Assessments of Posterior Orientation Errors after Anterior Cruciate Ligament injuries using deep learning based methods

Filip Kronström

March 1, 2021

Acronyms

AIK-HKD AI Challenger Human Keypoint Detection dataset

CNN Convolutional Neural Network

COCO Microsoft Common Objects in Context dataset

CORAL COnsistent RAnk Logits

COTE Collective Of Transformation-based Ensembles

DARK Distribution-Aware coordinate Representation of Key-point

DNN Deep Neural Network

GAP Global Average Pooling

GPU Graphics Processing Unit

Grad-CAM Gradient-weighted Class Activation Mapping

HIVE-COTE Hierarchical Vote Collective of Transformation-based Ensembles

HPE Human Pose Estimation

HRNet High-Resolution Net

MPII Max Planck Institute for Informatics dataset

POE Postural Orientation Error

ReLU Rectified Linear Unit

SLS Single Leg Squat

SOTA State of the Art

TSC Time Series Classification

UCR University of California, Riverside

XCM Explainable Convolutional Neural Network for Multivariate Time Series Classification

Contents

List of Figures

List of Tables

Chapter 1

Introduction

skriv om risker med bias fr dataset osv...

1.1 Medical background

skriv om acl och varf;r detta arbete beh;vs related work, se ref i mendeley osv

1.1.1 POEs

...

1.1.2 avgr'nsningar

typ om att bara SLS analyseras? och lite s[nt.. kanske att 3d inte utv'rderas? eller att det utv'rderades litgrann??

Chapter 2

Background - Deep learning

A supervised machine learning problem can be described as finding a mapping between some input and output data, e.g. an image and a category, based on labeled input-output combinations. The idea with such methods is that a mapping found for the available data also should represent unseen data of the same type, i.e. it should generalize. To be able to get a measure of this generalization the available data is commonly divided into two parts, training data and test data. The training data is used to find the mapping and the test data is used to evaluate how well it performs on unseen data [8].

This chapter gives a brief introduction to a special type of machine learning called deep learning, which forms the basis of this work.

FIXAA DEN HAR SECTION INDELNINGEN...

2.1 Deep Neural Networks

Deep Neural Networks (DNNs) are combinations of linear and non-linear functions trained to approximate some other, potentially very complicated, function. The output of the network is formed as $f(x) = f_n \circ f_{n-1} \circ ... \circ f_1 \circ f_0(x)$ resulting in the layer terminology since the output from one function is passed as input to the subsequent one [18].

Below the layers used in our work are briefly explained.

Dense layer

The dense, or fully connected, layer is the basic model for a feedforward network. The outputs of such a layer is formed as linear combinations of the inputs and bias terms. Usually a non-linear activation function is applied to to this to be able to capture more general behaviors, resulting in the output

$$y_i = h\Big(\sum_{j=1}^{D} w_{ij}x_j + b_i\Big). \tag{2.1}$$

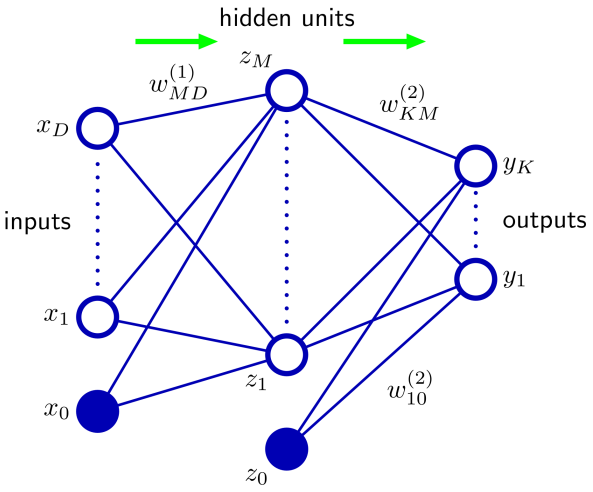


Figure 2.1.1: Feedforward neural network with two densely connected layers. Each line corresponds to one trainable parameter. x_0 and z_0 can be seen as ones added to the inputs introducing the bias terms [8].

$h(\cdot)$ is a, possibly non-linear, activation function. x_j, $j \in \{1, \dots, D\}$ are the inputs to the layer, w_{ij} and b_i are the weights and biases learned during training [8]. A network with two dense layers is shown in Figure 2.1.1.

Convolutional layers

Convolutional layers have proved successful for feature extraction from for instance time series or images. A reason for this is that they are equivariant to translation, meaning that patterns in a time series will be recognized in the same way no matter at which time steps they occur. The 1D convolution operation can be seen in (2.2). When applied to for instance images it is performed in two dimensions.

$$(x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{2.2}$$

x is the input and w is the kernel or filter which consist of the trainable parameters. As the kernel size is not affected by the input size the convolutional layer can be applied to inputs of different size, which is not possible with for instance the fully connected layer [18].

Pooling layers

Pooling layers are used to reduce the dimensionality of feature maps. Common types of poolings are the max and the average pooling methods. Traditional max pooling represents nearby numbers by it maximum value while average pooling uses their average. This type of max pooling has proved efficient together with convolutional layers for computer vision tasks. Figure 2.1.2 illustrates how the pooling works. It can also be performed globally, i.e. on the entire feature map, which can be a way of handling

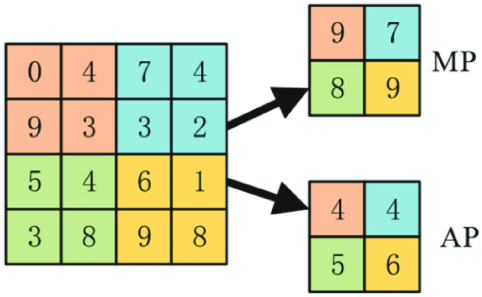
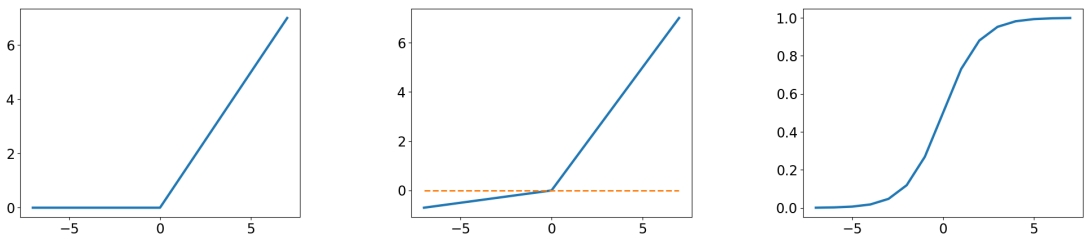


Figure 2.1.2: Illustration of max and average pooling with pooling size 2×2 and stride 2×2. Image from [47].

differently sized data. For a Time Series Classification (TSC) problem it is for instance possible to use size agnostic convolutional layers as feature extractors followed by a Global Average Pooling (GAP) layer resulting in a fixed size of the data to be classified [12].

Activation functions

The activation functions in a neural network has two main tasks. The first one is to introduce non-linearity to an otherwise linear model. For a dense layer performed with the function $h(\cdot)$ in (2.1). A common such function is Rectified Linear Unit (ReLU), $h(z) = \max\{0, z\}$. Benefits with ReLU is that it in its active region ($z > 0$) does not have a suppressing effect on the gradient and it is easily computable. A drawback, however, is that the gradient is zero in its inactive region ($z < 0$) meaning gradient based training methods does not work here. An alternative to avoid this issue is the leaky ReLU given by $h(z) = \max\{0.01z, z\}$. ReLU and leaky ReLU are shown in Figure 2.1.3a and 2.1.3b respectively. Activation functions are also used for the output of the network, e.g. to obtain outputs representing probabilities. The sigmoid function, $h(z) = 1/(1+\exp(-z))$, shown in Figure 2.1.3c, can be used for this. The sigmoid function will saturate the output between 0 and 1, however, if the model has several outputs e.g. representing the probabilities of the input belonging to different classes the total probability will not sum to 1. In this case the softmax function, $h(z)_i = \exp(z_i)/\sum_{j=1}^{K} \exp(z_j)$, can be used instead [18].



(a) ReLU (b) Leaky ReLU (c) Sigmoid

Figure 2.1.3: Different activation functions, note that the slope of leaky ReLU for negative numbers is exaggerated for visualization purposes.

2.2 Evaluation metrics

To be able to evaluate and compare models some evaluation metrics are needed. Table 2.2.1 shows four common classification metrics and the way they are calculated from the quantities defined in Definition 1.

FIXA FORMATERING P[DENNA DEFINITON

Definition 1 *True positives,* **TP***: Correctly classified positive samples*
False positives, FP: Incorrectly classified positive samples
True negatives, TN: Correctly classified negative samples
False negatives, FN: Incorrectly classified negative samples

Another way to present the result of a classification task is using the confusion matrix. This is a matrix where the columns corresponds to the predicted classes and the rows to the correct classes. Hence, this metric shows what kind of errors the model performs. An example of a confusion matrix is shown in Figure 2.2.1.

Table 2.2.1: Evaluation metrics using quantities in Definition 1.

Precision	$\frac{TP}{TP+FP}$
Recall	$\frac{TP}{TP+FN}$
F1 score	$2\frac{Precision \cdot Recall}{Precision+Recall}$
Accuracy	$\frac{TP+TN}{TP+FP+TN+FN}$

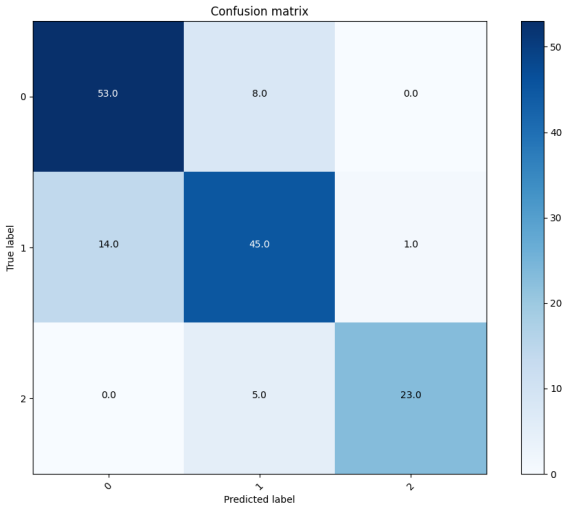


Figure 2.2.1: An example of a confusion matrix. The entries on the diagonal are correctly classified, while the position of an off-diagonal entry shows what kind of error has been made. The true class is given by the row and the predicted class by the column.

2.3 Training of

During training of a network a loss function, $\mathcal{L}$, which describes the desired behavior, is evaluated on the training data. To improve the performance of the model its parameters are changed to minimize this loss. In deep learning problems this optimization is usually performed with some gradient descent inspired method, shown in (2.3), where the parameters are updated in the direction which reduces the loss the most. With a large training data set the computation of the gradient quickly becomes expensive. A remedy for this has been to use stochastic or mini-batch gradient descent methods. Such algorithms use one or a few data points from the training set to estimate the gradient for each parameter update. Algorithms common today often use momentum, where previous gradients affect the parameter update direction, and adaptive learning rates (step size of parameter update), allowing different learning rate for different parameters [18]. One example of such a method is the Adam optimizer [25].

$$\boldsymbol{W}_{k+1} = \boldsymbol{W}_k - \alpha \boldsymbol{D} \tag{2.3}$$

where:

$\boldsymbol{W}_k$ = model parameters at iteration k
α = learning rate or step size
$\boldsymbol{D}$ = parameter update direction, e.g. $\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}}$ or a weighted average of earlier gradients

The gradients of the loss with respect to the model parameters are calculated using the back-propagation algorithm [40] which recursively uses the chain rule, (2.4), to propagate

the loss gradient through the network.

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx} \tag{2.4}$$

For a network where $f_0, f_1, ..., f_n$ denotes the outputs of the $n+1$ layers, with corresponding layer parameters $\boldsymbol{w}_0, \boldsymbol{w}_1, ..., \boldsymbol{w}_n$ and loss function $\mathcal{L}$ the gradient is calculated by first performing a forward pass of input $\boldsymbol{x}$. This allows for computation of the the gradient w.r.t. the output of the final layer, f_n, either analytically or using automatic differentiation. As both the structure and the parameters of the layers are known this can be used to calculate the gradient w.r.t. the parameters in that layer, $\boldsymbol{w}_n$, as well as the output of the previous layer, f_{n-1}. By applying (2.5a) recursively the gradient is propagated through the network and from this (2.5b) gives the gradients needed for the optimization.

$$\frac{\partial \mathcal{L}}{\partial f_k} = \frac{\partial \mathcal{L}}{\partial f_{k+1}}\frac{\partial f_{k+1}}{\partial f_k} \tag{2.5a}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_k} = \frac{\partial \mathcal{L}}{\partial f_k}\frac{\partial f_k}{\partial \boldsymbol{w}_k} \tag{2.5b}$$

Loss functions

For a classification problem with K mutually exclusive classes the categorical cross-entropy is commonly used. With this loss the labels are one-hot encoded meaning that each label is represented by K binary variables, i.e. $y_n \in \mathbb{Z}_2^K$. Each variable represents a class and $y_n^{(k)} = 1$ for the k corresponding to the class of the label and 0 otherwise. The final layer of the model has K outputs with softmax activation. The loss to be minimized is shown in (2.6) [8].

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{W}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} \lambda^{(k)} y_i^{(k)} \log \hat{y}_n^{(k)}(x_n, \boldsymbol{W}) \tag{2.6}$$

where: $y_n^{(k)}$ = the correct binary label of class k for data point n in the training set
$\hat{y}_n^{(k)}$ = the corresponding prediction from the model
$\lambda^{(k)}$ = weight for class k.

The categorical cross-entropy will aim to maximize the predicted probability for the correct class. However, incorrect probabilities have no direct effect on the loss. To be able to affect what kind of errors the model makes in its predictions a modification of this loss can be used. This modified loss, here referred to as confusion-entropy, introduces a matrix, U, which can be seen as a target confusion matrix distribution. Entries in U rewards predictions at the corresponding positions in the confusion matrix, including

possibly incorrect classifications. The confusion-entropy loss is shown in (2.7) [1].

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{W}, U) = -\sum_{i=1}^{K}\sum_{j=1}^{K} u_{ij} \log \sum_{n=1}^{N} y_n^{(i)} \hat{y}_n^{(j)}(x_n, \boldsymbol{W}). \tag{2.7}$$

2.4 Historical background of deep learning

In 1943 McCulloch and Pitts [32] presented a mathematical model of a neuron which at the time had limited capabilities (e.g. it did not learn), but lay the foundations for much of what today is considered to be deep learning. Ivakhnenko and Lapa [23] introduced what would later be called deep learning with the first multi-layered network in 1965. The first convolutional network was introduced by Fukushima in 1980 [17]. A few years later, in 1989, LeCun et al. [27] showed it possible to train such networks with backpropagation and illustrated their effectiveness for computer vision problems. In 2009 Raina et al. [38] suggested that DNNs could efficiently be trained on Graphics Processing Units (GPUs). Krizhevsky et al. [26] used this when they with AlexNet proved it possible to train deeper networks which also greatly outperformed models of the time at computer vision tasks. Since then deep learning based methods has been adopted in various fields, such as computer vision, natural language processing, and even autonomous vehicles [35].

2.5 Explainability

Much of the recent progress in the deep learning space is inherently incomprehensible for us humans, due to its black-box nature and the size of the models [14]. However, explainability is important at many stages of the development of an AI-system. When the systems performance is at sub-human levels it simplifies for human experts to improve it. When the system achieves similar results human experts it can help enforce trust to the system. Finally, in a scenario where the AI outperforms humans it can help us get a better understanding of the problem [41]. With these methods playing a bigger role in fields such as healthcare the importance of explainable decisions also grows from a legal and ethical perspective [4].

Gradient-weighted Class Activation Mapping (Grad-CAM)

Although most deep learning models are not interpretable there are post-hoc methods which tries to explain decisions. Selvaraju et al. [41] suggested one such method, called Gradient-weighted Class Activation Mapping (Grad-CAM), where an activation map is calculated which shows what parts of the data is important for the decisions. Considering a neural network with convolutional layers as feature extractors followed by GAP and dense layers for classification Grad-CAM is based on the final part of the network. Let y_c be the output corresponding to class c and A be the final feature map of height H, width W, and with F filters. The Grad-CAM activation, M_{GC}, is then calculated as follows:

$$
\begin{aligned}
w_k^c &= \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} \frac{\partial y_c}{\partial A_{ij}^k} \\
M_{GC} &= ReLU(\sum_{k=1}^{F} w_k^c A^k)
\end{aligned}
\tag{2.8}
$$

The resulting activation map is importance values $\in \mathbb{R}^{H \times W}$. If the input is a time series this means that by designing the network to not alter the time dimension an importance value is obtained for each time step.

2.6 Consistent Rank Logits (CORAL)

Categorical data with a natural ordering are considered to be ordinal, examples of such data are the response to some medical treatment (e.g. poor, fair, good) [2] or the age of a person [9].

When classifying ordinal data it is desirable to exploit the fact that the categories are ordered [2]. An ordinal classification problem, or ordinal regression as it is also referred to, can be formulated as assigning labels, $y \in \mathcal{Y} = \{\mathcal{C}_0, \mathcal{C}_1, ..., \mathcal{C}_{K-1}\}$, to inputs $\boldsymbol{x}$, where the classes $\mathcal{C}_0 \prec \mathcal{C}_1 \prec ... \prec \mathcal{C}_{K-1}$ according to some ordering relation [9].

Li and Lin [28] presented a method for ordinal regression where the combined result of $K-1$ binary classifiers for K classes were used. Each classifier checked whether the rank of the sample class was larger than rank $r_k \in \{r_1, ... r_{K-1}\}$. Niu et al. [37] developed this further using a multi-output Convolutional Neural Network (CNN) as $K-1$ binary classifiers, called OR-CNN. The classifiers share all weights except the ones in the output layer. This method achieved State of the Art (SOTA) performance on datasets where age was estimated based on facial images. However, consistency was not guaranteed in the predictions, e.g. sometimes simultaneously predicting an age under 20 and over 30. Cao et al. [9] addressed this issue with COnsistent RAnk Logits (CORAL) which is an architecture-agnostic method that can extend any neural network based classifier. Similarly to OR-CNN CORAL uses $K-1$ binary classifiers, here however sharing all weights parameters apart from the biases in the output layer. Instead of representing the labels as one-hot encodings they are now formed as $K-1$ binary labels, i.e. $y_n \in \mathbb{Z}_2^{K-1}$, where $y_n^{(k)} = 1$ if the the rank of the class is greater than r_k and 0 otherwise. By minimizing the loss function

$$
\mathcal{L}(\boldsymbol{x}, \boldsymbol{W}, \boldsymbol{b}) = -\sum_{n=1}^{N} \sum_{k=1}^{K-1} \lambda^{(k)} [\log(\sigma(g(\boldsymbol{x}_n, \boldsymbol{W}) + b_k)) y_n^{(k)} + \log(1 - \sigma(g(\boldsymbol{x}_n, \boldsymbol{W}) + b_k))(1 - y_n^{(k)})],
\tag{2.9}
$$

where: $\boldsymbol{W}$ = all model parameters except biases of final layer
$\boldsymbol{b}$ = bias weights of final layer
$\lambda^{(k)}$ = loss weight for class k
$g(\boldsymbol{x}_n, \boldsymbol{W})$ = output of penultimate layer
$\sigma(z)$ = logistic sigmoid function, $1/(1+\exp(-z))$
$\sigma(g(\boldsymbol{x}_n, \boldsymbol{W}) + b_k)$ = predicted output of binary classifier k

it can be shown that

$$b_1 \geq b_2 \geq ... \geq b_{K-1}. \tag{2.10}$$

The proof can be found in [9] and from this and the shared weights it follows that

$$\widehat{P}(y_n > r_1) \geq \widehat{P}(y_n > r_2) \geq ... \geq \widehat{P}(y_n > r_{K-1}) \tag{2.11}$$

since the only thing that differs between the predictions is the bias. The probabilities for the individual classes are computed from this as

$$\begin{aligned} \widehat{P}(\mathcal{C}_0) &= 1 - \widehat{P}(y_n > r_1) \\ \widehat{P}(\mathcal{C}_1) &= \widehat{P}(y_n > r_1) - \widehat{P}(y_n > r_2) \\ &\vdots \\ \widehat{P}(\mathcal{C}_{K-1}) &= \widehat{P}(y_n > r_{K-1}). \end{aligned} \tag{2.12}$$

Chapter 3

Related work - Human Pose Estimation

Human Pose Estimation (HPE) is a well explored problem which, like many other computer vision tasks has developed rapidly in the recent years. The reasons behind this progress can mainly be explained by two factors. Firstly the emergence of computing power discussed in Section 2.4, allowing more powerful deep learning models. Secondly several datasets with images labeled with human body joints has been made available [10]. These datasets not only provide data, but also introduces competition in the research community making it possible to compare the results of different approaches.

3.1 Datasets

Some of the widely used datasets today are Max Planck Institute for Informatics dataset (MPII) [5], Microsoft Common Objects in Context dataset (COCO) [29], AI Challenger Human Keypoint Detection dataset (AIK-HKD) [50], and COCO-wholebody [24].

The COCO dataset consists of 328k images containing 91 different object types. The images come from Google, Bing, and Flickr image search and are mainly hand annotated through Amazon Mechanical Turk. The interesting part of the dataset for this work is the one with human poses. In total there are 250k instances of people labeled with joint locations [29]. The joints, 17 per person, in the dataset can be seen in Figure 3.1.1a. Along with the datasets containing body keypoints mentioned above there are also datasets with dense keypoints for specific bodyparts, e.g. OneHand10k [48]. COCO-wholebody is an attempt to combine these two types of datasets by extending COCO with dense keypoints at hands, feet, and faces. The resulting 133 joints can be seen in Figure 3.1.1b.

(a) Regular COCO. (b) COCO-wholebody.

Figure 3.1.1: Keypoints for the two COCO datasets.

3.2 Background - Human pose estimation

The HPE problem has been explored since long before the most recent deep learning era. Pictoral Structures were introduced by Fishler and Elschlager in the 1970s. This meant identifying individual parts or features in images modeled with pair-wise spring-like connections [16]. After the progress of deep learning mentioned in Section 2.4 Toshev and Szegedy [45] presented DeepPose, the first HPE method based on DNNs, in 2014. Today's HPE methods are generally categorized as top-down or bottom-up approaches. This has to do with how they handle multiple persons. Bottom-up models starts by finding all keypoints for all persons in an image and then match them together to form persons. Top-down models on the other hand starts by finding bounding boxes for all individuals and then identifies keypoints for one person at a time. The sequential nature of the top-down methods and the fact that two models are needed means that bottom-up models scale better with the number of persons to analyze. However top-down models tends to be more accurate [11].

3.3 Pose estimation models

Below the HPE models used in our work are presented. As we are interested in single person recognition the model used is of top-down type.

3.3.1 High-Resolution Net (HRNet)

Sun et al. presented the High-Resolution Net (HRNet) [43] architecture in 2019, initially for HPE, but also for other computer vision tasks such as semantic segmentation and object detection. Such problems had traditionally been solved using networks built on high-to-low resolution convolutions with increasing numbers of feature maps (e.g ResNet [19], VGGNet [42]). The classification task was solved in the low-resolution space and then transformed back to form the high-resolution representation needed for e.g. the HPE. Sun et al's proposed architecture preserves a high resolution representation throughout the network. It does so while also producing low-resolution/high dimensional representations suitable for classification.

The network architecture is shown in Figure 3.3.1 and consists of four stages (blue blocks in depth direction in Figure) with convolutional layers. After each stage a new low-resolution representation is created by performing strided convolutions. At these instances the existing representations also exchange information by either nearest neighbor upsampling or strided convolutions. The K estimated keypoints are represented as heatmaps, $\{\mathbf{H}_1, \ldots, \mathbf{H}_K\}$, indicating the locations. These heatmaps are formed from the last high-resolution feature map (top right in Figure 3.3.1). Corresponding ground truth heatmaps are generated by applying 2D Gaussians to the correct keypoint locations and the model is trained by minimizing the mean squared error between these [43]. Although a high resolution heatmap is desirable as it gives smaller quantization errors, the computational cost increases quadratically with the size [51]. Hence, the performance of

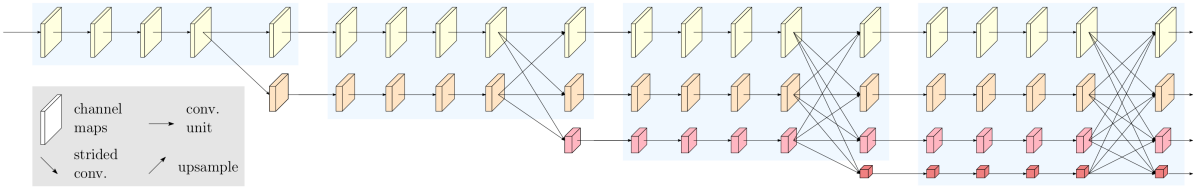


Figure 3.3.1: Network architecture for HRNet. The top row shows high resolution representations with fewer number of feature maps. Each step downwards reduces the resolution with a factor of two while the number of feature maps are doubled [46].

the model can be improved by extracting the region of interest from the input image. This can for instance be done using an object detection model trained to find humans.

Object detectors usually work by first producing a large number of regions of interest in the image which are then classified to either belong to some object class or the background. Faster R-CNN by Ren et al. [39] is an example of such a detector where these steps are performed by a single CNN. The model outputs bounding boxes and class scores for the objects in the image deemed not to be part of the background.

3.3.2 Distribution-Aware coordinate Representation of Key-point (DARK)

As discussed above a high resolution heatmap should result in higher accuracy, but is computationally expensive. Zhang et al. propose a method they call Distribution-Aware coordinate Representation of Key-point (DARK) [51] to reduce the quantization error by i) analyzing the distributions of the predicted heatmaps, and ii) creating the training heatmaps in a slightly new fashion.

The actual keypoint location is found at the maximal activation of the heatmap. Since it is smaller than the actual image this turns into a sub-pixel localisation problem. Newel et al. [36] empirically found that a weighted average between the two highest activations, according to (3.1), yielded a good result.

$$\boldsymbol{p} = \boldsymbol{m} + \frac{1}{4}\frac{\boldsymbol{s}-\boldsymbol{m}}{\|\boldsymbol{s}-\boldsymbol{m}\|_2} \tag{3.1}$$

where: $\boldsymbol{p}$ = predicted maximum
$\boldsymbol{m}$ = highest activation
$\boldsymbol{s}$ = second highest activation

This has been the de facto standard heatmap decoding, but Zhang et al. suggests using the fact that the heatmaps used for training usually are created as 2D Gaussian distributions, i.e. that the heatmaps can be expressed as (3.2).

$$\mathcal{G}(\boldsymbol{x};\boldsymbol{\mu},\Sigma) = \frac{1}{2\pi \mid \Sigma \mid^{\frac{1}{2}}} \exp\Big(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\top \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\Big) \tag{3.2}$$

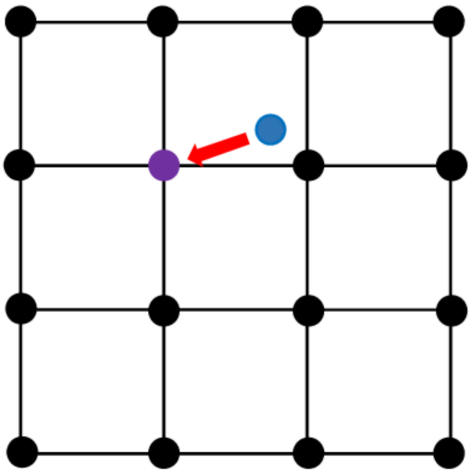


Figure 3.3.2: Quantization error due to off-grid keypoint location. Correct location (blue) represented by on-grid coordinate, here using floor quantization [51]. SKRIV ANTAGLIGEN NAGOT BATTRE HAR!!!!!

where: $\boldsymbol{\mu}$ = maximum of heatmap
$\boldsymbol{x}$ = pixel location
Σ = diagonal covariance matrix

By Taylor expanding of the logarithm of (3.2) in the point $\boldsymbol{m}$, i.e. the point with the highest sampled activation, an expression for $\boldsymbol{\mu}$ is obtained:

$$\boldsymbol{\mu} = \boldsymbol{m} - \Big(\mathcal{D}''(\boldsymbol{m})\Big)^{-1}\mathcal{D}'(\boldsymbol{m}) \tag{3.3}$$

where: $\mathcal{D}(\boldsymbol{x}) = -\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^{\top}\Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})$,
i.e. the non constant term in the logarithm of G (3.2)

The derivatives $\mathcal{D}'(\boldsymbol{m})$ and $\mathcal{D}''(\boldsymbol{m})$ are efficiently estimated from the heatmap. As this approach strongly assumes a Gaussian structure it is proposed to modulate the heatmap before estimating the maximal activation. This is done by performing a convolution with a Gaussian kernel with the same covariance as the one used for the training data.

The second improvement suggested by Zhang et al. concerns the creation of the training heatmaps. Traditionally these have been created from the quantized keypoint locations, resulting in a slightly biased heatmap. In Figure 3.3.2 this would correspond to having the peak activation in the purple dot. By instead using the non-quantized location an unbiased heatmap is obtained. This would correspond to having the peak off-grid, in the blue dot in Figure 3.3.2.

Chapter 4

Related work - Time Series Classification

4.1 Background - Time series classification

Time series are sequences of data ordered in time [H'R KANSKE TA EN FIN DEFINITION FR[N ANDREAS...].

SKA JAG HA DESSA DEFINITIONER? HANVISAR JAG TILL DEM NGNSTANS?

Definition 2 *A univariate time series of length n, with ordered indices*

$$X = [x_1, x_2, ..., x_n]^\top$$

Definition 3 *A multivariate time series of length n, with M channels*

$$\boldsymbol{X} = [X_1, ..., X_M]$$

The TSC task is about finding a function, $f : \mathbb{R}^{n \times M} \rightarrow \mathbb{R}$, that assigns one label to each, possibly multivariate, time series. The problem bares strong resemblance with that of image classification, but with the two spatial dimensions replaced by one temporal dimension. Despite this the use of end-to-end deep learning models is not as dominant in the TSC community [21]. Similarily to the fields of computer vision various datasets has emerged recently. This has been important for the development of TSC as it allows for fair comparison between methods. One of the most widely used dataset collections today is the University of California, Riverside (UCR) archive [13] containing 85 different time series datasets.

Traditionally a nearest neighbor method together with dynamic time warping has been used for classification [7]. Simply put, this means that a time series during classification is

compared to the training data and assigned the class of the most similar time series. Lines and Bagnall suggested a method where an ensemble of 11 nearest neighbor classifiers with different similarity measures [31] yielded SOTA results. Bagnall et al. [6] developed the idea of ensemble based classifiers with Collective Of Transformation-based Ensembles (COTE), where 35 different classifiers using different transforms was used. Lines et al. [30] extended COTE further with two new classifiers resulting in Hierarchical Vote Collective of Transformation-based Ensembles (HIVE-COTE). One drawback with HIVE-COTE is the computational intensity, both during training and test time. Training time is large partly due to one of the transforms used is the Shapelet Transform with a time complexity of $O(n^2l^4)$, n being the number of time series and l the length of them. Due to the nature of the nearest neighbor algorithm the result of the 37 classifiers during test time needs to be compared to the corresponding result for each time series in the training set, yielding this method impractical for real-time use [21].

In 2016 Zheng et al. [52] presented a neural network model based on convolutional layers for the classification task. Wang et al. [49] developed these ideas and presented models with performance close to that of COTE on the UCR archive. The development of neural network based classification has since then continued and below the two architectures inspiring our model are presented.

4.2 Deep learning architectures

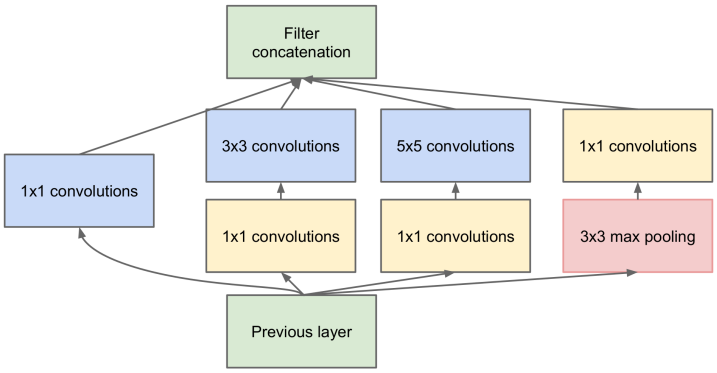
The last few years the number of proposed neural network based time series classifiers has increased drastically. Below the two most influential architectures for our work are presented.

4.2.1 InceptionTime

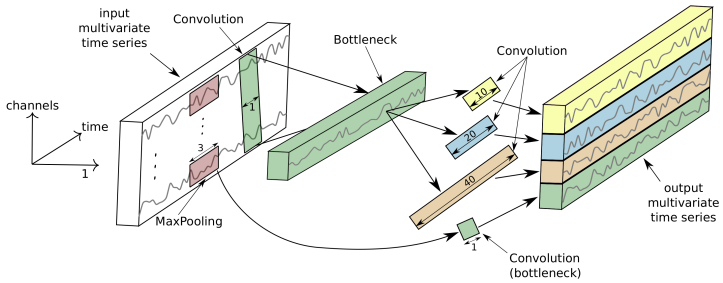
InceptionTime, presented by Fawaz et al. [22], is, as the name suggests, inspired by Inception [44] which is an architecture successful in computer vision tasks. It is comprised of several stacked Inception modules consisting of differently sized convolutions as well as pooling layers. To reduce the number of parameters in the network 1×1 convolutions are often used as a dimensionality reduction. One such module can be seen in Figure 4.2.1a. The architecture of Fawaz et al. is similar, but with only one temporal dimension instead of two spatial dimensions. As with the computer vision task the dimensionality is reduced, here through a bottleneck of size m. The bottleneck is achieved by convolutions with m filters of length 1. The InceptionTime module is shown in Figure 4.2.1b.

Figure 4.2.2 shows how stacked modules makes up the InceptionTime architecture. Residual connections are used to decrease the risk of vanishing gradients once the network becomes deeper, as suggested by He et al. [19]. The InceptionTime modules are followed by a GAP layer which averages each time series over its time dimension. The classification is performed by fully connected layers with softmax activations.

Many deep learning based time series classifiers experiences a significant variance in



(a)



(b)

Figure 4.2.1: Inception modules for computer vision (a) with dimensionality reduction ahead of the 3×3 and 5×5 convolutions and InceptionTime module for TSC (b), here illustrated with a bottleneck size of 1. Figures from [44] and [22] respectively.

their accuracy, especially when evaluated on the UCR archive with rather small training sets [20]. To overcome this Fawaz et al. suggest the use of an ensemble of identical InceptionTime networks, but with randomly initialized weights before training. The ensemble's output is then calculated as the average of the outputs of the individual models. Such an ensemble of five models achieves a performance similar to that of HIVE-COTE on the UCR archive.

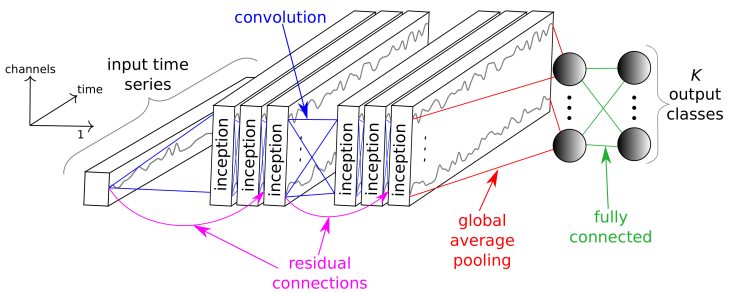


Figure 4.2.2: InceptionTime architecture for TSC [22].

4.2.2 Explainable Convolutional Neural Network for Multivariate Time Series Classification (XCM)

As discussed in Section 2.5 explainability is desirable, but not inherent in most black-box deep learning models. Fauvel et al. [15] propose an architecture, Explainable Convolutional Neural Network for Multivariate Time Series Classification (XCM), which allows for tracking which time steps and which inputs are important for the classification decision. By using 2D convolutions with kernels of size $ks \times 1$, where ks is the kernel size hyperparameter, the convolution is only performed in the time dimension and the input channels are kept separated throughout the feature extraction. Through dimensionality reduction from a 1×1 2D convolution a single feature map for each input is produced. From this the importance of input channels and time steps can be traced using Grad-CAM, described in Section 2.5. In parallel to the channel specific features Fauvel et al. also suggests using 1D convolutions over all channels resulting in a combined feature map along the time dimension. The XCM architecture is depicted in Figure 4.2.3.

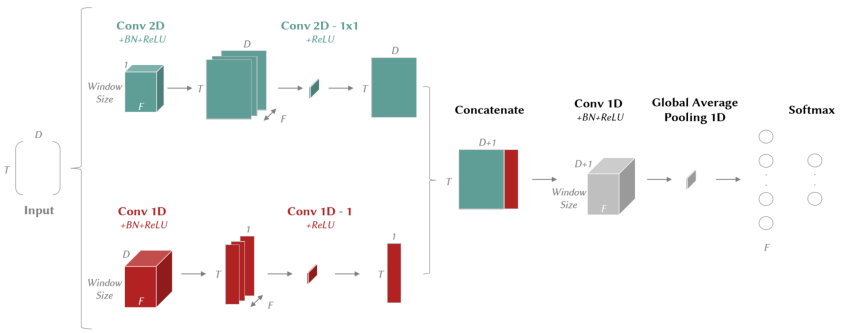


Figure 4.2.3: The XCM architecture with BN - Batch Normalization, D - number of input channels, F - number of filters, T - length of time series [15].

Chapter 5

Methods

5.1 Overview

In this chapter the system for assessing POEs will be presented. This system is naturally divided into two parts where firstly the videos are analyzed. The first subsystem extracts body part coordinates of the subjects. This information is then passed to the second subsystem where it is used to calculate a score according to [34]. The data is presented in Section 5.2 and the two subsystems are described in Sections 5.3 and 5.4 respectively.

5.2 Data

The data available is in the form of videos each containing one subject, recorded from the front, performing three-five repetitions of specific motions. The motions are *Single-leg squat, Forward lunge, Stair descending, Forward lunge, Single leg hop for distance, and Side hop*. Each motion has a number of POE scores associated with them. The motion-POE combinations are shown in Table 5.2.1. In Sections 5.2.1-**??** the motions and POEs evaluated in this project are described.

bara skriva om sls? om det bara 'r sls jag bed;mt.. ocks[skriva det som avgr'nsningar

5.2.1 Single-leg squat, SLS

The subject performed a squat standing on one leg to a knee angle of approximately 60°. The exercise was repeated five times and the entire movement was used to assess the POEs [3]. An illustration is shown in Figure **??**.

5.2.2 Stair descending, SD

The subject stepped down from a 30 cm step board. The exercise was repeated five times and POEs were evaluated for the loaded leg during loading phase [3]. An illustration is

Table 5.2.1: Motion-POE combinations available in the data.

POE \ Motion	Single leg squat	Stair descending	Forward lunge	Single leg hop for distance	Side hop
Trunk	x	x		x	x
Hip	x	x	x	x	x
Femoral valgus	x	x	x	x	x
Knee medial to foot position	x	x	x	x	x
Femur medial to shank	x	x	x	x	x
Foot	x				

shown in Figure **??**.

5.2.3 Forward Lunge, FL

Helo I am now mispealing.

5.3 Body part localization

The pose estimation is built around the open-source toolbox MMPose [33] from MMLab. Each frame is considered to be an independent image and is analyzed with a HRNet model with the DARK extension trained on the COCO-wholebody dataset[1]. Both the model and the dataset is described in Section 3.

To get comparable results some of the videos were rotated and flipped before inferring the keypoints. This was needed since the videos were recorded in different orientations and the actions were performed with different legs. The rotations were based on the orientation of the subject (position of head w.r.t. the feet) in the first frame to have it standing up in the y-direction. Videos where the squats were performed with the left leg were then flipped around the y-axis to be able to use the same model for the left and right leg in a more efficient manner.

A bounding box for the subject is found using a Faster R-CNN model trained on the COCO dataset[2]. The content of this bounding box is resized to match the input size of the HPE model used, 384×288 pixels in our case. Each video analyzed results in sequences of x- and y-coordinates for all the keypoints in the dataset used to train the model.

[1]The model used can be found here: `https://mmpose.readthedocs.io/en/latest/top_down_models.html`.

[2]The model used can be found here: `https://github.com/open-mmlab/mmdetection/tree/master/configs/faster_rcnn`.

5.4 Classification

5.4.1 Preprocessing and dataset blabla..

Before assessing the Postural Orientation Errors (POEs) based on the body part positions a number of preprocessing steps are conducted. Firstly the data is resampled as the videos are recorded with a number of different frame rates ranging from 25 to 60 Hz. The resampling is performed using linear interpolation to a new sample frequency of 25Hz. This data is then low pass filtered through a fourth order Butterworth filter with a cutoff frequency of 2.5Hz.

While the POE assessment, see Section **??**, is performed on a per repetition basis the body part coordinates are extracted on a per video basis. Hence, the sequences corresponding to the entire video is split up in the individual repetitions. This splitting algorithm is presented in Algorithm 1 and is based on finding the edges of the peaks in specific position data. For the Single Leg Squat (SLS) task the y-coordinate of the right shoulder is used. The number of points extracted for each repetition depends on the width of the peak. The length of the observed repetitions varies from about 1 to 8 seconds. For practical reasons, such as handling of data and training performance[3], it is desirable to save the data as multidimensional arrays with the same dimensions. Two different ways of solving this problem is evaluated, namely i) padding the sequences and use maskings for the padded samples in the models, and ii) alternate the sample frequency to thereby achieve sequences of the same length.

Algorithm 1: Extraction of repetitions from sequences

```
right_edges, left_edges = find_edges(sequence);
for peak, right, current_left, next_left in peaks, right_edges, left_edges do
    split_index = mean(right, next_left);
    start = max(current_left - extra_points, 0);
    end = min(right + extra_points, split_index);
    repetition = normalize_length(sequence[start:end]);
    sequence = sequence[end:];
end
```

Finally the data is normalized. All coordinates are moved to put the mean position of the first five right hip-samples in the origin and are scaled to set the distance between the right shoulder and right hip to one, according to (5.1).

$$\begin{aligned}(x,y)_i &= (x,y)_i - \overline{(x,y)}_{rh} \\ (x,y)_i &= \frac{(x,y)_i}{\|\overline{(x,y)}_{rs}\|_2} \qquad ,\forall i\end{aligned} \tag{5.1}$$

[3]All data in one batch must have the same size. Hence, to be able to train with a batch size larger than 1, which usually improves training performance [18], all data in the same batch needs to have the same dimensions.

where: $\overline{(x, y)}_i$ = mean over first five samples for body part i
rh = right hip
rs = right shoulder
i $\in$ Available body parts

After these preprocessing steps a dataset with inputs $\in \mathbb{R}^{N \times T \times F}$ and corresponding labels $\in \mathbb{Z}_3^N$ is created. The inputs consists of N multivariate time series of length T with F channels. These channels are a subset of the extracted x- and y-coordinates as well as angles and differences between keypoints.

5.4.2 Models

For the modeling we evaluated some different deep learning based model architectures. The models eventually used were InceptionTime (Section 4.2.1) with different loss functions as well as an architecture designed by us, inspired by XCM (Section 4.2.2) and InceptionTime. We call this model X-InceptionTime and it is presented below.

skriv om ensemble, vilka losses, maskning, icke maskning

X-InceptionTime

The idea with this model was to combine the explainability of XCM with the inception module from InceptionTime. This was done by separating the inputs and having individual inception modules for each input channel as can be seen in Figure 5.4.1. After the final module (the depth can be seen as a hyperparamtere and needs to be tuned) a bottleneck of size one is applied reducing the dimensionality of each input channel back to $T \times 1$. The features for the individual inputs are concatenated resulting in a feature map of size $T \times F$ where each input feature is only affected by that input. This makes it possible to use Grad-CAM to see the importance of each time step for each input. Along with this it is also possible, thanks to the GAP layer, to get a measure of the importance of each input which can be used to reduce the dimensionality of the input.

grad cam fr[n xinception

coral osv. custom losses osv?

5.4.3 Training

5.4.4 Choice of input features

5.4.5 Combined score

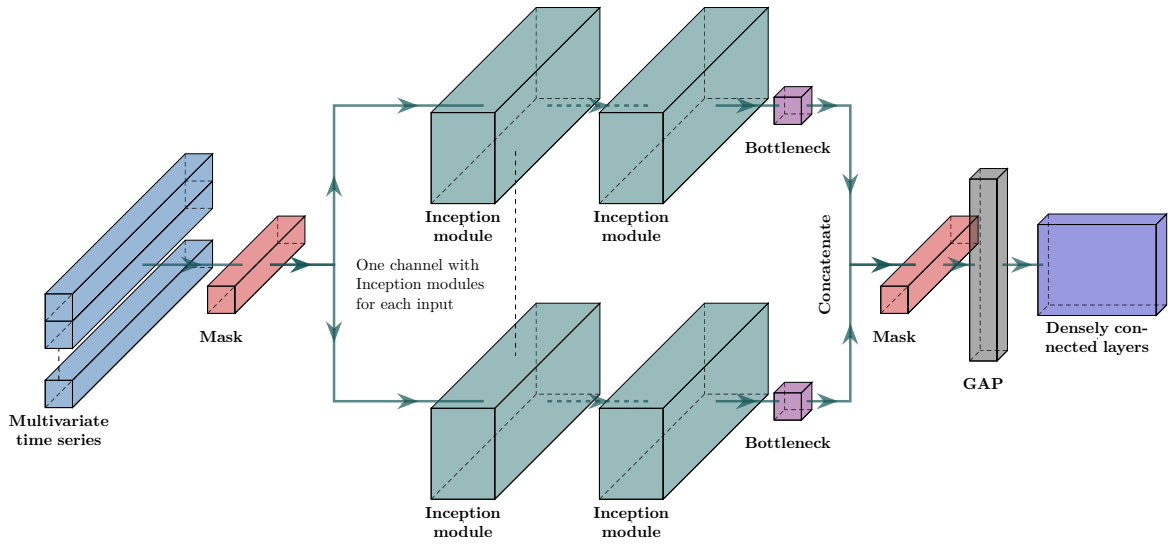


Figure 5.4.1: The X-InceptionTime architecture developed in this work.

Chapter 6

Results

Chapter 7

Conclusions and Discussion

Bibliography

[1] Hussein A. Abbass, Jason Scholz, and Darryn J. Reid. *Foundations of Trusted Autonomy*. Springer, 2018. DOI: 10.1007/978-3-319-64816-3.

[2] Alan Agresti. *An Introduction to Categorical Data Analysis (2nd ed.)* Hoboken, New Jersey: Wiley, 2007.

[3] Jenny Älmqvist Nae. "Is seeing just believing? Measurement properties of visual assessment of Postural Orientation Errors (POEs) in people with anterior cruciate ligament injury". English. PhD thesis. Department of Health Sciences, June 2020. ISBN: 978-91-7619-940-4.

[4] Julia Amann, Alessandro Blasimme, Effy Vayena, Dietmar Frey, and Vince I. Madai. "Explainability for artificial intelligence in healthcare: a multidisciplinary perspective". In: *BMC Medical Informatics and Decision Making* 20.1 (Dec. 2020), p. 310. ISSN: 14726947. DOI: 10.1186/s12911-020-01332-6.

[5] M. Andriluka, L. Pishchulin, P. Gehler, and B. Schiele. "2D Human Pose Estimation: New Benchmark and State of the Art Analysis". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 3686–3693. DOI: 10.1109/CVPR.2014.471.

[6] A. Bagnall, J. Lines, J. Hills, and A. Bostrom. "Time-Series Classification with COTE: The Collective of Transformation-Based Ensembles". In: *IEEE Transactions on Knowledge and Data Engineering* 27.9 (2015), pp. 2522–2535. DOI: 10.1109/TKDE.2015.2416723.

[7] Anthony Bagnall, Jason Lines, Aaron Bostrom, James Large, and Eamonn Keogh. "The great time series classification bake off: a review and experimental evaluation of recent algorithmic advances". In: *Data Mining and Knowledge Discovery* 31.3 (May 2017), pp. 606–660. ISSN: 1573756X. DOI: 10.1007/s10618-016-0483-9.

[8] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[9] Wenzhi Cao, Vahid Mirjalili, and Sebastian Raschka. "Rank consistent ordinal regression for neural networks with application to age estimation". In: *Pattern Recognition Letters* 140 (Jan. 2019), pp. 325–331. DOI: 10.1016/j.patrec.2020.11.008. arXiv: 1901.07884.

[10] Yucheng Chen, Yingli Tian, and Mingyi He. "Monocular Human Pose Estimation: A Survey of Deep Learning-based Methods". In: *Computer Vision and Image Understanding* 192 (June 2020). DOI: 10.1016/j.cviu.2019.102897. arXiv: 2006.01423.

[11] Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S. Huang, and Lei Zhang. "HigherHRNet: Scale-Aware Representation Learning for Bottom-Up Human Pose Estimation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (Aug. 2019), pp. 5385–5394. arXiv: 1908.10357.

[12] François Chollet. *Deep Learning with Python*. Manning, 2018.

[13] Hoang Anh Dau, Anthony Bagnall, Kaveh Kamgar, Chin-Chia Michael Yeh, Yan Zhu, Shaghayegh Gharghabi, Chotirat Ann Ratanamahatana, and Eamonn Keogh. "The UCR Time Series Archive". In: *IEEE/CAA Journal of Automatica Sinica* 6.6 (Oct. 2018), pp. 1293–1305. arXiv: 1810.07758.

[14] Mengnan Du, Ninghao Liu, and Xia Hu. "Techniques for Interpretable Machine Learning". In: *arXiv* (July 2018). arXiv: 1808.00033.

[15] Kevin Fauvel, Tao Lin, Véronique Masson, Élisa Fromont, and Alexandre Termier. "XCM: An Explainable Convolutional Neural Network for Multivariate Time Series Classification". In: (Sept. 2020). arXiv: 2009.04796.

[16] Martin A. Fischler and Robert A. Elschlager. "The Representation and Matching of Pictorial Structures Representation". In: *IEEE Transactions on Computers* C-22.1 (1973), pp. 67–92. ISSN: 00189340. DOI: 10.1109/T-C.1973.223602.

[17] Kunihiko Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position". In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. ISSN: 03401200. DOI: 10.1007/BF00344251.

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

[19] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2016-December. IEEE Computer Society, Dec. 2016, pp. 770–778. ISBN: 9781467388504. DOI: 10.1109/CVPR.2016.90. arXiv: 1512.03385.

[20] H. Ismail Fawaz, G. Forestier, J. Weber, L. Idoumghar, and P. Muller. "Deep Neural Network Ensembles for Time Series Classification". In: *2019 International Joint Conference on Neural Networks (IJCNN)*. 2019, pp. 1–6. DOI: 10.1109/IJCNN.2019.8852316.

[21] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre Alain Muller. "Deep learning for time series classification: a review". In: *Data Mining and Knowledge Discovery* 33.4 (July 2019), pp. 917–963. ISSN: 1573756X. DOI: 10.1007/s10618-019-00619-1. arXiv: 1809.04356.

[22] Hassan Ismail Fawaz, Benjamin Lucas, Germain Forestier, Charlotte Pelletier, Daniel F. Schmidt, Jonathan Weber, Geoffrey I. Webb, Lhassane Idoumghar, Pierre Alain Muller, and François Petitjean. "InceptionTime: Finding AlexNet for time series classification". In: *Data Mining and Knowledge Discovery* 34.6 (Nov. 2020), pp. 1936–1962. ISSN: 1573756X. DOI: 10.1007/s10618-020-00710-y. arXiv: 1909.04939.

[23] A. G. Ivakhnenko and V. G. Lapa. *Cybernetic Predicting Devices*. CCM Information Corpo-ration, 1965.

[24] Sheng Jin, Lumin Xu, Jin Xu, Can Wang, Wentao Liu, Chen Qian, Wanli Ouyang, and Ping Luo. "Whole-Body Human Pose Estimation in the Wild". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 12354 LNCS (July 2020), pp. 196–214. arXiv: 2007.11858.

[25] Diederik P. Kingma and Jimmy Lei Ba. "Adam: A method for stochastic optimization". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, Dec. 2015. arXiv: 1412.6980.

[26] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: ed. by Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, Léon Bottou, and Kilian Q. Weinberger. 2012, pp. 1106–1114.

[27] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. "Backpropagation Applied to Handwritten Zip Code Recognition". In: *Neural Computation* 1.4 (Dec. 1989), pp. 541–551. ISSN: 0899-7667. DOI: 10.1162/neco.1989.1.4.541.

[28] Ling Li and Hsuan-tien Lin. "Ordinal Regression by Extended Binary Classification". In: *Advances in Neural Information Processing Systems*. Ed. by B. Schölkopf, J. Platt, and T. Hoffman. Vol. 19. MIT Press, 2007, pp. 865–872.

[29] Tsung Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. "Microsoft COCO: Common objects in context". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8693 LNCS. PART 5. Springer Verlag, May 2014, pp. 740–755. DOI: 10.1007/978-3-319-10602-1_48. arXiv: 1405.0312.

[30] J. Lines, S. Taylor, and A. Bagnall. "HIVE-COTE: The Hierarchical Vote Collective of Transformation-Based Ensembles for Time Series Classification". In: *2016 IEEE 16th International Conference on Data Mining (ICDM)*. 2016, pp. 1041–1046. DOI: 10.1109/ICDM.2016.0133.

[31] Jason Lines and Anthony Bagnall. "Time series classification with ensembles of elastic distance measures". In: *Data Mining and Knowledge Discovery* 29.3 (Apr. 2015), pp. 565–592. ISSN: 13845810. DOI: 10.1007/s10618-014-0361-2.

[32] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 00074985. DOI: 10.1007/BF02478259.

[33] *MMPose - OpenMMLab Pose Estimation Toolbox and Benchmark*. URL: https://github.com/open-mmlab/mmpose.

[34] Jenny Nae, Mark W. Creaby, Gustav Nilsson, Kay M. Crossley, and Eva Ageberg. "Measurement Properties of a Test Battery to Assess Postural Orientation During Functional Tasks in Patients Undergoing ACL Injury Rehabilitation". In: *Journal of Orthopaedic & Sports Physical Therapy* 47.11 (Oct. 2017), pp. 1–42. ISSN: 0190-6011. DOI: 10.2519/jospt.2017.7270. URL: http://www.jospt.org/doi/10.2519/jospt.2017.7270.

[35] Md Nazmus Saadat and Muhammad Shuaib. "Advancements in Deep Learning Theory and Applications: Perspective in 2020 and beyond". In: *Advances and Applications in Deep Learning*. IntechOpen, Dec. 2020. DOI: 10.5772/intechopen.92271.

[36] Alejandro Newell, Kaiyu Yang, and Jia Deng. "Stacked hourglass networks for human pose estimation". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9912 LNCS. Springer Verlag, 2016, pp. 483–499. ISBN: 9783319464831. DOI: 10.1007/978-3-319-46484-8_29. arXiv: 1603.06937.

[37] Zhenxing Niu, Mo Zhou, Le Wang, Xinbo Gao, and Gang Hua. "Ordinal Regression with Multiple Output CNN for Age Estimation". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4920–4928. DOI: 10.1109/CVPR.2016.532.

[38] Rajat Raina, Anand Madhavan, and Andrew Y. Ng. "Large-Scale Deep Unsupervised Learning Using Graphics Processors". In: *Proceedings of the 26th Annual International Conference on Machine Learning*. ICML '09. Montreal, Quebec, Canada: Association for Computing Machinery, 2009, pp. 873–880. ISBN: 9781605585161. DOI: 10.1145/1553374.1553486.

[39] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.6 (June 2017), pp. 1137–1149. ISSN: 01628828. DOI: 10.1109/TPAMI.2016.2577031. arXiv: 1506.01497.

[40] D. E. Rumelhart and J. L. McClelland. "Learning Internal Representations by Error Propagation". In: *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*. 1987, pp. 318–362.

[41] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization". In: *International Journal of Computer Vision* 128.2 (Oct. 2016), pp. 336–359. DOI: 10.1007/s11263-019-01228-7. arXiv: 1610.02391.

[42] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*. International Conference on Learning Representations, ICLR, Sept. 2015. arXiv: 1409.1556.

[43] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. "Deep high-resolution representation learning for human pose estimation". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 2019-June. IEEE Computer Society, June 2019, pp. 5686–5696. ISBN: 9781728132938. DOI: 10.1109/CVPR.2019.00584. arXiv: 1902.09212.

[44] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "Going deeper with convolutions". In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. Vol. 07-12-June-2015. IEEE Computer Society, Oct. 2015, pp. 1–9. ISBN: 9781467369640. DOI: 10.1109/CVPR.2015.7298594. arXiv: 1409.4842.

[45] A. Toshev and C. Szegedy. "DeepPose: Human Pose Estimation via Deep Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1653–1660. DOI: 10.1109/CVPR.2014.214.

[46] J. Wang, K. Sun, T. Cheng, B. Jiang, C. Deng, Y. Zhao, D. Liu, Y. Mu, M. Tan, X. Wang, W. Liu, and B. Xiao. "Deep High-Resolution Representation Learning for Visual Recognition". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020), pp. 1–1. DOI: 10.1109/TPAMI.2020.2983686.

[47] Shui-Hua Wang, Chaosheng Tang, Junding Sun, Jingyuan Yang, Chenxi Huang, Preetha Phillips, and Yu-Dong Zhang. "Multiple Sclerosis Identification by 14-Layer Convolutional Neural Network With Batch Normalization, Dropout, and Stochastic Pooling". In: *Frontiers in Neuroscience* 12 (2018), p. 818. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00818.

[48] Yangang Wang, Cong Peng, and Yebin Liu. "Mask-Pose Cascaded CNN for 2D Hand Pose Estimation from Single Color Image". In: *IEEE Transactions on Circuits and Systems for Video Technology* 29.11 (Nov. 2019), pp. 3258–3268. ISSN: 15582205. DOI: 10.1109/TCSVT.2018.2879980.

[49] Zhiguang Wang, Weizhong Yan, and Tim Oates. "Time series classification from scratch with deep neural networks: A strong baseline". In: *Proceedings of the International Joint Conference on Neural Networks*. Vol. 2017-May. Institute of Electrical and Electronics Engineers Inc., June 2017, pp. 1578–1585. ISBN: 9781509061815. DOI: 10.1109/IJCNN.2017.7966039. arXiv: 1611.06455.

[50] Jiahong Wu, He Zheng, Bo Zhao, Yixin Li, Baoming Yan, Rui Liang, Wenjia Wang, Shipei Zhou, Guosen Lin, Yanwei Fu, Yizhou Wang, and Yonggang Wang. "AI Challenger : A Large-scale Dataset for Going Deeper in Image Understanding". In: *arXiv* (Nov. 2017). arXiv: 1711.06475.

[51] Feng Zhang, Xiatian Zhu, Hanbin Dai, Mao Ye, and Ce Zhu. "Distribution-Aware Coordinate Representation for Human Pose Estimation". In: Institute of Electrical and Electronics Engineers (IEEE), Aug. 2020, pp. 7091–7100. DOI: 10.1109/cvpr42600.2020.00712. arXiv: 1910.06278.

[52] Yi Zheng, Qi Liu, Enhong Chen, Yong Ge, and J. Leon Zhao. "Exploiting multi-channels deep convolutional neural networks for multivariate time series classification". In: *Frontiers of Computer Science* 10.1 (Feb. 2016), pp. 96–112. ISSN: 20952236. DOI: 10.1007/s11704-015-4478-2.