Universiteit Leiden
The Netherlands

Opleiding Informatica

Full-Body Action Recognition from Monocular RGB-Video:

A multi-stage approach using OpenPose and RNNs

Kalvin Maas

Supervisors:
Erwin M. Bakker & Michael S. Lew

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

23/02/2021

Abstract

Action recognition plays an important role in establishing a more natural and effective Human-Computer Interaction(HCI) and Human-Robot Interaction(HRI). We present a novel method for action recognition using monocular RGB video input captured from a fixed viewpoint. The presented system consists of a multi-stage approach using OpenPose and Recurrent Neural Networks (RNNs). It does not require any additional sensors or specialized hardware other than the monocular RGB camera. The method is validated in low resource scenarios showing its potential and wide applicability in real-time HCI and HRI. Finally, it is shown that, although designed for fixed viewpoint scenarios, the method also gives a competitive performance in multi-viewpoint scenarios.

Contents

1 Introduction

Action recognition is an active topic in computer science and has a wide variety of applications. Especially when considering Human-Robot Interaction(HRI), the use of a controller or keyboard to issue commands is often undesirable, thus various other interaction methods, such as speech recognition and emotion recognition, are in active development. Also action recognition is a method that has great potential as a primary or auxiliary interaction method. However, for action recognition to become viable in actual use cases it needs to be fast, accurate, and reliable.

For any actions to be processed by a computer it needs to be represented in a way such that a machine could work with it. There are various methods to achieve this. These methods include using one or multiple cameras and various specialized sensors attached to the individuals performing the actions and sensors placed in the room or at the robot.

Using a variety of sensors is however rather cumbersome due to the requirement of equipping either the individuals or the room with such sensors. A time-of-flight(ToF) sensor, such as LiDar, or structured light sensors, such as the Microsoft Kinect, offer solutions that allow for very effective gathering of data in a small package. These sensors and cameras are however not as widely available as a monocular RGB-camera. That is to say, it is more likely for general devices to be equipped with an RGB-camera rather than a ToF device. Therefore, performing action recognition using a monocular RGB-camera has the advantage of being a straightforward and widely accessible setup.

There are many machine learning approaches applied to the problem of action recognition. During the last decade, neural networks have gained a lot of popularity due to their great efficiency in a wide variety of applications. Various types of neural networks have proven to be effective at solving the action recognition problem. A type of neural network that has displayed potential for processing sequence- or time-sensitive data is the recurrent neural network(RNN). This is the type of data the action recognition problem has to deal with since it is often not enough to look at a single image of someone performing a action to know what they are doing.

In this work, we study the effectiveness of recurrent networks combined with 2D pose estimation to classify full-body actions using a single RGB-camera for input data and compare our method to existing methods. We also evaluate several variations in our RNN models, which involve adjustments to the overall architecture of our models and identify how these adjustments affect performance.

Additionally, we want to make sure that the system can perform in "real-time" using current technology. This is a particularly important factor when considering robot interactions. However, *current tech real-time* is a rather ambiguous term since even when a robot is collecting the input data, it is not necessarily the robot itself that processes the data, and that technology is perpetually advancing. In essence, this goal merely puts additional limits and considerations onto the system as a whole. For now, it would suffice for a modern conventional home PC[1] to be able to handle the workload and provide accurate classification with at least ∼5 FPS on input from a monocular RGB-camera. More specifically, our focus for this issue is to classify the actions the moment they have been completely performed. While predicting using the onset is interesting, it is not what we attempt to achieve with our method.

In this paper we made the following contributions made as a result of our research:

- *We evaluate the effectiveness of RNNs for action classification.*

[1] i7-8700K@3.7GHz, NVIDIA GTX 1070 8GB,32GB DDR4 RAM@3000MHz, Running Ubuntu 18.04.5 LTS

- *We present a multi-stage method for performing action classification from monocular RGB video input.*
- *We provide a real-time application that is capable of executing this method using a real-time RGB-camera feed.*
- *We compare the effectiveness of our method in a multi-view scenario.*

2 Related Work

Various methods for action recognition were previously proposed. Some earlier implementations focused on the classification of hand actions and facial expressions from camera images through the use of hidden Markov models, finite-state machines, and various types of neural networks[1].
Using additional data from a scene such as depth information proved valuable and the availability of cameras that could provide this data such as the Microsoft Kinect paved the way for more effective methods for action recognition. By using depth images it was possible to classify simple motion actions with support vector machines [2].
Various developments in pose estimation using the Kinect sensor [3] led to the availability of a pose estimation system in the Kinect SDK which can accurately estimate human joint positions in 3D space from data provided by the Kinect sensor. This pose estimation system allowed for the successful application of data mining classification methods for action recognition such as decision trees and naive Bayes[4].
Methods originally used in speech recognition also proved to be effective when using joint positions as input data, such as Dynamic Time Warping[5].
Recurrent Neural Networks (RNNs) can be used to effectively perform action recognition due to the ability to learn patterns in data where an order is significant, in this case, chronological order of input frames. The implementation of RNN used most in previous works related to the action recognition problem is the Long-Short Term Memory(LSTM) implementation. A newer implementation is the Gated Recurrent Unit implementation. Chung et al. [6] have found these two implementations to be comparable in performance, while outperforming the traditional *tanh* implementation, but noted that further experimentation would be required for evaluation. A 2-Stream RNN was used previously to recognize actions from a Kinect video feed by focusing on identifying hand and face positions[7].
Rwigma et al. proposed a differential evolution approach to optimize the weights of dynamic time warping for multi-sensory based action recognition which also used additional sensory data, such as from wearable inertial sensors and a Kinect sensor, included in the UTD-MHAD dataset and achieved an accuracy of 99.4%[8]. A method for extracting joint positions from images without additional depth information was previously proposed[9] and an application using this method is currently available for use in the form of OpenPose.
OpenPose determines skeleton keypoints from RGB-video and previous methods, such as dynamic time warping, have been shown to work on the data obtained through OpenPose as well. Such a method was proposed by Schneider et al. [10] who used the UTD-MHAD action dataset for their work, achieving an accuracy of 77.4%, but considering only a select few actions in the dataset. This shows that OpenPose can be used instead of specialized hardware and the potential results that could be obtained by using normal RGB-cameras.
Another challenge for action recognition methods is to be able to perform in view-invariant scenarios. For these scenarios. Ghorbel et al. [11] proposed a pipeline where 3D skeletons are first extracted using a CNN-based pose estimator, such as VNect, then applying feature extraction methods and a Support Vector Machine model to classify actions. The methods they proposed for feature extraction are Lie Algebra Representation of body-Parts (LARP), based on describing the geometric relationship between different coupled body segments, and Kinematic Spline Curves (KSC), utilizing the computation of kinematic values such as joint position, velocity, and acceleration. They applied their method to the Northwestern-UCLA multi-view action dataset, outperforming various

previously proposed methods by achieving a mean accuracy of 77.5%.
Baptista et al. [12] proposed a method for view-invariant action recognition, using a 3D pose estimator and a Long Short Term Memory(LSTM) based RNN to classify actions. On the NW-UCLA dataset, they achieved a mean accuracy of 79.9%.
Noori et al. [13] proposed a method combining keypoints generated by OpenPose with a LSTM-based RNN model to classify actions, regardless of the viewing angle of the recorded video. By calculating what they call "temporal motion features" from keypoints for any sequence of consecutive frames before passing it to an RNN, they show that a robust view-invariant action recognition system can be created. Through their method, they achieved an accuracy of 92.4% on the Berkeley MHAD dataset. By comparing their results with conventional approaches such as SVM, Decision Trees, and Random Forests, they display how a RNN-based model can outperform these approaches at performing activity recognition.

These previous skeleton-based RNN approaches have relied on either the availability of 3D pose estimations or feature extraction from 2D pose estimations. It is often assumed that 3D data is more informative than 2D data when it comes to the problem of action recognition. This is however not necessarily correct, as demonstrated by Marshall et al. [14]. Because of this, and since 3D pose estimation is generally more computationally intensive, we use 2D pose estimation in our work. Above that, we further limit the processing of the estimated joint positions, before applying any model, to only scaling to further reduce the computational power required and also improve real-time performance. Furthermore, we evaluate the overall performance of GRU compared to LSTM for our specific representation of data, carefully taking memory usage, classification speed and classification performance into consideration.

3 Fundamentals

First, we explain the performance metrics used to evaluate our method's performance. We will also provide a brief introduction to OpenPose and recurrent neural networks, the core components of our approach to multi-stage action recognition.

3.1 Performance metrics

There are several metrics which we can consider to illustrate the performance of our method. We will mainly be looking at the overall accuracy for this multiclass-classification problem. This is also the metric we can best compare to with previous methods. The accuracy is defined as the correct prediction divided by the total predictions which can be described by the following formula:

$$Acc = \frac{P_{true}}{P_{true} + P_{false}} \tag{1}$$

Another metric of interest is the time required to perform the classification. This is important due to our goal of having the application perform in real-time. While this metric is somewhat ambiguous considering the differences in computing power between machines and advances in technology, it is still important to achieve real-time viability using what is conventionally available at this time. For this we look at the Frames per Second we can achieve in the real-time application.

3.2 OpenPose

An important stage of our action recognition approach is OpenPose[9]. OpenPose is used to perform joint keypoint estimation. OpenPose is a system that is capable of detecting multiple individuals and estimate pre-defined joint keypoints, such as at the wrists or shoulders. By acquiring the keypoints it can form a skeleton for the people in the images by connecting their keypoints. It offers a few different keypoint models to choose from including a 15-, 18- and 25-keypoint body model. The difference between these models originates from the datasets they were trained on. Specifically, the 15 points model was trained using the MPII dataset, the 18 points used the COCO dataset and the 25 points used an altered COCO dataset. The models have different characteristics in terms of speed and accuracy, but at the time of writing the 25 point model is stated to be both the fastest and most accurate model. This 25 point model will be what we use in our method.
Optionally, additional hand and/or face keypoints could be included in the model at the cost of some performance loss. In its pipeline, OpenPose uses a convolutional neural network model created with Caffe to generate Part Confidence Maps(PCM) and Part Affinity Fields(PAF), an important part of the OpenPose pipeline which are then combined in a following step to form keypoint based skeletons of every individual. The architecture for this CNN model can be found in Figure 2. Perhaps most importantly, OpenPose performs rather well on a modern (2020) computer, allowing for its estimation to be performed in real-time on camera or (pre-recorded) video input making it suitable for our purposes.
OpenPose is an open-source library for real-time multi-person pose estimation. It relies on a greedy bottom-up approach to achieve high-quality results while not requiring additional computing power with an increasing amount of people in a scene. It achieves this by using convolutional neural networks to generate confidence maps for body part locations and a set of vector fields to encode

affinities between body parts. By using both these maps it can perform bipartite matching to find associations between candidate bodyparts to construct limbs. An illustration of this process used can be found in Figure 1 sourced from [9]. Conventional top-down approaches have used

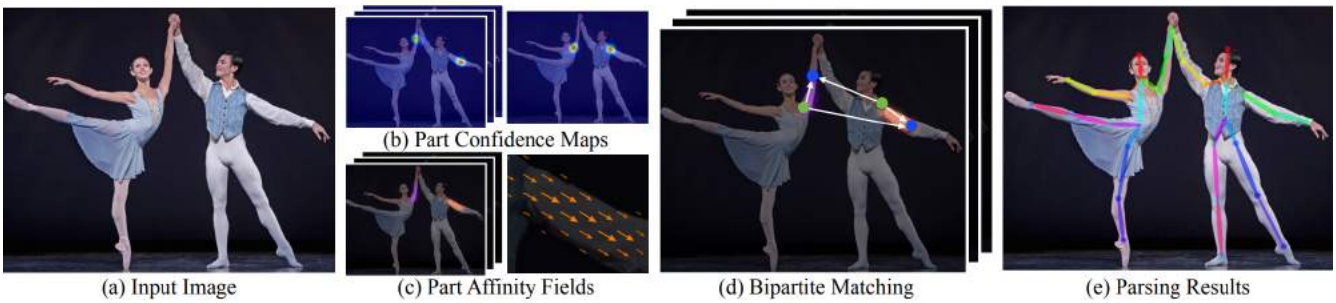


Figure 1: *OpenPose processing pipeline. OpenPose takes the entire image as the input(a) for a two-branch CNN to jointly predict confidence maps for body part detection, shown in (b), and Part Affinity Fields for parts association, shown in (c). The parsing step performs a set of bipartite matchings to associate body parts candidates (d). And finally, assemble and output them as full-body poses for all people in the image (e). Image from [9]*

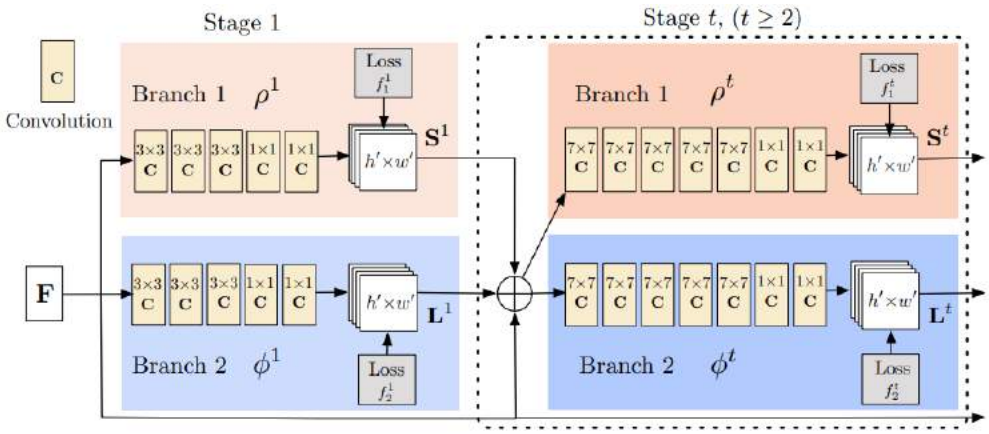


Figure 2: *OpenPose Architecture Neural Network, a two-branch multi-stage CNN. This network generates Part Confidence Maps and Part Affinity Fields, (b) and (c) in Figure 1. The image is first analyzed by a convolutional network, generating a set of feature maps F that is input to the first stage of each branch. Each stage in the first branch predicts confidence maps S^t, and each stage in the second branch predicts Part Affinity Fields L^t. After each stage, the predictions from the two branches, along with the image features, are concatenated for the next stage. The top branch predicts the confidence maps and the bottom branch predicts the affinity fields, which are subsequently used during bipartite matching and parsing to form full-body pose outputs. Image from [9]*

person detectors to identify individual bodies in images such that single person pose-estimation can be performed. This approach to multi-person pose estimation requires an increasing amount of computational performance with an increasing number of bodies in the scene while suffering accuracy degradation when people cluster together. The bottom-up approach as employed by OpenPose mitigates these issues through their part-affinity field(PAF) method.
More details about the exact operations performed by OpenPose can be found in their paper [9].

3.3 Recurrent Neural Network

Recurrent Neural Networks(RNNs) are a type of neural network that make use of recurrent layers or recurrent nodes. These nodes come equipped with a form of memory that allows them to store their own output from a previous input iteration and include it in the calculation for the output of subsequent iterations. This enables RNNs to recognize features in the input data where the chronological sequence is significant. In our case that would be the sequence of keypoint locations extracted from subsequent frames of the video.
Strictly, the only difference between recurrent nodes and simpler conventional nodes is the fact that recurrent nodes can use their previous state in their calculations for their following state, given subsequent iterations of inputs. The exact method of how this is achieved differs based on the implementation of the recurrent nodes. In Figure 3, the working of an RNN cell can be viewed unfolded over several timesteps. Essentially, for any given input x_t in timestep s_t, a recurrent cell can use some of its own output from the previous timestep s_{t-1} to produce the output for the current timestep o_t.

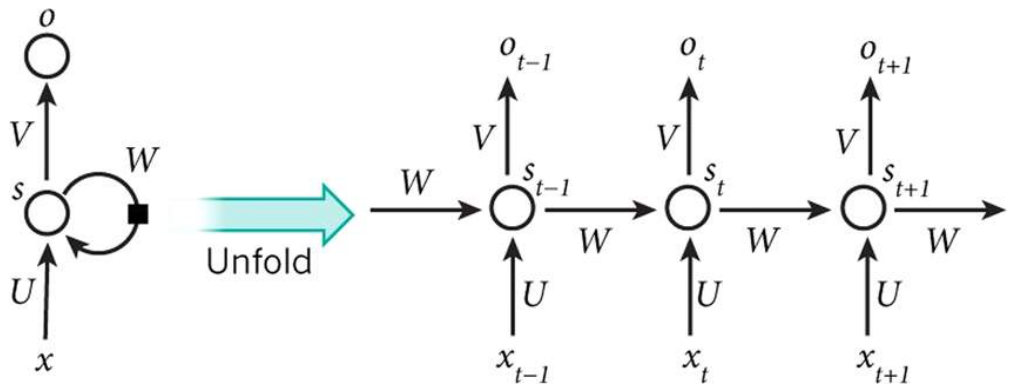
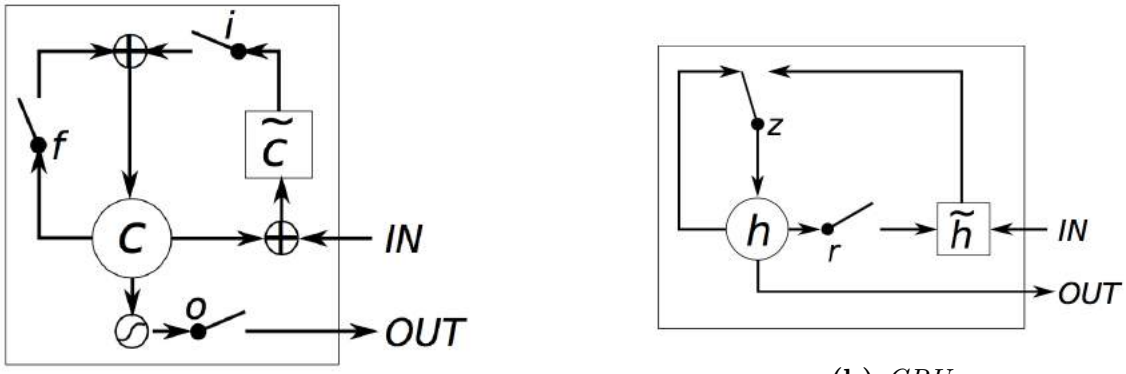


Figure 3: *A recurrent neural network and the unfolding in time of the computation involved in its forward computation. LeCun et al. [15]*

A commonly used implementation is Long Short-Term Memory (LSTM) implementation. A newer implementation that boasts performance similar to LSTM is the Gated Recurrent Unit(GRU) implementation.
Both LSTM and GRU make use of gates to control the flow of information through these nodes. These gates dictate how much influence its input and previous states have on calculations. There are connections into and out of these gates which have trainable weights that essentially determine how a gate handles the flow of information. The notable difference between the LSTM and GRU implementations is the layout of these gates. LSTM units use three gates, an input, output and a forget gate. GRU units only use input and forget gates in a configuration that differs from LSTM. Gates offer trainable parameters for a network model meaning that GRU models will have fewer trainable parameters than LSTM models if they are otherwise configured identically. Figure 4 illustrates a general layout for an LSTM and GRU.



(a) *LSTM* **(b)** *GRU*

Figure 4: *Illustration of an LSTM-node and a gated recurrent unit (GRU). (*a*) In an LSTM-node i, f, and o are the input, forget, and output gates, respectively. c and $\tilde{c}$ denote the memory cell and the new memory cell content respectively. (*b*) in a GRU-node r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation. Chung et al. [6]*

4 Baseline Methods

As a baseline method, we use Schneider et al. [10]. Their proposed method for monocular action recognition is based on Dynamic Time Warping(DTW). They also use OpenPose, but with the COCO trained 18-keypoint model. It is the most similar recent method on this subject we could find. It differs from our method that instead of DTW we propose to use RNNs. More specifically, we use LSTMs and GRUs in our method.

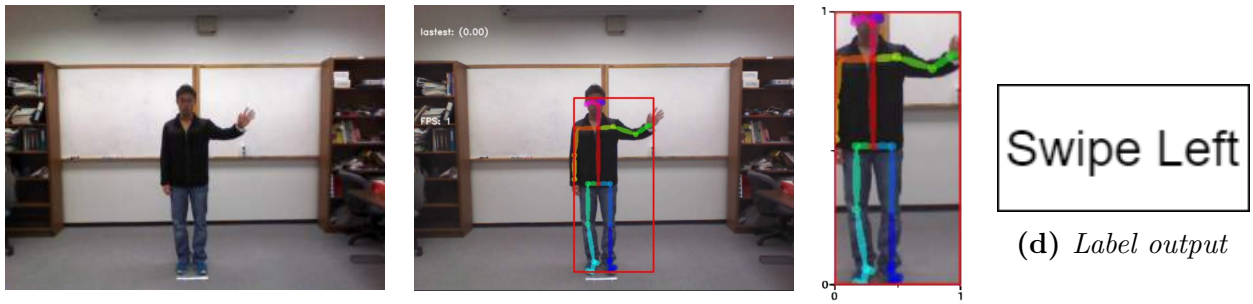
For our method, we also do not need to select a subset of actions from the dataset to perform classification on. This gives us more samples to work with while also including some action sets that are rather similar in terms of movements performed which may make the initial classification problem more challenging overall.

Furthermore, we also include some comparisons of our method with the performance of more "basic" neural networks which use only dense layers to illustrate the effectiveness of recurrent layers for this problem.

Lastly, we pay attention to the performance of our method compared to methods that use the additional sensor data available in the same dataset we used. While these methods should have an advantage over our method due to them simply having more data to work with, it is also interesting to see how close we are to matching the performance of these methods.

5 OPNNAR

In this section we describe our method, OPNNAR, in full. OPNNAR uses RNNs to classify actions from RGB-video by applying 2D joint estimation on the video and performing action classification by using a neural network on the sequence of joint positions extracted from the video. This can be viewed as a two-step approach of joint estimation followed by action classification. This method could potentially prove more effective than performing classification directly on the input video. This is because extracting keypoints effectively filters out large portion of data insignificant to action recognition, but there is also the risk that reducing an image to a 2D keypoint skeleton is too restrictive, filtering out too much data.
There will be no use of any data other than the RGB-video. This will exclude any depth imagery or additional sensory data. Further in this section, a more detailed explanation of the method can be found from the pre-processing steps up to the implementation method in the real-time application. A global overview of the method can be found in Figure 5 and the pipeline involved in Figure 6



(a) *Input image sequence* **(b)** *Pose estimations* **(c)** *Scaling* **(d)** *Label output*

Figure 5: *Overview of OPNNAR process. We use OpenPose to get pose estimations from an input image sequence. Then, we scale the collected data before it is fed to an RNN model, which in-turn generates a label for the action.*

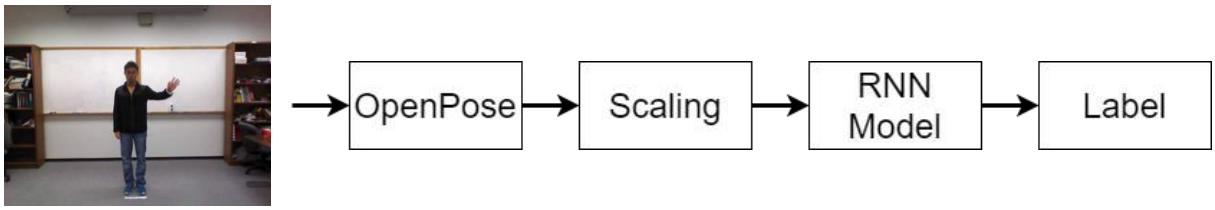


Figure 6: *OPNNAR pipeline. RGB video is fed to OpenPose. OpenPose outputs 25 keypoint coordinates per frame which are stored in a table. When we have collected 5 seconds worth of keypoints in the table (75 frames in the UTD-MHAD dataset) we scale the keypoints such that they all fall within a [0,1] range. This table is then fed to an RNN model which attempts to classify the action represented in the table and generates an output label. An example of one such RNN model can be found in Figure 8 on page 14.*

We use OpenPose to extract keypoint coordinates for each frame of the input video and scale the sequence of coordinates (5.1). We feed this data to an RNN model which then classifies the input as one of the actions which it has trained on in the dataset (5.2). The real-time implementation

contains some minor modifications in terms of pre-processing, details of which are further explained in Section 5.3.

5.1 Data pre-processing

We have chosen to represent the data in a way that is as consistent as possible across samples while simple to apply. The values are scaled such that the keypoint coordinates will be values within the interval [0,1]. In short, this is done by cutting out a bounding box which preferably exactly includes the person performing the action. To achieve this we use OpenPose to get an array of coordinates spanning a fixed time-frame and then translate and rescale the keypoint coordinates such that they match our criteria.
The following sections will explain how we approach, keypoint extraction and which scaling methods we propose and evaluate.

Keypoint Extraction

First, the video will be run through OpenPose to get the 2D keypoint coordinates. For this, the BODY_25 model is used which attempts to extract 25 keypoints from a frame of the video giving us 25 (x, y) points in the plain. This is done for each individual frame in the video. The keypoint data is stored in a datagram, or 2D array, consisting of 50 columns (25 x columns followed by 25 y columns), as 1 row for each frame.
For training we want the shape of the input datagrams to be fixed. Since the videos are of variable length adjustments need to be made to achieve this. We have selected the input shape to consist of 5 seconds of input (75 frames on 15 fps video) which means that longer actions will be cut off and shorter actions will be padded. For shorter inputs, the array is padded at the front rows with copies of the row corresponding to the first frame such that the actual action is contained in the latter rows of the array. For longer inputs, only the last 5 seconds of input are taken, and thus anything before that is cut out. The reason for using this approach is that for this problem it is assumed that the latter section of the input is more significant for being able to recognize a action.

Scaling Methods

There exist several methods for scaling. Regardless of the method we choose, the values that the model has to evaluate be within the interval [0,1].

Exact inclusion approach

in the exact inclusion approach the person is cut out of the frame, thus translating and scaling such that we subtract all x and y coordinates with the minimum of all X coordinates and the minimum of all Y coordinates respectively. After this mapping of (x, y) to ($\tilde{x}$, $\tilde{y}$) divide the $\tilde{x}$ and $\tilde{y}$ by the maximum of all $\tilde{x}$ coordinates and $\tilde{y}$ coordinates respectively. This means that in the image we end up with a rectangle bounding box which exactly includes the area where the subject performing the action has been for the last 5 seconds of input and the area itself is scaled for [0,0] top-left to [1,1] bottom-right.

Figure 7 shows an example of the exact inclusion scaling method. It essentially draws a bounding box around a detected person such that it is exactly contained within the rectangle [0,1]x[0,1] where the top-left is [0.0, 0.0] and the bottom-right is [1.0, 1.0].

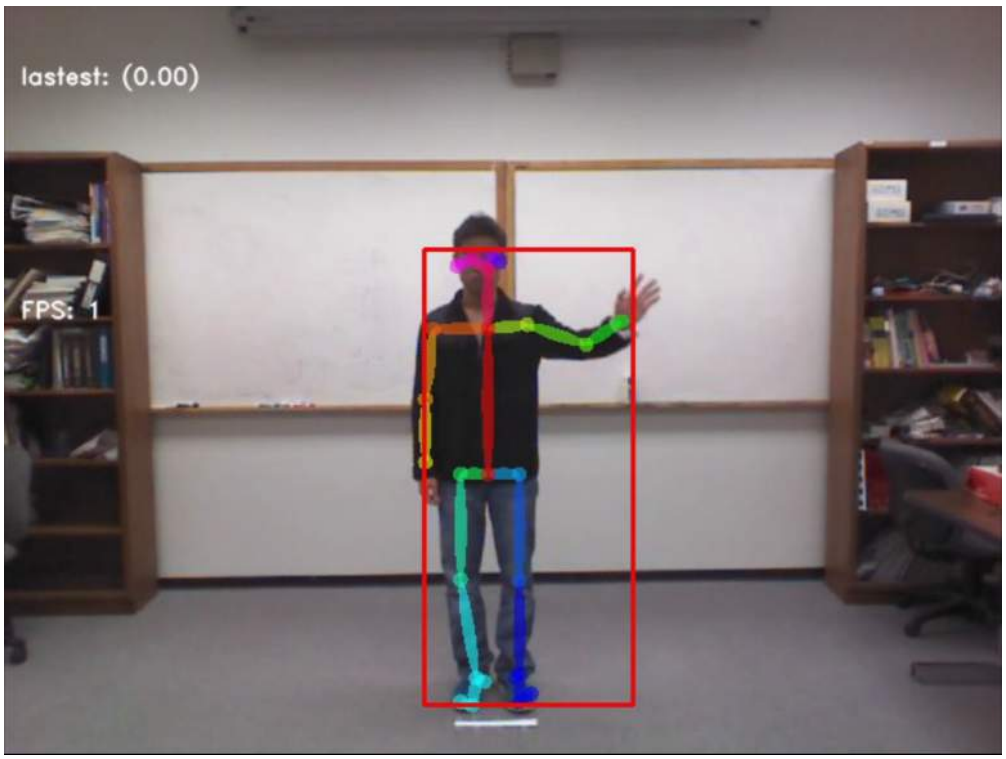


Figure 7: *Example of the scaling method on a video from the UTD-MHAD dataset. A bounding box is drawn to exactly contain the area of the keypoints, designating the area to be evaluated by the model.*

Fixed-Center approach

The Fixed-Center approach attempts to do something similar to the first approach but introduces a fixed point of reference to each input sequence. In this case, we adjust the values such that the neck keypoint in the first frame is always in the position [0.5,0.5]. Additionally, this method applies the same scaling factor to both the X- and Y-axis, meaning that the bounding box will effectively be a perfect square, which is not necessarily the case when using the first method. This is achieved by first translating the data such that the neck is at [0,0] then dividing all values in the array by twice the maximum absolute value found in the array and then translating again such that the neck keypoint of the first frame is at [0.5,0.5]. This will make sure the array values meet the described criteria.
Another variation on this would be to set the neck keypoint to 0.5 in every frame. However, doing so would cause a lot of variation in all other values when the neck moves around in the video causing a severe amount of additional unwanted noise and as such, we chose not to use this variation.

Other notable methods

Other methods of scaling such as scaling based on the distance between two fixed keypoints were attempted but in the set of 2-dimensionally represented keypoints there is no set of 2 keypoints that can be consistently used to scale all values. This is due to the different orientations of the people in the video and their respective body parts. When this would be attempted anyway it causes other values to reach extremes which is not desired for both training or in a future application. However, a result of the chosen method of scaling may be that the model will likely not work or not work as well when for example only half of a body is visible in the frames of the input video.
This method was used in some previous methods, notably in [10], where they also designed a method to select and filter keypoints specific for each action. However, we chose not to use this method of scaling.

5.2 OPNNAR Architecture

The OPNNAR architecture is designed such that it suits our problem best. Since one of our goals is to minimize the time required for classification there is no need to explore complex networks or networks that are computationally demanding, thus we attempt to create models that achieve a high degree of accuracy while minimizing the overall size of our networks.

5.2.1 LSTM vs GRU

We do not intend to design our own implementation of RNN cells or layers during this research, thus we will use some of the most popular existing implementations for our evaluations, namely LSTM and GRU as explained in a previous section. The LSTM and GRU layers are in practice very similar performance-wise with the favoured type varying on a case by case basis[6]. We need to establish if there would be any benefit in picking one over the other, mainly considering model accuracy, but also keeping in mind efficiency in terms of execution speed and model size.

5.2.2 Considerations

The variables considered when evaluating which models work best are:

- The number of subsequent recurrent layers.
- The number of units contained in each layer.
- The possible influence of the addition of one or several dense layers.

For comparison of the effectiveness of the use of recurrent layers, several different models consisting of only dense layers are also evaluated. This is also to verify our hypothesis that recurrent networks work well for a problem such as this one. Since the recurrent layers iterate over the input and thus can consider each value in the input data, the dense layer would need more units to provide a "fair" comparison. As such, a dense network with significantly more available hidden units is taken into consideration.
Recurrent layers can be stacked which influences the overall performance of the network. This is done by iterating over the activations of the previous layer per input iteration. For example, consider our input of 75 rows of 50 coordinates. The first recurrent layer, consisting of 100 units, would then iterate over this input row by row and calculate its activations per row giving us 75 rows of 100 activation states. A subsequent second layer would iterate over 75 rows of activations as if it was another input.
The number of units to take for each layer also needs to be evaluated. Observe that if a unit would be capable of learning the pattern of either an increase or decrease of values within a column over iterations, then we should have twice the number of input values per row than the number of units, which in our case means we have 100 units in the first layer. Any subsequent layers would then serve to recognize patterns in the activations of the previous layer. It should be noted that due to lack of comprehensible visualizations of said activations on the type of input data we weren't able to confirm said behaviour. However, the results still serve to confirm any hypotheses.

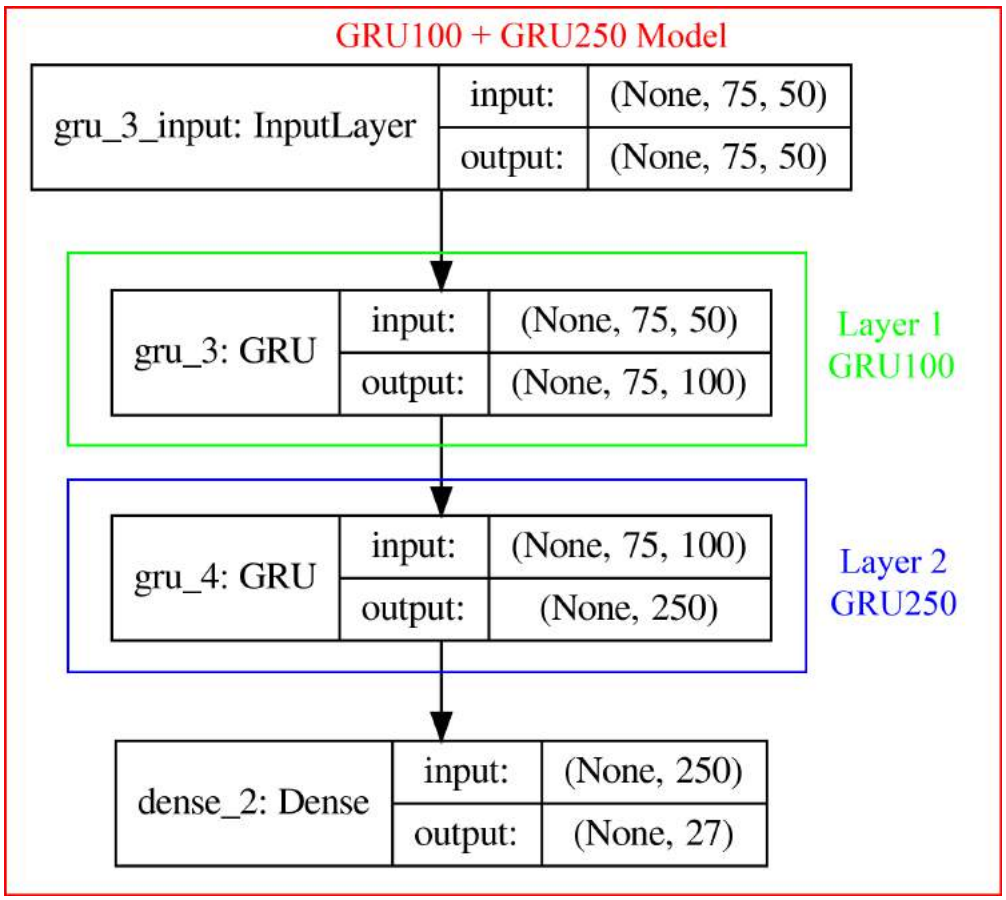


Figure 8: *Example model diagram (GRU100 + GRU250). The model takes a 75x50 array as input and feeds this to the recurrent layers for feature extraction. This example model has 100 units in the first GRU layer and 250 units in the second GRU layer, as can be seen from the output values. In the models we evaluate, any of the numbered layers can be a Dense, LSTM or GRU layer. These numbered layers being Layer 1, Layer 2, and sometimes a third layer, which is not present in this example, Layer 3. The final dense layer classifies these features to 1 of 27 actions.*

An example of a model we evaluate can be found in Figure 8. It consists of an input layer, a dense layer functioning as an output layer and up to 3 layers sequentially connected between them. This specific example contains 2 layers between the input and the output layer. The number of units that a layer consists of can be derived from the value in the output of a layer. For simplicity we refer to the different layer types by their types and unit counts, thus a GRU layer consisting of 100 units would be referred to as GRU100.

5.3 OPNNAR implementation and Real-time Application

We constructed an example application which makes use of the camera of the user for input. Firstly it uses OpenPose to estimate the keypoint positions for any given input frame. Then it checks if enough time has passed since the last frame to include the keypoints in the current frame in input for our model. This needs to be done since our model expects 5 seconds of input spread over 75 rows containing 25 keypoint coordinates each. If enough time has passed the keypoints are stored

in (or appended to) an array. If that would result in the array being longer than 75 rows then the oldest rows are removed from the array, essentially forming a FIFO queue. If too much time had passed since the last such that two or more steps would need to be stored to keep up then duplicate rows are appended until it caught up. This process ensures that the input requirements are met for our model, or more specifically that it matches the format of the dataset the model has trained on as closely as possible. Essentially, we extract data from live footage in a similar way as explained in the pre-processing section, with some alterations in order to reliably apply to camera input, such that we end up with an array of keypoint data that is effectively the same in format as what the model trained on.

Naturally, scaling is also necessary on the live input and will be performed before we feed the data to the model. Due to both our scaling methods, as described in section 5.1, requiring knowledge of the highest and lowest value in the entire sequence of keypoint data, we cannot simply re-use previously scaled values and thus we need to rescale the entire array of keypoint data for every frame of continuous camera input. Lastly, we let our model perform the estimation and then display the estimated action with the highest confidence on screen. The application makes use of the model with 2 stacked GRU layers consisting of 100 and 250 units for its predictions.

6 Dataset

As with most machine learning methods, we need data to train, validate and test on. A number of datasets exist which are suitable for action recognition, however, since the focus here mainly lies on being able to use actions as a means of providing commands to either a computer or robot it would be preferred to select a dataset which includes actions convenient for issuing commands. For this purpose, the UTD-MHAD[2] dataset was used. This publically available dataset containing a reasonable set of samples of relevant actions. We also evaluate the effectiveness of OPNNNAR on the NW-UCLA dataset. This is a multiview dataset which is used in several other action recognition methods such as in [12] and [11] with which we can compare our results.

6.1 UTD-MHAD dataset

The dataset contains nearly a thousand short videos of several people performing 27 distinct actions where one person performs one action at a time. The setting/scene is consistent across all videos such that the only distinct features are the people and the performed actions. Each of the actions is performed up to 4 times by 8 different people, 4 male and 4 female. The videos are recorded by a Kinect camera in a 640 by 480 format at 15 frames per second. The videos themselves are of variable lengths ranging between 2 to 7 seconds. Examples of what the videos look like can be found in Figure 9. A description of the action contents of the dataset can be found in Table 1.

#	Action	Samples	#	Action	Samples
1	Swipe left	31	15	Tennis swing	32
2	Swipe right	32	16	Arm curl	32
3	Wave	32	17	Tennis serve	32
4	Clap	32	18	Push	32
5	Throw	32	19	Knock	32
6	Arm cross	32	20	Catch	32
7	Basketball shoot	32	21	Pickup and throw	32
8	Draw X	31	22	Jog	32
9	Draw circle(CW)	32	23	Walk	31
10	Draw circle(CCW)	32	24	Sit to stand	32
11	Draw triangle	32	25	Stand to sit	32
12	Bowling	32	26	Lunge	32
13	Boxing	32	27	Squat	31
14	Baseball swing	32			

Table 1: *Actions and number of samples in the UTD-MHAD dataset*

The complete dataset also contains depth images and skeletal joint positions as estimated by the Kinect camera as well as extra inertial sensor data from wearable devices. For our purposes, this additional data would be irrelevant for anything other than means of comparison in performance and as such we ignore it.

[2]University of Texas at Dallas Multimodal Human Action Dataset

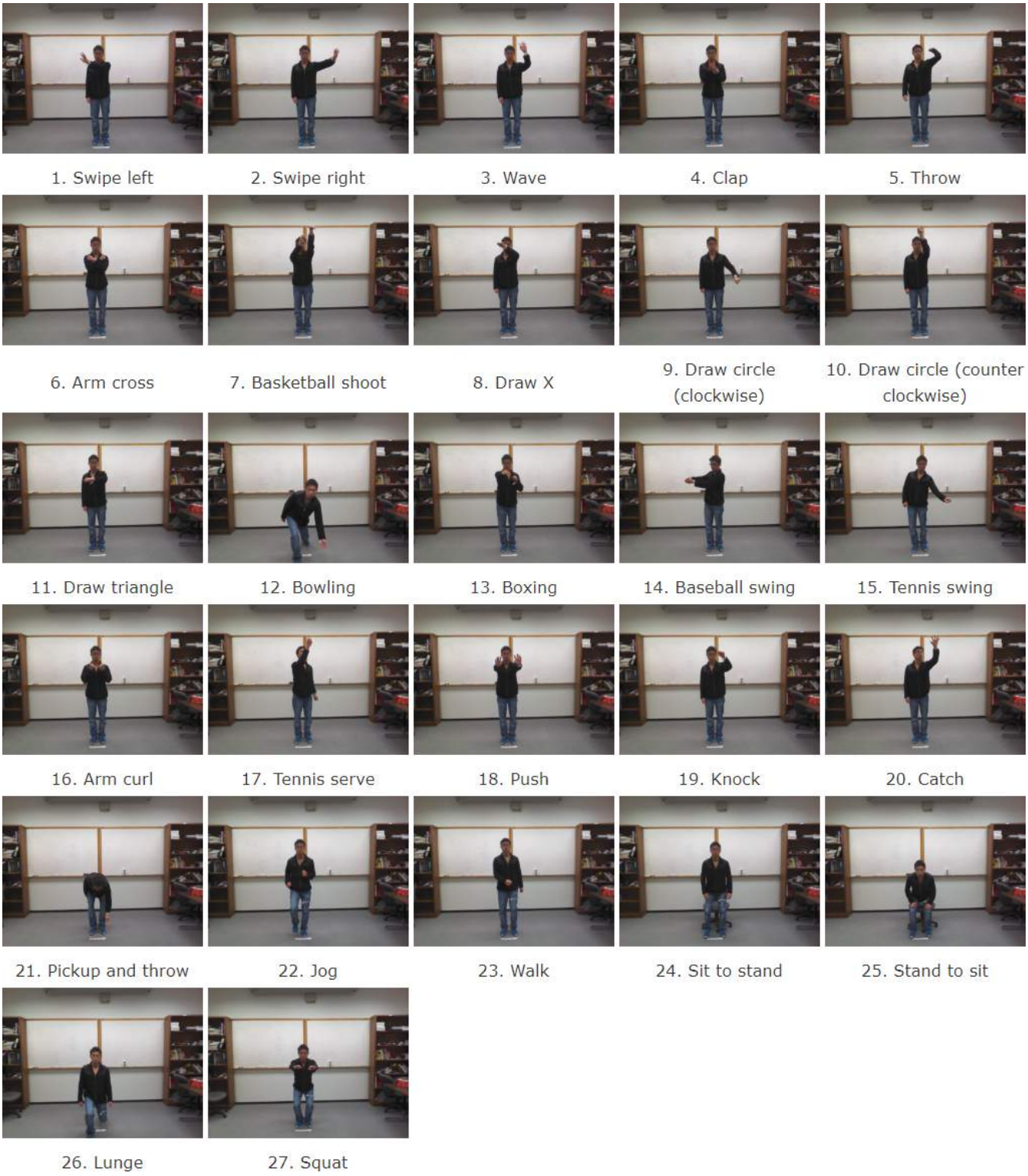


Figure 9: *UTD-MHAD action examples.*[3]

[3] https://personal.utdallas.edu/~kehtar/UTD-MHAD.html

6.2 NW-UCLA dataset

The NW-UCLA dataset consists of over a thousand videos of actors performing ten different actions. These videos are provided in a 640 by 480 pixels format at 12 frames per second. Each action has samples performed by 10 different actors. The actions are recorded from 3 different viewpoints and the actors also perform the actions in different orientations, thus facing different directions. The actions available in this dataset can be found in Table 2. Example images of the different viewpoints can be seen in Figure 10.

#	Action	Total Samples
1	Pick up with one hand	148
2	Pick up with two hands	146
3	Drop trash	140
4	Walk around	173
5	Sit down	148
6	Stand up	149
7	Donning	142
8	Doffing	142
9	Throw	145
10	Carry	142

Table 2: *Actions and number of samples per action in the NW-UCLA dataset. Note that any one action recorded from one view counts as one sample, thus one action recorded from 3 views totals 3 samples.*

The NW-UCLA dataset is considered one of the more challenging of the available datasets due to the different orientations as well as the overall similarity between individual actions. The method we proposed is not specifically designed to handle view-invariance but it is still interesting to evaluate how our method stacks up against the state-of-the-art in a similar problem.

(a) **(b)** **(c)**

Figure 10: *The NW-UCLA dataset contains actions recorded from 3 different viewpoints. These viewpoints can be seen in 10a, 10b and 10c, further referred to as viewpoint 1, 2 and 3 respectively.*

7 Experimental Setup

Here we briefly explain the environment used for training, testing and ultimately using neural networks. For this purpose we use Keras, which is a neural network API written in python, using TensorFlow as a backend. Keras allows us to design and run various types of neural networks.

7.1 Parameters and Settings

First we have to mention a number of parameters that we used during pre-processing with OpenPose and parameters that were used during training. Any parameter not mentioned in this section were left unchanged from their default values in both OpenPose and Keras.

- We have set the network resolution for OpenPose to "256x-1" (default: "-1x368). OpenPose will configure the "-1" accordingly to properly scale the input image. This was done in order to improve performance while minimizing the accuracy loss.
- During training we use a dropout of 30% on the input and between layers.
- We use the Adam optimizer for our models, with a learning rate set to "1e-3" and a decay of "1e-5".
- We use Categorical Cross-Entropy for our loss function.
- Any model is trained in 1000 epochs. This is the point where every model had already stopped showing significant improvements. We also do not use early-stopping to keep the number of epochs consistent.

7.2 Experiments Performed

In our experiments we seek to answer to following questions:

1. **For the action recognition problem, using the data format as described previously, does LSTM or GRU perform better?**
   We evaluate this by setting up and training a few networks which are simple, yet comparable to a design we would use in further experiments. We compare a single 100-unit layer design, as well as a design with two stacked 100-unit layers. We mainly want to identify if there is any notable performance difference and choose one to use in further experiments. Overall behaviour for both LSTM and GRU is expected to be similar, such as stacked versus non-stacked within the class of LSTM or GRU.
2. **Which form of scaling performs better? Exact Inclusion or the Fixed Center approach?**
   For this evaluation we use a similar approach as for the RNN-layer type comparison; A single 100-unit layer and a design with a 100-unit layer followed by a 250-unit layer, using the RNN-layer type as chosen through the comparison evaluation. Additionally, we also evaluate an RNN-layer followed by dense for comparison in case it might favour either method.

3. **How does an RNN compare to a dense network? And what kind of network design would be preferred?**
   We compare various designs of RNN network design as well as some dense networks designs and networks using both dense and RNN layers. for both RNN and dense networks we compare single 100-unit layer design. Since dense layers are generally much simpler compared to RNN-layers, and also because they are therefore quicker to train, we include a dense layer network with significantly more units in its dense layer than what we use for our RNN-layers for comparison. For the RNN designs we want to evaluate the performance differences when:

   - Stacking multiple RNN-layers.
   - Increasing the number of units in the RNN layers.
   - Combining the RNN-layers with a dense layer before the output layer.

   We mainly want to know if effectively increasing the number of trainable parameters would have any major influence on the accuracy of our models. Effectively we want to find the model which performs best with preferably fewer trainable parameters.

4. **How does our method perform in view-invariant scenarios?** For this, we look at the performance when using the NW-UCLA dataset. We will use the same training/testing method as used in [12], where two viewpoints at a time are used for training and validation and the remaining viewpoint is used for testing.

5. **Does our method perform in "real-time"?** Using what we previously described as our definition of real-time, does our method work and result in accurate action recognition with good frametimes.

8 Experimental Results

With the experiments performed in this section, we assess the effectiveness of recurrent neural networks when compared to previous methods or a basic network consisting of only dense layers. For both of the scaling methods mentioned in previous sections a training, validation and test set are constructed. The ratio used is 70%, or 591 examples, for training, 15%, or 135 examples, for validation and another 15% for testing. The example data is split such that every one of the actions appears as close to an equal amount of times as any other action for all sets mentioned.

8.1 UTD-MHAD dataset results

The following section contains the result gathered from experiments using the UTD-MHAD dataset. We performed the majority of our experiments on this dataset since this dataset depicts the situation our method was initially designed for.

8.1.1 LSTM vs. GRU

The results of these evaluations can be found in table 3.

Type	Units per layer		Trainable Param Count	Test Accuracy (%)
	Layer 1	Layer 2		
LSTM	100	x	63,127	94.1
LSTM	100	100	143,527	95.6
GRU	100	x	48,027	**96.2**
GRU	100	100	108,327	**96.2**

Table 3: *LSTM vs GRU performance. The type column denotes the kind of layer used for the numbered layers in the following columns. An* x *denotes that the layer was not present in the model.*

Performance is indeed very similar, yet, as can be seen in the table, during our testing GRU performed slightly better than LSTM overall. It also achieves this while having to utilize less trainable parameters. For those reasons, we selected GRU as the favoured implementation for further use.

8.1.2 Scaling method comparison

The results of comparing the two methods using some select model designs can be found in table 4. More models were evaluated on both methods, in fact, most of the model designs evaluated during this project were attempted on both input data sets but the results overall were the same.
From these results, we can conclude that the centred approach performs slightly worse on average. The idea that a central point of reference would be beneficial does not necessarily apply to systems like these and needed to be tested. The reason behind this decline in performance is possibly that reducing the variance of a certain keypoint by always centring the neck would unintentionally lead to more variance when presented with different body builds. Whether or not that's the actual reason behind it would require further investigation which is a subject for future studies.

Type		Units per layer		Test Accuracy (%)
		Layer 1	Layer 2	
Exact	GRU	100	x	96.2
Inclusion	GRU	100	250	**97.8**
Scaling	GRU + Dense	100	100	94.8
Fixed	GRU	100	x	**91.1**
Center	GRU	100	250	89.6
Scaling	GRU + Dense	100	100	85.2

Table 4: *Scaling method results. The type column denotes the design of the model, eg.* GRU + Dense *means that we used a GRU layer for Layer 1 and a Dense layer for Layer 2. Additionally, the keyword* Centered *denotes that the Fixed-Center scaling method was used for that set.*

8.1.3 Model design results

A number of models were tried and tested for this classification problem. Some notable ones can be found in Table 5.

Type	Units per layer			Trainable Param Count	Test Accuracy (%)
	Layer 1	Layer 2	Layer 3		
Dense	100	x	x	207,627	85.2
Dense	3750	x	x	7,785,027	83.7
Dense	3750	250	x	1,635,277	78.5
GRU	100	x	x	48,027	96.2
GRU	100	100	x	108,327	96.2
GRU	100	100	100	168,627	97.0
GRU + Dense	100	100	x	58,127	94.8
2x GRU + Dense	100	100	100	118,427	97.0
GRU	100	250	x	315,327	**97.8**
GRU	100	500	x	960,327	95.6
GRU	250	x	x	232,527	97.0
GRU	250	250	x	608,277	**97.8**
GRU	500	x	x	840,027	**97.8**

Table 5: *Model design results. Again, the type column denotes the types of layers used in the numbered layers, eg.* 2x GRU + Dense *means that 2 stacked GRU Layers were used for Layer 1 and Layer 2 followed by a single dense layer for Layer 3. If only one type is defined in the type columns then all numbered layers are of that type.*

What stands out in these results is that using recurrent layers provides actual improvements when compared to using only dense layers regardless of the amount of hidden dense units used. This more than anything proves our theory that using recurrent networks and their capability to apply some form of chronologically sensitive analysis using their memory is beneficial when encountering this problem.
Furthermore, the 3750 unit single dense layer model has an outstandingly high number of trainable params. This is due to the flatten layer used before the output layer, which in turn causes a large

number of trainable params towards the output layer due to it having weights proportional to the output of the layer before it.
Experiments with various different model designs using the recurrent layers did not result in any major improvements out of the margin of error in most of the observed cases. Something worth mentioning are some limits to this approach. During the experiments, some actions such as *arm curl* and *push*, and various other pairs, would often be misinterpreted to be the other. Upon further inspection, it could be concluded that these actions are in some cases nearly indistinguishable when just viewing them as 2-dimensional keypoints. This is where 3d data, including depth information or even a convolutional model directly trained on the input video, may still be able to provide some improvements in terms of accuracy. However, given the level of accuracy we already managed to reach using just this 2d information, any improvements would likely be minor.

8.2 NW-UCLA dataset results

Table 6 shows our performance results compared to other RGB-based methods on the NW-UCLA dataset. We used the *cross-view* splitting protocol as proposed in [16]. This protocol uses 2 viewpoints for training while leaving out the last for testing. We do this for each combination of viewpoints.

Method	{Training VPs} — {Test VP}			Mean
	{1,2} — 3	{1,3} — 2	{2,3} — 1	
Hankelets (Li et al. [17])	45.2	-	-	-
dvv1 (Li and Zickler [18])	58.5	55.2	39.3	51.0
CVP (Zhang et al. [19])	60.6	55.8	39.5	52.0
AOG (Wang et al. [16])	73.3	-	-	-
nCTE (Gupta et al. [20])	68.8	68.3	52.1	63.0
NKTM (Rahmani and Mian [21])	75.8	73.3	59.1	69.4
VNect+KSC (Ghorbel et al. [11])	86.3	79.7	66.5	77.5
VE-LSTM (Baptista et al. [12])	87.2	82.1	70.4	79.9
OPNNAR (ours)	**70.8**	**75.0**	**71.8**	**72.5**

Table 6: *Accuracy of recognition(%) on the NW-UCLA dataset. The accuracy values are obtained by using the two viewpoints for training and one viewpoint for testing. Mean is acquired by averaging the three tests.*

As can be noticed from the table, our method performs quite well even in a situation it was not designed for. It does not perform as well as on the UTD-MHAD dataset but that is expected considering this dataset is more challenging due to view-invariance and action similarity. However, considering our method is mainly focused on HCI/HRI use, the overall significance of view-invariance is up for debate.
A confusion matrix of the situation where viewpoint 3 is used as the test set can be found in Figure 11.

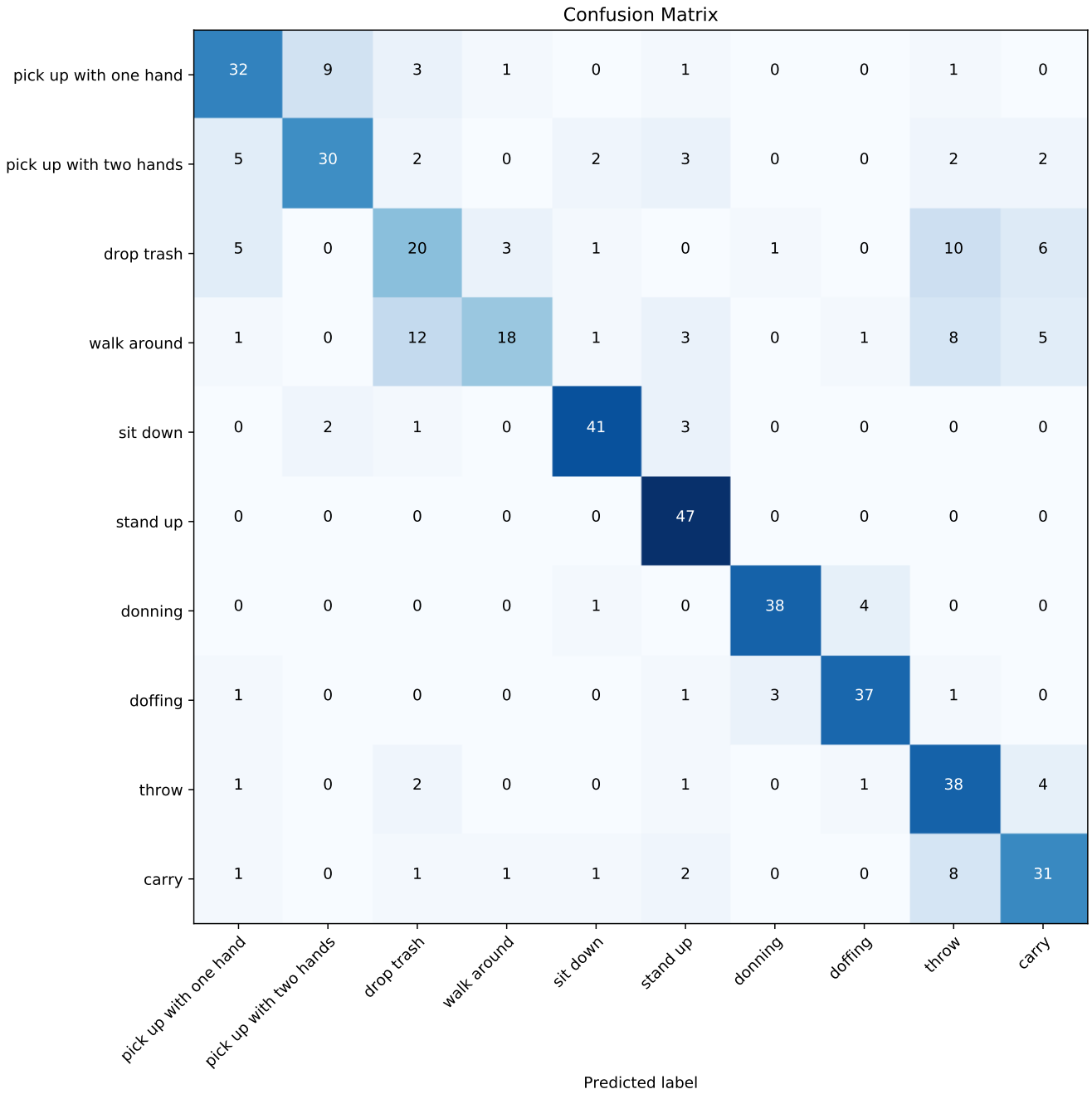


Figure 11: *Confusion matrix for a* (GRU250+GRU250) *model on the NW-UCLA dataset (using viewpoint 3 as the test set)*

8.3 Real-time input

The application was capable of classifying some simple and some more ambiguous actions in the original dataset accurately. It also remains somewhat accurate as the user starts moving closer to the camera leaving some less significant bodyparts, and thus keypoints, out of the frame. This is a good result considering that the model was trained on different people entirely, in a different setting and where most, if not all, parts of the body were in frame at all times. This serves to prove how well this method is able to generalise the problem as well as illustrate its potential.

Situation	FPS Maintained
OpenPose only	29
Openpose + Array-ops	19
Full	15

Table 7: *Maintained FPS on the following system:* i7-8700K@3.7GHz, NVIDIA GTX 1070 8GB, 32GB DDR4 RAM@3000MHz, Running Ubuntu 18.04.5 LTS.

The entire process can be maintained at ∼15 Frames per Second(FPS) on the used system. For comparison, when not performing any of our tasks and just letting OpenPose perform its pose estimation on the input while retaining every other parameter results in ∼29 maintained FPS. Performing the processing before feeding the data to our model but not letting our model estimate the performed action results in ∼19 maintained FPS. From this, we can tell that a relatively low portion of computing resources is spent on our classification model each frame compared to the task of pre-processing the data in our used method. An example of the application classifying a action can be found in the images of Figure 12.

(a) *Start state*

(b) *Raise Arm*

(c) *Swipe*

(d) *Swipe Detected with 1.00 confidence*

Figure 12: *Example application run sequence. The action is performed starting from a neutral stance(a). Similar to the dataset videos we raise the arm (b) before swiping (c). Once returned to the neutral state the action is classified and displayed with the confidence value on the left (d).*

9 Conclusions

To conclude, we presented a novel method for performing action recognition through the use of OpenPose keypoint estimation and recurrent neural networks and explored various variations in the interpretation of that method. We have shown how a relatively small dataset can be used to train a model to accurately classify a variety of different actions while not requiring any special hardware other than a monocular RGB camera. We have shown that this method performs well when compared to using purely dense neural networks and that it holds up against methods proposed in previous research. Additionally, we proved the generalization capability of our method by applying the created models by applying it in a view-invariant situation. We also used our method in a real-time application. With this application, we were able to classify live performed actions in an unknown setting to a great level of accuracy.

With our 97.8% accuracy top results on the test set, we have managed to improve on the previous method on just RGB-videos by Schneider et al. [10] (77.4%). As opposed to their method, it was not required for us to select a small subset of actions. Instead, we perform classification over all actions included in the dataset. This meant that there was a greater overall similarity in actions than if we had chosen to select a more distinct subset. Regardless, our method managed to correctly discriminate similar actions in the dataset (eg. walk vs. jog) and actions that would be difficult to distinguish through single-frame analysis (eg. left- vs right swipe).

Our method also managed to closely approach the 99.4% accuracy reached by the method proposed by Rwigema et al. [8] without needing the additional sensor data that they used.

Even when applied in a view-invariant scenario, our method performs very well, achieving a mean accuracy of 72.5%, approaching the scores achieved by Baptista et al. [12](79.9%) and Ghorbel et al. [11](77.5%) and beating many of the methods that came before them. A great result considering our method was not designed with view-invariance in mind.

10 Future Research

There are a variety of opportunities for improvement in the system we presented. For now, we used on OpenPose to provide keypoint estimation, mainly because of its open availability, ease of use thanks to the API, great performance and its relative recency with the API being released in early 2019. However, it could potentially be *replaced by another keypoint estimation system* if available and performance could be compared and possibly improved.

We also opted not to use any of the *multi-person features* available in OpenPose, which was arguably the main distinctive feature of the method. Through an extension of our proposed method, it may be possible to track people and classify actions for each person individually. Due to the nature of our method, this would likely lead to a linear performance loss with an increasing amount of people in the scene but that would need to be determined.

It may also be possible to *directly incorporate the part affinity fields and part confidence maps* as generated and used by OpenPose rather than just the OpenPose output for input data in a model. Which will likely have an effect on the overall performance of the system. To be more specific, our method relied on keypoint locations over several video frames to perform action classification. It could be possible that different attributes or representations of movements in the video input could provide improved performance.

Lastly, we will mention that the operations we performed on the input data in the real-time application may not have been optimal. As mentioned, with our method we weren't able to use any of the previously processed inputs for the next frame. This meant that for each new frame in a sequence of 75 total frames we were unable to re-use any of the 74 still relevant previously scaled frames. Therefore causing a large amount of calculation and operations which are possibly redundant. By *changing the method used for scaling* or through improvements on the existing method it is likely that the performance of the real-time application overall could be significantly enhanced.

References

[1] Sushmita Mitra and Tinku Acharya. Gesture recognition: A survey. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 37(3):311–324, 2007.

[2] Kanad Biswas and Saurav Basu. Gesture recognition using microsoft kinect®. pages 100–103, 12 2011. doi: 10.1109/ICARA.2011.6144864.

[3] Pushmeet Kohli and Jamie Shotton. Key developments in human pose estimation for kinect. In *consumer depth cameras for computer vision*, pages 63–70. Springer, 2013.

[4] Orasa Patsadu, Chakarida Nukoolkit, and Bunthit Watanapa. Human gesture recognition using kinect camera. pages 28–32, 05 2012. ISBN 978-1-4673-1921-8. doi: 10.1109/JCSSE.2012.6261920.

[5] Sait Celebi, Ali Selman Aydin, Talha Tarik Temiz, and Tarik Arici. Gesture recognition using skeleton data with weighted dynamic time warping. In *VISAPP (1)*, pages 620–625, 2013.

[6] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[7] Xiujuan Chai, Zhipeng Liu, Fang Yin, Zhuang Liu, and Xilin Chen. Two streams recurrent neural networks for large-scale continuous gesture recognition. In *2016 23rd International Conference on Pattern Recognition (ICPR)*, pages 31–36. IEEE, 2016.

[8] James Rwigema, Hyo-Rim Choi, and TaeYong Kim. A differential evolution approach to optimize weights of dynamic time warping for multi-sensor based gesture recognition. *Sensors*, 19(5):1007, 2019.

[9] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields. In *arXiv preprint arXiv:1812.08008*, 2018.

[10] Pascal Schneider, Raphael Memmesheimer, Ivanna Kramer, and Dietrich Paulus. Gesture recognition in rgb videos using human body keypoints and dynamic time warping. In *Robot World Cup*, pages 281–293. Springer, 2019.

[11] Enjie Ghorbel, Konstantinos Papadopoulos, Renato Baptista, Himadri Pathak, Girum Demisse, Djamila Aouada, and Björn Ottersten. A view-invariant framework for fast skeleton-based action recognition using a single rgb camera. In *14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, Prague, 25-27 February 2018*, 2019.

[12] Renato Baptista, Enjie Ghorbel, Konstantinos Papadopoulos, Girum G Demisse, Djamila Aouada, and Björn Ottersten. View-invariant action recognition from rgb data via 3d pose estimation. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2542–2546. IEEE, 2019.

[13] Farzan Majeed Noori, Benedikte Wallace, Md Zia Uddin, and Jim Torresen. A robust human activity recognition approach using openpose, motion features, and deep recurrent neural network. In *Scandinavian conference on image analysis*, pages 299–310. Springer, 2019.

[14] Fiona Marshall, Shuai Zhang, and Bryan Scotney. Comparison of activity recognition using 2d and 3d skeletal joint data. In *Irish Machine Vision & Image Processing IMVIP 2019*, page 13. Irish Pattern Recognition and Classification Society, 2019.

[15] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[16] Jiang Wang, Xiaohan Nie, Yin Xia, Ying Wu, and Song-Chun Zhu. Cross-view action modeling, learning and recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2649–2656, 2014.

[17] Binlong Li, Octavia I Camps, and Mario Sznaier. Cross-view activity recognition using hankelets. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1362–1369. IEEE, 2012.

[18] Ruonan Li and Todd Zickler. Discriminative virtual views for cross-view action recognition. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2855–2862. IEEE, 2012.

[19] Zhong Zhang, Chunheng Wang, Baihua Xiao, Wen Zhou, Shuang Liu, and Cunzhao Shi. Cross-view action recognition via a continuous virtual path. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2690–2697, 2013.

[20] Ankur Gupta, Julieta Martinez, James J Little, and Robert J Woodham. 3d pose from motion for cross-view action recognition via non-linear circulant temporal encoding. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2601–2608, 2014.

[21] Hossein Rahmani and Ajmal Mian. Learning a non-linear knowledge transfer model for cross-view action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2458–2466, 2015.