technische universität
dortmund

End-to-end Human Activity Recognition on Video Datasets

Master thesis

Matthias Jakobs
November 3, 2020

Supervisors:
Prof. Dr.-Ing. Gernot A. Fink
Fernando Moya Rueda, M.Sc.

Fakultät für Informatik
Technische Universität Dortmund
http://www.cs.uni-dortmund.de

CONTENTS

1 INTRODUCTION

Understanding human behaviour is one of the main goals of artificial intelligence research. One approach towards achieving this goal is called Human Activity Recognition. Human Activity Recognition (HAR) is the process of recognizing specific gestures or actions performed by humans from different sources, e.g., images, videos or sensor data. Knowledge of actions performed by humans is useful in different application contexts, such as video surveillance, human-machine interaction or for evaluating worker performance in a warehouse setting [RMRHF18].

One approach for predicting human actions is to use pose information as an input to a HAR machine learning model, such as a neural network. A pose is a set of keypoints on the person's body, usually joints between limbs. For images, a pose is usually given by a set of pixel coordinates. Each pixel coordinate refers to one estimated joint position. Poses are estimated using a dedicated machine learning model, which is often based on neural networks as well. In [JGZ+13], the authors find that using human pose information is useful when training HAR models, more so than other features such as image features.

In Human Activity Recognition research, a pose estimation model is often used to precompute the pose for each image. In [LPT18], the authors argue that jointly learning the pose and the action using a convolutional neural networks may improve the learning process of the action predictor. This approach of jointly learning was previously not possible with many pose estimators since the output needed additional postprocessing steps. Many pose estimation models output joint heatmaps, which contain the likelihood of a joint being present for each pixel in the input image. A function called argmax extracts the pixel coordinates with the highest likelihood from the joint heatmaps. The argmax function is, however, not differentiable. The optimization algorithms used for training neural networks require a fully differentiable network for learning the parameters of the network. With the introduction of the Soft-argmax function in [LTP17], the authors presented an approach for extracting exact pose from the joint heatmaps using a differentiable function. This function makes it possible to train the pose estimator and action recognition model jointly.

The first objective of this thesis is to use the Convolutional neural network model proposed by [LPT18] to recreate their results. This includes investigating the performance of the network using different hyperparameters. Second, the model is evaluated

using a more challenging benchmark to gain a better understanding of how well the model performs on different data. Third, the network is trained in an end-to-end approach. The authors in [LPT18] use a technique called pretraining for the pose estimator part of the network. Pretraining means that the pose estimator part of the network is trained indepently of the rest of the network. The pretrained parameters of the pose estimator are then transferred to the complete network and the network is finetuned. A different approach for training the network is called end-to-end learning and refers to the method where the parameters of the network are initialized randomly. From this random initialization, the entire network is trained jointly. While pretraining certain parts of a network often leads to a faster convergence of the model, it also adds more complexity to the training process, because two training processes need to be optimized.

This thesis is divided into six chapters, including this introduction. Chapter 2 explains the fundamentals of Human Activity Recognition in the context of video data, pose estimation using image data as well as artificial and convolutional neural networks. Recent relevant work in the fields of HAR and pose estimation are discussed in detail in Chapter 3. There, a focus is set on Human Activity Recognition methods using pose information. Next, a detailed explanation of the methods of [LPT18] are presented in Section 4.1. Chapter 4.1 also discusses limitations to the approach by [LPT18] and further experiments conducted in this thesis. In Chapter 5, the experiments, as well as the used datasets and metrics, are explained. Moreover, the results of the proposed experiments are discussed. Finally, a conclusion of the findings from the experiments are presented in Chapter 6.

2 FUNDAMENTALS

After introducing and motivating the problems this thesis focuses on in the previous chapter, this chapter gives an overview over the fundamentals of Human Action Recognition, Pose estimation as well as Neural Networks in general.

First, the fundamental concepts of Human Action recognition are discussed in Section 2.1, specifically in the context of video data. Afterwards, 2D pose estimation for still images is introduced in Section 2.2. Since many methods discussed in this thesis, including the approach by [LPT18], use Neural networks, this thesis also introduces them in Section 2.3, beginning with *Artificial Neural networks* and how they are trained. Afterwards, the use of *Convolutional Neural networks*, often used in image processing, is motivated and explained.

2.1 HUMAN ACTION RECOGNITION

In *Human Action Recognition*, often referred to simply by its acronym *HAR*, the task is to attach a label of an *action* to a signal, e.g. an image, video or sensor meassurements. Thus, HAR is a *classification* problem where the classes are human actions. It is important to understand that HAR recognizes the action after it is completed, as opposed to *action prediction*, which tries to predict the action while it is still happening [KF18].

Typical use-cases of Human Action Recognition consists of evaluating human behaviour, for example in a warehouse context [RMRHF18], or analyzing video surveillance footage [HKRA14]. An example of simple actions performed by multiple subjects is provided in (**Fig.** 2.1.1).

2.1.1 *Action Granularity*

When defining what an *action* is, it is important to think about the degree of granularity needed. As an illustration, consider an image of a person waving with her left hand at the camera. It is not inherently clear whether the label attached to this action should be *waving* or *raising left arm*. This is why the choice of labels is often domain and use-case specific.

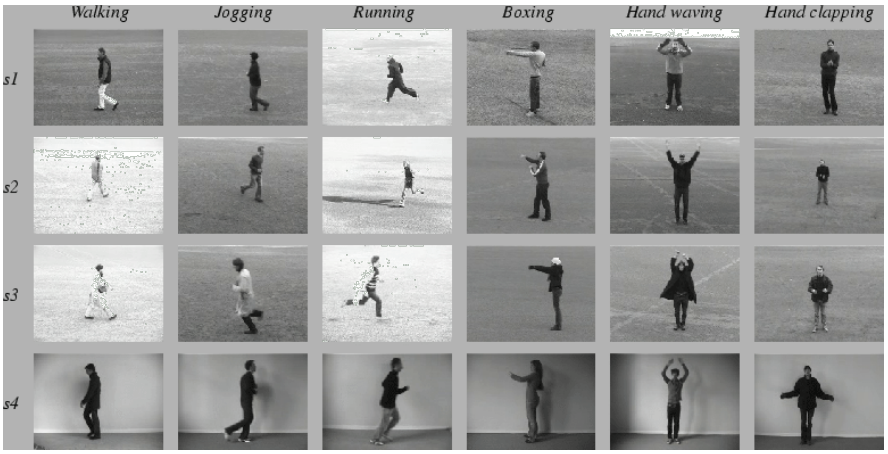


Figure 2.1.1: Example of six actions performed by four different subjects, annotated with their corresponding label (top). Image taken from [LMSR08].

According to [ZWN+17], human actions can be categorized into three levels, referred to as *action primitives, activities and interactions*. *Action primitives* consists of actions where one part of the body is performing the actions. For example, *waving* or *clapping* would constitute action primitives since a specific part of the body is responsible. In contrast, *activities* are actions where the whole body is involved in performing the action. As an example, consider *jumping* or *jogging*. When considering actions performed involving objects or other persons, e.g. *shaking hands* or *throwing a ball*, [ZWN+17] categorize these actions as *interactions*.

2.1.2 *Video-based HAR*

When considering HAR on video clips, some domain-specific problems arise, which are outlined in [ZWN+17].

Firstly, depending on the camera and scene, the background of the image can be highly dynamic. This means that the amount of information irrelevant for identifying the action can be very high and might change constantly. As an example, consider a video filmed with a hand-held modern smart phone. The camera is not stationary, so the background will shift while recording. Also, depending on the scene, illumination changes and occlusion of the human subject might occur.

Secondly, different actions can have similar visual shapes. Consider *talking on the phone* and *military salute*. In both cases, the dominant hand is positioned at the side of the subjects head. Depending on factors like image quality and the subjects

rotation towards the camera, these two actions might be hard to differentiate. Also, consider *walking* and *running*. It is not inherently clear where the boundary between these two classes are, i.e., up to which point is a subject still *walking* and when does the subject begin to *run*? This problem is referred to as *interclass similarity*. Another similar problem is the *intraclass variation*, where the same action performed by different subjects might look very different. As an example, consider *throwing a ball*. The scene will differ a lot when considering different clothing of the subject, different shape and color of the ball as well as different contexts where the action is happening, e.g., in a backyard or in a baseball stadium. Also, performing actions at different intensities can alter their appearence, for example *running* can be performed at slow to high speed and might even involve small jumps [KF18].

Thirdly, [ZWN+17] mention *group activities*. When there are multiple subjects performing actions in a frame, it can be difficult to differentiate between many individual actions and a group action. An example would be the difference between *running* and *playing football*. Not only is a lot of context necessary to be able to determine a group activity but also to determine which subjects are part of the group activity and which are not. Consider the case where, in a single frame, two subjects are playing football and two subjects are sitting, reading a book. No single label is able to fully describe the human actions present in the image. Thus, a sizeable portion of literature focuses on single human, single action problems.

2.2 POSE ESTIMATION

Pose estimation is defined by modeling human joint positions from an image or from other signals. In this thesis, however, the main focus is performing pose estimation using image-based methods, where the pixel positions for each joint are estimated. The joint positions are then often represented in a tree structure representing a skeleton. See (**Fig.** 2.2.1) for an example.

Such a representation of the human pose is useful in many use-cases. Action recognition on images, presented in Chapter 2.1.2, often incorporates the pose of a human to identify the action performed by that human. Additionally, since computer generated characters in movies become even more prevalent in recent years, pose estimation is often used for motion capturing, where an actor's pose is used for animating a computer generated character.

The skeleton structure shown is just a visualization of the data generated. When computing pose, two main approaches are used. Firstly, many approaches use regression to directly compute the image coordinates. This approach was used in the early

Figure 2.2.1: Examples of human pose estimation. The pose is represented using a tree structure of joint positions, forming a skeleton. Image taken from [NYD16].

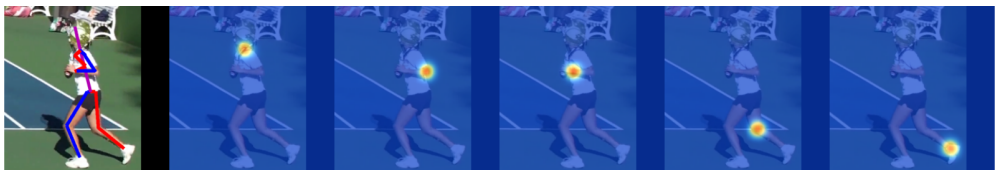


Figure 2.2.2: Representing joint coordinates as probability heatmaps. From left to right: Tree structure representation of all joints, left shoulder, left elbow, left hand, right knee and right ankle. Image taken from [NYD16].

work because its reasoning is intuitive. Secondly, heatmaps are used that represent a discrete probability distribution over the x and y coordinates. One 2D heatmap corresponds to the position of one joint. An example is provided in (**Fig.** 2.2.2). To extract the joint coordinates from heatmaps, a post processing step like the *argmax* is needed. According to [LPT18], in practice, heatmap based methods outperform regression methods.

When computing pose from images, one obvious challenge is the variety in appearance because of different choices in **clothing**. Consider the difference between detecting elbows in a picture with a person wearing a T-shirt and a picture with a person wearing a jacket. Not only is the naked elbow exposed in the first example but

the overall shape of the person is distorted because of the thick fabric of the jacket in the second example. This problem also applies when considering the substantial differences in human appearance based on height and weight.

Other challenge are **occlusion** and **self-occlusion**. Occlusion happens when an object is in the line of sight of the camera, occluding parts of a joint or the whole joint. Self-occlusion means that the subject in the image is positioned to the camera in such a way that their own body occludes the joints.

When focusing on detecting the pose of a single human, other **humans in the background** might complicate the process because their joints could be wrongly recognized as belonging to the desired subject. Consider again the examples provided in (**Fig.** 2.2.1). It is easy to see how such errors can occur in crowded environments.

According to [Zhu16], there are two general approaches for pose estimation. First, in *top-down pose estimation*, a generic model of a human is used as a starting point. This model is then updated based on the information gathered from the images. That way, there is always a model where all joints are present, i.e., occlusion is handled easily. This approach, however, requires a priori assumptions about the human body and may lead to bad results when these assumptions are not true, i.e., people with disabilities or exceptionally tall or heavy people. Second, *bottom-up pose estimation* focuses on detecting individual body parts. These individual parts can then be combined together into the final pose representation. As part detectors got significantly more accurate in recent years due to the development of *deep convolutional neural networks*, this is the predominant approach for pose estimation in the current literature.

2.3 NEURAL NETWORKS

Neural networks are becoming increasingly popular in modern computer vision and machine learning pipelines due to their high classification accuracy. In the following chapter, an introduction into their functionality is given, starting with the McCulloch-Pitts-Neuron. Then, the Perceptron is discussed, which generalizes the McCulloch-Pitts-Neuron to real numbers. Also, an outline of how Perceptrons learn from data is presented. Afterwards, multiple Perceptrons are combined into a network to solve more complex tasks. Finally, convolutional neural networks are discussed, which are very commonly used in modern computer vision literature due to their ability to learn how to extract meaningful visual features from data.

2.3.1 *Artificial Neural Networks*

Artificial Neural Networks are unidirected graphs where *neurons* are used as vertices. A neuron is a compute unit, which performs an action upon its inputs and propagates its output along the output edge. Each neuron has a set of internal parameters that determine how its output is computed. In the following sections, two approaches for how a neuron is defined are presented, as well as mechanisms for determining their internal parameters automatically. Also, the approach of constructing a neural network from a collection of neurons to compute more complex functions will be discussed.

2.3.1.1 *McCulloch-Pitts-Neuron*

One of the earliest definitions of a neuron was proposed by Warren McCulloch and Walter Pitts in 1943 [MP43]. The McCulloch-Pitts-Neuron (MCP), also referred to as the McCulloch-Pitts unit, takes binary input values $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{B}^n$ and computes a binary value $f(\boldsymbol{x}) \in \mathbb{B}^n$. Additionally, the neuron contains a threshold value $\theta \in \mathbb{N}$. After adding all input signals, the sum (also referred to as *excitation*) is compared to θ (**Eq.** 2.3.1). The output of the neuron is 1 if the excitation is greater or equal to θ and 0 otherwise.

$$f(\boldsymbol{x}, \theta) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} x_i \geqslant \theta \\ 0 & \text{otherwise} \end{cases} \tag{2.3.1}$$

This simple neuron is capable of realizing some binary operators by choosing different values for θ. For example, the **boolean OR** operator is realized by setting $\theta = 1$ and the **boolean AND** operator (over n inputs) can be implemented by choosing $\theta = n$ [Roj96].

A geometrical explanation of how the MCP works is that it separates its input space into two half-spaces, assigning the output 1 to all input combinations on one side and 0 on the other. For example, for two dimensional input spaces (two input variables x_1 and x_2), a MCP defines a separating line while for three dimensional input spaces the MCP becomes a separating hyperplane. A visualization for the **boolean OR** function with three input variables is shown in (**Fig.** 2.3.1).

The MCP looked at so far is also called an *uninhibited* MCP. [Roj96] show that *uninhibited* MCP's can only model monotonic logical functions. By adding *inhibitory*

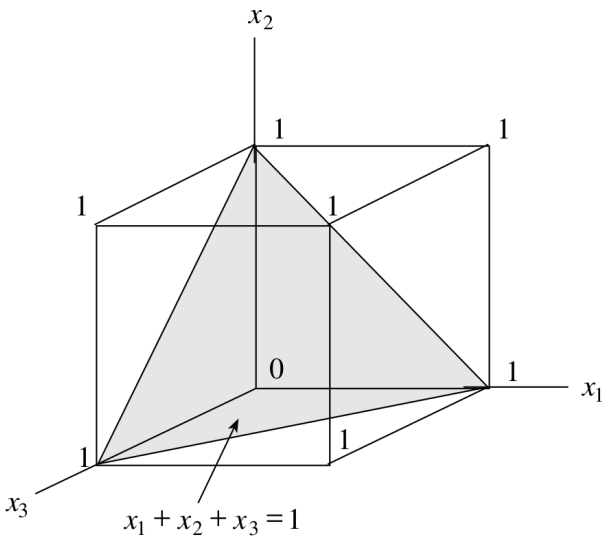


Figure 2.3.1: Example of MCP dividing the three-dimensional input space using a hyperplane. The MCP is configured to model the **boolean OR** function. Image taken from [Roj96].

inputs $\boldsymbol{y} = (y_1, \ldots, y_m) \in \mathbb{B}^m$ to the MCP, however, non-monotonic logical functions like **boolean NOT** can be implemented. The output of the MCP changes to

$$\hat{f}(\boldsymbol{x}, \boldsymbol{y}, \theta) = f(\boldsymbol{x}, \theta) \cdot \prod_{j=0}^{m} (1 - y_j). \tag{2.3.2}$$

With *uninhibited* and *inhibited* inputs a neuron can model any conjugation of negated and non-negated inputs. For example, modeling the boolean function $x_1 \wedge \neg x_2 \wedge x_3$ results to

$$\hat{f}(x_1, x_2, x_3, \theta = 1) = f(x_1, x_3, \theta) \cdot (1 - x_2). \tag{2.3.3}$$

To compute more complex functions, multiple neurons can be grouped together into *layers*, which, in turn, are connected into a *neural network*. An example of a neural network made up from multiple layers can be seen in (**Fig.** 2.3.2). The input layer does not contain neurons, however, since the nodes to not perform any computation. Rather, the input layer abstracts the input values into the neural network framework. Any layer between the input layer and the final layer is referred to as an *hidden layer*.

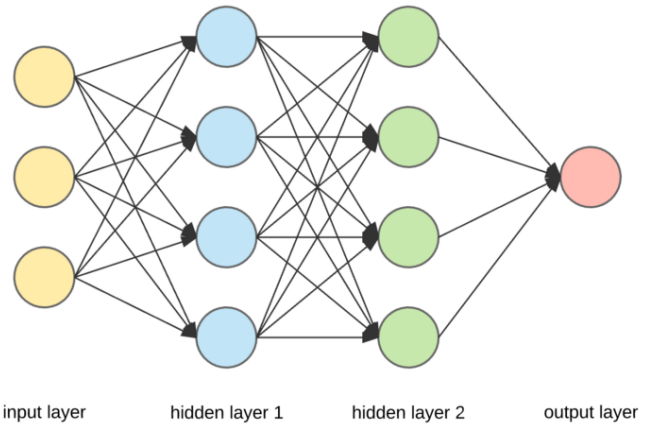


Figure 2.3.2: Example of a simple neural network, made up of two hidden layers. Image taken from [Der17a].

Each neuron in a layer receives the output of all neurons in the previous layer as input. When defining the number of layers a network has, the input layer is typically not counted because it does not contain compute units. This means that the example network in (**Fig.** 2.3.2) would be considered a three layer network.

By using a two-layer neural network it is possible to model any boolean function $f : \mathbb{B}^n \to \mathbb{B}$. The first layer consists of neurons that model conjunctions over the inputs just as presented above. The second layer is made up of a single neuron, which is configured to compute **boolean OR**. By making the outpus of the first layer the input of the disjunction in the second layer any boolean function f can be computed because any such function can be represented in disjunctive normal form.

The obvious limitation of McCulloch-Pitts-Networks is that they are limited to the domain of logical functions. Additionally, they have to be constructed rather than being able to learn the desired function because they rely on fixed connections to model relations between input variables.

2.3.1.2 *Perceptron*

In contrast to the McCulloch-Pitts-Neuron, a Perceptron uses real valued inputs $\boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$ as well as a set of real valued weights $\boldsymbol{w} = (w_1, \ldots, w_n) \in \mathbb{R}^n$:

$$f(\boldsymbol{x}, \boldsymbol{w}, \theta) = \begin{cases} 1 & \text{if } \sum_{i=0}^{n} w_i \cdot x_i \geqslant \theta \\ 0 & \text{otherwise.} \end{cases} \tag{2.3.4}$$

Instead of setting θ as part of the neuron it is preferred to treat it as an additional trainable parameter. To achieve this, a new fixed input value $x_b = 1$ with the corresponding weight $w_b = -\theta$ is added to the model and the previously *internal* θ is fixed to 0 (**Eq.** 2.3.5). This additional weight is called the *bias* of the Perceptron.

$$f(\boldsymbol{x}, \boldsymbol{w}) = \begin{cases} 1 & \text{if } -w_b + \sum_{i=0}^{n} w_i \cdot x_i \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3.5}$$

For notational convenience, from now on the bias is assumed to be part of $\boldsymbol{w}$, i.e., $\boldsymbol{w} = (w_1, \ldots, w_n, -\theta)$ and the additional input $x_b = 1$ is part of $\boldsymbol{x}$, i.e., $\boldsymbol{x} = (x_1, \ldots, x_n, 1)$.

The *excitation* of a Perceptron is still just a (weighted) linear combination. This means that the Perceptron, like the MCP, separates the input space by a hyperplane. In (**Fig.** 2.3.3) examples for some common logical functions for two input variables are provided.

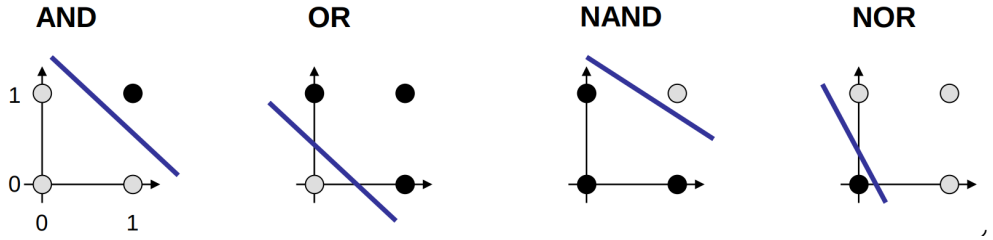


Figure 2.3.3: Logical functions modelled by a single Perceptron. The blue line indicates where the input space is divided. All input combinations on one side of the line are going to be assigned to 1 while being assigned to 0 on the other side. Notice that there are infinitely many possibilities for dividing lines since the input space is real valued. Image taken from [Rud18].

While MCP's were designed to model logical functions like **boolean OR** or **boolean AND** by setting θ as well as categorizing the input variables as either *inhibitory* or *non-inhibitory*, Perceptrons are able to infer these parameters through a *learning process*.

Perceptrons learn from a *training set* $\boldsymbol{M} = \boldsymbol{P} \cup \boldsymbol{N}$ comprised of *positive examples* and *negative examples*. A positive example $\boldsymbol{p} \in \boldsymbol{P}$ is defined as an input for which the Perceptron should output 1. Analogously, the Perceptron should output 0 for each negative example. Learning in the context of Perceptrons then means determining a parameter vector $\boldsymbol{w}$, which satisfies the following inequalities for all positive and negative examples:

$$\begin{aligned} \boldsymbol{w} \cdot \boldsymbol{p} &\geqslant 0 \ \forall \boldsymbol{p} \in \boldsymbol{P} \\ \boldsymbol{w} \cdot \boldsymbol{n} &< 0 \ \forall \boldsymbol{n} \in \boldsymbol{N}. \end{aligned} \tag{2.3.6}$$

A parameter vector, which satisfies these inequalities, then defines the hyperplane separating positive from negative training examples. The Perceptron can then assign an output value to non-training examples by computing its output using the learned weight vector. The training examples are required to be linearly separable in order to find an optimal separating hyperplane.

The general algorithmic approach determining an optimal weight vector is the following:

1. Start with a random weight vector $\boldsymbol{w}$
2. Evaluate how accurate the hyperplane defined by $\boldsymbol{w}$ separates the input space
3. If all positive and negative examples are separated correctly:
   a) **Done**
4. Else
   a) Update weight vector in a way which further reduces the error function
   b) Go to step 2

For the learning algorithm to determine the accuracy of a given weight vector $\boldsymbol{w}_i$, an **error function** or **loss function** needs to be provided. Such a function takes all positive and negative examples and calculates the amount of error, i.e., number of wrongfully classified examples.

One example of a loss function is the **sum of squared error** function:

$$\mathrm{SSE}(\boldsymbol{w}) = \sum_{\boldsymbol{x} \in \boldsymbol{M}} (\hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x)^2. \tag{2.3.7}$$

This function computes the output of the Perceptron $\hat{f}(\boldsymbol{x}, \boldsymbol{w})$ for a given weight vector $\boldsymbol{w}$ and subtracts the expected output y_x for the input $\boldsymbol{x}$. It is trivial to see that the minimum error $\mathrm{SSE}(\boldsymbol{w}) = 0$ is achieved if and only if $\hat{f}(\boldsymbol{x}) = 1$ for all positive examples and $\hat{f}(\boldsymbol{x}) = 0$ else.

Iteratively updating the weight vector needs a strategy that guarantees that the error will be less than it was before after updating. One algorithm, presented in [Roj96] and simply called *Perceptron learning algorithm*, uses the following method.

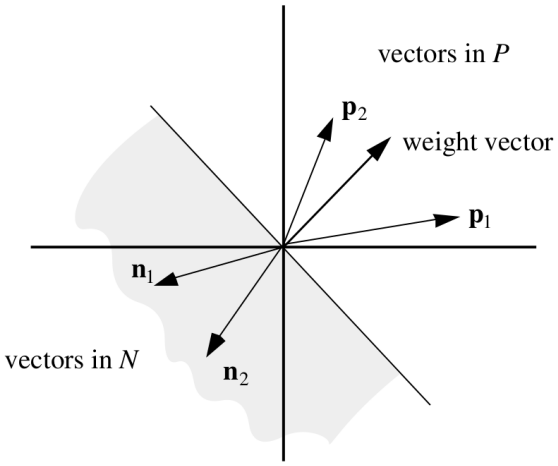


Figure 2.3.4: Visualization of the weight plane $\boldsymbol{w} \cdot \boldsymbol{x}$ separating two positive and two negative vectors. The weight vector $\boldsymbol{w}$ is the normal of the hyperplane. Image taken from [Roj96]

A training example $x \in M$ is chosen randomly. Also, as discussed before, a random weight vector $\boldsymbol{w}_t = \boldsymbol{w}_0$ is initialized. If $x \in P$ and $\boldsymbol{w}_t \cdot x \leqslant 0$ then the weight vector needs to be updated. The idea is that, in the case above, the two vectors $\boldsymbol{x}$ and $\boldsymbol{w}$ must have an angle bigger than 90 degrees (see (**Fig.** 2.3.4)). By rotating $\boldsymbol{w}$ towards $\boldsymbol{x}$ the angle will be reduced, eventually putting $\boldsymbol{x}$ on the correct side of the hyperplane. To rotate $\boldsymbol{w}$ the algorithm proposes $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t + \boldsymbol{x}$. Analogously, if $x \in N$ and $\boldsymbol{w}_t \cdot x \geqslant 0$ the algorithm proposes $\boldsymbol{w}_{t+1} = \boldsymbol{w}_t - \boldsymbol{x}$. This is done for each $x \in M$ in a random order.

As P and N are required to be linearly separable and there are a finite number of examples it can be proven that, after a finite amount of steps, the error will be reduced to zero and the hyperplane will correctly separate the two sets [Roj96].

2.3.1.3 *Gradient Descent Learning*

Another approach to learning the weights of a Perceptron is *gradient descent*. Given an error function E with a set of initial weights $\boldsymbol{w_t}$ and an input example $\boldsymbol{x}$ the amount of error is given by $E(\boldsymbol{w}, \boldsymbol{x})$. If the error function is differentiable, one can calculate the gradient of the error function for each weight $w_i \in \boldsymbol{w}$ using

$$\frac{\partial E}{\partial w_i}, \tag{2.3.8}$$

which points toward the steepest ascend of the error function.

Consider the error function defined earlier in (**Eq.** 2.3.7). The partial derivative given each $w_i \in \boldsymbol{w}$ is then given by:

$$\begin{aligned}
\frac{\partial \mathrm{SSE}(\boldsymbol{M}, \boldsymbol{w})}{\partial w_i} &= \frac{\partial}{\partial w_i} \left(\sum_{x \in \boldsymbol{M}} (\hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x)^2 \right) \\
&= \sum_{x \in \boldsymbol{M}} \frac{\partial}{\partial w_i} (\hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x)^2 \\
&= 2 \cdot \sum_{x \in \boldsymbol{M}} \hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x) \cdot \frac{\partial}{\partial w_i} (\hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x) \\
&= 2 \cdot \sum_{x \in \boldsymbol{M}} \hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x) \cdot \frac{\partial}{\partial w_i} \hat{f}(\boldsymbol{x}, \boldsymbol{w}).
\end{aligned} \tag{2.3.9}$$

This presents a challenge, however, since $\hat{f}(\boldsymbol{x}, \boldsymbol{w})$ is not differentiable. Until now, the definition of a single Perceptron was the following:

$$\begin{aligned}
\hat{f}(\boldsymbol{x}, \boldsymbol{w}) &= \begin{cases} 1 & \text{if } \boldsymbol{w} \cdot \boldsymbol{x} \geqslant 0 \\ 0 & \text{otherwise} \end{cases} \\
&= \phi(\boldsymbol{x} \cdot \boldsymbol{w}) \\
&= \phi(\psi(\boldsymbol{x}, \boldsymbol{w})),
\end{aligned} \tag{2.3.10}$$

where ψ is called the *integration function*, which computes the excitation and ϕ the *activation function* that computes the activation of the neuron.

This results in a non-differentiable activation function since the thresholding approach is not continuous, which means that it cannot be differentiable. A popular choice for a differentiable activation function is the *sigmoid activation function* $S(x)$ given by:

$$S(x) = \frac{1}{1 + e^{-x}}. \tag{2.3.11}$$

One can easily see that the sigmoid function is differentiable and that the derivative is given by:

$$\begin{aligned} \frac{d}{dx} S(x) &= \frac{e^{-x}}{(1 + e^{-x})^2} \\ &= S(x)(1 - S(x)). \end{aligned} \tag{2.3.12}$$

By then choosing $\phi = S$ and keeping $\psi(\boldsymbol{x}, \boldsymbol{w}) = \boldsymbol{x} \cdot \boldsymbol{w}$ in (**Eq.** 2.3.9) the partial derivative of the error function becomes:

$$\begin{aligned} \frac{\partial SSE(\boldsymbol{M}, \boldsymbol{w})}{\partial w_i} &= 2 \cdot \sum_{x \in \boldsymbol{M}} (\hat{f}(\boldsymbol{x}, \boldsymbol{w}) - y_x) \cdot \frac{\partial}{\partial w_i} \hat{f}(\boldsymbol{x}, \boldsymbol{w}) \\ &= 2 \cdot \sum_{x \in \boldsymbol{M}} (S(\boldsymbol{x} \cdot \boldsymbol{w}) - y_x) \cdot \frac{\partial}{\partial w_i} S(\boldsymbol{x} \cdot \boldsymbol{w}) \\ &= 2 \cdot \sum_{x \in \boldsymbol{M}} (S(\boldsymbol{x} \cdot \boldsymbol{w}) - y_x) \cdot \frac{\partial}{\partial w_i} S\left(\sum_j x_j \cdot w_j \right) \\ &= 2 \cdot \sum_{x \in \boldsymbol{M}} (S(\boldsymbol{x} \cdot \boldsymbol{w}) - y_x) \cdot S(\boldsymbol{x} \cdot \boldsymbol{w}) \cdot (1 - S(\boldsymbol{x} \cdot \boldsymbol{w})) \cdot \frac{\partial}{\partial w_i} \sum_j x_j \cdot w_j \\ &= 2 \cdot \sum_{x \in \boldsymbol{M}} (S(\boldsymbol{x} \cdot \boldsymbol{w}) - y_x) \cdot S(\boldsymbol{x} \cdot \boldsymbol{w}) \cdot (1 - S(\boldsymbol{x} \cdot \boldsymbol{w})) \cdot x_i. \end{aligned} \tag{2.3.13}$$

After computing the partial derivative for each $w_i \in \boldsymbol{w}$ the *gradient* then is given by:

$$\nabla SSE(\boldsymbol{M}, \boldsymbol{w}) = \left(\frac{\partial SSE(\boldsymbol{M}, \boldsymbol{w})}{\partial w_1}, \ldots, \frac{\partial SSE(\boldsymbol{M}, \boldsymbol{w})}{\partial w_n} \right). \tag{2.3.14}$$

Finally, the current weights $\boldsymbol{w_t}$ can be updated by changing the weights by a certain *step size or learning rate* γ towards a local minimum of the error function by applying

$$\boldsymbol{w_{t+1}} = \boldsymbol{w_t} - \gamma \, \nabla \mathrm{SSE}(\boldsymbol{M}, \boldsymbol{w}). \tag{2.3.15}$$

Since the gradient points toward the steepest ascent of the error function a negation is needed to approach the local minimum.

2.3.1.4 *Multi-Layer Perceptron*

The assumption was made that the two sets of points P and N are linearly separable. Many problems, however, are more complex and cannot be easily separated linearly. One example from the realm of logical functions is the **XOR** function. As an example, consider **XOR** with two input variables. If visualized in the same way as in (**Fig.** 2.3.3) it is quite trivial to see that no single line is able to divide the positive from the negative points. A more complex model is necessary for modeling functions with *convex solution spaces* such as **XOR** (**Fig.** 2.3.5). Similar to 2.3.1.1, Perceptrons can be arranged into layers to compute more complex functions. Such a network is also known as a *multi-layer perceptron (MLP)*.

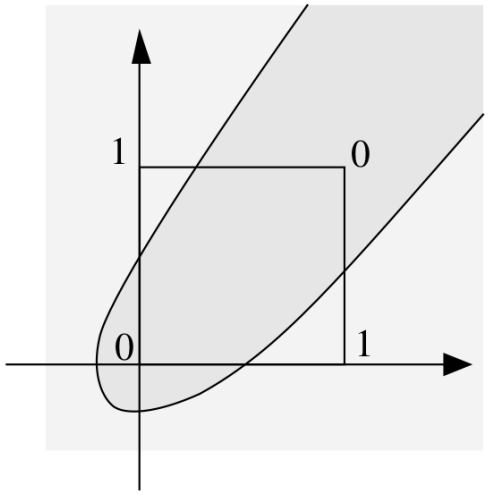


Figure 2.3.5: Solving the **XOR** problem by using separating *regions* instead of hyperplanes. Image taken from [Roj96]

By constructing an MLP with one input layer, one hidden layer as well as one output layer, the network is capable of modeling every convex solution space [Roj96]. The Perceptrons in the hidden layer each learn to linearly separate the solution space into two parts like described earlier. By using just one Perceptron in the second layer, which is able to learn the **boolean AND** function over all outputs from the previous layer it is possible to learn any convex solution space. A visualization is provided in (**Fig.** 2.3.6).

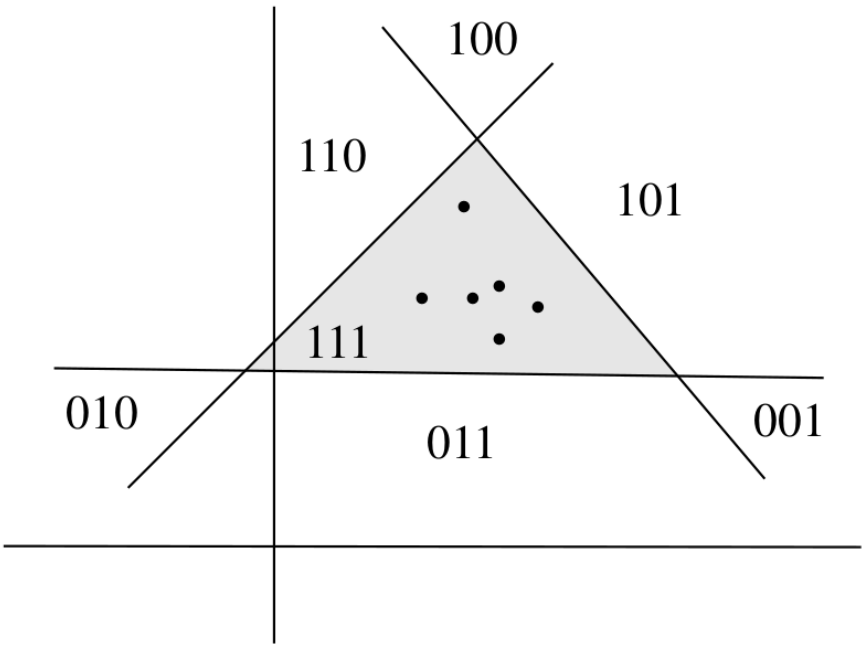


Figure 2.3.6: Example of a convex solution space using a two-layered MLP with three Perceptrons in the first and one Perceptron in the second layer. Each bit vector $b = (x_1, x_{2,3})$ shows the output of Perceptron 1, 2 and 3 respectively. Image taken from [Roj96]

Even with the ability to model any convex solution space, there are still problems which have a *non-convex solution space*. However, by adding another hidden layer to the MLP these problems can also be solved.

The first hidden layer behaves just like in the previous example with each Perceptron splitting the input space into two. Each Perceptron in the second layer again learns **boolean AND** functions, resulting in as many convex regions as there are neurons in the layer. In the third layer, a Perceptron then combines these regions into non-convex

regions using the learned **boolean OR** operator. In fact, a three-layer MLP is able to model any arbitrary function (given enough Perceptrons per layer) [Roj96].

2.3.1.5 *Backpropagation*

A single Perceptron can be trained using gradient descent. A network of many Perceptrons, however, a different algorithm is needed.

Probably the most common learning algorithm for training MLPs is the **backpropagation algorithm**, initially proposed by [Wer74] and popularized by [RHW86]. It uses gradient descent with the addition of propagating the error backwards through the network by making use of the chain rule of derivation.

For each example $\boldsymbol{x_i}$ in the training data, the algorithm follows these steps:

1. Start with random weights $\boldsymbol{W}_0 = (\boldsymbol{w_0}, \ldots, \boldsymbol{w_n})$ for each neuron.
2. Feed $\boldsymbol{x_i}$ into the network (called a *forward pass*)
   - $\hat{f}(\boldsymbol{x_i}, \boldsymbol{w}_0) = \boldsymbol{y_i}$
3. Compute gradient at the last layer by using a loss function and the expected result $\hat{\boldsymbol{y}}_i$.
4. Compute gradient for the previous layers, incorporating the error from the consecutive layers (called a *backward pass*).
5. Update the weights with their corresponding gradients.

In order to show how the Backpropagation algorithm works consider a MLP with one hidden layer L_1, one output layer O_1 and n inputs. Also, let $\|L_1\| = \|O_1\| = M$ without loss of generality.

First, the forward pass is performed. The output of each neuron $h_j(\boldsymbol{x}, \boldsymbol{w_j}) = y_j$ in the hidden layer can be described using the following equation:

$$\begin{aligned} h_j(\boldsymbol{x}, \boldsymbol{w_j}) &= S(\boldsymbol{x} \cdot \boldsymbol{w_j}) \\ &= \frac{1}{1 + e^{-(\boldsymbol{x} \cdot \boldsymbol{w_j})}}. \end{aligned} \tag{2.3.16}$$

Similarly, after the output layer, the output g_k for each neuron given the network input can be described as:

$$\begin{aligned} g_k(\boldsymbol{x}, \boldsymbol{w_k}) &= S\left(\sum_{j=0}^{M} w_{jk} \cdot h_j(\boldsymbol{x}, \boldsymbol{w_j})\right) \\ &= S\left(\sum_{j=0}^{M} w_{jk} \cdot S\left(\sum_{i=0}^{N} w_{ij} \cdot x_i\right)\right). \end{aligned} \tag{2.3.17}$$

Every differentiable loss function can be used in the Backpropagation algorithm. For consistency, consider the *sum of squared error* function defined in (**Eq.** 2.3.7). After the forward pass through the network the loss for a single output neuron g_k is given by:

$$\begin{aligned} \mathrm{SSE}(\boldsymbol{x}, (\boldsymbol{w_j}, \boldsymbol{w_k})) &= (y_k - \hat{y}_k)^2 \\ &= (g_k(\boldsymbol{x}, \boldsymbol{w_k}) - \hat{y}_k)^2 \\ &= \left(S\left(\sum_{j=0}^{M} w_{jk} \cdot h_j(\boldsymbol{x}, \boldsymbol{w_j})\right) - \hat{y}_k\right)^2 \\ &= \left(S\left(\sum_{j=0}^{M} w_{jk} \cdot S\left(\sum_{i=0}^{N} w_{ij} \cdot x_i\right)\right) - \hat{y}_k\right)^2 \end{aligned} \tag{2.3.18}$$

By using the same approach as with gradient descent, the partial derivative of the loss function with regards to the weights of the output layer is given by:

$$\frac{\partial\, \mathrm{SSE}(\boldsymbol{x}, \boldsymbol{w})}{\partial w_{jk}} = \frac{\partial}{\partial w_{jk}} (g_k(\boldsymbol{x}, \boldsymbol{w_k}) - \hat{y}_k)^2. \tag{2.3.19}$$

Using the chain rule for derivation

$$(p(b(x)))' = p'(b(x)) \cdot b'(x), \tag{2.3.20}$$

the partial derivative can be broken down :

$$\begin{aligned}
\frac{\partial\ \mathrm{SSE}(\boldsymbol{x},\boldsymbol{w})}{\partial w_{jk}} &= \frac{\partial}{\partial w_{jk}}\ (g_k(\boldsymbol{x},\boldsymbol{w_k}) - \hat{y}_k)^2 \\
&= 2\cdot(g_k(\boldsymbol{x},\boldsymbol{w_k}) - \hat{y}_k)\cdot S'\left(\sum_{j=0}^{M} w_{jk}\cdot h_j(\boldsymbol{x},\boldsymbol{w_j})\right)\cdot h_j(\boldsymbol{x},\boldsymbol{w}_j) \\
&= 2\cdot(y_k - \hat{y}_k)\cdot y_k\cdot(1-y_k)\cdot h_j(\boldsymbol{x},\boldsymbol{w_j}) \\
&= \delta_k\cdot h_j(\boldsymbol{x},\boldsymbol{w_j}) \\
&= \delta_k\cdot y_j.
\end{aligned} \tag{2.3.21}$$

One can observe that the output of all neurons from layer N_k are needed to compute the partial derivate for neurons in layer N_{k+1}, which is why the initial forward pass through is necessary.

Similarly, the partial derivative of the loss function with regards to w_{ij} is given by (**Eq.** 2.3.22). However, while running Backpropagation, the partial derivatives of all nodes in layer N_{k+1} need to be computed in order to determine the partial derivative of nodes in layer N_k [Roj96].

$$\begin{aligned}
\frac{\partial\ \mathrm{SSE}(\boldsymbol{x},\boldsymbol{w})}{\partial w_{ij}} &= \frac{\partial}{\partial w_{ij}}\ (g_k(\boldsymbol{x},\boldsymbol{w_k}) - \hat{y}_k)^2 \\
&= 2\sum_{k=0}^{K}(g_k(\boldsymbol{x},\boldsymbol{w_k}) - \hat{y}_k)\cdot S'\left(\sum_{j=0}^{M} w_{jk}\cdot h_j(\boldsymbol{x},\boldsymbol{w_j})\right)\cdot w_{jk}\cdot S'(\boldsymbol{x}\cdot\boldsymbol{w_j})\cdot x_i \\
&= 2\sum_{k=0}^{K}(g_k(\boldsymbol{x},\boldsymbol{w_k}) - \hat{y}_k)\cdot g_k(\boldsymbol{x},\boldsymbol{w_k})\cdot(1-g_k(\boldsymbol{x},\boldsymbol{w_k}))\cdot w_{jk}\cdot h_j(\boldsymbol{x},\boldsymbol{w_j})\cdot(1-h_j(\boldsymbol{x},\boldsymbol{w_j}))\cdot x_i \\
&= 2\sum_{k=0}^{K}(y_k - \hat{y}_k)\cdot y_k\cdot(1-y_k)\cdot w_{jk}\cdot y_j\cdot(1-y_j)\cdot x_i \\
&= x_i\cdot y_j\cdot(1-y_j)\cdot\sum_{k=0}^{K} 2\cdot(y_k-\hat{y}_k)\cdot y_k\cdot(1-y_k)\cdot w_{jk} \\
&= x_i\cdot y_j\cdot(1-y_j)\cdot\sum_{k=0}^{K}\delta_k\cdot w_{jk} \\
&= x_i\cdot\delta_j
\end{aligned}$$

(2.3.22)

Generally, the error signal δ can be computed for all layer L_g in the network using:

$$\delta_j = \begin{cases} y_j \cdot (1 - y_j) \cdot (y_j - \hat{y}_j) & \text{if } L_g \text{ is output layer} \\ y_j \cdot (1 - y_j) \cdot \sum_{k \in L_{g+1}} \delta_k \cdot w_{jk} & \text{else.} \end{cases} \tag{2.3.23}$$

After computing the error signal, the weight can be updated, similarly to (**Eq.** 2.3.15), using the following formula:

$$w_{jk_{(t+1)}} = w_{jk_{(t)}} - \gamma \cdot \delta_k \cdot y_j. \tag{2.3.24}$$

One problem that held the development of deep networks with many hidden layers back is the **vanishing gradient** problem [Hoc91][Hoc98]. As seen previously, part of the gradient that gets propagated back through the network to update the parameters is the derivative of the activation function. When choosing Sigmoid as the activation function, the following was shown by [Hoc98] for its derivative:

$$\frac{dS(x)}{dx} \leqslant 0.25 \tag{2.3.25}$$

To compute the gradients of the weights in the first layer, all gradients of all following layer need to be known. When considering (**Eq.** 2.3.23) for a layer which is not the output layer, it becomes clear that the gradient of the next layers are mulitplied together to compute the error signal, and thus gradient, of the current layer. Since the gradient is limited by the derivative of the Sigmoid function, the gradient decreases exponentially with the number of layers. In the literature, this is then referred to as the gradient *vanishing*, which means that the gradients towards the beginning of the network approach zero. This then means that the weights are not changed enough to further learn these layers.

One way of dealing with this problem is using the **Rectified Linear Unit (ReLU)** non-linearity as activation function, as proposed in [NH10]:

$$\text{ReLU}(x) = \max(0, x) \tag{2.3.26}$$

When computing the derivative of **ReLU**, it becomes clear that the vanishing gradient problem cannot occur because the gradient is either propagated or not as opposed to being diminished with every layer:

$$\frac{\mathrm{dReLU}(x)}{\mathrm{d}x} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x < 0 \end{cases} \tag{2.3.27}$$

2.3.2 *Convolutional Neural Networks*

Until now, the only kind of neural network layer discussed consisted of neurons, which take all inputs, compute the weighted sum using a weight matrix and then computing the activation. For certain input data, like images, this approach is not efficient, because each pixel intensity value would need a dedicated weight value, resulting in a large weight matrix. Instead, *convolutional layers* were proposed, which drastically reduce the amount of weight values necessary to process image data.

2.3.2.1 *Convolutional Layer*

In computer vision, image features like edges are traditionally computed by first designing two-dimensional matrices K called *kernels* or *filters*. Let W, H be the width and height of the image I. For each $(i, j) \in W \times H$, the *excitation* $E(i, j)$ is computed using the following formula:

$$E(i, j) = \sum_m \sum_n I(i + m, j + n) \cdot K(m, n). \tag{2.3.28}$$

A visualization of this process can be seen in (**Fig.** 2.3.8). This process relates to the mathematical concept of *convolution*, where the amount of overlap is computed when sliding one function over the other. Specifically, computing the excitation is a discretized, two-dimensional version of the general convolution. In its most general form, a convolution over functions x and w is defined in [GBC16] as:

$$s(t) = (x * w)(t) = \int_{a=-\infty}^{\infty} x(a) w(t - a) \mathrm{d}a. \tag{2.3.29}$$

The excitation values then form the *output feature map*.

One approach for detecting edges was introduced in [SF68], where two 3×3 kernels were introduced for detecting horizontal (G_y) and vertical (G_x) edges in an image (**Eq.** 2.3.30). In the case of the Sobel operator, the output feature map is a horizontal or vertical edge image. See (**Fig.** 2.3.7) for a visualization.

Figure 2.3.7: Output feature maps after applying G_x and G_y, in comparison to the original image (left).

$$G_x = \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix}, \qquad G_y = \begin{pmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} \tag{2.3.30}$$

Convolutional layers generalize the approach of convolving input feature maps with kernels by using kernels whose parameters are learned through backpropagation. This means that the kernels do not need to be designed manually, because the network can learn which kernels are needed, depending on the task. Instead of computing the excitation for each (i, j) position, a *stride* can be defined. A stride of x indicates that the positions to apply the kernel are incremented by x, instead of 1.

In practice, each convolutional layer consists of multiple kernels. This results in a output feature map with a depth corresponding to the number of different kernels. The output feature map can then be processed by another convolutional layer. A neural network with at least one convolutional layer is also often referred to as a **Convolutional Neural Network** or a **CNN**.

Since the kernel is centered at each coordinate (i, j) of the input feature map, a decision has to be made when not all input values are present, e.g., at the edges of an input image. Generally, there are two approaches used in the literature. First, (i, j) positions where not all input feature map values are available can be ignored, resulting

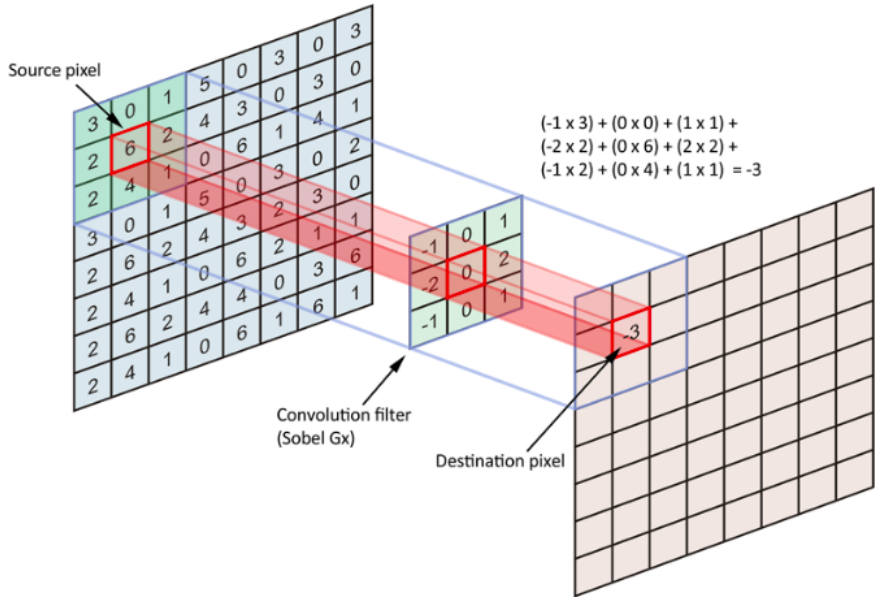


Figure 2.3.8: Visualisation of using a Sobel filter for detecting vertical edges. Image taken from [Der17b].

in a output feature map with smaller spatial dimensions since the convolution was not computed for every (i, j) position. Second, the missing values can be substituted. This is referred to as **padding**. Common padding approaches are to fill the missing values with 0 or duplicating with the nearest input feature map values.

2.3.2.2 *Pooling Layer*

Another important layer type when using Convolutional Neural Networks is the *pooling layer*. Pooling layers are used to reduce the size of the activation map and introduce invariance to small translations when applied to the spatial dimensions [GBC16]. There are different types of pooling layer, however the most popular is the **Max-pooling** layer where, for each $n \times n$ pixel block, the maximum value is used as the output. Consider the output feature map of the previously discussed Sobel filter, where high values indicate the presence of a horizontal or vertical edge, depending on the kernel. In a similar sliding-window approach to the convolutional layer, the Max-pooling layer computes the maximum values for $n \times m$ blocks at each pixel position. Additionally, a stride s can be defined as well and, in practice, the stride is often set to $s = n$. This results in a reduction of the spatial dimension of the

feature map by a factor n in both dimensions. Also, invariance to small translations is introduced because the maximum value is propagated, regardless of where in the $n \times n$ window it occured. See (**Fig.** 2.3.9) for a visualization.

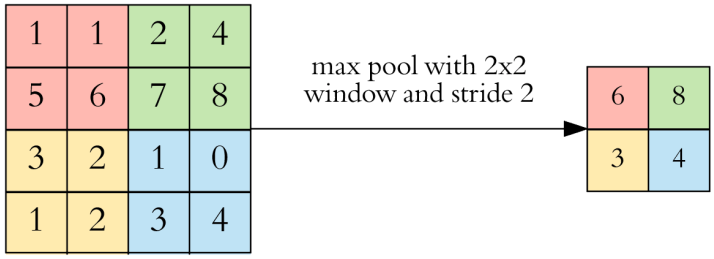


Figure 2.3.9: Example of a 2×2 Max-pooling operation with a stride of 2. Image taken from [Cor18].

2.3.2.3 *Fully-Connected Layer*

As discussed previously, in convolutional neural networks, convolution and pooling layers are used to extract visual features from the input. The features can then be used for classifying or regression. For this purpose, layers of neurons, as introduced earlier, can be added to a convolutional neural network. In the context of CNNs, these layers are referred to as **fully-connected** layers. An example of a CNN where features are computed and then classified can be seen in (**Fig.** 2.3.10).

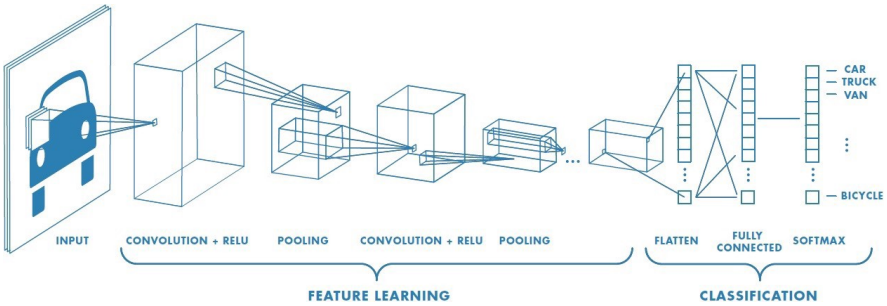


Figure 2.3.10: Example of a Convolutional Neural network, consisting of a feature extractor, followed by a fully connected network, which uses the Softmax activation function for classification. Image taken from [Sah18].

3 RELATED WORK

In the previous chapter, the fundamentals of Human Action Recognition, Pose estimation, as well as Artificial and Convolutional Neural networks were introduced. This chapter discusses current methods for both HAR and Pose estimation found in the literature. Current and historical approaches to estimating pose are presented in Section 3.1. These include shallow approaches based on the *Pictoral Structure Framework*, as well as deep learning based approaches. In Section 3.2, both shallow methods as well as methods using deep learning are discussed for HAR on video datasets. This thesis focuses on methods incorporating pose information in the Human Activity Recognition decision process as features, since this is similar to the approach presented in [LPT18].

3.1 POSE ESTIMATION

While there are many different approaches for pose estimation, one of the most widely used in the literature was the *Pictoral Structure Framework*. In the following section, the foundation of this framework will be explained, along with extensions to the model from . Afterwards, deep learning based methods became dominant in the field of pose estimation, which is why they will be explained in the subsequent section.

3.1.1 *Pictoral Structure Framework*

In 1973, [FE73] presented a general framework for object recognition problems, including pose estimation. The so called *Pictoral Structure Framework* is comprised of parts, that are connected by *spring-like connections*. In the context of pose estimation, they used the term *part* to be synonymous with *limb*. A part is defined by a collection of parameters l_i which could include center coordinates and rotation. The connection between two parts is modelled using a mechanism inspired by physical springs. Such connections can be relaxed or be under tension. This expresses how realistic a certain connection is, based on the amount of tension required to connect two parts.

The graph that is created from connected parts models the desired object, for example the pose or facial features of a human, and it is also referred to as a *deformable*

structure. In the context of pose estimation, limbs are modelled as parts, connected together at the joints.

The output of the alorithm is expressed as $L = (l_0, l_1, \ldots, l_n)$, called the *configuration* of all parts l_i. To compute the optimal configuration, the authors proposed the following minimization problem:

$$L^* = \arg\min_L \left(\sum_{i=0}^{n} m_i(l_i) + \sum_{(v_i,v_j)\in E} d_{ij}(l_i, l_j) \right). \tag{3.1.1}$$

In (**Eq.** 3.1.1), m_i is a function evaluating the placement of part i at configuration l_i. This is also often referred to as the *unary* term. d_{ij} measures the mismatch when parts i and j, which are connected, are placed according to l_i and l_j respectively.

Minimizing the energy function is computationally expensive since the space of possible positions for each part spans the entire image. According to [FH05], there are methods using heuristics to make the computation more efficient, but they do not find optimal solutions. They proposed to transform the problem into a statstical framework and solve it by estimating a posterior distribution.

One advantage of this statistical model is that the parameters can then be estimated using training examples, similar to the principles of learning presented in (**Sec.** 2.3). Also, it is possible to get multiple solutions to the minimization problem by sampling the posterior distribution as opposed to a single solution with the energy minimization approach.

The authors model the problem as follows. Let $p(L \mid I, \theta)$ be the desired posterior distribution, where θ is a set of model parameters and I is the image. When applying Bayes' formula, one can express the posterior as:

$$p(L \mid I, \theta) \propto p(I \mid L, \theta) p(L \mid \theta). \tag{3.1.2}$$

The likelihood $p(I \mid L, \theta)$ is approximated as $\prod_{i=1}^{n} p(I \mid l_i, \theta_i)$. This approximation, however, is bad if recognized parts occlude each other [FH05]. This problem is often referred to as *double counting* and it is one of the most prominent error of Pictoral Structure based models. The authors tackle this problem by sampling multiple times from the posterior, and by evaluating them using a separate measurement.

To approximate the prior distribution $p(L \mid \theta)$, the authors utilize the joint distribution of a tree structured Markov-random field. This tree is also referred to as the *kinematic pose prior*, since it encodes prior information of how limbs should be

connected together. The vertices are modeled by l_i and the edges show connections among the different parts. Since the denominator incorporated the absolute position for each part, and the authors are only interested in modeling the relative position between parts, they argue that the denominator can be constant. Thus, for simplicity, they set the denominator to one:

$$\begin{aligned} p(L \mid \theta) &\propto \frac{\prod_{(i,j)\in E} p(l_i, l_j \mid \theta)}{\prod_{i\in V} p(l_i \mid \theta)^{\deg v_i - 1}} \\ &\propto \prod_{(i,j)\in E} p(l_i, l_j \mid \theta). \end{aligned} \tag{3.1.3}$$

This then leads to the following approximation of the posterior distribution:

$$\begin{aligned} p(L \mid I, \theta) &\propto p(I \mid L, \theta) p(L \mid \theta) \\ &\propto \prod_{i=1}^{n} p(I \mid l_i, \theta_i) \prod_{(i,j)\in E} p(l_i, l_j \mid \theta). \end{aligned} \tag{3.1.4}$$

To show the equivalence to the original energy minimization problem, the authors take the formula presented in (**Eq.** 3.1.4), and they apply the logarithmic function. Then, they negate the result. This then leads to the following formulation:

$$\begin{aligned} &-\log \left(\prod_{i=1}^{n} p(I \mid l_i, \theta_i) \prod_{(i,j)\in E} p(l_i, l_j \mid \theta) \right) \\ &= -\log \left(\prod_{i=1}^{n} p(I \mid l_i, \theta_i) \right) - \log \left(\prod_{(i,j)\in E} p(l_i, l_j \mid \theta) \right) \\ &= \sum_{i=1}^{n} -\log p(I \mid l_i, \theta_i) + \sum_{(i,j)\in E} -\log p(l_i, l_j \mid \theta). \end{aligned} \tag{3.1.5}$$

When comparing (**Eq.** 3.1.5) to (**Eq.** 3.1.1) it is easy to see that $m_i(l_i) = -\log p(I \mid l_i, \theta_i)$ and $d_{ij}(l_i, l_j) = p(l_i, l_j \mid \theta)$.

After presenting the statistical framework for general object recognition tasks, the authors explain their approach to pose estimation using this framework. First, they specify that the input image needs to be a binary image where the person is separated from the background. Then, the objective is to maximize the number of foreground

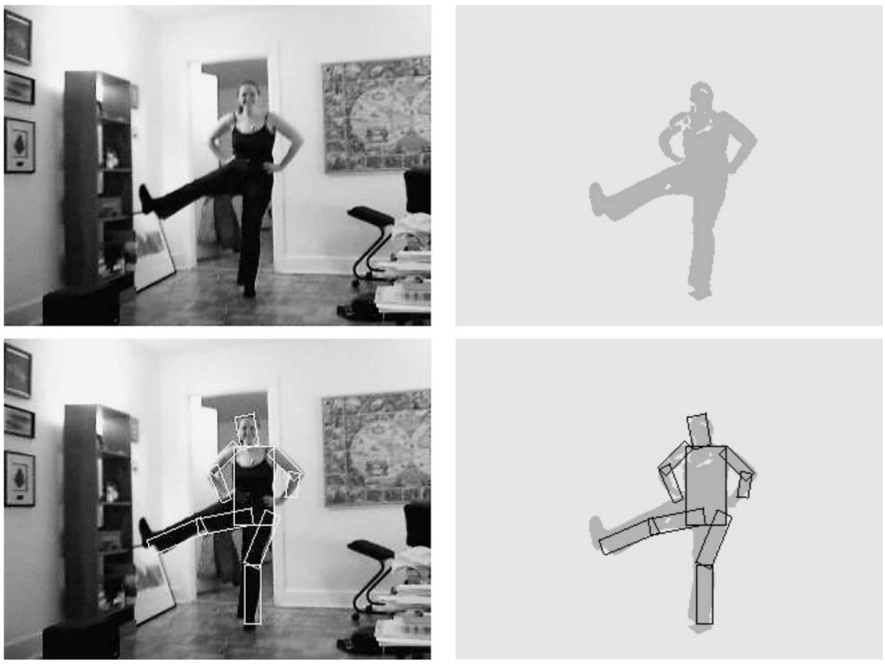


Figure 3.1.1: Overview of pose matching algorithm. **Upper left**: Original image. **Upper right**: binary image where foreground pixels (containing the human figure) are separated from the background. **Lower right**: One result from matching the limb rectangles to maximize the amount of foreground pixels covered. **Lower left**: Final result with estimated pose layered on top of original image. Image taken from [FH05].

pixels covered by the detected parts in their calculated configuration. A visualization of this process can be seen in (**Fig.** 3.1.1).

The authors model the human pose as a collection of 10 different parts. Two parts per arm and leg as well as one torso and one head part (**Fig.** 3.1.1). Each part configuration $l_i = (x_i, y_i, s_i, \varphi_i)$ contains the x and y coordinate of the center of the rectangle representing the part. In addition, $s_i \in [0, 1]$ defines the length of the rectangle and φ_i its rotation. The width is fixed.

To model $p(I \mid l_i, \theta_i)$, they use the formula in (**Eq.** 3.1.6), which utilizes $\theta_i = (q_1, q_2)$. q_1 is the probability of pixel inside l_i being a foreground pixel whereas q_2 is the probability of pixels closely around l_i being foreground pixels. The area around l_1 is referred to as $area_2$ whereas the area of l_i is referred to as $area_1$. $count_i$ referres to

the number of foreground pixels in $area_i$ and t is the total number of pixels in the image.

$$\begin{aligned} p(I \mid l_i, (q_1, q_2)) = & q_1^{count_1} * (1 - q_1)^{(area_1 - count_1)} \\ & * q_2^{count_2} * (1 - q_2)^{(area_2 - count_2)} \\ & * 0.5^{(t - area_1 - area_2)} \end{aligned} \tag{3.1.6}$$

Also, they smooth the distribution to prevent peaks by using the principle of annealing with a constant factor T (**Eq.** 3.1.7). This is important because a distribution with strong peaks is more likely to return results around these pearks when sampled from. As discussed earlier, sampling is needed because of the approximation of the unary term the authors used and its problem with overlapping limbs. A distribution with strong peaks would not allow for sufficiently diverse pose samples.

θ_i is then estimated using the mean values from annotated training data points, and the authors set T to to 10 for smooting the peaks:

$$p(I \mid l_i, \theta_i) \propto p(I \mid l_i, \theta_i)^{\frac{1}{T}} \tag{3.1.7}$$

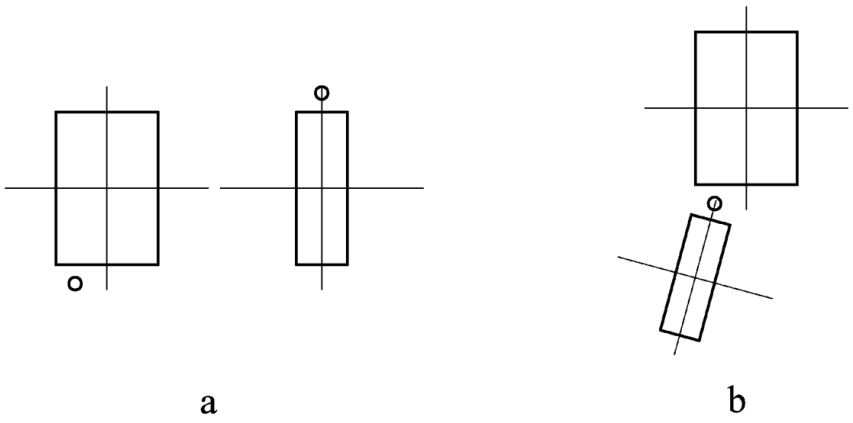


Figure 3.1.2: Visualization of connecting limbs. **a**: Two limbs in their own coordinate system. The circles indicate connection joints. **b**: Possible, anatomically plausible connection. Image taken from [FH05].

Next, the authors model the prior $p(L \mid \theta)$. Each spatial relation $p(l_i, l_j \mid \theta)$ between two parts is modelled in the following way: First, each part is placed inside its

own coordinate system with the origin being the center point of the part (see (**Fig.** 3.1.2)a). The connection point (joint) between parts i and j is defined as (x_{ij}, y_{ij}) in the coordinate system of part i and as (x_{ji}, y_{ji}) in the coordinate system of part j. When transformed to the image coordinate system, these points should be as close together as possible (see (**Fig.** 3.1.2)b). The authors express this using Gaussian distributions:

$$\begin{aligned} &N((\hat{x}_i - \hat{x}_j), 0, \sigma_x^2) \\ &N((\hat{y}_i - \hat{y}_j), 0, \sigma_y^2), \end{aligned} \tag{3.1.8}$$

where $\hat{x}_i$ and $\hat{y}_i$ are transformations of the joint position into the image coordinate system:

$$\begin{bmatrix} \hat{x}_i \\ \hat{y}_i \end{bmatrix} = \begin{bmatrix} x_i \\ y_i \end{bmatrix} + s_i R_{\varphi_i} \begin{bmatrix} x_{ij} \\ y_{ij} \end{bmatrix}. \tag{3.1.9}$$

$\hat{x}_j$ and $\hat{y}_j$ are computed analogously. R_{φ_i} is a matrix which rotates around the origin for φ_i radiants. Additionaly, the authors define that the difference between s_i and s_j should be close to zero. Again, this is modelled using a Gaussian distribution:

$$N((s_i - s_j), 0, \sigma_s^2). \tag{3.1.10}$$

Lastly, the authors specify that the difference between the two relative angles φ_i and φ_j be close to a parameter φ_{ij}. They use a von Mises distribution, given by

$$M(\theta, \mu, k) \propto \exp(k \cdot \cos(\theta - \mu)). \tag{3.1.11}$$

A von Mises distribution, also referred to as a circular normal distribution, can be thought of as a normal distribution around a circle. This means that the support of the distribution is $x \in [-\pi, \pi]$ which is why the authors use it for periodic input like angles. Using $\mu = \varphi_{ij}$, they specify the constrained described earlier. The parameter k determines how strong the peak at φ_{ij} should be or, in other words, how constrained the joints should be.

In summary, each spatial relation is expressed in the following way:

$$\begin{aligned} p(l_i, l_j \mid \theta) &= p(l_i, l_j \mid c_{ij}) \\ &= N((\hat{x}_i - \hat{x}_j), 0, \sigma_x^2) \\ &* N((\hat{y}_i - \hat{y}_j), 0, \sigma_y^2) \\ &* N((s_i - s_j), 0, \sigma_s^2) \\ &* M((\varphi_i - \varphi_j), \varphi_{ij}, k). \end{aligned} \tag{3.1.12}$$

The prior parameters c_{ij} are then given by $c_{ij} = (x_{ij}, x_{ji}, y_{ij}, y_{ji}, \sigma_x, \sigma_y, \sigma_s, \varphi_{ij}, k)$ and they are estimated using maximum the maximum likelihood estimation.

The authors merely provide example images instead of a quantitative analysis. This is most likely because of the lack of benchmark datasets available at the time. They labeled 10 images by hand and used these to estimate the parameters. Then, for each of their unlabeld test images, they sampled 200 poses and calculated the Chamfer distance, which measures the binary correlation. The best pose with regards to the Chamfer distance is then returned. Some example images can be seen in (**Fig.** 3.1.3)

3.1.1.1 *Approaches for parts detection*

Over the years, many approaches build on top of [FH05] by improvig parts of the framework. One approach, where the parts detector was improved, is presented in the following section.

In [ARS09], the authors propose a different appearance model $p(I \mid L, \theta)$, which is based on individual parts detectors. [FH05] used background separation to extract the silhouette of the subject from the background. Then, the objective was to configure the parts in such way that the covered foreground area was maximized (see (**Eq.** 3.1.6)). This resulted in a large search space since all different rotations and scalings for each part need to be placed inside the foreground area to evaluate the overall fit. The authors in [ARS09] limit the search space by using a detected *parts evidence map* d_i for each part i. An example evidence map can be seen in (**Fig.** 3.1.4b). These evidence maps are the result of individual parts detectors trained on annotated images such as presented in (**Fig.** 3.1.4a).

The authors compute dense *shape context descriptors* [BMP02] for each training image. This local descriptor takes gradients of edge pixels into account and discretizes them into bins in a log-polar histogram. The authors chose 12 bins for the location where the gradients are put inside 8 bins, resulting in a 96 dimensional descriptor for each sampled point. Descriptors with a center point inside a part bounding box given by

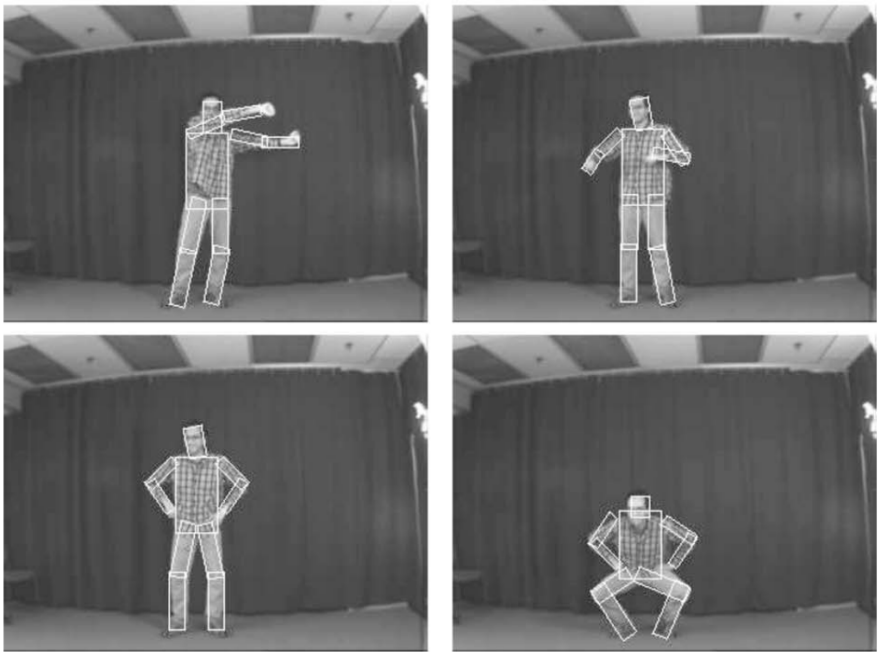


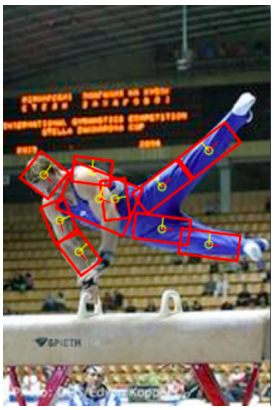
Figure 3.1.3: Example of poses estimated by the statistical framework. Image taken from [FH05].

the annotation are then concatenated and used as positive training examples to train the parts detectors.

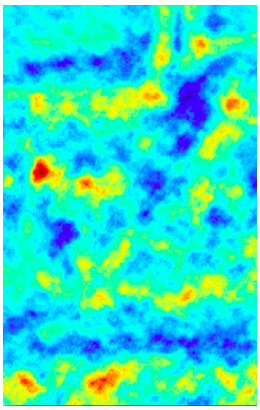
For detectors, the authors use the concept of *boosting* where the decisions of many weak classifiers are combined into a strong classifier [FS99]. Specifically, the authors use the *AdaBoost* algorithm proposed by [FS97]. Each weak detector takes a bin from the feature vector and compares it to a threshold. The sign of the result is then used as the output of the weak detector following

$$h_t(\boldsymbol{x}) = \text{sign}(\xi_t(\boldsymbol{x}_{n(t)} - \varphi_t)), \tag{3.1.13}$$

where $\boldsymbol{x}$ is the feature vector, $\xi_t \in \{-1, 1\}$ and $n(t)$ is the index of the bin, which is used to compare to φ_t. The authors do not, however, specify how to set ξ_t or which threshold φ_t was used.



(a) Parts-annotated example image.



(b) Part evidence map for part *head*. Notice the peak (in red) where the head is in the left image.

Figure 3.1.4: Images taken from corresponding talk to [ARS09].

Combining t weak detectors by summing over the weighted outputs, the complete part detector for part i is given by

$$H_i(\boldsymbol{x}) = \text{sign}\left(\sum_t a_{i,t} h_t(\boldsymbol{x})\right) \tag{3.1.14}$$

According to the authors, $a_{i,t}$ are weights learned by each weak detector. To fit the parts detectors into the Pictoral Structure Framework, the authors compute a pseudo-probability based on the part detectors:

$$p(d_i \mid l_i) = \max\left(\frac{\sum_t a_{i,t} h_t(\boldsymbol{x}(l_i))}{\sum_t a_{i,j}}, \epsilon_0\right). \tag{3.1.15}$$

$\boldsymbol{x}(l_i)$ is the feature vector for part l_i whereas ϵ_0 is set to 10^{-4}. The authors claim that this pseudo-probability achieves good results in practice.

Lastly, the authors make two assumptions to embed the parts detectors into the Pictoral Structute Framework. First, different part evidence maps d_i are independent

from each other given L. Second, d_i only depends on l_i on not on other parts configurations. This leads to the following likelihood

$$p(I \mid L, \theta) = p(D \mid L) = \prod_{i=0}^{N} p(d_i \mid l_i), \tag{3.1.16}$$

where N is the number of parts to detect in the given image.

The authors use two datasets to evaluate their model. The *Buffy* dataset provides annotated upper-body poses from 3 episodes of the TV series *Buffy The Vampire Slayer* [FMJZ08]. Also, for evaluating full-body poses, they used the *Iterative Image Parsing* dataset provided by [Ram07]

As an evaluation metric, the authors use the *Percentage of Correct Parts (PCP)*, proposed by [FMJZ08]. For each annotated body part, PCP is calculated by first computing the length of the ground truth body part l_{gt}. Then, a threshold value $t_{gt} = 0.5 \cdot l_{gt}$ is defined as half the length of the segment. If both endpoints of a prediction fall within t_{gt} radius of their ground truth annotation the part is considered detected.

First, they compare their upper-body model to [FMJZ08]. In contrast to [FMJZ08], however, the authors also compute PCP for each part, whereas [FMJZ08] only provide the overall accuracy of 57.9%. Using the pose dataset of [Ram07] for training the pose priors and part detectors, the authors achieve an overall accuracy of 71.3%. They notice that, while torso, head and upper arm accuracy is fairly high (from 78% for upper arm to 95.9% for head), forearm accuracy (40%) was significantly lower. They argue that is is due to the challenging nature of the dataset. Also, forearm parts are smaller, which means it is more difficult for these parts to be considered correct. The authors also tried exclusively using frames from the Buffy dataset, which were not used for testing, to learn the pose priors. This lead to a slight increase in accuracy (73.5%) in comparison to generic priors, mainly by increasing the forearm accuracy scores. This illustrates that generic pose priors can achieve comparable results to domain-specific priors.

For evaluating full body pose, the authors compared their results to the results presented by [Ram07], the authors of the dataset. Overall, they achieve an overall accuracy of 55.2%, which is significantly higher than the 27.2% achieved by [Ram07]. The authors argue that this large improvement is due to their strong part detector. This is illustrated by the accuracy achieved on the head part. The accuracy of the authors is higher than [Ram07], even when not using the pose prior information and just using the part detector.

3.1.1.2 *Articulated pose estimation with flexible mixture-of-parts*

Another approach building on the Pictoral Structure Framework was proposed by [YR11].

The proposed model differs from the orignal framework by using parts that are not rotated or forshortened. To gain more flexibility, each part i has a type t_i, which can encode arbitrary information. One possibility, which is also what the authors used in their experiments, is to let the types correspond to different orientations. For example, two types for the part *head* could be *upright* or *tilted to the left*. In addition, in this model, it is possible to constraint two adjacent parts' types, as it will be seen later.

First, the authors change the definition of a part i and its parametrization l_i from the one presented in [FH05]. A part is defined as a joint instead of a limb, because the datasets used annotated joint positions as opposed to parts positions. Thus, the parametrization changes to $l_i = (x_i, y_i)$ since no rotation and forshortening occurs.

The Buffy and Iterative Image Parsing datasets, which the authors used for evaluation, do not provide annotations for type labels. Thus, they decided to compute these beforehand using the following steps. First, the authors define a graph where the vertices correspond to joints. The edges of the graph are defined in a way that model a human skeleton. See (**Fig.** 3.1.5) for an example graph.

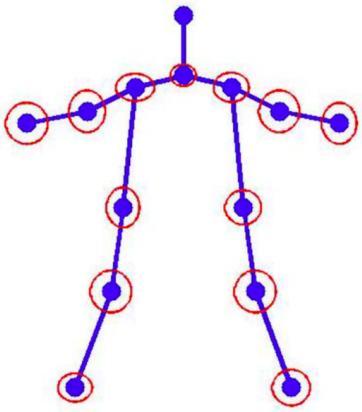


Figure 3.1.5: Visualisation of the graph resulting from connecting the joint positions, which form the vertices, together into a human skeleton. Image taken from [YR11].

Second, the orientation of a part i is considered relative to its parent part j. For instance, consider the relative position of the part corresponding to *neck* to its parent part *head*. Necks tend to be below heads (when considering upright people). The

authors use all relative positions found by iterating the dataset and cluster the relative positions into $T = 4$ clusters. The cluster that the part i is put in then defines the type label t_i. See (**Fig.** 3.1.6) for an example. This results in a fully annotated dataset with location information l_i as well as type t_i for each part i.

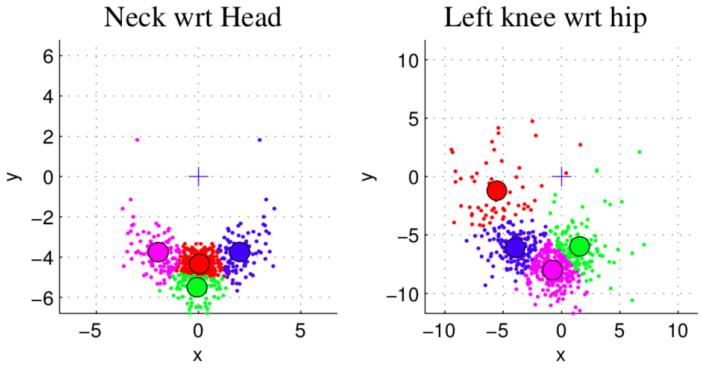


Figure 3.1.6: The relative positions of *neck* with regard to its parent *head* (**left**) and *left knee* with regard to *hip* (**right**). The origin marks the location of the parent. Notice that the neck is generally placed below the head (since most images contain upright people) while the left knee is mostly placed left of the hip. Each color indicates a cluster and a thus a corresponding type label. Image taken from [YR11].

Instead of using one *template* for each limb, the authors propose to use many small templates per limb. Each template can be of a specific type t_i, which can encode semantic features, e.g., open versus closed hand, or orientational information.

The authors provide the following score function to evaluate a predicted pose:

$$\begin{aligned} S(I, l, t) = &\sum_{i \in V} b_i^{t_i} + w_i^{t_i} \cdot \phi(I, l_i) \\ &+ \sum_{(i,j) \in E} b_{ij}^{t_i, t_j} + w_{ij}^{t_i, t_j} \cdot \psi(l_i - l_j) \end{aligned} \tag{3.1.17}$$

In (**Eq.** 3.1.17), the variable I refers to the image that is evaluated. $l = (l_1, \cdots, l_k)$ is the configuration containing all part positions while $t = (t_1, \ldots, t_k)$ corresponds to the types of parts. $\phi(I, l_i)$ is a Histogram of Oriented Gradients (HoG) feature vector [DT05] computed around the part position l_i. $\psi(l_i - l_j) = [(x_i - x_j), (x_i - x_j)^2, (y_i - y_j), (y_i - y_j)^2]^T$ encodes the relative location of part l_i to l_j.

$w_i^{t_i}$, $w_{ij}^{t_i, t_j}$ as well as $b_i^{t_i}$ and $b_{ij}^{t_i, t_j}$ are parameters estimated from annotated training data, which encode learned relationships between parts as well as type assignments

for each part. During training, a supervised approach is used where the score for an annotated training image is maximized while utilizing regularisation on the models parameters $\beta = (\boldsymbol{w}, \boldsymbol{w})$.

To evaluate their model, the authors use *Percentage of Correct Parts (PCP)*, presented in (**Sec.** 3.1.1.1), to be able to compare it to earlier work. In addition, they define a new metric, *Probability of Correct Keypoints (PCK))*. In terms of datasets, they use Buffy and Iterative Image Parsing.

PCK was designed to improve upon two disadvantages of PCP. First, the toolkit provided with the Buffy dataset [FMJZ08] used a definition of PCP different from the one proposed in the paper, which makes the comparison of PCP values in literature hard, because it is unclear which definition was used. The paper definition was provided in (**Sec.** 3.1.1.1). The Buffy toolkit defines a part as detected if the distance of the average point of both endpoints is within the threshold to the average of the ground truth endpoints. Second, foreshortening changes the length of the part, which means that the threshold is dependent upon the amount of foreshortening. This means that foreshortened parts require more accurate predictions to be considered detected. PCK requires each test image to have a person bounding box annotation. Using the bounding box dimensions $(\mathit{width}, \mathit{height})$, a keypoint is considered detected if the distance between the ground truth and prediction is less than $\alpha \cdot \mathit{max}(\mathit{width}, \mathit{height})$. This gives a clear reference point for comparison and makes the metric less susceptible to errors due to foreshortened parts. The paramter α determines how strict the meassurement should be. For the Buffy dataset, the authors set $\alpha = 0.2$, while they chose $\alpha = 0.1$ for the Iterative dataset. They argue that they relaxed the metric for Buffy since the dataset only contains upper-body poses.

As mentioned earlier, the definition of PCP is not clear since the toolkit implementation and paper definition differ. Thus, the authors computed two different PCP values for each dataset. First, they use definition presented in the paper, where the distance of both keypoints need to be smaller than the threshold for a part to be defined as detected. This is referred to as the strict definition. Second, they use the definition provided by the Buffy toolkit, which is more loose since the averages of the keypoints are more likely to be close to each other than both endpoints of the parts.

Even when considering the strictest definition, the model outperforms the previous approaches, including [ARS09] (see (**Sec.** 3.1.1.1)), on the Iterative dataset. The authors achieve an overall PCP score of 61.5%, which is significantly larger than the 55.2% achieved by [ARS09]. Most notably, the detection rate of the legs is about 10 percentage points higher when compared to [ARS09]. Since they also propose a new metric, they provide the PCK metric results as well. Overall, the accuracy achieved is 72.9%, which

is higher than the overall PCP value. The authors argue that this might be due to the fact that foreshortened limbs are not penalized unfairly by PCK.

When evaluating the upper-body Buffy dataset, the authors model outperforms the previous methods, when considering the loose definition, on overall accuracy. However, when considering the strict definition, they outperform [ARS09] with 83.5% (compared to 73.5%, see (**Sec.** 3.1.1.1)), but are themselves outperformed by other competitors. When using PCK, the model achieves an overall accuracy of 72.9%, which is also higher than the corresponding PCP score.

The authors also evaluate the effect of the number of mixtures defined as well as the number of parts to detect. The results can be seen in (**Fig.** 3.1.7). While there is a significant increase in accuracy (in terms of PCK) when increasing the number of parts from 14 to 26, the effect decreases when further increasing the number of parts. When evaluating the number of different types, accuracy increases again when increasing the number of mixtures. They argue that 26 parts and 6 mixtures is a good trade-off when considering both accuracy and runtime.

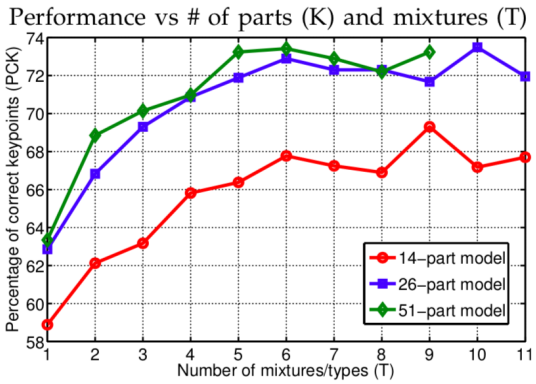


Figure 3.1.7: The authors find that the higher the number of mixtures (T) the higher the accuracy. This is also true, to an extend, with number of parts to detect (K). The authors suggest a 26 part, 6 mixture model for a good trade-off between accuracy and performance. Image taken from [YR11].

3.1.2 *Deep Learning Methods*

In the following Sections, this thesis presents methods for estimating pose using deep learning approaches. In contrast to the previously presented Pictoral Structure Framework, these methods do not learn statistical representations of pose. Rather, they

use CNNs to detect the joints between limbs from image features and either regress the location of the limbs directly or output a *heatmap*, which indicates the likelihood of a certain limb being present for each pixel pair (x, y) in the image. Another term used in the literature for heatmap is *belief map*. In addition, all approaches use the principle of a *stacking architecture*, where initial pose estimations are refined by later stages of the network.

3.1.2.1 *DeepPose*

One of the first approaches for pose estimation using deep convolutional neural networks was proposed by [TS14]. They achieved state-of-the-art results on two benchmark datasets using a neural network architecture based on the ImageNet network [KSH12]. In contrast to the Pictoral Structure framework discussed earlier, the approach was to regress the image coordinates directly without specific prior constrains on how the joints are allowed to be connected.

The authors define their network in terms of multiple connected *stages*. Each *stage* is comprised of the neural network architecture provied in (**Tab.** 3.1.1). The only change the authors made from the architecture originally proposed in [KSH12] was to use the output of the last fully connected layer to regress the joint coordinates as opposed using the softmax activation function. This means that, assuming k joints should be detected, the output of each stage is of size $2k$, because each joint position is given by image coordinates (x, y). Stage s_0 is trained using the full images of size 220x220 as its input.

Once the first stage is trained until convergence, the authors use the output coordinates to crop sub images around the prediction. By feeding these subimages into the next stage s_1 they argue that the predictions of s_1 get more precise, because local context around the prediction becomes more important. The sub image size is dependend upon the diameter of the torso bounding box given by the annotations.

This cascading approach is then repeated with the next stages. In the end, they use 3 stages $S = (s_0, s_1, s_2)$ after evaluating different numbers of stages on a held-out training set.

After training their network, the authors evaluate the final model on two datasets, FLIC [ST13] and LSP [JE10]. Frames Labeled In Cinema (FLIC) is a dataset, which contains 5000 images of Hollywoord movie scenes where 10 upper-body joint positions are labeled for the subject of the scene. In that regard, FLIC is similar to the Buffy dataset. Leeds Sports Pose (LSP) is a dataset of 2000 annotated images of subjects in a sport context, which means that the subjects are highly articulated and thus challenging to detect. LSP contains 14 different joint positions of the entire body.

nr.	layer type	filter size	nr. of filters / neurons	stride
1	convolutional	11 x 11 x 3	96	4
2	local response normalization			
3	pooling	3 x 3		2
4	convolutional	5 x 5 x 48	256	1
5	local response normalization			
6	pooling	3 x 3		2
7	convolutional	3 x 3 x 256	384	1
8	convolutional	3 x 3 x 192	384	1
9	convolutional	3 x 3 x 192	256	1
10	fully connected		4096	
11	fully connected		4096	
12	fully connected		2k	

Table 3.1.1: One *stage* of the DeepPose network. Architecture based on [KSH12].

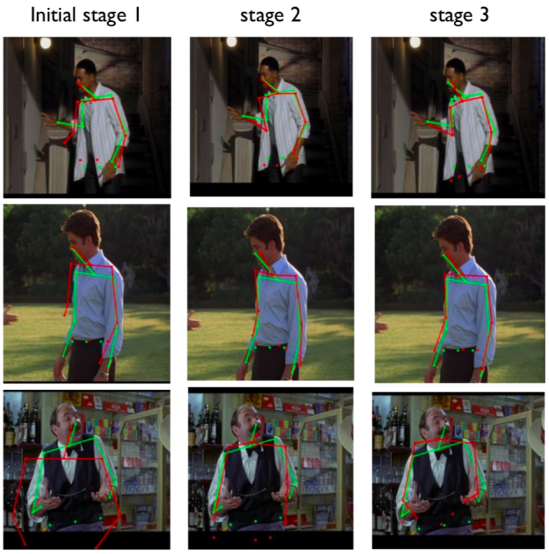


Figure 3.1.8: Qualitative evaluation of the cascading architecture. Ground truth joints are shown in green while predicted poses are shown in red. Notice how the pose gets more accurate with consecutive stages. Also, notice that the biggest improvement can be seen after stage 1. The authors argue that this is due to the fact that, with an increase in stages, the subimage gets smaller and smaller and thus harder to estimate the pose. Image taken from [TS14].

The authors use two different metrics to evaluate their model. First, they use PCP (see (**Sec.** 3.1.1.1)) for comparison with the literature.

Also, they define a new metric, *Percentage of Detected Joints (PDJ)*. They argue, similar to [YR13], that PCP unfairly penalizes shorter limbs. With PDJ, a joint is considered detected correctly if the distance between the predicted joint position and the ground truth d_p is smaller than a fraction of the torso's diameter d_t. This fraction is set to be 0.5, meaning a joint is considered detected if $0 <= d_p <= 0.5 \cdot d_t$. This metric is similar to PCK in that it compares the distance between two joints to a subject-specific value like torso diameter or subject bounding box.

When comparing the results on the LSP and FLIC datasets, the neural network outperforms the state-of-the-art approaches when considering the PCP metric. However, like mentioned before, PCP accuracies are hard to compare since there are multiple definitions available. Thus, the authors reevaluate previous approaches using PDJ and compare their model against these new meassurements. Again, they outperform the previous state-of-the-art models on both LSP and FLIC.

The authors additionally evaluate their model on the Buffy and Iterative Image Parse datasets, presented in (**Sec.** 3.1.1.1). This allows a comparison to [YR13]. Specifically, they trained their model on LSP and evaluated it on Iterative Image Parse for evaluating full-body poses. Analogously, they train their model on FLIC and evaluate it on the Buffy dataset when considering upper-body poses.

After evaluating on the Iterative Image dataset, they do not share the overall PCP values. Instead, they compare their results on upper and lower arms as well as upper and lower legs. These are considered to be the hardest to detect. On this subset, the model outperforms [YR13] on average by 13 percentage points (69% average acccuracy as opposed to 56%).

For the Buffy dataset, they do not publish explicit values, but rather the plot in (**Fig.** 3.1.9). From the graphic, it becomes clear that the author's model also outperforms [YR13] in terms of PDJ.

The authors provide three examples, which illustrate the importance of the cascading architecture. These examples can be seen in (**Fig.** 3.1.8). They argue that stage s_1 has the biggest effect on the accuracy of the joints, since further stages use smaller and smaller subimages for regression and thus use a smaller context. To support their hypothesis, they evaluating three networks in terms of PDJ on the FLIC dataset, with one, two and three stages respectively. Their findings show the same behaviour observed in (**Fig.** 3.1.8).

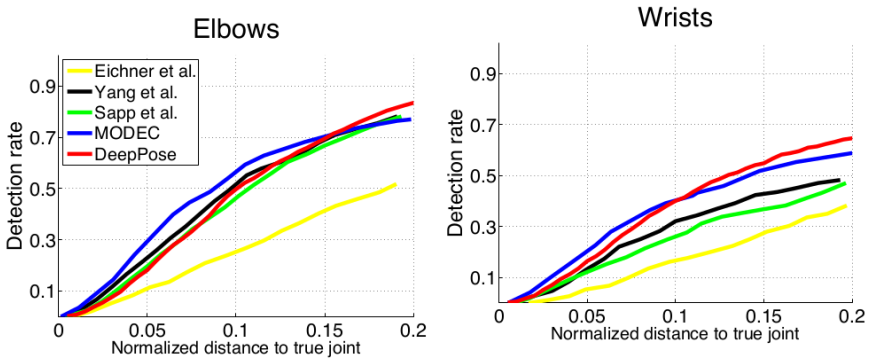


Figure 3.1.9: Percentage of Detected Joints for multiple models (elbows and wrists), including [TS14] and [YR13], on the Buffy dataset. Image taken from [TS14].

3.1.2.2 *Convolutional Pose Machine*

Another approach for pose estimation was proposed by [WRKS16]. The authors base their work on *traditional* Pose machines proposed by [RMH+14]. In comparison to [TS14], they do not regress the joint coordinates directly using fully-connected layers, but use a fully convolutional architecture to compute *belief maps*. These maps are very similar to the part evidence maps discussed in (**Sec.** 3.1.1.1), since they indicate the belief of a joint being present at a certain pixel position.

Pose machines are a general framework for estimating pose, focusing on refining initial pose predictions based on context from previous decisions. The modular nature allows to easily use different predictors and encoders, which will be explained in the next section.

The Pose machine is made up of cascaded *stages*. The first stage uses a predictor g_1, which predicts, for each pixel in the input image, how likely it belongs to part p. The results are P belief maps, one for each part to detect. In all subsequent stages, the output belief maps of the previous stage are encoded back into image features using an encoder function φ. Then, these encoded features are combined with the local image features of the current stage and passed into the stage's predictor. This way, the predictor not only has the image features as a basis for decision, but also the previously estimated joint location for context. See (**Fig.** 3.1.10) for a visualization of the pipeline.

In [RMH+14], the authors use boosted random forests for their stage predictors g_i. To extract features, they utilize *Histogram of Oriented Gradients (HoG)* descriptors [DT05]. Their encoder φ is made up of two parts. The first part is referred to, by the

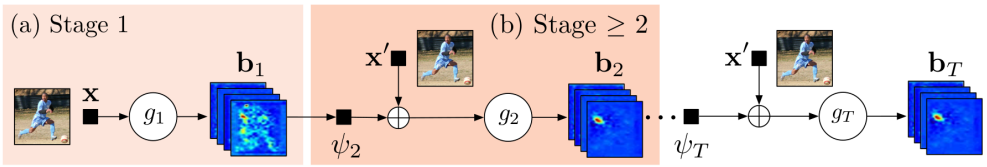


Figure 3.1.10: Cascading stages of the Convolutional Pose Machine. Image taken from [WRKS16].

authors, as *context patch features*. For a given pixel coordinate $z = (x, y)$ for each belief map, the authors concatenate the belief scores in a 5x5 grid around z. Afterwards, the concatenated scores from each belief map are again concatenated across belief maps into a final feature vector φ_1. The second part, which the authors refer to as *context offset features*, again iterates over all pixel locations z. Given the pixel location of the three highest peaks in the belief map, the feature vector is given by concatenating the relative position to these peaks, given z, in polar coordinates. Again, they concatenate all feature vectors over all belief maps, resulting in φ_2. Finally, their context feature vector is given by $\varphi = [\varphi_1, \varphi_2]$. See (**Fig.** 3.1.11) for a visualization of φ_1 and φ_2.

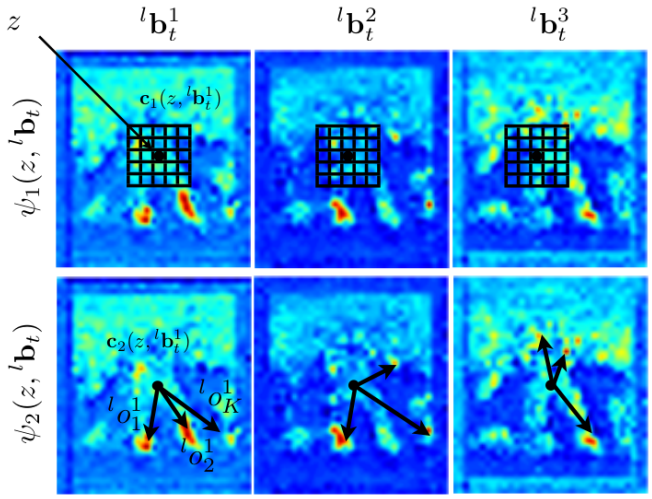


Figure 3.1.11: Example of context feature extraction, as proposed by [RMH+14], for three belief maps ${}^l\mathbf{b}_t^1$, ${}^l\mathbf{b}_t^2$ and ${}^l\mathbf{b}_t^3$ (columns). **Top row:** Extraction from scores in a 5x5 grid around the pixel z. **Bottom row:** Each arrow points towards one of the top 3 peaks in the belief map. The polar coordinates with regards to z are then concatenated into the feature vector. Image taken from [RMH+14].

In [WRKS16], the authors used convolutional neural networks for their predictors g_i. In fact, the complete architecture is a fully convolutional neural network, allowing end-to-end training of both the decoder and predictor. Each stage consists of a feature extraction network, of which the architecture is presented in (**Tab.** 3.1.2) (layers 1 to 7). This leads to the following formulation for the output belief maps $\boldsymbol{b}_1$:

$$\boldsymbol{b}_1 = g_1(x) \tag{3.1.18}$$

For context encoder φ, the authors first compute the receptive field size of the previous stage. Then, when computing the context of position $z = (x, y)$, they center a window of that size around z in the belief maps. This leads to

$$\boldsymbol{b}_t = g_t(x_z, \varphi_t(z, \boldsymbol{b}_{t-1})), \tag{3.1.19}$$

as the output belief maps of stage t with $t > 1$.

The layers in (**Tab.** 3.1.2) denoted with a_1 to a_3 refer to the layers after the feature extraction network in the first stage s_1. Analogously, b_1 to b_5 refer to the layers after the feature extraction network in all subsequent stages $s_i, i > 1$. The architecture of the first stage, compared to all subsequent stages, differs because the authors want the first stage to have a lower receptive field for a more precise extraction of local features. In fact, the design was chosen so that, after the second stage, the receptive field of the network is roughly the size of the input image. The authors claim that, through experimentation, this provided the best overall accuracy.

The authors evaluated the amount of stages to use. They find that, after 5 stages, the performance of the network stops increasing, which is why they choose to use 5 stages for all subsequent experiments.

To illustrate why context belief maps can help the network to correct itself, the authors provide the visualization in (**Fig.** 3.1.12). They argue that, even when the

nr.	1	2	3	4	5	6	7	a_1	a_2	a_3	b_1	b_2	b_3	b_4	b_5
layer type	conv	pool	conv	pool	conv	pool	conv	conv	conv	conv	conv	conv	conv	conv	conv
layer size	9 x 9	2 x 2	9 x 9	2 x 2	9 x 9	2 x 2	5 x 5	9 x 9	1 x 1	1 x 1	11 x 11	11 x 11	11 x 11	1 x 1	1 x 1

Table 3.1.2: Network architecture of the Convolutional Pose Machine. Layers 1 to 7 refer to the feature extraction subnet, which is identical (in terms of architecture) in each stage. However, stages $t > 1$ share the weights of the feature extraction layer to to share the computed image features across stages. a_1 to a_3 are layers attached to the feature extraction subnet in the first stage, while b_1 to b_5 are layeres attached to the feature extraction subnet in all subsequent stages.

belief maps after the first stage are noisy, they can provide useful context information since not every joint is equally hard to detect. As an example, they argue that *head* and *neck* joints are easier to detect than other joints. Instead of *right elbow* and *left elbow*, where there are two very similar joints in each image, *head* and *neck* are only present once. Thus, their position in the first stage belief maps can aid in finding more difficult joints like *right shoulder*. Similarly, the lack of context information can also help in lessening the score in the belief maps. For example, consider the local information around *right shoulder*. Normally, necks and shoulders are near to each other. When there is no indication of *neck* in the belief map around *right shoulder*, this may lead to a lower belief score of *right shoulder* being present at that pixel.

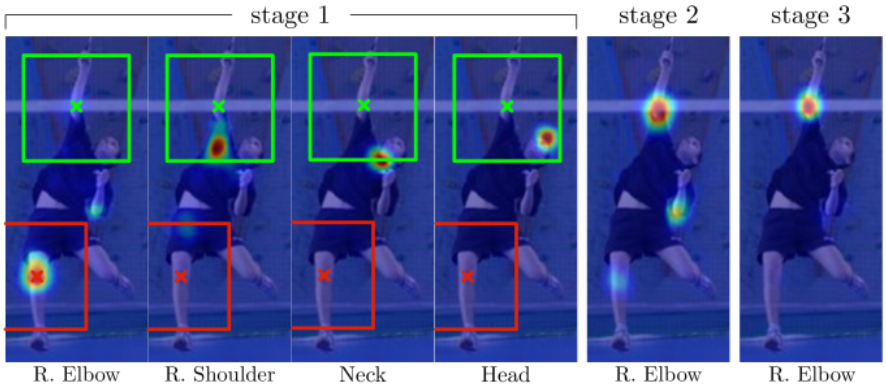


Figure 3.1.12: Visualization of how context information helps in detecting joint positions. The task at hand is to to find *right elbow*. The current belief map for *right elbow* in stage 1 is given in the left most image. In green, the authors visualized the receptive field around the ground truth coordinates for *right elbow*, while red illustrates the receptive field around the wrongfully detected joint. Notice that other, easier to detect joints such as *neck* and *head* fall inside the receptive field of the ground truth, suggesting that their positional information can help in locating *right elbow* in further stages due to the context information they provide. The right most two images show the result after the next two stages, where the network corrected itself successfully. Image taken from [WRKS16].

The authors evaluate their model using three widely used benchmark datasets. These are LSP and FLIC (**Sec.** 3.1.2.1) as well as *MPII Human Pose*. MPII Human Pose is a benchmark, which consists of 40,000 annotated images of people. These images are also often referred to as *in-the-wild* images, since they were downloaded from YouTube videos and are thus representative of every day activaties and situations. In addition, since most YouTube videos are uploaded by private individuals, the image

quality varies, making the benchmark more realisitic. For a complete overview of the MPII dataset, please refer to (**Sec.** 5.1.1).

For evaluating their model, the authors use both PCK (**Sec.** 3.1.1.1) and PCKh metrics. *PCKh* is used as the metric for the MPII dataset, which is very similar to both PCK and PDJ (**Sec.** 3.1.2.1). However, instead of using the person bounding box or torso bounding box for normalization, PCKh uses the head bounding box. [APGS14] argue that PCKh is more useful for evaluating highly articulated poses compared to PCK since the full body bounding box can be fairly large for some poses. Consider a person, jumping in the air, spreading their arms and legs away from their body. This maximizes the person bounding box and thus makes it easier for the PCK metric to consider predictions to be correct when they are fairly far away from the ground truth. The head size, aswell as torso size, however, is quite independent from the type of pose and thus more suitable for evaluation. In addition, they use PCK to compare their results on FLIC and LSP to the literature.

When evaluating their model using the LSP dataset, they found that, using a six stage model, they achieved 84.32 percent accuracy. In comparison, the previous state-of-the-art approach achieved 70 percent. The improvement is especially noticeable when evaluating the challenging joints like ankles. The accuracy increased to 90.5 percent when utilizing additional training data from the MPII dataset. The authors argue that this is due to the fact that the LSP datasets contains noisy annotations when compared to the annotations provided by MPII.

Afterwards, the authors compare their results on the FLIC dataset to many state-of-the-art models, including [TS14] (**Sec.** 3.1.2.1). They compare their result on two challenging type of joints, namely wrists and elbows. Again, they outperform all state-of-the-art approaches. When comparing their results specifically with [TS14] the authors outperform their result by approximately 15 percentage points for wrists and approximately 9 percentage points for elbows. Also, in contrast to their LSP evaluation, they did not incorporate training data from MPII.

When comparing their results with the traditional Pose machine, they find that the convolutional approach improves upon the results on the LSP dataset by an overall PCK accuracy improvement of 42.4 percentage points.

The authors also evaluated the result of intermediate supervision on the vanishing gradient problem (**Sec.** 2.3.1.5). They trained two networks, one with intermediate supervision and one without, for the same amount of time. Their results are visualized in (**Fig.** 3.1.13). The authors argue that the reason why intermediate supervision does help against the vanishing gradient problem is because it replenishes the gradient after each stage and thus keeps the gradient from decreasing when propagating back towards the network.

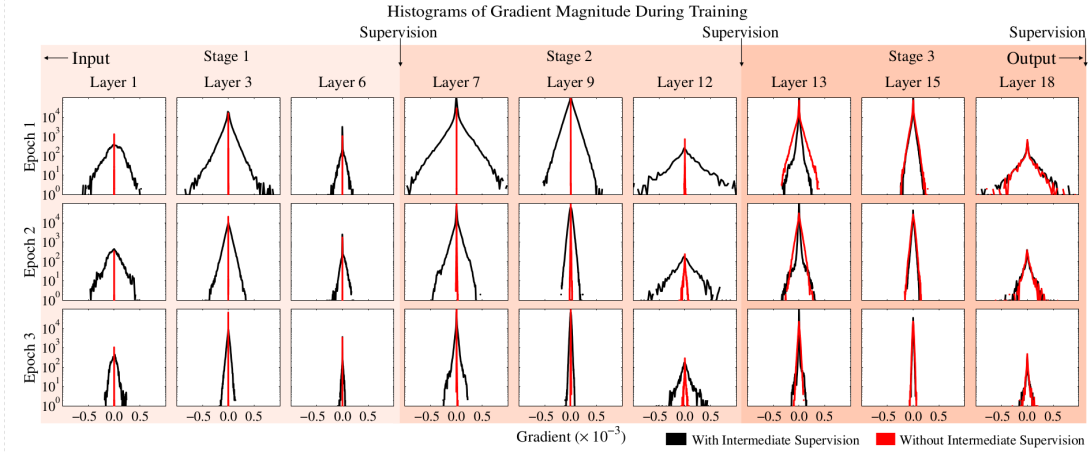


Figure 3.1.13: Gradient magnitude and variance across some layers of the Convolutional Pose Machine. When not using intermediate supervision (**red**), the gradient variance approaches zero towards the beginning of the network, effectively stopping the learning process. When using intermediate supervision, however, notice that the variance of the gradients generally does not approach zero. Image taken from [WRKS16].

3.1.2.3 *Stacked Hourglass*

Building on [TS14], [NYD16] proposed the *Stacked hourglass network*. Similar to [WRKS16], the authors compute *belief maps* encoding joint probabilities instead of regressing the coordinates directly. However, the authors refer to them as *heatmaps*.

Similar to the previously discussed approaches for convolutional architectures, the authors use a stacked architecture, where predictions get refined iteratively. In contrast to [TS14], they do not use crop the image around the estimated joint positions of the previous stage when passing the image to a later stage, but rather but keep the original image in every step (similar to [WRKS16]). The authors argue that refinement of joint positions inside a local subimage is not optimal since many errors occur when misattributing joint positions. Thus, limiting the network to local images might hinder its ability to make big changes in joint positions.

The building blocks of the stacked hourglass architecture are so-called *hourglass modules*. See (**Fig.** 3.1.15) for a visualization of a single hourglass module. The idea is to process the input of the module at different scales and then combine features from different scales using *skip-connections*.

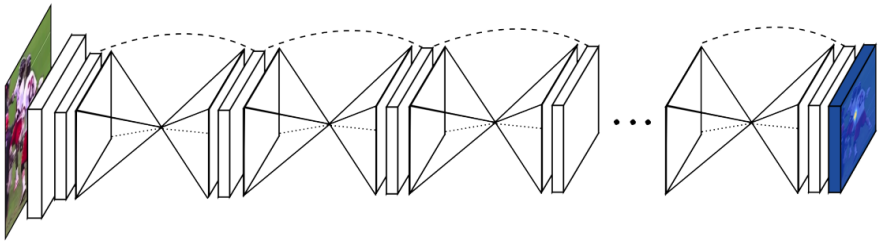


Figure 3.1.14: Schematic visualization of the stacked hourglass network. Notice the hourglass-shaped symmetric submodels connected together. Image taken from [NYD16].

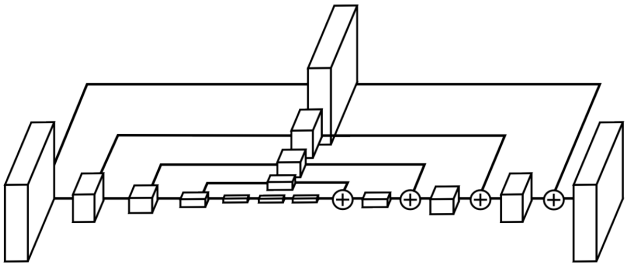


Figure 3.1.15: Visualization of a single hourglass module. Each block is a residual module, consisting of multiple convolutional layers with skip-connections. Notice how the skip-connections combine features at different scale levels. Image taken from [NYD16].

A skip-connection is a path in the network, which branches of at a certain point and gets combined with the original path again further down the network. For example, consider a neural network with three convolutional layers. A skip-connection could branch of before the first convolutional layer, apply some processing in forms of additional convolutional layers, and get added back into the original path after the second convolutional layer, thus effectively *skipping* the processing of the first and second convolutional layers. Thus, earlier features (in terms of network depth) can be reintroduced at a later stage and long-range dependencies between variables can be modelled. One common use-case for skip-connections, which the authors used, is to reintroduce lower level features and add them to higher level features. This way, tasks like pose estimation can incorporate global, high level features like body shape with low level, local features like faces.

Skip-connections get used in *residual blocks* [HZRS16], which are used for more efficient learning in deep networks. Consider two connected convolutional layer with

input x. Let $f(x)$ be the output of the second convolutional layer. Residual models model $f(x) + x$, because they use the skip-connection to add the input back to the output. The authors of [HZRS16] argue that this allows for deeper neural network architectures, because it is easier to fit identity mappings. [HZRS16] evaluated deep neural networks and they found that the error does not only not decrease for very deep networks but starts increasing. They argue that this would not happen if the layers would learn identity functions, because then the error would stagnate instead of increase. In addition, because residual blocks allow for very deep architectures, they achieved substantially better results in comparison to identical architectures without residual blocks.

An example of a residual blocks used in the stacked hourglass network can be seen in (**Fig.** 3.1.16). In fact, all blocks shown in (**Fig.** 3.1.15) are residual blocks.

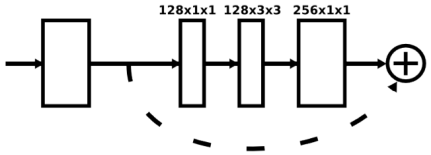


Figure 3.1.16: A residual module [HZRS16], as used in the stacked hourglass network. The module is made up of a batch normalization layer (left) and three convolutional layers, each utilizing the ReLU activation function. The dotted line visualizes the skip-connection. Image taken from [NYD16].

Each hourglass module is a network with a symmetrical architecture. For each downsampling step with a pooling layer there exists a corresponding upsampling step using *nearest-neighbour upsampling*. Consequently, the input and output dimensions of the hourglass module are identical, allowing multiple modules to be applied one after the other.

The input to the module is scaled down using pooling layers, until it reaches a resolution of 4x4 pixels. At each resolution level, features are extracted using a residual module. Once the image is scaled down to 4x4 pixels, upsampling is utilized until the image is back to its original size. Again, residual blocks are used to extract features at the different levels of resolution. Before each pooling layer, a skip-connection is utilized to apply another residual block, after which the resulting features are added to the output feature maps after upscaling. This effectively combines features from two different image scales, because they get combined before upsampling happens.

The authors use another skip-connection between the input and output of the module. This means that the input to each module, which is the output of the previous

module, is added onto its output. The authors argue that this allows the network to refine its prediction with each hourglass module since the high level features, which are the outputs of each module, get processed by consecutive hourglass modules again, creating higher order spatial relationships.

They qualitatively observe that errors made by early modules can be corrected by later modules. Also, they evaluated the difference between $2, 4$ and 8 stacked hourglasses, which will be referred to as s_2, s_4 and s_8. For a better comparability, they altered the hourglass archictecture for the s_2 and s_4 to keep the number of trainable parameters the same between architectures. For each residual block in each hourglass in s_2 the authors substituted 4 residual blocks, effectively quadrupeling the number of residual blocks. Analogously, they doubled the amount of residual blocks for s_4. When training s_2, s_4 and s_8, they found validation accuracies of 87.4%, 87.8% and 88.1% respectively, suggesting that the stacked architecture does contribute to higher overall accuracies. In the final architecture, the authors use 8 hourglass modules.

While training, the output of each hourglass module is not only feed into the next module, but also processed into a *heatmap* using a 1x1 convolutional layer. Using this heatmap, a loss is computed after each hourglass module, which is then propagated back. The authors refer to this process as *intermediate supervision* and they argue that it increases the accuracy of the final prediction, because the network needs to develop a high-level of understanding at each stage of the network. In their quantitative analysis, they showed an increase in validation accuracy of around 3 percent when utilizing intermediate supervision.

For evaluating the stacked archictecture, the authors used the *FLIC* (**Sec.** 3.1.2.1) and *MPII Human Pose* [APGS14] datasets.

In terms of metrics, the authors take the same approach as [WRKS16], where they use PCKh for MPII Human Pose and PCK for FLIC. For *FLIC*, they defined $\alpha = 0.2$, which means that 20% of the length of the head is used as the threshold. They compare their results to [TS14] and outperform their results on elbows and wrists by 6.7 percentage points (99.0% compared to 92.6%) and 15 percentage points (97% compared to 82%), respectively. In addition, they outperformed [WRKS16], the state-of-the-art appraoch at that time (**Sec.** 3.1.2.2).

With *MPII Human Pose*, the authors use a threshold of $\alpha = 0.5$ to compare their results to the state-of-the-art approaches. Also, PCKh was used instead of PCK, as described before, since PCKh was proposed in [APGS14], and thus most models using the MPII dataset use PCKh for evaluation. They outperformed the state-of-the-art approaches, including [WRKS16]. In particular, they achieved 90.9% overall accuracy as compared to 88.5% achieved by [WRKS16]. Also, they achieved significantly better results on each individual part, including a 3.5 percentage point increase on the average

over the most challenging joints (wrists, elbows, kness and ankles) in comparison to [WRKS16].

3.2 VIDEO-BASED HUMAN ACTION RECOGNITION

In the following chapter, approaches for Human Action Recognition on video data will be discussed. First, some prominent works utilizing hand-crafted features will be discussed. These approaches still perform well in modern scenarios and are sometimes used in conjunction with modern, neural network based approaches. Second, the Two-Stream architecture will be discussed, as it is the foundation for many state-of-the-art approaches today. Finally, Human Action Recognition approaches explicitly utilizing pose feature information are presented, since they show promise for end-to-end learning of the Human Action Recognition pipeline by training the pose estimator in tandem with the action classification network.

3.2.1 *Shallow Methods*

3.2.1.1 *Learning realistic human actions from movies*

For a long time, action recognition research focused on simple video datasets such as KTH [SLC04], which feature a small number of different actions, recorded in simple environments like laboratories. While important for the initial research into the methodology, such datasets do not represent real world scenarios and are thus not ideal when deciding whether an algorithm performs well in the real world.

The authors of [LMSR08] gathered a novel dataset, which will be referred to as the *Hollywood1* dataset. This dataset was gathered from movie scenes, which are close to real world scenarios in the sense that they have diverse backgrounds, feature occlusion pf the subject by other entities in the scene as well as differences in clothing, overall making the action recognition process more challenging than in previous datasets.

The dataset contains eight different actions, *Answer Phone, Get Out Of Car, Handshake, Hug Person, Kiss, Sit Down, Sit Up* and *Stand Up*, which were gathered from 12 movies for the training and from 21 different movies for the test set, resulting in 219 and 211 clips respectively.

In addition to the new dataset, the authors also propose a novel approach for action recognition, building on the work of [Lap05], where the Harris corner detector is expanded to also detect interest points in the temporal dimension, making it suitable for action recognition in videos.

The Harris corner detector [HS88] finds corners, also referred to in the literature as *interest points*, by finding locations where the image expresses significant changes of intensity in both x and y direction [Lap05]. By using a small window W, which is moved over the image I, and displacing the content of the window, an error term can be computed to the original, non-displaced image patch, using the following formula:

$$f(\Delta x, \Delta y) = \sum_{x_i, y_i \in W} (I(x_i, y_i) - I(x_i + \Delta x, y_i + \Delta y))^2. \tag{3.2.1}$$

By using the Taylor series to approximate the second term of the subtraction, the error formula simplifies to

$$\begin{aligned} f(\Delta x, \Delta y) &\approx \sum_{x_i, y_i \in W} (I_x(x_i, y_i)\Delta x + I_y(x_i, y_i)\Delta y)^2 \\ &\approx (\Delta x, \Delta y) M \begin{pmatrix} \Delta x \\ \Delta y \end{pmatrix}. \end{aligned} \tag{3.2.2}$$

$M = \sum_{x_i, y_i \in W} \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$ is the second-moment matrix, with I_x, I_y being the partial derivatives of I in x and y direction, respectively. According to [Lap05], M can be interpreted to describe the covariances around the window center. Using the eigenvalues λ_1, λ_2 of M, edges and corners can be detected in the following way. If $\lambda_1 >> \lambda_2$ or $\lambda_1 << \lambda_2$, then an edge has been detected, since large change is only detected in one direction. Similarly, if both eigenvalues are small, then very little change is detected in both directions, meaning that the patch does neither contain an edge nor a corner. However, if both eigenvalues are sufficiently large, a corner is detected, since significant change in both directions was observed.

To detect these interest points in a spatio-temporal context, [Lap05] extended the Harris corner detector to three dimensions. This means that the image sequence needs to express high changes in x, y and the temporal direction t. The first change is the dimension of M, which is now a 3×3 matrix, given by the following formula:

$$M_3 = \sum_{x_i, y_i, t_i \in W} \begin{bmatrix} I_x^2 & I_x I_y & I_x I_t \\ I_x I_y & I_y^2 & I_y I_t \\ I_x I_t & I_y I_t & I_t^2 \end{bmatrix}. \tag{3.2.3}$$

This in turn results in three eigenvalues λ_1, λ_2 and λ_3, which can then be used as discussed previously to detect interest points. An example of detected spatio-temporal interest points are shown in (**Fig.** 3.2.1).

Figure 3.2.1: Two examples of spatio-temporal interest points calculated on move frames. **Left**: Scene of two people shaking hands. Notice the cluster of interest points around the subjects hands. **Right**: A person gets out of the car. Again, notice the cluster forming around the car door as well as the person performing the action. Image taken from [LMSR08].

Using the computed spatio-temporal interest points, the authors proceed to extract image descriptors around them. First, they define a volume around each interest point, the size of which is dependent upon the image scale since they compute features for multiple scales of the original image. Each of those volumes is then further partitioned into subvolumes of size $3 \times 3 \times 2$, where 2 refers to the number of consecutive frames. On each of those subvolumes, the authors compute *Histogram of Oriented Gradients (HoG)* as well as *Histogram of Oriented Flow (HoF)* descriptors. Similar to HoG, Histogram of Oriented Flow discretises and clusters the values of a optical flow image into bins. Then, a histogram is created, containing the number of items for each bin. The authors, however, do not mention which algorithm was used to compute the optical flow. The histograms of all subvolumes are first normalized and then concatenated, resulting in a single HoG and HoF descriptor per volume.

Before classification, the authors utilize the bag-of-features approach to cluster the computed image descriptors. They take $100,000$ random features computed earlier and cluster them into $4,000$ clusters using the k-Means algorithm [Mac67][Llo82]. The number of clusters was chosen because the authors claim that, empirically, this leads to good results.

To compute the descriptor for a video clip, the authors divide the whole volume of the clip into three horizontal subvolumes. They refer to this as the spatio-temporal grid. For each subvolume, the descriptors computed are assigned to the nearest of the

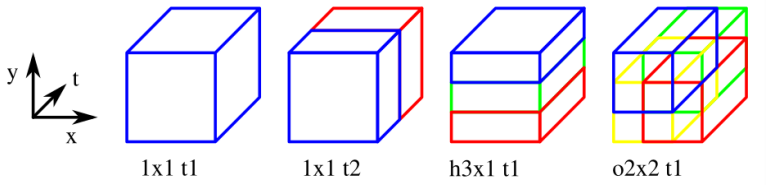


Figure 3.2.2: Four examples of spatio-temporal grid evaluated by the authors. **From left to right**: **(1)** Using the whole clip to compute the histogram. , **(2)** Split the number of frames in half, compute separate histograms and concatenate afterwards, **(3)** Split the clip horizontally into three subvolumes, spanning the whole temporal dimension, **(4)** Choose overlapping subvolumes in the spatial dimensions. The authors found **(3)** to achieve the best results. Image taken from [LMSR08].

4,000 cluster centers and a histogram is computed over the cluster centers. Finally, the histograms of all subvolumes are concatenated and normalized, resulting in the final descriptor of the video clip. The authors evaluated different methods of subdividing the video clip and found the approach using three horizontal subvolumes to perform the best. They argue that this is due to the nature of the dataset, where (mostly upright) subjects perform actions, and thus distinctions in horizontal direction are more important than in vertical direction. See (**Fig.** 3.2.2) for a visualization of some spatio-temporal grids evaluated by the authors. The grid described above is shown at position **(3)**.

First, the authors compare their approach to the state-of-the-art approaches on the KTH benchmark [SLC04]. This dataset contains the classes *Walking*, *Jogging*, *Running*, *Boxing*, *Waving* and *Clapping*. The video sequences are recorded in front of mostly uniform backgrounds such as white walls or on grass. The authors achieve an increase in accuracy of 5.1 percentage points over the previous best approach by [WC07]. The classes which the authors approach wrongly classified the most were *Jogging* and *Running*, which are very similar actions. When evaluating their approach on the *Hollywood1* dataset, they report average precision values of 18.2 percent for the worst class (*Sit Up*) and 53.3 percent for the most accurate class (*Kiss*). Overall, their approach achieves a mean average precision of 38.38 percent.

3.2.1.2 *Dense Trajectories*

Similar to [LMSR08], many approaches in the action recognition literature focused on creating spatio-temporal descriptors, often based on known 2D descriptors like 3D-SIFT [SAS07] and HOG3D [KMS08]. However, the authors of [WKSL13] argue

that, due to the different characteristics of the spatial and temporal domain, tracking 2D descriptors over time should lead to a higher accuracy in action recognition. They propose to densely sample points on the first frame of a video clip and then tracking them on trajectories throughout the clip. Afterwards, image descriptors are computed along these dense trajectories.

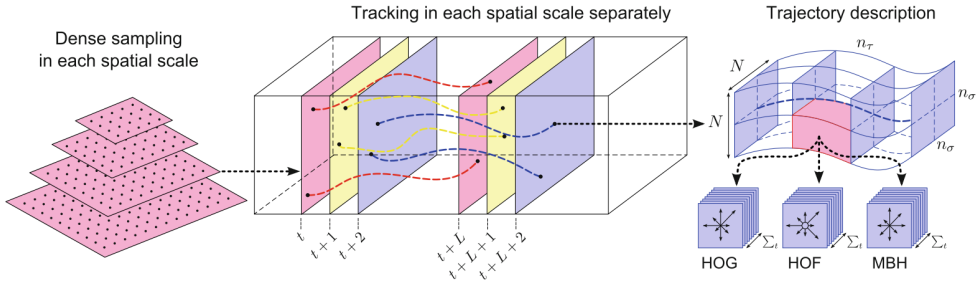


Figure 3.2.3: Overview of the Dense Trajectories method proposed by [WKSL13]. First, dense interest points are sampled at different scales of the first frame of the video clip. Then, using precomputed optical flow, these points are tracked through the video, resulting in a trajectory. Afterwards, image descriptors are computed along the trajectory. Image taken from [WKSL13].

They start by sampling the initial interest points in a 5 pixel grid. Afterwards, they decide to prune the grid to reduce the amount of computation necessary. This is achieved by removing sample points which are located inside homogeneous areas, e.g., walls in the background or other flat surfaces. This is necessary because reliably tracking points over homogeneous areas using optical flow is impossible [WS13]. Also, optical flow is computed on all consecutive frame pairs using the optical flow algorithm provided by the OpenCV library [Braoo]. Then, they proceed to track each interest point using the optical flow field for 15 frames, which results in a trajectory $(P_1, P_2, \ldots, P_t)$ with $t \in [1, 15]$. This process is repeated for different scales of the image in order to extract local and global image features. See (**Fig.** 3.2.3) for visualization of the process. The authors argue that limiting the length of a trajectory is necessary in order to avoid the trajectory from drifting away from the initial interest point too much. In the case where a trajectory is too stationary the authors prune it afterwards because they argue that such a trajectory does not contain sufficient motion information to be useful. Analogously, in the case where a trajectory contains a jump which is higher than 70 percent of the overall displacement since the initial interest point the trajectory is also pruned, since, according to the authors, such a large displacement is usually present due to errors. Also, in the case where an interest point vanishes from one

frame to another or a displacement larger than a certain threshold is occurring they decide to start a new trajectory by sampling a new initial point in the area where the last point was. The authors do not, however, specify how large this threshold is.

Along each trajectory, the authors compute four different descriptors. The first descriptor contains the relative motion between consecutive trajectory points $\Delta P_t = (P_{t+1} - P_t) = (x_{t+1} - x_t, y_{t+1} - y_t)$. This vector is further normalized by the sum over all relative displacements, resulting in the following descriptor T given the length of the trajectory L:

$$T = \frac{(\Delta P_t, \ldots, \Delta P_{t+L-1})}{\sum_t^{t+L-1} \|\Delta P_t\|} \tag{3.2.4}$$

The authors use *Histogram of Oriented Gradients (HoG)* [DT05], *Histogram of Optical Flow (HoF)* [LMSR08] (see (**Sec.** 3.2.1.1) for more information) as well as *Motion Boundary Histograms (MBH)* [DTS06] to extract features along the trajectories, similar to the approach of [LMSR08] (**Sec.** 3.2.1.1). Centered at each interest point along a trajectory, a $N \times N$ pixel area is defined with $N = 32$. These areas are then again subdivided into $n_\sigma \times n_\sigma$ subareas with $n_\sigma = 2$. The descriptors are then computed on each of these subareas and descriptors over $n_\tau = 3$ timesteps are then summed to form a single descriptor for each $n_\sigma \times n_\sigma \times n_\tau$ subvolume. Afterwards, these subvolume descriptors are aggregated, resulting in a single descriptor for each type (HoG, HoF, MBH) and each subvolume $N \times N \times n_\tau$. See (**Fig.** 3.2.3) (right) for a visualization of this process.

Motion Boundary Histograms, originally proposed by [DTS06], are computed using optical flow between consecutive frames. First, optical flow for both x and y direction is computed. Afterwards, gradients are computed on both x and y flow images. On these gradient images, a histogram is computed analogously to HoG using eight bins. It is argued that MBH is highly robust to camera motion since this kind of motion is mostly constant in a movie context and thus results in the gradient of the optical flow being close to zero. The authors decide to compute separate descriptors for x and y, resulting in MBHx and MBHy. For HoG and HoF, they also chose to use eight bins for quantization. Additionally, for HoF, a ninth bin is added in cases where the flow magnitude is lower than a not specified threshold.

The authors evaluate the contribution of each of the four descriptor types to the overall accuracy on multiple datasets. First, they find that the trajectory descriptor using the relative displacement vectors outperforms HoG on datasets where the background is not complex, i.e., in a laboratory environment. They argue that this is because tracking is significantly easier in such a environment. On datasets which focus

on sport activities, HoG consistently outperforms HoF. The authors argue that this is due to the importance of spatial context in sports actions since they often involve objects, i.e., a ball or a javelin, and specific environmental scenarios, such as arenas or track fields. Additionally, the authors find that MBH outperforms all other descriptors on most datasets. Especially in videos which are recorded by private individuals using cheap cameras or mobile phones MBH gains an advantage over other descriptors because it suppresses camera motion. In general, according to the authors, using all descriptors in combination for classification yields higher accuracies than only using a subset, suggesting that both temporal and spatial information are necessary for high accuracy.

In addition, they compared their findings against [LMSR08], discussed in (**Sec.** 3.2.1.1). For comparison, they used two datasets. The *Youtube* dataset [LLS09] contains video clips gathered from the video streaming platform YouTube. The dataset contains 11 different actions, mostly from the context of sport. The other dataset is *Hollywood2* [MLS09], which is an extension of the previously discussed *Hollywood1* dataset [LMSR08] (see (**Sec.** 3.2.1.1)). First, the number of actions was increased to 12 by adding *Fight Person, Run, Eat* and *Drive Car*. Additionally, the number of training video clips was increased to 810 while the number of test clips was increased to 884.

To accurately compare both approaches, they utilize the interest points computed by [LMSR08] to compute the HoG and HoF descriptors. In a direct comparison, both HoG and HoF descriptors on their own lead to the same accuracy regardless of whether the trajectory is used or the interest points computed by [LMSR08]. However, the MBH descriptor consistently outperforms both HoG and HoF, and by combining MBH with HoG and HoF the authors achieve a significant accuracy gain of 7.0 percentage points on the YouTube dataset and 4.2 percentage points on the Hollywood2 dataset over [LMSR08] (76.2 versus 69.2 percent and 51.9 versus 47.7 percent, respectively).

3.2.1.3 *Improved Dense Trajectories*

After their work in [WKSL13], some of the same authors improved upon their work in [WS13]. The new approach is named *Improved Dense Trajectories (IDT)*. The core idea is to remove the camera motion before computing the dense trajectories, resulting in a optical flow field which contains less background information. To achieve this, the authors assume that two consecutive frames can be related to each other using a homography.

In general, a homography H relates two points x and x' in the same plane by $x' = Hx$ [VL01]. H is defined as a 3×3 matrix since the image points $x = (x_x, x_y, 1)$ are projections of three dimensional points onto the camera plain. Also, a homography

can express the relationship between the same point but viewed by two different cameras, or, as shown later, a single camera before and after displacement. Let F_1, F_2 be two images, taken by the same camera, but at different world positions. Using matching points between F_1 and F_2, H can be computed. With H, the camera movement in F_2 can be removed, resulting in a reduction of background movement. This procedure is further referred to as warping.

Figure 3.2.4: Overview over the different stages of the Improved Dense Trajectories algorithm. **Outer left**: Two consecutive frames displayed over another. Notice the blurry background. **Middle left**: Computed optical flow without warping. **Middle right**: Optical flow computed where the second frame was warped according to the computed homography. Notice that the background motion has vanished and the motion of the subject is much more pronounced. **Outer right**: Trajectories computed on the warped flow. Background trajectories, which are removed, are displayed in white .Image taken from [WS13] and modified from vertical stack to horizontal stack.

The authors argue that the homography assumption, requiring the motion between frames to be planar, holds in their scenarios most of the times since the movement by independent subjects in the frame, such as actors, is small enough to be negligible. To compute H, the authors first compute SURF detectors [BTVG06], which are then used to compute point correspondences in consecutive frames by using the nearest neighbour algorithm. They argue that SURF detectors robust to motion blur and are thus more useful for this task than other descriptors. Additionally, they track points using the optical flow by sampling points and tracking them in the consecutive frame. Finally, they use the matched points to compute H using the RANSAC algorithm by [FB81]. This algorithm randomly samples a subset of the input point pairs and computes H by solving the following equation [VL01]:

$$x' \times Hx = 0 \tag{3.2.5}$$

The points pairs, which were not used for estimation, are then used to evaluate the accuracy of H. If the accuracy is above a certain threshold, the point pair is considered an *inlier*, and an *outlier* otherwise. This procedure is repeated a fixed number of times and the parameters of H that produced the most inliers are chosen.

Estimation of H can be problematic when a subject is very dominant in the frame since the homography assumption does not hold for independently moving objects in the scene [WS13]. Thus, the authors suggest to use a human detector in order to remove point pairs which lie inside a human bounding box. They use the, at that time, state-of-the-art human detector proposed by [PSF12].

After estimating H, the authors warp the second image in each consecutive frame pair using H and recompute optical flow between the frames. Then, using the new optical flow images, they compute the HoF, trajectory and MBH descriptors identically to [WKSL13]. The HoG descriptors, on the other hand, are created on the original frames without warping applied. As another improvement, they remove background trajectories based on the computed trajectory descriptors. This is achieved by computing the maximum magnitude of each trajectory and removing the trajectory if the magnitude is less than 1 pixel, since this means that the trajectory is consistent with the camera motion [WS13]. See (**Fig.** 3.2.4) for a visualization of these processing steps.

In their evaluation, the authors first evaluate the difference between the Improved Dense Trajectories approach to their baseline from [WKSL13]. They find that by using the warped optical flow alone the accuracy increases by 3 to 4 percentage points over the baseline, depending on the dataset. Also, if they remove background trajectories, they gain an additional 1 percentage points over the baseline.

When investigating the individual contributions of the descriptors, the first thing the authors notices was that the HoG descriptors do not significantly increase over the baseline. However, this is to be expected, since HoG is a static descriptor. The authors attribute the small accuracy increase in HoG to the removal of background trajectories. Also, they notice that the HoF descriptor is comparable to MBH if the warped flow field is used, whereas MBH in the original Dense Trajectories approach significantly outperformed HoF.

The authors additionally investigated the contribution of the human detector. For the experiment, they annotated a subset of the datasets with ground truth human bounding boxes and compared the accuracy gain over the baseline, which does not use human bounding boxes, and the bounding boxes computed by [PSF12]. They found that, using the computed bounding box, accuracy increases upon the baseline by 1.8 percentage points. Using the ground truth, an additional increase in accuracy of 1.2 percentage points was observed. The authors argue that the accuracy increases because the homography H is more accurate when using a human bounding box, which then leads to a better approximation of the camera motion and in turn to better motion descriptors like HoF and MBH.

3.2.2 *HAR using Two-Stream Convolutional Neural Networks*

3.2.2.1 *Two-Stream CNN*

Recently, many approaches for action recognition in videos incorporate a concept referred to as a *Two-Stream Convolutional neural network*. For a better understanding of the approaches discussed later, a short overview over this concept is provided.

As discussed previously (see (**Sec.** 3.2.1)), the combination of spatial features, such as the HoG descriptor, and temporal features, such as the HoF descriptor, yielded significant improvements over the individual components. Many methods utilized precomputed optical flow fields for computing temporal features (see (**Sec.** 3.2.1.1) and (**Sec.** 3.2.1.2)). With the success that CNNs achieved in computer vision tasks on still images, such as image classification and image segmentation, [KTS+14] proposed a similar approach for use in video action recognition. The authors, however, did not incorporate temporal information in form of optical flow fields into their architecture. Instead, they relied solely on passing multiple consecutive frames to 2D convolutional layers. The authors were not able to achieve comparable results to the hand-crafted feature approaches proposed by [WS13] (see (**Sec.** 3.2.1.3)), gaining 65.4 percent accuracy compared to 85.9 percent on the UCF101 dataset [SZS12].

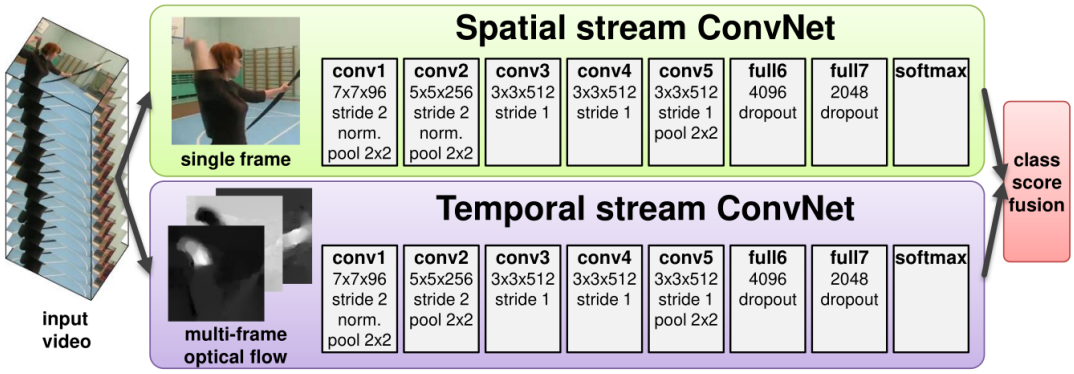


Figure 3.2.5: The Two-Stream architecture used by [SZ14]. Notice that the spatial and temporal networks have nearly identical architecture, with the difference that the second convolutional layer in the temporal network does not have batch normalization to reduce memory consumption. Image taken from [SZ14].

In order to improve upon this first approach, [SZ14] proposed the *Two-Stream Convolutional neural network*. Such a network is comprised of two individual networks. The first network extracts features from a single RGB frame of the video and, by using

a softmax layer, classifies the present action. Simultaneously, the second network processes L consecutive optical flow images and also outputs a classification using a softmax layer. Both classification outputs are then fed to a multiclass linear Support Vector machine which then outputs the final action prediction. In [SZ14], the networks have almost the same, shallow architecture, consisting of 5 convolutional and 2 fully-connected layers, followed by a softmax layer. See (**Fig.** 3.2.5) for a visualization of the used architecture.

Because the spatial network classifies static images, the authors pretrain it on the ImageNet dataset [DDS+09]. This step alone improves the accuracy achieved on the UCF101 dataset by 20.5 percentage points from 52.3 percent when training from scratch, suggesting that it is highly beneficial to design action recognition architectures in a way that they can be pretrained on large static image datasets. The authors did not pretrain the temporal network since sufficiently large datasets were not available.

The authors evaluated the length of consecutive optical flow frames L of an RGB frame to pass into the flow network. They noticed that $L = 5$ and $L = 10$ flow frames lead to similar performance, with $L = 10$ being slightly better. Thus, they decided to set $L = 10$. Additionally, they found that subtracting the mean flow image from every flow frame significantly improved the overall accuracy. They argue that this is due to the fact that such a subtraction reduces the camera motion, similar to the homography approach by [WS13] (see (**Sec.** 3.2.1.3)).

To train their model, the authors utilized *multi task learning*. They combined two datasets for training, UCF101 and HMDB51 [KJG+11], by changing the architecture so that there are two final softmax layers, one which outputs the prediction for UCF101 and one which outputs the prediction for HMDB51. The authors found that training in such a way has a regularisation aspect, reducing the risk of overfitting on any of the two datasets. For computing optical flow, they use the algorithm provided by the OpenCV library [Bra00].

The authors also evaluated their approach against the Improved Dense Trajectories approach by [WS13] (see (**Sec.** 3.2.1.3)). On the UCF101 dataset, the Two-Stream network outperformed IDT by 2.1 percentage points (85.9 percent versus 88.0 percent). They argue that this is the case because the Two-Stream network can generalise the HoG, HoF and MBH descriptors used in IDT using its convolutional layers.

3.2.2.2 *Quo Vadis, Action recognition?*

Recently, [CZ17] published a video dataset called *Kinetics*, which claims to be big enough to be used for pretraining video processing neural networks, similar to how ImageNet [DDS+09] is used for pretraining static image processing pipelines. To

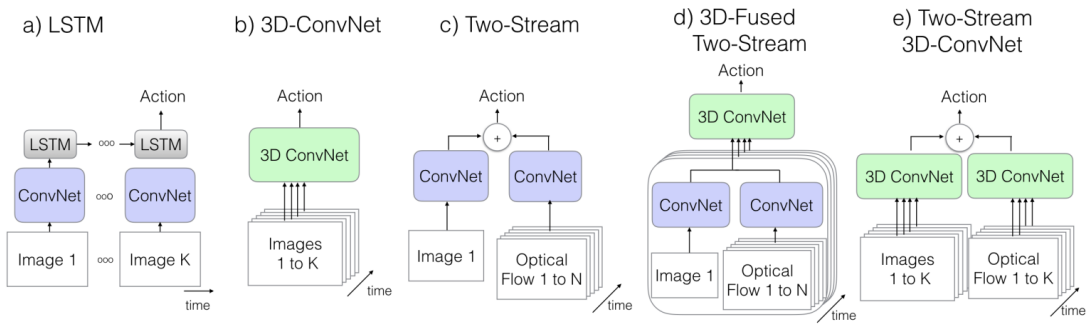


Figure 3.2.6: The five architecture types compared in [CZ17]. **e)** is a novel approach proposed by the authors, while **c)** is the Two-Stream approach proposed by [SZ14]. Image taken from [CZ17].

inspect how much impact such pretraining in the domain of video has the authors reimplemented 5 popular action recognition architecture types.

First, an architecture where each frame of a video is fed through a 2D convolutional network, whose outputs get aggregated over time using a LSTM layer [HS97]. Second, the authors implement a model which uses 3D convolutional layer to process k frames of a video simultaneously. The idea is that such a model would not need additional temporal components as the 3D layers already capture temporal information. The architecture is based on [TBF+15], however, the authors add additional batch normalization to allow for an accurate comparison to other architectures who use batch normalization. Third, the authors took the Two-Stream architecture by [SZ14]. Fourth, an extension to the Two-Stream architecture by [FPZ16] is added. The extension exchanges the fusion stage of the traditional Two-Stream network with 3D convolutional layers. Also, this approach achieved state-of-the-art results on both UCF101 as well as HMDB51. Lastly, the authors propose a novel architecture based on both the Two-Stream architecture as well as the 3D convolutional network. They use two 3D convolutional networks to extract spatial and temporal classifications and then fuse them together, similar to [SZ14]. They utilize a novel approach for pretraining 3D networks by inflating the parameters to the third dimension. First, they pretrain two 2D networks on ImageNet. Then, they inflate the convolutional layers filter sizes from $N \times N$ by repeating each filter N times and dividing all filters by N. Thus, they change the dimensionality from $N \times N$ to $N \times N \times N$ for each filter. They claim that this produces identical output to the 2D network. An overview over all 5 architectures is provided in (**Fig.** 3.2.6).

For evaluation, the authors pretrained the models on ImageNet and then fine-tuned them on UCF101, HMDB51 as well as Kinetics individually. In general, they noticed that the Two-Stream approaches outperform the other approaches on all datasets. The authors argue that this is due to the flow used since the flow network contributes more to the overall accuracy when compared to the spatial network. Also, they observe that all architectures achieve higher accuracies when pretrained using Kinetics, suggesting that the dataset is suitable for pretraining action recognition models in general.

Moreover, the architecture proposed by the authors achieves state-of-the-art accuracies on both UCF101 (98 percent) as well as HMDB51 (80.7 percent) when first pretrained on ImageNet, followed by additional pretraining on Kinetics. The previous state-of-the-art approach by [FPW16] achieved 94.5 and 70.3 percent, respectively. The improvement in accuracy over [SZ14] is thus 10 percentage points on UCF101 and 21.3 percentage points on HMDB51. Analogously, the performance improvement over [WS13] is 11.6 percentage points on UCF101 and 19 percentage points on HMDB51. Overall, the authors argue that using Kinetics for pretraining and transfer learning is a feasible approach to achieve similar transfer learning capabilities for video clips as with ImageNet and static images.

3.2.3 *HAR using pose information*

3.2.3.1 *Towards understanding action recognition*

In [JGZ+13], the authors evaluate the effect of different parts of a state-of-the-art action recognition pipeline [WKSL13] in order to gain a deeper understanding of the problem (see (**Sec.** 3.2.1.2) for a detailed explanation of the algorithm).

In order to evaluate the impact of different parts of an action recognition pipeline they needed a fully annotated dataset. They took a subset of the HMDB dataset [KJG+11], which they argue is a challenging dataset, and additionally annotated pose and created binary human segmentation maps, which they refer to as Puppet masks (see (**Sec.** 5.1.3) for more information on the dataset).

In addition to the annotated poses and Puppet masks, they computed optical flow on consecutive frames for each video clip as well as annotate a rectangular bounding box around the subject. By substituting ground truth pose, optical flow and bounding boxes into the pipeline of [WKSL13] the authors studied how much these individual features contribute to classification accuracy.

To establish a baseline, they extract features using [WKSL13] for each video clip and then train a Support Vector Machine for action classification. This resulted in an overall accuracy of 56.6 percent. Afterwards, they evaluated different ways to

Figure 3.2.7: Three masked optical flow variations used in [JGZ+13]. **Left**: The estimated optical flow of [WKSL13] masked using the ground truth Puppet mask annotation. **Middle**: Optical flow computed only on the masked subject. **Right**: Combination of both. Notice the sharp edges between the subject and the surrounding flow. Image taken from [JGZ+13].

substitute the ground truth optical flow into the algorithm. The highest accuracy gains over the baseline (11 percentage points) were achieved by using the ground truth flow computed on the masked subject in combination with using the optical flow from around the Puppet mask borders. See (**Fig.** 3.2.7) for a visualization. The authors argue that is most likely due to the fact that the separation between the background flow and subject flow is more distinct, resulting in a sharp motion boundary.

Next, the authors investigate the impact of masking the subject in the RGB image. They use two types of masking. First, they use a bounding box around the subject. Second, they utilize the Puppet mask for masking. When applying the masking in such a way that the image is masked before computing the features (as opposed to densely computing features and then masking afterwards) they find that both bounding box masking and puppet flow masking improve significantly upon the baseline with 5.6 and 8.0 percentage points respectively. They argue that the Puppet mask likely leads to a better accuracy because the optical flow algorithm is easier to compute on the subjects boundary. In addition, they also utilize a state-of-the-art person detector from [BMBM10] to compute the bounding box but find that the computed bounding box is not accurate enough to improve significantly upon the baseline. This suggests that future progress in person detection algorithm could improve upon current action recognition algorithms.

Afterwards, the impact of using pose as a feature was evaluated. The authors generate two types of pose related features. First, they use the joint positions over time as trajectories. Second, they compute additional features related to pose. This includes calculating the distances and relative angles between all joint pairs. In addition, they iterate every triplet of joints and calculate the inner angle spanned by two relative vectors anchored at the third joint. See (**Fig.** 3.2.8) for a visualization of this procedure. Besides this, they encode temporal information by computing the change in the

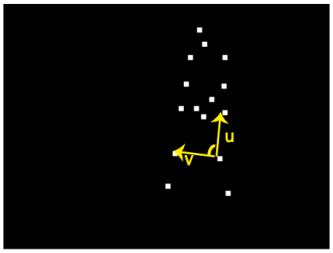


Figure 3.2.8: Visualization of computing the inner angle for each joint triplet. The vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ are calculated from the first joint of each triplet to the other two. Then, the angle between these vectors is computed. Image taken from [JGZ+13].

previously mentioned values for consecutive frames. They find that the accuracy improves by 19 percentage points over the baseline, suggesting that pose information is by far the most discriminative feature to use for action recognition [JGZ+13]. Also, adding the image features to the pose features does not significantly improve upon the accuracy. The authors thus argue that, once pose features are used, the image feature contribution is negligible. Additionally, they evaluate the use of estimated poses using the approach presented by [YR11] (see (**Sec.** 3.1.1.2) for a detailed explanation). They find that, while the estimated pose is of low quality compared to the ground truth, this approach still outperforms the baseline approach, suggesting that even low quality pose information is highly relevant as a feature for action recognition.

3.2.3.2 *Pose-CNN*

After the findings of [JGZ+13] discussed previously, [CLS15] build on their work by proposing a new action descriptor extracted using a Convolutional Neural network. Specifically, they use a Two-Stream network approach similar to [SZ14] (see (**Sec.** 3.2.2.1)) where the network is split into two parallel parts. The first part processes the individual RGB frames, resulting in an appearance feature vector. The second part processes flow images and outputs another feature vector. These two vectors are then concatenated, which results in the final feature vector for the entire video clip. Each part consists of 5 convolutional layers, followed by 3 fully-connected layers. The RGB part is pretrained using the ImageNet dataset [DDS+09] while the flow part is pretrained using the UCF101 dataset [SZS12]. Once pretrained, the last fully-connected layer of both parts is removed, resulting in a output size of the previous layer of 4096, which they use as the extracted feature. See (**Fig.** 3.2.9) for a network visualization.

The authors compute optical flow using [BBPW04] for consecutive frames in each video clip beforehand. They also scale the x and y flow values to the interval $[0, 255]$.

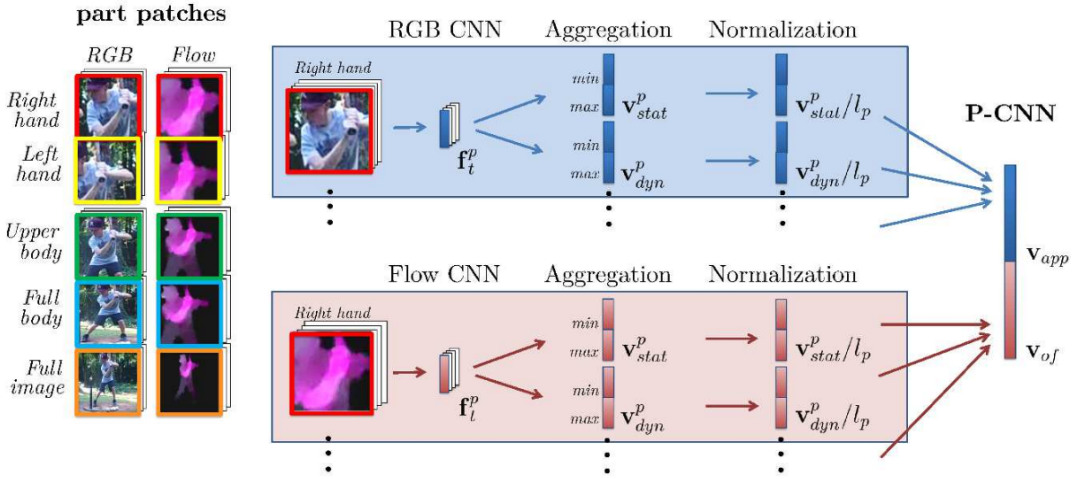


Figure 3.2.9: Visualization of the network architecture used to extract the video descriptor in [CLS15]. **In blue**: The part of the Two-Stream network which extracts the visual feature vector v_{app}. **In red**: The flow feature extraction network part. Image taken from [CLS15].

Then, they concatenate these two flow maps and add a third map with the flow magnitude values, resulting in a 3 channel flow image. They also preestimate the joint positions using the approach by [YR11]. Additionally, they use a dynamic programming approach by [CMAS14] to refine the estimated pose by incorporating flow information as well as temporal information from previous and following poses. Afterwards, using the precomputed joint positions, they extract subimages around the right hand, left hand, upper body, full body and full image from the RGB frames as well as from the flow images.

After passing all subimages through the network, the resulting feature vectors f^p_t for time step t and body part p are aggregated by first computing minimum and maximum values m_i, M_i for each descriptor dimension i. Then, the minimum values for each dimension are concatenated, followed by another concatenation of the maximum values. The authors refer to this vector as the static video descriptor since it is computed per frame without taking future or past frames into account. They compute another descriptor, called the dynamic video descriptor, where the original feature descriptors f^p_t get subtracted by $f^p_{(t-4)}$, thus encoding temporal information. The aggregation of minimum and maximum values following that is identical to the aggregation for the static video descriptor. Finally, both descriptors are normalized and again concatenated, yielding the final video descriptor. The resulting feature

vector is then used as input for a linear Support Vector Machine, used for action classification. See (**Fig.** 3.2.9) for a visualization.

The authors compare their results to the descriptor presented in (**Sec.** 3.2.3.1) (from now on referred to as *HLPF* for High Level Pose Features) as well as to the Improved Dense Trajectories (*IDT*) descriptor presented in (**Sec.** 3.2.1.3), which was the state-of-the-art approach at that time.

When training on the JHMDB dataset, the authors find that their approach does not outperform *HLPF* when using ground truth pose annotations (74.6 and 77.8 percent accuracy respectively). However, when using estimated pose, their approach significantly outperforms *HLPF* by almost 35 percentage points (61.1 as opposed to 25.3). While the authors give no explanation for why this might be the case, one possibility would be that the conclusion of [JGZ+13] that visual features and flow do not significantly improve upon accuracy when also utilizing pose might not hold in all situations. For example, consider two actions like *military salute* and *talking on the phone*, which have similar poses. When also considering image features, the distinction between an empty hand (*military salute*) and a hand containing an object (*talking on the phone*) can aid in the decision process. Also, since the authors use a different approach for estimating pose as opposed to [JGZ+13] and they do not evaluate the accuracy of the pose estimator individually it might be the case that the pose information itself is not of sufficient quality for the pose features computed by *HLPF* to be discriminative enough.

Interestingly, the authors find that, while their approach does not outperform *IDT*, a combination of their descriptor with *IDT* significantly outperforms *IDT* on its own. The authors argue that, because *IDT* uses a larger time window for feature aggregation, actions like *kick ball* achieve higher accuracy while on actions like *shoot gun* and *waving* the image features extracted by their method provide more information and thus higher accuracy. A combination thus improves the accuracy in both scenarios simultaneously.

3.2.3.3 *PoTion*

Building on the work of [SZ14] (see (**Sec.** 3.2.2.1)), [CWRS18] propose a complementary pose-based feature called *Pose motion (PoTion)*. The idea is to track joint positions over time and displaying them in a single frame using a color gradient. The authors argue that such a compact representation can be used in combination to recent Two-Stream approaches to achieve state-of-the-art performance. See (**Fig.** 3.2.10) for a visualization of this approach.

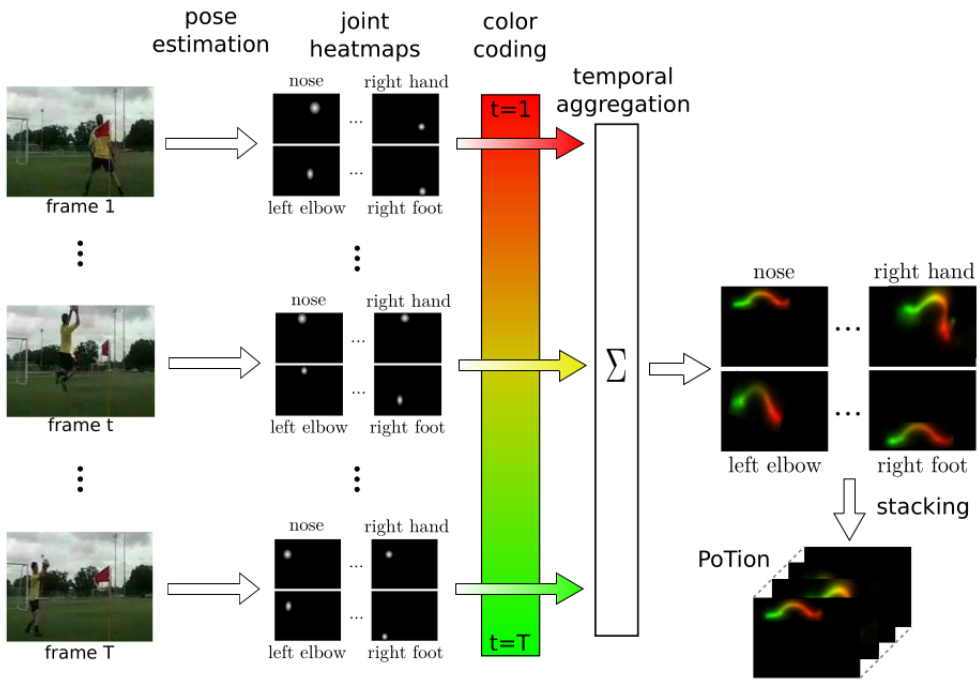


Figure 3.2.10: The approach presented in [CWRS18] first estimates the joint positions for each frame of a video. Then, depending on the frame, the position is colored according to a gradient from, for example, red to green. By aggregating all frames in to a single image the path the joint moved in the frame becomes visible. Image taken from [CWRS18].

First, the authors compute joint positions for each frame of a clip using a pose estimator proposed in [CSWS17]. Then, the authors define C colors to encode the movement for each joint. Using a gradient between these colors, the authors color each joint position based on the frame position in the video. For example, consider $C = 2$ and let these two colors be red and green. Then, the joints in the first frame would be colored red. Analogously, the joints in the last frame would be colored green. In between, the authors define a function for interpolating between the different colors. Notice that the color is solely dependent on the frame position t. See (**Fig.** 3.2.11) for a visualization of these functions. In their evaluation, the authors find that, initially, the accuracy on the datasets increases with the number of colors used. They find that this is the case until $C = 6$, where the accuracy stops increasing. Thus, the authors choose $C = 4$ in order to get both a high accuracy and remain a compact representation.

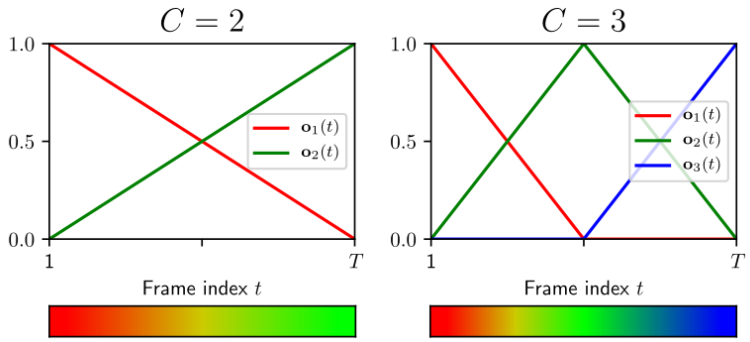


Figure 3.2.11: Example of the color interpolation for two and three different colors. On the bottom, the resulting color for each frame t is shown. Image taken from [CWRS18].

Afterwards, the authors aggregate the colorized frames for each joint into a single RGB image. They evaluate different aggregation techniques. First, they compute the normalized sum U_j for each joint j by summing all colorized frames and dividing by the maximum pixel value over all pixels. The authors notice that, if a joint stays at any position for a period of time, that the intensity increases strongly at this particular position. Thus, they also compute an intensity image $I_j = \sum_c U_j(c)$ by summing all color channels for all pixels in U_j. Then, they compute an additional representation $N_j = \frac{U_j}{\epsilon + I_j}$, which does not contain intensity information. In their evaluation, the authors find they achieve the highest accuracy by using all three representations.

For evaluation, the authors also propose a shallow CNN for classification using only the precomputed representations discussed earlier. The network is made up of 6 convolutional layers, followed by a single full-connected layer. However, the authors find that this shallow classification network alone does not achieve high accuracies compared to the previous work. On the JHMDB dataset, the authors achieve 57.0 percent accuracy (as compared to 61.1 percent achieved by [CLS15] (see (**Sec.** 3.2.3.2))) and on HMDB, the authors achieve 43.7 percent accuracy (as compared to 59.4 and 61.7 percent by [SZ14] (see (**Sec.** 3.2.2.1)) and [WS13] (**Sec.** 3.2.1.3) respectively).

The authors also evaluate the effect of adding their representation to other previous approaches and find that adding it to the previous state-of-the-art approach by [CZ17] (see (**Sec.** 3.2.2.2)) they were able to improve the accuracy slightly by 0.5 (87.9 percent as opposed to 87.4 percent) percentage points using estimated poses on the JHMDB dataset. When using ground truth annotation for the joint positions, the authors observe an improvement of 3 percentage points to 90.4 percent upon [CZ17], indicating

that their representation is complementary to the Two-Stream approach and that it is worthwhile to improve upon the pose estimation component.

3.2.3.4 *Three-Stream Network*

Inspired by the success of the Two-Stream architecture (see (**Sec.** 3.2.2.1)), [KY18] expanded the idea to a Three-Stream approach, using pose information as the third stream.

First, they estimate the poses for each frame using the same pose estimator by [CSWS17] that [CWRS18] use. The authors argue, however, that missing joints need to be interpolated because this leads to higher accuracy on the benchmark datasets. However, they do not provide evidence in form of a direct comparison of accuracy with and without interpolation. They propose two different interpolation strategies, referred to as *temporal interpolation* and *spatial interpolation*. *Temporal interpolation* is used whenever a joint is not visible for a small number of frames. Let f_l be the frame where the joint was last visible. This frame is then followed by n frames, where it is not visible, followed by f_k, where it becomes visible again. Then, the authors linearly interpolate the position of the missing joint for each frame $f_{l+1}, \ldots, f_{l+n}$ using the positions in f_l and f_k. They do, however, not specify what constitutes as a *small number of frames*, i.e., how small n has to be in order for this approach to lead to good results. *Spatial interpolation* is used for long-term occlusion, i.e., for large n. The authors argue that this is necessary since *temporal interpolation* leads to worse results the bigger n gets. Again, they do not specify for which value of n *spatial interpolation* becomes necessary. For *spatial interpolation*, they divide the joints into 5 groups, which can be seen in (**Fig.** 3.2.12). The groups are iterated, starting with the group the missing joint is in. If the other joints in a group are present, they *vote* on the position of the missing joints based on a statistical model of relative joint positions computed on training data beforehand. The average vote is then assumed to be the missing joints position. The authors argue that this leads to good results since the position of joints is highly correlated for certain joints. As an example, they mention the correlation of head and neck positions. This approach and reasoning is similar to [YR11] (see (**Sec.** 3.1.1.2)), where the relative orientation between joints was used as a feature for action recognition.

Once the joint positions are estimated for each frame, the authors order them in a way so that the neighbourhood relationship between joints is kept. They argue that this leads to higher accuracy since the relationship of the joints is a useful feature, but they do not compare it to other approaches for joint ordering. The ordering process is visualized in (**Fig.** 3.2.13). By traversing the tree in such a way, some nodes are

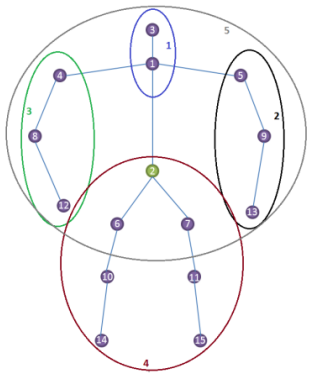


Figure 3.2.12: The authors divide the joints into 5 groups in order to estimate relative positions of joints to each other. Image taken from [KY18].

visited multiple times and thus present multiple times in the pose tensor. That way, the authors argue, neighbourhood relationships are expressed more directly in comparison to simply starting at joint 1 and concatenating the x and y positions. The pose tensor then contains the x and y information for each frame in a row. The authors process the video clips in chunks, so that the pose tensor always has identical dimensions. Also, in the second and third tensor dimension they compute the first and second order derivatives of the joint position coordinates. Finally, the joint positions P_i are normalized with regards to the middle point between *neck* and *belly* joint P_{middle} and with regards to the torso length d using the following two formulas:

$$P_i^{norm} = \frac{P_i}{d}, \qquad P_i^{rel} = P_i^{norm} - P_{middle}. \tag{3.2.6}$$

The authors propose a shallow CNN for classification, similar to the one proposed by [CWRS18] (see (**Sec.** 3.2.3.3)). The network is made up of two convolutional layers, followed by a max pooling layer and a fully-connected layer. Finally, a layer with the softmax activation function is used for classification. The other layers all utilize the ReLU activation function. Since the network is so small, the authors do not pretrain it. For the Two-Stream architecture, they use [WXW+16], which is a Two-Stream network achieving high accuracy on the UCF101 and HMDB datasets. They fuse their pose network into the existing Two-Stream approach by equally weighting all three components.

First, the authors evaluate each component of the Three-Stream architecture independently and find that the pose component (using estimated pose) and spatial

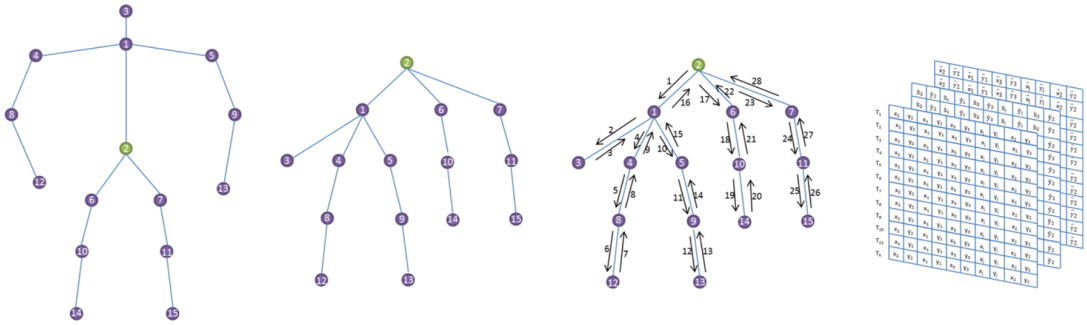


Figure 3.2.13: Visualization of the ordering process. **From left to right**: **a)** The original pose skeleton. The belly joint is highlighted and used as the root node. **b)** A tree is computed based on the previously chosen root node. **c)** Traversion algorithm example. **d)** The final pose vector. Each row contains x and y coordinates of the persons joint locations of one frame. Image taken from [KY18].

component perform worse than the components using optical flow and ground truth pose components, suggesting the importance of motion information for action recognition. When combining the different components, the Three-Stream network performs the best with 78.81 percent accuracy on the JHMDB dataset using estimated pose. When substituting ground truth pose, the accuracy further increases to 83.05 percent.

When comparing the Three-Stream network to previously discussed methods, it performs slightly worse (0.7 percentage points) on the JHMDB dataset, using estimated pose, when compared to [CLS15] (see (**Sec.** 3.2.3.2)). Even when utilizing ground truth pose information, the authors are not able to outperform [CWRS18], who achieved 87.9 percent accuracy on JHMDB using estimated pose. Notice that the pose estimator was identical in both cases, suggesting that either the architecture proposed by [KY18] is not deep enough or that estimating missing joint positions by linear interpolation and statistical modeling leads to worse results.

4

METHOD

4.1 DEEP HAR

In the previous chapter, approaches for estimating pose from still images and predicting actions from video were presented. Although some approaches for HAR incorporate pose information as features in their decision process, estimating the pose and predicting actions were not used in conjunction during the training process. For example, [CWRS18] (**Sec.** 3.2.3.3) estimated the human pose using a dedicated model, before using the information to train their model.

This chapter presents the approach by [LPT18], who argue that training the pose estimator and action recognition model jointly may help in the learning process. In Section 4.1.2, the authors present a novel way to infer joint coordinates from a heatmap, which allows the model to be fully differentiable and thus able to be trained in an end-to-end approach. Next, the model architecture the authors used is presented in Section 4.1.3. For training the model, an approach called *intermediate supervision* is used, which is presented in Section 4.1.4. Lastly, this thesis presents limitations of the authors work in Section 4.1.5, in addition to proposed experiments to overcome some of those limitations.

4.1.1 *Approach*

In [LPT18], the authors propose a network, which estimates pose and then uses the estimation, in combination with low level image features, to predict the action performed by a subject in a video clip. The network is completely differentiable, which allows the authors to pretrain certain parts, like the post estimator, and then fine-tune the network in an end-to-end fashion. The authors claim that the combination of a pose estimator and activity recognition is novel as pose estimators were not fully differentiable. Many pose estimators output a heatmap, like the Stacked Hourglass network [NYD16] (**Sec.** 3.1.2.3), for each joint position. Such a heatmap requires the use of the argmax function in a postprocessing step to compute the x and y image coordinates of the highest peak of the heatmap. Thus, the authors propose the *Soft-*

argmax function as an differentiable alternative to the regular argmax function (**Sec.** 4.1.2).

In addition, the authors propose methods to use both 3D and 2D pose datasets while training their network. However, these are not discussed here since this thesis focuses on 2D pose information.

4.1.2 *Soft-argmax*

The authors propose an approach for computing the x and y pixel coordinates from a joint heatmap using a method called *Soft-argmax* [LTP17]. An example of a joint heatmap can be seen in (**Fig.** 4.1.1). After computing the Softmax $\Phi(h)$ of a joint heatmap h, the x and y pixel coordinates of the highest image value need to be regressed. This coordinate represents the highest peak in the probability map computed using the softmax. This is achieved by computing the expectation in x and y direction on the probability map. In the implementation, the authors first define a fixed weight matrix for a convolutional layer in the following way:

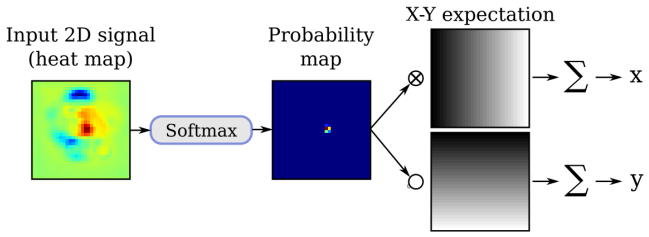


Figure 4.1.1: The approach presented by the authors to regress x and y coordinates from heatmaps. First, the authors in [LPT18] apply a Softmax. Then, they compute the expectation in both x and y direction using a *2D ramp function*. Summation of the result leads to the regressed coordinates. Image taken from [LPT18].

$$\boldsymbol{W}_{i,j,x} = \frac{i}{W}, \; \boldsymbol{W}_{i,j,y} = \frac{j}{H}, \tag{4.1.1}$$

where H, W refer to the width and height of the heatmap and i, j are the coordinates for each element in the heatmap. This leads to ramp functions for the horizontal and vertical dimensions, which are visualized in (**Fig.** 4.1.1). By convolving a part heatmap with both $\boldsymbol{W}_x$ and $\boldsymbol{W}_y$ the expectations are computed, leading to the regressed coordinate $(\psi_x(h), \psi_y(h)) = (x_{exp}, y_{exp})$ with $x_{exp}, y_{exp} \in [0, 1]$ (see (**Eq.** 4.1.2)). The

regressed coordinates need to be multiplied with the width and height to obtain integer coordinates.

$$\psi_d(h) = \sum_{i=1}^{W} \sum_{j=1}^{H} \boldsymbol{W}_{i,j,d} \Phi(h_{i,j}) \tag{4.1.2}$$

The authors further prove that the Soft-argmax function is differentiable by providing the derivative of $\psi_d(h)$:

$$\frac{\partial \psi_d(h_{i,j})}{\partial h_{i,j}} = \boldsymbol{W}_{i,j,d} \frac{exp(h_{i,j})(\sum_{k=1}^{W} \sum_{l=1}^{H} exp(h_{k,l} - exp(h_{i,j}))}{(\sum_{k=1}^{W} \sum_{l=1}^{H} exp(h_{k,l}))^2} \tag{4.1.3}$$

Further, the authors argue that such a method for regressing x and y coordinates is more accurate and it requires fewer model weights than directly regressing them using, for example, a fully-connected layer [LTP17]. As proof, they compare their methods to other state-of-the-art methods that directly regress the coordinates. The authors observe that their approach consistently outperform approaches regression approaches, such as [CAFM16] and [SSLW17], on the MPII benchmark. Specifically, when comparing to the previously best approach for regressing the pose by [SSLW17], the authors observe an overall increase in PCKh accuracy of 5.2 percentage points from 86.4 to 91.2 percent.

4.1.3 *Architecture*

The architecture used by the authors can be divided into several blocks. First, a feature extraction block, referred to as *Stem*, is used to extract visual features from each frame of the input video separately. These features are used to predict the joint heatmaps in the pose estimation block. The heatmaps are used to compute the x and y coordinates of each joint position using the Soft-argmax function. Second, the estimated poses are used to predict the action performed in the video clip. Third, the computed image features from the Stem block are similarly used for a separate action prediction. Both predictions are combined to form the final action prediction of the network. Throughout the network, the authors use the ReLU activation function if not otherwise stated. See (**Fig.** 4.1.2) for a visualization of the network and its separate components.

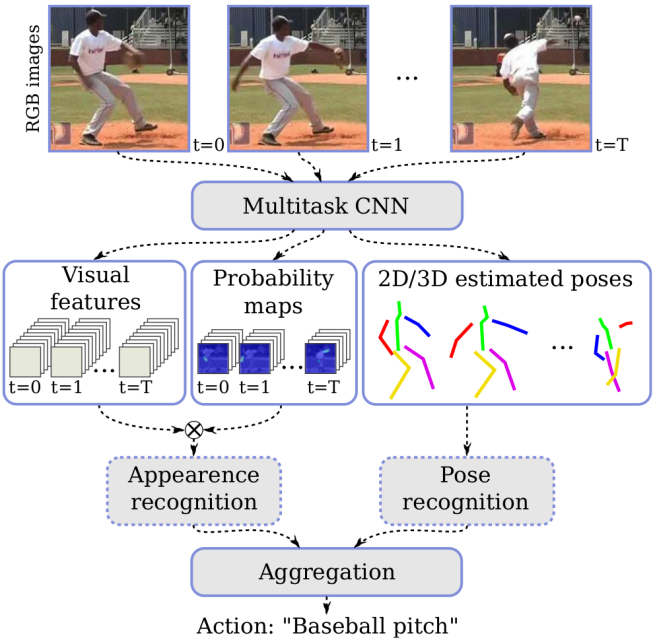


Figure 4.1.2: High level visualization of the network used by [LPT18]. The images features and poses are computed frame-by-frame. The output is then passed to the appearance and pose recognition subnetworks, where they are jointly processed to predict the action of the video clip. Image taken from [LPT18].

4.1.3.1 *Feature extraction (Stem)*

The authors base their feature extraction network on the *Inception v4* network proposed by [SIVA17]. The last convolutional layer of the Inception v4 network was replaced by a *depthwise separable convolutional layer*, which will be explained in the next section. The authors do not use pretrained weights for the Stem feature extractor, which is a method often used to reduce training time of the network. A visualization of the Stem network is shown in (**Fig.** 4.1.4a).

4.1.3.2 *Depthwise separable convolution*

A *depthwise separable convolution* [Sif14] [Cho17] is used to reduce the number of parameters and matrix multiplications needed for convolutional layers with many channels. Consider an input matrix to a convolutional layer of size $n \times n \times p$, where p indicates the number of channels. Without loss of generality, a square input as well as a square kernel size is assumed. If a regular convolutional layer is used, and the

desired number of output channels is given by m with $m \gg p$, then one approach is to use m kernels in the convolutional layer of size $a \times a \times p$. These result in $a^2 * p * m$ parameters of the convolutional layer that need to be learned. In a depthwise separable convolutional layer, two convolutional layers are used sequentially, referred to as the *depthwise convolutional layer* and *pointwise convolutional layer*. The *depthwise convolutional layer* consists of p filters of size $a \times a \times 1$. Each filter processes one channel of the input matrix. The resulting feature map is fed through the *pointwise convolutional layer*, which consists of m filters of size $1 \times 1 \times p$. See (**Fig.** 4.1.3) for a visualization. Using a depthwise separable convolutional layer results in

$$p \cdot m + p \cdot a^2 \Leftrightarrow p \cdot (m + a^2)$$

learnable parameters. It is then easy to see that

$$p \cdot (m + a^2) \ll p \cdot m \cdot a^2.$$

Additionally, the number of matrix multiplications needed to compute the convolution is reduced significantly.

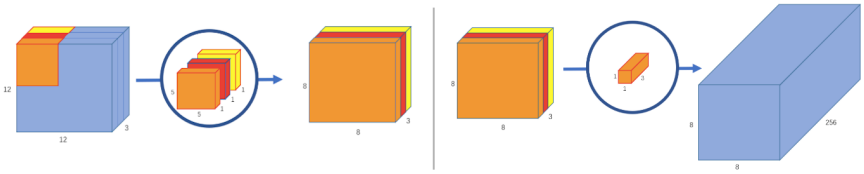
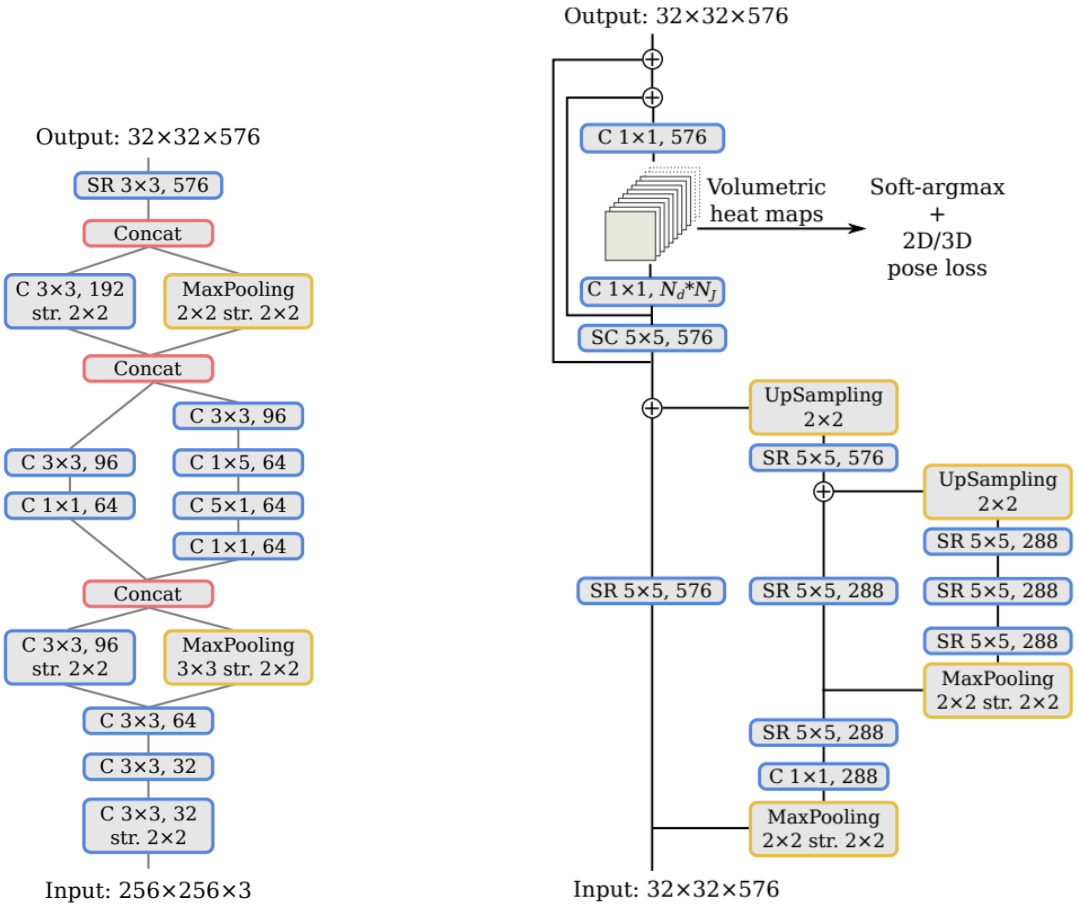


Figure 4.1.3: Visualization of a depthwise separable convolutional layer. **Left:** The depthwise layer consists of p kernels of size $a \times a \times 1$. Each kernel processes one channel of the input image. **Right:** The pointwise convolutional layer takes the output of the depthwise layer as input and consists of m filters of size $1 \times 1 \times p$ to compute the desired output feature map. Image taken from [Wan18].

4.1.3.3 *Pose estimation*

For the pose estimation network, the authors build upon the work by [NYD16], presented in (**Sec.** 3.1.2.3). Their pose estimator is constructed from multiple *prediction blocks*, which are visualized in (**Fig.** 4.1.4b). Similar to the stacked hourglass architecture, the prediction block computes features on different scales of the input by utilizing *Max-pooling* and *Upsampling* layers. In particular, the authors use *separable residual blocks* in place of regular convolutions to extract these scale-dependant features.



(a) *Stem* network.

(b) *Prediction block* network

Figure 4.1.4: **Left:** The feature extraction network used in [LPT18], also referred to as *Stem*. Each RGB frame of the video is resized to $256 \times 256 \times 3$. *SR* refers to a *depthwise separable convolution*. *C* refers to a regular convolutional layer. The extracted features are of size $32 \times 32 \times 576$. **Right:** One *prediction block* used in the pose estimation network in [LPT18]. Notice that the input and output dimensionality are identical, which allows the authors to connect multiple blocks together. *C* and *SC* refer to regular convolutional layers and depthwise separable convolutional layers, respectively. *SR* refers to a *Separable residual block*. The estimated pose of each prediction block can then also be used for intermediate supervision (**Sec.** 4.1.4). Images taken from [LPT18] supplementary material.

A *separable residual block* is defined by the authors in their previous work [LTP17] as a depthwise separable convolutional layer, whose input is added to the output using a residual connection (**Sec.** 3.1.2.3). For the case where the number of output channels p_{out} is different to the number of input channels p_{in}, the authors add a convolutional layer with p_{out} kernels of size $1 \times 1 \times p_{in}$ to ensure that the addition of the input with the output of the depthwise separable convolutional layer is possible.

After the extraction of features on different scales, the features are passed to a convolutional layer with $N_j + N_j * N_c$ filters of size $1 \times 1 \times 576$. N_j refers to the number of desired heatmaps, i.e., the number of joints that need to be detected. N_c is defined by the authors as the number of additional heatmaps per joint heatmap, called *context heatmaps*. The Soft-argmax computes the predicted joint locations on all heatmaps and aggregates the results for each joint using (**Eq.** 4.1.4). There, $\alpha \in [0, 1]$ determines how much influence the context heatmaps $(\boldsymbol{h}_{1,n}, \ldots, \boldsymbol{h}_{N_c,n}$ should have on the prediction of the joint $n \in [1, N_j]$. SA refers to the Soft-argmax function, computing joint position estimates from heatmaps. Additionally, $(\boldsymbol{p}_{1,n}, \ldots, \boldsymbol{p}_{i,n}, \ldots, \boldsymbol{p}_{N_c,n}$ refers to the predicted visibility of joint n in context heatmap i, which the authors estimate by using Max-pooling on heatmap $\boldsymbol{h}_n$, followed by a sigmoid activation function. In [LTP17], the authors present this approach and argue that, for the final joint location prediction, this approach leads to more accurate results, without specific evaluation of accuracy with and without context heatmaps.

$$y_n = \alpha \cdot SA(\boldsymbol{h}_n) + (1 - \alpha) \frac{\sum_{i=1}^{N_c} \boldsymbol{p}_{i,n} \cdot SA(\boldsymbol{h}_{i,n})}{\sum_{i=1}^{N_c} \boldsymbol{p}_{i,n}}. \tag{4.1.4}$$

Similar to the stacked hourglass approach by [NYD16], the prediction block is designed in a way that the input and output dimensions are identical, which allows the authors to connect multiple prediction blocks. When training the pose estimator for evaluating its accuracy, the authors use 8 prediction blocks, whereas a smaller pose estimator of 4 blocks is used when incorporating it into the human activity recognition pipeline. The authors do not motivate their choices or evaluate the accuracy of different number of prediction blocks, which is why we chose to evaluate the accuracy experimentally in (**Sec.** 5.3.2).

4.1.3.4 *Pose Model*

After computing image features in the *Stem* network and feeding them into the pose estimator, the authors decide to split their network into two parts for estimating the performed action. The first path, further referred to as *Pose Model*, aggregates the

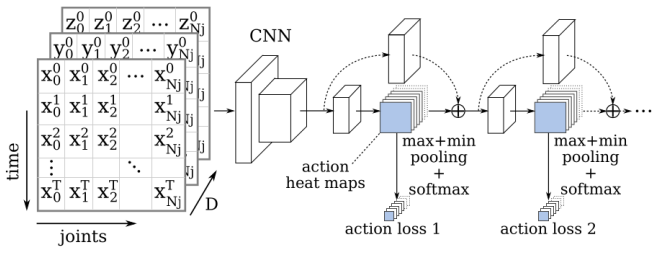


Figure 4.1.5: The approach presented by [LPT18] to aggregate the estimated pose for each frame and use it to estimate the performed action. The chosen pose representation is also referred to as a *pose cube*, since represents a three dimensional matrix. Notice that the pipeline produces intermediate action recognition results, which can be used for intermediate supervision (**Sec.** 4.1.4). Image taken from [LPT18].

predicted x and y coordinates of the joints for each frame of the input video into a three-dimensional matrix representation of size $a \times b \times z$, also referred to as the *pose cube*. b refers to the number of frames of the input clip, while a determines the number of joints to detect. To ensure that the pose matrix has identical dimensionality, regardless of the original input video length, the authors decide to subdivide the video into chunks of 16 frames, resulting in $b = 16$. The last dimension contains the x and y coordinates, respectively, resulting in $z = 2$ and a total dimensionality of $16 \times 16 \times 2$. Next, the pose cube is processed by a fully-convolutional neural network, which can be seen in (**Fig.** 4.1.6), before being fed into a so called *action prediction block*. The authors decide to use a function called *MaxPlusMin pooling* instead of regular *Max-pooling*. This function is implemented by the authors using a *Max-pooling* operation with kernel size (4×4) in the following way:

$$\mathrm{MaxMinPooling}(x) = \mathrm{MaxPool}(x) - \mathrm{MaxPool}(-x). \tag{4.1.5}$$

However, the authors do not explain the benefit of such a pooling operation over a regular *Max-pooling* layer.

The purpose of the fully-convolutional block is to extract features from the pose matrix. A processing block called *action prediction block* is used to classify the action performed, based on the features computed from either the pose or the appearance cube.

The *action prediction block* is similar to the *prediction block* used in the pose estimator, and follows the ideas of stacking, since the input and the output dimensions of the action prediction block are identical. As an intermediate representation, *action heatmaps*

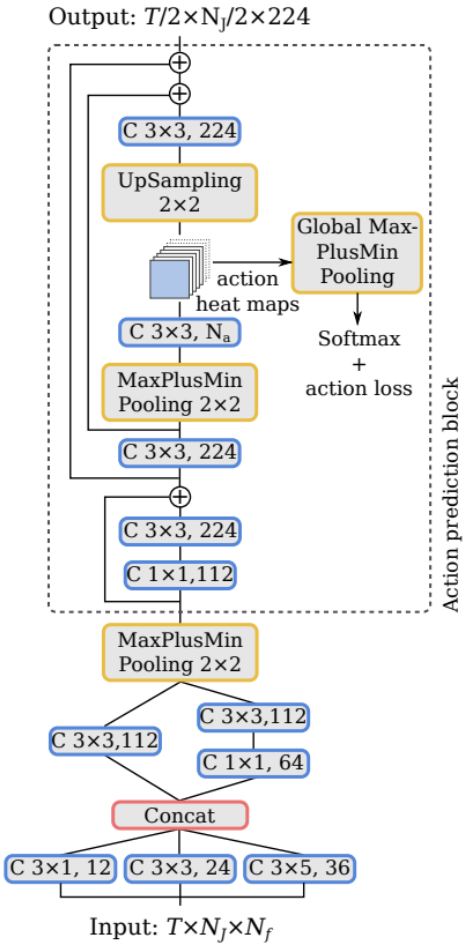


Figure 4.1.6: The fully-convolutional network used to process the *pose cube*, followed by a single *action prediction block*. T refers to the number of frames, N_j is defined as the number of joints and N_f represents the third dimension, i.e., $N_f = 2$ in the case of the pose cube. In the action prediction block, the number of actions to detect can be set using N_a. Image taken from [LPT18] supplementary material.

are created, which are in turn used in training and to predict the action performed in the video clip using the Softmax activation function for evaluation. Notice that the output of the action prediction block is not the predicted action itself, but rather features of the same dimensionality as the input to the block. The predicted actions are only used for computing the loss during training and for evaluation.

4.1.3.5 *Visual Model*

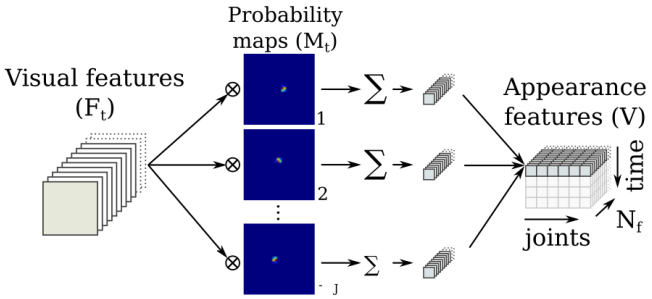


Figure 4.1.7: The approach proposed in [LPT18] to aggregate image features for multiple frames into a *appearance cube*. This cube can then be used for further processing, similar to the *pose cube*. Image taken from [LPT18].

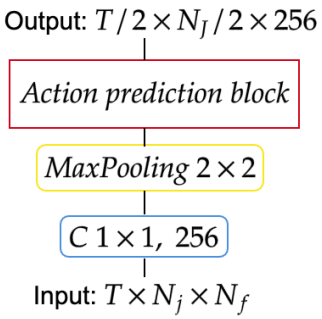


Figure 4.1.8: The *Visual Model*, according to the code provided as supplementary material to [LPT18]. The architecture differs to the one presented in the paper, where the authors claim that both *Visual Model* and *Pose Model* have identical architectures.

The authors also incorporate low level image features from the *Stem* block into their action recognition approach, in addition to the estimated pose information. Thus, they extract visual features for each frame and aggregate them in a matrix, which is used to predict the action performed in the video clip. This matrix is referred to as the *appearance cube*. The computed features f_d for a frame d have the dimensionality $32 \times 32 \times 576$. The authors multiply f_d with each computed joint heatmap, after the heatmaps were further processed by applying the Softmax activation function. They do not multiply the features with the context heatmaps, but only with the N_j joint heatmaps. The resulting features are summed over the first two dimensions, resulting

in a new feature of size $1 \times 1 \times 576$. Then, these new features are aggregated into the appearance cube, resulting in a cube of dimensionality $16 \times 16 \times 576$. See (**Fig.** 4.1.7) for a visualization of this process. The authors use a different feature extraction step as in the *Pose Model*. The feature extraction consists of a single convolutional layer, followed by a batch normalization and a *Max-pooling* layer. In [LPT18], the authors claim that they use a similar architecture to the *Pose Model*. However, when examining the code, published as supplementary material, we found that they actually use the shallow model mentioned before. A visualization of the shallow feature extractor can be seen in (**Fig.** 4.1.8). The action prediction block is identical to the one presented in the *Pose Model* (see (**Fig.** 4.1.6)).

Recall that each action prediction block computes action heatmaps, which are used in training by passing them through a *Max-pooling* layer, followed by applying the Softmax activation function. For combining both the predictions based on the pose cube and appearance cube, the authors take the output of the last action prediction block in both network paths, add the action heatmaps and apply a *Max-Pooling* and Softmax activation to form the final action classification of the video clip.

4.1.4 *Intermediate supervision*

One aspect of the training process is called *intermediate supervision*. Instead of only using the final output of the network to compute the error and start the backpropagation algorithm, multiple intermediate results are created, each of which can then be used to compute the error and start backpropagation. After each prediction block in the pose estimator, a intermediate prediction is performed by applying Soft-argmax to the joint heatmaps created by that prediction block. Similarly, in each action prediction block, intermediate results are created by performing pooling and Softmax to the action heatmaps, which can be seen in (**Fig.** 4.1.5). The authors claim that this approach increases the accuracy of the overall network, since the network needs to be able to predict the action as early and as accurately as possible. However, they do not provide evidence to proof their claim.

4.1.5 *Limitations*

There are four limitations to the approach proposed by the authors and their evaluation of the method.

First, the authors evaluate their model for 2D action recognition solely on the Penn Action dataset. While they do not focus exclusively on 2D action recognition in

their paper, a comparison to state-of-the-art results on other complex datasets such as JHMDB would be desirable. This is especially true considering that Penn Action focuses on 15 different actions, which are predominantly from the sports domain, while JHMDB contains 21 classes, including every day activity such as *brushing hair* and *climb stairs*. Thus, the JHMDB dataset is a more complex dataset in terms of action diversity. We thus argue that evaluation on the JHMDB dataset will lead to valuable insight into the effectiveness of the authors approach outside of the domain of sports action recognition.

Second, they argue that their approach for training the action recognition network constitutes end-to-end learning. This would imply that the network weights are not pretrained. However, as discussed earlier, the pose estimator is pretrained. While this approach leads to competitive results, it is misleading to call it end-to-end training. Later in their paper, the authors refer to their approach as fine-tuning on a pretrained model, which is a more accurate description of the process.

Third, certain choices of hyperparameters used for configuring and training the model are not evaluated in the original work. For example, the authors decided to utilize context heatmaps (as discussed in (**Sec.** 4.1.3.3)) as well as setting the number of prediction blocks of their pose estimator to 8. They do not provide explanations or evaluations as to why these decisions were made and thus further analysis will give a better insight into the differences these choices make, considering the overall performance of the model.

Fourth, the pose estimator processes the input videos frame by frame. This means that temporal information, which could aid in pose estimation, is not utilized for training. As an example, [GGT+18] adopt a frame by frame approach to video data by incorporating 3D convolutional layer to capture temporal information. They show an increase in accuracy when comparing predicting the pose for each frame individually, suggesting that capturing the temporal dimension can aid in learning pose in videos.

4.2 PROPOSED EXPERIMENTS

Apart from recreating some key findings of the authors and evaluating certain decisions by the authors, we propose the following extensions to the network architecture and learning process.

First, we propose to combine the losses of the pose estimator with the losses of the overall action recognition pipeline in a similar manner to the intermediate supervision approach. A combination of the loss function allows for the network to be trained in an end-to-end fashion, as opposed to fine-tuning the network after pretraining

certain components. This will give an insight not only into the feasibility of the end-to-end training approach but also on the results regarding the pose estimator. [IGG16] argue that pose estimation accuracy increases when given an action prior. Also, in the approaches discussed earlier (**Sec.** 3.2.3), using pose information for action recognition significantly increases the overall accuracy. Thus, we propose that, by using an end-to-end learning approach, both pose estimation and action recognition networks could benefit from each other when learned jointly.

Second, as explained before, evaluating the methods on a more complex video dataset is desirable. Thus, in this thesis, it was decided to train and evaluate the network on the JHMDB dataset. Even though this dataset is smaller in overall number of images, it contains more diverse classes.

Third, the authors presented the Soft-argmax approach in [LTP17] without evaluating the accuracy of the approach. Thus, this thesis quantitatively and qualitatively evaluates the accuracy by using synthetic test images derived from the ground truth pose positions. These experiments will give an insight into whether or not the Soft-argmax is a suitable approach for computing the joint positions from heatmaps, when compared to using the argmax in a postprocessing step.

5 EXPERIMENTS

In the previous chapter, the method for Human Activity Recognition by [LPT18] was introduced by explaining the model architectures and the Soft-argmax function proposed in [LTP17]. Afterwards, this thesis proposed experiments to gain a better insight into the authors method by evaluating the model for different hyperparameters, as well as to investigate whether end-to-end training is feasible. First, this chapter introduces the datasets in Section 5.1, which consist of the *MPII Human Pose* dataset for 2D pose estimation and the *Penn Action* and *JHMDB* datasets for Human Activity Recognition. Second, the metrics used in the authors work, as well as the experiments done in this thesis, are discussed in Section 5.2. Third, this chapter discusses the experiment setups, as well as the results, in Section 5.3. The pose estimation and 2D HAR experiments from [LPT18] are recreated first. Further experimentation towards understanding the capabilities of the model are presented, including a qualitative and quantitative evaluation of the Soft-argmax function, the accuracy of the authors model achieved on the complex JHMDB dataset, as well as an approach for training the model in an end-to-end approach, without using a pretrained pose estimator.

5.1 DATASETS

In the following section, three datasets are introduced. The MPII Human Pose and Penn Action datasets are used to recreate the authors work. The JHMDB dataset, which contains video clips with pose annotations for each frame, is used to evaluate how the model proposed in [LPT18] adapts to a more challenging video dataset.

5.1.1 *MPII Human Pose*

In [APGS14], the authors present a dataset for estimating two dimensional human pose on image data. It contains pose annotations for 40.000 persons in 25.000 images. The annotations include 16 joint positions and an indicator of whether or not the joint is occluded or not. The 16 joints are *left / right ankle*, *left / right knee*, *left / right hip*, *left / right elbow*, *left / right shoulder*, *left / right wrist*, *pelvis*, *thorax*, *upper neck* and *top of the head*. See (**Fig.** 5.1.1) for example images of the dataset. In addition, the body

center coordinates, as well a scale indicating the size of the person bounding box are given by the dataset. The scale is given with regards to a 200 pixel bounding box, meaning that the bounding box side lengths can be computed by multiplying the scale with 200. Also, a bounding box of the head is given, which is used to compute the *PCKh* metric (see (**Sec.** 5.2.2)). The images are extracted from YouTube videos and they do not contain artifacts commonly found in videos, e.g., compression or blur. Additionally, each image is assigned with an activity, totalling 401 activities. However, these annotations are not used for Human Action Recognition since the number of samples per activity is too low for training a model as complex as the HAR pipeline presented in (**Sec.** 4.1.1).

Figure 5.1.1: Four example images from the MPII dataset [APGS14].

We follow the approaches from [LPT18] for preprocessing. First, a person bounding box is estimated using the center body annotation from the dataset. The authors multiply the scale given by the annotation s_{orig} by 1.25, resulting in s_{new}. They do not motivate the reason for using this specific value, but it most likely was used to enlarge the bounding box to contain more context around the person. Next, they compute the width and height using $s_{new} \cdot 200$, since the scale parameter in the dataset is given w.r.t. a 200×200 pixel bounding box. In addition, the authors also alter the center position (c_x, c_y) by computing $(c_x^{new}, c_y^{new}) = (c_x, c_y + s_{new} \cdot 12)$. Again, the authors do not provide a reasoning for increasing the center y position. Once the bounding box is computed, the image is cropped to the size of the bounding box around the newly computed center coordinate and rescaled to a size of 256×256. In the case where a joint annotation falls outside of the now cropped image, the authors set the visibility of the joint to 0 and set the (x, y) coordinates of the joint to $(-1e9, -1e9)$.

Additionally, the authors introduce parameters used for image augmentation by rotating, scaling and mirroring the image randomly. These values are sampled from their respective sets whenever augmentation is performed. Specifically, they introduce $s_{aug} \in \{0.7, 1, 1.3\}$, which gets multiplied with s_{new} computed earlier. This results in zooming into or out of the image by 30% when $s_{aug} \neq 1$. Also, $r_{aug} \in \{-40, -35, \ldots, 35, 40\}$ is introduced to rotate the image r_{aug} degrees around its center, possibly introducing black borders around the image. Moreover, when augmenting, the image is horizontally mirrored with a chance of 50 percent. See (**Fig.** 5.1.2) for examples of different augmented images, including augmented pose.

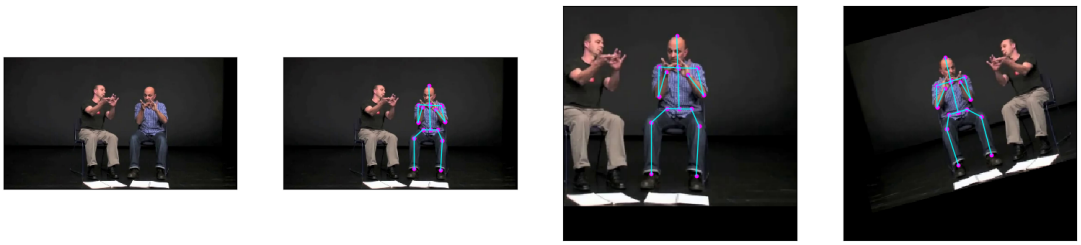


Figure 5.1.2: **From left to right**: **1.** Original image from the MPII dataset. **2.** Original image with the ground truth pose superimposed. **3.** Image after estimating the bounding box, cropping and rescaling. **4.** Augmented image using $s_{aug} = 1.3$, $r_{aug} = -15$ degrees and flipping the image horizontally.

The dataset does not contain annotations for the test data, other than the scale and center coordinates. To evaluate the test images, the joints need to be evaluated and the results need to be send to the authors of the dataset at the *Max Planck Institute for Intelligent Systems* for comparison to the ground truth pose. This ensures that the test data annotations are not mistakenly or maliciously used in training. For all datasets used in the experiments, 10 percent of the training datapoints are withheld from training and used as validation data.

5.1.2 *Penn Action*

Another dataset used in [LPT18] is the Penn Action dataset [ZZD13]. It contains 2326 video clips of 15 different, resulting in 149.355 frames. The actions are mostly sports related and they include *baseball swing*, *clean and jerk*, *jumping jacks*, *pushup*, *strum guitar*, *bench press*, *golf swing*, *baseball pitch*, *situp*, *tennis forehand*, *bowling*, *jump rope*, *pullup*, *squat* and *tennis serve*. Additionally, the authors provide annotations for 13 body joints,

including *left and right shoulders, elbows, wrists, hips and knees* and *head*. Some example images can be seen in (**Fig.** 5.1.3).

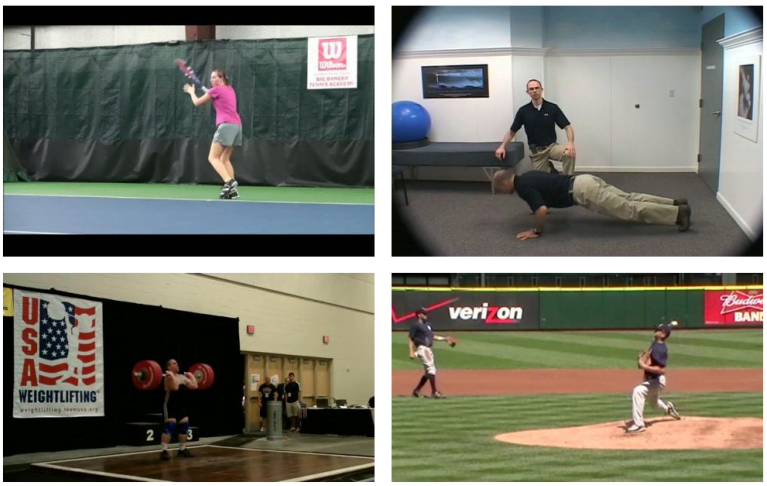


Figure 5.1.3: Four example images from the Penn Action dataset [ZZD13].

It was decided by [LPT18] that the number of joints should be identical to the ones presented in [APGS14] (**Sec.** 5.1.1). This is important, because the network architecture is designed in a way that assumes 16 joints per video frame because the estimated poses for each frame of a video clip are aggregated in a fixed size *pose cube* representation (see (**Sec.** 4.1.3.4)). To achieve this, the *head* annotation of the Penn Action dataset is mapped to the *upper neck* joint of the MPII dataset and the missing joints were interpreted as not visible.

Additional augmentation is introduced to make the training process more robust. Two additional augmentation methods are used. First, salt and pepper augmentation is applied to the training images with a probability of 0.5. In this augmentation method, each pixel is set to either black or white with probability $p_{sp} = 0.02$. Second, black rectangles are placed randomly inside the image, occluding parts of the original image. The network needs to make decisions on partially occluded images, making it more robust, because it forces the network to utilize alternative paths on some images, effectively making the model more general. Dropout is applied with a probability of 0.5 as well.

For some experiments, ground truth bounding boxes of the person are calculated by taking the minimum and maximum of the x and y coordinates from the ground truth pose and defining these as the corners of the bounding box. Additionally, to include more context around the person, the bounding box is increased in both

dimensions. This was done to add more context around the subject, but also because the Soft-argmax function is unprecise around the image borders (see (**Sec.** 5.3.1)). For high uncertainty of the pose location, the Soft-argmax function is unprecise for approximately 20 pixels around the image border (see (**Fig.** 3.1.8)). Thus, the amount of context added is set to 30 pixels to make certain that the Soft-argmax function does not wrongly estimates the pose position.

Additionally, the authors in [LPT18] decide to process the video clips in chunks of 16 frames. This is important since the network architecture assumes a specific dimensionality for the *pose cube* (see (**Sec.** 4.1.3.4)). These chunks are further referred to as *fragments*.

5.1.3 *JHMDB*

Similar to [ZZD13] (**Sec.** 5.1.2), the JHMDB dataset [JGZ+13] contains annotations for the pose and the action in video clips. The JHMDB dataset contains 928 video clips, totalling 31.838. The dataset was created by taking a subset of the HMDB action recognition dataset [KJG+11], which was annotated using the *puppet tool* [ZFB12]. This tool allows to not only annotate the pose of the person but also automatically computes a binary segmentation map of the person, further referred to as the *puppet mask*. See (**Fig.** 5.1.4) for a visualization of the annotation process and the puppet tool.

The clips in the HMDB dataset are taken from YouTube. The annotated subset contains the following actions: *brush hair*, *catch*, *clap*, *climb stair*, *golf*, *jump*, *kick ball*, *pick*, *pour*, *pullup*, *push*, *run*, *shoot ball*, *shoot bow*, *shoot gun*, *sit*, *stand*, *swing baseball*, *throw*, *walk* and *wave*. Some example images of the dataset can be seen in (**Fig.** 5.1.5). Notice that the actions are more diverse, in comparison to the Penn Action dataset, specifically because it also contains non-sport activities.

The puppet tool defines 15 different joints. They are identical to the ones annotated in the Penn Action dataset, but they additionally contain the *neck* and the *belly*. Also, the approaches for computing ground truth bounding boxes, preprocessing and augmenting the *fragments* are identical to the ones presented earlier in (**Sec.** 5.1.2).

5.2 EVALUATION METRICS

In this section, the *PCK* and *PCKh* metrics are presented, which are used to evaluate the accuracy of the predicted pose in comparison to the ground truth annotation. Also, two approaches for assessing the accuracy in video clips, *Single-* and *Multi-Clip accuracy*, are presented.

Figure 5.1.4: Example of an image, annotated using the puppet flow. The dots indicate the joint positions, while the transparent human figure prior automatically adjusts, yielding the human segmentation map. Notice that, by using the figure, even joints that are not visible (like the ankles) still get anatomically plausible annotations. Image taken from [Max].

5.2.1 *PCK*

The *Probability of Correct Keypoints (PCK)* metric [FMJZ08] is often used in the literature to evaluate an estimated pose when a human bounding box is given. See (**Sec.** 3.1.1.2) for a motivation of this metric. Let $k_{est} = (x_{est}, y_{est})$ be a predicted joint location and let $k_{gt} = (x_{gt}, y_{gt})$ be the corresponding ground truth joint location. First, the absolute distance $d = \|k_{est} - k_{gt}\|$ is computed. Second, the maximum of the bounding box side lengths $l_{max} = max(bbox_{height}, bbox_{width})$ is computed and multiplied by a hyperparameter α, resulting in $l_{comp} = l_{max} \cdot \alpha$. Then, an estimated joint location is determined to be estimated correctly if $d < l_{comp}$. The hyperparameter α is a threshold, which determines how close the predicted joint location has to be to the ground truth joint location in order to be classified as correctly estimated. Typical values for α found in the literature are 0.2 and 0.1. This metric is used primarily for evaluating the pose on the JHMDB dataset since the dataset does not provide head bounding box annotations, which are necessary for using the PCKh metric.

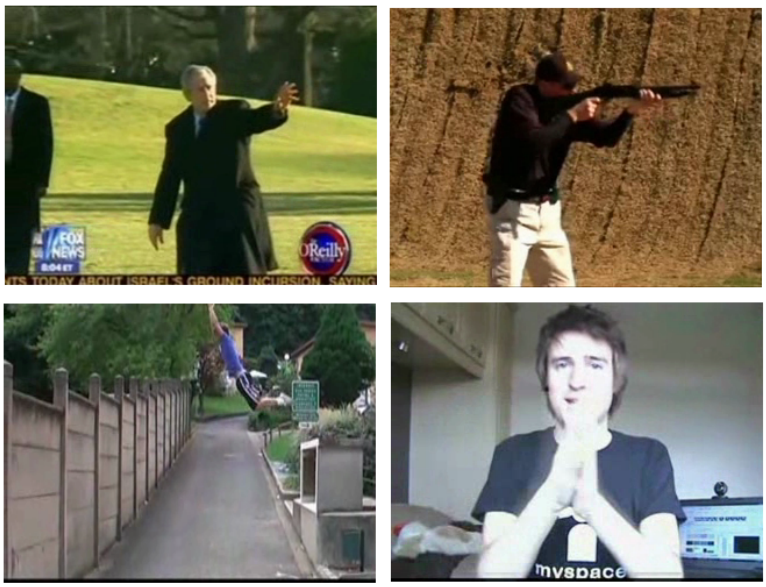


Figure 5.1.5: Four example images from the JHMDB dataset [JGZ+13].

To compensate for the lack of head annotation, this thesis proposes to use an additional metric for evaluating predicted poses, referred to as *PCKu*. Instead of using the bounding box side lengths to determine l_{comp}, the distance between the neck joint position and pelvis joint position is used. While this is not as invariant to different actions and poses as PCKh, it is more robust in comparison to traditional PCK. This metric is computed in addition to PCK for datasets which do not provide a head bounding box annotation.

5.2.2 *PCKh*

One downside of using the PCK metric based on person bounding box side lengths is that poses where one or more limbs are stretched out far from the subjects torso the bounding box is not an accurate representation of the body size. Thus, the threshold l_{comp} highly depends on the pose of the person, which can lead to a higher threshold, depending on the action. Consider the difference in subject bounding box size between a person sitting on a chair versus a person jumping in the air while spreading their arms.

In [APGS14], the authors propose the head bounding box diameter, assuming that this bounding box does not change for different poses. See (**Sec.** 3.1.2.2) for a

motivation of the metric. Similar to PCK, PCKh also defines a threshold parameter α. Typical values for α found in the literature are 0.5 and 0.2.

5.2.3 *Single- and Multi-Clip Accuracy*

To evaluate the accuracy of the HAR pipeline, the authors in [LPT18] use two different approaches. First, they take the video to evaluate and extract a 16 frame long clip from the middle of the video. Second, they classify the action of that 16 frame clip. Since the output of the network is a Softmax activation, they use the *argmax* function to determine the action with the highest score. This is referred to further as *Single-Clip accuracy*.

Additionally, the authors first extract multiple 16 frame long clips from the video by starting at frame 0 and then incrementing the starting position by 8 for as long as there are at least 16 frames left in the clip. They do not motivate their choice of incrementing the starting position by 8. Second, for each clip, they predict the action. Third, they determine the class most often predicted for the set of clips, which is then used to compare to the ground truth class of the video. For example, if 4 clips are extracted from the video using the method presented above, and 3 of these clips are classified as *shooting bow*, then *shooting bow* is used for comparison to the ground truth action of the video. The accuracy of this method is referred to as *Multi-Clip accuracy*.

5.3 EXPERIMENTAL RESULTS

This section presents the results of the experiments conducted in this thesis. First, an evaluation of the accuracy of the Soft-argmax function is presented. Second, this thesis replicates the work of [LPT18] by recreating the pose estimation and 2D HAR experiments. Third, the pose estimator from [LPT18] is evaluated against the JHMDB dataset. Fourth, the HAR model is trained and evaluated using data from the JHMDB dataset. Fifth, we present the results of training the HAR model in an end-to-end approach without pretraining the pose estimator weights.

5.3.1 *Accuracy of Soft-argmax function*

In [LTP17], the authors propose a method for finding pixel coordinates of the maximum pixel value in an image, which they refer to as *Soft-argmax* (see (**Sec.** 4.1.2)). They argue that this method can be used instead of the argmax function, which is often used for the same purpose. This suggests that the Soft-argmax should be as accurate

in determining the maximum values as the argmax function. To investigate this assumption, this thesis performs two experiments.

In the first experiment, we analyze the accuracy quantitatively on data from the MPII dataset. For each joint position (i, j) of a pose, a synthetic joint heatmap is generated by placing a two-dimensional gaussian with mean (i, j) and covariance c. The Soft-argmax is applied to the heatmap, which results in estimates (x_{est}, y_{est}) of the maximum. The distance from (x_{est}, y_{est}) to (i, j) is computed and an accuracy meassurement is proposed by the following formula:

$$f((x_{est}, y_{est}), (i, j), d) = \begin{cases} 1 & \text{if } |x_{est} - i| \leqslant d \text{ and } |y_{est} - j| \leqslant d \\ 0 & \text{otherwise} \end{cases} \tag{5.3.1}$$

In (**Eq.** 5.3.1), d refers to a threshold, which is necessary because of the way (x_{est}, y_{est}) is computed. The output of the Soft-argmax function are fractions of width and height with $(x_{frac}, y_{frac}) \in [0, 1]$. These fractions need to be multiplied by the width and height and they need to be rounded to the nearest integer afterwards to get valid image coordinates. This rounding process might introduce rounding errors. This thesis evaluates the accuracy of the Soft-argmax for different values of d on a subset of 1000 random images from the MPII dataset. Moreover, different covariance values $c \in \{1, 2, 5, 10, 20, 50\}$ are used to simulate different uncertainties in the heatmap. The results can be seen in (**Tab.** 5.3.1).

As can be seen in (**Tab.** 5.3.1), the Soft-argmax function is accurate in practice for a 2 pixel threshold. For higher values of c, the Soft-argmax becomes less accurate. This is to be expected, given that the heatmap is less certain of the true pixel coordinate. Notice that the accuracies are significantly lower for a threshold of $d = 1$. This contrast between $d = 1$ and $d = 2$ is most likely due to the rounding errors discussed earlier.

In the second experiment, a synthetic image of size 255×255 pixels is created, since this is the size of the input images to the network after preprocessing. For each (i, j) position of the image, a two-dimensional gaussian with mean (i, j) and covariance c is placed. Next, (x_{est}, y_{est}) are computed using the Soft-argmax function. Then, the estimations are compared to the ground truth mean value of the gaussian, i.e., (i, j), using the accuracy function defined in (**Eq.** 5.3.1) with $d = 2$ to allow for a small rounding error. This procedure is repeated for each (i, j) coordinate of the image. See (**Fig.** 5.3.1) for a visualization of the results, where violett pixels indicate a wrong prediction and yellow pixels indicate a correct prediction. As can be seen in the visualization, the Soft-argmax function accurately regresses the gaussian mean values for small covariances c for most of the image coordinates. As the covariance

threshold	$c = 1$	$c = 2$	$c = 5$	$c = 10$	$c = 20$	$c = 50$
1	93.137	93.123	93.069	92.988	92.811	92.201
2	99.952	99.931	99.843	99.741	99.469	98.598
3	99.959	99.945	99.891	99.809	99.639	99.054
4	99.959	99.952	99.911	99.829	99.700	99.197

Table 5.3.1: Mean average accuracy (in percent) of Soft-argmax when detecting ground truth coordinates from synthetic joint heatmaps. Threshold refers to the amount of pixels the estimate is allowed to deviate from the ground truth annotation. c refers to the covariance used for creating the synthetic heatmaps. The large discrepancy between a threshold of 1 and a threshold of 2 is most likely due to rounding errors.

increases, the accuracy decreases, especially around the borders of the image. While this behaviour indicates that the Soft-argmax function is less accurate around the borders of the image, this does not negatively effect the accuracy when estimating actual pose coordinates, according to the quantitative experiment.

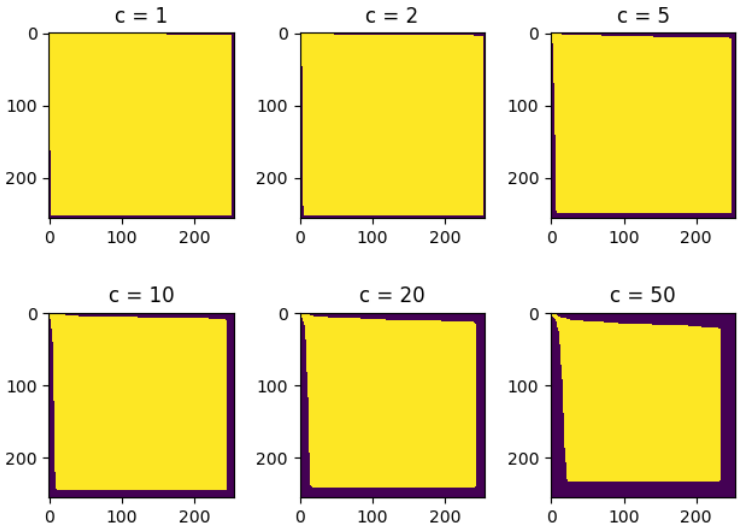


Figure 5.3.1: Evaluation of the accuracy of the Soft-argmax function using synthetic data. Yellow pixels (i, j) indicate where the Soft-argmax function correctly regressed the peak of the gaussian with mean value (i, j), while violet indicates wrong predictions. Notice that the accuracy decreases when approaching the border of the image and when the covariance is increasing.

5.3.2 *Replication of Original Work*

First, to recreate the work of [LPT18], a pose estimator using the MPII dataset is trained. In addition to the authors work, the number of prediction blocks as well as the number of context heatmaps are varied to observe the effect on the overall accuracy of the model. Second, the 2D HAR experiments, performed by the authors on the Penn Action datasets, are recreated. The experiment settings are taken from [LPT18] as well as the supplemental material provided by the authors, if this thesis does not mention otherwise.

5.3.2.1 *Pose estimation*

In [LPT18], the authors use 8 prediction blocks in their pose estimator (see (**Sec.** 4.1.3.3)) for evaluating the accuracy of the model on the MPII dataset, using the PCKh metric (**Sec.** 5.2.2). In this thesis, additional experiments using 2 and 4 prediction blocks are performed. In addition, we evaluate all models without context heatmaps as well as with 2 context heatmaps, to gain an insight into how using context maps effects the model performance. From now on, this thesis uses the term *model configuration* when refering to models with different number of prediction blocks and context heatmaps.

The training data of the MPII dataset is split into a training and validation set for evaluating the model's accuracy on unseen data. An iteration is defined as the processing of one batch. The batch size varies, depending on the experiment, since the models are of different size. This means that, for smaller models, a bigger batch size is used since more data fits in GPU memory. The batch size is set to $45, 30$ and 20 for $2, 4$ and 8 prediction blocks, respectively. As an optimizer, the authors use the RMSProp algorithm. An initial learning rate of 10^{-3} is used in accordance to the work of the authors.

$$L_p = \frac{1}{N_j} \sum_{i=1}^{N_j} \|(p_{est} - p)\|_1 + \|p_{est} - p\|_2^2. \tag{5.3.2}$$

The authors define a loss function named *Elastic Net Loss*. For each joint in the pose, the L_1 and L_2 norms between the estimated and ground truth joint are computed and added. Then, these sums are again summed and normalized by the number of joints in the pose. See (**Eq.** 5.3.2), where p_{est} refers to the estimated joint position by the pose estimator and p refers to the ground truth pose. In addition to computing the

Elastic net loss, the authors predict the visibility of each joint. The annotations of MPII provide ground truth labels $v \in \{0, 1\}$ of whether a joint is occluded or not. The binary cross-entropy is used to compute a loss L_v between the predicted visibility vector v_{est} and the ground truth v. Finally, both losses are combined in the following way:

$$L = L_p + 0.01 \cdot L_v. \tag{5.3.3}$$

For all subsequent pose estimation experiments, L is used to compute the loss of the pose estimator.

As discussed before, for evaluating a model on the test dataset, the estimations need to be send to the *Max Planck Institute for Intelligent Systems* for evaluation. The amount of submissions is limit to 4. This means that it is not possible to evaluate all pose estimator configurations on the test dataset. Thus, this thesis chooses to report the validation accuracy for comparison between the different model configurations and only submit estimations from the best performing model.

The PCKh accuracies achieved on the validation dataset from the different model configurations are shown in (**Tab.** 5.3.2). For comparison, the authors report a validation accuracy of 89% for 8 prediction blocks and 2 context heatmaps. As can be seen in (**Tab.** 5.3.2), the higher the number of prediction blocks, the more accurate the model performs on the validation data. In addition, using context heatmaps appears to increase the accuracy only when using 8 blocks. To test whether or not the differences in accuracy between the configurations are statistically significant, this thesis performs randomization tests with a significance level of 0.05. For the randomization tests, 3000 permutations are used. As can be seen in (**Tab.** 5.3.2), the only statistically significant increases in accuracy are observed when comparing 2 and 8 prediction block models with and without context heatmaps. This suggests that a certain depth of network is necessary in order for context heatmaps to contribute significantly to the model performance.

Next, the best performing model (8 prediction blocks, 2 context heatmaps) is used to estimate the pose on the test dataset and the results were submitted to the *Max Planck Institute for Intelligent Systems*. The results can be seen in (**Tab.** 5.3.3), in direct comparison to the values reported by the authors. As can be seen, our model performs significantly worse in comparison to the reported values. One possible reason for this significant difference is the method of how the test poses are computed. The authors first estimate the pose for the test image. Afterwards, they obtain another pose estimation by mirroring the image horizontally, estimating the pose on the mirrored version and then mirror the estimated pose again. Moreover, they report that they

blocks	context	PCKh @ 0.5	p-values	blocks	context	PCKh @ 0.5	p-values
2	0	84.15	0.85	4	0	85.78	0.31
2	2	84.01		8	0	86.71	
4	0	85.78	0.85	4	2	85.64	0.12
4	2	85.64		8	2	**87.00**	
8	0	86.71	0.73	2	0	84.15	**0.006**
8	2	**87.00**		8	0	86.71	
2	0	84.15	0.074	2	2	84.01	**0.001**
4	0	85.78		8	2	**87.00**	
2	2	84.01	0.076	8	2	87.00	**0.025**
4	2	85.64		8*	2*	89.00*	

Table 5.3.2: Different model configurations and their corresponding PCKh validation score. The p-values were computed using a randomization test. p-values below the significance level of 0.05 are shown in bold, indicating that the change in accuracy is significant. Additionally, the highest accuracy is also shown in bold. The configuration marked with a star is the model from [LPT18]

	Head	Shoulder	Elbow	Wrist	Hip	Knee	Ankle	Total	p-value
our recreation	96.1	92.5	83.8	78.2	84.5	77.2	71.5	84.1	
[LPT18]	98.1	96.6	92.0	87.5	90.6	88.0	82.7	91.2	**0.0**

Table 5.3.3: PCKh test results of our recreation in direct comparison to the original work by [LPT18] using $\alpha = 0.5$. The change in total accuracy is statistically significant with a significance level of 0.05.

randomly displaced the bounding box, which is used for cropping the image around the subject. On these cropped image, the authors also estimate the pose. The final pose estimation is then computed by taking the average of each joint position per image over all computed poses. The authors do not describe the method of displacement in detail, which might cause the difference in test accuracy reported. Moreover, they report a mean validation accuracy over all joints of 89 percent, which is close to the 87 percent achieved by our model. In this thesis, it was decided to compute 4 displaced bounding boxes by moving the center randomly by $(x_d, y_d) \in [-8, \ldots, -3, 3, \ldots, 8]$. The results suggest that the displacement method the authors use for computing the test poses has a significant impact on the final score, since the discrepancy between our validation and test scores are significantly larger than the ones reported by the authors. Some example pose estimations on the validation dataset are given in (**Fig.** 5.3.2).

Figure 5.3.2: Example images from the validation dataset after training a pose estimator using 8 prediction blocks and using 2 context heatmaps. The joints on the right side of the body (from the subjects perspective) are shown in magenta, while the joints on the left side are shown in yellow. The remaining joints are shown in cyan. **From left to right**: **1.** The pose estimator was able to estimate the ground truth pose correctly. **2.** The pose estimator confused right and left joints. Also, the estimator was not able to estimate the right ankle joint position correctly and wrongly estimated the joint position at the left ankle. **3.** The belt of the subject was wrongly estimated to be the right wrist. **4.** The pose estimator was not able to estimate the subjects pose, except for the head joint. This is most likely due to the fact that the subject is underwater and because the subject is wearing a diving suit.

In conclusion, while the test accuracy scores are significantly lower in comparison to the scores provided by the authors, this is most likely due to the difference in

evaluation technique. Moreover, there were no statistically significant differences found between the different model configurations, except for the difference between the 2 and 8 prediction block models. Additionally, using context heatmaps does not significantly increase the accuracy of the pose estimator.

5.3.2.2 *HAR using Penn Action dataset*

To recreate the Human Activity Recognition results from [LPT18] on the Penn Action dataset, we first train a pose estimator. Once the pose estimator is trained, its pretrained weights are transferred to the HAR model. The authors used a hybrid dataset for training, consisting of 75% MPII and of 25% Penn Action training data. They do not, however, motivate this decision. The pose estimator contains 4 prediction blocks, uses 2 context heatmaps and it is trained using a batch size of 30 and a learning rate of 10^{-3}.

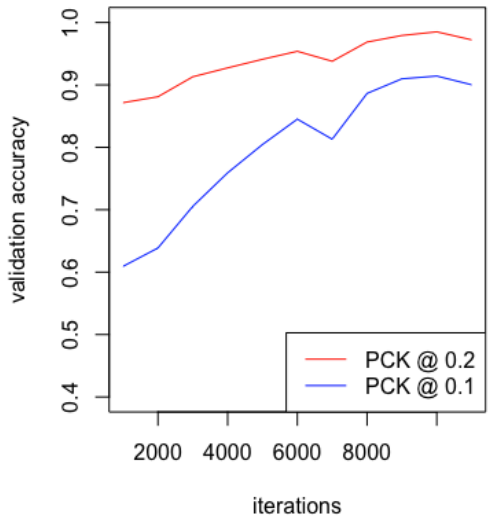


Figure 5.3.3: Validation accuracies computed during the training of the pose estimator. The training dataset is a mixture of 75% MPII and 25% Penn Action data. The validation accuracies are computed on the validation split of the Penn Action dataset.

The validation accuracy scores achieved on the Penn Action validation dataset can be seen in (**Fig.** 5.3.3). As can be seen, the validation accuracy of the pose estimator stops increasing after approximately 9000 training iterations. In (**Tab.** 5.3.4), the test accuracies of the model after 9000 iterations using both PCK with $\alpha = 0.2$ and $\alpha = 0.2$

PCK @ 0.2	PCK @ 0.1
80.26	59.28

Table 5.3.4: Test results for pose estimation on the Penn Action test set, using a pose estimator with 4 prediction blocks and 2 context heatmaps. The pose estimator is trained on a mixture dataset, containing 75% MPII and 25% Penn Action training data.

can be seen. It is difficult to compare the results to [LPT18], since the authors do not provide their results of the pose estimator training. In addition, comparison with different approaches in the literature is difficult because of the mixture of training data. To gain a better insight into the performance of the pose estimator, this thesis evaluated the accuracy of the pose estimator w.r.t. PCK on a per joint basis. These results can be seen in (**Tab.** 5.3.5). In comparison to the pose estimation results of Section 5.3.2.1, it can be seen that the accuracy of the joint associated with the head (upper neck) is much lower. This is most likely due to the fact that the position of the upper neck is slightly different in the MPII dataset in comparison to the Penn Action dataset. Thus, when mixing the datasets, the pose estimator likely learns the position of the head joint in the MPII dataset, since there are more training images present in the mixture dataset. This leads to a higher prediction error in the Penn Action test dataset. It can also be seen in (**Tab.** 5.3.5) that the pose estimator more accurately estimates joints on the right side of the body (from the position of the subject in the image) in comparison to the left body side. To evaluate whether this difference is statistically significant, we used a randomization test with a significance level of 0.05. The resulting p-value for the change in accuracy between left and right arms is 0.0, while the p-value for the difference between right and left legs is 0.0. The p-values are zero because they are too small for the floating point precision of the machine, which means that the change in accuracy between left and right sides are statistically significant. One reason for the significant difference in accuracy might be that, for most people, the predominant side of their body is the right side. This might then lead to an overrepresentation of actions performed by right-handed people in the dataset. Some example images of the predicted poses on the test dataset can be seen in (**Fig.** 5.3.4).

The pretrained pose estimator weights are inserted into the HAR model after the training process of the pose estimator finished. The HAR model is trained using Penn Action training data. Also, the weights of the pose estimator are initially frozen. The HAR model is then trained until the validation accuracies stops increasing, at which point the authors unfreeze the weights of the pose estimator and lower the learning

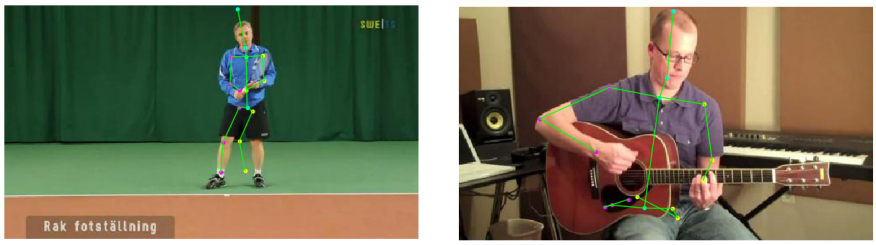


Figure 5.3.4: Two example poses estimated using the pose estimator trained on the mixed dataset of MPII and Penn Action. **Left:** An example of an image where the left leg joints were not estimated correctly. **Right:** Example, where the legs are wrongly estimated because they are not seen in the frame.

Ankle (r)	**Knee (r)**	**Hip (r)**	**Hip (l)**	**Knee (l)**	**Ankle (l)**	**Upper neck**	**Wrist (r)**
70.24	65.11	68.98	64.81	62.23	71.92	36.92	45.3
Elbow (r)	**Shoulder (r)**	**Shoulder (l)**	**Elbow (l)**	**Wrist (l)**	**Total**		
54.86	69.12	67.71	54.53	42.50	59.28		
Arms (l)	**Arms (r)**	**Arms (both)**	**Legs (l)**	**Legs (r)**	**Legs (both)**	**Upper body**	
54.92	56.45	55.68	66.34	68.11	67.22	46.37	

Table 5.3.5: Per joint accuracy, computed on the Penn Action test set using PCK @ 0.1 measure. In addition, aggregated accuracy values are given in the third row for different sets of joints. *Arms* for both left and right are computed by taking the average of the shoulder, elbow and wrist accuracies. *Legs* are computed the same way, using knee, ankle and hip accuracy values. For *Upper body*, the upper neck is added to *Arms (both)*.

rate. The model is then fine-tuned in an end-to-end approach. As an optimizer, the Stochastic Gradient Descend is used, with the addition of a momentum value $\gamma = 0.98$ and the use of a method called *Nesterov Accelerated Gradient (NAG)* [Nes83].

Momentum is used to accelerate the gradient descent process by incorporating a fraction γ of the previous gradients when computing the new gradients. Consider a regular update to the weights w of a neuron using gradient descent, as explained in (**Sec.** 2.3.1.3), given by

$$w_{t+1} = w_t - \eta \nabla w_t, \tag{5.3.4}$$

where η refers to the learning rate and ∇w_t refers to the gradients w.r.t. w_t. When using momentum, a update term v_t is computed following

$$v_t = \gamma v_{t-1} + \eta \nabla w_t. \tag{5.3.5}$$

Afterwards, the new weights are computed by subtracting v_t from w_t:

$$w_{t+1} = w_t - v_t. \tag{5.3.6}$$

The intuition behind momentum is that the descend towards the local minimum of the loss function can be accelerated by assuming that the gradients do not change significantly. If gradients with a similar direction were multiple times before, then the assumption is that the next gradients will be similar as well. This assumption, however, can lead to the case where the local minimum of the loss function is surpassed, which is often referred to as *overshooting* the minimum. To minimize the probability overshooting while still accelerating the descend towards the local minimum, Nesterov Accelerated Gradient adds an additional step to the momentum process. It calculates a *look-ahead* weight $w_{la} = w_t \gamma v_{t-1}$ and evaluates the gradient using this new weight. If the gradient at w_{la} points to a different direction than γv_{t-1} it will correct the surpassing of the minimum to an extend by reducing changing the direction of the gradient slightly. The following formula describes the update process of weights w_t using the NAG:

$$w_{t+1} = w_t - (\gamma v_{t-1} + \eta \nabla w_{la}). \tag{5.3.7}$$

The results of the training process can be seen in (**Fig.** 5.3.5). A learning rate of 2^{-5} is used, as suggested by the authors in their work. The training process with frozen pose estimator weights is shown to the left of the blue vertical line. We show the validation accuracy, which is computed by predicting the action for every 16 frame fragment in the validation dataset. Additionally, we show the training accuracy of the pose estimator using the PCK metric with $\alpha = 0.2$, as well as the training accuracy of the action classification. As expected, the pose estimation training accuracy does not change, because the weights are frozen. Unexpectedly, the validation accuracy did not increase after unfreezing the pose estimator weights and after finetuning the network. In fact, the training accuracy of the action classification also decreased. Because finetuning the network did not lead to a better result in terms of validation accuracy, it was decided to continue the evaluation using the non-finetuned model.

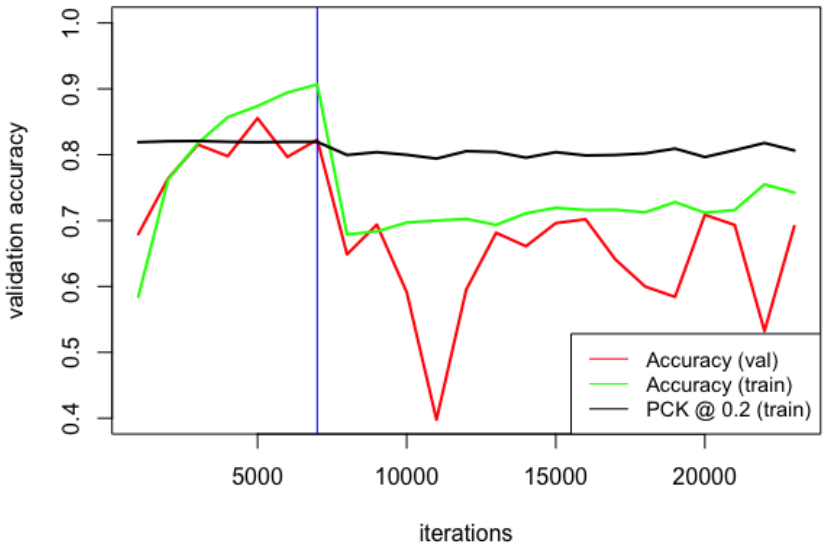


Figure 5.3.5: Validation and training accuracies for the HAR model, trained on Penn Action data. Validation accuracy is computed by classifying each 16 frame fragment and comparing it to the ground truth class. Validation accuracy is computed identically, except on the validation dataset. The pose estimator training accuracy is computed using the PCK metric with $\alpha = 0.2$. We show the results without finetuning, i.e., with the pose estimator weights frozen, to the left of the blue vertical line. To the right, the results for finetuning the network in an end-to-end approach are shown.

The results on the Penn Action test dataset for the model at iteration 5000 can be seen in (**Tab.** 5.3.6). In comparison to the results provided by [LPT18], it is apparent that our model performs significantly worse. One possible explanation could be that the pose estimator of the authors achieved a higher accuracy, which will lead to a higher action classification accuracy. Since the authors did not publish an evaluation of the pose estimator, a comparison between their pose estimator and the one trained in this thesis is not possible. Another possible explanation could be that the authors forgot to mention a crucial step in training their model. For example, they do not mention how many iterations both the pose estimator and action recognition model were trained.

Single Clip accuracy	Multi Clip accuracy	[LPT18]
79.56	81.66	97.40

Table 5.3.6: Test results for action classification on the Penn Action test set, in comparison to the result from [LPT18]. The authors do not mention whether the reported accuracy value was determined using a single clips or multiple clips.

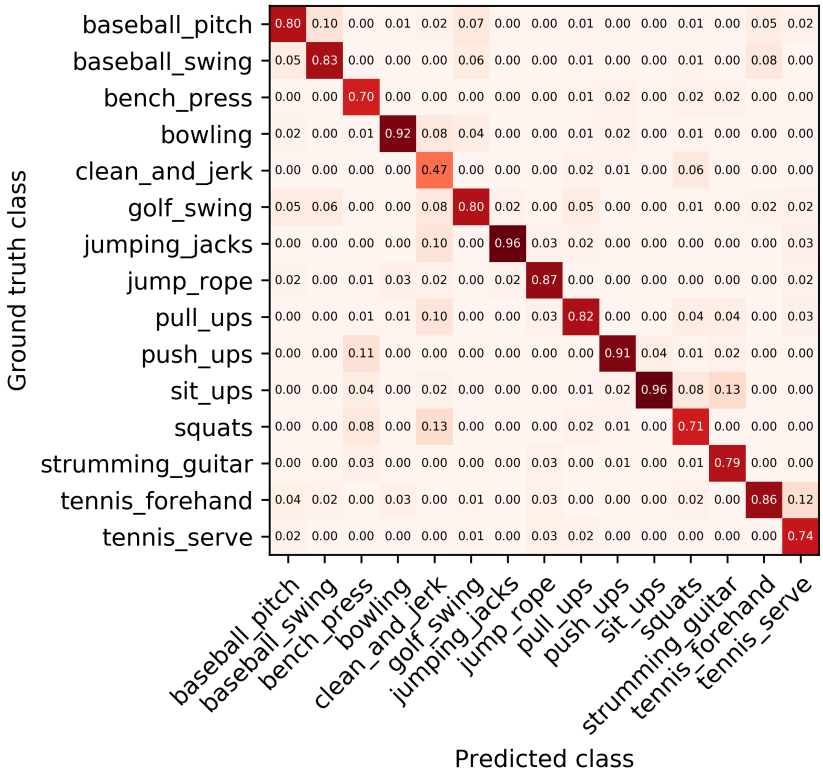


Figure 5.3.6: Confusion matrix of the Penn Action test set after training the HAR classifier. Notice that the model is most uncertain with the class *clean and jerk*.

To gain a better insight into the types of errors the network produces, we computed a confusion matrix of the test data. A confusion matrix shows not only the accuracy of the model for each action, but it also shows which classes the model often confuses. The confusion matrix is shown in (**Fig.** 5.3.6). As can be seen, the network is most uncertain when predicting the class *clean and jerk*. Clean and jerk is a weight lifting

action, where the person bends their knees while holding a weight lifting bar and then lifts the weights while standing up. See (**Fig.** 5.3.7) for an example of this action. Notice that the *clean and jerk* action starts with bend knees. This is similar to the action of *squat*, which might explain why the network gets confused between these two classes. In (**Sec.** 4.1.3), it was explained that the network processes a video in sets of 16 consecutive frames. This means that, if a person takes a long time to stand up after bending their knees, the network might classify the 16 frame excerpt as *squats*, because the poses are very similar. A network architecture, which takes more frames of a video into account before predicting an action, might be able to distinguish these classes better. On the other hand, actions with highly distinct poses, such as *sit ups* and *jumping jacks*, are seldom confused with other classes. These two findings indicate that very precise pose information is necessary to reduce the amount confusion between similar classes.

Figure 5.3.7: Example frames of a video where a person performs the *clean and jerk* action. Notice the similarities in pose to *squats*, where the person also bends their knees.

In conclusion, while our results do not match the results in [LPT18], this is most likely due to the performance of the pose estimator. More experimentation towards increasing the accuracy of the pose estimator would be necessary to decrease the amount of confusion between certain actions. It was shown that the pose estimator is more accurate on joints of the person's right side, as opposed to the left side. For future work, a deeper investigation into this phenomenon would be necessary.

5.3.3 *Pose estimation on JHMDB dataset*

Similar to the experiment performed in (**Sec.** 5.3.2), this thesis evaluates different pose estimator configurations using the JHMDB dataset. Specifically, pose estimators with $2, 4$ and 8 prediction blocks are trained with and without context heatmaps to assess the performance of the network on a more challenging video dataset in terms of action diversity. In terms of the pose estimator architecture, there are no changes to the experiments on the MPII dataset.

One assumption this thesis makes is that the ground truth bounding boxes for training, validation and test datasets are given. This assumption was also made by [LPT18] for training and evaluating the pose estimator on the MPII dataset. The accuracies, computed using three different metrics, are shown in (**Tab.** 5.3.7). As can be seen, the best performing pose estimator configuration is 4 prediction blocks and 0 context heatmaps. Notice that the test accuracies when using 8 prediction blocks are significantly lower in comparison to the other configurations. This is most likely due to the limited amount of data of the JHMDB dataset. This hypothesis is supported by the fact that the accuracies when using context heatmaps are lower, in comparison to not using context heatmaps. Context heatmaps introduce additional parameters to the pose estimator model, which need to be trained. In general, the higher the number of parameters of a model, the more training data is needed to fit the model.

The p-values in (**Tab.** 5.3.7) indicate that, for a pose estimator configuration with 2 prediction blocks, adding context heatmaps significantly decreases the accuracy of the model. This is most likely due to an increase in parameters of the pose estimator, which means that more data is necessary to fit the model. This also applies to the configurations using 4 prediction blocks, where the evaluated metrics are also lower when using context heatmaps. These findings imply that the utility of context heatmaps is highly dependent on the training dataset size. Interestingly, when using 8 prediction blocks, the opposite appears to be true. In (**Sec.** 5.3.2.1), it was found that the use of context heatmaps only significantly increased the accuracy of the pose estimator if 8 prediction blocks were used. This finding, in combination with the findings on the pose estimator trained on the JHMDB dataset, indicate that using context heatmaps only positively impacts the accuracy of a pose estimator if a high number of prediction blocks are used. One reason for this might be that context heatmaps benefit from the refinement process of the stacked architecture more than the regular heatmaps (see (**Sec.** 4.1.3)).

In (**Tab.** 5.3.8), the accuracies of different pose estimator configurations are compared using randomization testing. The configurations with 8 prediction blocks are omitted from this comparison, because their accuracies are much lower in comparison to the other configuration and there is no need for statistical significance testing. It is found that there are no statistically significant differences between the accuracies of the tested configurations for any metric. This is consistent with the findings in (**Sec.** 5.3.2.1), where the only significant differences were found between 8 and 2 prediction blocks.

Next, the accuracy of best performing pose estimator configuration (4 prediction blocks, 0 context heatmaps) with regards to individual joints and groupings of joints is evaluated in (**Tab.** 5.3.9). As a metric, PCK using $\alpha = 0.1$ is used, since this metric more precise in comparison to PCK using $\alpha = 0.2$, which aids in determining

			p-value			p-value			p-value
nr_blocks	2	2		4	4		8	8	
nr_context	0	2		0	2		0	2	
PCK @ 0.2	96.85	95.61	**0.0**	**96.94**	96.06	**0.0**	74.95	75.47	0.408
PCK @ 0.1	**92.03**	90.20	**0.0**	91.69	90.95	0.072	50.13	52.14	**0.005**
PCKu @ 0.2	87.32	86.21	**0.031**	**87.49**	86.92	0.224	44.03	45.30	0.085

Table 5.3.7: Test accuracies of the different pose estimator configurations, computed on the JHMDB test set. PCK is computed for two threshold values, $\alpha = 0.2$ as well as $\alpha = 0.1$. Additionally, PCKu is computed using $\alpha = 0.2$. Maximum values per metric, as well as p-values below the significance level of 0.05, are shown in bold.

			p-value			p-value
nr_blocks	2	4		2	4	
nr_context	0	0		2	2	
PCK @ 0.2	96.85	**96.94**	0.689	95.61	96.06	0.121
PCK @ 0.1	**92.03**	91.69	0.396	90.20	90.95	0.089
PCKu @ 0.2	87.32	**87.49**	0.742	86.21	86.92	0.164

Table 5.3.8: Different model configurations for estimating pose on the JHMDB dataset, with their corresponding accuracy values. The configurations using 8 prediction blocks are omitted because the results are significantly worse in comparison to the other configurations. The p-values were computed using a randomization test. None of the p-values are below the significance level of 0.05.

the differences between the individual joint accuracies. In comparison to the joint accuracies computed in (**Tab.** 5.3.3), it becomes apparent that left and right ankles are more accurately estimated on the JHMDB dataset. On both datasets, the joints associated with the head and torso region of the subject, i.e., upper neck, pelvis and head, are the most accurate. One explanation for this is that there is less confusion with other, similar looking joints. For example, while the left ankle might easily be confused with the right ankle, the head and torso region joints do not have a similarly looking counterpart. In addition, these joints are less articulated, meaning that they do not change in position between different actions performed. Similar to (**Sec.** 5.3.2.2), right arms and legs are more accurately detected. The difference in accuracies is also statistically significant with a significance level of 0.05 (p values 0.003 and 0.0).

As discussed before, this thesis assumed that human bounding boxes are given for training, validation and testing data. When comparing our results to the state-of-the-art approach by [SWGH17], it was noted that our results outperforms the state-of-the-art. The authors report a PCK accuracy of 81.6% using $\alpha = 0.2$ and 68.7% using $\alpha = 0.1$. In comparison, our best performing model with 4 prediction blocks and 0 context heatmaps achieved 96.94% PCK accuracy using $\alpha = 0.2$ and 91.69% using $\alpha = 0.1$. However, the authors of [SWGH17] do not mention whether or not they used ground truth bounding boxes for the subject while training or evaluating their model. Thus, it was decided to estimate the bounding box of the test data before estimating the pose. To achieve this, we first estimate the pose for each frame without taking the ground truth bounding box into account. The estimated poses are used to define a bounding box by choosing the largest and smallest x and y values over all joints as the corner points of the bounding box. Additionally, the resulting estimated bounding box is increased by 30 pixels in both dimensions, since the estimated poses are not accurate enough to ensure that the subject is fully visible. Then, the estimated bounding box for each frame is used to crop the image. The results can be seen in (**Tab.** 5.3.10). When comparing the results to [SWGH17] using estimated bounding boxes on the test data, it becomes clear that our model is not able outperform the state-of-the-art. A more accurate pose estimator would be needed to increase the accuracy of the initial pose estimation used for estimating the bounding box. This would lead to less discrepancy between the pose accuracy using the ground truth bounding box and the estimated bounding box.

In conclusion, pose estimation on the JHMDB dataset appears to be harder in comparison to the MPII dataset. One reason might be the small size of the JHMDB dataset. Also, since the JHMDB dataset is a video dataset, it contains video artifacts such as blur and camera jitter. Moreover, the resolution of the frames is significantly lower in comparison to the MPII dataset. This means that, since images from both

Ankle (r)	Knee (r)	Hip (r)	Hip (l)	Knee (l)	Ankle (l)	Pelvis	Upper neck
0.87	0.84	0.90	0.88	0.83	0.85	0.93	0.94
Head top	**Wrist (r)**	**Elbow (r)**	**Shoulder (r)**	**Shoulder (l)**	**Elbow (l)**	**Wrist (l)**	**Total**
0.97	0.83	0.87	0.89	0.89	0.82	0.82	0.92
Arms (l)	**Arms (r)**	**Arms (both)**	**Legs (l)**	**Legs (r)**	**Legs (both)**	**Upper body**	**Lower body**
0.84	0.86	0.85	0.85	0.87	0.86	0.88	0.87

Table 5.3.9: Per joint accuracy w.r.t. PCK @ 0.1, computed on the JHMDB test set using 4 prediction blocks and 0 context heatmaps. In addition, aggregated accuracy values are given in the third row for different sets of joints. *Arms* for both left and right are computed by taking the average of the shoulder, elbow and wrist accuracies. *Legs* are computed the same way, using knee, ankle and hip accuracy values. For *Upper body*, upper neck and head top are added to both *Arms* aggregations. *Lower body* is computed by aggregating both *Legs* accuracies and pelvis accuracy.

							[SWGH17]
nr_blocks	2	2	4	4	8	8	-
nr_context	0	2	0	2	0	2	-
PCK @ 0.2	72.20	**75.94**	74.80	73.25	51.10	51.78	81.6
PCK @ 0.1	41.53	**45.77**	43.82	41.76	20.82	21.42	68.70
PCKu @ 0.2	33.77	**38.02**	35.47	34.55	19.22	19.81	-

Table 5.3.10: Test accuracies of the JHMDB pose estimation models when using estimated bounding boxes for the test data. In comparison to the state-of-the-art approach by [SWGH17], our model achieves notably lower accuracy values in both metrics. The highest accuracies achieved by our methods are shown in bold.

dataset are resized to 256×256 pixels, the images from the JHMDB dataset contain more artifacts due to scaling and compression.

5.3.4 *HAR on JHMDB Dataset*

To assess the performance of the action recognition model on the JHMDB dataset, the pretrained pose estimator with 4 blocks and 2 context maps from (**Sec.** 5.3.3) is used, in order to compare the results to the performance on the Penn Action dataset experiment (**Sec.** 5.3.2.2). The batch size is set to 12 since the GPU used for experimentation did not allow for a bigger batch size. Keep in mind that, since JHMDB is a video dataset, each item in the batch contains 16 frames (see (**Sec.** 5.1.3)). Effectively, this leads to a batch size of $16 \cdot \texttt{batch\_size}$ for the pose estimator since the pose for each frame is computed independently, before aggregating the estimated poses into the pose cube (see (**Sec.** 4.1.3.4)). As an initial learning rate, 2^{-5} is chosen, since this is the value used in the Penn Action HAR experiment. Similar to before, the point where the validation data accuracy does not increase further is found by training the model with a pose estimator with fixed weights. The results of the training process can be seen in (**Fig.** 5.3.8). It is observed that the action training accuracy quickly approaches 100% while the validation accuracy does not show an upward trend throughout the training process. This is especially interesting considering the results in (**Sec.** 5.3.5). One explanation could be that the network overfit the training data, which is supported by the fact that both training accuracies computed quickly approach 100%. In the time of writing this thesis, we did not find a solution to this problem. In order to not discuss possibly erroneous results, it was decided to omit a discussion of testing results regarding this experiment.

5.3.5 *Effect of Combining Loss Functions*

Next, this thesis proposes a method for training the network in an end-to-end approach. This means that no part of the network is pretrained. To achieve this, the loss of the pose estimator and the loss of the action recognition pipeline are combined. First, during the training process, the losses of all intermediate results from the pose estimator are computed using the elastic net loss. This is equivalent to the way the pose estimator is trained in [LPT18]. Second, the loss of all intermediate results from the action recognition are computed using categorical cross-entropy. It was decided to weigh both losses equally, i.e., sum them without weighting either the pose estimation

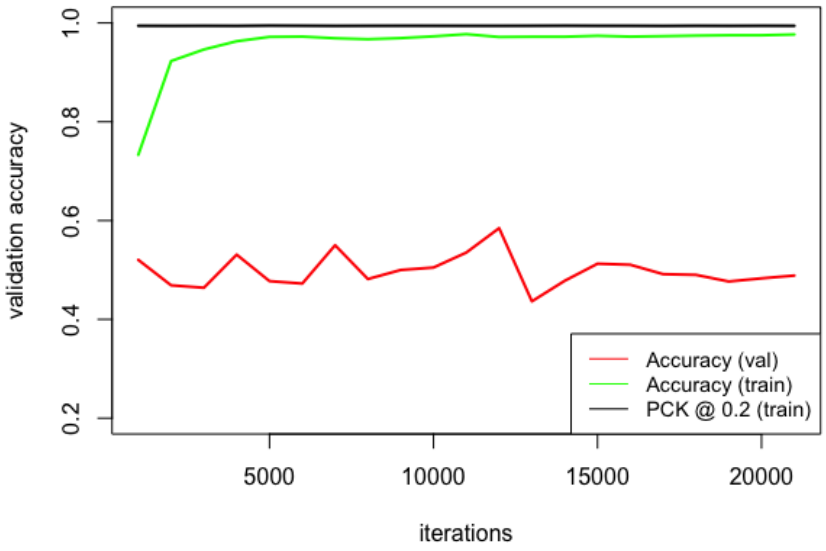


Figure 5.3.8: The training and validation accuracies computed during the training process of the HAR model on JHMDB training data.

or action recognition loss more in order to not make any assumptions about which part is more important.

The network needs a high amount of iterations for training until the validation accuracy stops increasing. This is to be expected, since no part of the network is pretrained and thus all parts have to be trained from random initialization. See (**Fig.** 5.3.9) for a graphical representation of the training process. As can be seen, the training pose accuracy increases at a slower rate in comparison to the action recognition training accuracy. Both training accuracies approach 100%, albeit after much more iterations in comparison to (**Sec.** 5.3.2.2). After approximately 228.000 iterations, the validation accuracy stops increasing, which is why it was decided to further evaluate the model at that point. The results of the evaluation on the JHMDB test set can be seen in (**Tab.** 5.3.11).

When comparing the pose accuracy of the end-to-end model to the pose estimators trained in (**Sec.** 5.3.3), it is apparent that the end-to-end model is significantly worse in all three metrics related to pose estimation (see (**Tab.** 5.3.12)). Further experimentation is necessary to assess whether or not the model hyperparameters like learning rate and

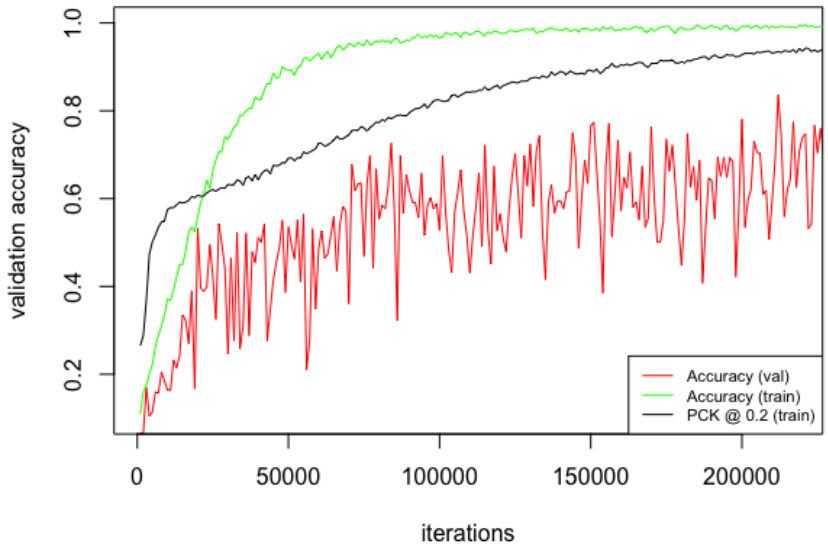


Figure 5.3.9: Training and validation accuracies of the HAR model trained using an end-to-end approach, without pretraining individual parts of the model. Validation accuracy given as percentage of correctly classified validation video clips.

batch size can be optimized in order for the end-to-end model to achieve comparable results to the pretrained model. In comparison to the state-of-the-art approaches for HAR on the JHMDB dataset in (**Sec.** 3.2.3.3) and (**Sec.** 3.2.3.4), the action recognition is also significantly lower (see (**Tab.** 5.3.11)). However, the findings are promising, because the network is, in general, able to achieve a high classification accuracy on the action recognition metrics by being trained in an end-to-end approach. This suggests that, while the pose estimator might be weaker in comparison to a standalone pose estimator, it is enough to make accurate action classifications. In future work, an investigation into changing the architecture of the HAR model might lead to comparable results to the literature.

A confusion matrix of the action classification on the test dataset is computed to gain a better understanding of the types of misclassification of the network (see (**Fig.** 5.3.10)). From the confusion matrix, it becomes clear that the network is highly uncertain between the classes *sit* and *stand*. Only 38% of the test datapoints classified as *stand* were actually from the *stand* class. In addition, in 29% of the cases where the

Single Clip accuracy	Multi Clip accuracy	[KY18]	[CWRS18]
65.88	68.09	78.81	**87.90**

Table 5.3.11: Test results of the action recognition accuracy on the JHMDB test set after 228.000 iterations of training the model in an end-to-end approach. The accuracy values are much lower in comparison to the state-of-the-art methods described in (**Sec.** 3.2.3).

	PCK @ 0.2	PCK @ 0.1	PCKu @ 0.2
End-to-end model	81.12	49.23	35.63
JHMDB pose estimator (**Sec.** 5.3.3) 4 prediction blocks, 0 context heatmaps	**96.94**	**91.69**	**87.49**

Table 5.3.12: Pose estimation accuracy, meassured using three different metrics, in comparison to the findings in (**Sec.** 5.3.3).

network predicted *stand*, the ground truth class was *sit*. This is most likely due to the fact that the first frames of many clips labeled as *sit* show people standing. These people are about to sit down, which might explain the strong confusion between the two classes. See (**Fig.** 5.3.11) for an example clip which were wrongly classified as *stand* when the ground truth class was *sit*. Notice that, for the majority of the extracted frames, the person is still standing. One way of reducing this error would be to include more temporal context to the decision process. In the model architecture, a video clip is divided into smaller clips of 16 frames. This amount of temporal context appears to be too small for some classes, causing the network to confuse these classes. Another approach would be to relabel the dataset and to ensure that every clip strictly belongs to one class.

In (**Tab.** 5.3.13), the accuracy of the poses are given for each joint, as well as for some aggregations like *upper body*. On average, the network is more precise on the joints associated with the legs in comparison to the arm joints. Also, joints associated with the right side of the subjects are detected with a higher accuracy than the joints associated with the left side. The differences in accuracies are statistically significant with a p-value too close to zero to be accurately reported ($p = 0.0$). This finding supports the findings in (**Sec.** 5.3.2.2) and (**Sec.** 5.3.3). This indicates that, independent of the dataset, the pose estimator is able to more accurately estimate right side body joints from left side body joints. Whether this is due to the fact that most images in these datasets feature right-handed persons of because of the particular pose estimator architecture needs to be evaluated in future work.

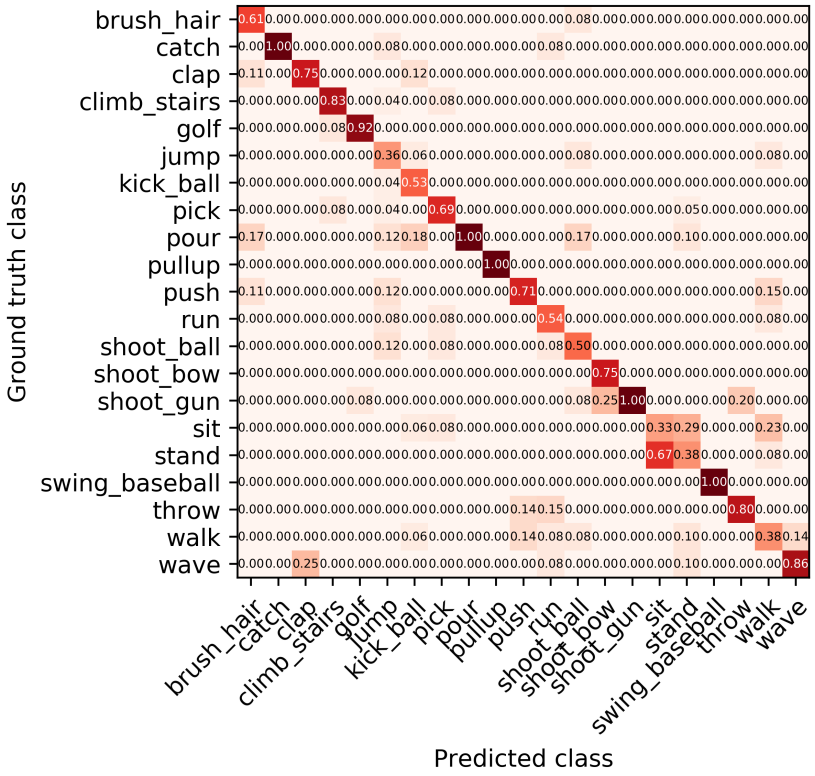


Figure 5.3.10: Confusion matrix, computed on the JHMDB test dataset, using the end-to-end model. Notice the strong confusion between the classes *sit* and *stand*.

In conclusion, while the end-to-end approach does not achieve state-of-the-art performance for action recognition on the JHMDB benchmark, it is a promising approach, since the training process complexity is significantly lower. Instead of pretraining weights of a pose estimator, which involves separate hyperparameter tuning, the end-to-end approach can be learned from randomly initialized weights. With changes in the model architecture, such as using more temporal context, and by using a larger dataset, the network might be able to reach a performance close to the state-of-the-art.

Figure 5.3.11: Frames extracted from a video clip from the JHMDB test dataset with the ground truth class *sit*. Notice that the person, which sits down in the video clip, is standing for the majority of the clip. This explains the strong confusion between the classes.

Ankle (r)	**Knee (r)**	**Hip (r)**	**Hip (l)**	**Knee (l)**	**Ankle (l)**	**Pelvis**	**Upper neck**
47.82	55.66	60.17	61.18	55.66	54.32	54.15	67.82
Head top	**Wrist (r)**	**Elbow (r)**	**Shoulder (r)**	**Shoulder (l)**	**Elbow (l)**	**Wrist (l)**	**Total**
74.51	32.44	32.29	43.80	51.94	36.29	42.47	49.24
Arms (l)	**Arms (r)**	**Arms (both)**	**Legs (l)**	**Legs (r)**	**Legs (both)**	**Upper body**	**Lower body**
0.49	0.39	0.44	0.54	0.57	0.55	0.51	0.56

Table 5.3.13: Per joint accuracy, computed on the JHMDB test set using PCK @ 0.1 meassure. In addition, aggregated accuracy values are given in the third row for different sets of joints. *Arms* for both left and right are computed by taking the average of the shoulder, elbow and wrist accuracies. *Legs* are computed the same way, using knee, ankle and hip accuracy values. For *Upper body*, upper neck and head top are added to both *Arms* aggregations. *Lower body* is computed by aggregating both *legs* accuracies as well as pelvis.

6 CONCLUSION

In [LPT18], the authors presented a convolutional neural network for Human Activity Recognition, which is able to jointly learn the pose of a person and the action performed in video clips. The authors introduced the Soft-argmax function [LTP17], which allows for this novel approach of learning. One downside of the approach by [LPT18] is that they only evaluated their network on the Penn Action dataset. This thesis evaluated the performance of the network using the more complex, albeit smaller JHMDB dataset [JGZ+13]. We also performed two experiments to evaluate the accuracy of the Soft-argmax function. Finally, the network was trained without using pretrained parameters for the pose estimator to investigate whether or not pretraining is necessary.

First, the Soft-argmax was evaluated in a qualitative and quantitative manner. It was found that the Soft-argmax appears to not be able to correctly extract the pose information at the border of the input image. However, this does not impact its performance on extracting pose from synthetic pose heatmaps from the MPII dataset. This suggests that the Soft-argmax is a viable alternative to the argmax function, which was previously necessary to extract pose coordinates joint heatmaps.

Second, this thesis was not able to achieve identical results in terms of pose estimator accuracy on the MPII dataset. The authors report a pose accuracy, using the PCKh metric, of 91.2 percent. This thesis achieved 84.10 percent accuracy using the same metric. The authors used multiple pose estimations of the test images, which were averaged to get the final pose estimation. However, the authors did not publish a detailed explaination of this procedure. It was found that the results on the validation data was only slightly below the validation accuracy reported by the authors (87 percent PCKh accuracy in comparison to 89 percent). This suggests that their procedure for estimating the pose of the test images is the deciding factor to explain the difference in test accuracy.

Third, we were not able to recreate the results of the authors regarding the accuracy of the HAR model. The authors achieved 97.40 percent accuracy, in comparison to 81.66 percent achieved in this thesis. This discrepancy is most likely due to the fact that the pose estimator, whose weights were pretrained on a mixture dataset of MPII and Penn Action data, was not accurate enough. A comparison of pose estimator accuracy with [LPT18] was not possible because the authors did not publish their results for this experiment. When investigating the accuracy of the pose estimator

for each type of joint, it was found that the pose estimator performed significantly better on joints associated with the right side of the persons body, in comparison to the left side of the person. These findings could be replicated with the JHMDB dataset, suggesting that either the model is predisposed to be more accurate on right side body joints or that both datasets contain a bias towards right side body joints. This bias could be that both datasets are unbalanced w.r.t. the handedness of the subjects in the videos.

Fourth, when evaluating the pose estimator on the challenging JHMDB dataset, it was found that the pose estimator is able to adapt to a different, smaller, but more complex dataset, achieving accuracy results only slightly below the state-of-the-art approaches in the literature. This suggests that the pose estimator architecture generalizes well to different use cases and and it can serve as a baseline for future work in this field.

Fifth, this thesis tried to train the HAR model from [LPT18] on the more challenging JHMDB dataset. However, the results appeared to be erroneous, which is why further analysis was ommited in order to not report and interpret wrong results.

Sixth, it was found that training the network in an end-to-end approach is feasible. The accuracy of the end-to-end model (68.09 percent) was significantly lower in comparison to the state-of-the-art approaches for HAR on the JHMDB dataset by [CWRS18] (87.90 percent) and [KY18] (78.81 percent). However, the results are promising and future work could improve upon the results presented. Additionally, it was found that the HAR models struggle with interclass variance on the JHMDB dataset, because some classes (like *sit*) are very similar to other classes (like *stand*). A video clip labeled as *sit* often contains persons in the process of sitting down, which means that some frames contain persons standing as well. A network architecture using more temporal information would most likely increase the accuracy on this dataset.

6.1 FUTURE WORK

Finally, this thesis makes some suggestions regarding possible future work, based on the findings presented earlier. First, the pose estimator architecture used in the HAR pipeline computes the pose for each frame in a video clip without taking poses of previous frames into account. Utilizing a pose estimator capable of processing video clips directly and utilizing previous poses could lead to better results, since the change in pose between consecutive frames of a video are very small.

Second, a HAR architecture using a larger temporal context would decrease the amount of error due to interclass variance. The model presented in [LPT18] divides

each video clip into chunks of 16 frames, which this thesis found leads to confusion when predicting the actions. This is because, for actions such as *sit* and *stand*, large parts of the video clips are identical. Ideally, the clip size would not determine the network architecture. To that end, using 3D convolutional layers could be implemented, as well as recurrent layers.

Third, larger and fully annotated video datasets could be gathered. While the JHMDB dataset, in terms of action diversity, is a more complex dataset in comparison to the Penn Action dataset, the later dataset is significantly larger in terms of number of frames. For large networks, such as the pose estimator with 8 prediction blocks, it was found that, even using strong augmentation, the JHMDB dataset did not contain enough data to train large pose estimators. One approach could be to increase the size of the JHMDB dataset by annotating more clips found in the HMDB dataset [KJG+11] using the proposed puppet tool. This approach would make the JHMDB dataset more useful for evaluating deep neural networks found in modern literature. Additionally, the images in the JHMDB dataset are small compared to the MPII dataset. [LPT18] resizes the images to 256×256 pixels, after cropping the image around the person. This leads to artifacts in the JHMDB dataset, because the cropped images need to be scaled up. A dataset containing fully annotated, high resolution images would result in less artifacts and, possibly, higher pose estimation accuracy.

BIBLIOGRAPHY

[APGS14] Andriluka, Mykhaylo ; Pishchulin, Leonid ; Gehler, Peter ; Schiele, Bernt: 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH, Juni 2014, S. 3686–3693

[ARS09] Andriluka, Mykhaylo ; Roth, Stefan ; Schiele, Bernt: Pictorial structures revisited: People detection and articulated pose estimation. In: *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Miami, FL, Juni 2009, S. 1014–1021

[BBPW04] Brox, Thomas ; Bruhn, Andrés ; Papenberg, Nils ; Weickert, Joachim: High Accuracy Optical Flow Estimation Based on a Theory for Warping. In: Pajdla, Tomás (Hrsg.) ; Matas, Jiří (Hrsg.): *Proceedings of the 8th European Conference on Computer Vision (ECCV)*. Prague, CZ, Mai 2004, S. 25–36

[BMBM10] Bourdev, Lubomir ; Maji, Subhransu ; Brox, Thomas ; Malik, Jitendra: Detecting People Using Mutually Consistent Poselet Activations. In: Daniilidis, Kostas (Hrsg.) ; Maragos, Petros (Hrsg.) ; Paragios, Nikos (Hrsg.): *Proceedings of the 11th European Conference on Computer Vision (ECCV)*. Heraklion, GR, September 2010, S. 168–181

[BMP02] Belongie, Serge ; Malik, Jitendra ; Puzicha, Jan: Shape matching and object recognition using shape contexts. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 24 (2002), April, Nr. 4, S. 509–522

[Bra00] Bradski, Gary: The OpenCV Library. In: *Dr. Dobb's Journal of Software Tools* (2000)

[BTVG06] Bay, Herbert ; Tuytelaars, Tinne ; Van Gool, Luc: SURF: Speeded Up Robust Features. In: Leonardis, Aleš (Hrsg.) ; Bischof, Horst (Hrsg.) ; Pinz, Axel (Hrsg.): *Proceedings of the 9th European Conference on Computer Vision (ECCV)*. Graz, AT, 2006, S. 404–417

[CAFM16] Carreira, Joao ; Agrawal, Pulkit ; Fragkiadaki, Katerina ; Malik, Jitendra: Human Pose Estimation with Iterative Error Feedback. In:

Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). Las Vegas, NV, Juni 2016, S. 4733–4742

[Cho17] Chollet, Francois: Xception: Deep Learning With Depthwise Separable Convolutions. In: *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, Juli 2017, 1251–1258

[CLS15] Cheron, Guilhem ; Laptev, Ivan ; Schmid, Cordelia: P-CNN: Pose-Based CNN Features for Action Recognition. In: *Proceedings of the 2015 International Conference on Computer Vision (ICCV)*. Santiago, CL, Dezember 2015, 3218–3226

[CMAS14] Cherian, Anoop ; Mairal, Julien ; Alahari, Karteek ; Schmid, Cordelia: Mixing Body-Part Sequences for Human Pose Estimation. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH, Juni 2014, 2353–2360

[Cor18] Cornelisse, Daphne: *An intuitive guide to Convolutional Neural Networks*. https://www.freecodecamp.org/news/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050/. Version: April 2018

[CSWS17] Cao, Zhe ; Simon, Tomas ; Wei, Shih-En ; Sheikh, Yaser: Realtime Multi-person 2D Pose Estimation Using Part Affinity Fields. In: *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, Juli 2017, 1302–1310

[CWRS18] Choutas, Vasileios ; Weinzaepfel, Philippe ; Revaud, Jérôme ; Schmid, Cordelia: PoTion: Pose MoTion Representation for Action Recognition. In: *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, Juni 2018, 7024–7033

[CZ17] Carreira, Joao ; Zisserman, Andrew: Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset. In: *Proceedings of the 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu, HI, Juli 2017, 4724–4733

[DDS+09] Deng, Jia ; Dong, Wei ; Socher, Richard ; Li, Li-Jia ; Li, Kai ; Li, Fei F.: ImageNet: a Large-Scale Hierarchical Image Database. In: *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Miami, FL, Juni 2009, S. 248–255

[Der17a] DERTAT, Arden: *Applied Deep Learning - Part 1: Artificial Neural Networks*. https://towardsdatascience.com/applied-deep-learning-part-1-artificial-neural-networks-d7834f67a4f6. Version: November 2017

[Der17b] DERTAT, Arden: *Applied Deep Learning - Part 4: Convolutional Neural Networks*. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2. Version: November 2017

[DT05] DALAL, Navneet ; TRIGGS, Bill: Histograms of Oriented Gradients for Human Detection. In: *Proceedings of the 2005 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. San Diego, CA, Juni 2005, S. 886–893

[DTS06] DALAL, Navneet ; TRIGGS, Bill ; SCHMID, Cordelia: Human Detection Using Oriented Histograms of Flow and Appearance. In: LEONARDIS, Aleš (Hrsg.) ; BISCHOF, Horst (Hrsg.) ; PINZ, Axel (Hrsg.): *Proceedings of the 9th European Conference on Computer Vision (ECCV)*. Graz, AT, Mai 2006, S. 428–441

[FB81] FISCHLER, Martin A. ; BOLLES, Robert C.: Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography. In: *Communications of the ACM* 24 (1981), Juni, Nr. 6, S. 381–395

[FE73] FISCHLER, Martin A. ; ELSCHLAGER, Robert A.: The Representation and Matching of Pictorial Structures. In: *IEEE Transactions on Computers* 22 (1973), Januar, S. 67–92

[FH05] FELZENSZWALB, Pedro F. ; HUTTENLOCHER, Daniel P.: Pictorial Structures for Object Recognition. In: *International Journal of Computer Vision* 61 (2005), Januar, Nr. 1, S. 55–79

[FMJZ08] FERRARI, Vittorio ; MARIN-JIMENEZ, Manuel ; ZISSERMAN, Andrew: Progressive search space reduction for human pose estimation. In: *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Anchorage, AK, Juni 2008, S. 1–8

[FPW16] FEICHTENHOFER, Christoph ; PINZ, Axel ; WILDES, Richard: Spatiotemporal Residual Networks for Video Action Recognition. In: LEE, D. D.

(Hrsg.) ; Sugiyama, M. (Hrsg.) ; Luxburg, U. V. (Hrsg.) ; Guyon, I. (Hrsg.) ; Garnett, R. (Hrsg.): *Advances in Neural Information Processing Systems 29*. 2016, S. 3468–3476

[FPZ16] Feichtenhofer, Christoph ; Pinz, Axel ; Zisserman, Andrew: Convolutional Two-Stream Network Fusion for Video Action Recognition. In: *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, Juli 2016, S. 1933–1941

[FS97] Freund, Yoav ; Schapire, Robert E.: A decision-theoretic generalization of on-line learning and an application to boosting. In: *Journal of Computer and System Sciences* 55 (1997), Nr. 1, S. 119–139

[FS99] Freund, Yoav ; Schapire, Robert E.: A short introduction to boosting. In: *Japanese Society For Artificial Intelligence* 14 (1999), Nr. 5, S. 771–780

[GBC16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016

[GGT+18] Girdhar, Rohit ; Gkioxari, Georgia ; Torresani, Lorenzo ; Paluri, Manohar ; Tran, Du: Detect-and-Track: Efficient Pose Estimation in Videos. In: *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, Juni 2018, S. 350–359

[HKRA14] Htike, Kyaw K. ; Khalifa, Othman O. ; Ramli, Huda A. M. ; Abushariah, Mohammad A. M.: Human activity recognition for video surveillance using sequences of postures. In: *Proceedings of the 3rd International Conference on e-Technologies and Networks for Development (ICeND)*. Beirut, LBN, April 2014, S. 79–82

[Hoc91] Hochreiter, Josef: *Untersuchungen zu dynamischen neuronalen Netzen*, Technische Universität München, Diplomarbeit, Juni 1991

[Hoc98] Hochreiter, Josef: The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions. In: *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 06 (1998), April, Nr. 02, S. 107–116

[HS88] Harris, Christopher G. ; Stephens, Mike: A combined corner and edge detector. In: *Proceedings of the 4th Alvey Vision Conference (AVC)*. Manchester, UK, September 1988, S. 1–6

[HS97] Hochreiter, Josef ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Computation* 9 (1997), November, Nr. 8, S. 1735–1780

[HZRS16] He, Kaiming ; Zhang, Xiangyu ; Ren, Shaoqing ; Sun, Jian: Deep Residual Learning for Image Recognition. In: *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas, NV, Juni 2016, S. 770–778

[IGG16] Iqbal, Umar ; Garbade, Martin ; Gall, Juergen: Pose for Action - Action for Pose. In: *Proceedings of the 12th IEEE International Conference on Automatic Face and Gesture Recognition (FG)*. Washington, DC, Mai 2016

[JE10] Johnson, Sam ; Everingham, Mark: Clustered Pose and Nonlinear Appearance Models for Human Pose Estimation. In: *Proceedings of the 2010 British Machine Vision Conference (BMVC)*. Aberystwyth, UK, August 2010, S. 12.1–12.11

[JGZ+13] Jhuang, Hueihan ; Gall, Juergen ; Zuffi, Silvia ; Schmid, Cordelia ; Black, Michael J.: Towards understanding action recognition. In: *Proceedings of the 2013 International Conference on Computer Vision (ICCV)*. Sydney, AU, Dezember 2013, S. 3192–3199

[KF18] Kong, Yu ; Fu, Yun: Human Action Recognition and Prediction: A Survey. In: *arXiv:1806.11230 [cs]* (2018), Juni. http://arxiv.org/abs/1806.11230. – arXiv: 1806.11230

[KJG+11] Kuehne, Hildegard ; Jhuang, Hueihan ; Garrote, E. ; Poggio, Tomaso ; Serre, Thomas: HMDB: A large video database for human motion recognition. In: *Proceedings of the 2011 International Conference on Computer Vision (ICCV)*. Barcelona, ES, November 2011, S. 2556–2563

[KMS08] Kläser, Alexander ; Marszałek, Marcin ; Schmid, Cordelia: A Spatio-Temporal Descriptor Based on 3D-Gradients. In: *Proceedings of the 2008 British Machine Vision Conference (BMVC)*. Leeds, UK, September 2008

[KSH12] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: ImageNet Classification with Deep Convolutional Neural Networks. In: *Advances in Neural Information Processing Systems 25*. 2012, S. 1097–1105

[KTS+14] Karpathy, Andrej ; Toderici, George ; Shetty, Sanketh ; Leung, Thomas ; Sukthankar, Rahul ; Fei-Fei, Li: Large-scale Video Classification

with Convolutional Neural Networks. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH, Juni 2014, S. 1725–1732

[KY18] Khalid, Muhammad U. ; Yu, Jie: Multi-Modal Three-Stream Network for Action Recognition. In: *Proceedings of the 24th International Conference on Pattern Recognition (ICPR)*. Beijing, CHN, August 2018, S. 3210–3215

[Lap05] Laptev, Ivan: On Space-Time Interest Points. In: *International Journal of Computer Vision* 64 (2005), September, Nr. 2, S. 107–123

[Llo82] Lloyd, S.: Least squares quantization in PCM. In: *IEEE Transactions on Information Theory* 28 (1982), März, Nr. 2, S. 129–137

[LLS09] Liu, Jingen ; Luo, Jiebo ; Shah, Mubarak: Recognizing realistic actions from videos "in the wild". In: *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Miami, FL, Juni 2009, S. 1996–2003

[LMSR08] Laptev, Ivan ; Marszałek, Marcin ; Schmid, Cordelia ; Rozenfeld, Benjamin: Learning Realistic Human Actions from Movies. In: *Proceedings of the 2008 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Anchorage, AK, 2008

[LPT18] Luvizon, Diogo C. ; Picard, David ; Tabia, Hedi: 2D/3D Pose Estimation and Action Recognition Using Multitask Deep Learning. In: *Proceedings of the 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Salt Lake City, UT, Juni 2018, S. 5137–5146

[LTP17] Luvizon, Diogo C. ; Tabia, Hedi ; Picard, David: Human Pose Regression by Combining Indirect Part Detection and Contextual Information. In: *arXiv:1710.02322 [cs]* (2017), Oktober. http://arxiv.org/abs/1710.02322. – arXiv: 1710.02322

[Mac67] MacQueen, James: Some methods for classification and analysis of multivariate observations. In: *Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability* Bd. 1, 1967

[Max] Max Planck Institute for Intelligent Systems: *JHMDB Dataset Puppet tool*. http://jhmdb.is.tue.mpg.de/puppet_tool

[MLS09] Marszałek, Marcin ; Laptev, Ivan ; Schmid, Cordelia: Actions in Context. In: *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Miami, FL, Juni 2009, S. 2929–2936

[MP43] McCulloch, Warren S. ; Pitts, Walter H.: A logical calculus of the ideas immanent in nervous activity. In: *The Bulletin of Mathematical Biophysics* 5 (1943), Nr. 4, S. 115–133

[Nes83] Nesterov, Yurii: A method for unconstrained convex minimization problem with the rate of convergence o(1/k^2). In: *Doklady AN USSR* 269 (1983), S. 543–547

[NH10] Nair, Vinod ; Hinton, Geoffrey E.: Rectified Linear Units Improve Restricted Boltzmann Machines. In: *Proceedings of the 27th International Conference on Machine Learning (ICML)*. Haifa, ISR, Juni 2010, S. 807–814

[NYD16] Newell, Alejandro ; Yang, Kaiyu ; Deng, Jia: Stacked Hourglass Networks for Human Pose Estimation. In: *Proceedings of the 14th European Conference on Computer Vision (ECCV)*. Amsterdam, NL, Oktober 2016, S. 483–499

[PSF12] Prest, Alessandro ; Schmid, Cordelia ; Ferrari, Vittorio: Weakly Supervised Learning of Interactions between Humans and Objects. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 34 (2012), März, Nr. 3, S. 601–614

[Ram07] Ramanan, Deva: Learning to parse images of articulated bodies. In: *Advances in Neural Information Processing Systems 19*. MIT Press, 2007, S. 1129–1136

[RHW86] Rumelhart, David E. ; Hinton, Geoffrey E. ; Williams, Ronald J.: Learning representations by back-propagating errors. In: *Nature* 323 (1986), Oktober, Nr. 6088, S. 533–536

[RMH+14] Ramakrishna, Varun ; Munoz, Daniel ; Hebert, Martial ; Bagnell, J. A. ; Sheikh, Yaser: Pose Machines: Articulated Pose Estimation via Inference Machines. In: *Proceedings of the 13th European Conference on Computer Vision (ECCV)*. Zürich, CH, 2014, S. 33–47

[RMRHF18] Reining, Christopher ; Moya Rueda, Fernando ; Hompel, Michael ten ; Fink, Gernot A.: Towards a Framework for Semi-Automated

Annotation of Human Order Picking Activities Using Motion Capturing. In: *Proceedings of the 2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. Poznań, PL, September 2018, S. 817–821

[Roj96] Rojas, Raúl: *Neural Networks: A Systematic Introduction*. 1996

[Rud18] Rudolph, Günther: *Lecture notes from Introduction to Computational Intelligence*. https://ls11-www.cs.tu-dortmund.de/people/rudolph/teaching/lectures/CI/WS2018-19/lecture.jsp. Version: Oktober 2018

[Sah18] Saha, Sumit: *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b116 Version: Dezember 2018

[SAS07] Scovanner, Paul ; Ali, Saad ; Shah, Mubarak: A 3-dimensional Sift Descriptor and Its Application to Action Recognition. In: *Proceedings of the 15th ACM International Conference on Multimedia*. Augsburg, DE, 2007, S. 357–360

[SF68] Sobel, Irwin ; Feldman, Gary: *A 3x3 isotropic gradient operator for image processing*. Stanford, CA, 1968

[Sif14] Sifre, Laurent: *Rigid-Motion Scattering For Image Classification*, Echole Polytechnique, PhD Thesis, 2014

[SIVA17] Szegedy, Christian ; Ioffe, Sergey ; Vanhoucke, Vincent ; Alemi, Alexander A.: Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. In: *Proceedings of the 31st AAAI Conference on Artificial Intelligence*. San Francisco, CA, Februar 2017, S. 4278–4284

[SLC04] Schuldt, Christian ; Laptev, Ivan ; Caputo, Barbara: Recognizing human actions: a local SVM approach. In: *Proceedings of the 17th International Conference on Pattern Recognition (ICPR)* Bd. 3. Cambridge, UK, August 2004, S. 32–36

[SSLW17] Sun, Xiao ; Shang, Jiaxiang ; Liang, Shuang ; Wei, Yichen: Compositional Human Pose Regression. Venice, IT, Oktober 2017, 2602–2611

[ST13] Sapp, Benjamin ; Taskar, Ben: MODEC: Multimodal Decomposable Models for Human Pose Estimation. In: *Proceedings of the 2013 IEEE*

International Conference on Computer Vision (ICCV). Sydney, AU, Dezember 2013

[SWGH17] Song, Jie ; Wang, Limin ; Gool, Luc V. ; Hilliges, Otmar: Thin-Slicing Network: A Deep Structured Model for Pose Estimation in Videos. In: *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Honolulu : IEEE, 2017. – ISBN 978–1–5386–0457–1, 5563–5572

[SZ14] Simonyan, Karen ; Zisserman, Andrew: Two-Stream Convolutional Networks for Action Recognition in Videos. In: Ghahramani, Z. (Hrsg.) ; Welling, M. (Hrsg.) ; Cortes, C. (Hrsg.) ; Lawrence, N. D. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 27*. 2014, S. 568–576

[SZS12] Soomro, Khurram ; Zamir, Amir R. ; Shah, Mubarak: UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild. In: *arXiv:1212.0402 [cs]* (2012), Dezember. http://arxiv.org/abs/1212.0402. – arXiv: 1212.0402

[TBF+15] Tran, Du ; Bourdev, Lubomir ; Fergus, Rob ; Torresani, Lorenzo ; Paluri, Manohar: Learning Spatiotemporal Features With 3D Convolutional Networks. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*. Santiago, CL, Dezember 2015, 4489–4497

[TS14] Toshev, Alexander ; Szegedy, Christian: DeepPose: Human Pose Estimation via Deep Neural Networks. In: *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, OH, Juni 2014, S. 1653–1660

[VL01] Vincent, E. ; Laganiere, Robert: Detecting planar homographies in an image pair. In: *Proceedings of the 2nd International Symposium on Image and Signal Processing and Analysis (ISPA)*. Pula, HRV, Juni 2001, S. 182–187

[Wan18] Wang, Chi-Feng: *A Basic Introduction to Separable Convolutions*. https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728. Version: August 2018

[WC07] Wong, Shu-Fai ; Cipolla, Roberto: Extracting Spatiotemporal Interest Points using Global Information. In: *Proceedings of the 2007 International*

Conference on Computer Vision (ICCV). Rio de Janeiro, BR, Oktober 2007, S. 1–8

[Wer74] Werbos, Paul J.: *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Harvard University, PhD Thesis, 1974

[WKSL13] Wang, Heng ; Kläser, Alexander ; Schmid, Cordelia ; Liu, Cheng-Lin: Dense Trajectories and Motion Boundary Descriptors for Action Recognition. In: *International Journal of Computer Vision* 103 (2013), Mai, Nr. 1, S. 60–79

[WRKS16] Wei, Shih-En ; Ramakrishna, Varun ; Kanade, Takeo ; Sheikh, Yaser: Convolutional Pose Machines. In: *Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Las Vegas. NV, Juni 2016, S. 4724–4732

[WS13] Wang, Heng ; Schmid, Cordelia: Action Recognition with Improved Trajectories. In: *Proceedings of the 2013 IEEE International Conference on Computer Vision (ICCV)*. Sydney, AU, Dezember 2013, S. 3551–3558

[WXW+16] Wang, Limin ; Xiong, Yuanjun ; Wang, Zhe ; Qiao, Yu ; Lin, Dahua ; Tang, Xiaoou ; Van Gool, Luc: Temporal Segment Networks: Towards Good Practices for Deep Action Recognition. In: Leibe, Bastian (Hrsg.) ; Matas, Jiri (Hrsg.) ; Sebe, Nicu (Hrsg.) ; Welling, Max (Hrsg.): *Proceedings of the 14th European Conference on Computer Vision (ECCV)*. Amsterdam, NL : Springer International Publishing, Oktober 2016, S. 20–36

[YR11] Yang, Yi ; Ramanan, Deva: Articulated pose estimation with flexible mixtures-of-parts. In: *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Colorado Springs, CO, Juni 2011, S. 1385–1392

[YR13] Yang, Yi ; Ramanan, Deva: Articulated Human Detection with Flexible Mixtures of Parts. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)* 35 (2013), Dezember, Nr. 12, S. 2878–2890

[ZFB12] Zuffi, Silvia ; Freifeld, Oren ; Black, Michael J.: From Pictorial Structures to deformable structures. In: *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Providence, RI, Juni 2012, S. 3546–3553

[Zhu16] ZHU, Aichun: *Articulated human pose estimation in images and video*, Troyes, PhD Thesis, 2016

[ZWN+17] ZHANG, Shugang ; WEI, Zhiqiang ; NIE, Jie ; HUANG, Lei ; WANG, Shuang ; LI, Zhen: A Review on Human Activity Recognition Using Vision-Based Method. In: *Journal of Healthcare Engineering* 2017 (2017)

[ZZD13] ZHANG, Weiyu ; ZHU, Menglong ; DERPANIS, Konstantinos G.: From Actemes to Action: A Strongly-Supervised Representation for Detailed Action Understanding. In: *Proceedings of the 2013 IEEE International Conference on Computer Vision (ICCV)*. Portland, OR, Dezember 2013, S. 2248–2255