1 Models

The following section covers the architectures of the various models that will be used in Section **??** when we will be performing our experiments.

1.1 Baseline

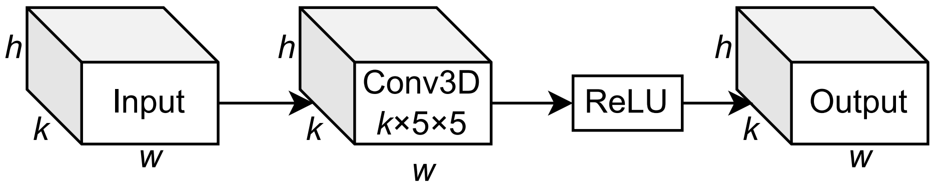

Figure 1: Illustration of the implemented baseline model.

The first model we will be using in Section **??** is a very simple baseline model based on a 3-dimensional convolution. Figure 1 illustrates the architecture of the model.

The model takes a sequence of $T \in \mathbb{R}$ estimated poses $\hat{\mathcal{P}} = \{\hat{\boldsymbol{P}}^t\}_{t=1}^T$ as input, where each estimated pose $\hat{\boldsymbol{P}}^t \in \mathbb{R}^{K \times h \times w}$ is represented using heatmaps, such that $K \in \mathbb{R}$ is the amount of keypoints in each estimated pose, $h \in \mathbb{R}$ is the height of each heatmap and $w \in \mathbb{R}$ is the width of each heatmap.

Once the data has been passed to the model, the processing of the data is very simple. As illustrated in Figure 1, the model starts by applying a 3-dimensional convolutional layer to the input data. The convolutional layer consists of K filters, each with a kernel-size of $T \times 5 \times 5$. To ensure the input and output of the convolutional layer has the same shape, we pad the input with zeros.

Once the convolutional layer has processed the data, the ReLU activation-function is applied element-wise to the data, resulting in the final prediction of the model.

1.2 Bidirectional Convolutional LSTM

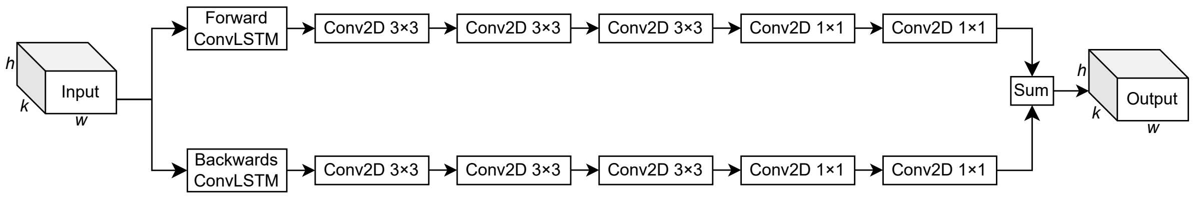

Figure 2: Illustration of the implemented bidirectional convolutional LSTM, where the two sequence orders are summed together.

Our second and third model are based on the LSTM-extension of Unipose-LSTM by Artacho and Savakis [1]. Artacho and Savakis' model was based on a unidirectional convolutional LSTM. However, as we do not require our model to work in real-time, and as we believe a bidirectional convolutional LSTM would be benifical, our models will be based on a bidirectional convolutional LSTM. The major difference between our second and third model is how they combine the information of the two sequence directions.

Both models take a sequence of $T \in \mathbb{R}$ estimated poses $\hat{\mathcal{P}} = \{\hat{\boldsymbol{P}}^t\}_{t=1}^T$ as input, where each estimated pose $\hat{\boldsymbol{P}}^t \in \mathbb{R}^{K \times h \times w}$ is represented as a set of heatmaps, where $K \in \mathbb{R}$ is the amount of keypoints, $h \in \mathbb{R}$ is the height of each heatmap and $w \in \mathbb{R}$ is the width of each heatmap.

1.2.1 Summing the Sequence Directions

Figure 2 illustrates the architecture of the second model. The model starts by branching into two seperate branches, that processes the estimated poses in opposite sequence order. Each branch processes the estimated poses one frame at a time. This is done by first applying a convolutional LSTM to the input frame at time step $t \in \mathbb{R}$, using the preceding output of the convolutional LSTM as the hidden state. Each convolutional LSTM is followed by five 2-dimensional convolutional layers, each applying 128 filters, except for the last convolutional layer of each branch, which aplies K filters. The first three convolutional layers use a kernel size of 3×3, whereas the following two convolutional layers use a kernel size of 1×1. The outputs of the two branches are then summed together element-wise.

All convolutional layers use a stride of one and zero-padding on the input, such that the output of each convolutional layer has the same dimensions as the input to the convolutional layer.

1.2.2 Concatenating the Sequence Directions

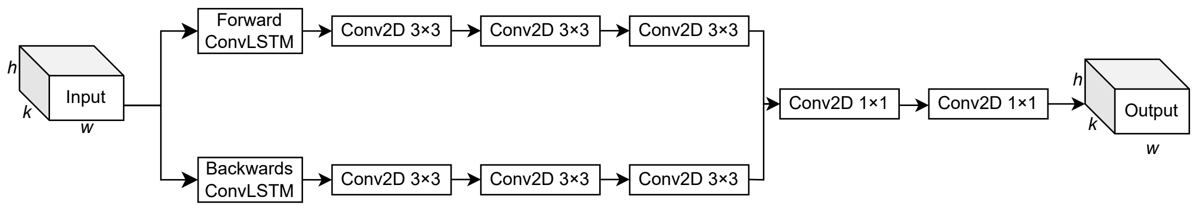

Figure 3: Illustration of the implemented bidirectional convolutional LSTM, where the two sequence orders are concatenated together and processed by two convolutional layers.

Looking at our second model, we see two problems with the architecture behind it: (1) the model does not have any opportunity of priorizing one processing order over the other, as the two branches are just summed together, and (2) by summing the two branches together it can occur, that some of the elements of the two branches are evened out, removing the prediction of the branches. Our third model aims at solving these two issues.

Figure 3 illustrates the architecture of our third model. This model is very similar to the previous model. It also starts of by branching into two seperate branches, that processes the estiamted poses in opposite sequence orders, where each branch processes the estimated poses one frame at a time. Similarly to the previous model, this model also starts of by applying a convolutional LSTM to the input frame, by using the preceding output of the convolutional LSTM as the hidden state. Further, each convolutional lstm is also fillowed by three 2-dimensional convolutional layers, each applying 128 filters with a kernel size of 3×3.

Different from the previous model, this model concatenates the output of each branch to form a $256 \times h \times w$ tensor, which is then processed by two 1×1 2-dimensional convolutions, with 128 and K filters, respectively.

Similarly to the second model, all convolutional layers also use a stride of one and zero-padding on the input, such that the output of each convolutional layer has the same dimensions as the input to the convolutional layer.

1.3 DeciWatch

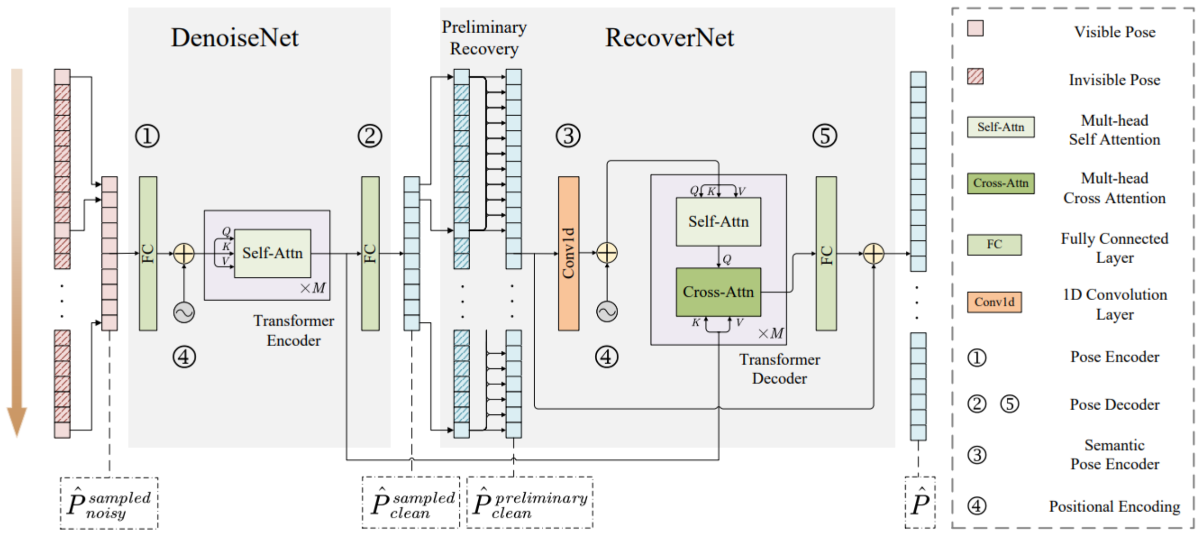

Figure 4: Illustration of the DeciWatch [2]

The last model we implement is a transformer-based model named *DeciWatch*, introduced by Zeng *Et al.*, which is illustrated in Figure 4. The model works by only processing some of the input-frames. It consists of two parts: the *DenoiseNet* and the *RecoverNet*. The aim of DenoiseNet is to denoise the estimated poses given as input to the model, whereas the aim of RecoverNet is to recover the poses of the missing frames. The following description of the model is based on an interpotation of the official paper behind DeciWatch [2].

More specifically, the model takes a sequences of $T \in \mathbb{R}$ estimated poses $\hat{\mathcal{P}} = \{\hat{\boldsymbol{P}}^t\}_{t=1}^T$ as input, where $\hat{\boldsymbol{P}}^t$ is represented by 2-dimensional keypoint position. Due to redundancy in consecutaive frames and continuity of human poses, the model starts by sampling every nth frame to select sparse poses $\hat{\boldsymbol{P}}^{sampled}_{noisy} \in \mathbb{R}^{\frac{T}{n}\times(2K)}$, where $K \in \mathcal{R}$ is the number of keypoints. These sampled poses are then passed to DenoiseNet.

The goal of DenoiseNet is to denoise the sparse poses, that were estimated by a single-frame pose estimator. The denoise process can be formulated as

$$\hat{\boldsymbol{F}}^{sampled}_{clean} = \textbf{TransformerEncoder}\left(\hat{\boldsymbol{P}}^{sampled}_{noisy}\boldsymbol{W}_{DE} + \boldsymbol{E}_{pos}\right). \tag{1}$$

That is, $\hat{\boldsymbol{P}}^{sampled}_{noisy}$ is first encoded through a linear projection matrix $\boldsymbol{W}_{DE} \in \mathbb{R}^{2K\times C}$ and summed with a positional embedding $\boldsymbol{E}_{pos} \in \mathbb{R}^{\frac{T}{n}\times C}$. This is then passed to a transformer-encoder consisting of $M \in \mathbb{R}$ multi-head Self-Attention blocks, resulting in the noisy poses being embedded into a clean feature $\hat{\boldsymbol{F}}^{sampled}_{clean} \in \mathbb{R}^{\frac{T}{n}\times C}$, where $C \in \mathbb{R}$ is the embedding dimensions. Lastly, another linear projection matrix $W_{DD} \in \mathbb{R}^{C\times 2K}$ is used to obtain the denoised sparse poses

$$\hat{\boldsymbol{P}}^{sampled}_{clean} = \hat{F}^{sampled}_{clean} W_{DD}. \tag{2}$$

After the sparse poses has been denoised as $\hat{\boldsymbol{P}}^{sampled}_{clean} \in \mathbb{R}^{\frac{T}{n}\times 2K}$, the data is passed to the RecoverNet, whose goal is to recover the absent poses. First, a linear transformation $\boldsymbol{W}_{PR} \in \mathbb{R}^{T\times\frac{T}{n}}$ is applied to perform preliminary sequence recovery to get $\hat{\boldsymbol{P}}^{preliminary}_{clean} \in \mathbb{R}^{T\times 2K}$ by

$$\hat{\boldsymbol{P}}^{preliminary}_{clean} = \boldsymbol{W}_{PR}\hat{\boldsymbol{P}}^{sampled}_{clean}. \tag{3}$$

To improve the recovery of the absent poses a transformer-decoder and positional embedding is used together with a 1D convolutional layer to bring temporal semantics into pose encoding to encode the neighboring $D \in \mathbb{R}$ frames' poses into pose tokens. Thus, RecoverNet, and the final prediction of DeciWatch, can be summarized by

$$\hat{\boldsymbol{P}} = \textbf{TransformerDecoder}\left(Conv1d\left(\hat{\boldsymbol{P}}_{clean}^{preliminary}\right) + \boldsymbol{E}_pos, \hat{\boldsymbol{F}}_{clean}^{sampled}\right)\boldsymbol{W}_{RD} + \hat{\boldsymbol{P}}_{clean}^{preliminary} \quad (4)$$

where $\boldsymbol{W}_{RD} \in \mathbb{R}^{C \times 2K}$ is another linear transformation layer. Further, as illustrated by Figure 4, key information is drawn in the the Cross-Attention block by leveraging the denoised features $\hat{\boldsymbol{F}}_{clean}^{sampled}$.

To avoid overfitting, dropout is applied to the input of each sub-layer and sums of the embeddings of the transformer-encoder and transformer-decoder, as well as to the positional encodings.