects and depth values. The LSTM based model gained more testing accuracy by the involvement of the object detection compared to the feed forward neural network, between 6% and 15%. Especially the 1S-LSTM gains almost 15% accuracy for 2D joint coordinates and objects as input features compared to just 2D joint coordinates as input. However the biggest improvement occurs for all models, when being trained on all features combined. Especially the 1S-LSTM and FNN gain almost 20% performance improvement, while the TS-LSTM performance increases around 9%.
Overall table 6.12 shows that the CCD task yields a higher prediction accuracy

Control Cabinet Door				
	2D	**3D**	**2D+Objects**	**3D+Objects**
Neural Network	65.06%	70.57%	71.33%	72.09 %
1S-LSTM	80.17%	80.98%	82.24%	83.61%
3S-LSTM	-	80.46%	82.32%	88.56%
TS-LSTM	80.53%	82.09%	83.45%	84.58%

Table 6.12: Classification accuracy for the control cabinet door assembly task for each trained model and each feature configuration

for all configurations compared to the ECH task, even though the training set was just half as big. However the relations between different models and input configurations are similar. The FNN performs worse than the LSTM based models, but just by over 10% for most input configurations. Additionally does the 3S-LSTM outperform all other models for all input configurations by over 10% for the FNN and around 4% for the 1S-LSTM and the TS-LSTM. Also the performances for only 2D coordinates as input are already around 20% compared to the results for training the models on the EHC data set with only the 2D as input features. Adding depths values or object probabilities does increase the model performance, but just by 5% and 6% for the FNN and around 1% and 2 % for the LSTM base values respectively. Choosing all features as input for the training yields the biggest performance improvement for the 3S-LSTM with around 8%.

6.4 Feature Importance

In order to be able to classify actions correctly, it is important to know how an action is recognized. More specifically it is of interest how a neural network is ranking the features based on how important they are for a classification of an action. To this purpose the ELI5 library [1] is explained in depth in section 5.5.3 and is used to evaluate the the feature importance for the most successful model for both assembly setups. In both cases this is the 3S-LSTM according to the results shown in section 6 and ablation study 6.3. After training the 3S-LSTM with ELI5, a list of importance weights for each feature is provided, which are drawn to given frames in the shape of red circles. They are placed on the location of the joints on top of the extracted skeleton with the higher

[1] https://eli5.readthedocs.io/en/latest/autodocs/keras.html

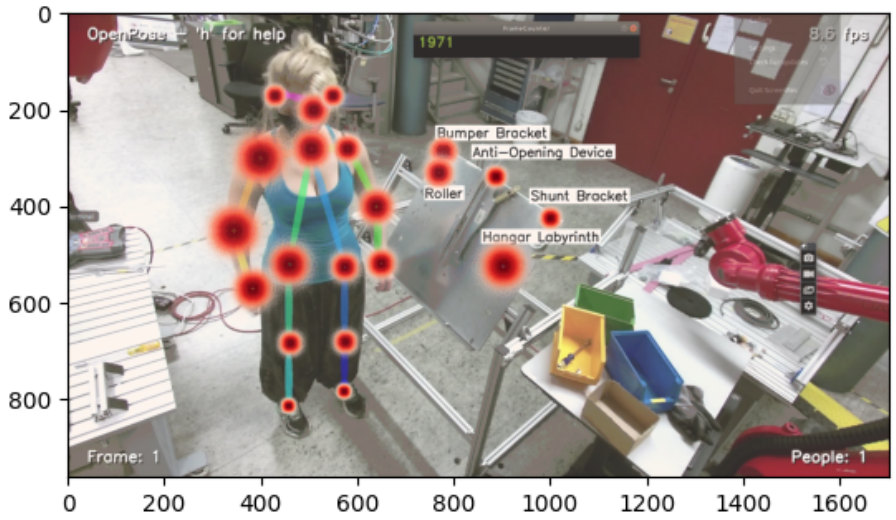


Figure 6.37: Heat map displaying joint and object importance for ECH task

weights increasing the radius of the circle, as well as the objects needed for the assembly process, which are additionally marked with bounding boxes. The heat map for the elevator assembly task is shown in image 6.37. It is observable, that joints on the right side of the body have a higher influence on the performance than joints on the left side. Especially the right shoulder and hand are more involved, since the operators are installing the work pieces mainly with their right hand and holding it in position with the left hand. Also the handover is done predominantly with the right hand. Also the roller and hangar labyrinth are reaching high ELI5 weights, since they are attached very early and detected very easily. The bumper bracket and shunt bracket however, are harder to detect and prone to occlusion, thus diminishing their influence on the model capability. The anti-opening device has more impact compared to the bracket and shunt bracket, but the placement at the very end of the assembly task gives and indication why it has a lower influence compared to roller or labyrinth and a lot of joints, because it is present in way fewer frames. The heat map for the control cabinet door task is shown in image 6.38. Likewise

for the first heatmap, a bigger diameter of the red heat circles, are indicating a bigger influence of the feature for the model performance. There are similarities to the elevator assembly task, especially with the arm and shoulder joints contributing heavily towards the classification performance. However for this setup the left elbow and wrist do have a higher influence compared to their right sided counterparts. Also neck and nose are very involved. The feet and legs do have a higher influence in this scenario, since the operator has to move around the work table more often. On top of that the used work pieces aren't as influential as in the elevator task. For the ECH task, the operator executes most actions on a locally narrow working space compared to the CCD task, which reduces the wight of the 2D coordinates. However all of the tools are similar in importance, which is on one hand due to occurrences of occlusion. Furthermore the number of frames for the different classes that are involved are increasing towards the end of the scenario. This means the installation of the locker as last work piece is included in the action with the most amount of frames for the cabin door scenario, while it was the other way around in the elevator scenario, where the installation of the rollers at the beginning are involved in the action with the most amount of frames. This shows that an uneven distribution of features is making it harder for the model to use them for a better classification performance.
In both cases there is a huge impact by the information delivered of the joints related to the arms, neck, hips and nose and therefor on the torso related part of the body. Head joints like ears and eyes or the joint related to lower body parts like knees and ankles. Detected objects do have an influence on the model performance, however this depends on the amount of frames they occur in and doesn't reach the same importance like joint related to the torso. This shows how important the spatial information is, compared to the sequential nature of the actions or the occurrence of certain work pieces.

6.5 Experiments Discussion

In this section the results and findings of the experiments for the elevator task 5.1.1 and the control task 5.1.2, as well as the ablation study 6.3 are discussed in terms of the applicability for human action recognition. It is observable that the actions belonging to elevator task are harder to classify correctly by a difference of 12% compared to the actions of the control task, when comparing

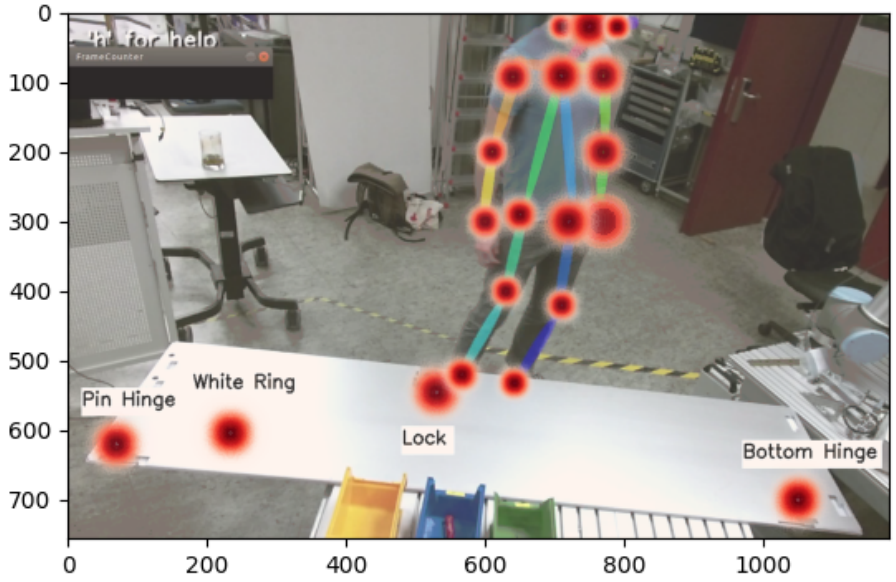


Figure 6.38: Heat map displaying joint and object importance for CCS task

the model performances of the tables 6.11 and 6.12. This is surprising since almost double the amount of recordings for the elevator task are used in the training process compared to the control task. Analyzing the ROC curves, confusion matrices and distribution plots of the probability predictions in the corresponding result section shows, that the models have a harder time to differentiate between actions of the elevator task compared to actions of the control task. The average AUC for all classes and models for the elevator task is 0.94, while the average AUC for the control task over all classes and models reaches 0.97. This is due to the nature of the set up of the first assembly task, where in a lot of frames the assembled work pieces are corresponding to actions that are executed in locally similar places and their limited variety in the skeleton pose. This is also underlined by the results of section 6.1. The actions of installing the hanger labyrinth and shunt bracket are very hard to distinguish for the models and show overlapping prediction probabilities, because the locations for executing those tasks lie very close together and are executed successively. This limits the influence of the spatial and temporal information for the classification success. Furthermore the ablation study reveals that adding additional features is beneficial for the classification accuracy for both assembly tasks. However the ECH task does gain a lot more accuracy by adding further information, especially the detected objects. On average all trained models are showing an increasing accuracy of 11% between 2D joints as input and 2D joints and objects as input for the ECH task and 9% for the CCD task. For the ECH task additional information of the detected objects for those stages are enhancing the classification accuracy and therefore support classification certainty where the variety in joint poses are sparse. The influence of the depth information also enhances the results, with a gain of 9% for the first task and 6% for the second task. The smaller improvement for adding the depth value can be explained with the setup of the tasks, which requires the operators to not change their distance towards the RGB-D camera much and offers more movement on the image plane of the RGB part from the camera. However it is observable that for the CCD task those benefits aren't as high, since the classification performance for just 2D coordinates is already starting with a high value. Furthermore the probability distributions for both tasks are showing the problematic nature of the transition from one action to another. This is observed in the probability density plots over all models and tasks, where most falsely predicted frames are occurring during the beginning

and ending of the actions. furthermore the chosen models showed a similar performances distribution during both experiments. For both assembly tasks the neural networks had the lowest classification accuracy and were outperformed by all LSTM based models. This shows the importance of the temporal information on top of the spatial information. Varying the windows size didn't show an enhanced performance of the TS-LSTM compared to the 1S and 3S-LSTM. Instead it even performs worse as the 1S-LSTM for the ECH task. In this case the initial windows size had already to be chosen as three, which is smaller than the initial window size of 12 for the second task. This shows that a smaller window size in general and tailored for the amount of frames belonging to each action, is more important for the classification success, instead of longer sequences that contain more frames of different actions. This is more severe for the elevator task, since it contains more actions with fewer frames like the final position on top of the starting position and handover actions. Additionally classes with fewer frames are detected well, if they contain distinct movements of the operators. The handover actions are detected with an accuracy of over 80% for both task on average, while actions containing more frames like the attachment of the shunt bracket or pulling of the metal plate achieved lower accuracies with around 50% or 15% respectively. However actions which are too short like the starting position, are sometimes not detected at all. The 3S-LSTM shows the highest classification accuracy for both tasks with 75.68% and 88.56% respectively. Also the results of the ablation study are showing higher results for each category. While the joint coordinates are present in each frame, the different objects are just bit introduced bit by bit throughout the scenario and therefore featured in fewer frames. On top of that 36 2D coordinates with a variance of and for the corresponding tasks, compared to 4 to 5 tools with a variance of and for the corresponding tasks, might be favored by the network for the training process. Therefore training fewer features like the objects on a separate model forces the model to make sense of the features provided by the objects. The same goes for the depth information, which have a way lower variance for both scenarios and provides just 18 features per frame compared to 36.

7 Conclusion

In this thesis an implementation of a light weight human action recognition framework for industrial assembly task was presented, in order to increase the performance of human-robot-collaboration in manufacturing settings. To this purpose two different industrial assembly task setups were created and recorded with the RGB-D camera Kinect v2. The recorded RGB frames and depth frames were subsequently used for feature extraction. The RGB frames were picked up by Openpose, which provided 36 2D skeleton joint coordinates and Yolov4 for object detection, that delivered class probabilities for the used work pieces. The information of the depth frames delivered a third dimension to the skeleton joint coordinates. In order to use those features for classification of the class actions, four different network models were designed and implemented. A feed forward neural network, a single Stream LSTM, a three stream LSTM and a temporal segment LSTM were implemented, in order to capture spatial and temporal dependencies for the classification of the actions and allowing analysis of important features due to different model architectures. The results are showing that the classification actions for industrial assembly related tasks is achievable with a high accuracy. The best model performance for the assembly of the elevator cabin hanger reached an accuracy value of 75.68% and for the assembly of the control cabinet door an accuracy value of 88.56% was achieved. In both cases the three stream LSTM model outperformed the other architectures, showing that training on each feature individually (2D joint pose, depth coordinates and object probabilities) enhances the classification success. Furthermore the ablation study revealed that increasing the amount of features leads to a higher classification accuracy. This means that across all models 2D joint coordinates combined with either depth information or object probabilities are showing an increased performance by a maximum value of 15% for the elevator assembly task and 6% for the control cabinet door task. Since both task setups differ, a comparison of both revealed that lower spatial variety of the operators pose makes it harder for the models to differentiate

the actions. The deficit of spatial information is especially compensated by the information of the detected objects, with an performance increase of roughly 11% for the first task and 9% for the second task. The influence of the depth information also enhanced the results, with a gain of 9% for the first task and 6% for the second task. Also including temporal information provides a better classification performance, with the FFN performing 10% and 16% worse than the three stream LSTM for the first and second assembly task respectively. However some limitations have to be considered. The chosen tasks are dependent on the execution order of the actions with the execution time being dependent on the operator, while tasks with arbitrary action order might not be predicted as well. The class labels were done by hand, making it harder to predict the correct class during transitions. Also class imbalance causes very short action to be harder to predict, as well as classes that face similar movements.

7.1 Future Work

In this section further improvements to the HAR framework, as well as further aspects for analysis of human action recognition are presented. Issues like sparse kinematic representations, uncertainties during transitions and more diverse task setups with more participants need to be addressed. Therefore in the following the classification of transitioning periods and multi-class label, focusing on kinematic features that are involving especially the hand joints, and finally action recognition for assembly tasks that are involving more than one human.

7.1.1 Multi-class Label

Since assembly tasks are activities build as sequence of actions that are executed in succession, ambiguity of the classification during the transition periods between actions is expected. To this purpose multi-class labelling could help. Frames that show a high classification uncertainty between different actions could be labelled by classes of both of those actions. Furthermore unsupervised machine learning methods could be used for clustering each frame instead of labelling each frame by hand.

7.1.2 Hand joints

The focus of this work was on human action recognition with analyzing two main components of what defines an action - the pose of the involved human, as well as the objects present in the scene. However, as seen in the results of the presented experiments there were two issues highlighted. First for industrial assembly tasks a lot of information is retrieved by the kinetic movement of the torso and arms which was highlighted by the evaluation of the feature importance 6.4. Secondly some assembly tasks require the operator to execute several actions in a similar location. This means the pose of the operator given by the COCO format doesn't provide a lot of variety and information for the recognition of the executed action. However all of those actions involved different hand and fine finger motions. Therefore further investigation of the different joints of the hands and fingers could lead to a more precise classification. This includes spatial as well as temporal information. Openpose already features a module for detecting hand key points on top of the COCO-format [72].

7.1.3 Multi-operator collaborative assembly

Some assembly projects require several operators in order to distribute the workload or to shoulder heavy tasks. This means the poses of interactions between several humans can also deliver valuable information for classification of the involved actions. Skeleton detection for multiple people is already included in the Openpose framework [8], which makes it easy for the presented HAR framework to be extended for this purpose.

Bibliography

[1] Iina Aaltonen, Timo Salmi, and Ilari Marstio. "Refining levels of collaboration to support the design and evaluation of human-robot interaction in the manufacturing industry". In: May 2018. DOI: 10.1016/j.procir.2018.03.214.

[2] J.K. Aggarwal and M.S. Ryoo. "Human Activity Analysis: A Review". In: *ACM Comput. Surv.* 43.3 (Apr. 2011). ISSN: 0360-0300. DOI: 10.1145/1922649.1922653. URL: https://doi.org/10.1145/1922649.1922653.

[3] Janis Arents et al. "Human-Robot Collaboration Trends and Safety Aspects: A Systematic Review". In: *Journal of Sensor and Actuator Networks* 10 (July 2021). DOI: 10.3390/jsan10030048.

[4] Tadele Belay Tuli, Valay Mukesh Patel, and Martin Manns. "Industrial Human Activity Prediction and Detection Using Sequential Memory Networks". In: *Proceedings of the Conference on Production Systems and Logistics: CPSL 2022.* 18.18 (2022), pp. 62–72. DOI: 10.1109/JSEN.2018.2859268.

[5] Estela Bicho et al. "The Power of Prediction : Robots that Read Intentions". In: Oct. 2012. DOI: 10.1109/IROS.2012.6386297.

[6] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. "YOLOv4: Optimal Speed and Accuracy of Object Detection". In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[7] Jason Brownlee. *Understanding LSTM Networks.* 2015. URL: https://machine%20learningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/ (visited on 05/20/2020).

[8] Zhe Cao et al. "OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields". In: *CoRR* abs/1812.08008 (2018). arXiv: 1812.08008. URL: http://arxiv.org/abs/1812.08008.

[9] Alexandros Andre Chaaraoui et al. "Evolutionary joint selection to improve human action recognition with RGB-D devices". In: *Expert Systems with Applications* 41.3 (2014). Methods and Applications of Artificial and Computational Intelligence, pp. 786–794. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2013.08.009. URL: https://www.sciencedirect.com/science/article/pii/S0957417413006210.

[10] Lei Chen et al. "A Method of Human Activity Recognition in Transitional Period". In: *Information* 11 (Aug. 2020), p. 416. DOI: 10.3390/info11090416.

[11] Wen-Hui Chen, Carlos Andrés Betancourt Baca, and Chih-Hao Tou. "LSTM-RNNs combined with scene information for human activity recognition". In: *2017 IEEE 19th International Conference on e-Health Networking, Applications and Services (Healthcom)*. 2017, pp. 1–6. DOI: 10.1109/HealthCom.2017.8210846.

[12] Marc Claesen and Bart De Moor. "Hyperparameter Search in Machine Learning". In: *CoRR* abs/1502.02127 (2015). arXiv: 1502.02127. URL: http://arxiv.org/abs/1502.02127.

[13] Wikimedia Commons. *File:MultiLayerNeuralNetwork english.png — Wikimedia Commons, the free media repository*. [Online; accessed 11-September-2022]. 2020. URL: https://commons.wikimedia.org/w/index.php?title=File:MultiLayerNeuralNetwork_english.png&oldid=481159061.

[14] Wikimedia Commons. *File:Recurrent neural network unfold.svg — Wikimedia Commons, the free media repository*. [Online; accessed 09-September-2022]. 2022. URL: %5Curl%7Bhttps://commons.wikimedia.org/w/index.php?title=File:Recurrent_neural_network_unfold.svg&oldid=655242383%7D.

[15] Wikimedia Commons. *File:Time of flight.svg — Wikimedia Commons, the free media repository*. [Online; accessed 01-September-2022]. 2022. URL: %5Curl%7Bhttps://en.wikipedia.org/wiki/File:20200501_Time_of_flight.svg%7D.

[16] Mejdi DALLEL et al. "InHARD - Industrial Human Action Recognition Dataset in the Context of Industrial Collaborative Robotics". In: *2020 IEEE International Conference on Human-Machine Systems (ICHMS)*. 2020, pp. 1–6. DOI: 10.1109/ICHMS49158.2020.9209531.

[17] Jeff Donahue et al. "Long-term Recurrent Convolutional Networks for Visual Recognition and Description". In: *CoRR* abs/1411.4389 (2014). arXiv: 1411.4389. URL: http://arxiv.org/abs/1411.4389.

[18] Dheeru Dua and Casey Graff. *UCI Machine Learning Repository*. 2017. URL: http://archive.ics.uci.edu/ml.

[19] Yogesh K. Dwivedi et al. "Climate change and COP26: Are digital technologies and information management part of the problem or the solution? An editorial reflection and call to action". In: *International Journal of Information Management* 63 (2022), p. 102456. ISSN: 0268-4012. DOI: https://doi.org/10.1016/j.ijinfomgt.2021.102456. URL: https://www.sciencedirect.com/science/article/pii/S0268401221001493.

[20] Florian Echtler et al. *OpenKinect*. 2015. URL: https://github.com/OpenKinect/libfreenect2 (visited on 06/11/2021).

[21] Bernard Ghanem Fabian Caba Heilbron Victor Escorcia and Juan Carlos Niebles. "ActivityNet: A Large-Scale Video Benchmark for Human Activity Understanding". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 961–970.

[22] Maël Fabien. *How to use OpenPose on macOS ?* [Online; accessed 11-September-2022]. 2019. URL: %5Curl%7Bhttps://maelfabien.github.io/project/open-pose_proj/#installation%7D.

[23] María García-Ordás et al. "Detecting Respiratory Pathologies Using Convolutional Neural Networks and Variational Autoencoders for Unbalancing Data". In: *Sensors* 20 (Feb. 2020). DOI: 10.3390/s20041214.

[24] Felix Gers, Nicol Schraudolph, and Jürgen Schmidhuber. "Learning Precise Timing with LSTM Recurrent Networks". In: *Journal of Machine Learning Research* 3 (Jan. 2002), pp. 115–143. DOI: 10.1162/153244303768966139.

[25] Negar Golestani and Mahta Moghaddam. "Human activity recognition using magnetic induction-based motion signals and deep recurrent neural networks". In: *Nature Communications* 11 (Mar. 2020). DOI: 10.1038/s41467-020-15086-2.

[26] Klaus Greff et al. "LSTM: A Search Space Odyssey". In: *IEEE Transactions on Neural Networks and Learning Systems* 28.10 (Oct. 2017), pp. 2222–2232. ISSN: 2162-2388. DOI: 10.1109/tnnls.2016.2582924. URL: http://dx.doi.org/10.1109/TNNLS.2016.2582924.

[27] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-term Memory". In: *Neural computation* 9 (Dec. 1997), pp. 1735–80. DOI: 10.1162/neco.1997.9.8.1735.

[28] Chaoxing Huang. "Event-based Action Recognition Using Timestamp Image Encoding Network". In: *CoRR* abs/2009.13049 (2020). arXiv: 2009.13049. URL: https://arxiv.org/abs/2009.13049.

[29] Wenbo Huang et al. "Shallow Convolutional Neural Networks for Human Activity Recognition Using Wearable Sensors". In: *IEEE Transactions on Instrumentation and Measurement* 70 (2021), pp. 1–11. DOI: 10.1109/TIM.2021.3091990.

[30] Zawar Hussain, Michael Sheng, and Wei Emma Zhang. "Different Approaches for Human Activity Recognition: A Survey". In: *CoRR* abs/1906.05074 (2019). arXiv: 1906.05074. URL: http://arxiv.org/abs/1906.05074.

[31] Javed Imran and Praveen Kumar. "Human action recognition using RGB-D sensor and deep convolutional neural networks". In: *2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*. 2016, pp. 144–148. DOI: 10.1109/ICACCI.2016.7732038.

[32] Roboflow Inc.) *Roboflow*. 2022. URL: https://https://blog.roboflow.com/ (visited on 02/25/2022).

[33] Francesco Iodice, Elena De Momi, and Arash Ajoudani. *HRI30: An Action Recognition Dataset for Industrial Human-Robot Interaction*. Version 1.0. Zenodo, Jan. 2022. DOI: 10.5281/zenodo.5833411. URL: https://doi.org/10.5281/zenodo.5833411.

[34] JavaTpoint. *Single Layer Perceptron in TensorFlow*. 2017. URL: https://www.javatpoint.com/single-layer-perceptron-in-tensorflow.

[35] Shuiwang Ji et al. "3D Convolutional Neural Networks for Human Action Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.1 (2013), pp. 221–231. DOI: 10.1109/TPAMI.2012.59.

[36] Bhavika Kanani. *Mathematics behind the Neural Network*. 2019. URL: https://studymachinelearning.com/mathematics-behind-the-neural-network/.

[37] Kyra Kerz. *Project Title*. https://github.com/InwoongLee/TS-LSTM. 2021.

[38] Nitish Shirish Keskar et al. “On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima”. In: *CoRR* abs/1609.04836 (2016). arXiv: 1609.04836. URL: http://arxiv.org/abs/1609.04836.

[39] Changhee Kim et al. “Color and Depth Image Correspondence for Kinect v2”. In: *Lecture Notes in Electrical Engineering* 352 (Jan. 2015), pp. 111–116. DOI: 10.1007/978-3-662-47487-7_17.

[40] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[41] Mikhail Korobov and Konstantin Lopuhin. *ELI5's documentation*. 2017. URL: https://eli5.readthedocs.io/en/latest/index.html (visited on 03/22/2022).

[42] Volker Krueger et al. “The meaning of action: A review on action recognition and mapping”. In: *Advanced Robotics* 21 (Sept. 2007), pp. 1473–1501. DOI: 10.1163/156855307782148578.

[43] H. Kuehne et al. “HMDB: A large video database for human motion recognition”. In: *2011 International Conference on Computer Vision*. 2011, pp. 2556–2563. DOI: 10.1109/ICCV.2011.6126543.

[44] Harold W. Kuhn. “The Hungarian Method for the Assignment Problem”. In: *Naval Research Logistics Quarterly* 2.1–2 (Mar. 1955), pp. 83–97. DOI: 10.1002/nav.3800020109.

[45] Thomas C.W. Landgrebe and Robert P.W. Duin. “Approximating the multiclass ROC by pairwise analysis”. In: *Pattern Recognition Letters* 28.13 (2007), pp. 1747–1758. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2007.05.001. URL: https://www.sciencedirect.com/science/article/pii/S016786550700150X.

[46] Viet-Tuan Le et al. “A Comprehensive Review of Recent Deep Learning Techniques for Human Activity Recognition”. In: *Intell. Neuroscience* 2022 (Jan. 2022). ISSN: 1687-5265. URL: https://doi.org/10.1155/2022/8323962.

[47] Inwoong Lee et al. “Ensemble Deep Learning for Skeleton-Based Action Recognition Using Temporal Sliding LSTM Networks”. In: Oct. 2017. DOI: 10.1109/ICCV.2017.115.

[48] Jiwon Lee et al. "Soccer object motion recognition based on 3D convolutional neural networks". In: Sept. 2018, pp. 129–134. DOI: 10.15439/2018F48.

[49] Yanfen Li et al. "A Deep Learning-Based Hybrid Framework for Object Detection and Recognition in Autonomous Driving". In: *IEEE Access* 8 (Oct. 2020). DOI: 10.1109/ACCESS.2020.3033289.

[50] Ruijie Liang et al. "A Three-Stream CNN-LSTM Network for Automatic Modulation Classification". In: *2021 13th International Conference on Wireless Communications and Signal Processing (WCSP)*. 2021, pp. 1–5. DOI: 10.1109/WCSP52459.2021.9613477.

[51] Ye Liu et al. "Action2Activity: Recognizing Complex Activities from Sensor Data". In: *CoRR* abs/1611.01872 (2016). arXiv: 1611.01872. URL: http://arxiv.org/abs/1611.01872.

[52] Yulong Lu and Jianfeng Lu. *A Universal Approximation Theorem of Deep Neural Networks for Expressing Distributions*. 2020. arXiv: 2004.08867 [cs.LG].

[53] DALLEL Mejdi et al. "An Industrial Human Action Recogniton Dataset in the Context of Industrial Collaborative Robotics". In: *IEEE International Conference on Human-Machine Systems ICHMS*. Apr. 2020. URL: https://github.com/vhavard/InHARD.

[54] Jeison Mejia-Trujillo et al. "Kinect™ and Intel RealSense™ D435 comparison: a preliminary study for motion analysis". In: Oct. 2019, pp. 1–4. DOI: 10.1109/HealthCom46333.2019.9009433.

[55] S. Nazir, M. H. Yousaf, and S. A. Velastin. "Inter and Intra class correlation analysis (IIcCA) for human action recognition in realistic scenarios". In: *8th International Conference of Pattern Recognition Systems (ICPRS 2017)*. 2017, pp. 1–6. DOI: 10.1049/cp.2017.0149.

[56] Christopher Olah. *Understanding LSTM Networks*. 2015. URL: https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (visited on 03/16/2020).

[57] Razvan Pascanu et al. "How to Construct Deep Recurrent Neural Networks". In: (Dec. 2013).

[58] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[59] Kari Pulli et al. "Realtime Computer Vision with OpenCV: Mobile Computer-Vision Technology Will Soon Become as Ubiquitous as Touch Interfaces." In: *Queue* 10.4 (Apr. 2012), pp. 40–56. ISSN: 1542-7730. DOI: 10.1145/2181796.2206309. URL: https://doi.org/10.1145/2181796.2206309.

[60] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[61] Joseph Redmon et al. "You Only Look Once: Unified, Real-Time Object Detection". In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[62] Bin Ren et al. "A Survey on 3D Skeleton-Based Action Recognition Using Learning Method". In: *CoRR* abs/2002.05907 (2020). arXiv: 2002.05907. URL: https://arxiv.org/abs/2002.05907.

[63] Antônio H. Ribeiro et al. "The trade-off between long-term memory and smoothness for recurrent networks". In: *CoRR* abs/1906.08482 (2019). arXiv: 1906.08482. URL: http://arxiv.org/abs/1906.08482.

[64] Marcus Rohrbach et al. "A database for fine grained activity detection of cooking activities". In: *2012 IEEE Conference on Computer Vision and Pattern Recognition* (2012), pp. 1194–1201.

[65] Alina Roitberg et al. "Human activity recognition in the context of industrial human-robot interaction". In: *Signal and Information Processing Association Annual Summit and Conference (APSIPA), 2014 Asia-Pacific*. 2014, pp. 1–10. DOI: 10.1109/APSIPA.2014.7041588.

[66] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[67] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575.

[68] Alex Sherstinsky. "Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) network". In: *Physica D: Nonlinear Phenomena* 404 (Mar. 2020), p. 132306. ISSN: 0167-2789. DOI: 10.1016/j.physd.2019.132306. URL: http://dx.doi.org/10.1016/j.physd.2019.132306.

[69] Shubham Shinde, Ashwin Kothari, and Vikram Gupta. "YOLO based Human Action Recognition and Localization". In: *Procedia Computer Science* 133 (2018). International Conference on Robotics and Smart Manufacturing (RoSMa2018), pp. 831–838. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.07.112. URL: https://www.sciencedirect.com/science/article/pii/S1877050918310652.

[70] Keras Special Interest Group (Keras SIG). *Layers API*. 2018. URL: https://keras.io/api/layers/core_layers/dense (visited on 01/22/2022).

[71] Gunnar A. Sigurdsson et al. "Charades-Ego: A Large-Scale Dataset of Paired Third and First Person Videos". In: *CoRR* abs/1804.09626 (2018). arXiv: 1804.09626. URL: http://arxiv.org/abs/1804.09626.

[72] Tomas Simon et al. "Hand Keypoint Detection in Single Images using Multiview Bootstrapping". In: *CoRR* abs/1704.07809 (2017). arXiv: 1704.07809. URL: http://arxiv.org/abs/1704.07809.

[73] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1409.1556.

[74] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. "UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild". In: *CoRR* abs/1212.0402 (2012). arXiv: 1212.0402. URL: http://arxiv.org/abs/1212.0402.

[75] Wenjin Tao et al. "Attention-Based Sensor Fusion for Human Activity Recognition Using IMU Signals". In: *CoRR* abs/2112.11224 (2021). arXiv: 2112.11224. URL: https://arxiv.org/abs/2112.11224.

[76] Burak Teke et al. "Real-time and Robust Collaborative Robot Motion Control with Microsoft Kinect ® v2". In: July 2018, pp. 1–6. DOI: 10.1109/MESA.2018.8449156.

[77] Zhiqiang Teng et al. "Structural Damage Detection Based on Real-Time Vibration Signal and Convolutional Neural Network". In: *Applied Sciences* 10 (July 2020), p. 4720. DOI: 10.3390/app10144720.

[78] Subir Varma and Sanjiv Das. *Deep Learning*. 2018. URL: https://srdas.github.io/DLBook/.

[79] Michalis Vrigkas, Christophoros Nikou, and Ioannis Kakadiaris. "A Review of Human Activity Recognition Methods". In: *Frontiers in Robotics and Artificial Intelligence* 2 (Nov. 2015). DOI: 10.3389/frobt.2015.00028.

[80] Xi Vincent Wang et al. "Human–robot collaborative assembly in cyber-physical production: Classification framework and implementation". In: *CIRP Annals* 66.1 (2017), pp. 5–8. ISSN: 0007-8506. DOI: https://doi.org/10.1016/j.cirp.2017.04.101. URL: https://www.sciencedirect.com/science/article/pii/S0007850617301014.

[81] Bruce X. B. Yu, Yan Liu, and Keith C. C. Chan. "Skeleton Focused Human Activity Recognition in RGB Video". In: *CoRR* abs/2004.13979 (2020). arXiv: 2004.13979. URL: https://arxiv.org/abs/2004.13979.

[82] Jianjing Zhang et al. "Recurrent neural network for motion trajectory prediction in human-robot collaborative assembly". In: *CIRP Annals - Manufacturing Technology* 69 (July 2020), pp. 9–12. DOI: 10.1016/j.cirp.2020.04.077.

[83] Liang Zhang, Peiyi Shen, and Juan Song. "An Online Continuous Human Action Recognition Algorithm Based on the Kinect Sensor". In: *Sensors* 16 (Jan. 2016), p. 161. DOI: 10.3390/s16020161.

[84] Yue Zhang et al. "A Lightweight One-Stage Defect Detection Network for Small Object Based on Dual Attention Mechanism and PAFPN". In: *Frontiers in Physics* 9 (2021). ISSN: 2296-424X. DOI: 10.3389/fphy.2021.708097. URL: https://www.frontiersin.org/article/10.3389/fphy.2021.708097.

[85] Zhong-Qiu Zhao et al. "Object Detection with Deep Learning: A Review". In: *CoRR* abs/1807.05511 (2018). arXiv: 1807.05511. URL: http://arxiv.org/abs/1807.05511.