FACULTY OF SCIENCE
UNIVERSITY OF COPENHAGEN

Master Thesis

2D Tracking in Climbing

Using Temporal Smoothing

André Oskar Andersen (wpr684)
wpr684@alumni.ku.dk

2023

Supervisor
Kim Steenstrup Pedersen kimstp@di.ku.dk

Abstract

Preface

Acknowledgement

Contents

Notation

$\boldsymbol{x}$	A vector
$\boldsymbol{X}$	A matrix
X	A Tensor
$\boldsymbol{X}_{ij}$	Element located at row i column j in matrix $\boldsymbol{X}$
$*$	The convolution operator
$\circ$	The Hadamard product
σ	The sigmoid activation function

1 Introduction

1.1 Related Work

2-dimensional pose estimation can be divided into either being image-based or video-based, where the methods in the latter case use the tempoeral information of the video to perform the pose estimation.

Image-based methods were initially based on the geometry between the joints of the taget image [14, 17, 22]. Following this, were the convolutional-based methods, that used convolutional neural networks [9] to perform the pose estimation [19, 12, 3, 6]. More recent methods use transformers [18] to deliver state-of-the-art results [20, 21].

Early video-based methods used 3-dimensional convolutions to capture the temporal information between neighboring frames [13, 4]. Other methods use LSTM's [8] to capture this temporal information [10, 2]. Like in the case of image-based methods, transformers [18] have recently been introduced to the video-based methods to capture the temporal information and deliver state-of-the-art-results [23].

1.2 Problem Definition

1.3 Reading Guide

2 Deep Learning Theory

The following section covers the most important background theory for the experiments in Section 5. This includes an introduction to various types of neural networks, as well as an introduction to the optimization of such networks.

2.1 Feedforward Neural Networks

Feedforward neural networks are the most basic type of neural networks. The aim of a feedforward neural network is to approximate some function f^*, by defining a mapping $\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta})$ and learning the parameters $\boldsymbol{\theta}$, that results in the best approximation of f^*. These models are called **feedforward** because there are no **feedback** connections in which the outputs of the model are fed back into itself. Instead, information flows through the function being evaluated from $\boldsymbol{x}$, through the intermediate computations used to define f, and finally to the output $\boldsymbol{y}$. Feedforward neural networks generally consists of multiple **layers**, arranged in a chain structure, with each layer being a function of the layer that preceded it [5].

2.1.1 Fully-connected Layers

The most simple type of layer found in a feeforward neural network is the **fully-connected layer**. The fully-connected layer usually consists of some learnable parameter matrix $\boldsymbol{W}$ and learnable parameter vector $\boldsymbol{b}$, as well as a non-linear **activation function** g (which will be covered further in Section 2.4.3). In this case, the i'th layer is defined as

$$\boldsymbol{h}^{(i)} = \begin{cases} g^{(i)}\left(\boldsymbol{W}^{(i)\top}\boldsymbol{h}^{(i-1)} + \boldsymbol{b}^{(i)}\right) & \text{if } i > 1 \\ g^{(1)}\left(\boldsymbol{W}^{(1)\top}\boldsymbol{x} + \boldsymbol{b}^{(1)}\right) & \text{if } i = 1 \end{cases}. \tag{1}$$

Thus, for a neural network with n layers, we have the mapping [5]

$$\boldsymbol{y} = f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{h}^{(n)}. \tag{2}$$

2.1.2 Convolutional Layer

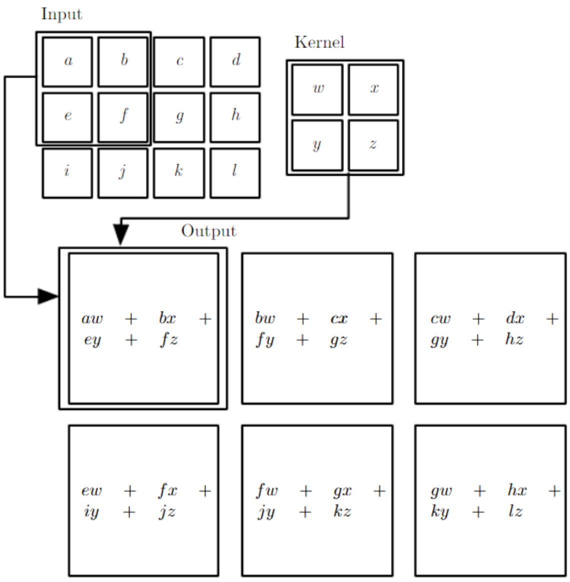

Figure 1: An example of applying a 2d kernel on an input [5].

A **convolutional layer** is a specialized kind of feedforward layer, usually used in analysis of time-series or image data. If a network has at least one convolutional layer, it is called a **Convolutional neural network (CNN)** [5].

The convolutional layer consists of a set of **kernels**, each to be applied to the entire input vector, where each kernel is a learnable parameter matrix $k \times k$ [15]. Each kernel is applied on the input to produce a **feature map**. The kernels are applied to the input by "sliding" over the input (where the step size is called **stride**). Each $k \times k$ grid of the input is then used to compute the dot-product between the grid and each kernel, which is then placed in the corresponding feature map of each kernel, as visualized in Figure 1. [1]. To control the dimensions of the output, one might **pad** the sides with a constant value. Commonly, zero is used as the padding-value.

As seen in Figure 1, each kernel produces a linear combination of all pixel values in a neighbourhood defined by the size of the kernel. Thus, unlike a fully-connected layer, a convolutional layer captures the high correlation between a pixel and its neighbours. Further, by limiting the size of the kernel, the network will use much fewer parameters, than if it was replaced by a fully-connected layer [5].

2.2 Recurrent Neural Networks

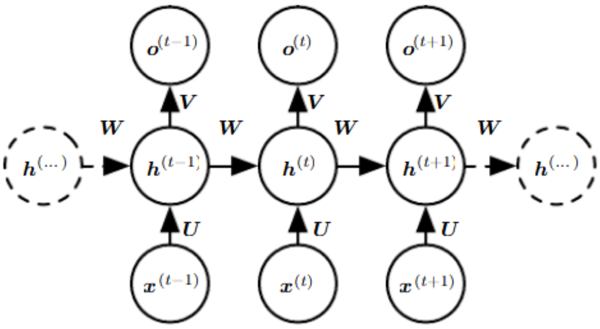

Figure 2: An illustration of an RNN [5].

Recurrent neural networks (RNNs) are a family of neural networks for processing sequential data. Figure 2 illustrates the general setup of such a network, which maps an input sequence of $\boldsymbol{x}$ values to a corresponding sequence of output $\boldsymbol{o}$ values. Generally, a RNN consists of three parts: (1) the input ($\boldsymbol{x}^{(i)}$), (2) the hidden state ($\boldsymbol{h}^{(i)}$), and (3) the output ($\boldsymbol{o}^{(i)}$). During inference, the model maps each input value to an output value in a sequential matter, where it first maps the first input value, then the second, then the third, and so forth. The network maps each input value to an output value by making use of the hidden state from the preceding step, where the first hidden state has to be initiallized [5].

2.2.1 Convolutional Long Short-Term Memory

One common recurrent neural network unit is the **convolutional long short-term memory (ConvLSTM)**, which is an adaptation of the standard **long short-term memory (LSTM)** for sequences of images. Both LSTMs work by potentially stacking multiple **cells** together, such that each LSTM cell is taking in outputs of the preceding LSTM cell as its input.

The idea of both LSTM cells is to create paths through time that have derivatives that neither vanish nor explode. This is done by introducing a memory cell C_t, which accumulates state information over a long duration. This cell is accessed, written and cleared by several controlling gates. The model learns during training, when to access, write and clear this memory cell. By using the memory cell and gates to control the flow of information, the gradient will be trapped in the cell and thus be prevented from vanishing too quickly [5, 16].

The ConvLSTM cell consists of three gates. The first gate is the input gate I_t, which controls whether or not to accumulate the information of a new input to the cell. This gate is defined as

$$\mathsf{I}_t = \sigma\left(\mathsf{W}_{xi} * \mathsf{X}_t + \mathsf{W}_{hi} * \mathsf{H}_{t-1} + \mathsf{W}_{ci} \circ \mathsf{C}_{t-1} + \mathsf{B}_i\right) \tag{3}$$

where X_t is the current input image, H_{t-1} is the hidden state of the previous time step, C_{t-1} is the cell output of the previous time step, and W_{xi}, W_{hi}, W_{ci} and B_i are learnable parameters

The second gate is the **forget gate** unit F_t (at time step t), which controls whether or not to "forget" the status of the cell output of the previous time step. The forget gate is defined as

$$\mathsf{F}_t = \sigma\left(\mathsf{W}_{xf} * \mathsf{X}_t + \mathsf{W}_{hf} * \mathsf{H}_{t-1} + \mathsf{W}_{cf} \circ \mathsf{C}_{t-1} + \mathsf{B}_f\right) \tag{4}$$

where W_{xf}, W_{hf}, W_{cf} and B_f are learnable parameters.

The last gate is the **output gate** unit O_t, which controls whether or not the latest cell output will be proagated to the final state H_t. This cell is defined as

$$\mathsf{O}_t = \sigma\left(\mathsf{W}_{xo} * \mathsf{X}_t + \mathsf{W}_{ho} * \mathsf{H}_{t-1} + \mathsf{W}_{co} \circ \mathsf{C}_t + \mathsf{B}_o\right) \tag{5}$$

where W_{xo}, W_{ho}, W_{co} and B_o are learnable parameters.

By combining the three gates, we get the following definition of the ConvLSTM Cell. Let $\mathsf{X}_1, \mathsf{X}_2, ..., \mathsf{X}_t$ be a sequence of input images. Then, at each time step t, we compute

$$\begin{aligned}
\mathsf{I}_t &= \sigma\left(\mathsf{W}_{xi} * \mathsf{X}_t + \mathsf{W}_{hi} * \mathsf{H}_{t-1} + \mathsf{W}_{ci} \circ \mathsf{C}_{t-1} + \mathsf{B}_i\right) && (6)\\
\mathsf{F}_t &= \sigma\left(\mathsf{W}_{xf} * \mathsf{X}_t + \mathsf{W}_{hf} * \mathsf{H}_{t-1} + \mathsf{W}_{cf} \circ \mathsf{C}_{t-1} + \mathsf{B}_f\right) && (7)\\
\mathsf{C}_t &= \mathsf{F}_t \circ \mathsf{C}_{t-1} + \mathsf{I}_t \circ \tanh\left(\mathsf{W}_{xc} * \mathsf{X}_t + \mathsf{W}_{hc} * \mathsf{H}_{t-1} + \mathsf{B}_c\right) && (8)\\
\mathsf{O}_t &= \sigma\left(\mathsf{W}_{xo} * \mathsf{X}_t + \mathsf{W}_{ho} * \mathsf{H}_{t-1} + \mathsf{W}_{co} \circ \mathsf{C}_t + \mathsf{B}_o\right) && (9)\\
\mathsf{H}_t &= \mathsf{O}_t \circ \tanh(\mathsf{C}_t) && (10)
\end{aligned}$$

For $t = 1$, both H_{t-1} and C_{t-1} have to be initialized [16].

2.2.2 Gated Recurrent Unit

2.3 Transformer

The sequential nature of RNNs precludes parallelization withing training examples, heavily slowing down the training of these models. Another type of model for sequential data is the **Transformer**, which eschews recurrence and instead relies on an **attention** mechanism to draw dependencies between input and output. The Transformer allows for more parallelization and can reach state of the art results [18].

2.3.1 Attention

An attention function is a function that maps a given query and a given set of key-value pairs to an output. In this function the query, keys, values, and output are all vectors, and the output is a weighted sum of the values, where the weight of each value if computed by a compatibility function of the query with the corresponding key [18]. Figure 4 illustrates the two types of

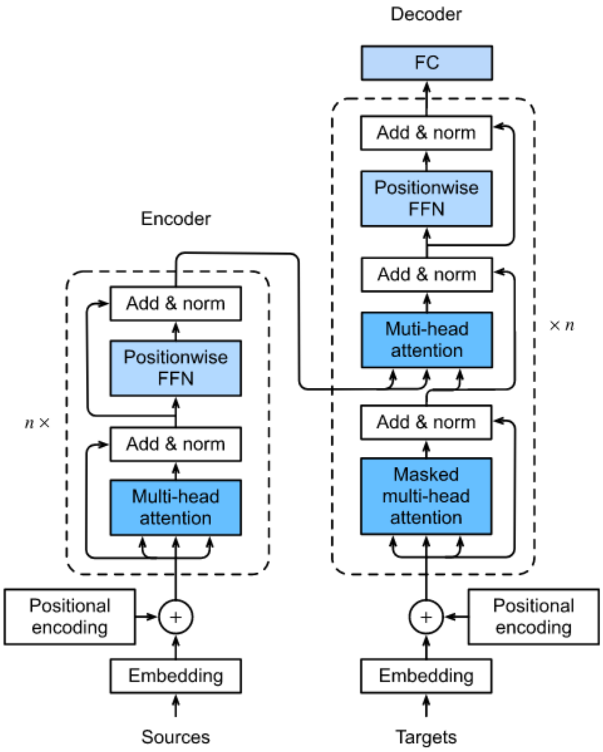

Figure 3: An illustration of the Transformer architecture [24].

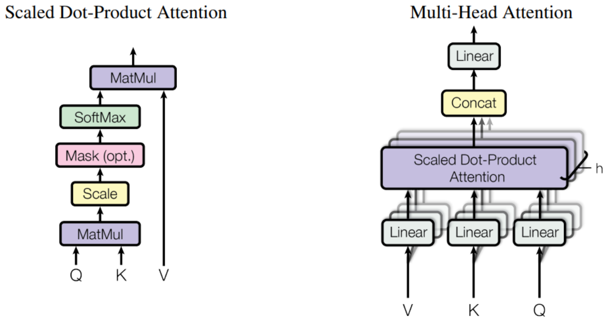

Figure 4: Illustration of (lef) the Scaled Dot-Product Attention, and (right) the Multi-Head Attention, which consists of several attention layers running in parallel [18].

attention functions used in the Transformer.

Scaled Dot-Product Attention: The scaled dot-product attention is the core attention function of the transform. The function takes queries and keys of dimensions d_k as input, and values of dimension d_v. In practice, the attention function is computed on a set of queries, keys and values simultaneously, by packing them into matrices $\boldsymbol{Q}, \boldsymbol{K}$, and $\boldsymbol{V}$, respectively. Thus, the scaled dot-product attention is computed as

$$\text{Attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \text{softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^T}{\sqrt{d_k}}\right)\boldsymbol{V} \tag{11}$$

where the scaling of $\boldsymbol{Q}\boldsymbol{K}^T$ is done to counteract the cases with a large d_k, which would result in the softmax-function having an extremely small gradient [18].

Multi-Head Attention: The **multi-head attention** is the used attention function by the Transformer and is an extention to the scaled dot-product attention. This attention function linearly projects the queries, keys and values h times with different, learned linear projections to d_k, d_k, and d_v dimensions, respectively. For each of the projected queries, keys and values, the scaled dot-product attention is applied, yielding d_v-dimensional output values. These output values are finally concatenated and further projected, resulting in the final values. By using multi-head attention, the model is allowed to jointly attend to information from different representation subspaces at different positions [18].

2.3.2 The Architecture

The Transformer follow an encoder-decoder structure, where the encoder maps an input sequence $\boldsymbol{x}$ to a sequence of continuous representations $\boldsymbol{z}$. The decoder then uses $\boldsymbol{z}$ to generate an output sequence $\boldsymbol{y}$, one element at a time. At each step the model consumes the previously generated output element as additional input when generating the next output [18]. Figure 3 illustrates the overall architecture of the Transformer.

Encoder: The encoder consists of N identical layers, where each layer consists of two sub-layers. The first sub-layer is a multi-head **self-attention** layer, and the second sub-layer is a position-wise fully-connected feedforward network. Around each sub-layer is a **residual connection**, which is followed by a round of **layer normalization**. In this self-attention layer the keys, values and queries come from the output of the previous layer in the encoder. Each position in the encoder can attend to all positions in the previous layer of the encoder [18].

Decoder: The decoder also consists of N identical layers. In addition to the two sub-layers in each encoder layer, the decoder also consists of a third sub-layer, which performs multi-head attention over the output of the encoder stack. Also here, is residual connections used around each of the sub-layers, followed by layer normalization. To ensure, that the predicitons for position i only depends on the known outputs at positions less than i, the self-attention sub-layer in the decoder stack is modified. This self-attention sub-layer, allows each position in the decoder to attend to all positions in the decoder up to and including that position. The self-attention sub-layer is modified by masking out all values in the input of the softmax which correspond to illegal connections [18].

Feedforward Networks: Each of the layers in the encoder and decoder contains a fully-connected feedforward layer, consisting of two linear transformations with the ReLU activation-function applied in between, as described below [18]

$$FFN(\boldsymbol{x}) = \max\left(0, \boldsymbol{x}\boldsymbol{W}_1 + \boldsymbol{b}_1\right)\boldsymbol{W}_2 + \boldsymbol{b}_2. \tag{12}$$

Positional Encoding: As the model does not contain any recurrences nor any convolutions, it has to carry some other type of information to know about the relative or absolute position of the elements in the input sequence. This is done by adding **positional encodings** to the input of the encoder and decoder stacks. The positional encoding is a vector of size d_model and is defined as the following

$$\boldsymbol{PE}_{pos,2i} = \sin\left(\frac{pos}{10000^{\frac{2i}{d_model}}}\right) \tag{13}$$

$$\boldsymbol{PE}_{pos,2i+1} = \cos\left(\frac{pos}{10000^{\frac{2i}{d_model}}}\right) \tag{14}$$

$$\tag{15}$$

where *pos* is the position and i is the dimension [18].

2.4 Training a Neural Network

2.4.1 Activation function

2.4.2 residual connection

2.4.3 layer normalization

3 Models

The following section covers the theory behind the various models that will be introduced in Section 5.

3.1 Mask R-CNN

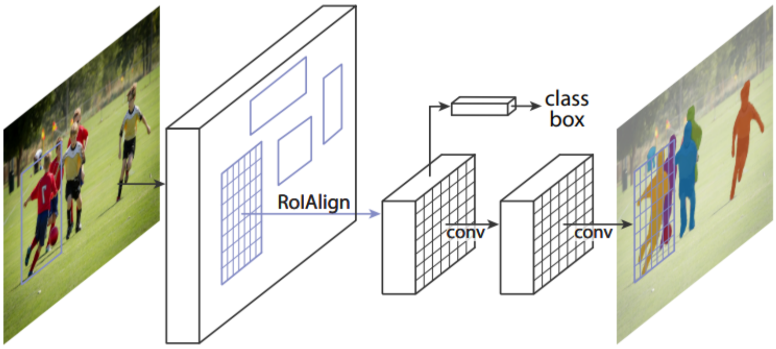

Figure 5: The Mask R-CNN framework for instance segmentation [6].

NOTE: MANGLER NOK AT SKRIVE NOGET MED, AT INPUT ER ET BILLEDE OG LIGNENDE NOGET OM HVAD OUTPUT ER.
When we will be performing the pose estimation in Section 5, our developed methods will be a variation of the *Mask R-CNN*, introduced by He *et al.* in 2018 [6]. The following subsection explains the architecture of the Mask R-CNN and is based on an interpretation of He *et al.* [6] and Zhang [26].

The Mask R-CNN can generally be split into various components, which we will explain in further details in the following subsections.

3.1.1 The Backbone

The first major component of the Mask R-CNN is the *Backbone*, which is a network used for extracting the features of the input image. Commonly, a pretrained variation of the *residual network* (*ResNet*) is used [7]. The backbone takes an image as input and returns a feature map.

3.1.2 Region Proposal Network (RPN)

The next major component of the Mask R-CNN is the *Region Proposal Network* (*RPN*). The RPN takes the feature map from the backbone as input, processes the feature map and proposes regions that may contain an object (the so-called *Region of Interests* or *RoI*) in the form of a feature map.

The RPN works by first processing the feature map with a convolutional layer that outputs a tensor with c channels, where each spacial vector (also with c channels) is associated with an anchor center. For each of these anchor centers a set of anchor boxes are generated. This convolutional layer is then followed by two 1×1 convolutional layers that independently processes this tensor. One of these 1×1 convolutional layers is a binary classifier that predicts whether each anchor box has an object. This is done by mapping each c-channel vector to a k-channel vector. The other 1×1 convolutional layer is an object bounding-box regressor, which predicts the offsets between the true object bounding-box and the anchor box. This is done by makking each c-channel vector to a $4k$-channel vector. For the overlapping bounding-boxes of the same object, we keep the one with the highest objectness score and discard the rest.

3.1.3 Region of Interest Alignment (RoIAlign)

The third major components of the Mask R-CNN is the *Region of Interest Alignment* (*RoIAlign*). This components takes the proposed RoIs from the previous components as input and finds where each RoI is in the feature map. This is done by extracting feature vectors from the output feature map from the RPN and transform them into a fix-sized tensor.

3.1.4 Object Detection Branch

3.1.5 Mask Generation Branch

3.2 UniPose-LSTM

3.3 DeciWatch

baseball_pitch	baseball_swing	bench_press
bowling	clean_and_jerk	golf_swing
jumping_jacks	jump_rope	pull_ups
push_ups	sit_ups	squats
strumming_guitar	tennis_forehand	tennis_serve

Table 1: The original 15 action-types in the Penn Action dataset.

4 Dataset

LAV EN LISTE OF HVILKE JOINTS ER ANNOTERET I DE FORSKELLIGE DATASÆT

4.1 The BRACE Dataset

The second dataset we will be using is the *BRACE* dataset [11]. We chose to use this dataset, as breakdancers tend be in acrobatic poses, similar to the ones that climbers tend to be in, making the poses relevant for our experiments in Section 5.

This dataset consists of $1,352$ video sequences and a total of $334,538$ frames with keypoints annotations of breakdancers. The frames of the video sequences are in RGB and have a resolution of 1920×1080 [11].

The frames of the video sequences have been annotated by initially using state-of-the-art human pose estimators to extract automatic poses. This was then followed by manually annotating bad keypoints, corresponding to difficult poses, as well as pose outliers. Finally, the automatic and manual annotations were merged, interpolating the keypoint seequence with Bézier curves. The keypoints is a list of 17-elements, following the COCO-format [11].

4.2 The Penn Action Dataset

One of the dataset we will be using is the *Penn Action* dataset [25]. This dataset consists of 2326 video sequences of 15 different action-types. Table 1 lists these 15 action-types [25].

Each sequence has been manually annotated with human joint annotation, consisting of 13 joints as well as a corresponding binary visibility-flag for each joint. The frames of each sequence are in RGB and has a resolution within the size of 640×480 [25].

Unlike the BRACE dataset, most of the poses in the Penn Action dataset are not very acrobatic and thus are not very relevant for the poses of climbers. For that reason, we have decided to focus on the action-types that may contain more acrobatic poses. Thus, we only keep the sequences that have `baseball_pitch`, `bench_press` or `sit_ups` as their corresponding action-type [25].

4.3 The ClimbAlong Dataset

5 Experiments

6 Discussion

7 Conclusion

8 References

[1] André Oskar Andersen. "2D Articulated Human Pose Estimation, Using Explainable Artifical Intelligence". Bachelor's Thesis. University of Copenhagen, Department of Computer Science, 2020.

[2] Bruno Artacho and Andreas Savakis. *UniPose: Unified Human Pose Estimation in Single Images and Videos*. 2020. DOI: 10.48550/ARXIV.2001.08095. URL: https://arxiv.org/abs/2001.08095.

[3] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2018. DOI: 10.48550/ARXIV.1812.08008. URL: https://arxiv.org/abs/1812.08008.

[4] Rohit Girdhar, Georgia Gkioxari, Lorenzo Torresani, Manohar Paluri, and Du Tran. *Detect-and-Track: Efficient Pose Estimation in Videos*. 2017. DOI: 10.48550/ARXIV.1712.09184. URL: https://arxiv.org/abs/1712.09184.

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

[6] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. *Mask R-CNN*. 2017. DOI: 10.48550/ARXIV.1703.06870. URL: https://arxiv.org/abs/1703.06870.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: 10.48550/ARXIV.1512.03385. URL: https://arxiv.org/abs/1512.03385.

[8] Sepp Hochreiter and Jürgen Schmidhuber. "Long Short-Term Memory". In: *Neural Computation* 9.8 (1997), pp. 1735–1780.

[9] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[10] Yue Luo, Jimmy Ren, Zhouxia Wang, Wenxiu Sun, Jinshan Pan, Jianbo Liu, Jiahao Pang, and Liang Lin. *LSTM Pose Machines*. 2017. DOI: 10.48550/ARXIV.1712.06316. URL: https://arxiv.org/abs/1712.06316.

[11] Davide Moltisanti, Jinyi Wu, Bo Dai, and Chen Change Loy. "BRACE: The Breakdancing Competition Dataset for Dance Motion Synthesis". In: *European Conference on Computer Vision (ECCV)* (2022).

[12] Alejandro Newell, Kaiyu Yang, and Jia Deng. *Stacked Hourglass Networks for Human Pose Estimation*. 2016. DOI: 10.48550/ARXIV.1603.06937. URL: https://arxiv.org/abs/1603.06937.

[13] Tomas Pfister, James Charles, and Andrew Zisserman. *Flowing ConvNets for Human Pose Estimation in Videos*. 2015. DOI: 10.48550/ARXIV.1506.02897. URL: https://arxiv.org/abs/1506.02897.

[14] Leonid Pishchulin, Mykhaylo Andriluka, Peter Gehler, and Bernt Schiele. "Poselet Conditioned Pictorial Structures". In: *2013 IEEE Conference on Computer Vision and Pattern Recognition*. 2013, pp. 588–595. DOI: 10.1109/CVPR.2013.82.

[15] Moacir Antonelli Ponti, Leonardo Sampaio Ferraz Ribeiro, Tiago Santana Nazare, Tu Bui, and John Collomosse. "Everything You Wanted to Know about Deep Learning for Computer Vision but Were Afraid to Ask". In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images Tutorials (SIBGRAPI-T)*. 2017, pp. 17–41. DOI: 10.1109/SIBGRAPI-T.2017.12.

[16] Xingjian Shi, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-kin Wong, and Wang-chun Woo. *Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting*. 2015. arXiv: 1506.04214 [cs.CV].

[17] Yuandong Tian, C. Lawrence Zitnick, and Srinivasa G. Narasimhan. "Exploring the Spatial Hierarchy of Mixture Models for Human Pose Estimation". In: *Computer Vision – ECCV 2012*. Ed. by Andrew Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 256–269. ISBN: 978-3-642-33715-4.

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. *Attention Is All You Need*. 2017. DOI: 10.48550/ARXIV.1706.03762. URL: https://arxiv.org/abs/1706.03762.

[19] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. *Convolutional Pose Machines*. 2016. DOI: 10.48550/ARXIV.1602.00134. URL: https://arxiv.org/abs/1602.00134.

[20] Yufei Xu, Jing Zhang, Qiming Zhang, and Dacheng Tao. *ViTPose: Simple Vision Transformer Baselines for Human Pose Estimation*. 2022. DOI: 10.48550/ARXIV.2204.12484. URL: https://arxiv.org/abs/2204.12484.

[21] Sen Yang, Zhibin Quan, Mu Nie, and Wankou Yang. *TransPose: Keypoint Localization via Transformer*. 2020. DOI: 10.48550/ARXIV.2012.14214. URL: https://arxiv.org/abs/2012.14214.

[22] Yi Yang and Deva Ramanan. "Articulated Human Detection with Flexible Mixtures of Parts". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35.12 (2013), pp. 2878–2890. DOI: 10.1109/TPAMI.2012.261.

[23] Ailing Zeng, Xuan Ju, Lei Yang, Ruiyuan Gao, Xizhou Zhu, Bo Dai, and Qiang Xu. *DeciWatch: A Simple Baseline for 10x Efficient 2D and 3D Pose Estimation*. 2022. DOI: 10.48550/ARXIV.2203.08713. URL: https://arxiv.org/abs/2203.08713.

[24] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. "Dive into Deep Learning". In: *arXiv preprint arXiv:2106.11342* (2021).

[25] Weiyu Zhang, Menglong Zhu, and Konstantinos G. Derpanis. "From Actemes to Action: A Strongly-Supervised Representation for Detailed Action Understanding". In: *2013 IEEE International Conference on Computer Vision*. 2013, pp. 2248–2255. DOI: 10.1109/ICCV.2013.280.

[26] Xiang Zhang. *Undertanding Mask R-CNN Basic Architecture*. 2021. URL: https://www.shuffleai.blog/blog/Understanding_Mask_R-CNN_Basic_Architecture.html.