Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Vetle Gustav Birkeland

Monocular Action Classification

Master's thesis in Computer Science
Supervisor: Kerstin Bach
March 2022

NTNU
Norwegian University of
Science and Technology

Vetle Gustav Birkeland

Monocular Action Classification

Master's thesis in Computer Science
Supervisor: Kerstin Bach
March 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

Abstract

Exergames are games that use elements like rewards or challenges from video games to get people to exercise. The use of exergames has been showing promise in physical rehabilitaion by helping patients do their exercises without the constant presence of a physician. In physical rehabilitation it is important that exercises are executed correctly in order to achieve a faster and better recovery. An exergame that can use an camera to monitor and provide real time feedback to patients undergoing treatment would allow for more flexibility for both patient and physician, both in terms of time commitment and location.

A vital part of such an exergame is the ability to recognize and classify human actions from videos. In this thesis three models are created that can perform classification of movements: two XGBoost models, with and without feature selection, and a CNN model. A data set of videos containing examples of exercises related to physical therapy is used to train the models. The video data is first converted into a data set of joint positions by using the freely available, state-of-the-art pose estimator called OpenPose. The resulting data set is used to train the three models. Experiment results all show that the models manage to classify the different exercises correctly. The XGBoost models both score higher and are more consistent than the CNN-based model, with the feature selected model showing a significant decrease in training time over the other two. All models show similar difficulties with separating exercises that are very similar to one another.

Sammendrag

Treningsspill er spill som bruker aspekter som belønninger eller utfordringer fra videospill for å få folk til å trene. Bruk av treningsspill har vist potensiale innenfor fysisk rehabilitering ved å hjelpe pasienter med å gjøre øvelsene sine uten konstant tilstedeværelse fra en lege. Når det kommer til fysisk rehabilitering er det viktig å gjøre øvelse på korrekt vis for å oppnå rask og god bedring. Et treningsspill som kan bruke et kamera for å observere og tilby sanntidstilbakemelding til pasienter under behandling vil kunne gi økt fleksibilitet for både pasient og lege, både med tanke på bruk av tid og sted.

En viktig del i et slikt treningsspill er evnen til å gjenkjenne mennesker og klassifisere handlingene deres ved hjelp av video. I denne masteroppgaven blir det laget tre modeller som kan klassifisere handlinger: to XGBoost-modeller, med og uten variabelvalg (feature selection), og en CNN-modell. Et datasett bestående av videoeksempler på øvelser rettet mot fysisk rehabilitering blir brukt til å trene modellene. Videoene blir først konvertert over til et datasett bestående av leddposisjoner ved hjelp av det fritt tilgjengelige, state-of-the-art positurestimeringssystemet OpenPose. Dette leddposisjons-baserte datasettet blir brukt for å trene de tre modellene. Gjennomførte eksperimenter viser at alle modellene klarer å klassifisere de ulike øvelsene riktig. XGBoost-modellene scorer begge høyere og er mer konsistente enn CNN-modellen. XGBoost-modellen med variabelvalg trener i tillegg betydelig fortere enn de to andre. Alle modellene viser lignende tendenser til problemer med å skille øvelser som er svært like fra hverandre.

Acknowledgements

I would like to thank my supervisor Kerstin Bach for her guidance, availability, help and patience during the process of writing this thesis. I would also like to thank Emanuel Alexander Lorenz and his supervisors Xiaomeng Su and Nina Skjæret Maroni for sharing their data set with me and letting me use it in my thesis. Further, I want to thank Trine Marie L'Abée-Lund and Henning Normann for the frequent follow-ups that helped me get and keep going, and for their help with taking the photos used in Appendix A.

Contents

Figures

Tables

Chapter 1

Introduction

Recognizing a person's movements and classifying them, called action classification, has applications in a variety of areas, such as physical therapy, rehabilitation, animation and controlling virtual avatars in video games. The capturing of this information is traditionally done with specialized equipment such as cameras with depth capturing capabilities and motion capture camera setups requiring multiple cameras while subjects wear special marker suits. Due to the need for specialized equipment or setups, these methods are both expensive and inaccessible to the average consumer. A system that can extract this information and perform a classification using a single consumer grade RGB camera, for example a web camera, would make action classification accessible to a much wider array of applications. One area that would benefit greatly from being able to use standard RGB cameras is exergaming, the use of serious games for exercise purposes. A computer program that can provide proper feedback and supervision for people going through physical therapy or other training programs would benefit both the user and the professionals that would otherwise guide them manually.

In this thesis a data set of films capturing movements related to physical rehabilitation exercises are used to create human action classifiers. OpenPose (Cao et al., 2019) is used to perform pose estimation in order to extract the joint locations of the films' subjects. The joint coordinates are used as input for training a Convolutional Neural Network model and two Extreme Gradient Boost models (with and without feature extraction) that classify the various exercise types.

1.1 Thesis Goals and Research Questions

The goal of this thesis is to develop a framework that can perform action classification on monocular video using machine learning methods based on state of the art approaches. To complete this goal current state of the art methods for pose estimation and action recognition are described. A fitting pose estimation model is chosen to be used as part of the framework, its results are attempted reproduced. Promising methods for action recognition are also chosen, and models are created

based on these. The models are then evaluated, and strengths and weaknesses are pointed out and compared.

Aim of the thesis *To create a system that is able to classify human actions from monocular RGB video.*

To achieve this aim, we formulate goals and define research questions that are addressed in this thesis:

Goal 1 *To describe existing state of the art approaches and reproduce their results.*

Research question 1.1 *What is the state of the art in research on human pose estimation and action classification systems?*

The current state of the art in pose estimation and action classification is explored. There are two main goals of this exploration: to find a promising pose estimation candidate for use as part of this thesis' framework, and to identify methods that show promise on the task of action recognition.

Research question 1.2 *Are reported results from state of the art methods reproducible?*

The data used to train action classifiers is created using an existing pose estimator. If this model does not perform as well as the authors claim it does it has the potential to undermine any results that are derived from or depend on it. It is therefore of great interest to be able to reproduce the model's reported performance.

Goal 2 *To train a model capable of classifying human actions*

Research question 2.1 *Can a model be trained to classify human actions*

The use of computer programs like exergames has great potential in field like physiotherapy and rehabilitation. Motivating users while simultaneously providing feedback on their exercises largely independent of human physicians allows for greater freedom and flexibility in treatments. For a computer program to be able to provide these services it must be able to identify classify human actions. The creation of a model that can do this is a core aspect of this thesis.

Research question 2.2 *What improvements can feature selection yield*

Feature selection is the process of reducing the amount of features used to create a model. This can have several upsides, like shortening training time, simplifying the resulting model and reducing variance. This is a potential source of performance gains for a model which is of interest to look into. In this thesis feature selection will be performed for an Extreme Gradient Boost classifier to see if it yields any benefits.

1.2 Research Methods

To get an overview over the state of the art in the field of pose estimation and action recognition a literature review is performed. This process begins with a literature search, where relevant literature from the field and state of the art methods are identified. This gives an understanding on previous work that has been done and what is relevant for further work. The findings from the literature search are outlined in section 2.7.

The purpose of this thesis is to answer the research questions posed in section 1.1. To that end, a number of experiments are designed and executed based on findings from the literature review. There are two types of experiments being performed in this thesis. The first type is an experiment where we try to reproduce the results of an existing model. The second type consist of experiments that create new models. In both cases results are gathered and used to give an evaluation of the outcome of the experiment.

1.3 Thesis Structure

Chapter 2: Background and Related Work In this chapter relevant background theory is outlined, as well as related work.

Chapter 3: Method In this chapter the methods used to create the dataset and the motion classifiers are explained.

Chapter 4: Experiments and Results In this chapter the results of the experiments are reported.

Chapter 5: Discussion In this chapter results are compared and evaluated, and research questions are answered.

Chapter 6: Conclusion In this chapter the findings are summarized and potential further work is discussed.

Chapter 2

Background and Related Work

2.1 Pose Estimation

Human pose estimation is the task of estimating one or more humans' positions in 2D or 3D space from a video or image input (Y. Chen et al., 2020). In 2D pose estimation a system needs to provide some 2D representation of a 3D human being, for example in the form of joint coordinate values for each joint it is tasked with localizing in an image. The resulting points and the connections between them will yield a 2D skeletal representation. The exact amount of points and the connections between them vary somewhat between different datasets and estimators, but typical numbers for the amount of joints vary from 10-30. An example of one configuration used by the COCO keypoint evaluation task[1]is seen in 2.1. There are many challenges to take into account when designing a pose estimation system, such as truncation of the subject, occlusion of the subject by other objects, self-occlusion by the subject and the many different articles of clothing subjects wear.

In 3D pose estimation each joint location is defined in 3D space. The task faces the same challenges as its 2D counterpart with the added difficulty of needing to estimate depth information from RGB input, which is 2D. This can be done directly using an end-to-end 3D estimation technique, or a 2D pose estimation method can be used as a base which is then brought up into the third dimension (Martinez et al., 2017).

2.2 Monocular Action Classification

Action classification is the task of recognizing an action and correctly classifying it, e.g. seeing a video of a jumping human and correctly classifying it as a case of jumping. While humans see movement as a continuous change of position, a video of the same movement will be seen by a computer as a series of still images. In other words, action classification is a matter of recognizing a certain action

[1]https://cocodataset.org/#keypoints-2020

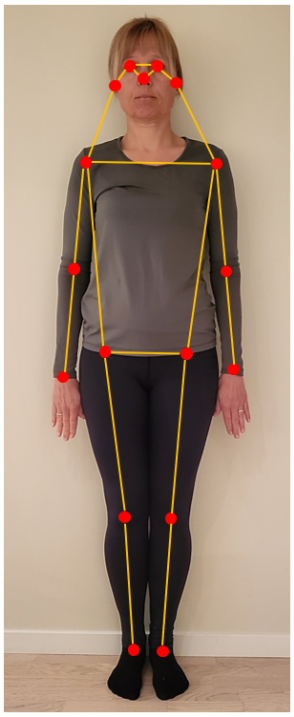


Figure 2.1: A skeleton representation with 18 joints used by the COCO dataset for their keypoint detection task.

from a series of poses in succession. Many action classification models therefore build upon the results of pose estimation or performs pose estimation as part of the action classification system (Luvizon et al., 2018; Nie et al., 2015; Yan et al., 2018). Accurate estimates of poses gives better data for a classifier, yielding better classification results. Making a human action classification system usable in a consumer or home setting poses challenges unique from those found in a lab or studio setting. Chief among these is the requiring the system to run using input from a single camera which can be of varying quality with regards to aspects such as aspect ratio, resolution, color, frame rate, etc. There can also be differences in image quality not directly caused by the camera itself, such as the quality of the lighting of the subject and surroundings, dirt on the camera lens or clothes obscuring the human shape. Restricting the system to only using one camera can also lead to problems with occlusion of body parts by objects in the scene or with the subject's own body occluding itself. All of these are "in the wild" problems that are easy to circumvent/prevent in a lab setting, but that a system nonetheless need to be able to handle.

2.3 Exergaming

Exergaming is the use of gamification, the usage of elements from (video) gaming such as challenges and reward systems, to encourage people to exercise and make it easier to adhere to a training regimen (Vonstad et al., 2021). This is achieved through the use of serious games, i.e. games made for a specific purpose other than leisure (Wiemeyer and Kliem, 2011). An area in which exergaming shows promise is the field of physical rehabilitation. An important element of physical rehabilitation is the use of specific exercises to meet a patient's needs. Making sure that the exercises are performed correctly is important for faster and better treatment, which has traditionally best been ensured through supervised exercise programs. One of the advantages of using exergaming for rehabilitation is that a game can provide guidance to patients without the need for a medical professional. For a game to provide proper guidance it must be able to identify exercises that a patient is supposed to perform, and provide some reward when these are performed correctly. What constitutes "correct" execution of a given exercise will wary from person to person. A system that can account for a patient's unique needs in order to provide a customized program would be a powerful tool for the field of physical rehabilitation. The biggest benefit would come from a system that can do this without the user needing specialized equipment, such as cameras able to capture depth information, or travel to a separate location (Vonstad et al., 2020). It is desirable to develop a system that can run on hardware available to most consumers and that does not require a large footprint in the patient's home. This thesis will use machine learning approaches to create models that can recognize a number of different exercises using a single RGB camera, such as a standard web camera found in many personal computers.

2.4 Neural Networks

2.4.1 Artificial Neuron

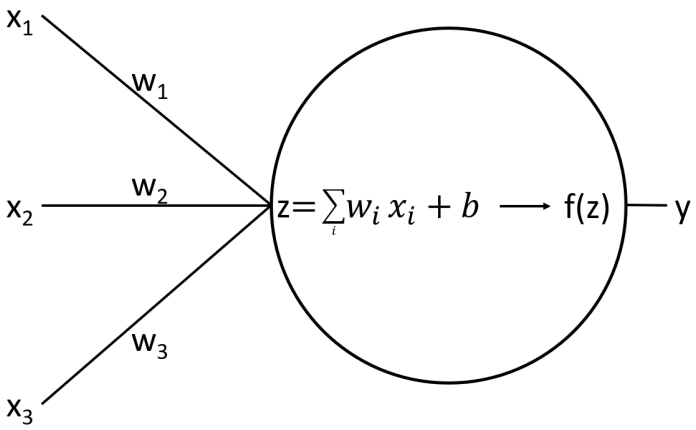


Figure 2.2: An artificial neuron. Inputs are weighted and a bias is added before the activation function is applied.

The basic building block of neural networks is the artificial neuron. Artificial neurons are inspired by biological neurons, like the ones found in the human brain (Mitchell, 1997). The task of the neuron is to take some number of inputs and provide a single output based on these. Each input x_i has a weight w_i associated with it, see figure 2.2 (Nielsen, 2015). In order to calculate its output the neuron will first calculate the sum of the inputs, with each input weighted by its corresponding weight.

$$\sum_{i=1}^{n} w_i x_i \tag{2.1}$$

The weights act as a measure of importance for each of the neuron's inputs. An input with a large weight, positive or negative, will have a greater impact on the final output value than the same input with a smaller weight. A bias b is then added to the weighted sum to get the weighted input z.

$$z = \sum_{i=1}^{n} w_i x_i + b \tag{2.2}$$

An activation function is then applied to the weighted input to get the neuron's final output, its activation.

$$a = f_a(z) = f_a(\sum_{i=1}^{n} w_i x_i - b) \tag{2.3}$$

The term f_a is the neuron's activation function. The activation function acts as the neurons mapping from weighted input to an output. An example of an activation function is the sigmoid function.

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

This function is illustrated in figure 2.3. The neuron's activation function determines how strong of an output the neuron has for a given input. It also dictates the possible range of output values. using the sigmoid function, for example, will constrain the neuron's output to fall between 0 and 1. A non-linear activation function will also introduce non-linearity to the neuron. This is especially useful when creating networks using many neurons. Since any function that is a combination of only linear function will itself be a linear function, this source of non-linearity is necessary if we are ever to represent non-linear functions. Looking at the sigmoid activation function, one can also see why a bias term is useful. If all of the neuron's inputs are 0 the weighted sum will also add up to 0, in which case the final output will be 0.5. Through the use of a bias this does not have to be the case. As shown in 2.4 changing the bias value shifts the activation function, allowing the neuron to map inputs to outputs it otherwise could not.

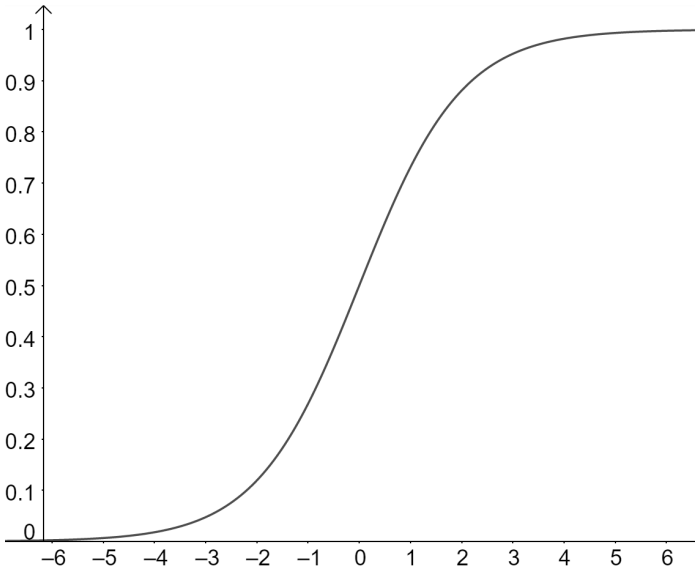


Figure 2.3: The Sigmoid activation function

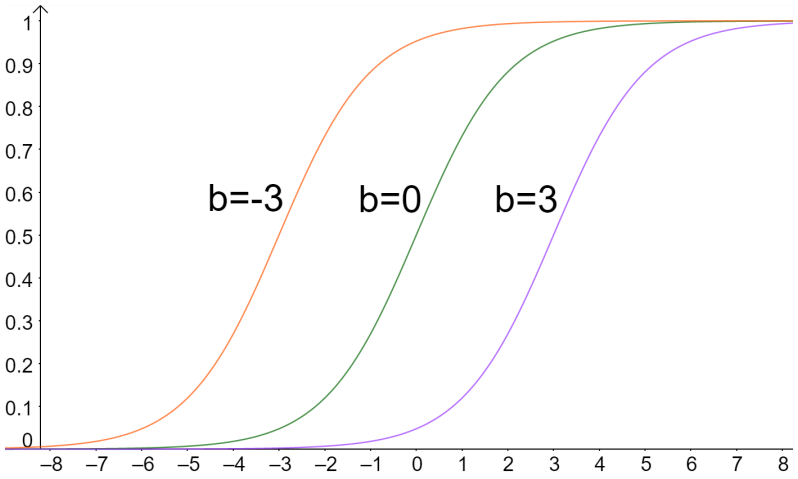


Figure 2.4: The effect of bias on neuron output.

2.4.2 Feedforward Neural Networks

By arranging neurons into groups and connecting these groups to one another, a neural network is formed (Russell and Norvig, 2010). While neural networks can have many different configurations they generally follow a general structure of an input layer, an output layer, and some number of hidden layers in between (Goodfellow et al., 2016; Nielsen, 2015). This general structure can be seen in figure 2.5. Each layer is connected by having the outputs from neurons in one layer act as the inputs for the neurons in the following layer. Since all neurons in one layer are connected to all neurons in the next, these layers are called fully-connected layers. When an input is applied by the input layer, the nodes in the first hidden layer will update their outputs accordingly. Their outputs will then do the same to the next layer and in this way the information from the input layer will propagate through the network until the output nodes update their outputs. This type of network where the information only flows one way is called a feedforward network.

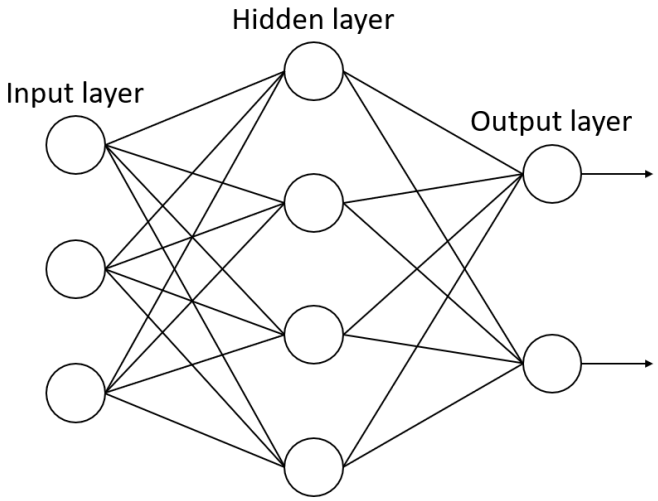


Figure 2.5: A neural network with an input, hidden and output layer.

2.4.3 Loss and Gradient Descent

The goal of a neural network is to give a good estimate or prediction given some input. This means finding some set of weights and biases that will transform inputs into good outputs. To do this a measure of "correctness" is needed, called a loss function (Goodfellow et al., 2016; Nielsen, 2015). An example of a loss function is mean squared error (MSE).

$$MSE(w,b) = \frac{1}{2n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{2.5}$$

Here n is the total number of inputs, y_i is the desired output for input i, $\hat{y}_i$ is the neural network's output from input i, and w and b indicates all the weights and biases of the network in question. The output of the loss function is called the loss. Looking at the formula, one can see that the loss approaches 0 if the network's outputs are all close to the desired output. The task of the learning process is therefore to minimize loss, which is done through a process called gradient descent (Mitchell, 1997). Formula 2.6 shows how changes to the parameters impact the MSE.

$$\triangle MSE \approx \frac{\partial MSE}{\partial w_1}\triangle w_1 + ... + \frac{\partial MSE}{\partial w_k}\triangle w_k + \frac{\partial MSE}{\partial b_1}\triangle b_1 + ... + \frac{\partial MSE}{\partial b_l}\triangle b_l \quad (2.6)$$

The partial derivatives contains information about how the function as a whole changes as each of the individual parameters are changed. The magnitude of the partial derivatives indicate how much of an impact a given change in a parameter will have on the MSE, whereas the parity of the partial derivatives indicate whether the change leads to an increase or decrease in overall MSE. The idea behind gradient descent is to use the information contained in the gradient vector to change each parameter in such a way as to make changes in weights and biases that will lead to a negative $\triangle MSE$ (Russell and Norvig, 2010). The gradient vector ∇MSE is defined as a vector containing the partial derivatives of the loss function.

$$\nabla MSE \equiv (\frac{\partial MSE}{\partial w_1}, ..., \frac{\partial MSE}{\partial w_k}, \frac{\partial MSE}{\partial b_1}, ..., \frac{\partial MSE}{\partial b_l})^T \quad (2.7)$$

$\triangle p$ is a vector where each element indicates the change to a given parameter.

$$\triangle p \equiv (\triangle w_1, ..., \triangle w_k, \triangle b_1, ..., \triangle b_l)^T \quad (2.8)$$

In both ∇MSE and $\triangle p$ T is the transpose operation. $\triangle MSE$ can now be rewritten using formula 2.7 and 2.8 to get $\triangle MSE \approx \nabla MSE \cdot \triangle p$. $\triangle p$ is set to the negative of the gradient multiplied by a small, positive parameter known as the learning rate.

$$\triangle p = -\alpha \nabla MSE \quad (2.9)$$

$\triangle MSE$ can now be written entirely in terms of its gradient.

$$\triangle MSE \approx -\alpha \nabla MSE \cdot \nabla MSE = -\alpha \|\nabla MSE\|^2 \quad (2.10)$$

Since $\|\nabla MSE\|^2 \geq 0$, it is guaranteed that $-\alpha\|\nabla MSE\|^2 \leq 0$. This means that changing the parameters according to formula 2.9 results in a reduction of the loss. This can be used to create update rules for the individual weights and biases.

$$w_k \leftarrow w_k - \alpha \frac{\partial MSE}{\partial w_k} \quad (2.11)$$

$$b_l \leftarrow b_l - \alpha \frac{\partial MSE}{\partial b_l} \tag{2.12}$$

The logic behind gradient descent is to that the gradient points in the direction that will increase the loss the most. By going in the opposite direction, which is where the negative sign in formula 2.9 comes in, a step can be taken in the direction that most reduces the loss. The idea is to take many small steps in this direction until the loss can be reduced no further, at which point a minimum has been reached. At each step all parameters are updated according to the rules in formulas 2.11 and 2.12. The learning rate introduced in formula 2.9 is included to keep the step size small. If the steps at each iteration are too large it can cause the process to overshoot the minimum and diverge, see figure 2.6. The learning rate should not be too small, however, as this can cause the process to take longer than necessary.

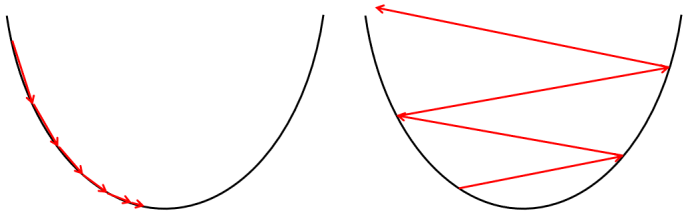


Figure 2.6: Conceptualization of how gradient descent can fail to locate a minimum if the step size is too large. On the left the steps are small enough for the process to approach and eventually reach the minimum. The figure on the right shows a process that is diverging; the steps are too large at each iteration, and the process will not reach the minimum.

2.4.4 Backpropagation

Gradient descent is wholly dependent on calculating the gradient of the loss function, seen in equation 2.7 (Nielsen, 2015). In the case of neural networks this is done using a process called backpropagation (Goodfellow et al., 2016; Mitchell, 1997). To make it easier to read without getting bogged down by indices, some notation is needed. w^l_{jk} refers to the weight associated with the connection between neuron k in layer $l-1$ to neuron j in layer l, b^l_j refers to the bias of neuron j in layer l, and a^l_j refers to the activation of neuron j in layer l, from formula 2.3. Figure 2.7 shows examples of the notation in use.

With the sigmoid function as activation function formula 2.3 can be rewritten using the new notation.

$$a^l_j = \sigma(\sum_k w^l_{jk} a^{l-1}_k + b^l_j) \tag{2.13}$$

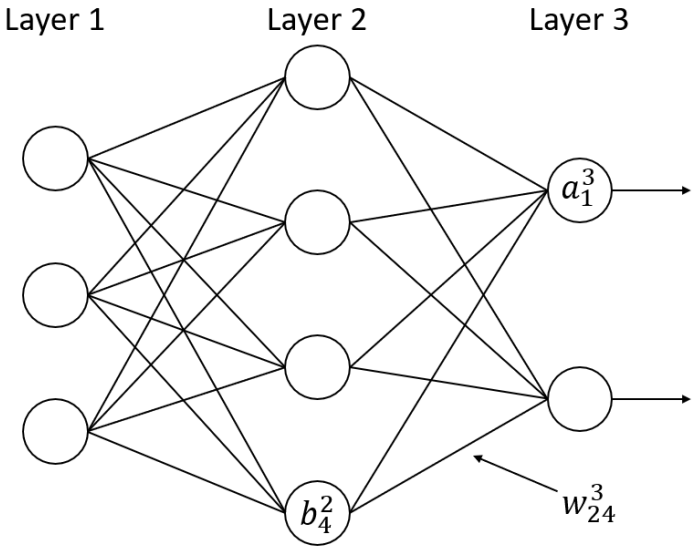


Figure 2.7: Illustrative example showing how biases, activations and weights are referred to using the notation in section 2.4.4.

The sum is summing over all k neurons in layer $l-1$. Defining w^l as a matrix containing the weights connecting layer l and $l-1$, b^l as a vector containing the biases in layer l, and a^l as a vector containing the activations of layer l, formula 2.13 can be rewritten in vectorized form.

$$a^l = \sigma(w^l a^{l-1} + b^l) \tag{2.14}$$

One more useful term when talking about backpropagation is the weighted input defined in equation 2.2, also rewritten in vectorized form.

$$z^l \equiv w^l a^{l-1} + b^l \tag{2.15}$$

Backpropagation begins by calculating an error term for each neuron in the network.

$$\delta_j^l \equiv \frac{\partial MSE}{\partial z_j^l} \tag{2.16}$$

The intuition here is similar to the one for equation 2.6: the partial derivatives $\frac{\partial MSE}{\partial z_j^l}$ says something about how big of an impact a small change to the weighted input (which is a function of weights and bias) of the neuron in question has on the total loss (Russell and Norvig, 2010). If δ_j^l is large it means that the neuron is comparatively far from being optimal, in turn meaning this is a neuron that changes should be made to. A small δ_j^l means changes to the neuron will not matter that much, and it is probably close to being optimal. Focusing only on the error in the output layer, the error is written as $\delta_j^L = \frac{\partial MSE}{\partial z_j^L}$, where the capital L

indicates that this is the output layer. This is rewritten using the chain rule from multivariable calculus.

$$\delta_j^L = \sum_{k \in L} \frac{\partial MSE}{\partial a_k^L} \frac{\partial a_k^L}{\partial z_j^L} \tag{2.17}$$

When $j \neq k$ the term $\frac{\partial a_k^L}{\partial z_j^L}$ will vanish, allowing equation 2.17 to be simplified to $\delta_j^L = \frac{\partial MSE}{\partial a_j^L} \frac{\partial a_j^L}{\partial z_j^L}$. Per formula 2.14 $a_j^L = \sigma(z_j^L)$, meaning the term $\frac{\partial a_j^L}{\partial z_j^L}$ can be written as $\sigma'(z_j^L)$. Using this δ_j^L can be further simplified into an equation for the error in the output layer.

$$\delta_j^L = \frac{\partial MSE}{\partial a_j^L} \sigma'(z_j^L) \tag{2.18}$$

The next piece of backpropagation is a formula that can calculate δ_j^l if the errors of the next layer are known. The starting point is once again equation 2.16. The chain rule is used to rewrite δ_j^l in terms of δ_k^{l+1}.

$$\delta_j^l = \sum_k \frac{\partial z_k^{l+1}}{\partial z_j^l} \delta_k^{l+1} \tag{2.19}$$

Using differentiation one can see that $\frac{\partial z_k^{l+1}}{\partial z_j^l} = w_{kj}^{l+1} \sigma'(z_j^l)$, which is substituted into 2.19 to yield the desired equation.

$$\delta_j^l = \sum_k w_{kj}^{l+1} \delta_k^{l+1} \sigma'(z_j^l) \tag{2.20}$$

Using equation 2.18 first to find the errors of the output layer L, equation 2.20 can then be used to find the errors of layer $L-1$. Knowing these errors equation 2.20 can be used again to find the errors in layer $L-2$ and so on, propagating the errors backwards through the network until every neuron's error is known. This is where the method gets its name. The only steps that remain are to use the errors to calculate partial derivatives for the weights and biases, $\frac{\partial MSE}{\partial w_{jk}^l}$ and $\frac{\partial MSE}{\partial b_j^l}$.

To find the equation for $\frac{\partial MSE}{\partial b_j^l}$, first rewrite it using the chain rule.

$$\frac{\partial MSE}{\partial b_j^l} = \sum_{i \in l} \frac{\partial MSE}{\partial z_i^l} \frac{\partial z_i^l}{\partial b_j^l} \tag{2.21}$$

Since $\frac{\partial z_i^l}{\partial b_j^l}$ vanishes when $i \neq j$ the summation over l disappears.

$$\frac{\partial MSE}{\partial b_j^l} = \frac{\partial MSE}{\partial z_j^l} \frac{\partial z_j^l}{\partial b_j^l} = \frac{\partial MSE}{\partial z_j^l} \frac{\partial \sum_{k \in l-1} w_{jk}^l a_k^{l-1} + b_j^l}{\partial b_j^l} \tag{2.22}$$

Since the weights w^l and activations a^{l-1} does not depend on b^l_j the term $\frac{\partial \sum_{k \in l-1} w^l_{jk} a^{l-1}_k + b^l_j}{\partial b^l_j}$ is equal to 1. Making this substitution and remembering the definition of δ^l_j from equation 2.16 yields the equation for partial derivative of biases.

$$\frac{\partial MSE}{\partial b^l_j} = \frac{\partial MSE}{\partial z^l_j} \cdot 1 = \delta^l_j \tag{2.23}$$

The only thing missing now is to find $\frac{\partial MSE}{\partial w^l_{jk}}$. The first step is once again to use the chain rule to rewrite the term.

$$\frac{\partial MSE}{\partial w^l_{jk}} = \sum_{i \in l} \frac{\partial MSE}{\partial z^l_i} \frac{\partial z^l_i}{\partial w^l_{jk}} \tag{2.24}$$

Since z^l_i only depends on w^l_{jk} when $j = i$, the summation over l disappears.

$$\frac{\partial MSE}{\partial w^l_{jk}} = \frac{\partial MSE}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} = \frac{\partial MSE}{\partial z^l_j} \frac{\sum_{i \in l-1} w^l_{ji} a^{l-1}_i + b^l_j}{\partial w^l_{jk}} \tag{2.25}$$

Since w^l_{ji} only depends on w^l_{jk} when $k = i$, $\frac{\sum_{i \in l-1} w^l_{ji} a^{l-1}_i + b^l_j}{\partial w^l_{jk}} = a^{l-1}_k$. Making this substitution and also using the definition of δ^l_j gives the equation for the partial derivatives of the weights.

$$\frac{\partial MSE}{\partial w^l_{jk}} = \frac{\partial MSE}{\partial z^l_j} a^{l-1}_k = \delta^l_j a^{l-1}_k \tag{2.26}$$

This is the last piece of backpropagation. After using equations 2.18 and 2.20 to calculate the errors in the whole network, equations 2.23 and 2.26 can use the errors to calculate the partial derivatives needed to perform gradient descent as described in section 2.4.3.

2.4.5 Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a variant of neural networks inspired by biological eyes. Their name comes from an operation known as a convolution that such networks utilize to great effect on grid-like data, e.g. the grid of pixels that form an image (Goodfellow et al., 2016; Nielsen, 2015). There are two layers that are characteristic to CNNs: convolutional layers and pooling layers. The convolutional layer is based around a filter (also called kernel), which consists of weights arranged in a matrix of size (m, n, d), as well as a bias. The d in the matrix dimensions refers to the depth of the input. In the case of an RGB image the depth would be 3, one for each color channel. For the purposes of this section, the depth will be ignored, as it does not change the concepts at hand. The filter

connects $m \times n$ adjacent neurons in one layer to a single neuron in the next layer, illustrated in figure 2.8.

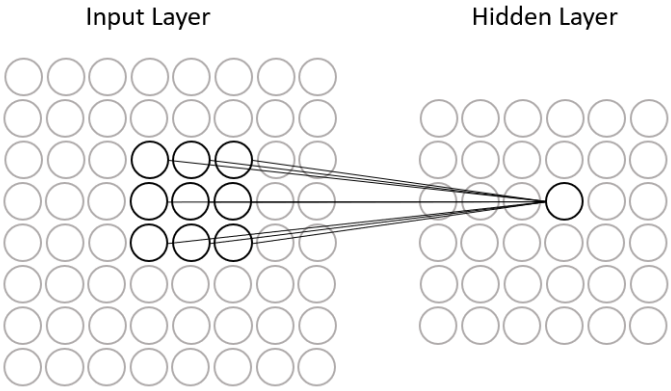


Figure 2.8: The filter connects a 3×3 region in the input layer to a single neuron in the hidden layer.

Beginning in one corner of the grid, for example the top left, it slides across the image from left to right, moving some number of columns to the right each time. When it reaches the right side of the grid it moves down some number of rows and slides from left to right again. This repeats until the bottom right of the grid is reached. How many rows or columns the filter moves at each step is called the stride length, and is a hyperparameter used to tune CNN models. The activation of the neuron at position (k, l) in the resulting hidden layer is conceptually the same as that for a neuron in a feedforward network.

$$a = f_a(b + \sum_{i=0}^{m} \sum_{j=0}^{n} w_{i,j} a_{k+i,l+j}) \tag{2.27}$$

f_a is the neuron's activation function, b is the bias, $w_{m,n}$ is the weight matrix of the filter and $a_{x,y}$ is the activation of the input layer at position (x, y). This equation assumes a stride length of one for both columns and rows. Adjusting for larger values is simply a matter of modifying the indices for the term $a_{k+i,l+j}$ to account for this. The equation also shows that throughout this whole process of sliding across the grid of neurons the filter is using the same weights and bias at every step. This is key because filters detect features. A feature is some pattern in the input that causes the hidden neuron to activate, for example a line. When the filter moves across the image it is essentially detecting the presence of its particular feature. As a result, the hidden layer becomes a map depicting how instances of a particular feature are related to one another spatially. For this reason the output layer that results from running a filter across the some input is called a feature map. By applying different filters to the same input the convolutional layer creates several different feature maps. These feature maps can then be used as inputs for a new convolutional layer. The relatively simple features detected by an early layer

can be combined into more complex features in later layers, which combine into even more complex features further into the network. In this way lines and edges becomes simple shapes, which becomes complex shapes, which becomes objects, faces, people, etc.

An important difference between convolutional layers and the fully-connected layers described in section 2.4.2 is that the convolutional layer requires far fewer parameters. Fully connecting a $x \times x$ input to a $y \times y$ output would require $x^2 \times y^2$ weights in addition to y^2 biases for a total of $x^2y^2 + y^2$ parameters. As the size of the input increases, this number quickly becomes very large, easily numbering in the billions. A $z \times z$ filter only requires z^2 weights and a bias meaning a convolutional layer utilizing such filter will only need a total of $z^2 + 1$ parameters per feature map, independent of the size of the input. As the filter is usually quite small, typical values for z being in the single digits, this represents an enormous difference in the number of parameters, even when using hundreds of filters. This efficient use of parameters is a huge advantage for CNNs over fully-connected networks.

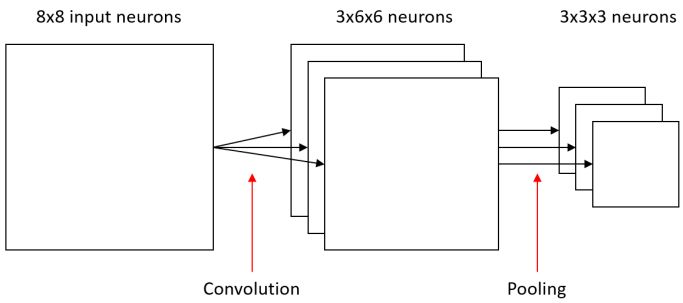


Figure 2.9: A convolutional layer outputting three feature maps, followed by a pooling layer halving the size of said feature maps.

Pooling layers are used to condense information contained in some input. They work similarly to the convolutional layer in that a filter of some size (m, n) slides across a grid, usually a feature map, according to some stride length, performing a pooling operation at each step. A typical size for a pooling filter is 2×2, with a stride of 2, which will result in halving the size of the input. They are usually used immediately following a convolutional layer to reduce the size of the feature maps output by the convolution, see figure 2.9. The main difference from the convolutional layer is in the operation being performed by the filter. Pooling does not involve weights and biases that are trained iteratively through some process. Instead, the operation is clearly defined when creating the network, and operates the same way at all times. Another difference is that a convolution layer takes some number of inputs d_i and outputs some number of feature maps d_o, where d_o is the number of filters in that layer. A pooling layer taking d_i feature map inputs will perform the exact same pooling operation on each one, outputting d_i outputs. The pooling does not change the number of feature maps, only their size. An example of a pooling operation is max-pooling. In max-pooling the maximum

activation among the neurons in the pooling filter is selected as the activation for the output neuron, see figure 2.10.

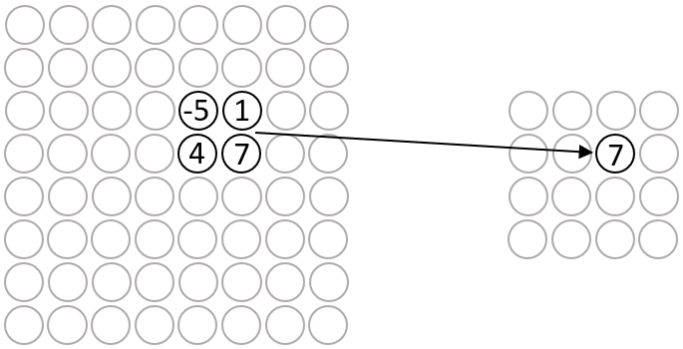


Figure 2.10: In max-pooling the output neuron outputs the same value as the maximum activation in its corresponding input region.

One benefit of pooling is to provide invariance to translations in the input, meaning that if the input is translated by some small amount, the pooled outputs do not change by a lot. This makes the model rely less on the exact position of a feature, and instead rely more on where features are located relative to one another. Another advantage of pooling is that it (usually) reduces the size of the input, reducing the amount of computation that needs to be done in later layers.

A general CNN structure involves alternating convolution and pooling layers before ultimately using some fully-connected layers at the end, see figure 2.11. Training a CNN is done in much the same way as a fully-connected network; an input is applied, propagated through the layers, loss is calculated, gradient descent using backpropagation is used to update weights and biases, repeat until desired performance is achieved (or it becomes clear desired performance will not be achieved).

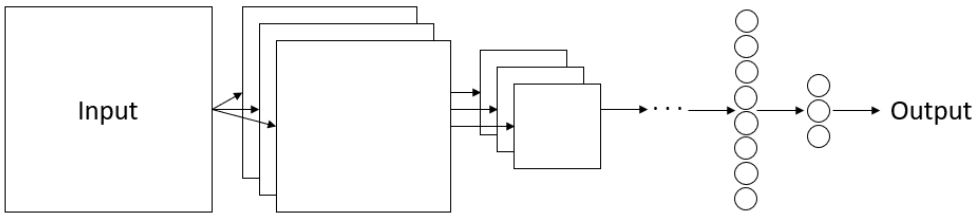


Figure 2.11: General structure for CNNs. The first part of the network consists of alternating convolution and pooling layers, represented by the three dots. At the end of the network there are some fully-connected layers before the final output.

2.5 Tree-Based Ensemble Methods

2.5.1 Decision Trees

Decision trees are structures that take some number of attributes as input, producing a single output through a series of tests (Mitchell, 1997). The decision making process begins at the root node, where one of the input attributes will be subject to a test. Then the branch corresponding to the result of the test is followed to the next node, where the process is repeated. This continues until a leaf node is reached, at which point the value of the leaf node is given as the decision tree's output. An example decision tree is depicted in figure 2.12.

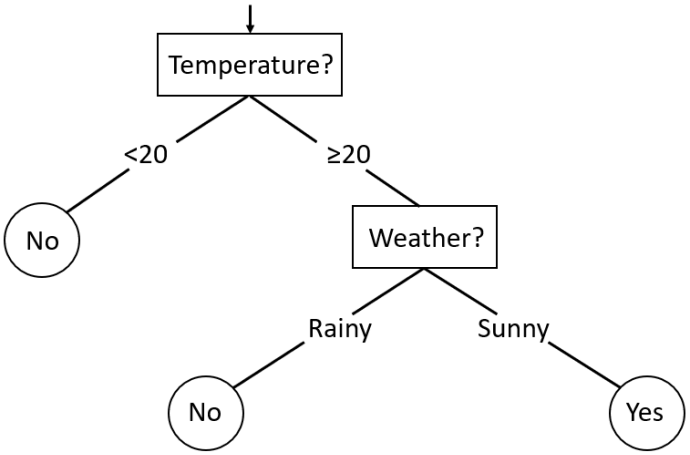


Figure 2.12: Example decision tree. The inputs are *temperature*, which will be a rational number, and *weather*, which can be *sunny* or *rainy*. The decision tree will determine whether one should go outside or not. The root node is marked with an arrow. The internal nodes are square, while the leaf nodes are circular.

Decision trees learn through a recursive greedy divide-and-conquer strategy (Russell and Norvig, 2010). Starting at the root node, determine which attribute will give the best split. Use this attribute as the test for the root and split the training data accordingly. Create a new node for each value this test can result in and sort the training data to the descendant nodes according to the test. At each of these nodes this process can be repeated in a recursive fashion. A leaf node will be created in three cases.

1. If the remaining examples after a split all correspond to the same output, create a leaf node with that output value.
2. If a set of examples after a split is empty, there are no examples for this combination of attribute values. Create a leaf node with a default value, for example the most common value among the examples in the parent node.
3. If all attributes have been used, but there are still examples corresponding to different outputs, create a leaf node with output set to the most common value of the remaining examples.

This process is dependent on some measure indicating which attribute will result in the best split. An example of such a measure is information gain, calculated using entropy.

2.5.2 Entropy and Information Gain

The entropy $E(S)$ of some collection S consisting of c classes is a measure of how impure or uneven S is (Mitchell, 1997; Russell and Norvig, 2010).

$$E(S) = \sum_{i=1}^{c} -p_i log_2 p_i \tag{2.28}$$

p_i is the proportion of the number of elements belonging to class i to the total number of elements in the set S. A collection of examples where all members belong to the same class will have an entropy of 0. If the members are equally split between c different classes the entropy will be $log_2 c$. For unequal splits the entropy will fall in between these extremes. By using entropy as a measure of "unevenness" in a set of data a method for deciding on a best attribute can be defined. The goal is to move from a single dataset into subsets where each entry of a given subset belongs to the same class. In other words, to go from one large dataset with $entropy > 0$, to some number of subsets each with $entropy = 0$. At each step the data should be split by using the attribute that will result in the largest reduction of entropy. Define $E(S|A)$ as the total entropy of the resulting n subsets after splitting a set S using attribute A.

$$E(S|A) = \sum_{i=1}^{n} p_t H(S_t) \tag{2.29}$$

p_t is the proportion of the number of elements in subset t to the total number of elements in S and $H(S_t)$ is the entropy of subset t. The reduction in entropy after using attribute A to split dataset S is called information gain, and is given by $IG(S,A)$.

$$IG(S,A) = H(S) - H(S|A) \tag{2.30}$$

The information gain acts as the measure of an attribute's impact on a set of data. When deciding which attribute to use at a node during the training process simply calculate $IG(S,A)$ for each attribute that is being considered, then choose the one with the highest value.

2.5.3 Ensemble Methods

The concept of ensemble learning involves using multiple different learners, an ensemble, to provide an output. Using decision trees as an example one could generate $T = 15$ decision trees. Given an input the 15 trees will provide their 10 outputs, which can then be combined into a single final output by means of

for example majority voting. The reason for combining multiple learners is that it will make misclassifications less likely (Russell and Norvig, 2010). Consider the situation where the 15-tree ensemble attempts a Boolean classification task, that is each input belongs to one of two classes. To misclassify an input would require at least 8 of the individual trees to get it wrong. The hope is that this is less likely to happen than a single learner making the mistake; the ensemble will suppress uncommon errors among the learners.

2.5.4 Boosting

One technique for creating an ensemble is called boosting. This method utilizes weighted data sets, in which each example used to train the learners also has a weight associated with it (Russell and Norvig, 2010). An example with a higher weight is deemed more important during learning, which the training algorithm will have to be modified to account for. Initially all the examples are weighted equally. An initial learner, for example a decision tree, is trained on the data as described in section 2.5.1. Assuming the first learner does not correctly classify every example, we want the next one to focus more on the misclassified examples. To do this the weight of each misclassified example is increased while the correctly classified examples have their weights decreased. This process repeats until T learners have been trained, where T is provided as input to the boosting algorithm.

2.5.5 Gradient Boosting

Gradient boosting is a boosting variant where an ensemble is created from weak learners, for example simple decision trees (Natekin and Knoll, 2013). A weak learner is a learner that performs slightly better than random guessing (Boehmke and Greenwell, 2020). Gradient boosting starts by creating a base estimator F_1 and fitting it to some training data x. Since this estimator is very basic it is probably not very accurate, so there is going to be some difference between its output $\hat{y}_1 = F_1(x)$ and the desired output y, called the residual. The next step is to fit a new estimator h_1 to the residual of the previous. The idea is that the contribution from h_1 will compensate for some of the error of F_1, and thereby reduce the residual. In order to limit how big the correction from h_1 is, a learning rate α is applied to it, which functions similarly to how it is used in section 2.4.3. In fact, the process is a form of gradient descent, which is where the "gradient" in "gradient boosting" comes from. The new estimator is then added to the old to create F_2. The steps can be combined into a simple four-step algorithm.

1. Calculate residuals from the difference between $\hat{y}_n = F_n(x)$ and y
2. Create a new estimator h_n
3. Fit h_n to residuals
4. Create new ensemble $F_{n+1} = F_n + \alpha h_n$

This is repeated until some mechanism tells the process to stop, for example due to the residuals becoming small enough or the process having met some up-

per limit for number of iterations. The result of the process is an ensemble F_b, consisting of b different estimators working together. Some important hyperparameters when creating models using gradient boosting with decision trees are the number of trees, learning rate and maximum tree depth. Without an upper limit to the number of trees that can be added to the ensemble, the final model is prone to overfitting. Overfitting is when a model has failed to generalize, and is instead memorizing the training data. The idea with this hyperparameter is to help prevent overfitting from occurring by stopping the training process after it has learned from the data, but before it begins to overfit. The learning rate has serves the same function as described in section 2.4.3. The depth of a decision tree is the largest number of splits between a leaf node and the root. The maximum tree depth enforces an upper limit on this quantity for each decision tree in the ensemble, which helps keep each tree simple. Deeper trees have the advantage of being able to capture interactions involving larger number of attributes, but they are both more computationally expensive and more prone to overfitting. Deciding on a good value for this hyperparameter is about finding a balance between these factors.

Extreme Gradient Boosting (XGBoost) is a gradient tree boosting system, seeking to provide an efficient and scalable implementation of the method (T. Chen and Guestrin, 2016). It provides features like parallell processing, the ability to use a variety of base learners and early stopping, and has implementations in several programming languages.

2.6 Parameter Search

Both CNNs and XGBoost models require tuning of hyper-parameters to achieve good results, or even do better than random noice at all. Manually tuning a model's hyper-parameters can be a very time consuming endeavour. Without some *a priori* knowledge about the specific domain or task at hand the process can quickly become a matter of trial and error using best practices that may or may not apply to said domain or task (Nielsen, 2015). Grid search is a method used to explore the hyper-parameter space at hand in order to find promising hyper-parameter values quicker than a manual process would allow. When performing a grid search a set of possible values is chosen for each parameter that is to be tuned. Then every combination of values across all the hyper-parameters are tested, and the combinations can be ranked by their performance (Bergstra and Bengio, 2012). While the method does not necessarily find the best values for the parameters, it will give a good indication of where in the hyper-parameter space the good combinations are. The main drawback of this method is that the number of parameter combinations grows exponentially with the number of hyper-parameters.

Random search explores the hyper-parameter space by generating parameter combinations through random sampling of values. For each parameter a range of possible values are provided to be used for said sampling. The search algorithm is also provided the number of combinations to be tested, so that it will have a

stopping condition. By narrowing or widening the parameter value ranges or by increasing or decreasing the number of combinations to be tested one can control how finely or coarsely the search space will be explored. The logic behind random search is that the rigidity of grid search causes it to miss important information. Consider a grid search that has a set of values $(v_1, ..., v_n)$ for some parameter p. No matter how many total combinations will be explored by the search, p will only ever take on n values. If the best values for p falls outside of exactly these n, the grid search will never find them. This can be mitigated by providing a more granular set of values, but doing so further amplifies the growth of the number of parameter combinations. The same parameter in a random search would have its values pulled from the range $v_1 < p < v_n$. This results in each combination being explored using a (probably) unique value for p. This makes the random search able to explore each parameter more thoroughly, even in cases where the total number of combinations is lower than an equivalent grid search.

2.7 Related Work

This section outlines existing approaches and methods for pose estimation and action recognition. These are closely related tasks, and the latter will often build upon the former. In both cases the input is usually visual data, i.e. images and videos. For the purposes of this thesis the focus will be on methods that utilize video input, since we have a data set of videos available to use.

Current state of the art pose estimation methods are generally based on deep learning models (Y. Chen et al., 2020). These can be divided into two general categories: regression-based or detection-based. Detection-based methods aim to give an approximation, using heatmaps to represent an approximate location of joint locations or body parts. Bulat and Tzimiropoulos, 2016 is one such method that uses a two-network architecture where the first network produces *part heatmaps* indicating probable locations for each body part. The second network refines the part heatmaps into confidence maps by narrowing down the probable locations into a final location for each joint. The final output of the model is this set of confidence heatmaps. Regression-based methods are characterized by mapping an input directly to the desired output, i.e. the output is a set of coordinates (Luvizon et al., 2018). One way of doing this is to have the model output location coordinates directly, such as in Toshev and Szegedy, 2014. This approach is limited by the fact that the regression method is often sub-optimal. A solution to overcome this weakness is to take a detection-based method and transform it into a regression-based method using soft-argmax (Luvizon et al., 2017). Converting heatmap representation into coordinate representation this way lets models retain the advantages of both representations (Y. Chen et al., 2020). This is used by Nibali et al., 2018 in their Margipose model in which 3D joint coordinates are calculated using three marginal heatmaps representing the xy, xz and yz dimensions, illustrated in figure 2.13. To obtain a coordinate representation of the position soft-argmax is used on these three heatmaps, converting the model from

detection to regression.

A common usage of heatmaps is the concept of *confidence heatmaps* (Y. Chen et al., 2020). A confidence heatmap is a heatmap indicating a particular joint's approximate position in an image. Each confidence heatmap only contains information about one joint, meaning models need to create one heatmap per joint for a full representation of a person. In Mehta et al., 2017 confidence heatmaps are used to give an initial 2D joint position, before this is lifted into 3D. Cao et al., 2019 uses one confidence maps per joint to indicate the joint locations of all of that particular joint, i.e., the confidence map for left shoulder contains locations of all left shoulders in the image. The joint location heatmaps are then combined with Part Affinity Fields (PAFs) to obtain a pose estimate. The PAFs are vector fields indicating how closely associated a pair of joints are with one another, allowing the system to connect the correct shoulder and elbow joints, for example. This approach allows the method to give pose estimates for multiple people in an image in one pass.

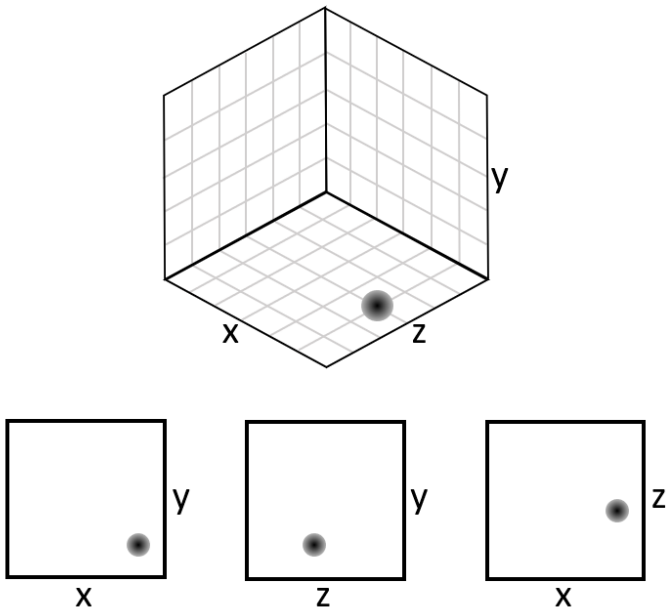


Figure 2.13: Multiple marginal heatmaps, each providing postitional information in two dimensions, can fully describe a location in three dimensional space.

Action recognition is the task of recognizing human actions from images or video. If the video contains only a single action that needs to be classified, it is called action classification. Action recognition methods can be roughly divided into two groups, depending on whether they are based purely on RGB inputs, or if they also utilize depth information (RGBD) (Zhang et al., 2019). Depth data carries with it several advantages over pure RGB, such as robustness with regards to changes in light conditions, environment and background, as well as enabling quick object segmentation. The main downsides comes from requiring special

depth cameras, leading to methods only performing well in specific environments as well as limiting their effective ranges. Current state of the art methods consists mostly of deep neural networks such as CNNs (Herath et al., 2016). A common factor for such action recognition systems is that they need some way to extract temporal information from their inputs (Zhu et al., 2020). This is difficult to obtain from raw video input. One technique to allow models to access the temporal information is through the use of optical flow (Horn and Schunck, 1981), which concerns patterns of motion in a scene due to the relative motion between said scene and its observer. In Simonyan and Zisserman, 2014 a two-stream neural network architecture is described. It consists of two separate networks whose outputs are combined to provide a prediction. One of the two networks utilizes optical flow to capture temporal information between frames in the input video to recognize motion, while the other is RGB-based and performs object recognition.

Recent developments in human action recognition involves using skeleton sequences (Zhang et al., 2019). Such methods use poses represented by skeletons as input instead of RGB or RGBD images or films. In Luvizon et al., 2018 a joint pose estimation and action recognition system is described. The pose estimation part of the system uses RGB images as its input and outputs coordinates from a heatmap through the use of soft-argmax, as described in section 2.7. The action recognition half of the system is further divided into two, with one part making predictions based on RGB images and the other making predictions based on time series of poses, each pose being represented by a skeleton of joint coordinates. This results in a two-part structure, similar in concept to Simonyan and Zisserman, 2014, where the two combine their predictions to produce a final output. Yan et al., 2018 describes an action recognition system based purely on skeleton sequences to make predictions. The skeletons are framed as spatial-temporal graphs with spatial edges connecting the joints in a skeleton to each other (i.e. the limbs) and temporal edges connecting joints across time steps (i.e. connecting a joint j at time step t_n to that same joint at time step t_{n+1}). A special graph convolutional network called a *Spatial-Temporal Graph Convolutional Network* was used to provide action predictions.

In this thesis models using skeleton sequences as input will be used as action classifiers to classify exercises related to physiotherapy. The skeletal information will be obtained by using OpenPose to acquire joint positions from videos of participants performing said excersises. Action classifier models will then be trained on this data.

2.8 Reproducibility of Results

Being able to reproduce results is important in order to verify research results. In the field of machine learning models are trained to do some task, tested on data and evaluated based on the results. The model that scores the highest on some metric can then be said to be the best for that task (Gundersen et al., 2022). This is, however, dependent on the reproducability of the results. If the results of the

highest scoring method are unable to be reproduced, confidence in the method is reduced. Gundersen et al., 2022 found that it is in fact almost impossible to reproduce the exact results of an experiment. Platform, processing unit, software versions, bugs, and the general stochasticity of machine learning models create so many points at which models created from the same code can begin to diverge from one another that any two such models are essentially bound to be different from one another. Instead of trying for a perfect reproduction of the results, the best that can realistically be done is to create a model that allows for the same overall conclusions to be drawn.

A common starting point when applying a method to new data or domains is to reproduce reported results to verify that the method was installed or implemented correctly, and that code was set up and run properly (Tatman et al., 2018). If results are inherently non-reproducible it can become a challenge just to get started with using an existing method. It is thus not only for scientific or academic purposes that reproducability is important, but also for any practical application of a machine learning method. Pineau et al., 2021 suggests four categories depending on whether data or code is the same or different when attempting to recreate experiment results, see figure 2.14. Under this system results are reproducible when a new experiment that is run using the same code and data as the original experiment manages to recreate the original results. It is this notion of reproducibility that will be used in this thesis when using existing methods.

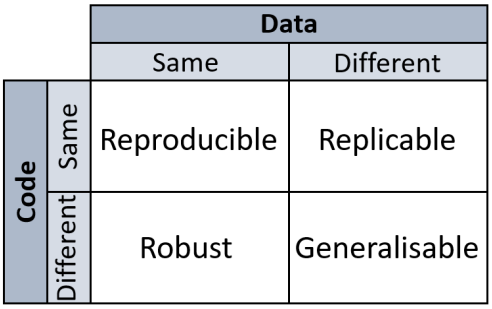


Figure 2.14: Four categories of recreating results as defined in Pineau et al., 2021

Because OpenPose (Cao et al., 2019) is used to transform a video-based data set into one of skeletal sequences it is important to have confidence in the OpenPose model. After all, if the pose estimation system does not work there is no foundation for other methods to build upon. For this reason the results reported in the OpenPose paper will be attempted reproduced by testing the publicly available OpenPose implementation[2] on data that was used in the paper to report performance.

[2]Download and installation instructions located at https://github.com/CMU-Perceptual-Computing-Lab/openpose

Chapter 3

Method

This chapter contains information on the methods used to create a data set, train models and perform feature selection for an XGB Classifier. All code used to perform these tasks can be found in the github repository located at https://github.com/ntnu-ai-lab/excergame-pose-estimation. Figure 3.1 illustrates the total process from exercise videos to predictions.

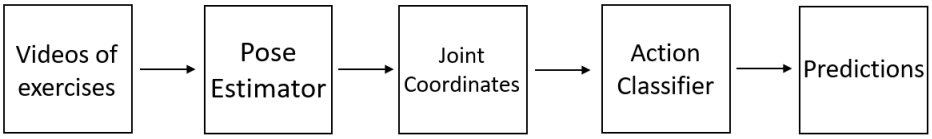


Figure 3.1: The three first squares correspond to the process of creating the data set used to train classifiers, covered in section 3.1. The last three correspond to the process of classification. The creation of the classifiers is covered in sections 3.2.1 and 3.2.2.

3.1 Creating the Data Set

The data used to train the models is based on an unpublished dataset[1] filmed at NTNU consisting of videos capturing participants performing a series of 17 different exercises filmed at 25 frames per second. The exercises were repeated 10 times, and each video features a single repetition of an exercise. These exercises are intended for use in physiotherapy, specifically related to recuperation after procedures like e.g. hip replacements, and are therefore fairly simple, slow and deliberate. Illustrations of the exercises can be seen in appendix A. Each video was between 4 and 8 seconds long, corresponding to a length of 100 to 200 frames. The participants were captured by multiple cameras placed around the capture area. The capture area itself was free of equipment and other foreign objects, resulting in no occlusion apart from self occlusion by the participant. The lighting

[1]Dataset created by Emanuel A. Lorenz, Xiaomeng Su and Nina Skjæret Maroni

was good and consistent between subjects. All these factors contribute to a "clean" dataset which makes it easier for a model to achieve good results on, but could fail to prepare it for real world applications. For the purposes of this thesis only the camera facing the participant head on was used, resulting in a smaller data set consisting of 170 videos per participant. OpenPose (Cao et al., 2019) was used to perform 2D pose estimation on these front-facing videos. The output from OpenPose was one JSON object per frame containing the joint coordinates of each joint. These objects were grouped together according to which video they came from, then each group was sorted chronologically by frame number and combined into a single csv file per video. Each of these csv files has 25 sets of (x, y) coordinates per row for a total length of 50, one row per frame of its corresponding video. Each column thus contains a chronological series of joint coordinate values for the coordinate corresponding to said column. The resulting set of csv files, one for each video, was ultimately used as data for training action classifiers. In order to provide the classifier models with uniform input sizes, some processing of the data was performed. 100 was chosen as the number of columns in each input. For each video the first, last and middle 100 columns were used to create three $50x100$ input grids, for a total of 5000 features per input.

3.2 Training Models

Two different types of models were trained as classifiers, an XGBoost Classifier (XGBC) and a Convolutional Neural Network (CNN). For both models parameter searches were performed to identify promising hyper-parameter values. The XGBC model also underwent feature selection, for a total of three models: XGBC with and without feature selection, and a CNN.

3.2.1 XGBoost

To find parameters that would yield good results two parameter searches were performed. The first was a random parameter search fetching parameter values from a wide range of values in order to narrow down the parameter search space. The results of the random search was used as indicators for which value ranges showed promise. With the parameter value ranges narrowed down a more targeted grid search was performed. Each of these parameter searches were done using 5-fold cross validation. This process yielded final parameter values seen in table 3.1.

To evaluate the performance of the parameter values leave-one-group-out cross-validation was used. At each iteration one participant was left out to be used as the test set, repeating until all participants had been left out once. The data not being left out was split into a validation and training set, using a 10-90 split. The training used early stopping to evaluate when the model was finished training, with the early stopping set to stop after 10 epochs without improvements to the validation set performance.

Parameter	Value
learning rate	0.1
max depth	2
n_estimators	150

Table 3.1: Final parameter values for XGBoost classifier after parameter search. n_estimators refers to the number of trees that can be added to the ensemble, and max depth is the maximum depth any individual tree is allowed to grow to.

After the leave-one-group-out cross-validation, feature selection was performed. An initial model was trained using the parameters in table 3.1. Before feature selection the input to this model had 5000 features. Using XGBC's built-in feature importance evaluation method revealed that 3614.0 ± 20.7 features had zero impact on the model's predictions; they only served to make the input larger than necessary and prolong the training process. In order to find the features that had a significant impact on the model's performance, an iterative process of eliminating features was performed. At each iteration the features were sorted according to their importance and the top n most important features were kept. A new data set was generated keeping only these *important* features, and a new model was trained on the data. This process was repeated, halving n at each iteration, until the performance of the model was noticeably impacted. Then a more exhaustive search through values of n close to this value could be performed. Through this process only the features that actually had an impact on the model's ability to classify exercises were kept.

3.2.2 Convolutional Neural Network

In addition to the XGBoost classifiers a CNN was trained on the data. Parameter search was used to find good parameters for this model too, but it was done through a combination of manual and automated searches. The initial manual exploration revolved around finding a functional shape for the network, i.e. how many layers, how many features per layer, etc. Once promising network candidates were found random searches were set up and ran for these. Using the results of these searches, narrower grid searches could be used to find final hyperparameter values. The resulting values and the shape of the network can be seen in table 3.2 and figure 3.2

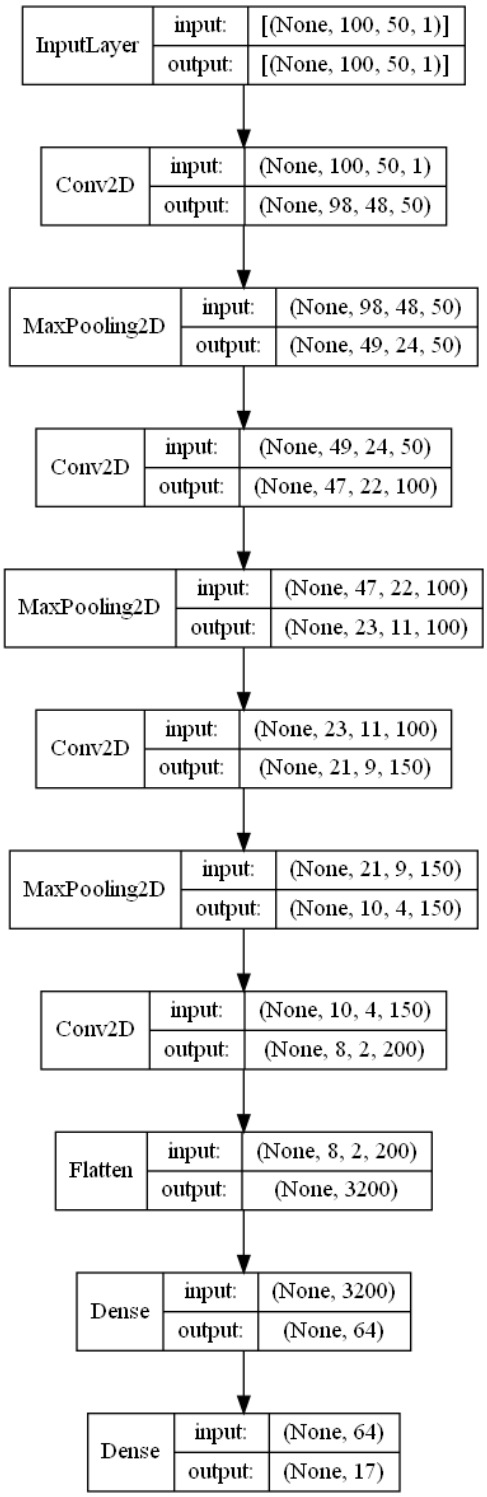


Figure 3.2: The shape of the CNN after hyper-parameter searches. For the MaxPooling2D and Conv2D layers numbers indicate the shape of the input and output, given as (height, width, depth). For the Dense layers the number represents the incoming and outgoing connections.

Parameter	Value
learning rate	0.001
optimizer	Adam
epochs	15
loss function	categorical cross-entropy

Table 3.2: Resulting parameter values for the CNN classifier after parameter searches.

Chapter 4

Experiments and results

4.1 Reproducing OpenPose Results

Cao et al., 2019 used, among others, the COCO dataset to report results and compare their model's performance to that of other methods. When evaluating keypoint[1] detection performance COCO uses Average Precision (AP). To calculate the AP for keypoints a measure called Object Keypoint Similarity[2] (OKS) is used. This metric gives each joint location prediction a score ranging between 0 and 1, with 1 being a perfect match and 0 a complete fail, based on the distance to the ground truth. By using the OKS score as a threshold, each keypoint can now be classified in a binary manner as correct or incorrect. If the OKS threshold is set to 0.5 this means that any keypoint with OKS > 0.5 is considered a correct prediction, while points with OKS < 0.5 are considered incorrect. Doing this calculation for each prediction allows for the calculation of the AP for the total set of predictions.

Method	AP	AP^{50}	AP^{75}	AP^{M}	AP^{L}
OpenPose test-dev	64.2	**86.2**	70.1	61.0	68.8
OpenPose validation	**65.3**	85.2	**71.3**	**62.2**	**70.7**
Reproduction	59.2	83.2	59.8	62.0	68.4

Table 4.1: Results from OpenPose evaluated on the COCO test-dev and validation data sets as reported by Cao et al., 2019, and from attempting to reproduce the results using the COCO validation set. All values are percentages.

Table 4.1 shows the reported results of OpenPose on the COCO test-dev and validation data sets. Of the two, the results based on the validation data is most directly comparable to our attempt at reproducing the results, as this is the data that was used to attempt said reproduction. AP^{50} and AP^{75} are AP values using 0.5 and 0.75 as OKS thresholds. The score simply labeled AP is an average over 10 OKS-thresholded AP scores, with the thresholds ranging from 0.5 to 0.95 with

[1] In the COCO dataset joint location are referred to as keypoints

[2] https://cocodataset.org/#keypoints-eval

a 0.05 interval. AP^M and AP^L are the same as AP, just filtered to only consider medium or large sized objects respectively.

4.2 Model Results

4.2.1 Performance

To evaluate the models, F1 and accuracy are used as metrics, shown in table 4.2 and 4.3, in addition to a confusion matrix for each model, see figures 4.1, 4.2 and 4.3. Accuracy is the ratio of correctly classified instances to the total amount of instances. This can result in misleading scores if the classes in the dataset are unevenly distributed. This is not a problem for our dataset, which is very well balanced. The F1 score is the harmonic mean of precision and recall. A high F1 score means that both precision and recall are high, indicating a model that has a high amount of true positive classifications, and comparatively few false positives or negatives. A visualization of the distribution of true and false predictions can be seen in the confusion matrices. A square in row j, column k is colored according to how many times an input of class j was classified as belonging to class k. The optimal result is to have all classifications fall along the diagonal. Figure 4.1 shows that the greatest sources of confusion for the XGBC model before feature selection was distinguishing between exercises 1 and 2 and 16 and 17, as well as there being a tendency to confuse 6 for 10, but not the other way around. After feature selection the XGBC model still has some trouble separating 1 from 2, as well as issues distinguishing 15, 16, and 17; 1 and 5; and 3 and 8. The CNN confusion matrix show pillar-like patterns in columns 10, 11 and 15, and a weaker diagonal. As with the XGBC models there is also confusion between 15, 16 and 17. The errors seem to be concentrated in the mentioned areas, as there are not any strong activations outside of them.

Not shown in the tables is the result of the feature selection process described in section 3.2.1, which resulted in the number of features in the input being reduced to 150. This represents a 97% reduction in the number of features.

Model	Accuracy	St.Dev.
XGBC before feature selection	**0.9741**	± 0.0391
XGBC after feature selection	0.9714	± **0.0355**
CNN	0.8839	± 0.2622

Table 4.2: Accuracy scores for the classifiers.

4.2.2 Training Time

All training was performed on a system with an intel i5-6600K CPU running at 3.5 GHz, 16 GB RAM and an NVIDIA GeForce GTX 1070 GPU. The time spent training

Model	F1	St.Dev.
XGBC before feature selection	**0.9682**	± **0.0468**
XGBC after feature selection	0.9656	± 0.0488
CNN	0.8770	± 0.2768

Table 4.3: F1 scores for the classifiers.

the XGBoost Classifiers differed greatly depending on whether feature selection was performed or not, as shown in table 4.4. On the non-feature-selected model training took 52 seconds. After performing feature selection the time needed was down to 6.4 seconds, a reduction of roughly 88%. The CNN classifier used about 4 seconds per epoch, for a total of just above one minute of training time when set to train for 15 epochs.

Model	Training Time
XGBC without feature selection	52s
XGBC with feature selection	6.4s
CNN (15 epochs)	61s

Table 4.4: Time used to train the three different classifiers.

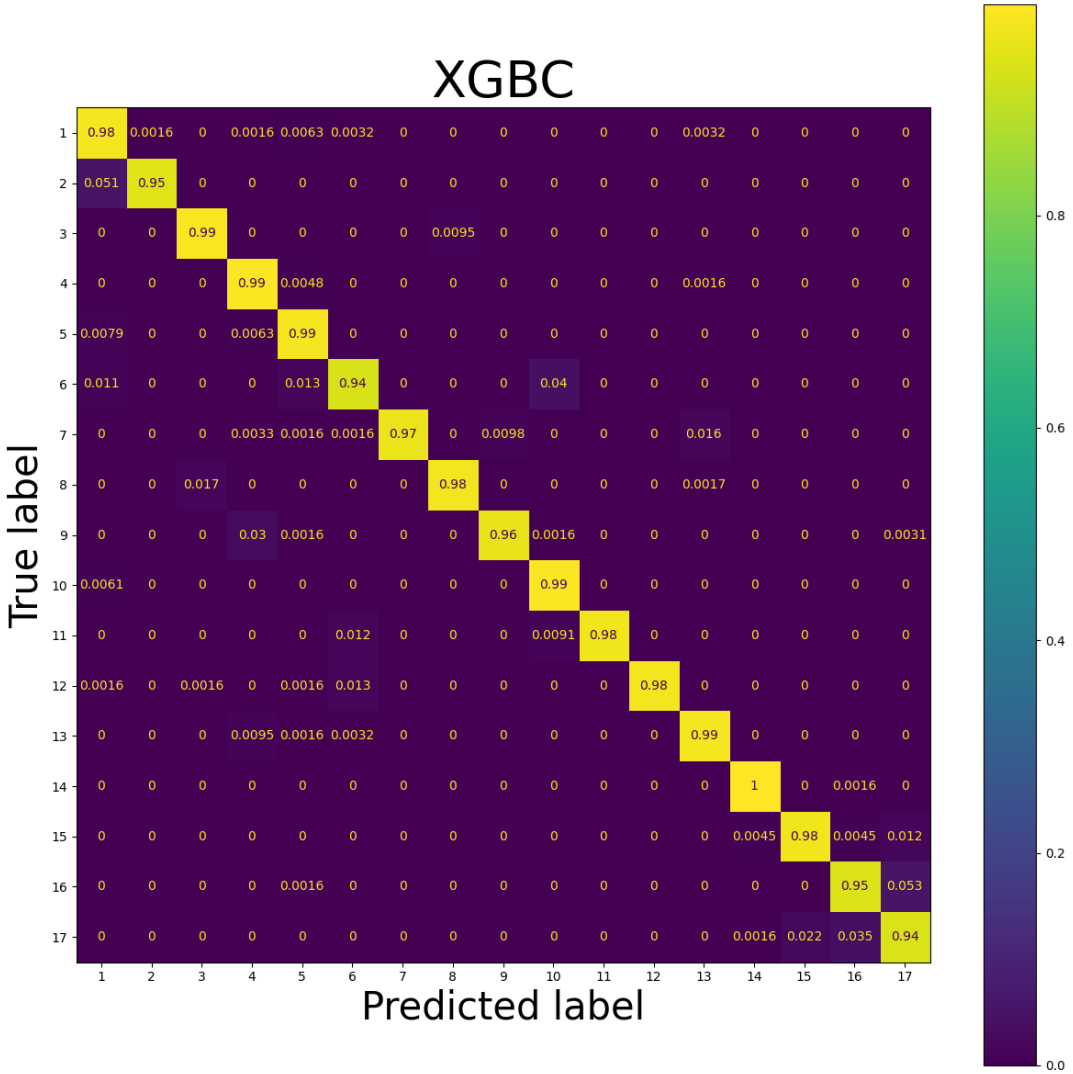


Figure 4.1: Confusion matrix for the XGBoost classifier before feature selection.

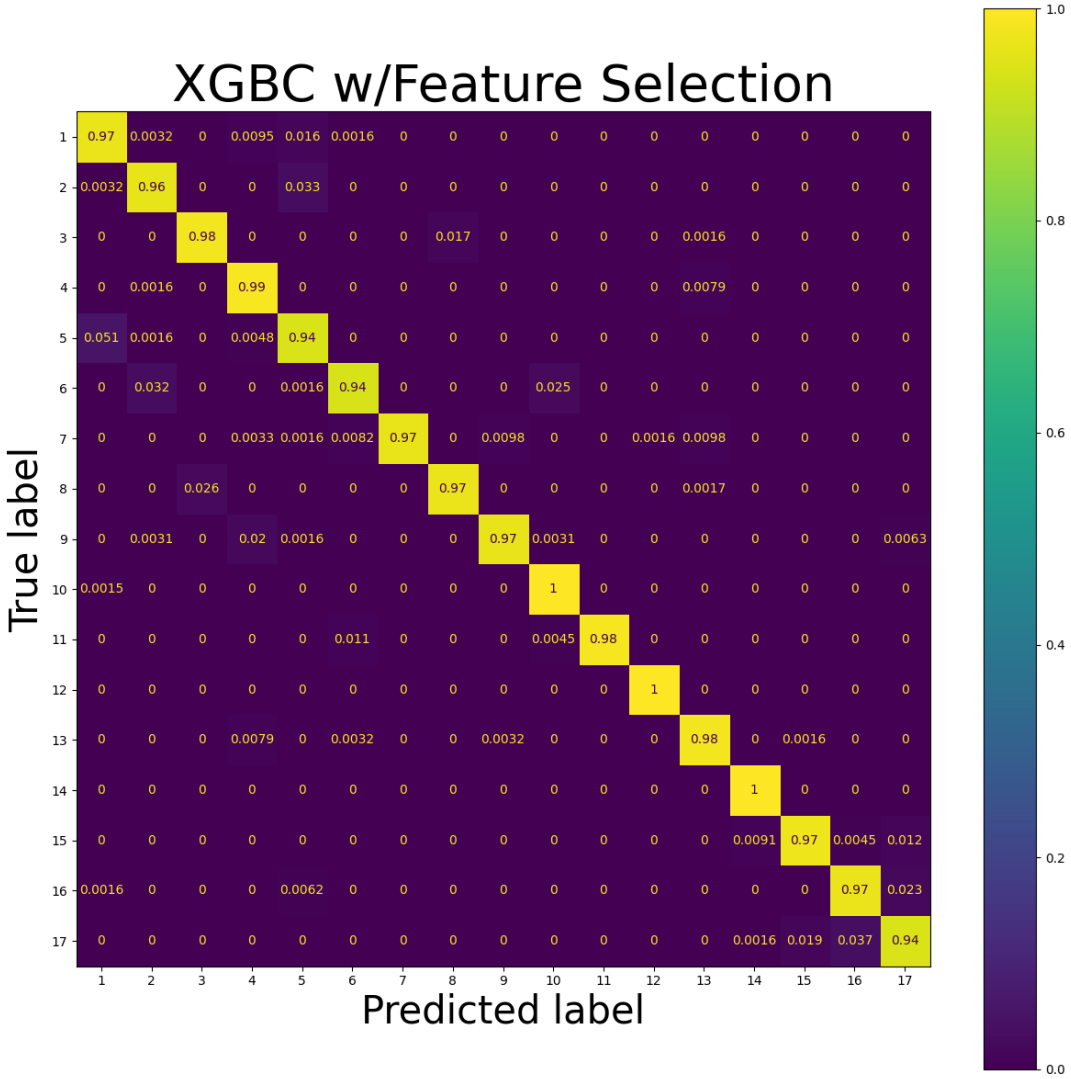


Figure 4.2: Confusion matrix for the XGBoost classifier after feature selection.

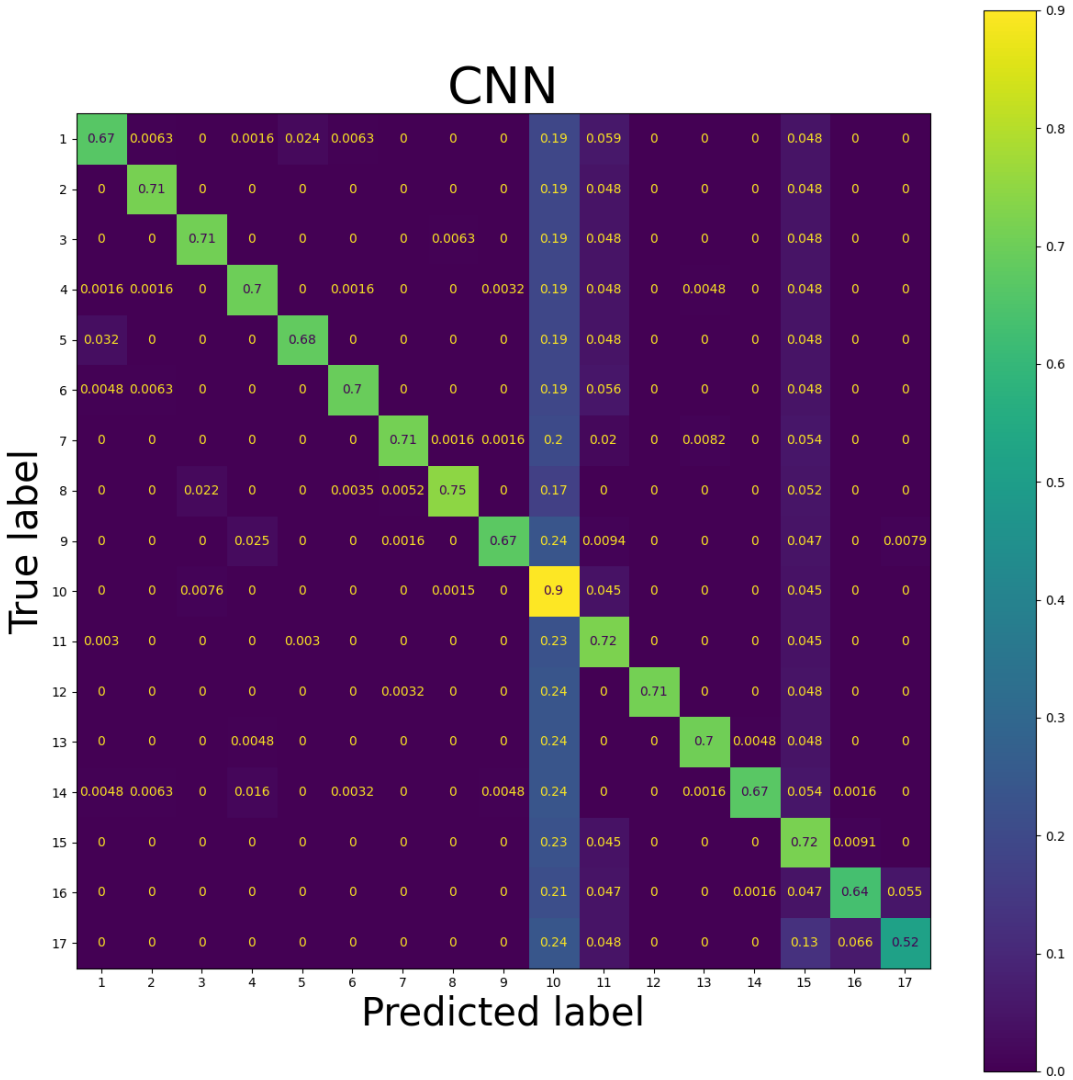


Figure 4.3: Confusion matrix for the CNN classifier

Chapter 5

Discussion

Research Question 1.1 asked what the state-of-the-art in pose estimation and action recognition is. Through the literature review process mentioned in section 1.2 and the findings from this outlined in section 2.7 an approach using skeleton sequences as inputs showed promise as a basis for our models. Since the data set described in section 3.1 consisted of videos, it was decided that an existing pose estimator would be used to extract information about joint positions from the films. Several pose estimation systems were explored, and OpenPose Cao et al., 2019 was chosen based on reported performance and availability. Together, the information gathered as part of the literature study was used to plan the rest of the thesis.

Research Question 1.2 asked whether results from state of the art methods are reproducible. The state-of-the-art method that was of specific importance for this thesis was the OpenPose pose estimator. This model was used to transform a data set of videos into one of joint positions to work as the basis for the classification models, so a sense of confidence in the method was imperative. Our results were compared to the results from the paper by evaluating performance on the COCO keypoint dataset. As can be seen in table 4.1 our results, while not too far off, did not quite reach the same results as reported. The results for AP^{50}, AP^{M} and AP^{L} are all close to meeting the reported numbers, only being behind the best scores by 3, 0.2 and 2.3 percentage points respectively. It should be noted that AP^{M} is only 2 points behind the reported number on the validation set, which is the data set that was used to get our numbers and is therefore a more direct comparison. The AP falls a bit further behind with a difference of 6.1 points. AP^{75} shows the largest discrepancy, being 11.5 points behind the best reported value.

Not being able to reproduce the exact same results was to be expected, as explained in section 2.8. The question therefore becomes whether or not the numbers are close enough to where the model can be said to provide the needed functionality. For our purposes it is not as important to have pin-point accurate joint positions as it is to have joint positions that are good enough to be able to distinguish movements from each other. While AP^{75} was the metric with the biggest difference, it is also a fairly strict metric requiring OKS > 0.75 for a point to be

classified as correct. The inability for our local OpenPose instance to reach the reported values for this metric may not matter that much if that level of precision is not required. Compare to AP^{50} where the difference is much smaller. If this lower level of precision in the joint positions is sufficient for a model to classify movements, then the lackluster AP^{75} score does not really matter for our purposes. This is not that unlikely as the temporal information contained in the positions of joints over time is not dependent on exact location, instead relying more on how joint locations change over time relative to themselves and others. A less strict demand on the joint locations is therefore not unreasonable. The Ap score, which covers both strict and less strict OKS thresholds, falls in between AP^{50} and AP^{75} with its 6.1 percentage point difference. Comparing our score to the worse score of 64.2 reported from the test-dev set, however, reduces the difference to 5 points. While it can not be said that the model did succeed in reproducing the results of all the metrics, it does not need to achieve the strictest goals. The metrics corresponding to less rigorous requirements all reached scores close to those reported, which is good enough for the system to be likely to capture enough information for a classifier to classify different exercises.

Research Question 2.1 asked whether a model can be trained to classify human actions. The results seen in section 4.2 indicate that the XGBC models managed to learn the task at hand very well. Their accuracy and F1 scores are high, with low standard deviations. This indicates consistently high results throughout the whole cross-validation process. This is further corroborated by their confusion matrices showing clear diagonals with few errors. As reported in section 4.2, however, there were some exercises that both models consistently struggled to differentiate between, namely 15, 16 and 17. Looking at the movements in appendix A, these are exercises that share some similarities, as these are the exercises where the participant stays seated the whole time. It is therefore not unexpected that a classifier would struggle with differentiating these exercises more than other combinations. The non-feature-selected XGBC model showed a tendency for confusing exercise 1 and 2, which both involve a forward vertical movement of the arm. The XGBC model after feature selection showed a tendency to confuse 1 and 5 with each other. These movements are again similar to one another, as they both involve lifting an arm into a vertical position, with the start, end and middle part of the movements being essentially identical. The model also showed a slight issue with differentiating exercises 3 and 8. Both of these movements begin and end identically, which could lead to some confusion similar to that between exercise 1 and 5. In general there is a pattern where exercises involving similar movements tend to be confused for one another. The CNN model results tell a different story. It scored lower on the metrics and had a much higher standard deviation value, indicating an inconsistent classifier. Looking at its confusion matrix columns 10, 15, and, to a lesser degree, 11 are all showing substantial amounts of incorrect classifications. However, the CNN models did not all fail. Even though the diagonal is weaker than in the XGBC models, it is still where most of the predictions fell overall. One explanation for this pattern is that a majority of the models did

indeed learn to differentiate between the exercises, but that some amount of the time the learning process fails, perhaps because of gradient descent ending up in a local minimum, bringing the overall score down. Outside of the three columns the only exercises that are consistently misclassified are 15, 16 and 17. This does hint at the possibility of the CNN-based classifier being better in the cases where it manages to learn properly, the issue being that that does not always happen.

Looking at the results of the three models overall, it seems that the most consistent problem the models have in common is differentiating between the chair-based exercises. Outside of this they are all able to produce classifiers that differentiate between the exercises, though inconsistently in the CNN's case. The main reason for such high scores is that we used a well behaved data set. This is due to several factors. The area used to capture the films was identical for each participant. It was well lit, without any foreign objects that could interfere with their ability to move or cause occlusion. This is different from what such a framework would probably encounter "in the wild", in a consumer product for home use, or even for use by a specialist in a controlled environment, e.g. a physical therapist. The exercises themselves are simple, deliberate and mostly distinct from one another. This is not such a far fetched expectation for use cases such as physical therapy or rehabilitation, where these kinds of exercises may be the main focus. For a data set containing faster, larger and more similar sets of moves the system might struggle more with the learning, for instance if it were to be used to evaluate gym exercises as part of a workout program. The participants in the videos also wore clothes that did not hide or cover body parts, think typical exercise clothing one would wear for a run. This is fairly likely to match what patients would wear to a physiotherapy appointment they would go to. However, if an action classification system is to be used in a home setting, there is less insight into and control over what people would wear. The data set also contains no incorrectly performed exercises, so a system trained using the data would be unable to learn what a bad example of an exercise looks like. Ideally the classifier should not only classify the movements, but also give a rating or provide feedback as to whether an exercise was done well or not, and why.

Research Question 2.2 asked what improvements feature selection can yield. From table 4.4 it becomes clear that removing insignificant features significantly reduces training time. Tables 4.2 and 4.3 show that this time improvement can be had without negative impact on the accuracy, F1-score or variance. This can be of great use during model development as it lets developers get feedback on how a model performs much faster. Speeding up development time allows for a larger amount of model configurations to be trained in a given amount of time, allowing for a larger parameter space to be explored.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

The overall aim of this thesis was to create a system that is able to classify human actions based on monocular RGB video input, more specifically exercises related to physiotherapy and rehabilitation. To achieve this goal three human action classification models were created. A data set containing RGB video was converted into one containing skeleton sequences using an existing pose estimator. The pose estimator performance was evaluated by attempting to reproduce some of its reported results, and comparing the resulting numbers with those that were reported. The resulting data set was used to train the models to distinguish between 17 different exercises. Each model underwent leave-one-group-out cross-validation, and the results were used to compare the models.

The results from the models reported in section 4.2 and the discussion in section 5 show that all three models learned to separate the classes from one another, though inconsistently in the case of the CNN-based model. This result indicates that making a complete end to end system able to take RGB video as input and identify which exercise, if any, is present in said video in real time is a feasible endeavour. This has great potential in areas such as physical therapy or rehabilitation. Being able to receive real time feedback as to whether an exercise was executed correctly or not without needing constant supervision from a professional opens up for more freedom in when and where a patient wants to exercise. This freedom could in turn lead to increased levels of both motivation and happiness, which are positive attributes for a patient to have.

6.2 Future Work

As mentioned in chapter 5 the data set used to train classifiers is very well-behaved, potentially too much so. A data set that more realistically mirrors what the system would be exposed to in real scenarios could allow for training classifiers that are able to handle badly executed exercises, unfamiliar movements, multiple in-

dividuals in frame, various environments, etc. Another aspect of the data set used in this thesis is that it consists of neatly segmented videos, each containing exactly one exercise. A classifier that can work on a video stream, for example from a web camera, is a necessity for real life applications.

An end to end system able to identify actions from RGB video and classify them in real time, as mentioned in section 6.1, is another step between what has been done in this thesis and a hypothetical rehabilitation exercise evaluation program. Building further off of this, it would also be of great use to have a system that not only says whether an exercise was executed correctly or not, but also provides some sort of feedback, e.g. a score. By telling a user how well they performed their exercise they can quickly detect when they are doing something wrong, and can take immediate steps to correct the mistake.

Bibliography

Bergstra, J. & Bengio, Y. (2012). Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, *13*(1), 281–305.

Boehmke, B. & Greenwell, B. (2020). *Hands-on machine learning with r*. Chapman; Hall/CRC. https://bradleyboehmke.github.io/HOML/

Bulat, A. & Tzimiropoulos, G. (2016). Human pose estimation via convolutional part heatmap regression. *Computer vision – ECCV 2016* (pp. 717–732). Springer International Publishing. https://doi.org/10.1007/978-3-319-46478-7_44

Cao, Z., Hidalgo Martinez, G., Simon, T., Wei, S. & Sheikh, Y. A. (2019). Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Chen, T. & Guestrin, C. (2016). Xgboost: A scalable tree boosting system. *CoRR*, *abs/1603.02754*. http://arxiv.org/abs/1603.02754

Chen, Y., Tian, Y. & He, M. (2020). Monocular human pose estimation: A survey of deep learning-based methods. *Computer Vision and Image Understanding*, *192*, 102897. https://doi.org/10.1016/j.cviu.2019.102897

Goodfellow, I., Bengio, Y. & Courville, A. (2016). *Deep learning* [http://www.deeplearningbook.org]. MIT Press.

Gundersen, O. E., Shamsaliei, S. & Isdahl, R. J. (2022). Do machine learning platforms provide out-of-the-box reproducibility? *Future Generation Computer Systems*, *126*, 34–47. https://doi.org/https://doi.org/10.1016/j.future.2021.06.014

Herath, S., Harandi, M. & Porikli, F. (2016). Going deeper into action recognition: A survey. https://doi.org/10.48550/ARXIV.1605.04988

Horn, B. K. & Schunck, B. G. (1981). Determining optical flow. *Artificial Intelligence*, *17*(1), 185–203. https://doi.org/https://doi.org/10.1016/0004-3702(81)90024-2

Luvizon, D. C., Picard, D. & Tabia, H. (2018). 2d/3d pose estimation and action recognition using multitask deep learning. https://doi.org/10.48550/ARXIV.1802.09232

Luvizon, D. C., Tabia, H. & Picard, D. (2017). Human pose regression by combining indirect part detection and contextual information. https://doi.org/10.48550/ARXIV.1710.02322

Martinez, J., Hossain, R., Romero, J. & Little, J. J. (2017). A simple yet effective baseline for 3d human pose estimation. *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*.

Mehta, D., Sridhar, S., Sotnychenko, O., Rhodin, H., Shafiei, M., Seidel, H.-P., Xu, W., Casas, D. & Theobalt, C. (2017). Vnect: Real-time 3d human pose estimation with a single rgb camera. *ACM Transactions on Graphics*, *36*(4). https://doi.org/10.1145/3072959.3073596

Mitchell, T. M. (1997). *Machine learning*. McGraw-Hill.

Natekin, A. & Knoll, A. (2013). Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, *7*. https://doi.org/10.3389/fnbot.2013.00021

Nibali, A., He, Z., Morgan, S. & Prendergast, L. (2018). 3d human pose estimation with 2d marginal heatmaps. https://doi.org/10.48550/ARXIV.1806.01484

Nie, B. X., Xiong, C. & Zhu, S.-C. (2015). Joint action recognition and pose estimation from video. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1293–1301. https://doi.org/10.1109/CVPR.2015.7298734

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press. http://neuralnetworksanddeeplearning.com/

Pineau, J., Vincent-Lamarre, P., Sinha, K., Larivière, V., Beygelzimer, A., d'Alché-Buc, F., Fox, E. & Larochelle, H. (2021). Improving reproducibility in machine learning research: A report from the neurips 2019 reproducibility program. *Journal of Machine Learning Research*, *22*.

Russell, S. & Norvig, P. (2010). *Artificial intelligence: A modern approach* (3rd ed.). Prentice Hall.

Simonyan, K. & Zisserman, A. (2014). Two-stream convolutional networks for action recognition in videos. https://doi.org/10.48550/ARXIV.1406.2199

Tatman, R., VanderPlas, J. & Dane, S. (2018). A practical taxonomy of reproducibility for machine learning research.

Toshev, A. & Szegedy, C. (2014). DeepPose: Human pose estimation via deep neural networks. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. https://doi.org/10.1109/cvpr.2014.214

Vonstad, E. K., Su, X., Vereijken, B., Bach, K. & Nilsen, J. H. (2020). Comparison of a deep learning-based pose estimation system to marker-based and kinect systems in exergaming for balance training. *Sensors*, *20*(23). https://doi.org/10.3390/s20236940

Vonstad, E. K., Vereijken, B., Bach, K., Su, X. & Nilsen, J. H. (2021). Assessment of machine learning models for classification of movement patterns during a weight-shifting exergame. *IEEE Transactions on Human-Machine Systems*, *51*(3), 242–252. https://doi.org/10.1109/THMS.2021.3059716

Wiemeyer, J. & Kliem, A. (2011). Serious games in prevention and rehabilitation—a new panacea for elderly people? *European Review of Aging and Physical Activity*, *9*, 41–50.

Yan, S., Xiong, Y. & Lin, D. (2018). Spatial temporal graph convolutional networks for skeleton-based action recognition. https://doi.org/10.48550/ARXIV.1801.07455

Zhang, H.-B., Zhang, Y.-X., Zhong, B., Lei, Q., Yang, L., Du, J.-X. & Chen, D.-S. (2019). A comprehensive survey of vision-based human action recognition methods. *Sensors*, *19*(5). https://doi.org/10.3390/s19051005

Zhu, Y., Li, X., Liu, C., Zolfaghari, M., Xiong, Y., Wu, C., Zhang, Z., Tighe, J., Manmatha, R. & Li, M. (2020). A comprehensive study of deep video action recognition. https://doi.org/10.48550/ARXIV.2012.06567

Appendix A

Exercises

Figure A.1: While being kept straight the right arm is brought up to a vertical position by moving it 180° in a forward arching movement, then the arm is brought back down in the same way.

Figure A.2: The forearm is brought to a vertical position by moving it 180° in a forward arching movement, then brought back down the same way.

Figure A.3: While keeping both arms stretched out to the sides, the right leg is first lifted forwards by 90°, then stretched out slightly behind the body, before it is brought back to the starting position.

Figure A.4: From a standing position, the right lower leg is brought back until parallel with the ground, then lowered back into starting position.

Figure A.5: While being kept straight the right arm is brought up to a vertical position by moving it 180° in a sideways arching movement, then the arm is brought back down in the same way.

Figure A.6: Starting with the lower arm extended forward parallel to the ground, it is first turned 90° outwards pointing away from the body, then brought back 180° before being returned to the starting position.

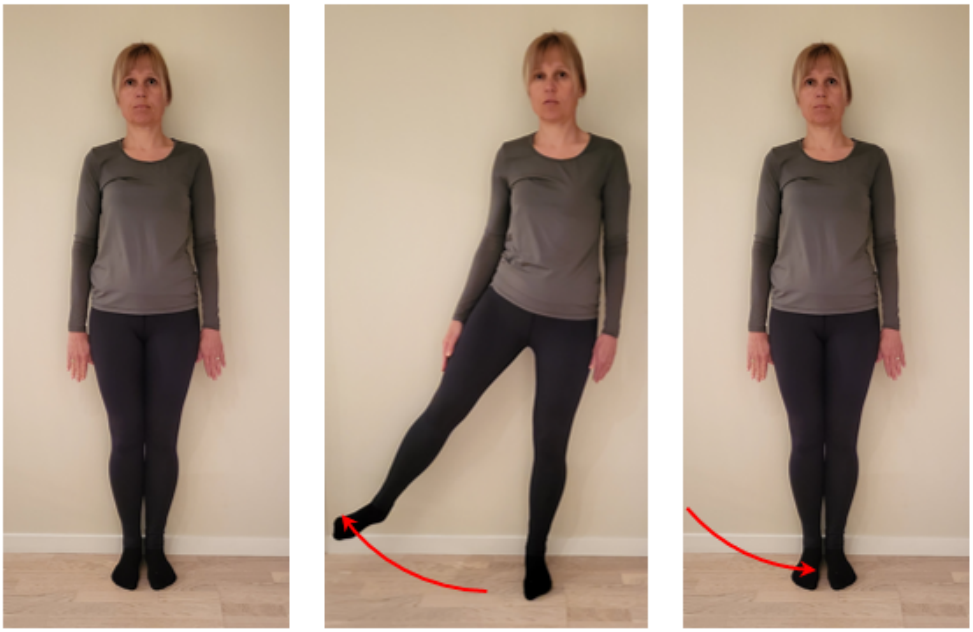


Figure A.7: From a standing position, the right leg is brought approximately 45° out sideways, then return to the starting position.

Figure A.8: Starting with arms stretched out to the sides parallel to the ground, the leg is lifted up to also be parallel to the ground, turned 90° outwards, brought back over by 180°, then brought back into the forward position before being lowered back into the starting position.

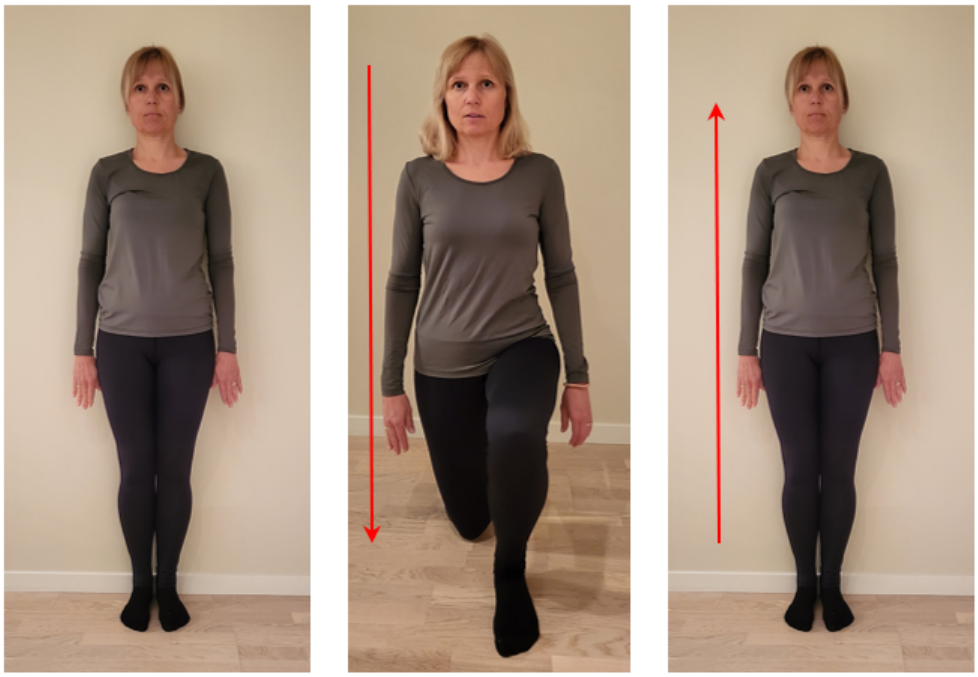


Figure A.9: From a standing position, the right leg is brought backwards and down until the right knee touches the ground, then back up into the starting position.

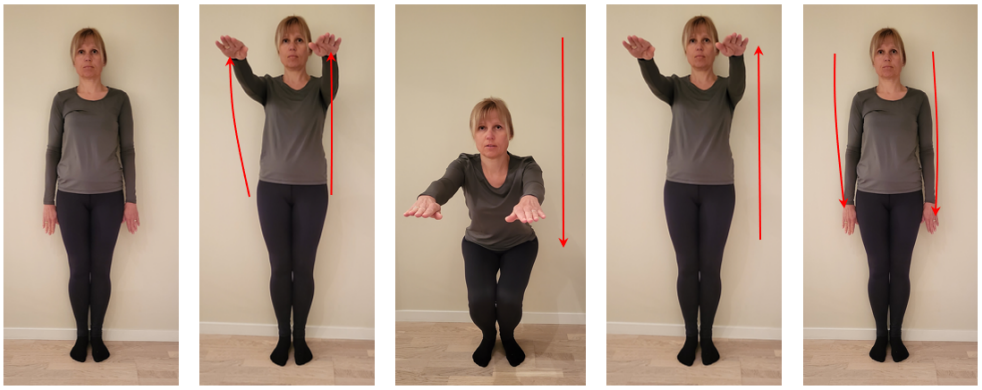


Figure A.10: From a standing position both arms are lifted forward until parallel to the ground, followed by the knees being bent until the legs are also parallel to the ground. Then the process is reversed to arrive back at starting position.

Figure A.11: Starting with the right arm stretched to the side parallel to the ground, it and the right foot is brought across the body and back.

Figure A.12: Starting with arms stretched out to the sides parallel to the ground the torso is twisted 90° to the right and back.

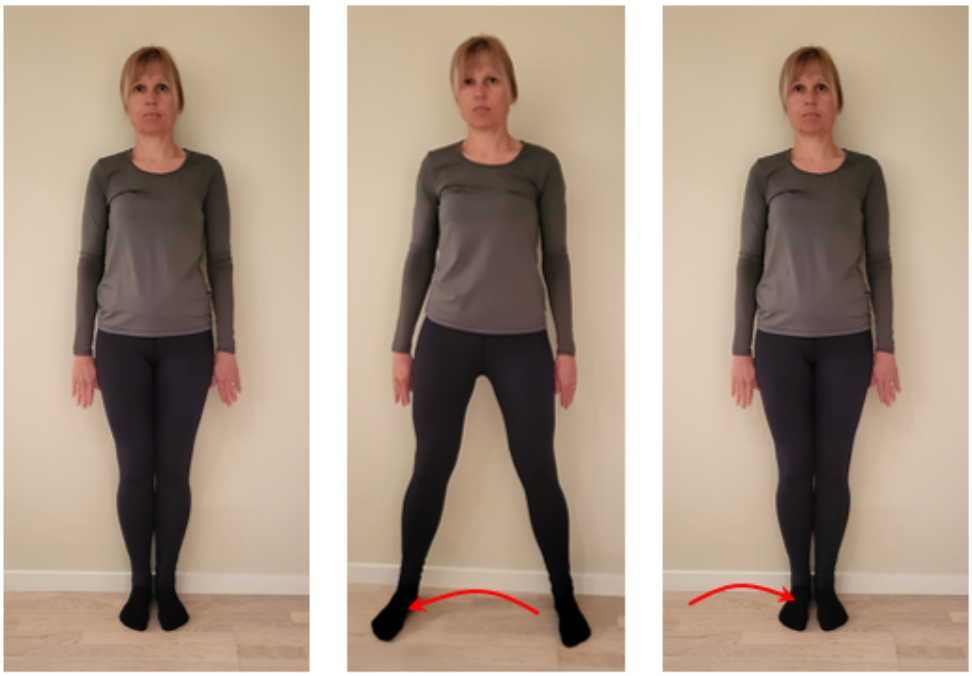


Figure A.13: From a standing position an outwards sideways step is taken with the right leg, then an inwards step to return to starting position.

Figure A.14: From a sitting position, the body is brought into a standing position then back down to a sitting position.

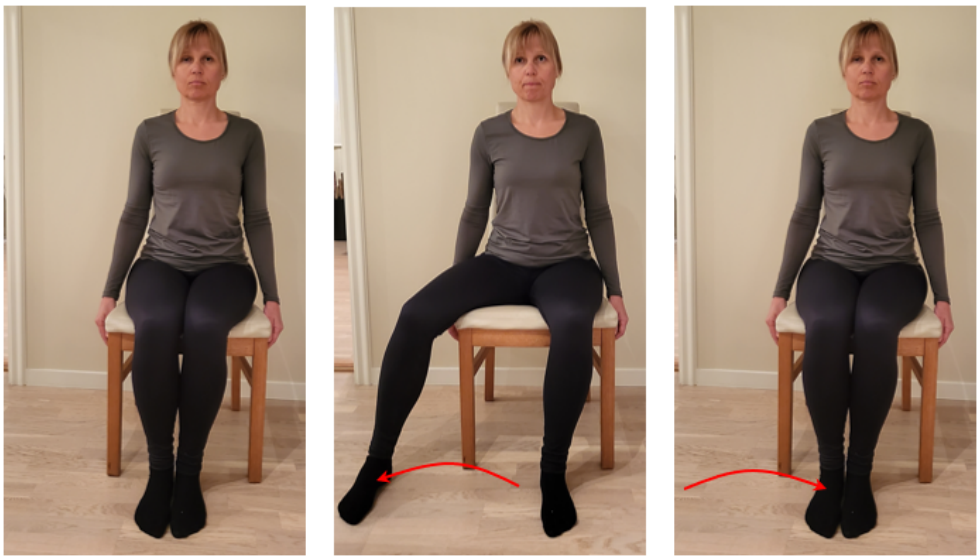


Figure A.15: From a sitting position the right foot is moved a step to the right, then back.

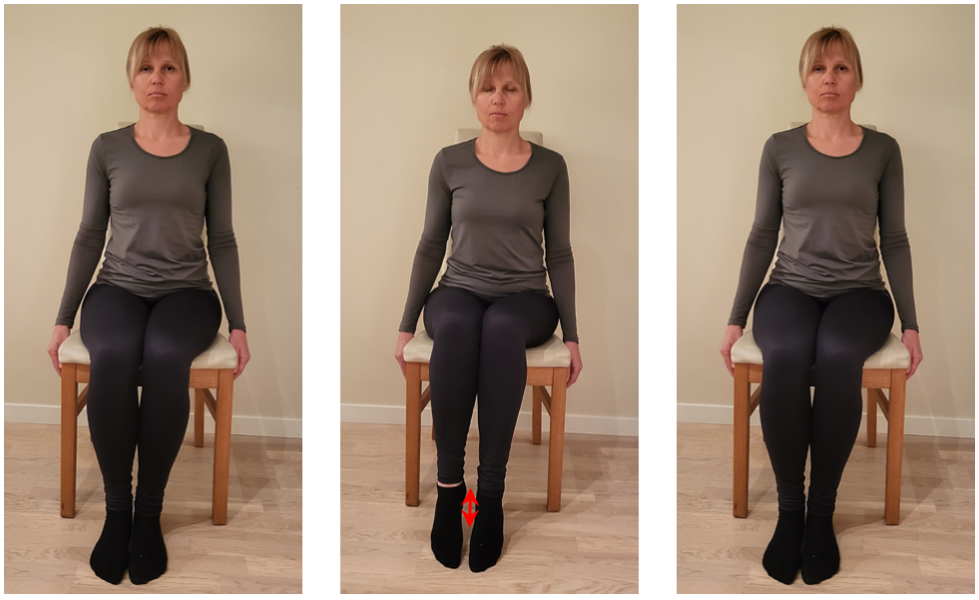
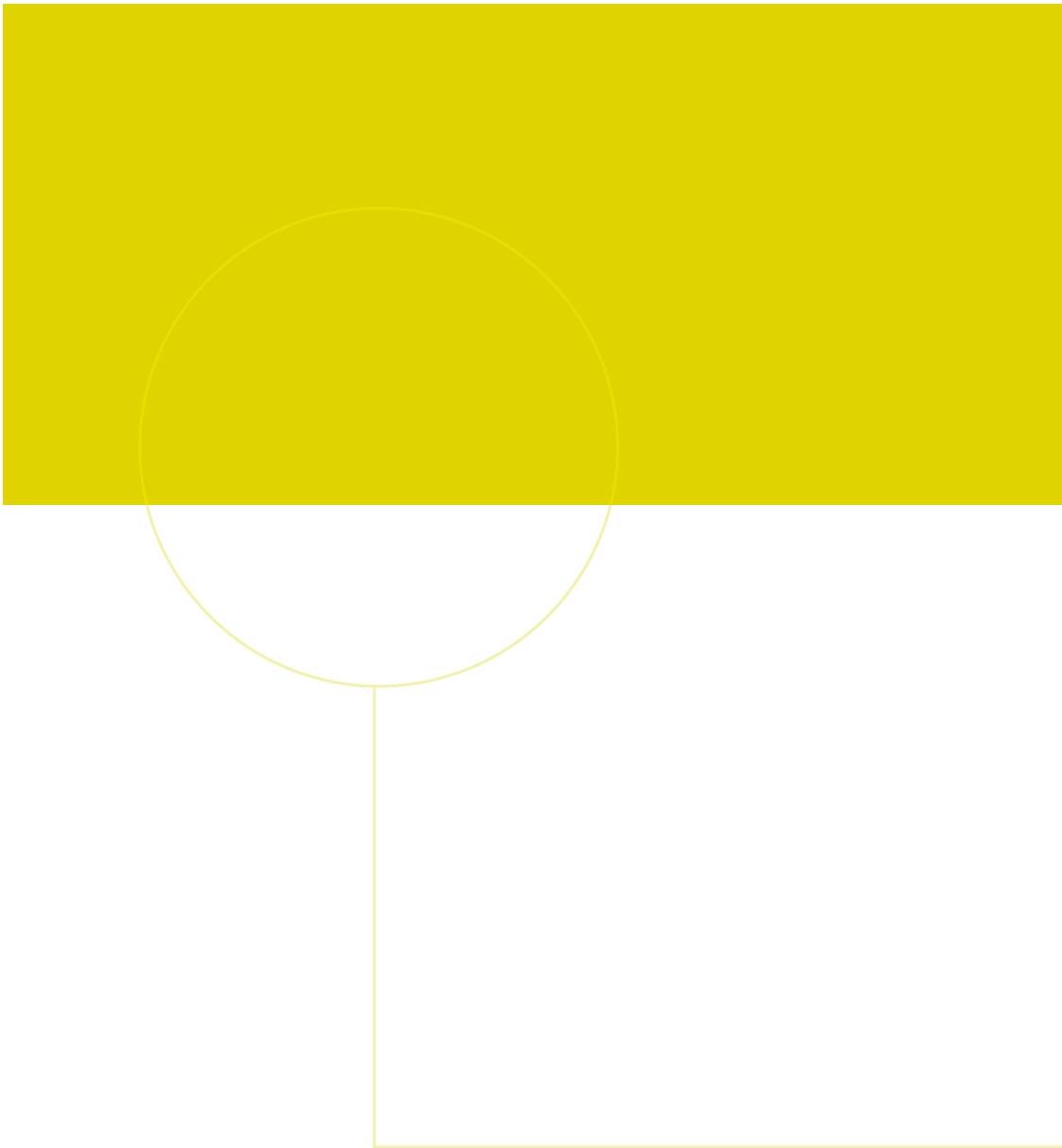


Figure A.16: From a sitting position the heels are lifted, then lowered.

Figure A.17: From a sitting position the lower right leg is brought up and forward until parallel to the ground, then back down again.



NTNU

Norwegian University of
Science and Technology