Universiteit Leiden
The Netherlands

Opleiding Informatica

CHAD-net: Continuous Human Action Detection in RGB video

Oscar T.C. Bergman

Supervisors:
Erwin M. Bakker & Michael S.K. Lew

BACHELOR THESIS

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

20/01/2022

Abstract

In this paper, the problem of Human Action Detection (HAD) in live video feeds is studied. A novel architecture for Continuous Human Action Detection (CHAD) is introduced that differs from traditional HAD algorithms, in that the CHAD-net is trained on sub-actions that make up larger actions. This enables the network to start detecting actions taking place before they have been completed, giving it an edge in Human-Robot Interaction (HRI) scenarios or when larger videos consisting of more than a single action are processed. The CHAD-net architecture uses ShuttleNet to detect sub-actions, and an Action Deducer (AD) to detect which action is taking place based on the detected sub-actions. This approach is evaluated on a HAD dataset (UCF101) and a specially selected subset C_UCF101 that is more themed towards continuous human actions. Finally, experimental results show that state-of-the-art solutions for HAD outperform CHAD-net on the UCF101 dataset, but CHAD-net shows promising results on the C_UCF101 dataset.

Contents

1 Introduction

Human Action Detection (HAD) in video footage has been a sizeable topic in Computer Vision research for the last decades. Detecting what a human is doing at any given time in a video has various applications in fields such as robotics, surveillance and Human-Computer Interaction (HCI). Optimally the HAD methods should detect actions as fast and accurately as possible, so that the program can be used on live video streams encountered in various HCI and Human-Robot Interaction (HRI) scenarios. In reality, many HAD architectures are designed for and evaluated on datasets where every video begins at the start of an action and ends at the end of an action, whereas in more realistic scenarios the action would only be a small part of the video. Hence, this is not representative of a live video feed, because in a live video feed a program does not know where an action begins or ends. Naive solutions to this problem, such as splitting the incoming video into parts of a few seconds, have the potential of completely missing the action by cutting it in half.
To solve the problem of using HAD on continuous video feeds, now referred to as Continuous Human Action Detection (CHAD), a change in approach was necessary. A decision was made to focus on the smaller parts of actions, aptly named sub-actions, rather than the whole action itself. That way, even if a specific sub-action was not detected, the architecture could still deduce the action from the other sub-actions it did detect. This also has the potential of being able to detect an action before it is completely finished.
HAD can be performed using a variety of modalities. The most popular and readily available modality is RGB video data. Many HAD methods try to improve their accuracy by adding more modalities, such as optical flow, depth or multiple viewpoints. Most of these modalities require extra pre-processing time to derive them and/or extra hardware/sensors. While accuracy might increase, the amount of cases where the system can be used decreases. Here a decision was made to use only RGB video data, furthermore the focus of this paper, with HRI in mind, was on the requirement of handling a live video feed, making speed/complexity an important factor.
In this paper a model was created building on these ideas, consisting of a Sub-Action Specialist (SAS) to detect which sub-action has taken place in an RGB video, and an Action Deducer (AD) which attempts to deduce the action taking place based on the detected sub-actions. To determine the accuracy of the model, named CHAD-net, it was compared against state-of-the-art methods on the UCF101[1] dataset and on a subset of the same dataset called C_UCF101. The UCF101 dataset is a widely used action recognition dataset, containing realistic action videos collected from YouTube. It contains 13320 total videos spanning 101 different action categories in total, and is widely used for the evaluation of many state-of-the-art HAD methods, amongst others. The dataset does contain some actions that are repetitive, or actions without a clear beginning, middle or end. This works fine for HAD methods, but would make it harder to test a CHAD method. That is why we selected 27 action categories of the original UCF101 dataset that are not repetitive and have a clear beginning, middle and end. The resulting dataset is called C_UCF101.

The following contributions were made in this paper:

- We propose CHAD-net, a multi-stage method for detecting actions from their sub-actions.
- We compare the effectiveness of our method to state-of-the-art for both HAD and CHAD and show that although the CHAD-net performance on the HAD problem is disappointing it has potential for CHAD.

The rest of this paper is organized as follows: Section 2 discusses related work in the HAD state-of-the-art for different kinds of data modalities. In Section 3 we briefly introduce some fundamental concepts used in this paper, such as Recurrent Neural Networks (RNNs) and performance metrics used. The baseline method we chose, ShuttleNet, is introduced in Section 4, where it is described in detail. Section 5 presents our proposed CHAD-net, which consists of ShuttleNet as our Sub-Action Specialist (SAS) and an Action Deducer (AD). We also describe the inputs used for the AD and describe a null-method used to compare results of CHAD-net to those of ShuttleNet. We list the datasets used and describe why we use them in Section 6. The experimental setup is described in Section 7. There the HAD and CHAD experiments are described and is shown what AD model is used for each input type of the AD. Experimental results are discussed in Section 8, where we compare CHAD-net to the state-of-the-art on HAD and to ShuttleNet on CHAD. Finally, we conclude the paper in Section 9.

2 Related Work

In this section the various methods for human action detection are discussed. The different HAD methods are grouped based on differences in approach and/or data modalities. The final selection of our baseline method is based on our findings while reviewing the state-of-the-art in HAD research.

2.1 RGB Data

Ghorbel et al. [2] made use of VNect [3] to estimate a 3D skeleton from a single RGB image. This skeleton was then used to extract view-invariant skeleton-based features. This resulted in an architecture that handled cross-view action detection very well, with only a single RGB camera as input. Cross-view action detection is beyond the focus of our work.
Liu et al. [4] opted to use convolutional pose machines to predict pose estimation maps of each body part. These maps were then aggregated to form a heatmap and a pose for each frame. The evolution of the heatmaps were described as a body shape evolution image, via spatial rank pooling. With body guided sampling the evolution of poses were described as a body pose evolution image. Deep features were then extracted from both these images and used to predict the action labels. This method can capture movements of both body shape and body parts, and yielded results that outperformed many state-of-the-art 3D and 2D pose-based methods. Generating the posemaps and heatmaps is a computationally very demanding task, as a result this method is not currently suitable for HAD processing the live video feeds in HRI scenarios.
Noori et al. [5] decided to use the open source library OpenPose [6] to extract a 2D skeleton from an RGB image. They then computed magnitude and angle for each of the generated body joints as motion features and fed these features into an RNN(Section 3.1) with LSTM(Section 3.1.1) cells. This resulted in an approach that worked well for person-independent and view-invariant action detection. Although this method is very promising, its accuracy is lower than other HAD methods.
A similar approach using OpenPose was created by K. Maas [7], where after estimating the 2D skeleton the image was scaled down to the rectangular area containing the detected the skeleton. This image is then fed into an RNN model consisting of GRU (Section 3.1.2) cells to get results. This method was not considered further, due to its lower accuracy.
J. He et al. [8] first extracts optical flow data from RGB images. They then grab a random sample

from the data and put it into a 'sampling stack'. These sampling stacks are then fed into a Stack Representation Learner (SRL). The RGB frames are fed into a Spatial SRL, while the optical flow data is fed into a Temporal SRL, both of which contain CNNs. The SRLs then each produce a feature stack, which is used to model the temporal pattern using their novel Densely-connected Bi-directional LSTM. Finally, a fusion layer is used to combine both spatial and temporal data and produce a result. While this method outperforms many of the state-of-the-art in HAD, generating optical flow data is computationally demanding and thus not currently suitable for HAD processing live video feeds in HRI scenarios.

Y. Shi et al. [9] introduce a CNN-RNN network structure called ShuttleNet that is inspired by the cortical pathways in the brain. It can process both RGB data and optical flow data. The convolutional neural network (CNN) part is used for feature learning. Both GoogLeNet[10] and Inception-ResNet-v2[11] were tested as CNN in this method. The output from the CNN is then given to the RNN part, which is used to learn temporal features. While in most network structures layers are stacked one by one, with the final layer producing the final output, this method arranges its layers in a circle, feeding output from one layer into the next clockwise with a predetermined stride. This allows layers to be reused multiple times in each pathway, decreasing the number of parameters while keeping the network depth almost fixed, consequently reducing over-fitting. Each layer in this method consists of RNNs such as LSTMs or GRUs. The amount of layers in the RNN part and the amount of steps the data cycles through it are variable. Finally, the best pathway through the RNN part is then selected by an attention mechanism. This method outperforms many other state-of-the-art HAD methods in accuracy, and we were able to validate the reported results.

2.2 Depth Data

Some HAD methods enhanced their RGB data with depth data. For example, S. Das et al. [12] uses the output of a Microsoft Kinect Sensor combined with a glimpse sensor to focus on three specific parts of the image: the left hand, the right hand, and the full body. Each of these 3 parts are fed into their own I3D sub-network. The outputs of these networks are then combined with an RNN based attention model, which takes the 3D skeleton from the depth map captured by the kinect and uses it to dictate how much attention should be given to each body part. The results of the combining of the I3D and RNN networks are aggregated and the final results are outputted. By assigning different importance to different bodyparts, this method was able to compete with state-of-the-art models which also used multiple modalities.

M. Al-Faris et al. [13] makes use of a fuzzy algorithm to give each generated depth motion map various weights to assign importance to the motion information. These depth motion maps are then concatenated to create a fuzzy weighted multi-resolution depth motion map(FWMDMM). It also uses two parallel AlexNet[14] networks to compute features of the RGB and Depth information. These features and the FWMDMM are then inserted into a deep motion model to get the results. The resulting model can help to provide improved differentiation between similar actions.

A. Elboushaki et al. [15] use two sub-networks. The first of these is the 3D Color-Depth Convolutional Network (3D-CDCN) which takes RGB and Depth as an input. This network specializes in spatiotemporal features. The second sub-network is the 2D Motion Representation Convolutional Network (2D-MRCN), which specializes in motion. The 2D-MRCN takes the improved Motion History Image (iMHI) and the improved Depth Motion Map (iDMM) as inputs, which are generated from RGB and Depth data, respectively. The results of both the 3D-CDCN and 2D-MRCN are fused

in a class score fusion layer to obtain the result. The resulting model performs at state-of-the-art level without making use of deep learning.
As we focus on using RGB data only, depth based methods were not further considered in our work.

2.3 Skeleton Data

F. Baradel et al. [16] uses skeleton data and a glimp sensor for spacial attention, meaning it focuses on the hands of the 2 people performing the action in the RGB input. It also makes use of temporal attention, making the model focus more on specific hands depending on the action. With this spatial and temporal attention mechanism it extracts features from both the RGB data and the pose data. Both streams are then fused to get the results. This method gives state-of-the-art results by combining poses with hand shapes and manipulated objects.
G. Evangelidis et al. [17] created a method for continuous human action detection with skeletal data by formulating it as a labeling problem. A Gaussian Mixture Model (GMM) takes skeleton data as an input and outputs skeletal quads [18]. These skeletal quads are then encoded into a Fisher vector. Then, a multi-class SVM classifier assigns a cost per label and a global energy minimizer uses these costs to provide a piece-wise constant labeling. Although this method works for continuous gesture detection, the focus of our work, its performance compared to the state-of-the-art is unclear, as it was only evaluated in the ChalearnLAP-2014 challenge (track 3) [19] and no source code was available.

2.4 View-Invariance

A method that focuses on detecting actions from multiple viewpoints, is introduced by Y. Kong et al. [20]. It uses RGB videos of multiple views of the same action as their input. They then create a Sample-Affinity Matrix to measure the similarity between the pairs of video samples. The model attempts to learn both the shared features between the views and private features of each view. It uses an Autoencoder [21] on these features and sends them to a deep model, which outputs the detected action as a result. The model essentially tries to recognize how an action varies if a view changes, resulting in a view-invariant model that outperformed other state-of-the-art approaches at the time.
H. Rahmani et al. [22] makes use of motion capture to get skeletal data of real action sequences. It then applies these sequences to a 3D human model, that is projected to plains viewed from 108 angles. This results in 108 sequences of 2D pointclouds, which are connected sequentially to determine the trajectories that are used to train a Robust Non-linear Knowledge Transfer Model. It is then possible to extract dense trajectories from an RGB video and use these as input for the model. This view-invariant approach outperforms state-of-the-art on cross-view action recognition. This method was not considered further, since it would require motion capture equipment.
R. Bapista et al. [23] uses VNect [3] to get a 3D skeleton from RGB input. After alignment pre-processing and data expansion this data is fed into an LSTM, which gives the results. The 3D skeleton generation would currently be too computationally intensive for CHAD in live video feeds.
M.A. Musallam et al. [24] first extracts a 2D skeleton from an RGB image using AlphaPose [25]. Then a fully convolutional architecture is used to estimate the 3D skeleton from this 2D skeleton. It then uses a TCN [26] to learn temporal features directly from the 3D skeleton, thus creating a view-invariant method that performs at a state-of-the-art level.

K. Papadopoulos et al. [27] uses data adaptation to directly estimate 3D poses from RGB video. Each sequence of poses is then rotated according to the position of virtual cameras to see joint trajectories from different points of view. It then smooths the joint trajectories and feeds these into Spatial Temporal Graph Convolutional Networks (ST-GCN) [28] to get results. This leads to a model that can handle cross-view situations at a state-of-the-art level, but is computationally very demanding.

2.5 Teacher-Student

So-called teacher-student methods for HAD make use of multiple modalities during training, but are able to run with missing modalities at test time. This makes them usable on pure RGB input, while gaining an edge over pure RGB methods by having trained on extra modalities.
N. Garcia et al. [29] makes use of RGB and depth data while training, but only RGB data while testing. It first trains the RGB stream and depth stream separately, and subsequently trains them further after combining the streams with cross-entropy loss. It then uses the weights of the depth stream and attempts to learn a hallucination stream with RGB data as input to imitate the depth stream at feature and prediction layers. It then fine-tunes the model, now consisting of an RGB stream and a hallucination stream, connected with cross-entropy loss. This way the model's accuracy increases from training with depth data, but is still able to run tests with only RGB data.
N. Garcia et al. also have a more recent improved approach called Distillation Multiple Choice Learning (DMCL) [30]. DMCL makes use of 3 types of input: RGB, depth and optical flow. Each of these modalities has its own specialist of which the loss is determined for every training sample. The specialist with the lowest loss is chosen as the teacher, while the other 2 specialists are chosen as the students. The teacher distills knowledge to the students because it is incorporated in the student loss function. This way modalities get strengthened compared to only training the modalities on their own. At test time any number of available modalities can be used to get results. This method has been considered as a baseline, but was abandoned due to very high hardware requirements for training that prohibited us from validating their reported results.

For our work we want to create a CHAD method that would be usable for HCI or HRI. In these fields you have a live video feed as an input and to make your method widely usable in these fields you need to use no special hardware. This means that for our baseline method we decided to choose a method using solely RGB data, so that only an RGB camera would be needed. In the RGB methods examined, the method that seems the most promising for CHAD is Y. Shi et al.'s [9] ShuttleNet, since the accuracy reported is higher than many other state-of-the-art methods, and we were able to validate their reported results.

3 Fundamentals

Here we give some background information on important concepts used in this work. We assume that standard Neural Networks are already known. First we explain recurrent neural networks (RNNs), which are a core component of our approach for CHAD. We also describe Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRUs), which are two implementations of RNNs. After that we explain how we measure the performance of our method.

3.1 Recurrent Neural Networks

In an ordinary feed-forward neural network information moves from the input nodes, through the hidden nodes to the output nodes. Here information never visits a node twice, meaning that these feed-forward neural network have no memory of what input they received earlier and are bad at predicting what is coming next. In Recurrent Neural Networks (RNNs) recurrent layers which consist of recurrent nodes are added. The recurrent nodes are equipped with a form of memory. This allows them to not only take information from the input nodes as an input, but also their own output from the previous iteration. This makes RNNs able to recognize features in data where the order of arrival is of importance.

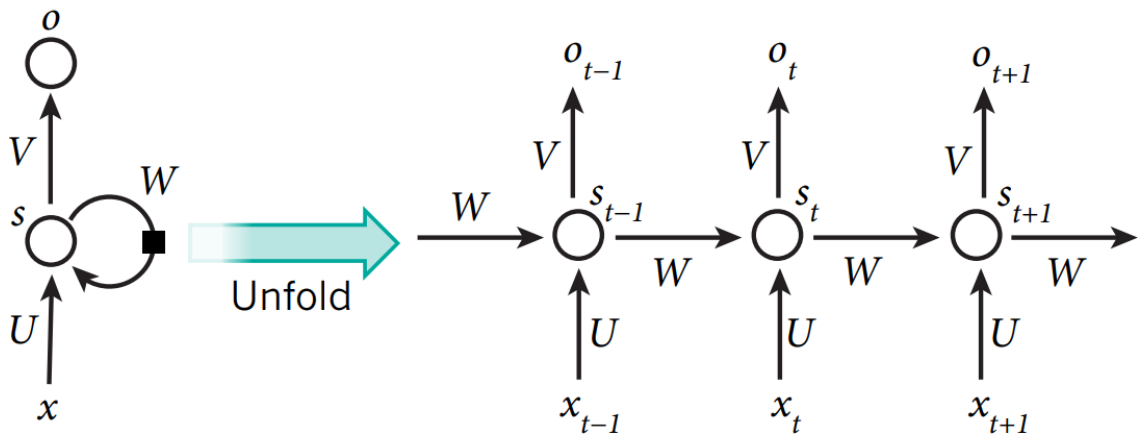


Figure 1: A recurrent neural network and the unfolding in time of the computation involved in its forward computation. (Y. LeCun et al. [31])

In Figure 1 the working of an RNN cell (left) is shown, unfolded over steps of time (right). At time t every recurrent node s_t gets input data U from the input node x_t as well as its own data W from the previous time step (s_{t-1}). It then outputs data V to the output node o_t and outputs data W to itself in the next time step (s_{t+1}).

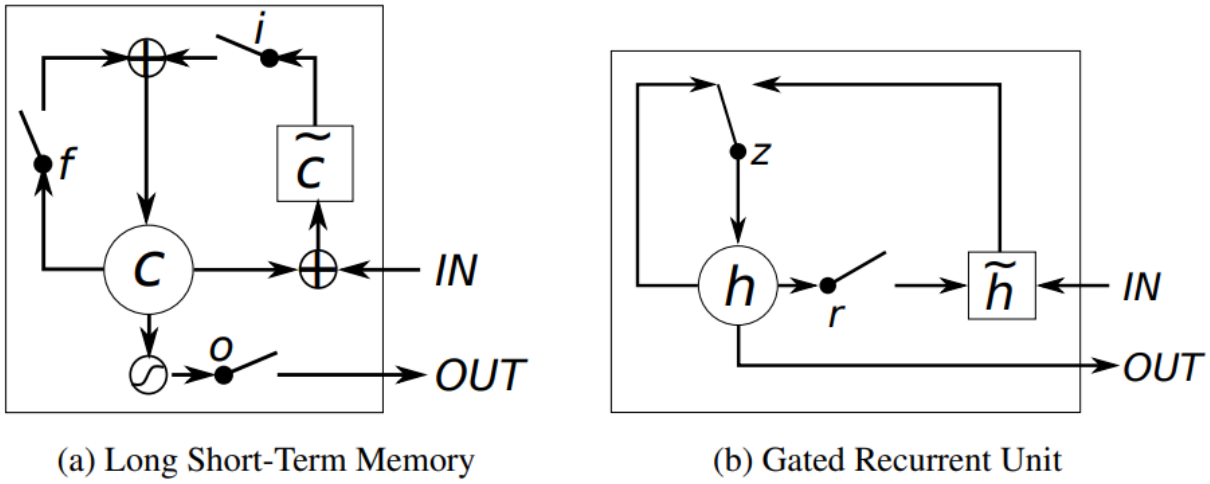
3.1.1 Long Short-Term Memory

A standard RNN has short-term memory, because it only has the capacity to carry over its recent memory. Long Short-Term Memory (LSTM) became one of the most commonly used implementations of RNNs by essentially extending the memory of RNNs. LSTMs are able to learn

from information that has very large intervals between them. The memory of an LSTM is a gated cell, where the cell itself determines which gates to open. LSTMs have 3 of these gates: An input gate, which determines whether or not to let new input into the cell, a forget gate, which can be used to delete information that is no longer important and an output gate to influence the output of the current time step. Figure 2a depicts the layout of an LSTM cell.

3.1.2 Gated Recurrent Unit

The Gated Recurrent Unit (GRU) is a newer implementation of an RNN that is similar to an LSTM. It works with gates, just like an LSTM, but it only uses 2 gates: an update gate, which decides what information to add and what information to throw away (like the LSTM's input- and forget-gates), and a reset gate, which decides how much past information the GRU should forget. Since every gate has trainable parameters, the GRU has fewer trainable parameters than the LSTM, and they are faster to train. Figure 2b depicts the layout of a GRU cell.



(a) Long Short-Term Memory (b) Gated Recurrent Unit

Figure 2: Illustration of (a) an LSTM cell and (b) a GRU cell. In the LSTM cell, i, f and o are the input, forget and output gates, respectively. c and $\tilde{c}$ depict the memory cell and the new memory cell content. In the GRU cell, r and z are the reset and update gates, and h and $\tilde{h}$ are the activation and the candidate activation. (J. Chung et al. [32])

In the architecture of our method there are two places where we use RNNs. Firstly in our sub-action specialist, for which we chose ShuttleNet(Section 4). ShuttleNet has an RNN layer in every one of its processors, which can be either a GRU or an LSTM layer. To achieve the highest accuracy with ShuttleNet, as reported by Y. Shi et al.[9], we use GRU layers here. We also use RNNs in our Action Deducer(Section 5.2). The Action deducer consists of one or multiple RNN layers followed by a dense layer. In our implementation, we opted for GRUs.

3.2 Performance

There are various ways to measure performance in CHAD. For HAD, accuracy is a widely used performance metric, but for CHAD we are not only interested in accuracy for the prediction of actions, but also for sub-actions. Furthermore top k, which is a way to make classification problems with a large number of classes computationally easier to solve. Lastly confusion matrices, can be used for both HAD and CHAD to show what labels are not predicted correctly and which labels they get confused with.

3.2.1 Accuracy

We will mainly be looking at the accuracy of the predictions the model makes. This metric is also the best to use to compare to other methods, since it is widely used. Accuracy is defined as the correct amount of predictions divided by the total amount of predictions. This is described by the following formula:

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

3.2.2 Top k

To make the classification problem of CHAD easier to solve for datasets with a large amount of classes, we make use of top k. Top k means that we do not only count a prediction as correct if the number 1 prediction is actually correct. Instead we look at the top k predicted answers. If the correct answer is in the top k predictions, we count the answer as correct. With datasets that have a large number of classes, there is a large chance of suffering of class ambiguity, where some classes are very similar to another. An algorithm would spend a lot of computational power trying to differentiate between two ambiguous classes, with next to no success. So, to make the classification problem easier to solve and to yield better accuracy, we choose to use top k. In our experiments, we make use of $k = 5$, since that is also the value used by Y. Shi et al.[9].

3.2.3 Confusion Matrix

To evaluate and get further insights in our experimental results for CHAD, we used so called confusion matrices. Confusion matrices are used to present the performance of a classification algorithm in a more detailed way. With it you are able to see what labels the algorithm has trouble assigning correctly and is confusing with other labels. This can be used to clarify situations where certain classes have a significantly lower accuracy than the average. An example of a confusion matrix can be seen in Table 1.

	Predicted Label					
Actual Label	Wave	Walk	Run	Sit	Total Predictions	Prediction Accuracy
Wave	7	0	0	1	8	87.5%
Walk	1	2	8	1	12	8.3%
Run	0	1	8	1	10	80.0%
Sit	0	1	0	11	12	91.7%

Table 1: An example of a confusion matrix for classification of 4 labels. You can see that the accuracy of the label 'Walk' is low, because it gets confused with the label 'Run'.

4 Baseline HAD: ShuttleNet

For our baseline method, we chose ShuttleNet as proposed by Y. Shi et al. [9], since this method only uses RGB data as an input and has state-of-the-art results. The overall architecture is depicted in Figure 3. Every video is split up into frames beforehand. First a random frame is chosen. Then every subsequent Kth frame is chosen as well to get a total of N frames, after which they are pre-processed. To learn features, these frames are each put into a Convolutional Neural Network (CNN), for which the authors tested both GoogLeNet [10] and Inception-ResNet-v2 [11]. While the implementation with Inception-ResNet-v2 yields better performance (94.4% accuracy on UCF101(Section 6.1)) than the GoogLeNet implementation (92.3% accuracy on UCF101), we opted to use the GoogLeNet implementation, since the Inception-ResNet-v2 implementation has high hardware requirements. They then send output of these CNNs, also called x_t, to their own ShuttleNet to learn temporal features.

ShuttleNet is a deep neural network with loop connection and a processor-sharing mechanism. As depicted in Figure 4, first the output of the CNNs (x_t) is projected by a fully-connected layer with batch normalization [33], which makes sure the inputs of the processors and hidden states have the same number of dimensions. The input is then, 1 frame per time step t, fed into all N processors named p_i in the figure. These processors are essentially RNN layers, consisting of either an LSTM or a GRU. In our implementation we chose to use GRU layers, since Y. Shi et al. also opted for GRU layers. Every step these processors work simultaneously to generate an output. Each processor then updates their personal cells and states ($h^d_{t,n}$), sends the output to another specific processor p_{i+K} based on stride K. If $i + K > N - 1$, then processor p_{i+K-N} is chosen to receive the output instead. In the figure a stride of $K = 1$ is used, which means the output is send to processor p_{i+1}. Each processor then works on the new input and this cycle goes on for D steps, after which it receives new input if there are frames left to process. This effectively creates N pathways the data travels along. After all steps are done an attention mechanism is used to select the pathway that produces the best output y_t.
In Figure 5 an illustration of ShuttleNet is shown with $N = 4$ processors, $D = 3$ steps and a stride of $K = 1$ to show how the data moves between processors over time.

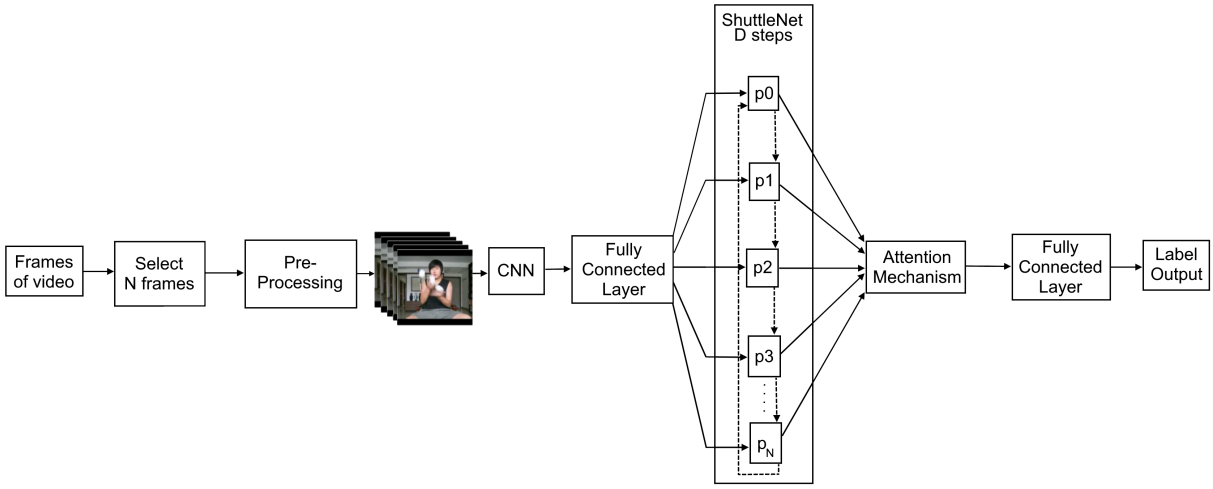


Figure 3: Diagram of Y. Shi et al. [9]. In this figure a stride of K=1 was chosen for simplicity's sake. First a random frame is chosen, after which every subsequent Kth frame is chosen as well to get a total of N frames, which are pre-processed. These frames first go through a CNN. The output then goes through a fully connected layer, after which it is inserted into all of the processors of ShuttleNet. Here it stays for D steps, moving to a different processor based on the stride after every step. This is done for every frame. The outputs of each processor are then given to an attention mechanism, which selects the pathway with the best output. The output of this pathway is put through a final fully connected layer, which outputs the activations for each label.

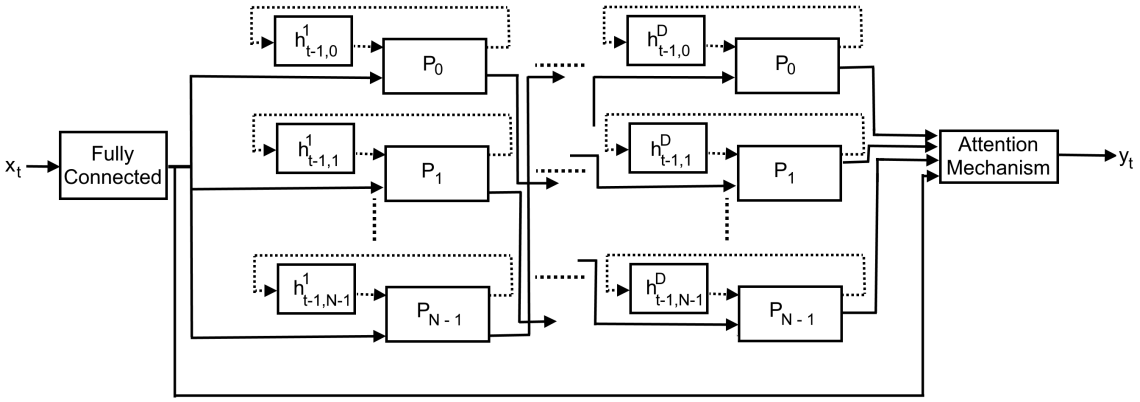


Figure 4: Diagram of ShuttleNet. Input x_t is put into a fully connected layer, which gives all N processors p_i (LSTM or GRU layer) their input. Each processor uses the input and their hidden state ($h^D_{t,n}$) to produce an output and update their hidden state. Subsequently, their output is passed to a different processor p_{i+K}, depending on the stride K. After D steps for every input, the outputs of all processors are grouped together with the output from the fully connected layer at the start and put into an attention mechanism, which determines what pathway produces the best output y_t.

We decided to use ShuttleNet for several reasons. First of all we are using this method as our Sub-Action Specialist (SAS) in our proposed method for CHAD. This means that we will be training ShuttleNet to detect sub-actions instead of actions and take the output from it to use further in our method. We will also use ShuttleNet as a baseline for HAD to test the performance on detecting both actions and sub-actions. This means that we will mainly compare the results from our own

method to results we get from running ShuttleNet on our own system to see if our proposed method is an actual improvement on ShuttleNet. We ended up choosing this HAD architecture over a state-of-the-art CHAD architecture, because there were no available CHAD methods that tested on commonly used datasets like UCF101, nor were there CHAD methods with available source code to confirm their findings.

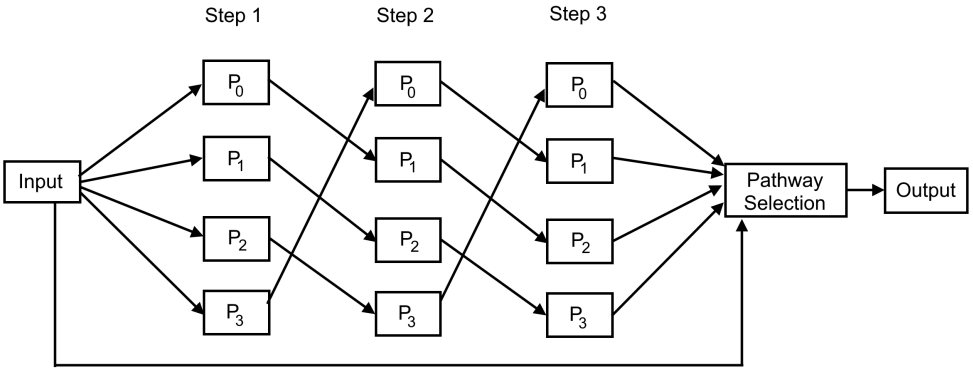


Figure 5: Illustration of a 4-processor-3-step ShuttleNet. This figure shows how the outputs move between processors with a stride of $K = 1$.

5 Method: CHAD-net

We propose CHAD-net as our method for CHAD. An illustration of CHAD-net can be seen in Figure 6. First, we split all the actions in our dataset into N equal parts, called sub-actions. Note: In our experiments we chose $N = 3$, where every video is split into 3 folders with a third of the videos frames in each. Order is taken into account when it comes to splitting up the video. Next we take all the N sub-actions of the same action and put each through their own instance of ShuttleNet(Section 4) at the same time. This has to be done to deal with the asynchronous nature of queues in the TensorFlow[34] version used by ShuttleNet. We then take the outputs of the N ShuttleNets, and provide these to our Action Deducer (AD), which predicts to what original action the detected sub-actions belong. Finally, the AD outputs activations for the labels of the original actions after processing all sub-actions. We also had the option of outputting activations before the AD has processed all sub-actions, to see if it can predict actions that take place before they are finished. This, however, is not a fair comparison with other HAD methods and therefore is not included in our experiments.

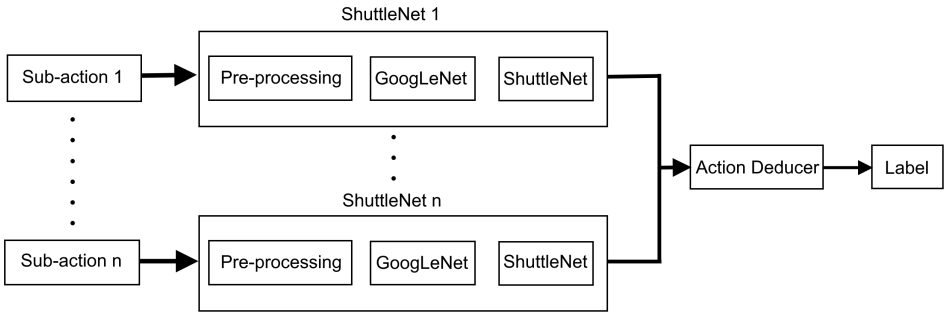


Figure 6: Diagram of CHAD-net for an N amount of sub-actions. A folder of RGB frames for each sub-action of the same action is given to a separate instance of ShuttleNet. ShuttleNet processes the data and outputs the activations for each label. The activation data for each sub-action is then combined and given to the Action Deducer. The Action deducer predicts which action the sub-actions belong to and outputs the activations for each label.

5.1 ShuttleNet

For our implementation we use a 2-processor-2-step ShuttleNet with GoogLeNet as the CNN. The 2-processor-2-step ShuttleNet was chosen because this was the implementation Y. Shi et al. [9] chose for their experiments. They found 2 processors yielded the best results and by picking 2 steps it reduced overfitting, hence improving the performance. Even though the reported accuracies of the Inception-ResNet-v2 implementation were higher, we opted to use GoogLeNet instead due to hardware limitations. The processor RNN layers in ShuttleNet will be GRUs, since that is how it was implemented by Y. Shi et al. for GoogLeNet. For our implementation we altered ShuttleNet so it handles all the sub-actions for one video in the same batch. This is done to bypass the asynchronous nature of queues in the TensorFlow version ShuttleNet used. N instances of ShuttleNet are run simultaneously, where N is the number of sub-actions in which the videos were divided. For our implementation, $N = 3$.

5.1.1 Training

For our implementation we narrowed down the options for training to the following two:

1. Train ShuttleNet first and the AD separately afterwards.
2. Train ShuttleNet and the AD together as one single network.

We chose option 1. Although option 2 might theoretically slightly increase accuracy, it also makes it hard to evaluate the strengths and weaknesses of the AD. Choosing option 1 instead allows us to evaluate the AD on a larger scale by having it as the only changing factor in our method. It also has the added benefit of only having to train the AD when testing a different amount of sub-actions, making experiments take less time, and giving the possibility of swapping out ShuttleNet for other HAD architectures.

We train ShuttleNet on different datasets for different purposes. It is trained on the following:

- **UCF101**: This is the standard trained ShuttleNet, trained on the UCF101 dataset, with 101 labels, from now on referred to as **ShuttleNet_1**.
- **UCF101_3**: This is ShuttleNet trained on the 3 sub-actions of actions within the UCF101 dataset, with 303 labels, from now on referred to as **ShuttleNet_3**.
- **C_UCF101**: This is ShuttleNet trained on the continuous actions within the UCF101 dataset, with 27 labels, from now on referred to as **C_ShuttleNet_1**.
- **C_UCF101_3**: This is ShuttleNet trained on the 3 sub-actions of actions within the C_UCF101 dataset, with 81 labels, from now on referred to as **C_ShuttleNet_3**.

An overview of the different datasets, networks and their differences can be seen in Table 2.

Network	Dataset	#Sub-actions	#Labels
ShuttleNet_1	UCF101	1	101
ShuttleNet_3	UCF101_3	3	303
C_ShuttleNet_1	C_UCF101	1	27
C_ShuttleNet_3	C_UCF101_3	3	81

Table 2: An overview of the different types of ShuttleNet. It includes what datasets they use, how many sub-actions they have and how many labels they have.

5.2 Action Deducer

The Action Deducer (see Figure 6) accepts N arrays per batch containing the activations for the labels of each sub-class as input with N being the number of sub-actions in which the videos were divided. It then outputs the predictions for the action labels. The accuracy of the predictions will be calculated based on the top k results (Section 3.2.2), as was done for ShuttleNet in Y. Shi et al. [9]. As described in Section 3.2.2, top k means checking if the correct label for an action is in the top k amount of predictions. A top k with $k = 5$ was used in ShuttleNet, so we will be using $k = 5$ as well. For our Action Deducer we make use of Keras, an API for neural networks written in python. A number of models for the AD were tested, for which an example is shown in Figure 7.

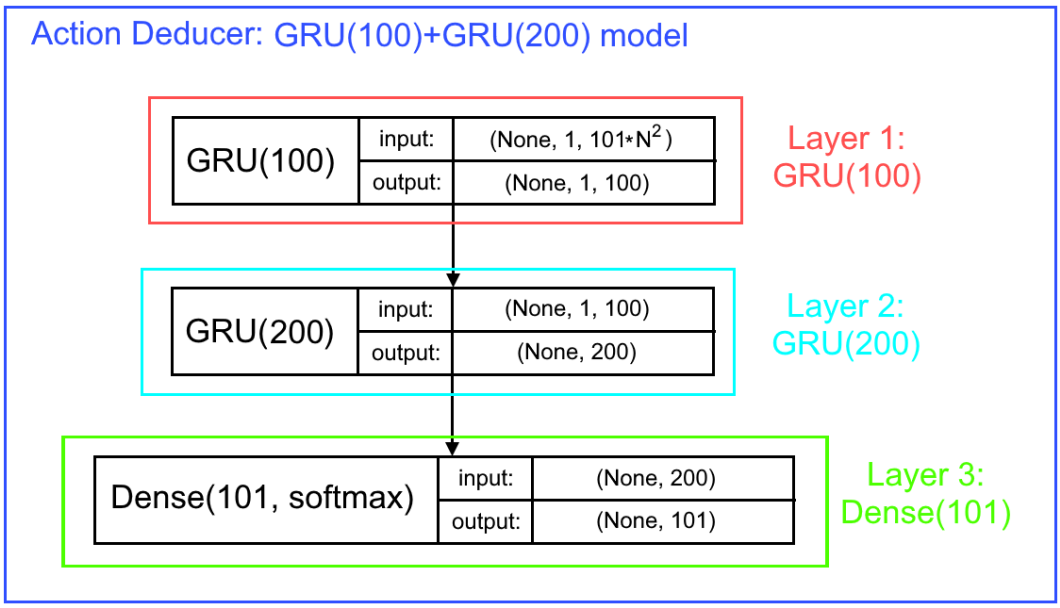


Figure 7: An instance of the Action Deducer (AD) model. The dimension of the input is based on the number (N) of sub-actions chosen. For example, if $N = 3$, then the amount of labels will be $101 * N = 101 * 3 = 303$. As there are 3 separate actions, the amount of inputs will be $N = 3$. This results in a total size of $N * N * 101 = 3 * 3 * 101 = 909$. This model uses a GRU layer with 100 units, followed by a GRU layer with 200 units and a Dense layer. The other instances of the AD model we evaluated consisted of either 1 or 2 GRU layers, with a last Dense layer to classify the inputs to 1 of the 101 labels of the original actions.

5.2.1 Action Deducer inputs

ShuttleNet handles all N sub-actions of the same action at the same time and then gives the output for all of them. As described in Table 2, ShuttleNet_3 has 3 sub-actions and 303 labels. That means as output from ShuttleNet_3 we have 3 arrays each consisting of 303 activations, one for each label. Since the AD handles all this data in the same batch, we have to combine them into a single array in a way that improves the performance of the AD. We decided to test 3 different ways of combining the data as input for the AD. We will first find the best performing AD model for each of the input types and then perform our experiments on all of them.
Here are the methods used in our experiments, with examples for ShuttleNet_3:

1. **CHAD-net-parallel**: Output combined into a single array. For ShuttleNet_3 the output would be combined into a (1,909) array.
2. **CHAD-net-sequence**: Output combined into a nested array, with each sub-action being a single array. For ShuttleNet_3 the output would be combined into a (3,303) array.
3. **CHAD-net-data**: Output, without going through the final fully connected layer in ShuttleNet, combined into a single array. For ShuttleNet_3 the output would be combined into a (1,3072) array.

A visual representation of how these inputs are created is shown in Figure 8. CHAD-net parallel (1) was chosen as one of the methods, because it is a good baseline method, showing the accuracy of a single pass of the data through the AD, therefore it could be used to evaluate the other two methods chosen. CHAD-net-sequence (2) was chosen because the AD is using GRU layers. By combining the data in this way, it will be fed to the GRU in 3 parts instead of 1, which should

improve the performance of the GRU. Finally, CHAD-net-data (3) was chosen to see if giving the model more data to work with (by skipping the fully connected layer) could improve the accuracy reached.

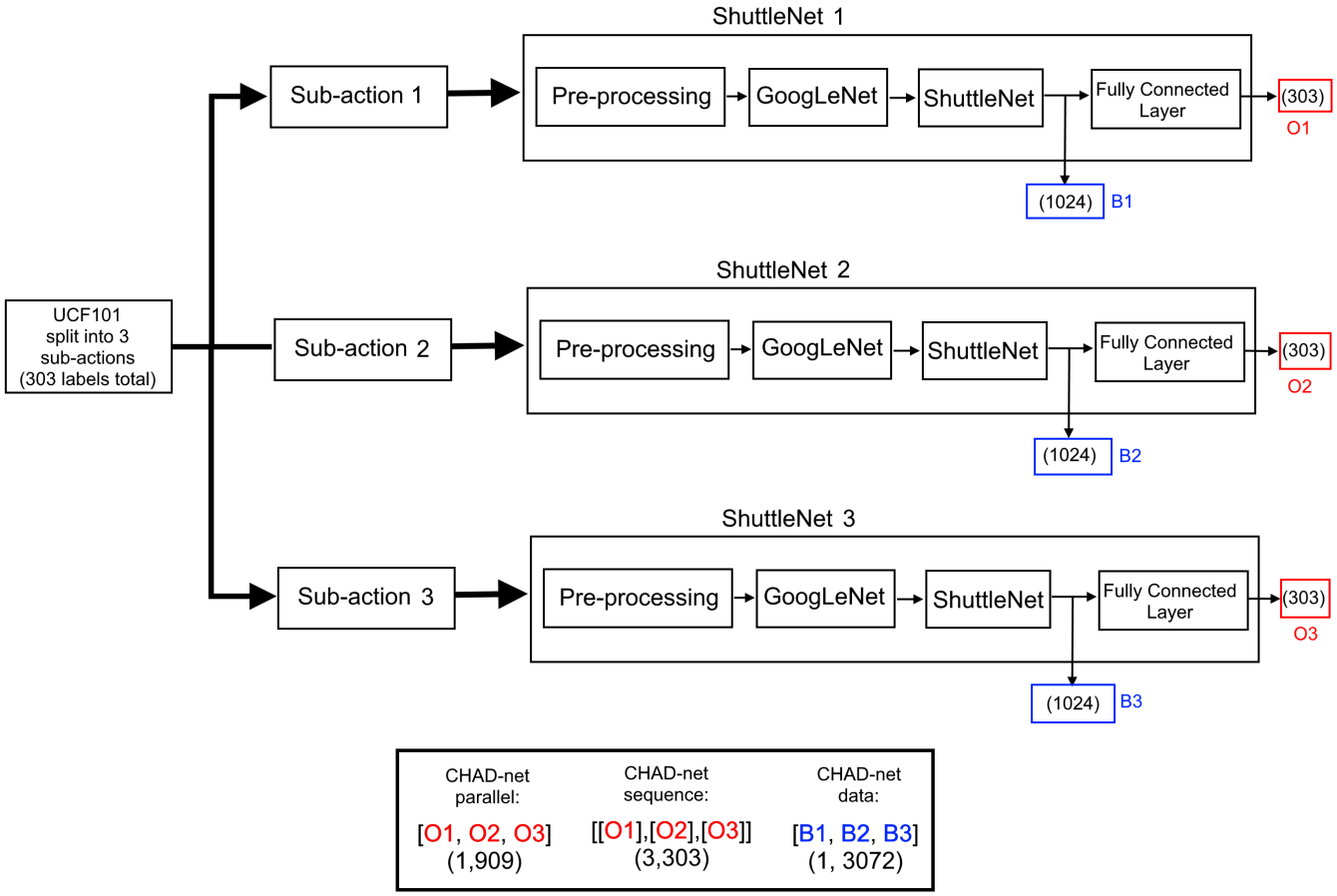


Figure 8: Example of how the different input types of CHAD-net are created.

5.3 Null-Method

Since there were no existing CHAD methods available that either tested on the UCF101 dataset or had source code available so they could be tested on it, we instead compared CHAD-net to ShuttleNet. To see if CHAD-net is an actual improvement over ShuttleNet when it comes to detecting actions we need to have something to compare it to, while using the same amount of labels as the dataset split into sub-actions. So we created the following null-method:

Take the accuracy of ShuttleNet on the dataset split into $N = 3$ sub-actions as A. The null-accuracy will then be calculated as A^3. This is the chance ShuttleNet will correctly predict all sub-actions on its own, so it will show if CHAD-net is an actual improvement.

6 Datasets

For our experiments we used the UCF101 dataset, which is a widely used dataset in HAD research. To use this dataset for CHAD-net, we split the dataset into 3 sub-actions, creating UCF101_3. To optimize UCF101 for CHAD, we made a selection of 27 of its actions and called the resulting dataset C_UCF101. To make this new dataset work for CHAD-net, we divided it up into 3 sub-actions, creating C_UCF101_3.

6.1 UCF101 & UCF101_3

UCF101[1] is a widely used dataset for HAD consisting of 101 different human actions. It has over 13.000 videos in total. Shuttlenet was originally tested on this dataset, so we decided to compare CHAD-net to ShuttleNet on UCF101. To use these videos in ShuttleNet, each video is converted into a folder containing its frames, of which some examples can be seen in Figure 9. To further use these in CHAD-net, the folders are divided into N sub-folders containing sub-actions, where the order of frames is preserved. The naming convention we will use to indicate these split versions of UCF101, will be UCF101_N, with N indicating the amount of sub-actions. If a certain amount of frames is required that these sub-folders don't have, then frames will be duplicated until the amount needed is reached. For our experiments, we decided to split the dataset into $N = 3$ sub-actions, thus creating UCF101_3, a dataset with 303 labels.

6.2 C_UCF101 & C_UCF101_3

UCF101 is a widely used dataset for HAD, but it has some flaws when it comes to CHAD. It includes many actions which do not have a clear beginning, middle and end and even actions that repeat themselves. That is why we took a sub-selection of the UCF101 labels where the actions abide to the following rules:

- The action has an obvious starting point, middle, and end-point.
- The action is non-repetitive.

This resulted in a smaller dataset, which was named C_UCF101, consisting of the 27 labels shown in Table 3. We created C_UCF101 to see if our method really performs better in situations where actions have a clear beginning-, duration- and ending-phase. This contrary to some of the actions in UCF101, that only show repetitions or continuations. To make this dataset ready for our CHAD-net experiments, we split the dataset into $N = 3$ sub-actions, thus creating C_UCF101_3, a dataset with 81 labels. A comparison between C_UCF101 and UCF101 can be seen in Figure 9, where you can see that actions without a clear beginning or end (ApplyEyeMakeup) and actions that repeat themselves (CuttingInKitchen, Rowing) were excluded from C_UCF101.

6.3 Train and Test sets

The UCF101 dataset has 25 groups of videos for each action. It is important to keep videos belonging to the same group separate in training and testing, since the videos in each group are obtained from a single long video. The UCF101 dataset already includes train and test sets for HAD, so we only

BalanceBeam	CleanAndJerk	FrisbeeCatch	ParallelBars	ThrowDiscus
BaseballPitch	CliffDiving	GolfSwing	PoleVault	UnevenBars
Basketball	CricketBowling	HammerThrow	Shotput	VolleyballSpiking
BasketballDunk	Diving	HighJump	SoccerPenalty	
Billiards	FieldHockeyPenalty	JavelinThrow	StillRings	
Bowling	FloorGymnastics	LongJump	TennisSwing	

Table 3: Labels of UCF101 included in C_UCF101.

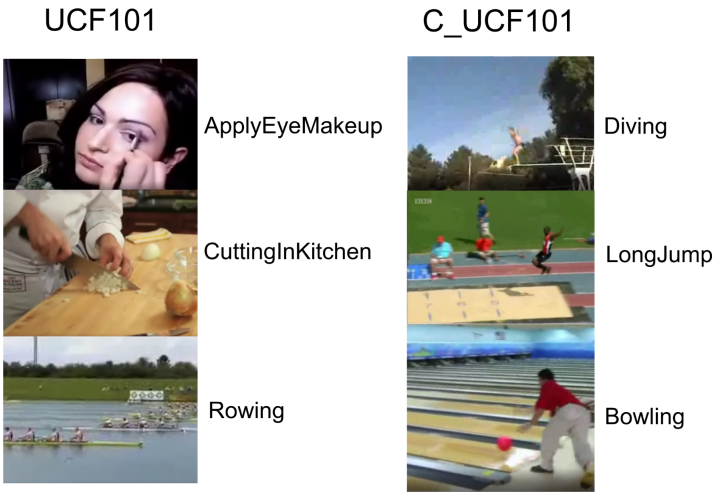


Figure 9: A selection of actions from both UCF101 and C_UCF101. UCF101 contains actions that don't have a clear beginning or end and actions that repeat themselves. C_UCF101 excludes these.

have to adjust these to only contain the 27 labels for C_UCF101. After that we take the train and test sets for UCF101 and C_UCF101 and edit them so that for every video of an action there are 3 videos of sub-actions, giving us train and test sets for UCF101_3 and C_UCF101_3, respectively.

7 Experimental Setup

Here we describe our experimental setup. First we will describe the questions we want to answer with our experiments. After that, we describe the experiments performed. For all CHAD-net-parallel, CHAD-net-sequence and CHAD-net-data (Section 5.2.1) the accuracies listed are measured with top $k = 5$ on the output of the AD after it has processed all 3 sub-actions. For ShuttleNet, accuracies listed are measured with top $k = 5$ on the output of ShuttleNet.

In our experiments, we ask the following questions:

1. What AD model yields the highest accuracy?
2. Can CHAD-net outperform state-of-the-art HAD architectures on an existing HAD dataset containing both single and repetitive actions?
3. Can CHAD-net outperform state-of-the-art HAD architectures on our CHAD dataset?

7.1 Action Deducer Experiments

To determine what AD model should be used in our final CHAD-net experiments we conducted multiple experiments on UCF101_3 (Section 6.1) for different types of inputs (Section 5.2.1). The experimental results (accuracies) are listed in Table 4.
From Table 4 we can see that the following models yield the highest accuracies for their respective input types:

- **CHAD-net-parallel**: a GRU(100) layer, followed by a Dense(101) layer with softmax.
- **CHAD-net-sequence**: a GRU(300) layer, followed by a Dense(101) layer with softmax.
- **CHAD-net-data**: a GRU(300) layer, followed by a Dense(101) layer with softmax.

These models prevent overfitting, due to the small amount of neurons and simplicity of the models, which allows them to reach the highest accuracy. Therefore, we used these models for their respective input type in the rest of our experiments.

	Units per layer			CHAD-net parallel	CHAD-net sequence	CHAD-net data
Layer Types	Layer 1	Layer 2	Layer 3	Accuracy	Accuracy	Accuracy
GRU, Dense	100	101	-	**50.56%**	54.55%	34.43%
GRU, Dense	200	101	-	49.58%	55.63%	37.84%
GRU, Dense	300	101	-	49.60%	**56.21%**	**40.01%**
2xGRU, Dense	100	200	101	37.78%	44.02%	22.08%
GRU, Dense[a], Dense	100	600	101	26.92%	31.25%	15.86%
GRU, Dense[b]	100	101	-	9.78%	11.42%	6.43%

Table 4: Accuracy of CHAD-net-parallel, -sequence and -data on the first test set of UCF101_3 (Section 6.1) for the different types of layers used in the AD. Every last dense layer, unless otherwise specified, makes use of the softmax activation function. Accuracy is based on the top $k = 5$ outputs of the AD.
[a] Dense layer is followed by a dropout(0.2) layer. [b] Dense layer does not use the softmax activation function.

7.2 HAD Experiments

For this experiment we compare the three different types of CHAD-net (Section 5.2.1) on UCF101_3 (Section 6.1) to ShuttleNet on UCF101 and the Null-method (Section 5.3) on UCF101_3. This shows whether the AD is a beneficial addition to ShuttleNet for HAD. We also compare CHAD-net to the state-of-the-art in HAD to see if CHAD-net can outperform them in accuracy. For CHAD-net-parallel, CHAD-net-sequence and CHAD-net-data we use the AD models described and selected in Section 7.1.

7.3 CHAD Experiments

In this experiment we compare the three different types of CHAD-net on C_UCF101_3 (Section 6.2) to ShuttleNet on C_UCF101 and the Null-method on C_UCF101_3. This shows if CHAD-net can outperform ShuttleNet when it comes to typical CHAD scenarios. We also compare the performance of ShuttleNet on UCF101, UCF101_3, C_UCF101 and C_UCF101_3, to see if the decrease in training data created by splitting the dataset into sub-actions leads to a decrease in accuracy. We then compare the amount of time it takes for the 3 types of CHAD-nets to see if CHAD-net is viable in scenarios where live video feeds are used as input. While this metric is ambiguous, due to the fact it depends on the hardware used, it is still important to show if CHAD-net is usable in real-time scenarios. Finally, we create a confusion matrix of the best performing version of CHAD-net on the C_UCF101_3 dataset, to gain a better insight into what actions are hard to detect. For CHAD-net-parallel, CHAD-net-sequence and CHAD-net-data we use the AD models selected in Section 7.1.

8 Experimental Results

In this section we look at the results from the experiments performed. We test CHAD-net in two different ways. Firstly we test CHAD-net on UCF101_3 (Section 6.1), to see how CHAD-net performs on a HAD dataset compared to both our baseline and the state-of-the-art in HAD. Next, we test CHAD-net on C_UCF101_3 (Section 6.2) to see its performance on CHAD. We also test the time it takes for CHAD-net to perform the classification on the second test set of C_UCF101_3, to see if CHAD-net is viable for real-time scenarios. Finally, we create a confusion matrix of the best performing version of CHAD-net on the C_UCF101_3 dataset to gain a better insight into what actions are hard to detect.

8.1 HAD

The results of this experiment are listed in Table 5. First we see that ShuttleNet has a lower accuracy on UCF101_3 than UCF101. We believe this is because the UCF101 dataset consists of a substantial amount of repetitive actions which, when divided into sub-actions, are resulting in quite similar sub-actions, i.e., no start, continuation and ending of the action. Next we can see that CHAD-net-sequence outperforms the other two iterations of CHAD-net. This shows that the (3,303) array input form makes the GRU layer work better than the other two input forms. Finally, we can see that the Null-method outperforms all versions of CHAD-net. This means that the model used for the AD does not work well enough for CHAD on the UCF101 dataset, which causes CHAD-net to perform worse than ShuttleNet and other state-of-the-art models for HAD on the UCF101 dataset, which can be seen in Table 6.

	UCF101 average	UCF101_3 average
ShuttleNet	**92.13**%	80.61%
Null-method	-	**52.38**%
CHAD-net-parallel	-	46.92%
CHAD-net-sequence	-	51.64%
CHAD-net-data	-	45.62%

Table 5: Accuracy of ShuttleNet and the 3 versions of CHAD-net for detecting actions on the UCF101 dataset. Accuracy shown is the average over the 3 test files of UCF101. Accuracy for ShuttleNet is based on the top $k = 5$ outputs of ShuttleNet. Output for the 3 versions of CHAD-net are based on the top $k = 5$ outputs of the AD.

Model	UCF101
Z. Wu et al. [35]	91.3%
LTC [36]	91.7%
sDTD [37]	92.2%
Conv Fusion [38]	92.5%
STS-ALSTM [39]	92.7%
Ge et al. [40]	92.8%
L^2STM [41]	93.6%
RNN-FV + iDT [42]	94.1%
Three-Stream TSN [43]	94.2%
DB-LSTM [8]	**97.3**%
ShuttleNet [9]	95.40%
CHAD-net-parallel	46.92%
CHAD-net-sequence	51.64%
CHAD-net-data	45.62%

Table 6: Comparison of CHAD-net to state-of-the-art methods for HAD on the UCF101 dataset.

8.2 CHAD

The results of the CHAD experiments on the C_UCF101 and C_UCF101_3 datasets are listed in Table 7. Here we see that once again, CHAD-net-sequence outperforms CHAD-net-parallel and CHAD-net-data, reinforcing the idea that the (3,303) array input form improves the GRU layer's performance. Furthermore, CHAD-net-data performs worse than the Null-method (Section 5.3) of taking the accuracy of ShuttleNet on C_UCF101_3 and cubing it, which shows that it is worse than ShuttleNet by itself.

We show ShuttleNet's performance as tested by us on UCF101, UCF101_3, C_UCF101 and C_UCF101_3 in Table 9. There we see that the performance of ShuttleNet decreases from 92.42% to 85.97%, while the amount of labels decreases from 101 to 81. This could be attributed to a lack of training data, since each label essentially has a third of the input to train on, because the C_UCF101 dataset was split into three equal parts to create C_UCF101_3. This idea is further reinforced by the fact that the accuracy of ShuttleNet on C_UFC101 is higher than the accuracy on UCF101, showing that a decrease in amount of labels should not lead to a decrease in accuracy.

We also see that, while the ShuttleNet accuracy only increased slightly from 80.61% on UCF101_3 to 85.97% on C_UCF101_3, CHAD-net's performance improved quite significantly, from 51.64% to 79.88%. This shows that CHAD-net is actually able to deduce the action based on the sub-actions.

On the second test set of C_UCF101_3, CHAD-net-sequence got a 84.57% accuracy, while ShuttleNet got a 86.94% accuracy, showing how close CHAD-net-sequence's performance can get to ShuttleNet's. Since we attributed the drop in accuracy of ShuttleNet on C_UCF101_3 compared to UCF101 (Table 9) to a lack of training data for each sub-action due to the original dataset being split in three, there

is a possibility that ShuttleNet's performance for CHAD scenarios could perform on a similar level as it does on HAD situations by increasing the amount of training data. Since CHAD-net-sequence's performance is so close to ShuttleNet's in CHAD situations, this could lead to CHAD-net-sequence giving a similar performance on CHAD to how ShuttleNet currently performs on HAD, while retaining its theoretical increased usefulness on situations where the start or end of actions are unclear.

	C_UCF101 average	C_UCF101_3 average
ShuttleNet	**92.42**%	85.97%
Null-method	-	63.54%
CHAD-net-parallel	-	67.62%
CHAD-net-sequence	-	**79.88**%
CHAD-net-data	-	49.77%

Table 7: Accuracy of ShuttleNet, the Null-method (Section 5.3) and the 3 versions of CHAD-net for detecting actions on the C_UCF101 dataset. Accuracy shown is the average over the 3 test files of C_UCF101. Accuracy for ShuttleNet is based on the top $k = 5$ outputs of ShuttleNet. Output for the 3 versions of CHAD-net are based on the top $k = 5$ outputs of the AD.

To see if CHAD-net would be usable for CHAD scenarios where live video feed is used as an input, we take a look at the time it takes for CHAD-net to produce output for a single action on the second test set of C_UCF101_3. The results of this can be seen in Table 8. From this we can see that ShuttleNet takes up the largest part of the time needed to produce an output. We also see CHAD-net-parallel has the lowest time needed for the AD, since it is the method where the AD has the least amount of neurons. CHAD-net-data has the highest amount of time needed for the AD, because of the large amount of input data in comparison to CHAD-net-parallel and CHAD-net-sequence. Since CHAD-net is able to predict at most 2.78 actions per second, it would certainly be able to handle a live video feed as input.

	CHAD-net-parallel	CHAD-net-sequence	CHAD-net-data
ShuttleNet (3 sub-actions)	0.35729 s	0.35729 s	0.35729 s
Action Deducer	0.00135 s	0.00197 s	0.00703 s
Total CHAD-net	0.35864 s	0.35925 s	0.36432 s
Predictions per second	2.78830	2.78354	2.74483

Table 8: Time it takes for the ShuttleNet, the AD and the total CHAD-net of the 3 versions of CHAD-net to predict an action in seconds on the second test set of the C_UCF101_3 dataset. The time shown for ShuttleNet is for predicting 3 sub-actions and for the AD it is for predicting 1 action. The amount of predictions per second shown is for the total CHAD-net.

The highest performing version of CHAD-net was CHAD-net-sequence on the second test set of C_UCF101_3, so a confusion matrix was created for this version. This confusion matrix can be seen in Figure 10. Here we see that CHAD-net has a hard time detecting the Basketball and the HighJump labels.

Basketball is often confused with VolleyballSpiking, SoccerPenalty, JavelinThrow, Golfswing and UnevenBars. Since the Basketball videos mostly consist of both indoor and outdoor basketball courts, where the action is a jump after which the ball is thrown, seeing it get mostly confused with VolleyballSpiking, which consists of a jump and then spiking a ball in similar areas, is not surprising. SoccerPenalty, JavelinThrow, and GolfSwing include fast moving objects from a side view, just like with Basketball, which could explain the confusion. VolleyballSpiking, SoccerPenalty and UnevenBars also have a lot of false positives, so it could be the case that the learning rate is too high for these particular labels.

HighJump is often confused with VolleyballSpiking, PoleVault, JavelinThrow and SoccerPenalty. VolleyballSpiking also includes a jump so seeing this being confused is not surprising. PoleVault is essentially the same as HighJump with an added pole, so seeing that is also not surprising. JavelinThrow and SoccerPenalty both start off with running, just like HighJump, which explains why these are primary targets for confusion.

	Amount of labels	ShuttleNet average accuracy
UCF101	101	92.13%
UCF101_3	303	80.61%
C_UCF101	27	**92.42**%
C_UCF101_3	81	85.97%

Table 9: Accuracy of ShuttleNet for detecting (sub-)actions on the UCF101, UCF101_3, C_UCF101, and the C_UCF101_3 datasets. Accuracy shown is the average over the 3 test files of each respective dataset and is based on the top $k = 5$ outputs of ShuttleNet.

Actual Label		Predicted Label																											Amount of	Prediction
Number	Label Name	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	Predictions	Accuracy
1	BalanceBeam	21	0	0	0.2	0	0	0	0.2	0	0	0	1.2	0	0	0.4	0	0	1.2	1.2	0	1.2	0	0	0	0	1.6	0.8	29	72.41%
2	BaseballPitch	0	22	0	0	0	0.2	0	0.8	0	0	0.4	0	0.2	2.2	0.2	0.2	1.8	0	0	0.2	0.8	0.4	1.6	2.4	3.2	0	2.4	39	56.41%
3	Basketball	0	0.4	4	0.6	1.2	0.2	0	0	0	0	0.8	0	0	3	0	0	3.4	0	0	0.8	0.2	3.8	2.2	2	1.6	3	5.8	33	12.12%
4	BasketballDunk	0	0	0	36	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36	100%
5	Billiards	0	0	0	0	46	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	46	100%
6	Bowling	0	0	0	0.2	0	42	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0.2	0	0	0	0.2	0.2	43	97.67%
7	CleanAndJerk	0	0	0	0	0	0	20	0	0.6	0	0	0	0	1	0	1.2	1	0.6	0.4	1.4	0.4	1.6	0.6	0.6	0.8	0.4	2.4	33	60.61%
8	CliffDiving	0	0	0	0	0.2	0	0	38	0	0	0	0	0.4	0	0	0	0	0	0	0.6	0.4	0	0	0	0.4	0.4	0.6	41	92.68%
9	CricketBowling	0	0	0	0	0	0	0	0	37	0	0	0	0.2	0.2	0	0.4	0	0	0	0	0	0.4	0.2	0	0	0.2	0.4	39	94.87%
10	Diving	0	0	0.2	0.8	0	0	0	0.6	0	36	0	0.2	0	0.4	0	0.2	0.4	0.2	0	1	0	0.8	0	0.4	0	1.4	1.4	44	81.82%
11	FieldHockeyPenalty	0	0	0	0	0.2	0	0	0	0.2	0	32	0	0.8	0.4	0	0.6	0	0.2	0	0	0.2	1	0	0.2	0	0.4	0.8	37	86.49%
12	FloorGymnastics	0	0	0	0.4	0	0.4	0	0	0	0	0	27	0.2	0.4	0	0.8	0.4	1.4	0	0.8	0.2	1.2	0	0	0	0.8	1	35	77.14%
13	FrisbeeCatch	0	0	0	0.2	0	0	0	0.2	0	0	0	0	32	0.2	0	0	0	0	0	0.2	0.2	0.2	0	0	0	0.4	0.4	34	94.12%
14	GolfSwing	0	0	0	0	0	0	0	0	0	0	0.4	0	0	31	0	0	0	0	0	0	0.4	0	0.4	0	0.4	0	0.4	33	93.94%
15	HammerThrow	0	0	0	0	0.2	0	0.2	0.2	0	0	0	0	0	0.2	34	0	0.2	0	1.4	0.4	0.2	0	1.4	1.2	2.4	1.6	2.4	46	73.91%
16	HighJump	0	1	0	2.2	1	0.4	0	0.4	0.2	0	0.4	0	0.6	0	0	1	4	2.2	0	6	1	3.2	0	0.4	2.4	1.2	6.4	34	2.94%
17	JavelinThrow	0	0	0	0.6	0.4	0.4	0	0	0	0	0	0	1.2	0.2	0	0.6	19	0.4	0	0.8	1.4	2.2	0.4	0.8	1.6	1.4	1.6	33	57.58%
18	LongJump	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0.4	0	0	0.2	35	0	1.2	1.2	1.4	0	0	1	0	1.4	42	83.33%
19	ParallelBars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33	0	0	0	0	0	0	0	0	33	100%
20	PoleVault	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0	37	0	0.4	0.4	0	0.2	0.4	0.4	39	94.87%
21	Shotput	0	0	0	0	0	0	0	0.4	0	0	0	0.2	0.4	0	0	0.6	0.2	0.2	0	0	38	0.6	0	0.2	0	0.4	0.8	42	90.48%
22	SoccerPenalty	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	0	0	40	100%
23	StillRings	0	0	0	0.2	0	0	0	0	0	0	0	0.2	0	0	0.2	0	0	0	0	0	0	0	29	0	0	0.2	0.2	30	96.67%
24	TennisSwing	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0.2	0	0	0.2	47	0.2	0	0.2	48	97.92%
25	ThrowDiscus	0	0.2	0	0.4	0	0	0	0	0	0	0	0	0	0	0.2	0	0	0	0	0.4	0	0	0	0	35	0.4	0.4	37	94.59%
26	UnevenBars	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	0	28	100%
27	VolleyballSpiking	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	37	37	100%

Figure 10: Confusion matrix of CHAD-net-sequence on test set 2 of the C_UCF101_3 dataset. Since top $k = 5$ was used, in cases where the actual label was in the top 5 the actual label was increased by 1. If the actual label was not in the top 5, all 5 predicted labels gain 0.2.

9 Conclusions and Future Research

In this paper we proposed CHAD-net, a CHAD architecture built on ShuttleNet to improve action detection in CHAD situations, so it could potentially be used on a live video feed for HRI and HCI situations. We achieved this by cutting up actions into 3 smaller segments, called sub-actions.

As demonstrated in our experiments, CHAD-net was not able to outperform ShuttleNet on HAD. Since CHAD-net was defeated by even the Null-method, this means that the AD model used was so inefficient for HAD that it lead to a decrease in accuracy. CHAD-net was, however, able to come close to ShuttleNet's performance on C_UCF101_3. Here ShuttleNet reached a 86.94% accuracy, while CHAD-net reached an 84.57% accuracy. We saw that the reason ShuttleNet has a lower accuracy on C_UCF101_3 than UCF101, while having less labels, is the fact that the amount of training data decreased by splitting each video into 3 parts. This means that the accuracy of ShuttleNet on CHAD situations can be increased with a dataset with more training data, causing the accuracy of CHAD-net to increase further.

For future research the following interesting directions can be seen:

1. The creation of a dataset focused on CHAD.
2. Improvements of the AD models.
3. Testing the predictive nature of the AD.
4. Implementations of the CHAD architecture for HRI applications.

A dataset specifically created for CHAD would have videos where the actions don't start immediately when the video starts and don't end right when the video ends. These videos would then be cut into sub-actions by hand on a frame by frame basis, which would then be manually labeled. With such a dataset it would become clear if CHAD-net is better than ShuttleNet on CHAD situations. It would also show if ShuttleNet's poor performance on C_UCF101_3 compared to UCF101 was due to a lack of training data. It would, however, require CHAD-net to work on a variable amount of sub-actions.

Experimentation with a variety of different AD models could lead to improvements for CHAD-net in both HAD and CHAD situations. Since CHAD-net's performance on CHAD is very close to that of ShuttleNet, we expect a different AD model to allow it to outperform ShuttleNet and possibly some state-of-the-art methods on CHAD. For HAD, this cannot be said for certain, since the performance of CHAD-net here was especially poor, but improving the AD would still improve the results attained.

Testing the accuracy of the Action Deducer for a certain amount of sub-actions handled would give an interesting insight into the workings of the AD. By comparing the accuracy of the AD after processing 1, 2 and 3 sub-actions, conclusions could be drawn about how well the AD is able to predict the action that it is processing before the action has fully finished. This could also be tested on a dataset with even more sub-actions. We believe the results from this could prove that CHAD-net is more useful in situations where the start or end of actions are unclear like a live video feed.

To be usable for HRI situations, CHAD-net would need to be able to handle a live video feed as

input. This would show if the AD improves time needed to fully detect an action and if it could predict an action taking place before it has actually finished. Currently CHAD-net is not able to handle a live video feed, primarily due to the older TensorFlow version ShuttleNet was built upon. This TensorFlow version works with queues, which are asynchronous, causing our method to have to include 3 versions of ShuttleNet running simultaneously instead of 1. This means that currently the time complexity of CHAD-net is 3 times that of ShuttleNet. To work for HRI, ShuttleNet would have to be altered to accept a live video feed as input and has to be updated to a more recent version of TensorFlow. We also currently run the ShuttleNets on the entirety of a dataset, before passing all the outputs to the AD. For HRI situations, the AD should be running parallel to ShuttleNet. ShuttleNet would then feed data into the AD 1 sub-action at a time on the same batch and output a result for each sub-action. For training and testing a new batch would be started for each new video, and for a live video feed a new batch would be started when sub-actions aren't detected for a period of time. This would also change CHAD-net's time complexity to be equal to that of ShuttleNet, which would lower the amount of time needed for a prediction from 0.35925 seconds (for CHAD-net-sequence) to an estimated 0.12106 seconds, thus increasing the amount of predictions per second from 2.78354 to 8.26020, further improving CHAD-net's performance on live video feeds.

References

[1] Soomro, K., Zamir, A. R., and Shah, M. Ucf101: A dataset of 101 human actions classes from videos in the wild. *Computing Research Repository (CoRR)*, 2012.

[2] Ghorbel, E., Papadopoulos, K., Baptista, R., Pathak, H., Demisse, G., Aouada, D., and Ottersten, B. A view-invariant framework for fast skeleton-based action recognition using a single rgb camera. in *Proceedings of the 14th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications - Volume 5: VISAPP*, pp. 573–582, 2019.

[3] Mehta, D., Sridhar, S., Sotnychenko, O., Rhodin, H., Shafiei, M., Seidel, H., Xu, W., Casas, D., and Theobalt, C. Vnect: Real-time 3d human pose estimation with a single rgb camera. *ACM Trans. Graph.*, 36(4):1–14, 2017.

[4] Liu, M. and Yuan, J. Recognizing human actions as the evolution of pose estimation maps. in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1159–1168, 2018.

[5] Noori, F.M., Wallace, B., Uddin, M.Z., and Torresen, J. A robust human activity recognition approach using openpose, motion features, and deep recurrent neural network. in *Image Analysis. SCIA 2019. Lecture Notes in Computer Science, vol 11482*, pp. 299–310. Springer, Cham, 2019.

[6] Cao, Z., Hidalgo, G., Simon, T., Wei, S., and Sheikh, Y. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2017*, pp. 7291–7299, 2017.

[7] Maas, K. Full-body action recognition from monocular rgb-video: A multi-stage approach using openpose and rnns. *BSc Thesis, Leiden University*, 2020.

[8] He, J., Wu, X., Cheng, Z., Yuan, Z., and Jiang, Y. Db-lstm: Densely-connected bi-directional lstm for human action recognition. *Neurocomputing*, 444:319–331, 2021.

[9] Shi, Y., Tian, Y., Wang, Y., and Huang, T. Learning long-term dependencies for action recognition with a biologically-inspired deep network. in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2017*, pp. 716–725, 2017.

[10] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. Going deeper with convolutions. in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2015*, pp. 1–9, 2015.

[11] Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. Inception-v4, inception-resnet and the impact of residual connections on learning. in *ICLR Workshop*, 2016.

[12] Das, S., Chaudhary, A., Bremond, F., and Thonnat, M. Where to focus on for human action recognition? in *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 71–80, 2019.

[13] Al-Faris, M., Chiverton, J., Yang, Y., and Ndzi, D. Deep learning of fuzzy weighted multi-resolution depth motion maps with spatial feature fusion for action recognition. *Journal of Imaging*, 5(10):82, 2019.

[14] Krizhevsky, A., Sutskever, I., and Hinton, G. Imagenet classification with deep convolutional neural networks. in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, p. 1097–1105. Curran Associates Inc., 2012.

[15] Elboushaki, A., Hannane, R., Afdel, K., and Koutti, L. Multid-cnn: A multi-dimensional feature learning approach based on deep convolutional networks for gesture recognition in rgb-d image sequences. *Expert Systems with Applications*, 139:112829, 2020.

[16] Baradel, F., Wolf, C., and Mille., J. Human activity recognition with pose-driven attention to rgb. in *BMVC 2018 - 29th British Machine Vision Conference*, pp. 1–14, 2018.

[17] Evangelidis, G., Singh, G., and Horaud, R. Continuous gesture recognition from articulated poses. in *European Conference on Computer Vision Workshops*, pp. 595–607, 2014.

[18] Evangelidis, G., Singh, G., and Horaud, R. Skeletal quads: Human action recognition using joint quadruples. in *2014 22nd International Conference on Pattern Recognition*, pp. 4513–4518, 2014.

[19] Chalearn 2014 looking at people eccv challenge. URL https://chalearnlap.cvc.uab.cat/challenge/7/description/, accessed: 28.06.2021.

[20] Kong, Y., Ding, Z., Li, J., and Fu, Y. Deeply learned view-invariant features for cross-view action recognition. *IEEE Transactions on Image Processing*, 26(6):3028–3037, 2017.

[21] Chen, M., Xu, Z., Weinberger, K., and Sha, F. Marginalized denoising autoencoders for domain adaptation. in *Proceedings of the 29th International Conference on Machine Learning*, pp. 767–774. ACM, 2012.

[22] Rahmani, H., Mian, A., and Shah, M. Learning a deep model for human action recognition from novel viewpoints. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):667–681, 2018.

[23] Baptista, R., Ghorbel, E., Papadopoulos, K., Demisse, G., Aouada, D., and Ottersten, B. 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)*, pp. 2542–2546, 2019.

[24] Musallam, M.A., Baptista, R., Ismaeil, K.A., and Aouada, D. Temporal 3d human pose estimation for action recognition from arbitrary viewpoints. in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*, pp. 253–258, 2019.

[25] Fang, H., Xie, S., Tai, Y., and Lu, C. Rmpe: Regional multi-person pose estimation. in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2017*, pp. 2334–2343, 2017.

[26] Pavllo, D., Feichtenhofer, C., Grangier, D., and Auli, M. 3d human pose estimation in video with temporal convolutions and semi-supervised training. in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) 2019*, pp. 7753–7762, 2019.

[27] Papadopoulos, K., Ghorbel, E., Oyedotun, O., Aouada, D., and Ottersten, B. Deepvi: A novel framework for learning deep view-invariant human action representations using a single rgb camera. in *2020 15th IEEE International Conference on Automatic Face and Gesture Recognition (FG 2020)*, pp. 138–145, 2020.

[28] Yan, S., Xiong, Y., and Lin, D. Spatial temporal graph convolutional networks for skeleton-based action recognition. in *Proceedings of AAAI*, vol. 35, pp. 1113 – 1122, 2018.

[29] Garcia, N., Morerio, P., and Murino, V. Modality distillation with multiple stream networks for action recognition. in *Proceedings of the European Conference on Computer Vision (ECCV) 2018*, pp. 103–118, 2018.

[30] Garcia, N., Bargal, S., Ablavsky, V., Morerio, P., Murino, V., and Sclaroff, S. Dmcl: Distillation multiple choice learning for multimodal action recognition. *2021 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pp. 2754–2763, 2021.

[31] LeCun, Y., Bengio, Y., and Hinton, G. Deep learning. *Nature*, 521:436–444, 2015.

[32] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. Empirical evaluation of gated recurrent neural networks on sequence modeling. in *NIPS 2014 Workshop on Deep Learning, December 2014*, 2014.

[33] Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. in *Proceedings of the 32nd International Conference on Machine Learning, PMLR*, vol. 37, pp. 448–456, 2015.

[34] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[35] Wu, Z., Wang, X., Jiang, Y., Ye, H., and Xue, X. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. in *Proceedings of the 23rd ACM International Conference on Multimedia*, pp. 461–470, 2015.

[36] Varol, G., Laptev, I., and Schmid, C. Long-term temporal convolutions for action recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40:1510 – 1517, 2018.

[37] Shi, Y., Tian, Y., Wang, Y., and Huang, T. Sequential deep trajectory descriptor for action recognition with three-stream cnn. *IEEE Transactions on Multimedia*, 19(7):1510–1520, 2017.

[38] Feichtenhofer, C., Pinz, A., and Zisserman, A. Convolutional two-stream network fusion for video action recognition. in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016*, pp. 1933–1941, 2016.

[39] Liu, Z., Li, Z., Wang, R., Zong, M., and Ji, W. Spatiotemporal saliency-based multi-stream networks with attention-aware lstm for action recognition. in *Neural Computing and Applications*, vol. 32, pp. 14593 – 14602. Springer, Cham, 2020.

[40] Ge, H., Yan, Z., Yu, W., and Sun, L. An attention mechanism based convolutional lstm network for video action recognition. *Multimedia Tools Appl.*, 78(14):20533—-20556, 2019.

[41] Sun, L., Jia, K., Chen, K., Yeung, D., Shi, B., and Savarese, S. Lattice long short-term memory for human action recognition. in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) 2017*, pp. 2147–2156, 2017.

[42] Lev, G., Sadeh, G., Klein, B., and Wolf, L. Rnn fisher vectors for action recognition and image annotation. in *Proceedings of the European Conference on Computer Vision (ECCV) 2016*, pp. 833–850, 2015.

[43] Wang, L., Xiong, Y., Wang, Z., Qiao, Y., Lin, D., Tang, X., and Van Gool, L. Temporal segment networks: Towards good practices for deep action recognition. in Leibe, B., Matas, J., Sebe, N., and Welling, M., eds. , *Proceedings of the European Conference on Computer Vision (ECCV) 2016*, pp. 20–36, 2016.