Using transfer learning on 2D skeleton-based action recognition networks to improve performance on data from previously unseen camera angles

Filip Cederquist & Gustav Harrysson

Master's thesis
2021:E16

Lund University

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Centrum Scientiarum Mathematicarum

Abstract

Action recognition is a task in computer vision of inferring an action performed by a subject given an image or video. In this thesis we looked at how the performance of an action recognition network changed when introduced to new angles and how incorporating that data in the training affects this performance. A state-of-the-art skeleton extraction network HRNet_w48+DARK was used together with a large action recognition dataset NTU RGB+D to create a baseline dataset containing skeleton data and actions labels. A new dataset was recorded which introduced a shift in vertical view point angle, which was used for analysis. Two different action recognition methods were used and compared to broaden the analysis. Results show that a small addition of 5-10% of the original amount of data from the new angles are enough to increase the accuracy on those angles greatly. The accuracy on the previous angles decreases but only by a small margin compared to the increased accuracy on the new angles. This implies that an extension of an action dataset to include another angle is a feasible task not requiring a large amount of data.

Contents

Chapter 1

Introduction

1.1 Background

Computer vision based human action recognition is a widely researched field and has many applications such as video surveillance, robotics and human-computer interactions. This is performed by analyzing images containing people and extracting information which is then used to infer the action performed by the subject.

Many different types of data can be utilized for this purpose, but the information about the subject examined in this thesis will only be the skeleton of the subject. This is extracted by using an artificial neural network (ANN) specialised in human pose estimation. Which is a task in computer vision in which the skeleton of the targeted person is estimated. This is done by detecting keypoints on the body and connecting them to each other to form a skeleton with the topology of a human body.

With a skeleton representation of a human, patterns can be detected by observing the joints and their connections both in the single-frame case and over multiple frames. These patterns can be learned by another ANN specialized in action recognition to predict the action performed by the person.

1.2 Goal and task

The overarching goal of this thesis is to improve on a skeleton-based action recognition network to perform better on data from a novel viewpoint.

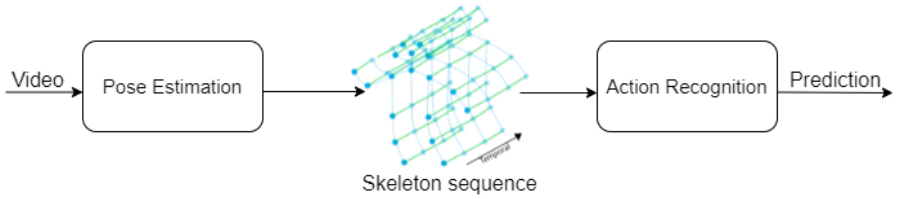


Figure 1.1: Pipeline from video to action prediction.

For neural networks, the data used for training is extremely important, without a well put together dataset it can be difficult to get the desired results. Many networks perform poorly when asked to perform on data unlike anything that it has seen before. In this thesis, the goal is to explore how a change in vertical angle from the original 0° vertical angle as seen in figure 1.2 a to the vertical angles in figure 1.2 b affects performance. Additionally, examine how adding different amounts of data from the new angles changes the performance on both the new vertical angles and the previous 0° ones.

The general pipeline we will use is illustrated in figure 1.1, a video file will be used as input to a pose estimation network that locates keypoints on the human body for every video frame. These sequences of keypoints will subsequently be used as input for an action recognition network that will attempt to classify the keypoint movement over time as a specific action.

To achieve this goal, a new dataset will be recorded from different vertical angles (figure 1.2 b) than the original dataset we have chosen [32] (figure 1.2 a) and serve as a performance benchmark. Data from the novel angles will be used to retrain the action recognition network with the goal of improving performance from the novel angles while not losing performance on the original angles.

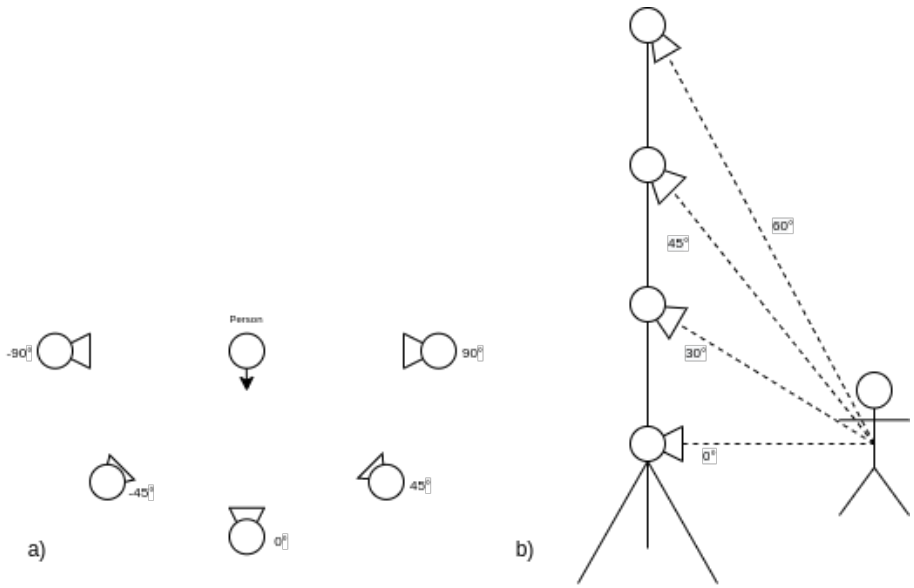


Figure 1.2: a. Camera setup used in the dataset NTU RGB+D seen from above [32]. b. Our camera setup seen from the side.

Chapter 2

Deep learning

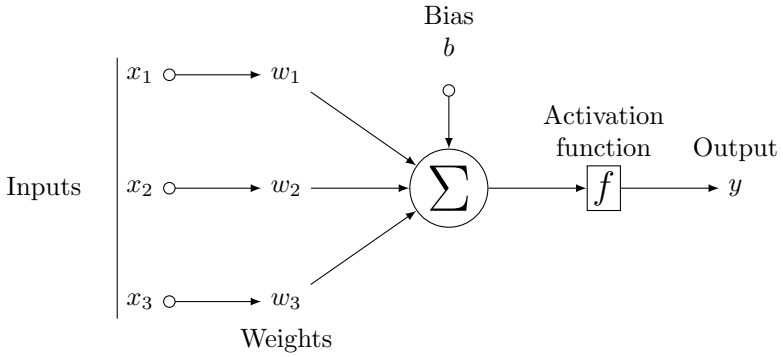
The basis for a deep learning model is an artificial neural network (ANN). An ANN is built up of layers of perceptrons. A perceptron is a node which receives input signals in the form of real numbers. Each input to the perceptron is associated with a weight. All the inputs are aggregated together with a bias before a non linear activation function feeds forward the output, see figure 2.1.

For the figure below demonstrating a single perceptron with 3 inputs, the output y is calculated as

$$y = f(\sum_{i=1}^{3} x_i w_i + b) \tag{2.1}$$

where f is a non linear activation function, w_i is the i'th weight, x_i is the i'th input and b is a bias. By combining several perceptrons and more layers. A neural network is created. Since the information only travels forward, this is called a feed forward neural network.

Figure 2.1: Perceptron.



The feed forward neural network is the quintessential deep learning model. Deep learning is a subfield of machine learning which aims to approximate some function f^*. For example, a classifier maps an input $\mathbf{x}$ to an output y through $y = f^*(\mathbf{x})$. A feed forward network wants to find the best approximation of $y = f(\mathbf{x}, \theta)$ by learning the parameters θ. These are called networks since they are most often represented by composing several functions. The depth of a model depends on how many of these functions are chained together [10]. A network takes an input to the first layer, the output of this layer is then sent to the next layer and so on. In the example of categorization with k different categories, the last layer has the size of all the different possible outputs $\mathbf{y} = (y_0, y_1, ..., y_k)$ and the output could be the networks best guess y_i for the given input.

2.1 Activation function

An activation function defines the output of a neuron when given inputs. There are several kinds of activation functions that are commonly used and some are explained below, as well as when they are most commonly used. In the middle layers of a network, a common activation function used is the Rectified Linear Unit (ReLU) [26]. This operation takes the input from a node and feeds forward the information if it is above 0. This is mathematically expressed as

$$f(x) = \max(0, x) \tag{2.2}$$

where x is the input to a node.

At the end of a neural network, an activation function called the softmax is often used. This is a normalized exponential function which transforms the output $\mathbf{z}$ of a network to a distribution over the predicted output classes. This softmax function σ is expressed as

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=0}^{K} e^{z_j}} \quad \text{for} \quad i = 1, ..K, \quad \text{and} \quad \mathbf{z} = (z_1, ..z_k). \tag{2.3}$$

In the case of a binary classification the sigmoid function $S(x)$ can be used which maps an input x to a value between 0 and 1. This is defined as

$$S(x) = \frac{1}{1 - e^{-x}} \tag{2.4}$$

where the threshold to determine if it belongs to the first or second class is $\frac{1}{2}$.

2.2 Loss function

A loss function is a function that maps a value or values to a real number to determine the cost of achieving that value compared to a desired output. A

standard loss function is the L2 loss function

$$L = \sum_{i=1}^{K} (y_i - \hat{y}_i)^2 \tag{2.5}$$

where L is the loss, y_i are the predicted values and $\hat{y}_i$ are the true values.

Another standard which is often used is the cross entropy loss, expressed as

$$L = -\sum_{i=1}^{K} (\hat{y}_i \log(y_i)). \tag{2.6}$$

Cross entropy is often used in classification where the output is a probability distribution. When the predicted value diverges from the true value, the cross entropy loss increases.

2.3 Back propagation

When calculating the gradient of the loss function backwards through the network, many partial derivatives must be calculated. A method for this is back propagation. Back propagation in a neural network is the calculation of the derivative of the cost function C with regards to weights and biases. The only variables in the network are the weights $\mathbf{w}$ and biases b, hence the derivative searched for is $\frac{\partial C}{\partial w_i}$ and $\frac{\partial C}{\partial b}$.

To understand how this works, lets look at the output, i.e activation a, of neuron j in a layer l, with and activation function σ defined as

$$a_j^l = \sigma(\sum_k w_{jk} a_k^{l-1} + b_j^l). \tag{2.7}$$

This equation describes a summation of all previous activations $\mathbf{a}^{l-1}$ multiplied with the weights to neuron j, w_j plus the bias for the neuron in the current layer b_j^l. In matrix form the activation of an entire layer can be written as

$$\mathbf{a}^l = \sigma(\mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l) \tag{2.8}$$

where the function σ is performed on each element of $\mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ in this vectorized notation. With the notation $\mathbf{z}^l = \mathbf{w}^l \mathbf{a}^{l-1} + \mathbf{b}^l$ we have

$$\mathbf{a}^l = \sigma(\mathbf{z}^l). \tag{2.9}$$

We can now define the equations that are used in back propagation where L is the last layer of the network.

$$\delta^L = \frac{\partial C}{\partial a_j^L}\sigma'(z_j^L) \tag{2.10a}$$

$$\delta^l = ((w^{l+1})^T\delta^{l+1} \odot \sigma'(z^l) \tag{2.10b}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{2.10c}$$

$$\frac{\partial C}{\partial w_{jk}^l} = a_k^{l-1}\delta_j^l \tag{2.10d}$$

where 2.10a describes the error of the last output layer and 2.10b describes the error of the hidden layers. Equations 2.10c and 2.10d describe the derivative of the cost function with respect to the bias and weights of the network. The operator $\odot$ is element-wise multiplication.

These function are the backbone of back propagation. To see more derivations of the equation and more detail on the algorithm, see [29] or in the 1986 paper [31].

2.4 Convolutional neural networks

A convolutional neural network (CNN) is a network often used in image analysis which uses the convolution operation in at least one layer. A convolution is an operation $s(t) = f * g$ where

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x)g(t-x)dx. \tag{2.11}$$

The expression can be thought of as a weighted average of the function $f(x)$ at the moment t of the function $g(-x)$ but g is shifted by a value t. In machine learning, the function f is often called the input I and the function g is the kernel K. In applications such as image analysis, this kernel is two dimensional, discrete and finite, resulting in the above expression translating to

$$(I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.12}$$

where m and n range over the values of the size of the kernel and (i,j) is the pixel position. Convolution is commutative, and in a machine learning setting it is often easier to implement the convolution in the following way

$$(K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{2.13}$$

since there is less variation in the values of m and n due to kernels often having a size of 3×3, 5×5, or 7×7.

When using convolution on an input image, the kernel slides across the entire image, and together with a non linear activation function and a pooling operation, creates a new image as output. The pooling operation is often max pooling or average pooling. In max pooling the maximum value of the convolution becomes the output for the corresponding coordinate. In average pooling, the average of the output of the convolution is used.

Often times the kernel is much smaller than the input. In a network, the values of the kernel are parameters that are learned, making the network find features on its own. The convolutional operation for an input differs depending on a set of hyper parameters;

The size of the kernel. This determines how big area around the central pixel should be used.

Padding is relevant in the edges and corners of an image where a number of zeros are added. A padding is introduced to be able to centralize around the pixels on the edge of the input and perform convolution there as well.

Stride determines how much the kernel moves to the right and down through an image. The bigger the stride, the less convolutions are made.

The number of channels determines how many kernels are used. This is used to make each kernel learn different features of an image. For example one kernel can identify round forms while another finds straight lines.

Padding, stride and kernel size determine how big the output is before the pooling layer. The reduction in output size is often done to reduce the feature space of the input.

2.5 Graph Convolutional Networks

A graph convolutional network is a form of neural network which utilizes the connections between nodes as well as the information of the nodes themselves [17]. A regular feed forward neural network has a layer wise propagation rule

$$H^{(l+1)} = \sigma(H^{(l)}W^{(l)}) \tag{2.14}$$

where $W^{(l)}$ is a trainable weight matrix in layer l, σ is an activation function, $H^{(l)}$ is the matrix of activations in layer l where $H^{(0)} = X$ and X is the input to the network.

In a graph convolutional network, an adjacency matrix A is integrated which contains information not present in the specific data points, but rather the connections between them. As an example, the x- and y coordinates of a hand and

an elbow describes the data, while the adjacency matrix contains information that these two data points are connected to each other on the human body. The adjacency matrix is a square matrix of size $n \times n$ where n is the size of the input X. The value of $A(i, j)$ describes the relation of node i to node j. These values are binary if they only describe that there exists a connection between the nodes, and contain real values if the strength of the connections is to be taken into account. The integration of this adjacency matrix changes the layer wise propagation rule to

$$H^{(l+1)} = \sigma(AH^{(l)}W^{(l)}). \tag{2.15}$$

Often the unit matrix is added to the matrix A to include self-connections, this new matrix is defined as $\tilde{A} = A + I_n$. To normalize these connections by the neighborhood sizes, the number of connections for each node is counted and put into the diagonal of another $n \times n$ matrix D. The function $\tilde{A}$ is squashed between the inverse root of this matrix D to normalize the values in the adjacency matrix by the neighborhood sizes of the nodes being connected. This results in the final version of the adjacency matrix being used is

$$\hat{A} = D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}} \tag{2.16}$$

thus making the layer wise propagation rule for a graph convolutional network to be

$$H^{(l+1)} = \sigma(D^{-\frac{1}{2}}\tilde{A}D^{-\frac{1}{2}}H^{(l)}W^{(l)}). \tag{2.17}$$

2.6 Regularization

Regularization is the technique of reducing testing error on a machine learning algorithm. A regularization technique often generalizes the network to reduce overfitting. Below are a couple of examples of regularization.

Dropout is a technique to make the network more robust [10]. In each layer of a network, some neurons outputs are set to zero with a probability $p \in (0, 1)$. This probability is a hyper parameter and does not need to be the same for each layer. The updating of the gradient is often not done after each new training sample but after a set number of samples, called a batch, often with a size of 32 or 64. For each batch, this process is done which means that after each batch, a new set of neurons and activations decide the final output of the network. This will reduce overly relying on certain activations and cause the model to be more general.

Batch normalization [14] is another technique which primarily improves optimization but a pleasant side effect is a regularizing effect. When upgrading the weights in a network, all layers are updated under the assumption that the other layers do not change. When updating all weights simultaneously, unexpected results may occur due to this assumption. To reduce the effect an update to

previous layers has on a future layer, batch normalization aims to normalize the output of the activations of each layer by introducing learnable variance and bias parameters γ and β.

Let the matrix H represent the activations of one layer from a certain batch. Each row in H corresponds to the activations for an example in the batch. Then we define the mean and standard deviation of H as μ and σ. The output H is replaced by

$$H' = \frac{H - \mu}{\delta}. \tag{2.18}$$

The output data is now normalized, however normalizing the output might reduce the expressive power of the network. To combat this, the previously mentioned learnable parameters γ and β replaces the output H' with

$$H'' = \gamma H' + \beta. \tag{2.19}$$

This new parameterization is easier to learn with gradient descent which is explained below and with the noise of varying mean and standard deviation between batches, it has a regularization effect [10].

2.7 Optimizers

When a network learns, it updates the weights to take a step in a direction which reduces the loss. A standard way of doing this is with gradient descent. This method works by taking a step in the direction of the negative gradient of the loss function at the current weight values $\mathbf{w}$ and updating $\mathbf{w}$ with the new value. The new weight $\mathbf{w}_{n+1}$ is updated by

$$\mathbf{w}_{n+1} = \mathbf{w}_n - \gamma \nabla L(\mathbf{w}_n) \tag{2.20}$$

where γ is the step size (also called learning rate), $\mathbf{w}_n$ are the current weights and $\nabla L(\mathbf{w}_n)$ is the gradient of the loss function with regards to the weights.

2.8 Data processing

Before a network can be trained the data must follow a specific structure, often limited to the required structure of the network. For example, the structure required for a Resnet [11] is an image with three depth channels, often rgb-values and a height and width in pixels. The height and width is recommended to be above 224×224 for the network to perform in a desired manner. A higher resolution image might yield more information but also requires more memory and longer computation time.

Another processing step is normalization. This is the process of normalizing the values of the inputs to a set interval, often between 0 and 1. This reduces error due to large value differences in the input.

2.9 Hyper parameters

When training a network, weights are parameters that are updated during the training. Hyper parameters are parameters that must be set prior to the training. These are typically learning rate, batch size or the algorithm used to minimize the loss function. It is hard to determine what values of the hyper parameters will yield the best model performance. A method for finding the hyper parameters that work best for the current problem is to use a grid search. This works by simply testing different combinations and see what works best [41].

2.10 Transfer Learning

Transfer learning is the method of training a network to complete a task in a new but similar domain as the network is initially trained on. For instance, a network trained to identify images of cats and dogs can be transfer learned to identify butterflies. Transfer learning is often used when the amount of data in the new domain is small and there exists a larger dataset that can be used to extract information relevant to your new problem [44].

When retraining a network to solve a new problem, there are usually two methods used. Deep retraining and shallow retraining. The difference between these are how many of the layers are updated with the new data. In deep retraining, all weights in the network are updated when training. In shallow retraining, only the last or the few last layers are trained. The rest of the network has its weights frozen. The reason behind this is that the first layers of a network often extracts low and mid level features such as edges or pixel intensities which are still useful for the new domain. Deep retraining is used when the input of the new data differs vastly from the dataset the network as initially trained on.

Chapter 3

Human pose estimation

Human pose estimation is a common task in computer vision where one attempts to predict an object's location and orientation, it is most often performed on a human, which will be our focus in this project. The first well-known instance of what is today referred to as human pose estimation can be traced back to 1973 when Fishler and Elschlager proposed a technique known as Pictorial Structures [9]. With the advent of Deep Learning, the efficacy of human pose estimation has greatly improved and Convolutional Neural Networks (CNN) is the primary type used for pose estimation today.

More practically, human pose estimation is performed by locating keypoints and other visual cues on the human body which are used in combination to form a skeleton that is the pose. Multiple different setups of joints have been used and they can differ both in the number of joints and the selected points on the bodies, one example is illustrated in figure 3.1.

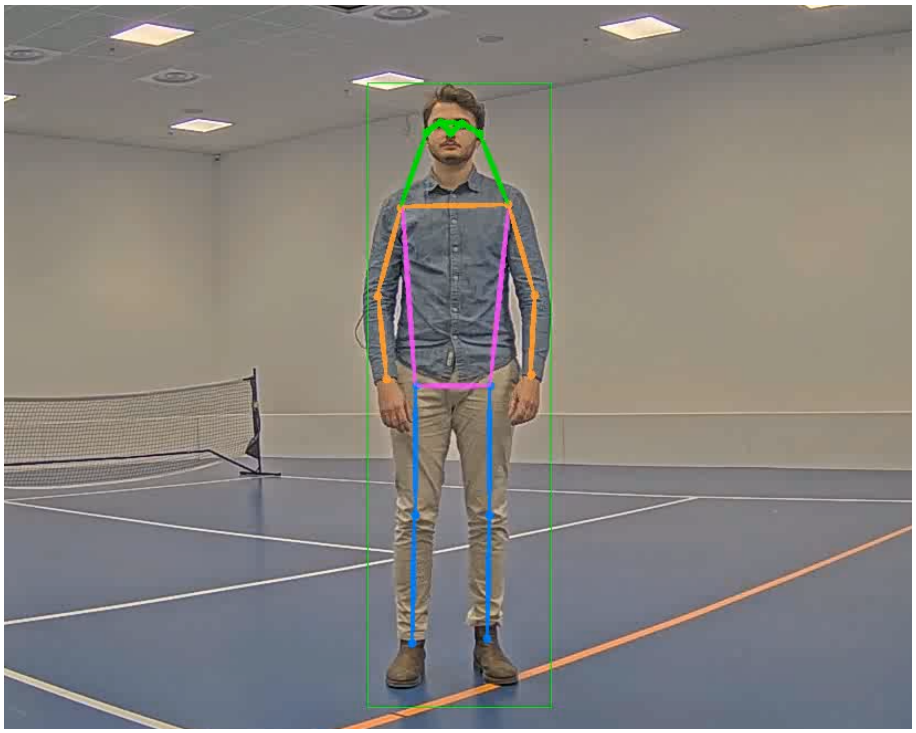


Figure 3.1: Keypoint setup used in the COCO [23] dataset.

To get a better understanding of how we performed pose estimation and what is possible, what follows in this chapter is an explanation of various pose estimators and datasets that can be used.

3.1 Datasets for human pose estimation.

In order to create a pose estimator, a dataset has to be used for training. Datasets produced for pose estimation are collections of videos or images with coordinates marking the joints of the people present in the scene. As previously mentioned, there is no standard for the amount of keypoints per body as well as the setup of those keypoints. For example, some datasets only contain a single keypoint for the entire head, while others contain as many as 5 keypoints for the head.

3.1.1 MPII Human Pose

MPII [2] is a dataset that consists of images taken of humans during various real-world activities, complete with full-body pose annotations. It contains around 25K images of over 40K people, each with 16 annotated body joints, some examples can be seen in figure. In addition to being annotated with pose data, the action performed by the subject is labeled as well.

3.1.2 COCO (Common Objects in Context)

COCO [23] is a widely used dataset for a variety of tasks. Including object detection, segmentation and keypoint detection. In the keypoint task, it consists of over 200K images and 250K subjects, labeled with 17 keypoints per person.

3.1.3 PoseTrack

PoseTrack [1] is a dataset that consists of video sequences as opposed to images. It contains 1356 video sequences which are made up of 46K annotated video frames and 276K body pose annotations. In addition to enabling the use of multi-frame pose-estimation, it enables the use of pose tracking over time.

3.2 Performance metrics

To evaluate the performance of human pose estimation is not trivial, some evaluation metric is necessary to measure the degree of error appropriately. In the following subsections a few of the most common performance metrics used will be explained.

3.2.1 Percentage of Correct Key-points - PCK

A keypoint detection is considered correct if the predicted position and the true position is within a certain distance of each other. This distance can be set a number of ways, a common choice is a percentage of the head diameter. If the head diameter is used. This measurement is referred to as PCKh [25]. PCKh-0.5 refers to half of the head diameter and is default metric for performance evaluation on the MPII dataset. Used by MPII [2] and PoseTrack [1].

3.2.2 Object Keypoint Similarity - OKS

Measures how close the predicted keypoints are to the ground truth for keypoints i.

$$OKS = \frac{\sum_i exp(-d_i^2/2s^2k_i^2)\delta(v_i > 0)}{\sum_i \delta(v_i > 0)}. \tag{3.1}$$

d_i - the Euclidean distance between the predicted keypoint and the true position.
s - object scale.

k_i - per-keypoint constant that controls falloff.
v_i - visibility flag of ground truth. An OKS-value of 1 means a perfect prediction.

Using OKS, the performance can be measured by average precision (AP) and average recall (AR).

Precision measures the percentage of correct predictions.

$$Precision = \frac{\text{True positive}}{(\text{True positive}) + (\text{False positive})} \tag{3.2}$$

Recall measures how well the positives are found.

$$Recall = \frac{\text{True positive}}{(\text{True positive}) + (\text{False negative})} \tag{3.3}$$

The metrics listed below are used for performance evaluation on the COCO challenge leaderboard[1].

- **AP**, averaged over multiple OKS values (0.50:0.05:0.95)
- $\mathbf{AP}^{50}$, AP for OKS=0.50
- $\mathbf{AP}^{75}$, AP for OKS=0.75
- **AR**, averaged over multiple OKS values (0.50:0.05:0.95)
- $\mathbf{AR}^{50}$, AR for OKS=0.50

3.3 Networks for human pose estimation.

One way in which to detect the pose of a human is by detecting a person first and then detecting the various joints of that person. This is often referred to as the top-down approach. Since the top-down approach has to detect people first, it requires a person detector which divides the pipeline into two parts, one for person detection and one for the keypoint detection.

Another possible way of performing human pose estimation is by flipping the pipeline from the top-down approach. Detecting every instance of each type of joint in an image first, the body pose can then be stitched together by pairing joints together in a way that conforms to the topology of the human body. This type of model is referred to as bottom-up. Currently, the best performing models are all using the top-down approach over the bottom-up approach[2][3][4]. Below

[1] https://cocodataset.org/#keypoints-eval
[2] https://cocodataset.org/#keypoints-leaderboard
[3] http://human-pose.mpi-inf.mpg.de/#results
[4] https://posetrack.net/leaderboard.php

follows a performance comparison and explanation of various pose estimation networks.

COCO test-dev (top-down)						
Model	Input Size	AP	AP^{50}	AP^{75}	AR	AR^{50}
CPM[39]	256x192	0.623	0.859	0.704	0.686	0.903
CPM[39]	384x288	0.650	0.864	0.725	0.708	0.905
HrNet_w48[38]	256x192	0.756	0.907	0.825	0.806	0.942
HrNet_w48[38]	384x288	0.767	0.910	0.831	0.816	0.946
HrNet_w48+ DARK[38][42]	256x192	0.764	0.907	0.830	0.814	0.943
HrNet_w48+ DARK[38][42]	384x288	0.772	0.910	0.836	0.820	0.946
Stacked Hourglass[28]	384x384	0.746	0.900	0.813	0.797	0.939
4-stage MSPN[22]	256x192	0.764	0.906	0.835	0.826	0.944
HrNet_w48 _udp[38][13]	384x288	0.772	0.910	0.835	0.820	0.945

Table 3.1: Results for various top-down networks on the COCO val2017 using person detector with AP of 56.4 on COCO val2017 dataset.

COCO test-dev (bottom-up)						
Model	Input Size	AP	AP^{50}	AP^{75}	AR	AR^{50}
HigherHRNet-w32[6]	640x640	0.686	0.871	0.747	0.733	0.898
HigherHRNet-w48[6]	512x512	0.686	0.873	0.741	0.731	0.892
AE+HRNet-w32[27][38]	512x512	0.654	0.863	0.720	0.710	0.892
AE+HRNet-w48[27][38]	512x512	0.665	0.860	0.727	0.716	0.889
HRNet-w48 _udp[38][13]	512x512	0.681	0.872	0.741	0.725	0.892
HigherHRNet-w48_udp[6][13]	512x512	0.690	0.872	0.750	0.734	0.891
OpenPose[3]	1312x736	0.642	0.862	0.701	-	-

Table 3.2: Results for various bottom-up networks on the COCO val2017 without multi-scale test.

Our choice of pose estimation network, HRNet_w48+DARK is further explained below. Brief explanations of the other networks referenced in the tables 3.1 and 3.2 can be found in appendix A.

3.3.1 Deep High-Resolution Representation Learning for Human Pose Estimation (HRNet)

HRNet [38] is an architecture which combines low- and high-resolution convolutional networks. It takes an image of size $W \times H \times 3$ and aims to estimate K heatmaps, one for each keypoint, resulting in heatmaps $\{\mathbf{H}_1, \mathbf{H}_2, ..., \mathbf{H}_K\}$ indicating the confidence of the kth keypoint.

The idea is to have multiple subnetworks running in parallel, each with a different resolution. Starting from a high-resolution subnetwork, lower-resolution subnetworks are gradually added in parallel by downsampling from the previous, higher-resolution subnetworks as can be seen in figure 3.2.

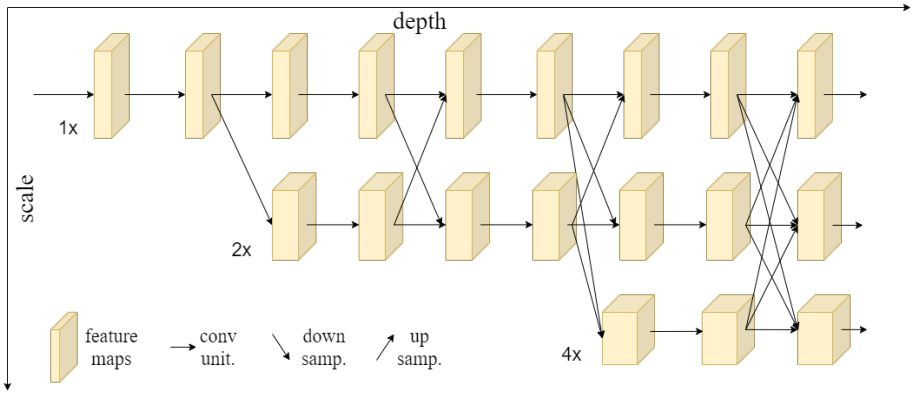


Figure 3.2: HRNet, architecture overview.

Information is shared across all subnetworks by utilizing an exchange unit

$$\mathbf{Y}_k = \sum_{i=1}^{s} a(\mathbf{X}_i, k) \tag{3.4}$$

where the input to the exchange unit are the s response maps $\{\mathbf{X}_1, \mathbf{X}_2, ..., \mathbf{X}_s\}$ from the parallel subnetworks. The output is s response maps $\{\mathbf{Y}_1, \mathbf{Y}_2, ..., \mathbf{Y}_s\}$. The function $a(\mathbf{X}_i, k)$ aggregates the input maps from resolution in stage i to the resolution in stage k by performing downsampling or upsampling. High-to-low resolution via strided 3x3 convolutions and low-to-high resolution via 1x1 nearest neighbor upsampling as can be seen in fig 3.3.

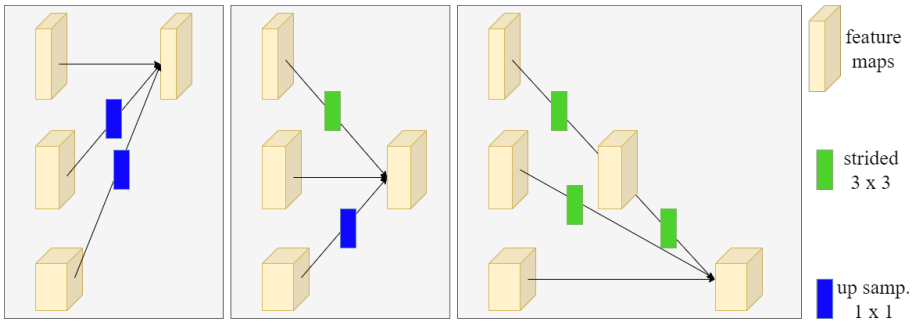


Figure 3.3: HRNet, exchange unit.

The heatmaps are then regressed from the highest-resolution representation of the final exchange unit. The HRNet body consists of four parallel subnetworks over 4 stages, containing 1, 4 and 3 exchange units for the 2nd, 3rd and 4th stage respectively. Two different versions of HRNet are used, HRNet-W32 and HRNet-W48, with a different amount of channels. HRNet is a simple yet effective architecture and it serves as a baseline architecture for many top-performing pose-estimation networks.

3.3.2 Distribution-Aware coordinate Representation of Keypoint - DARK

When running an image through a pose estimation network, it is most often resized to fit a specific dimension. This can give rise to an issue where confidence heatmaps of the keypoints in the dimensions of the resized image are not as reliable when they are upsampled to the original image dimensions. As a solution to this issue, DARK [42] utilizes encoding and decoding methods to better predict keypoints in shifting dimensions. Firstly, the ground truth keypoint is encoded to a gaussian distribution centered on the ground truth keypoint. The network is then trained with the target being that heatmap instead of the keypoint location.

When reducing the image dimensions, the ground truth-coordinate, denoted as $\mathbf{g} = (u, v)$ is represented in the reduced resolution as

$$\mathbf{g}' = (u', v') = \frac{\mathbf{g}}{\lambda} = (\frac{y}{\lambda}, \frac{v}{\lambda}) \tag{3.5}$$

where λ is the downsampling ratio. Usually, a quantisation function is used to assign the ground truth to a location in the downsampled image space which can lead to quantisation errors. In DARK no quantisation is used to avoid this, a heatmap can for example have it's center in between pixels. The heatmap G can then be synthesized as

$$G(x, y; \mathbf{g}') = \frac{1}{2\pi\sigma^2} exp(-\frac{(x-u')^2 + (y-v')^2}{2\sigma^2}) \tag{3.6}$$

where σ is a fixed spatial variance and (x, y) is a pixel location in the heatmap.

Decoding from a heatmap to a keypoint might seem like a trivial task and an often used coordinate decoding method is hand-designed with little justification, the prediction $\mathbf{p}$ is calculated as

$$\mathbf{p} = \mathbf{m} + 0.25 \frac{\mathbf{s} - \mathbf{m}}{\|\mathbf{s} - \mathbf{m}\|_2} \tag{3.7}$$

where p is the prediction, m is maximal activation, s is second maximal activation. Which is essentially the maximal prediction shifted 0.25 pixels towards the second maximal prediction. The final coordinate prediction $\hat{\mathbf{p}}$ is computed

$$\hat{\mathbf{p}} = \lambda \mathbf{p} \tag{3.8}$$

where λ is the resolution reduction ratio. This method produces relatively accurate results but the nature of how it is constructed makes it seem likely a better method exists. DARK uses another method which uses a Taylor series expansion to approximate the keypoint, this makes the assumption that the heatmap follows a Gaussian structure.

The actual predicted heatmaps have been empirically shown not to sufficiently resemble Gaussians. By modulating the predicted heatmaps, they can be made to more accurately resemble a Gaussian distribution. By using a Gaussian kernel K with variation like the encoded keypoints in the training data one can smooth out the heatmap h to resemble a Gaussian,

$$h' = K \circledast h \tag{3.9}$$

where h' is the Gaussian smoothed heatmap and $\circledast$ denotes the convolution operation. In order to keep the same magnitude as the original prediction,

$$h' = \frac{h' - min(h')}{max(h') - min(h')} * max(h) \tag{3.10}$$

where h' is scaled to have the same maximum activation as h.

Chapter 4

Action recognition

Action recognition is a common problem in computer vision in which the goal is to analyze an image or video and classify the action taken by the subject depicted. In the early days of human action recognition, the focus was on grayscale or RGB-videos to infer the action [30]. In recent years, many different modalities have been used for action recognition, skeleton, depth maps, IR-data and many more. In this thesis we have decided to focus on human skeleton data, to use this modality is more often than not a 2-part problem where the first step is to extract the information about the subject. The second part is analyzing the data and attempting to classify it as a specific action. In order to better understand the concept, what follows are descriptions of various datasets used in action recognition as well as explanations of a few relevant action recognition networks.

4.1 Datasets for human action recognition

Datasets for action recognition consist of images or videos containing a varying amount of people with labels denoting their actions. Below follows an explanation of some datasets used for action recognition.

4.1.1 NTU RGB+D

NTU RGB+D was created in 2016 and greatly surpassed the size of previous publicly available datasets in the field of action recognition [32]. The dataset consists of 56880 video samples with 60 classes and 40 different subjects aged between 10-35 years old. There are 1920 x 1080 RGB-videos for all the actions from the angles -45 (camera 2), 0 (camera 1) and 45 (camera 3) degrees. Each action was performed twice, once towards the camera 2 and once towards camera 3. This ensures that each action is visible from both the left and right side.

By using the Kinect v2, they were able to capture IR-data, 3D-skeletons and

depth maps, in addition to the RGB-video. Depth maps are sequences of two-dimensional depth values in millimeter with the resolution 512x424. The 3D-skeleton data consists of 3D-coordinates for 25 body joints. The corresponding pixel on the depth map and RGB-video is provided for each joint and frame. Complementary to this, infrared sequences are collected frame by frame with size 512x424. To evaluate a network on this dataset, two methods are common: cross view and cross subject validation.

Cross view evaluation is done by using camera 1 as test data and camera 2 and 3 for training. This means that the two front views and two 90 degree views are used for training. For testing, both 45 degree views are used. The training and testing set have the sizes 37920 and 18960 respectively.

In cross subject evaluation, the 40 subjects are divided into training and test sets. The training and validation sets consist of 40320 and 16560 samples, respectively. In addition to NTU RGB+D, there exists an extension called NTU RGB+D 120 which increases the amount of classes to 120 and 106 distinct subjects in over 11400 videos or 8 million frames [24]. Example images from videos from these sets are seen in figure 4.1.

Figure 4.1: Example images from the NTU RGB+D and NTU RGB+D 120 datasets.

4.1.2 Varying-view RGB-D Action-Skeleton

Varying-view RGB-D Action-Skeleton [16] is a dataset containing video samples captured in 8 fixed viewpoints spread out in a range of 360° around the target. 118 subjects act out 40 different actions resulting in 25,600 video samples. Data provided consists of: RGB videos, depth images and skeleton sequences. Skeletons contain 25 body joints per frame with coordinates in 3D.

4.1.3 Kinetics-Skeleton

Kinetics-Skeleton [36] is a multi-edition action recognition dataset. It only provides raw video data and action labels, there is no skeleton data provided. As of writing this report, three editions have been release, Kinetics-400, Kinetics-600 and Kinetics-700 which contain 400, 600 and 700 action classes, respectively. All video clips are taken from various YouTube videos, this can pose a problem as videos can be taken down and made unavailable. To address this problem, a new edition of the Kinetics-700 dataset has been released with the missing videos replaced, Kinetics-700-2020. The clips are approximately 10 seconds in length with a variable frame rate and resolution. Because of the larger amount of action classes, the dataset includes several very specific ones such as vacuuming car and tasting wine.

Kinetics-Skeleton			
Dataset	# classes	Average	Minimum
Kinetics-400	400	683	303
Kinetics-600	600	762	519
Kinetics-700	700	906	532
Kinetics-700-2020	700	926	705

Table 4.1: Statistics of each edition of Kinetics-Skeleton dataset, showing action classes, average and minimum videos per class [36].

4.1.4 J-HMDB

J-HMDB consists of 2D skeleton annotations manually annotated on a sub part of the HMDB51 database [18]. The dataset consists of 21 categories with single person actions. The clips are cut such that the video starts and ends roughly when the action starts and ends. There are 36-55 clips per action with each clip containing 15-40 frames. There are 31838 annotated frames in total [15].

4.2 Networks for human action recognition

Depending on the type of available data, there are many ways in which to perform action recognition, we have decided to solely focus on the skeleton of the subject, disregarding all other data, including RGB-video. Since the only data modality utilized is the skeleton, certain aspects of a performed action will be lost, e.g., facial expressions, objects being held, and the surrounding environment. Consequently, not all actions are of equal difficulty to infer for a skeleton based action recognition network, for instance, full-body actions are easier to infer than actions mainly performed with facial expressions. There are three different methods mainly used to perform skeleton based action recognition: Convolutional Neural Networks (CNN), Recurrent Neural Networks (RNN) and Graph Convolutional Neural Networks (GCN).

CNNs generally do this by encoding the positional information about a skeleton across a certain time interval into a pseudo-image, which then uses common CNN network to associate structures in the pseudo-image to behavioural patterns.
RNNs are utilized by taking the skeleton data represented as a sequence of coordinate vectors of the spatial and temporal dimensions with LSTM [12] networks as they are well-suited for time series data.
GCNs attempt to form a better understanding of the skeletal structure as it can be difficult to get a complete picture of the structure in the form of an image or sequence. The GCN uses the complete skeleton represented as a graph. To infer the action performed a convolution can be performed that is similar to a CNN but retrofitted to fit the structure of a graph.

Performance comparison of action recognition networks Below follows a few top-performing action recognition networks on NTU RGB+D. It should be noted that these performances are on the ground truth 3D spatial coordinates, while our training and testing is done on 2D spatial coordinates combined with prediction confidence. Meaning that these results do not represent how well these different networks would perform on our data.

NTU RGB+D			
Model	Cross-view	Cross-subject	Extra training data
Seq2Im (CNN) [37]	83.3	88.8	x
Ind-RNN (RNN)[21]	88.0	81.8	x
Dense IndRNN (RNN)[20]	94	86.7	x
ST-GCN (GCN) [40]	88.3	81.5	x
2S-AGCN (GCN)[35]	95.1	88.5	x
MS-AAGCN (GCN)[34]	96.2	90	x
RNX3D101+MS-AAGCN-C (GCN)[34]	99	96.1	✓
Action Machine (CNN) [43]	97.2	94.3	x

Table 4.2: Statistics of performance on NTU RGB+D (3D), showing cross-view, cross-subject and if there was extra training data other than skeleton-data.

We have chosen two networks to perform our tests with to get a sense of the difference the type of network can have. One GCN and one CNN will be used, they will be further explained below.

4.3 Using a convolutional neural network

Another model is to use a pose-estimation network to retrieve the 2D skeleton data given an RGB-video. In [19], OpenPose is used to retrieve the 2D skeleton data. The data from an entire video clip is then encoded into a pseudo RGB-image where each pixel represents a joint in a given time frame. The RGB-channels holds the coordinate and confidence of a joint. In the R channel, the X-coordinate is stored, in the G channel, the Y-coordinate, and lastly in the B channel, the confidence output of the pose estimation network is stored. This method then uses current state of the art image classifying networks to connect each pseudo image to an action. Since the pseudo images differ from traditional images quite a lot, a deep retraining for the image classifier is made. The images need to be resized to fit the CNN-architeture used. In this approach, they reach a CV (Cross-View) accuracy of 88.8% and a CS (Cross-Subject) accuracy of 83.3% on NTU RGB+D. This result is based only on the single person actions of the dataset.

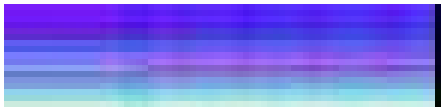
Convolutional neural networks are generally used for images and videos, such as the extraction of the keypoints and skeleton in this thesis. However if the action data can be presented as an image, it is possible to see how a convolutional neural network trained on image classification can be transfer learned to instead learn to recognise encoded actions in psuedo RGB images.

4.3.1 Converting skeleton data to an image

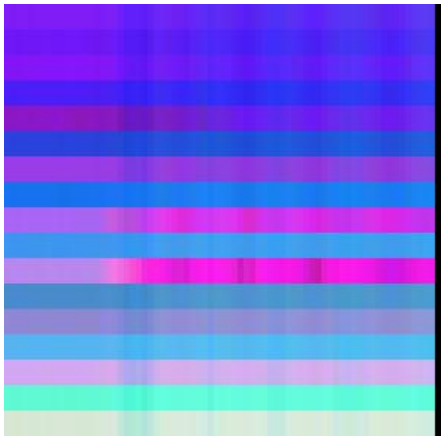
Using HRNet, 2D skeleton data and confidence is extracted. The data consists of X and Y coordinates and the confidence of 17 joints in each frame. The goal is to create an RGB image which can be used as input to an image classifier network. In [37], a method they call "seq2im" is used. The idea is to transform the skeleton data from a video into a psuedo RGB image which contains the same information in a different format.

Each pixel in this new image will have the X- and Y coordinates and confidence in the corresponding R, G and B channels. The i'th joint in frame j is stored in the pixel in the image corresponding to row i and column j. With a skeleton sequence from a video with N frames. The shape of this image would become (17,N,3). The number of frames vary between 80 and 300. This means that the images created will be stretched along the x-axis, see figure 4.2a. Since the input to most image classification networks often require a square image, an upscaling is required.

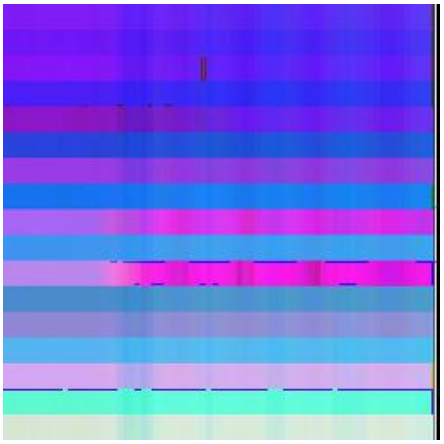
First the columns are repeated m_1 times where $m_1 = ceil(256/N)$ to extend the image to atleast 256 columns. Then the rows are repeated in the same manner to make an image as close to a square as possible, see figure 4.2b. This image is then scaled to 256×256 with bicubic interpolation, see figure 4.2c.



(a) Psuedo image before any size change. Has the size of (F,N,3) where F is the number of frames in the video and N is the number of joints. In this case, the size is (86,17,3).



(b) Psuedo image after upscaling to a square image. With the original having size (86,17,3), this image has the size (255,258,3).



(c) Final psuedo image with size (256,256,3) after scaling.

Figure 4.2: Psuedo images describing the process of extracting joints data from a video and turning it into s square image. This image describes action number 23 from the NTU RGB+D dataset, "Hand waving".

The images can be understood by thinking of the x-axis as time. The quite obvious rows are due to the upscaling and each represent a joint from the skeleton. An increase in red color would mean a movement to the right, an increase in green color would mean a movement upwards. The blue color describes the confidence of the network that the joint is correct. Another idea in [37] was to use the mean of the X- and Y values to fill the third dimension but confidence performed better.

4.3.2 Choosing a CNN architecture

In [37] Resnets gave the highest performance. Resnet151 was the best performing with Resnet34 less than one percentage point behind. Due to the small difference in performance but huge trade off in size, Resnet34 was chosen as the network to continue with due to time constraints.

4.3.3 Resnet34

Resnet is a CNN architecture which was first to utilize residual connections between the layers in the network. The residual connections made it possible for a network to become deeper without decreasing performance. Previously, deeper networks had seen an increase in performance but eventually reached a point where accuracy was saturated and using more layers led to a higher training error.

A residual connection is used such that instead of finding the mapping $F(x)$, the original input is added to the output. Hence the sought after mapping is $H(x) = F(x) + x$ between layers, see figure 4.4. The idea of a residual connection was that it would be easier for a network to learn the residual mapping instead of the original mapping. For example, if a new layer was added which would be superfluous, it would be easier to push the residual to zero than to find an identity mapping.

The entire network architecture of Resnet34 can be seen in figure 4.3. The 34 stands for the number of layers in the network. In the figure the last fully connected layer is 1000 since most networks are pretrained on ImageNet [8]. In our application with 49 actions, the last layer is simply made into one with 49 outputs. The network comes already implemented in PyTorch, a deep learning module for Python.

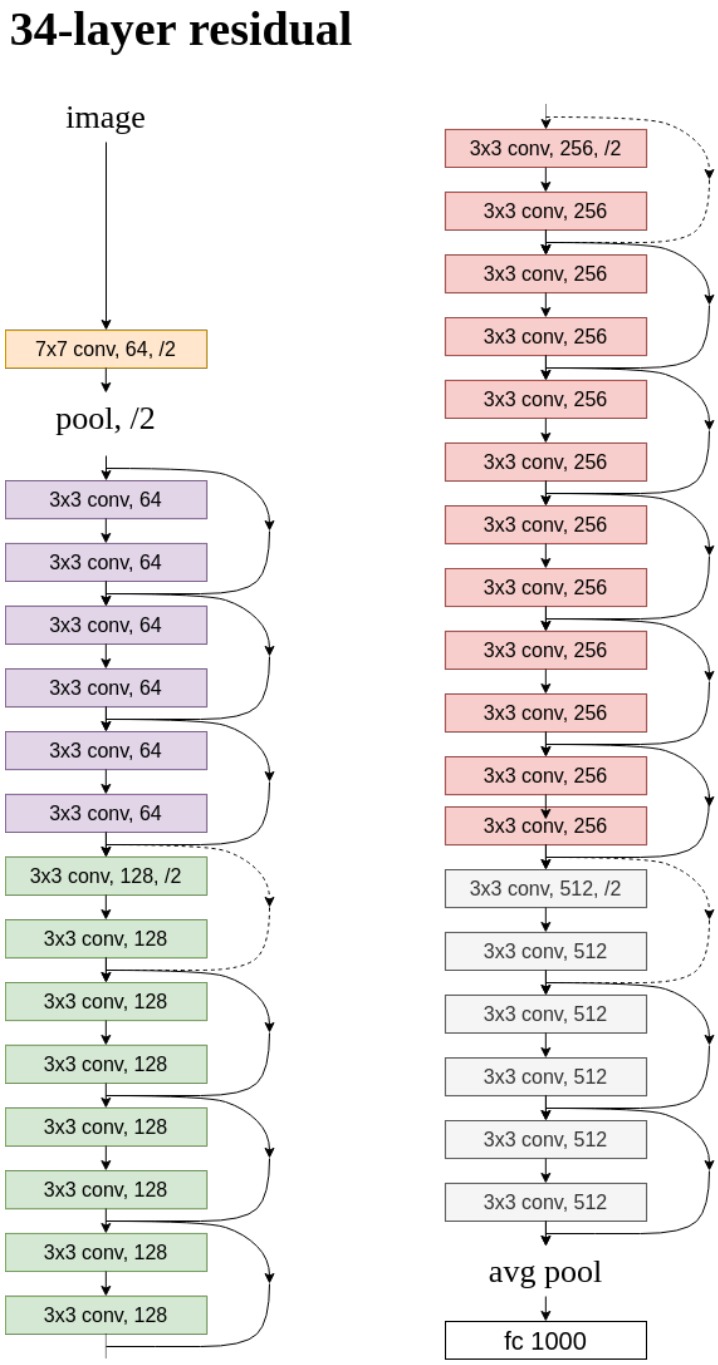


Figure 4.3: Image describing the Resnet34 architecture. The arrows between the blocks are the residual connections [11].

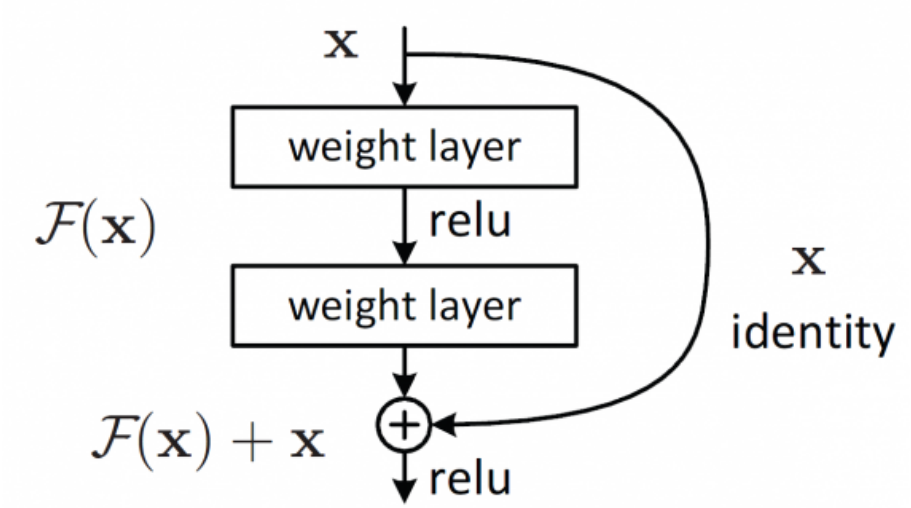


Figure 4.4: Image describing how the residual works between layers. Instead of finding the mapping $F(x)$, the original input to the layer is added to the output, thus instead looking for the mapping $H(x) = F(x) + x$ [11].

4.4 Multi-Stream Adaptive Graph Convolutional Networks (MS-AAGCN)

MS-AAGCN [35][34] is an attempt to utilize a graph convolutional network to model graphs in the shape of human skeletons. The structure of the input is a graph of N joint nodes over T frames, containing in total $N \times T$ vertices. Two types of edges connect these vertices as can be seen in figure 4.5. Firstly, intra-frame edges which represent the spatial edges between vertices in the same frame according to the natural topology of the human body. Secondly, inter-frame edges which connect a joint vertex at frame t with the same type of joint at frame $t-1$ and $t+1$.

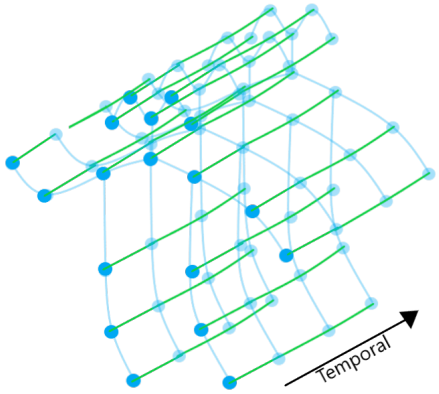


Figure 4.5: Image showing connection between vertices in spatial and temporal space. Light blue edges show spatial connections. Green edges show temporal connections.

To perform convolution on this graph, MS-AAGCN uses a special sampling function $B(v_{ti})$ defined as

$$B(v_{ti}) = \{v_{tj} | d(v_{tj}, v_{ti}) \leq D\} \tag{4.1}$$

on the node v_{ti} at frame t and joint i where $d(v_{tj}, v_{ti})$ denotes the shortest distance between vertices v_{tj} and v_{ti} in the graph. In Yan et al. [40] they use $D = 1$ which is the 1-neighbourhood of the root node. However, as the number of nodes connected by vertices to the root node can differ in a graph. MS-AAGCN utilizes a mapping function l_{ti} defined as

$$l_{ti} : B(v_{ti}) \rightarrow \{0, ..., K-1\} \tag{4.2}$$

which partitions the neighbouring nodes into a fixed number of subsets K to account for a fixed size weight vector. More specifically, the partitioning strategy

divides the set of nodes contained in the sampling function $B(v_{ti})$ into 3 subsets depending on the relative proximity to the mean of the skeleton compared to that of the root node as can be seen in figure 4.6. The first subset contains the nodes that are closer to the mean than the root node. The second subset contains the root node. The third subset contains the nodes that are further away from the mean than the root node.

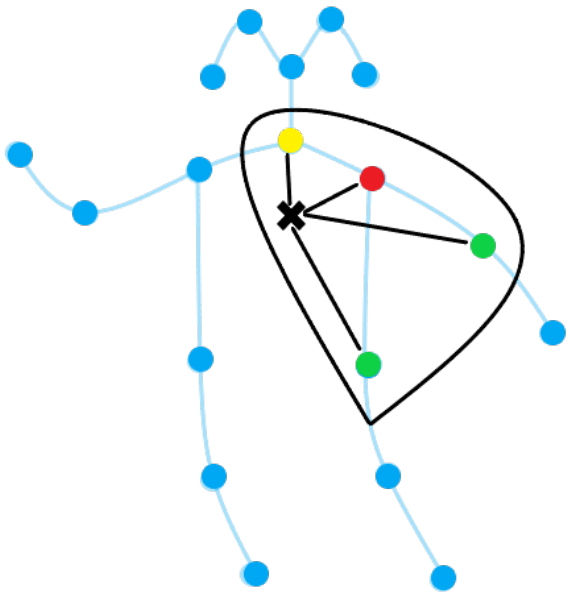


Figure 4.6: Circled nodes showing sampling area for convolution of a graph on the root (red) node. Yellow nodes are closer to the mean of the skeleton (cross) than the root node. Green nodes are further away from the mean than the root node.

4.4.1 Architecture

The architecture consists of 9 basic blocks shown in figure 4.7. Below follows a description of different kinds of layers.

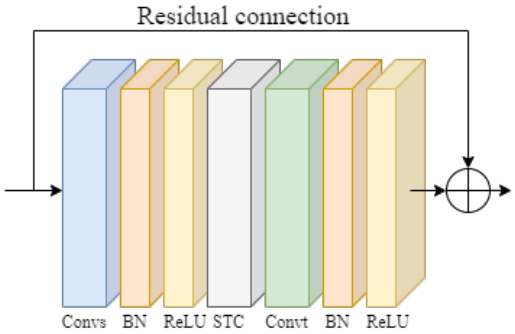


Figure 4.7: The basic block, Convs and Convt represents spatial and temporal convolution, followed by a batch normalization (BN) layer and ReLU layer. In the middle there is an STC-module and finally, there is a residual connection for each block as explained in 4.3.3.

4.4.2 Spatial Convolution (Convs)

Graph convolution usually incorporates the adjacency matrix $\mathbf{A}$ of the graph which shows the connections of the graph as explained in section 2.5. The implementation of the convolution in MS-AAGCN is a bit different and can be described as

$$\mathbf{f}_{out} = \sum_{k}^{K_v} \mathbf{W}_k \mathbf{f}_{in} \mathbf{A}_k \tag{4.3}$$

where K_v is the kernel size, which is 3 as explained above. The feature maps $\mathbf{f}_{out}$ and $\mathbf{f}_{in}$ denote the outputs and inputs respectively with $\mathbf{W}_k$ denoting the weight function for the corresponding subset k. Finally, $\mathbf{A}_k$ is similar to the adjacency matrix $\mathbf{A}$ but instead of $\mathbf{A}^{ij}$ indicating an edge between vertex v_i and v_j, $\mathbf{A}_k^{ij}$ indicates whether vertex v_j is in subset S_{ik} of vertex v_i according to the sampling function.

Expanding on equation 4.3 to allow for learnable edges, $\mathbf{A}_k$ is divided into sub-graphs $\mathbf{B}_k$ and $\mathbf{C}_k$,

$$\mathbf{f}_{out} = \sum_{k}^{K_v} \mathbf{W}_k \mathbf{f}_{in} (\mathbf{B}_k + \alpha \mathbf{C}_k) \tag{4.4}$$

where $\mathbf{B}_k$ is the global graph initialized with the values of $\mathbf{A}_k$ but with parameterized elements which are updated during the training process, meaning that new graphs can be learned better suited for the task. The second sub-graph $\mathbf{C}_k$ is an individual graph for each sample that is a data-driven similarity matrix whose element $\mathbf{C}_k^{ij}$ determines the similarity between vertex v_i and v_j. This can be seen as a soft edge between the two vertexes.

The global graph $\mathbf{B}_k$ determines the basic topology of the graph and the individual graphs $\mathbf{C}_k$ adds individuality to each sample. Experiments have determined a differing degree of importance between these two sub-graphs depending on the layer, where the individual graph is more important in the top layers than the bottom layers. Thus, a parameterized coefficient α that is unique to each layer and adjusts the importance of $\mathbf{C}_k$ is learned in the training process as well.

4.4.3 Temporal Convolution (Convt)

The convolution on the temporal dimension is more straightforward than the spatial convolution. Since the convolution is only performed in the temporal dimension, every vertex has a fixed number of 2 neighbours. The convolution is performed in the same manner as in equation 4.3 only this time, the convolution is performed on the temporal dimension instead of the spatial one.

4.4.4 STC-attention module

The purpose of the STC-attention (Spatial Temporal Channel) module is to help the network focus on specific joints, frames and channels for various actions. The spatial attention module $\mathbf{M}_s$ is an response map which highlights important joint. The map is defined as

$$\mathbf{M}_s = \sigma(g_s(AvgPool(\mathbf{f}_{in})) \tag{4.5}$$

where $AvgPool(f_{in})$ averages the input feature map $\mathbf{f}_{in} \in \mathbb{R}^{C_{in} \times T \times N}$ over all frames where T is the number of frames, N is the number of joints and C_{in} is the number of channels. A 1-D convolution g_s is performed with trainable weights $\mathbf{W}_{g_s} \in \mathbb{R}^{1 \times C_{in} \times K_s}$ where the convolution kernel $K_s = 3$. The results are passed through the *Sigmoid* activation function, denoted by σ. Consequently, the final response map is of dimension $\mathbf{M}_s \in \mathbb{R}^{1 \times 1 \times N}$.

The temporal attention module $\mathbf{M}_t$ is similar to the spatial one and is defined as

$$\mathbf{M}_t = \sigma(g_t(AvgPool(\mathbf{f}_{in})) \tag{4.6}$$

where instead of averaging over the frames, it instead averages over the joints and g_t uses weights along the temporal dimension instead. Resulting in a final response map $\mathbf{M}_t \in \mathbb{R}^{1 \times T \times 1}$.

The channel attention module $\mathbf{M}_c$ can help with amplifying certain features (channels) and is defined as

$$\mathbf{M}_c = \sigma(\mathbf{W}_2(\delta(\mathbf{W}_1(AvgPool(\mathbf{f}_{in}))))) \tag{4.7}$$

which averages the input $\mathbf{f}_{in}$ over all the joints and frames. δ is the ReLu activation function, $\mathbf{W}_1$ and $\mathbf{W}_2$ are weights of two fully-connected layers, resulting in the response map $\mathbf{M}_c \in \mathbb{R}^{C \times 1 \times 1}$.

The STC-module is introduced to the basic block as shown in figure 4.8.

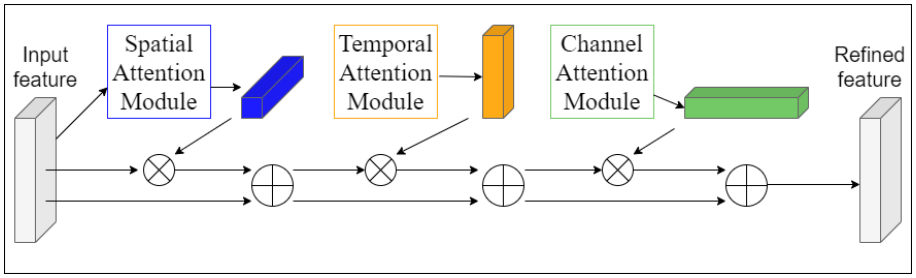


Figure 4.8: The structure of the STC-module in the basic block. Where $\otimes$ denotes element-wise multiplication and $\oplus$ denotes element-wise addition.

4.4.5 Modalities

In addition to joints, second-order information in the shape of bones is investigated, as well as both of their motions resulting in a total of four different modalities investigated. A bone is defined by two joints and the joint closest to the skeletons center of gravity is the source joint. The bone is represented by a vector pointing to the target joint from the source joint. The motion of the joints and bones is calculated as the difference between them in consecutive frames. Combining joints, bones, joint motion and bone motion it forms a multi-stream network that adds the softmax score of each to predict the action, illustrated in figure 4.9.

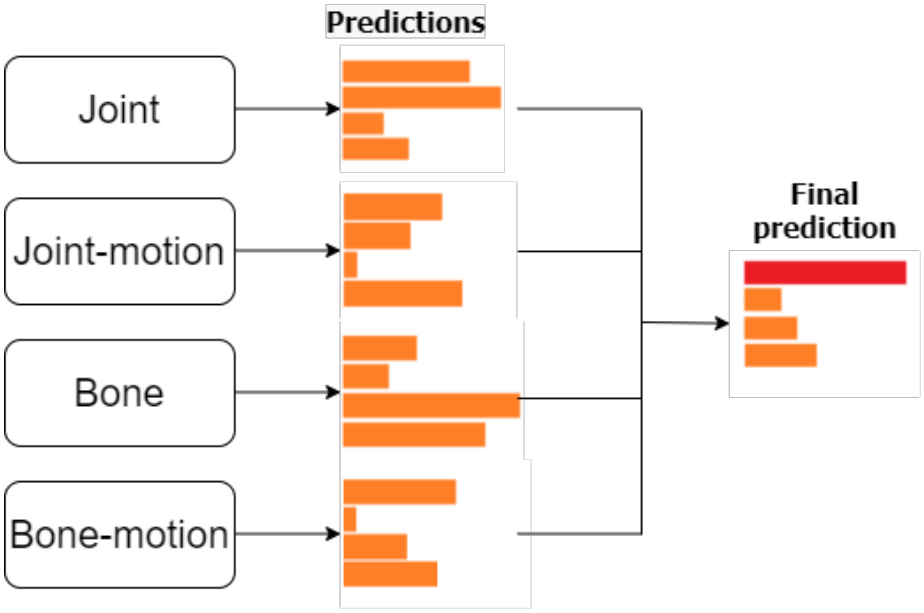


Figure 4.9: The predictions of four different modalities are aggregated to form a final prediction.

Chapter 5

Methodology

In the following subsections, a more detailed description of how the process of going from a video to an action classification is described. Two different methods will be used to classify the videos, one using a GCN and the other using a CNN. In addition, the manner in which we recorded our own dataset will be explained.

5.1 Choices and motivations

The goal is to be able to see how action recognition works on new angles with the input being a standard RGB-video. For this, only information that can be extracted from an RGB-video will have to be used. While there are many nuances to recognizing an action, most full-body movements can be recognized by examining the movement of the skeleton. Other modalities add information that might change in different settings, because of lighting, clothing, etc. To make it more robust to these changes we have decided to focus solely on the movement of the joints. Therefore, a video of a performed action can for our purposes be reduced to the movement of skeleton joints.

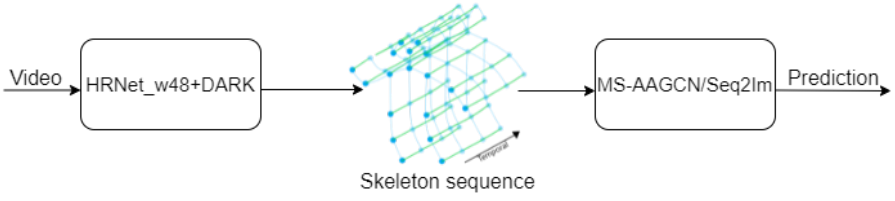


Figure 5.1: Pipeline from video to keypoints to action prediction.

To build our pipeline as can be seen in figure 5.1, we required a dataset with videos containing actions to serve as our baseline for what is possible. We decided to use the NTU RGB+D dataset [32], as it contained a large amount of

data from differing angles. Our feeling was that it was perfect in order to create a robust network for a baseline to add additional angles to. In order for the data to have the same conditions as our own dataset, we disregarded the ground truth keypoints and simply fed the videos into a pose estimation network to get different keypoints. As the ground truth keypoints were not manually annotated but rather inferred by a Kinect V2, the feeling was that the reliability of the new keypoint data will not be any less than that of the old one.

For this task, the top performing pose estimator HRNet_W48+DARK is used. At the time it was the best performing skeleton pose estimation network that we could find. We decided to use the keypoint setup from COCO as seen in figure 3.1, as the additional keypoints on the face added depth that could play a vital role in differentiating between actions involving head movement, especially in multiple angle situations.

For action recognition we wanted to try more than one type of network to ascertain if that played a part in the results. We decided to try one GCN-based and one CNN-based network.

5.2 Extracting Skeleton data

The videos from NTU RGB+D are run through HrNet_w48+DARK to extract 17 key points for each video and for each frame. The key points contain the X- and Y-coordinate and the confidence, $c \in (0, 1)$, of the networks guess of the position. The confidence will replace the Z-coordinate to represent a keypoint in 3D (x, y, confidence) for both action recognition networks. The framework and weights for performing the pose estimation were taken from the open-source toolbox for pose estimation MMPose [7].

5.3 Our own dataset

The dataset is created with 10 actions which are also contained in NTU RGB+D. The actions performed mimic the movements performed in NTU RGB+D. This means that an action such as "chest pain" has a standard movement pattern even though "chest pain" might be very different and subjective between subjects in a real world scenario.

5.3.1 Actions

The actions are the following:

- A7: Throw
- A8: Sit down
- A9: Stand up

- A23: Hand waving
- A24: Kicking something
- A28: Phone call
- A42: Staggering
- A43: Falling down
- A45: Chest pain
- A48: Nausea/vomiting

Since the original dataset consists of 49 single person actions, there is quite a large variety. Recording a dataset of the same size would be cumbersome and require too much time. Therefore the amounts of actions had to be reduced. When reducing the number of actions the variety must decrease. When selecting the above actions the motivation was to keep a variety, reduce the use of props and be movements one could see in an outside environment. For example, "brush hair" and "tear up paper" were left out. Since the goal is to examine performance change on varying angles, the actions themselves are not of utmost importance but instead the variety of actions.

5.3.2 Angles

In addition to the differing horizontal views of NTU RGB+D, our dataset contains angles varying in the vertical plane as well, which can be seen in figure 5.2, 5.3, 5.4 and 5.5. Each action will be repeated 8 times, where the subject rotates 45 degrees each time. Every subject repeats this for all ten actions. With four cameras, each subject will create $4 \cdot 10 \cdot 8 = 320$ videos. With 18 subjects, the dataset contains 5760 videos.

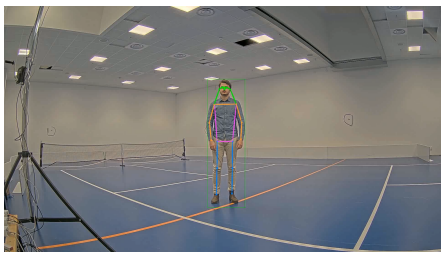
(a) 0°

(b) 30°

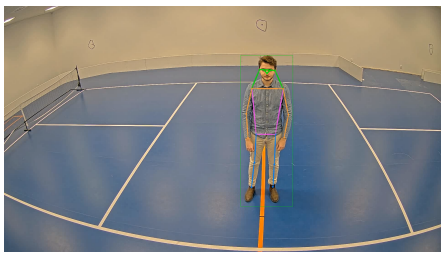
(c) 45°

(d) 60°

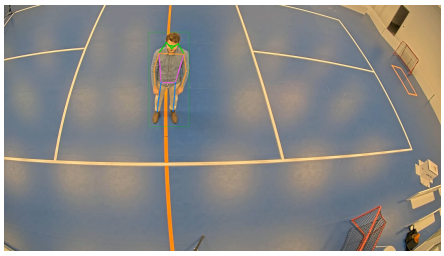
Figure 5.2: Example stills from our recorded dataset, raw video.



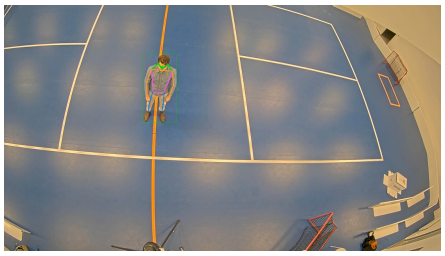
(a) 0°



(b) 30°



(c) 45°



(d) 60°

Figure 5.3: Example stills from our recorded dataset, with bounding box and skeleton.

(a) 0°

(b) 30°

(c) 45°

(d) 60°

Figure 5.4: Example stills from our recorded dataset, performing 'Throw', facing direction 90°, raw video.

(a) 0°

(b) 30°

(c) 45°

(d) 60°

Figure 5.5: Example stills from our recorded dataset, performing 'Throw', facing direction 90°, with bounding box and skeleton.

5.3.3 Camera Setup

The cameras used were PTZ 3255 cameras from Axis. The cameras are usually used in a security view setting, i.e the ceiling of convenience stores. In these applications one wants to increase the view of the camera, hence, a fish eye lens is used. During the recording of our dataset, the fish eye lens was active to better mirror a real world scenario.

For the setup, the vertical angles (0°, 30°, 45°, 60°) were strived for. Due to limitations in achievable height. The highest camera instead looked down on the subjects in an angle of about 58°, it will still be referred to as being 60° for the rest of the report for the sake of simplicity. The cameras were mounted on a tripod with an extension that could reach a height of 5 meters. The recordings took place in a gym at the Axis facilities with a sufficiently high ceiling. The distance from the tripod to the subject was the minimum distance required for a person to be visible in all cameras during the movements. The camera with vertical angle 0° had trouble fitting the subject in view. To increase the view of this camera, it was instead placed 1 meter further back in a 45° angle from the tripod where there was more room. The distance from the tripod to the subjects was 2.5 meters. Figure 5.6 shows the tripod with the highest camera attached.

Figure 5.6: Tripod with an extra broom on top to reach the desired height. In this image only the highest camera is mounted.

5.3.4 Recording

Each subject was instructed to perform each action in 8 horizontal directions. The subjects were instructed to try to keep the action as consistent as possible for each angles. For the actions 'sit' and 'stand' a chair was used and for the action "falling" a mattress was used to prevent injury.

5.3.5 Extracting

The videos were extracted through Axis Camera System (ACS). In this program the video streams from the different cameras were synchronized and recordings could be edited and extracted. Due to the automatic syncing, each cut in a recording would extract 4 videos, one from each camera. The cuts were made such that the videos starts and ends at the same time as the action. Due

to different length of actions and different subjects. The videos could range between 1.5-4 seconds. The videos have a frame rate of 25 frames per second and have a resolution of 1920x1080. The video compression standard used was h264 and the container format was MP4.

5.4 Hardware

All training, testing and pose extraction was made with a TITAN RTX 24GB. The large memory on the GPU was needed for the largest networks which used about 16GB memory.

5.5 Data post-processing

Due to the extreme angles and occlusion caused by facing 8 different direction around 360°, some keypoints were unreliable which post-processing attempted to solve. A tracker was used to interpolate missing bounding boxes for people as well as keypoints when there were dropped frames. There were certain video segments that were extremely difficult to affect in a meaningful way, such as when the subject sat down, facing away. As the back of the chair caused the occlusion of the legs and hips, getting even a decent approximation was difficult. This was expected and assessed as a non-issue as the angles facing away from the camera were gathered with the intention of gathering as much data as possible and was not intended to be used in a meaningful way. Additionally, in a subset of videos, an observer is visible in the recordings which caused the person and keypoint detector to gravitate towards that person instead of the subject. This was solved using a filter to ignore detections towards the edge of the screen since the intended subject was always center-screen.

5.6 Baseline network

The baseline network that serves as the starting point for every transfer learning was trained and validated on 60%/20% of the data from NTU RGB+D, the remaining 20% was set aside as a test set. As we decided to exclude multi-person actions from the dataset, the total amount of clips we had access to were 46,452.

Number of clips in each set:

- Train: 28,518
- Validation: 9,702
- Test: 8,232

The reason for the uneven amount of clips even though validation and test are both supposed to be 20% of the total set is that subjects from NTU RGB+D had a varying amount of clips, making an even split difficult.

5.7 Retraining

Creating a baseline network allowed us to start every transfer learning with identical conditions, making comparisons simple. Every transfer learning was performed with a k-fold cross validation for the training and validation data. The number of folds was determined on a case-by-case basis depending on how well the data could be divided. The transfer learning ran for 10 epochs, weights were saved on each epoch and we could then evaluate on whichever test set we required with the weights that had the lowest validation loss.

Due to long training times (up to 24 hours) we restricted the training to 10 epochs and chose not to perform k-fold cross validation with the inclusion of the test set.

Chapter 6

Tests

6.1 Tests to be run

With the new dataset there are many questions that can be asked and tests to perform. We limited ourselves to examining the actions performed in the front and side facing angles. Hence, the angles (135°, 180°, 225°) are removed for this analysis. The questions we want answered are the following.

- What is the performance change when increasing the vertical angle of the subjects?
- How does the performance change on an angle when adding data from that angle?
- When training on actions from the steeper angles, how is performance affected on actions not included in the training set but from the same steep angle?
- How does the performance change from all the new angles (30°, 45°, 60°), when only training on data from one of them?

6.2 Test specifics

In order to answer the above asked questions, 6 different tests were performed on 6 different subsets of training data.

Test 1, trained on data from 30°, 45° and 60°. This was our first test and was not supplemented with data from NTU RGB+D.

Test 2, trained on data from 0°.

Test 3, trained on data from 30°.

Test 4, trained on data from 45°.

Test 5, trained on data from 60°.

Test 6, trained on data from specific actions.

6.2.1 Training data size

Except for test 1, all tests will supplemented with a roughly equal amount of data from NTU RGB+D as to not overfit towards our subset of actions. Every type of test will be run with varying training set sizes as to ascertain the importance of training data size.

Test 1, training performed on 3, 6 and 9 subjects, resulting in 45, 90, 135 videos per action respectively.

Test 2-5, training performed on 3, 6 and 9 subjects, resulting in 15, 30, 45 videos per action respectively. Supplemented with roughly equal amount of training data from NTU RGB+D.

Test 6, training performed on 2, 4 and 6 actions, resulting in 90 videos per action. Supplemented with roughly equal amount of training data from NTU RGB+D.

6.3 Test sets

To evaluate performance there are several test sets.

Base contains subjects removed from the NTU RGB+D training process.

Base, only 10 is the same as base, but only the 10 actions also contained in our dataset.

Base, exclude 10 is the same as base, but excluding the 10 actions also contained in our dataset.

Own, 0° contains a subset from our own dataset removed from the training process, only from 0°.

Own, 30° contains a subset from our own dataset removed from the train-

ing process, only from 30°.

Own, 45° contains a subset from our own dataset removed from the training process, only from 45°.

Own, 60° contains a subset from our own dataset removed from the training process, only from 60°.

Base, only 3 is the same as base, but only the 3 actions not trained on from test 6.

Base, exclude 3 is the same as base, but excluding the 3 actions not trained on from test 6.

Own, only 3 contains a subset from our own dataset removed from the training process, only the 3 actions not trained on from test 6.

Chapter 7

Results

The results for tests 1-6 are illustrated below in the form of line graphs. In addition to the results on our own data, results are shown for the base dataset NTU RGB+D as well. For more detailed results see appendix C.

7.1 Test 1

The goal of this test was to ascertain how training on data from the novel angles (30°, 45°, 60°) affects performance, in both domains.

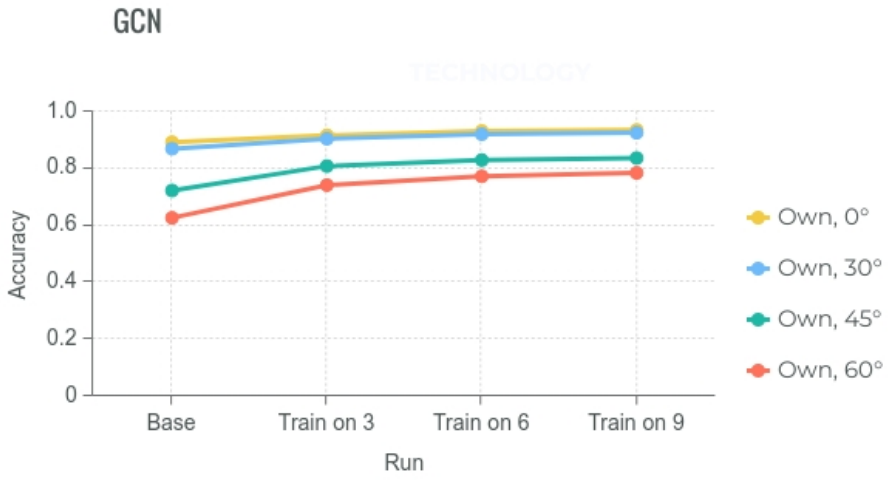


Figure 7.1: Results of GCN trained on data from 0°, 30°, 45° and 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

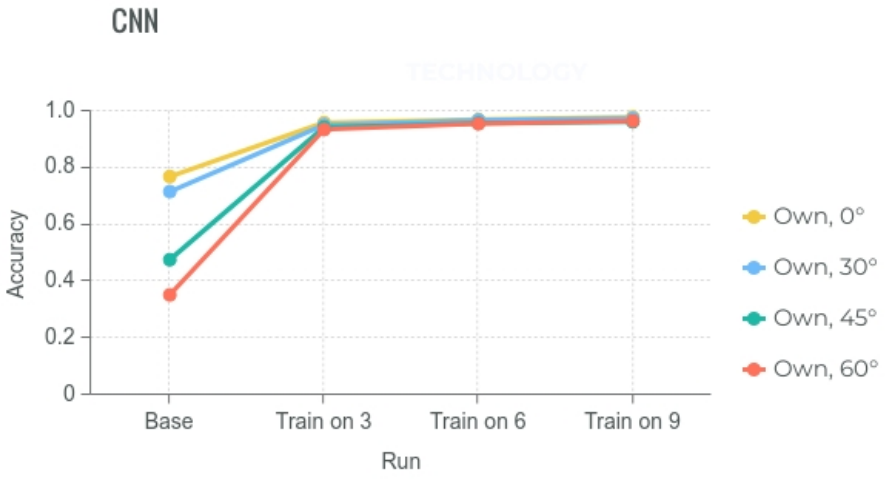


Figure 7.2: Results of CNN trained on data from 0°, 30°, 45° and 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

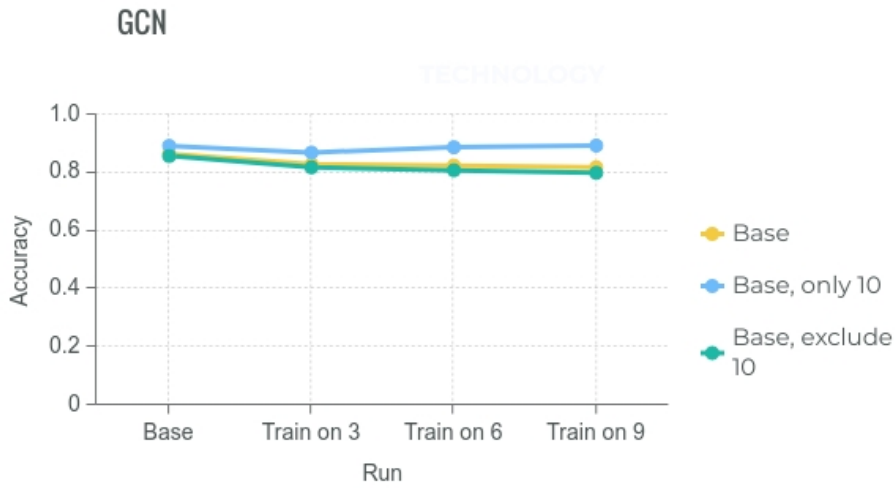


Figure 7.3: Results of GCN trained on data from 0°, 30°, 45° and 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

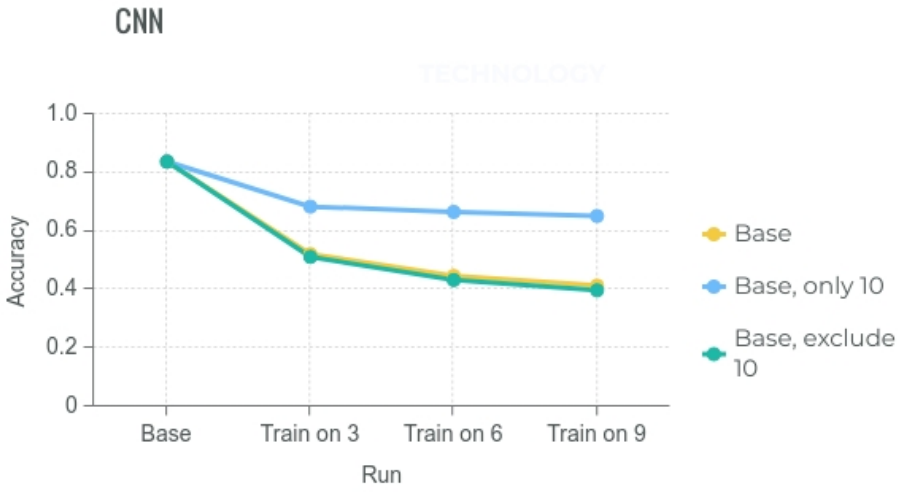


Figure 7.4: Results of CNN trained on data from 0°, 30°, 45° and 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

7.2 Test 2

The goal of this test was to see how training on angle 0° from our own dataset affects performance on the test sets to ascertain whether or not the manner in which we recorded our dataset has an effect. Seeing as both datasets are recorded from the same angle.

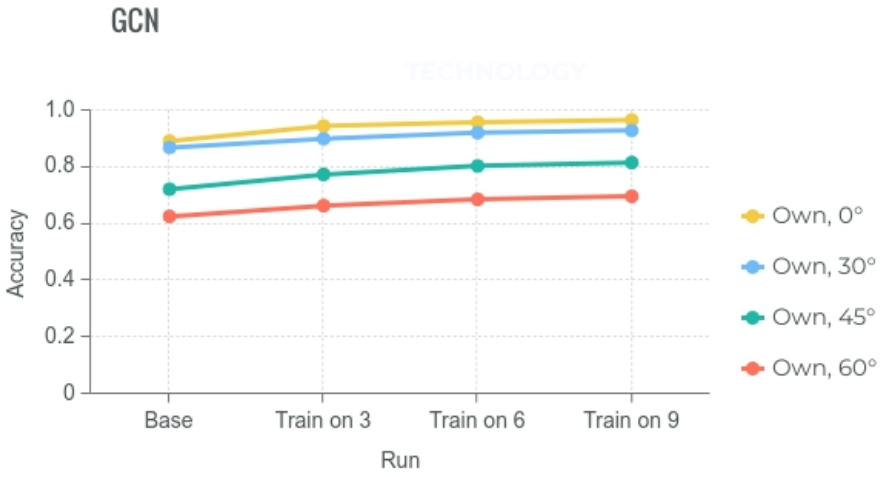


Figure 7.5: Results of GCN trained on data from 0°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

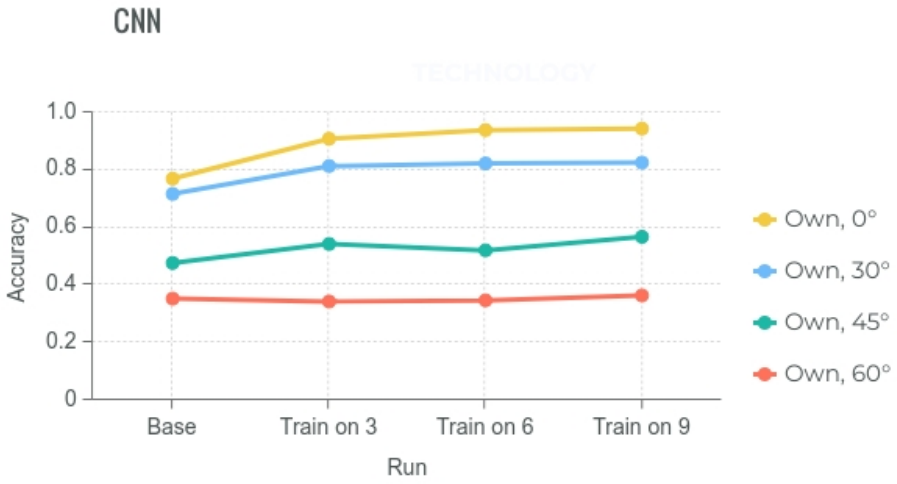


Figure 7.6: Results of CNN trained on data from 0°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

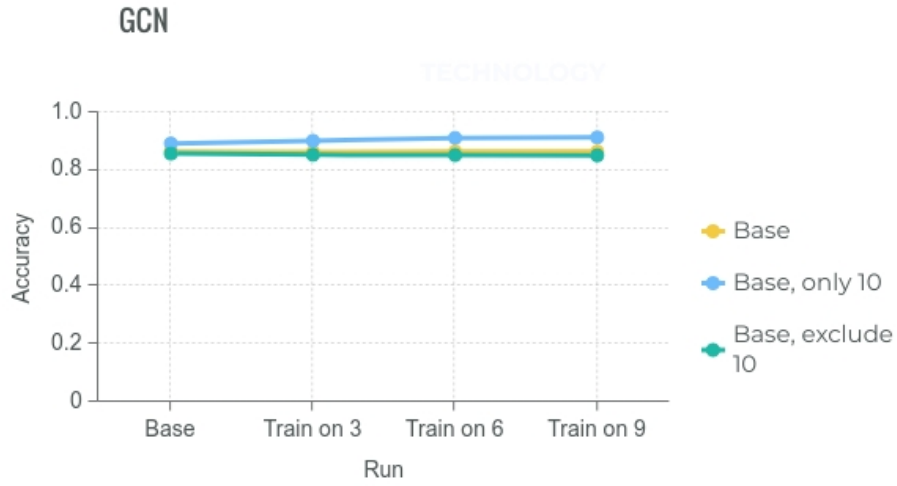


Figure 7.7: Results of GCN trained on data from 0°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

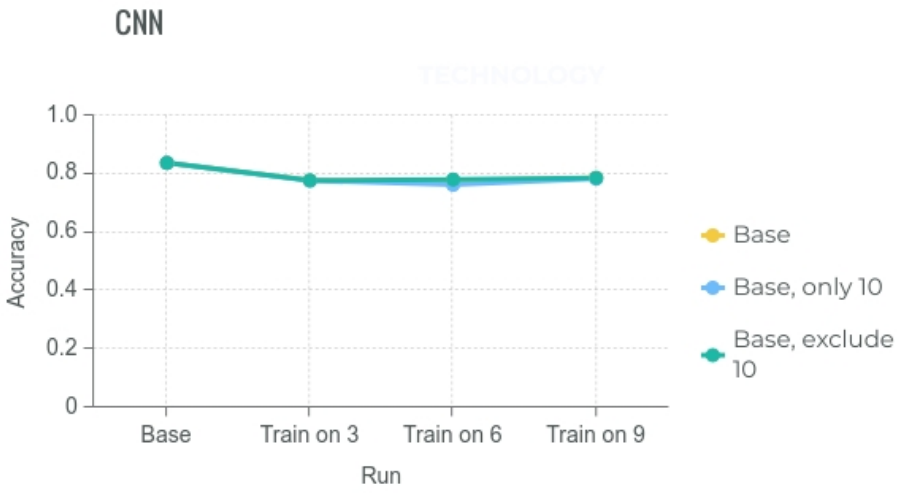


Figure 7.8: Results of CNN trained on data from 0°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

7.3 Test 3

The goal of this test was to see how training on one new angle (30°) affects the other new angles (45°, 60°)

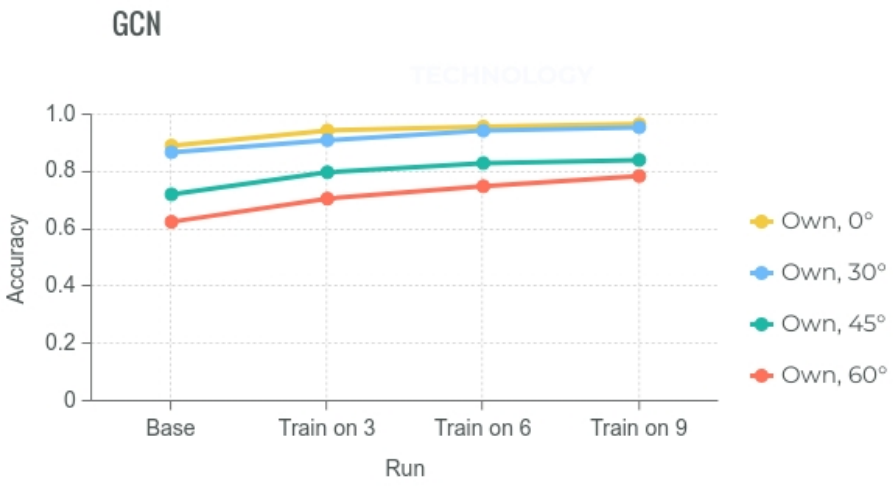


Figure 7.9: Results of GCN trained on data from 30°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

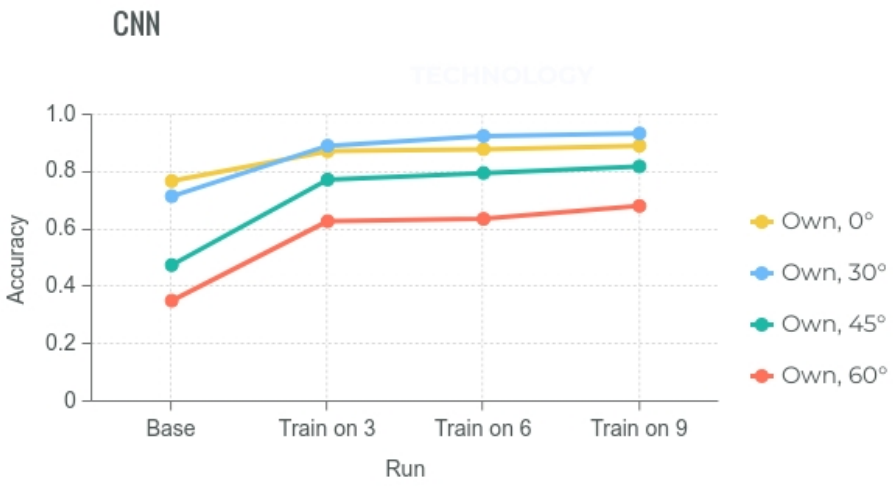


Figure 7.10: Results of CNN trained on data from 30°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

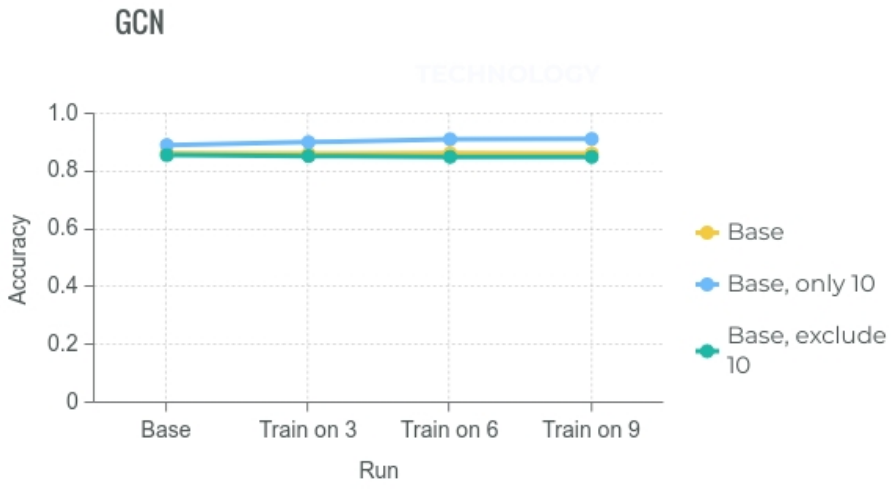


Figure 7.11: Results of GCN trained on data from 30°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

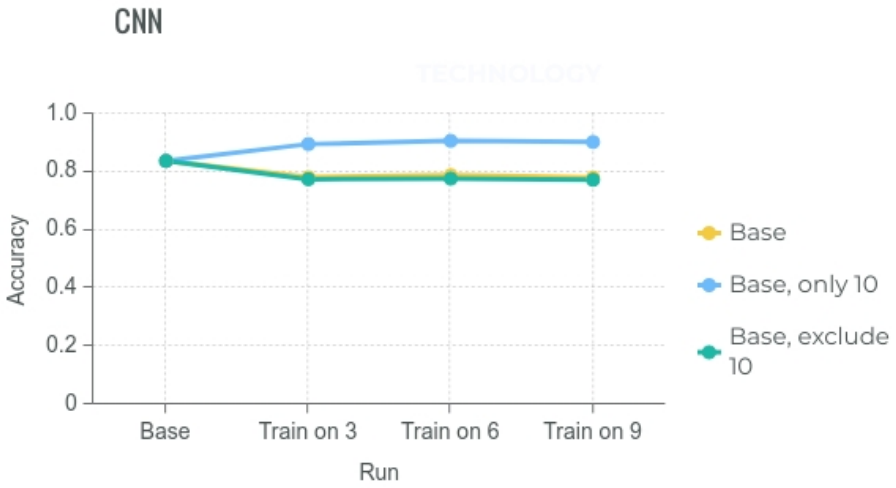


Figure 7.12: Results of CNN trained on data from 30°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

7.4 Test 4

The goal of this test was to see how training on one new angle (45°) affects the other new angles (30°, 60°)

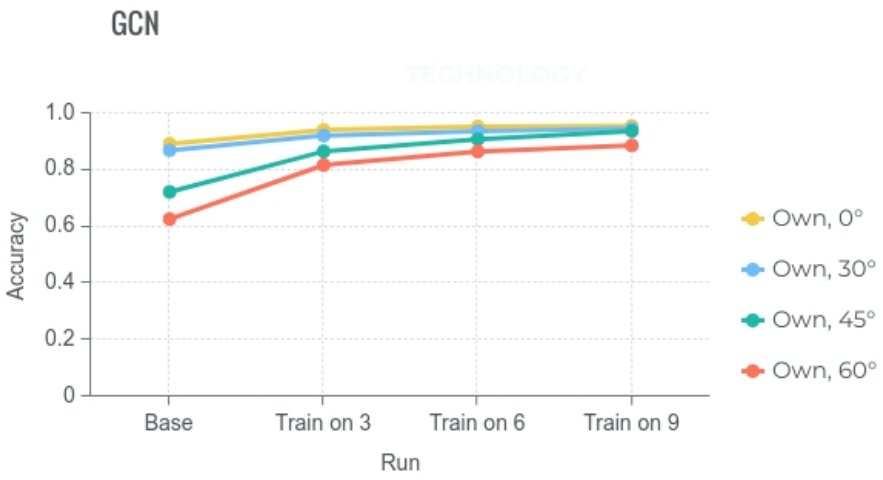


Figure 7.13: Results of GCN trained on data from 45°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

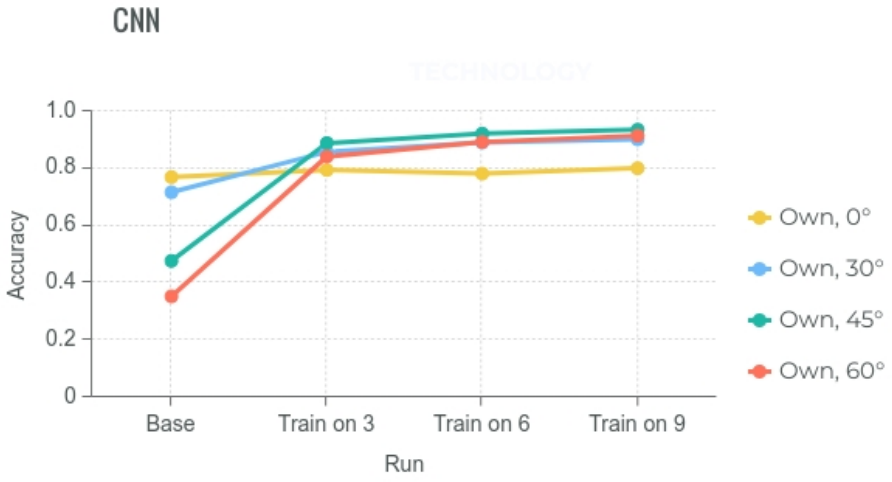


Figure 7.14: Results of CNN trained on data from 45°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

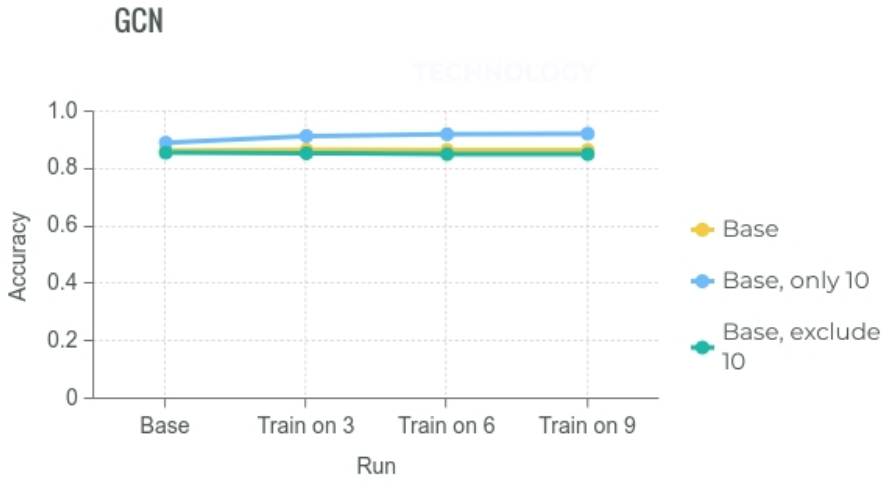


Figure 7.15: Results of GCN trained on data from 45°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

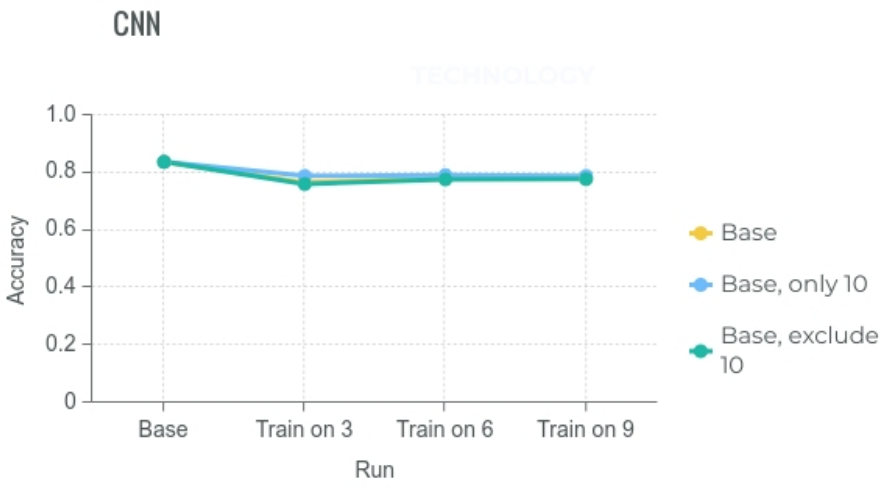


Figure 7.16: Results of CNN trained on data from 45°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

7.5 Test 5

The goal of this test was to see how training on one new angle (60°) affects the other new angles (30°, 45°).

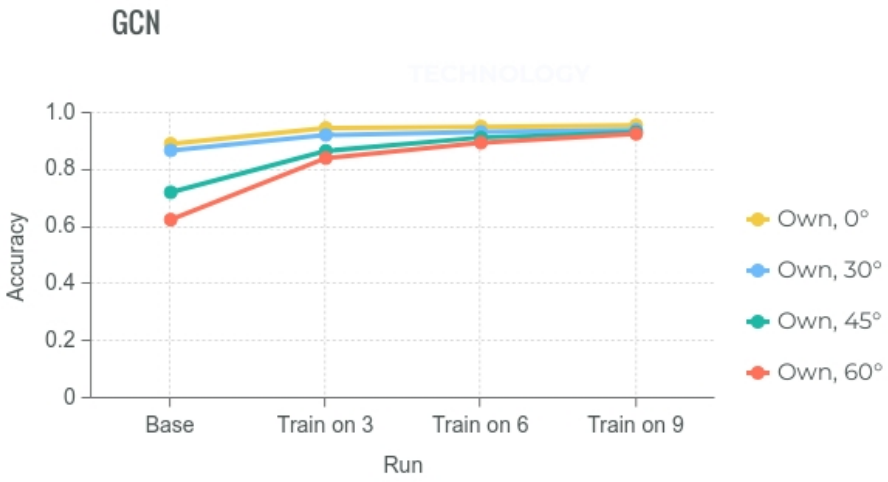


Figure 7.17: Results of GCN trained on data from 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

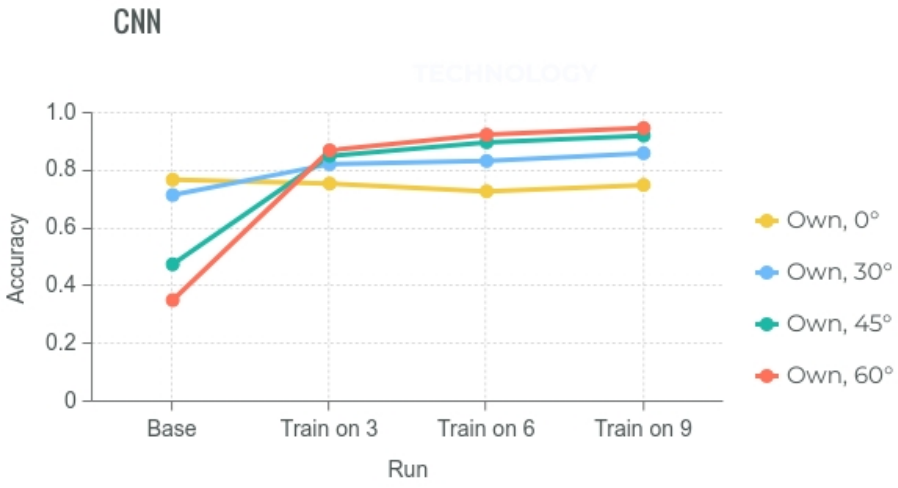


Figure 7.18: Results of CNN trained on data from 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

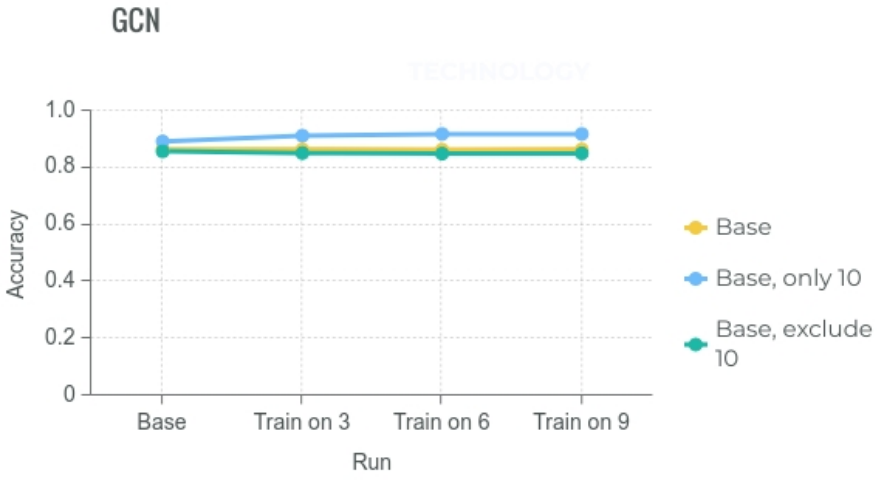


Figure 7.19: Results of GCN trained on data from 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

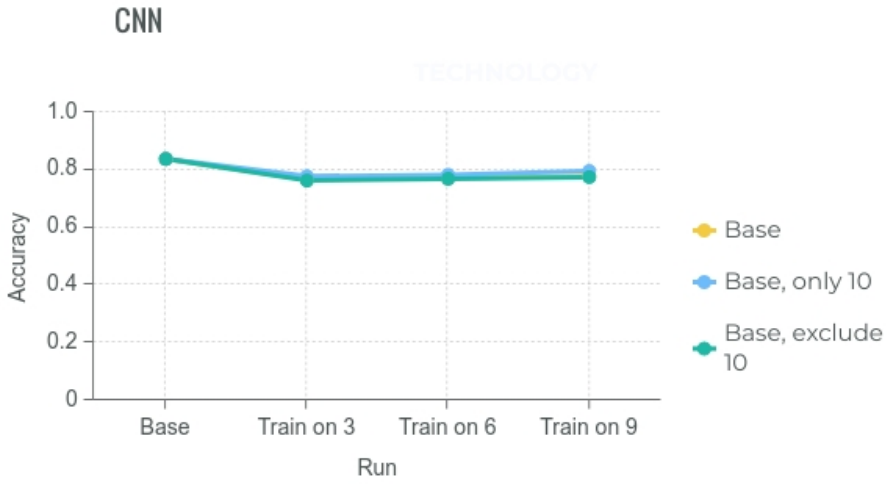


Figure 7.20: Results of CNN trained on data from 60°. **X-axis**, the data this particular network was trained on (base, 3, 6 and 9 subjects). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

7.6 Test 6

The goal of this test was to see how well fitting to a new angle works when only training on a subset of actions. Essentially, it is to test if training on actions from a new angle improves performance of other actions from that angle that have not been trained on.

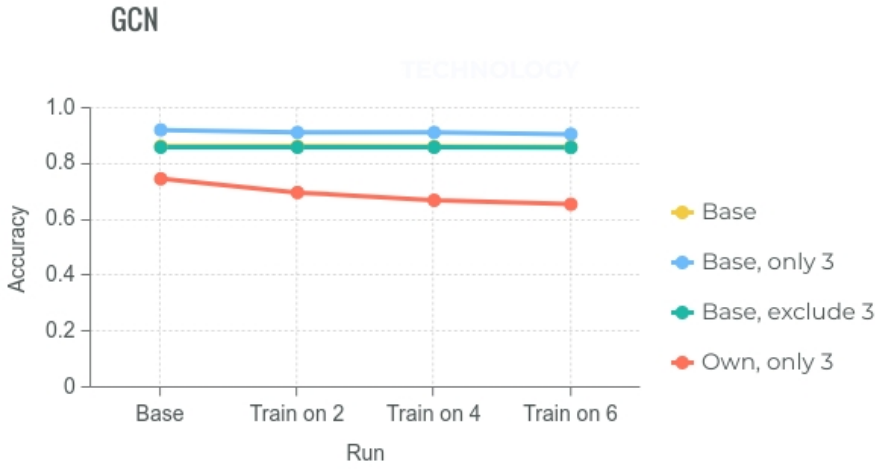


Figure 7.21: Results of GCN trained on data of specific actions from 45°. **X-axis**, the data this particular network was trained on (base, 2, 4 and 6 actions). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

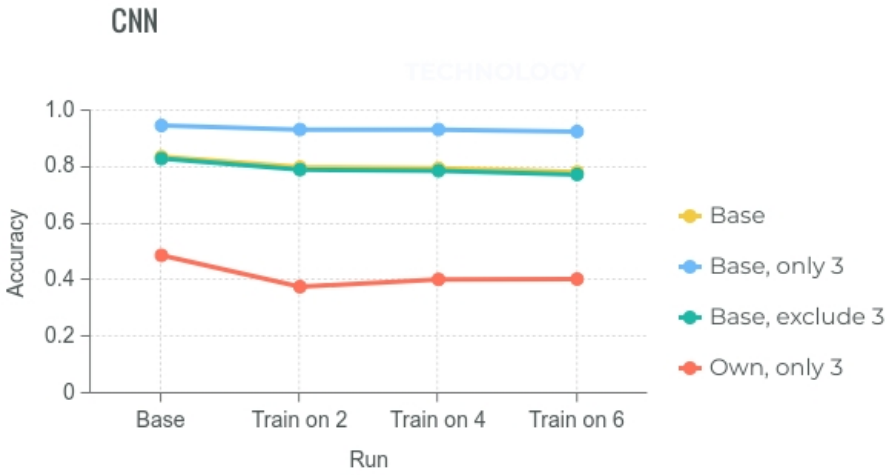


Figure 7.22: Results of CNN trained on data of specific actions from 45°. **X-axis**, the data this particular network was trained on (base, 2, 4 and 6 actions). **Y-axis**, correct predictions out of total predictions. **Lines**, represent the performance on the respective test set.

Chapter 8

Discussion

8.1 Test 1

This test was intended to be a baseline test to get an understanding of the difficulties and possibilities of testing. Looking at the result for the GCN in figure 7.1, we see a slight increase in accuracy across the board while a slight decrease in accuracy for all of the base tests in 7.3 except for 'Base, only 10'. We see that for the CNN, the accuracy on all of the base test sets in figure 7.4 decreases by quite a lot, while the accuracy on the new domain in figure 7.2 increases substantially. This is a sign of overfitting towards the 10 actions from our dataset. By not including any data from the other 39 actions, the predictions are swaying too much towards the 10 actions. This test as opposed to the rest contains only data from our dataset, this was a change made as a response to these test results. Including a roughly equivalent amount of data from each action is important not to overfit.

Otherwise, the performance on the new angles are improved but can not be used as a good indication of the methods efficacy because of previously mentioned overfitting.

8.2 Test 2, 3, 4 and 5

Test 3, 4 and 5 can be used to evaluate the importance of angles and the effect of the transfer learning. Test 2 was included in order to ascertain how big of a factor the different setup used in our dataset has in transfer learning.

For test 2, it is apparent in graphs 7.5 and 7.6 that accuracy generally increases, when transfer learning with an increasing amount of data. Optimally, we would see no change in accuracy in this test as the data from our own dataset that was added came from the same angle as the base NTU RGB+D dataset. As the amount of data of each action is roughly the same, this tells us that the setup

of our recording could have an effect on the results which will have to be taken into account when analyzing further. It may also be due to the fact that we are validating on data from our dataset, which means that it is only validating on 10 of the 49 actions and the final network was chosen by smallest validation loss.

It is clear that from test 3, 4 and 5 that adding data from a vertical angle improves performance not only from the angle of the training addition but from other angles as well. However, it is unclear how big of a factor the setup of the recording has played.

When looking at the amount of training data required, it is clear that even with a small amount of training data, a large performance improvement can be achieved. In graph 7.17, simply adding data from 3 subjects from 60° raises the accuracy from 62.7% to 84.3% on data from that angle, while keeping the overall accuracy on the base set seen in graph 7.19 the same. To put this in perspective, 3 subjects represent 150 short clips. While more training data improves accuracy further, in certain cases it negatively affects the performance on the base test. It does however indicate that a small amount of new data is required for a substantial increase in performance from a new domain.

8.3 Test 6

This was an interesting test to see whether or not the network could learn general features for a new angle. As can be seen in graph 7.21 and 7.22, it does little to improve the performance on 'Own, only 3'. It is likely that instead of generalizing for a new angle, it learns to associate the new angles as a feature of the actions themselves for the 7 actions from where it got data from.

8.4 CNN vs GCN

In this experiment, it was known that the performance of the GCN is overall higher than the CNN used. The comparisons that can be made are about the relative increase or decrease in performance in the different tests. As can be seen from the results from test 1 in graph 7.2 and 7.4, the CNN overfits a lot more than the GCN, 'Base, exclude 10' goes from 83.8% accuracy to 39.8% when training on 9 subjects. This difference might be due to the GCN being able to recognise the relationship between joints in a more sophisticated way that the CNN can not. This is also apparent when looking at the base models performance on the angles 45° and 60°. The GCN still manages to keep a relatively high accuracy for these angles with the values 72.3% and 62.7%. The CNN's performance is far lower with the values 37.7% and 35.3%.

In test 2, comparing graphs 7.5 and 7.6 we see that there is an increase in accuracy on our own dataset. For the CNN however, the accuracy for the 60°

angle only increases by 1 percentage point compared to 7 percentage points in the GCN. In test 3 for the CNN, the accuracy for the 60° angle is increased by 33% when trained on 9 subjects, see graph 7.10. At the same time the accuracy on our own dataset from degree 0° drops compared to test 2. The difference in accuracy on the 60° angle between these tests together with the decrease in accuracy on angle 0° shows that the CNN does not perform well on multiple angles at the same time.

The results in graph 7.14 seem to confirm this idea. When training on 45° instead of 30° the accuracy on 0° decreases from 89.3% to 80.1% while the increase on 60° is from 68.3% to 91.5%. Table 7.18 shows again that when the accuracy on a steeper angle increases, the accuracy lowers by a substantial amount for the less steep angle. In the case of training on 9 subjects, the accuracy on 0° and 60° is 75.2% and 94.9%. In test 6, the conclusion is the same as for the GCN, we can see in graph 7.22 that the accuracy decreases on every test.

Compared to the results for the GCN, where increasing the vertical angle of the training data, the performance on smaller angles remains largely unchanged. For example, the accuracy on 30° when training on 9 subjects is 95.7%, 94.7%, 94.2% when training on data from 30°, 45°, 60° respectively. This indicates that for the GCN it might be sufficient to include one rather high angle as training data, even for use cases with less steep angles.

The CNN seems to lack in generalization between different angles. This might be due to the CNN not having the power to deal with complex relations between joints. The reason why the CNN is not capable of dealing with the internal relationship between different joints could be due to the fact that convolutional operations on an image analyses the image locally. Looking at image 4.2 created for the CNN, even though for example joint 2 and 15 might be important to each other to classify a specific action, the image created with the proposed method will not see this relationship since the two joints are visualized as rows far from each other in the final psuedo image. On the other hand, the GCN can create new connections and learn new complex features that are specific to a particular action.

8.5 Applications

As the results indicate, quite a small amount of data can increase the performance of a networks ability to identify the action. In a real life scenario, a camera could be mounted high up in a security position, clips could be recorded and used to modify a pre-trained action recognition network. This could prove a way to easily adapt an action recognition network to a slight modification in viewing angle without having to record an entire new large dataset.

8.6 Future work

This thesis investigates a theoretical and quite constrained environment for action recognition networks performance on new angles. In a real scenario, the input videos are most likely continuous which will make the problem more difficult. A method for dealing with this would be a sliding window technique where every n:th frame, the networks try to identify what action the detected person was performing.

Another constraint in this work is the processing speed for both the skeleton extraction and action inference. It is not in real time. Another approach would be to use simpler networks to try and achieve a real time application.

Out of the 3 different types of skeleton-based action recognition networks, we only had time to investigate a CNN and GCN. How well an RNN responds to this type of transfer learning is yet to be seen.

Another possible way of adapting an action recognition network to function from a new angle that we would have liked to try given enough time, is an autoencoder. This autoencoder could learn in a general manner how to transform an action from 0° to a higher degree.

A more complete dataset would be able to further validate our findings. By having access to data from the new vertical angles from the 39 remaining actions we could make more general conclusions. Preferably would be to have the entire dataset be with the same setup. For example, if with our own setup we had thousands of clips of every action, we could train a baseline network on the clips from 0°. We could then test our hypotheses with the data from the larger vertical angles without worrying about the difference in setup and the effect it might have.

Chapter 9

Conclusion

When using an action recognition network trained for a different use case you require, it is evidently possible to adapt it to fit another vertical angle. The results from this investigation shows that a small dataset will significantly increase the performance of a pre-trained action recognition network for the new angle. As few as 15, 2-4 second long videos per action could increase the performance by 15-20 percentage points on those specific actions. The results are only apparent for the actions from the new angle that have been used in training. Otherwise, the results seem to worsen.

Of the two different action recognition networks tried, the GCN outperforms the CNN in almost every single way. It not only has a higher accuracy overall but responds to transfer learning in such a way that accuracy in the old domain remains largely the same when attempting to add a new one. In addition, it is able to generalize well enough that a single large vertical angle is sufficient for improving performance on any intermediate vertical angles.

It is obvious that it is more difficult to train a network when introducing new vertical angles since the domain size increases. For the case where a camera is to be used as stationary, the corresponding network should be tailored to that scenario since it will only see those certain angles.

Appendix A

Other pose estimation networks

A.1 OpenPose

OpenPose [3] is a well-performing pose estimation network that performs inference in real-time. It utilizes a bottom-up architecture which extracts features from images using a VGG-19 net. These features are then utilized to predict keypoints and part affinity fields (PAF) which are 2D vector fields for each limb. As a large collection of keypoints by itself is hard to correctly connect into accurate skeletons, the keypoints and orientation of the limbs through the PAFs are used together to associate keypoints. The runtime of OpenPose is constant no matter the amount of people in a scene.

A.2 HigherHRNet

HigherHRNet [6] is a bottom-up pose estimation network that utilizes HRNet as a backbone. Following the HRNet backbone is a 4x4 transposed convolution module that upsamples the concatenation of the feature maps and predicted heatmaps to a higher resolution. Which are then refined by 4 residual blocks. The output heatmaps from every scale are then aggregated by bilinear interpolation to the size of the input image, followed by averaging over all heatmaps.

A.3 Convolutional Pose Machines (CPM)

CPM [39] uses sequences of predictors that generates confidence heatmaps on types of keypoints, which are then refined over multiple stages. The initial stage onlyhas the image as input, every subsequent stage uses the previous heatmaps de-tected in addition to the original image to better predict the key-

points. Thisallows the CPM to better understand the relationship between keypoints.

A.4 Stacked Hourglass

Stacked Hourglass [28] utilizes CNNs on multiple scales which enables it to recognize larger features such as limb arrangement and keypoint relationships. It works by max pooling the input image several times and before each max pooling step, branching out to perform additional convolutions on the pre-pooling resolution. Reaching a minimum resolution of 4x4, it then upsamples and combines features across adjacent scales.

A.5 Multi-Stage Pose Estimation Network (MSPN)

MSPN [22] combines several GlobalNet of CPN [5] modules in sequence to form different stages. Information is aggregated from the previous stage and combined with the features of the current stage to be able to make full use of the previous information. In addition, intermediate supervision is utilized at the end of each stage. The ground truth keypoints which are represented as Gaussian heat maps use different size kernels for each stage. The later the stage is in the pipeline, the smaller the kernel, which refines accuracy gradually.

A.6 Unbiased Data Processing (UDP)

UDP [13] is a data processing step which minimizes error accrued when transforming coordinate systems and keypoint format. Heatmaps for keypoint confidence are becoming more common as targets for convolutional networks, UDP uses a transformation that minimizes precision degeneration when transforming between the two. Additionally, when using image augmentation such as, flipping, rotating, etc., precision degradation is common when transforming between coordinate systems. UDP proposes a unified definition of the data in continuous space as well as transformations for specific operations such as flipping, resizing, etc.

A.7 Associative Embedding (AE)

AE [27] is an easily integrated technique that simultaneously detects keypoints and groups them together into skeletons. For each keypoint detection, there is an identity tag embedding predicted which is trained such that two identity tags with keypoints belonging to the same body are to have similar values. When matching, joints around the head and torso are considered first, subsequent joints are matched by comparing identity tags and assigned to the most likely

person. When below a certain threshold of matching score, the joint is assumed to belong to a new person, perhaps only a leg is visible for instance.

Appendix B

Other action recognition networks

B.1 Ind-RNN

In [21], the idea is to use a Recurrent Neural Network [10] where the neurons are independent in a layer but connected across layers. An Ind-RNN can be regulated to avoid vanishing or exploding gradients as well as build deeper networks. Previously RNN's used saturated activation functions such as the sigmoid to keep the gradients from exploding but with this comes the problem of the gradient decay. This network manages to use non saturated activation functions such as ReLU which allows the network to become deeper without the gradient decay. The network reaches an accuracy of 81.8% cross subject and 87.97% cross view on NTU RGB+D.

B.2 Dense Ind-RNN

In [20] a version of the Ind-RNN is introduced which instead of using skip connections between the layers using a concatenation operation to combine all the features of the previous layers. Compared to a residual RNN, Dense-RNN creates an identity mapping between all layers, leading to feature reuse and reducing the amount of parameters. The Dense-RNN reaches an accuracy of 83.38% cross subject and 91.81% cross view on NTU RGB+D.

B.3 Action Machine

In [43], the idea is that to avoid overfitting due to similar backgrounds, the video is cropped using a network to detect humans and extract bounding boxes. This cropped video is used as input into I3D [4] for RGB-based action recognition. A 2D deconvolution is then added to the last layer of I3D for frame-wise pose

estimation. This pose in then used in a 2d CNN for pose-based action recognition. The output of the two methods are then summed to generate an output based on both predictions. This method achieves a CV accuracy of 97.2% and a CS accuracy of 94.3% on NTU RGB +D.

B.4 RNX3D101

By fusing skeleton data with RGB data, a richer representation can be found. Many actions contains objects which are nearly impossible to predict with only the skeleton. A two-stream framework is formed by combining MS-AAGCN with a pre-trained 3D convolutional network ResNeXt3D-101[33] model. Ultimately, this did not seem fitting as we did not have actual 3D spatial data but rather 2D spatial data with an added confidence third dimension. Additionally, this was outside of our scope and not included.

Appendix C

Tables of results

C.1 Test 1

GCN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.865	0.830	0.825	0.819
Base, only 10	0.893	0.870	0.888	0.894
Base, exclude 10	0.858	0.819	0.809	0.800
Own, 0°	0.893	0.917	0.932	0.936
Own, 30°	0.870	0.905	0.921	0.926
Own, 45°	0.723	0.809	0.830	0.837
Own, 60°	0.627	0.742	0.773	0.785

Table C.1: Performance of GCN while training on different amounts of data from angles 30°, 45° and 60°.

CNN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.838	0.522	0.447	0.414
Base, only 10	0.838	0.684	0.666	0.652
Base, exclude 10	0.838	0.512	0.433	0.398
Own, 0°	0.770	0.960	0.971	0.980
Own, 30°	0.717	0.952	0.969	0.975
Own, 45°	0.477	0.944	0.958	0.963
Own, 60°	0.353	0.936	0.956	0.966

Table C.2: Performance of CNN while training on different amounts of data from angle 30°.

C.2 Test 2

GCN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.865	0.864	0.865	0.865
Base, only 10	0.893	0.902	0.912	0.914
Base, exclude 10	0.858	0.854	0.853	0.852
Own, 0°	0.893	0.946	0.959	0.967
Own, 30°	0.870	0.901	0.922	0.930
Own, 45°	0.723	0.775	0.805	0.817
Own, 60°	0.627	0.665	0.687	0.698

Table C.3: Performance of GCN while training on different amounts of data from angle 0°.

CNN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.838	0.778	0.777	0.787
Base, only 10	0.838	0.777	0.763	0.784
Base, exclude 10	0.838	0.778	0.781	0.787
Own, 0°	0.770	0.909	0.939	0.944
Own, 30°	0.717	0.814	0.823	0.826
Own, 45°	0.477	0.543	0.521	0.568
Own, 60°	0.353	0.343	0.347	0.364

Table C.4: Performance of CNN while training on different amounts of data from angle 0°.

C.3 Test 3

GCN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.865	0.864	0.865	0.864
Base, only 10	0.893	0.903	0.913	0.914
Base, exclude 10	0.858	0.855	0.852	0.852
Own, 0°	0.893	0.946	0.959	0.968
Own, 30°	0.870	0.912	0.945	0.957
Own, 45°	0.723	0.800	0.832	0.843
Own, 60°	0.627	0.708	0.751	0.787

Table C.5: Performance of GCN while training on different amounts of data from angle 30°.

CNN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.838	0.782	0.789	0.781
Base, only 10	0.838	0.896	0.907	0.903
Base, exclude 10	0.838	0.775	0.777	0.773
Own, 0°	0.770	0.874	0.881	0.893
Own, 30°	0.717	0.893	0.927	0.936
Own, 45°	0.477	0.775	0.798	0.821
Own, 60°	0.353	0.630	0.638	0.683

Table C.6: Performance of CNN while training on different amounts of data from angle 30°.

C.4 Test 4

GCN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.865	0.868	0.867	0.867
Base, only 10	0.893	0.916	0.922	0.924
Base, exclude 10	0.858	0.856	0.853	0.853
Own, 0°	0.893	0.942	0.954	0.956
Own, 30°	0.870	0.922	0.938	0.947
Own, 45°	0.723	0.866	0.909	0.938
Own, 60°	0.627	0.818	0.866	0.887

Table C.7: Performance of GCN while training on different amounts of data from angle 45°.

CNN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.838	0.767	0.780	0.781
Base, only 10	0.838	0.789	0.792	0.789
Base, exclude 10	0.838	0.761	0.777	0.778
Own, 0°	0.770	0.795	0.782	0.801
Own, 30°	0.717	0.858	0.891	0.902
Own, 45°	0.377	0.889	0.922	0.936
Own, 60°	0.353	0.842	0.893	0.915

Table C.8: Performance of CNN while training on different amounts of data from angle 45°.

C.5 Test 5

GCN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.865	0.865	0.864	0.865
Base, only 10	0.893	0.913	0.919	0.919
Base, exclude 10	0.858	0.852	0.850	0.851
Own, 0°	0.893	0.948	0.953	0.958
Own, 30°	0.870	0.924	0.935	0.942
Own, 45°	0.723	0.868	0.915	0.934
Own, 60°	0.627	0.843	0.897	0.928

Table C.9: Performance of GCN while training on different amounts of data from angle 60°.

CNN				
Tests	Base	Train on 3	Train on 6	Train on 9
Base	0.838	0.766	0.772	0.779
Base, only 10	0.838	0.778	0.783	0.796
Base, exclude 10	0.838	0.763	0.769	0.775
Own, 0°	0.770	0.757	0.730	0.752
Own, 30°	0.717	0.824	0.835	0.862
Own, 45°	0.377	0.853	0.900	0.923
Own, 60°	0.353	0.872	0.926	0.949

Table C.10: Performance of CNN while training on different amounts of data from angle 60°.

C.6 Test 6

GCN				
Tests	Base	Train on 2	Train on 4	Train on 6
Base	0.865	0.865	0.864	0.863
Base, only 3	0.923	0.914	0.914	0.908
Base, exclude 3	0.861	0.861	0.861	0.860
Own, only 3	0.748	0.699	0.671	0.657

Table C.11: Performance of GCN while training on different amounts of actions from our own dataset from 45°.

CNN				
Tests	Base	Train on 2	Train on 4	Train on 6
Base	0.838	0.801	0.797	0.784
Base, only 3	0.948	0.934	0.934	0.927
Base, exclude 3	0.831	0.792	0.788	0.775
Own, only 3	0.489	0.378	0.404	0.405

Table C.12: Performance of CNN while training on different amounts of actions from our own dataset from 45°.

Bibliography

[1] Mykhaylo Andriluka, Umar Iqbal, Eldar Insafutdinov, Leonid Pishchulin, Anton Milan, Juergen Gall, and Bernt Schiele. Posetrack: A benchmark for human pose estimation and tracking. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5167–5176, 2018.

[2] Mykhaylo Andriluka, Leonid Pishchulin, Peter Gehler, and Bernt Schiele. 2d human pose estimation: New benchmark and state of the art analysis. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3686–3693, 2014.

[3] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1):172–186, 2021.

[4] João Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4724–4733, 2017.

[5] Yilun Chen, Zhicheng Wang, Yuxiang Peng, Zhiqiang Zhang, Gang Yu, and Jian Sun. Cascaded pyramid network for multi-person pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[6] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S Huang, and Lei Zhang. Higherhrnet: Scale-aware representation learning for bottom-up human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5386–5395, 2020.

[7] MMPose Contributors. Openmmlab pose estimation toolbox and benchmark. `https://github.com/open-mmlab/mmpose`, 2020.

[8] J. Deng, K. Li, M. Do, H. Su, and L. Fei-Fei. Construction and Analysis of a Large Scale Image Ontology. Vision Sciences Society, 2009.

[9] M. A. Fischler and R. A. Elschlager. The representation and matching of pictorial structures. *IEEE Transactions on Computers*, C-22(1):67–92, 1973.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[11] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory, 1995.

[13] Junjie Huang, Zheng Zhu, Feng Guo, and Guan Huang. The devil is in the details: Delving into unbiased data processing for human pose estimation. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[14] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[15] Hueihan Jhuang, Juergen Gall, Silvia Zuffi, Cordelia Schmid, and Michael J. Black. Towards understanding action recognition. In *2013 IEEE International Conference on Computer Vision*, pages 3192–3199, 2013.

[16] Yanli Ji, Yang Yang, Fumin Shen, Heng Tao Shen, and Wei-Shi Zheng. Arbitrary-view human action recognition: A varying-view rgb-d action dataset. *IEEE Transactions on Circuits and Systems for Video Technology*, 31(1):289–300, 2021.

[17] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

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

[19] Sohaib Laraba. *Deep Learning for Skeleton-Based Human Action Recognition*. PhD thesis, 10 2020.

[20] Shuai Li, Wanqing Li, Chris Cook, Yanbo Gao, and Ce Zhu. Deep independently recurrent neural network (indrnn), 2019.

[21] Shuai Li, Wanqing Li, Chris Cook, Ce Zhu, and Yanbo Gao. Independently recurrent neural network (indrnn): Building a longer and deeper rnn. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5457–5466, 2018.

[22] Wenbo Li, Zhicheng Wang, Binyi Yin, Qixiang Peng, Yuming Du, Tianzi Xiao, Gang Yu, Hongtao Lu, Yichen Wei, and Jian Sun. Rethinking on multi-stage networks for human pose estimation, 2019.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[24] Jun Liu, Amir Shahroudy, Mauricio Perez, Gang Wang, Ling-Yu Duan, and Alex C. Kot. Ntu rgb+d 120: A large-scale benchmark for 3d human activity understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(10):2684–2701, Oct 2020.

[25] T. L. Munea, Y. Z. Jembre, H. T. Weldegebriel, L. Chen, C. Huang, and C. Yang. The progress of human pose estimation: A survey and taxonomy of models applied in 2d human pose estimation. *IEEE Access*, 8:133330–133348, 2020.

[26] Vinod Nair and Geoffrey E. Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, page 807–814, Madison, WI, USA, 2010. Omnipress.

[27] Alejandro Newell, Zhiao Huang, and Jia Deng. Associative embedding: End-to-end learning for joint detection and grouping. In *Advances in neural information processing systems*, pages 2277–2287, 2017.

[28] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[29] Michael A. Nielsen. Neural networks and deep learning, 2015.

[30] R. Poppe. A survey on vision-based human action recognition. *Image Vis. Comput.*, 28:976–990, 2010.

[31] Geoffrey E. Williams Ronald J. Rumelhart, David E. Hinton. Learning representations by back-propagating errors. 323, oct 1986.

[32] Amir Shahroudy, Jun Liu, Tian-Tsong Ng, and Gang Wang. Ntu rgb+d: A large scale dataset for 3d human activity analysis, 2016.

[33] L. Shi, Y. Zhang, J. Hu, J. Cheng, and H. Lu. Gesture recognition using spatiotemporal deformable convolutional representation. In *2019 IEEE International Conference on Image Processing (ICIP)*, pages 1900–1904, 2019.

[34] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Skeleton-based action recognition with multi-stream adaptive graph convolutional networks, 2019.

[35] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Two-stream adaptive graph convolutional networks for skeleton-based action recognition, 2019.

[36] Lucas Smaira, João Carreira, Eric Noland, Ellen Clancy, Amy Wu, and Andrew Zisserman. A short note on the kinetics-700-2020 human action dataset, 2020.

[37] Joëlle Tilmanne Sohaib Laraba and Thierry Dutoit. Action recognition based on 2d skeletons extracted from rgb videos, 01 2019.

[38] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5686–5696, 2019.

[39] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 4724–4732, 2016.

[40] Sijie Yan, Yuanjun Xiong, and Dahua Lin. Spatial temporal graph convolutional networks for skeleton-based action recognition, 2018.

[41] Li Yang and Abdallah Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316, Nov 2020.

[42] Feng Zhang, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. Distribution-aware coordinate representation for human pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7093–7102, 2020.

[43] Jiagang Zhu, Wei Zou, Liang Xu, Yiming Hu, Zheng Zhu, Manyu Chang, Junjie Huang, Guan Huang, and Dalong Du. Action machine: Rethinking action recognition in trimmed videos, 2018.

[44] Fuzhen Zhuang, Zhiyuan Qi, Keyu Duan, Dongbo Xi, Yongchun Zhu, Hengshu Zhu, Hui Xiong, and Qing He. A comprehensive survey on transfer learning. *Proceedings of the IEEE*, 109(1):43–76, 2021.

Master's Theses in Mathematical Sciences 2021:E16
ISSN 1404-6342

LUTFMA-3441-2021

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/