1 Pretraining

As we expect our models to require more data than what is in the ClimbAlong dataset to reach their optimal performace, we have decided to pretrain our models on the BRACE and Penn Action datasets, followed by finetuning them on the ClimbAlong dataset. By doing so we expect our models to yield better results than if we were only using the ClimbAlong dataset, as the pretraining data will be used for adjusting the randomly initialized weights, whereas the finetuning-data will just be used for specializing the model in performing well on the ClimbAlong dataset. The following section describes the pretraining stage, including the data preprocessing, the used configuration details, as well as the obtained results.

In the pretraining stage we will not be using the already developed pose-estimator, but instead only use our temporal-inclusive models by adding noise to the data such that it simulates the output of the already developed pose-estimator on the ClimbAlong dataset. We do this, as the images of pretraining data is very different from the ClimbAlong data, making us believe that the already developed pose-estiamtor will yield some very inaccurate results, as well as some predictions that will be very different from the predictions of the model on the ClimbAlong dataset.

Throughout section 1.1 we describe the preprocessing of the pretraining dataset. This is then followed by section 1.2, where we describe the used configuration that we use. Then, in section 1.3 we cover the obtain training and validation results, which is followed by section 1.4, where we cover the testing results. Finally, in section 1.5 we cover the technical details of our pretraining.

1.1 Data Preprocessing

As our models take a sequence of estimated poses as input, we will not be using the images of the frames, hence why we discard the images of all frames from BRACE and Penn Action, such that we only keep the annotated poses. We further preprocess these annotations, such that they simluate the output of the already developed pose-estimator as closely as possible.

We start by extracting the bounding-box of the annotated pose in each frame by using the annotated keypoints. As the placement of the invincible keypoints are very inconsistently placed, we have decided to just discard these keypoints, as we expect them to result in sub-optimal results. For the extracted bounding-boxes, we expand each side by 10%, such that no keypoint lies on any of the boundaries of the bounding-box. This is followed by discarding everything outside the bounding-box and rescale the bounding-box to have a sidelength of 56, such that it has the same dimensions as the output of the already developed pose-estimator.

Next, we transform each frame into twenty five heatmaps. This is done by creating twenty five 56×56 zero-matrices for each frame, such that each zero-matrix represents a single keypoint of a single frame. Further, for each keypoint we insert a fixed value $c = 255$ at the position of the keypoint in its corresponding zero-matrix and apply a Gaussian filter with mean $\mu_{out} = 0$ and standard deviation $\sigma_{out} = 1$ to smear out each heatmap. For missing keypoints, we do not place the value c in the corresponding heatmap, making the heatmap consist of only zeros. Further, as Penn Action is the only dataset with the position of the head annotated, as well as the only dataset missing a annotation for the nose, we treat the head-annotation of Penn Action as if it was a nose-annotation, as the position of the two annotations would be very close to each other.

The heatmaps that we produce by following the above description will be used as the groundtruth

data. However, as we will be pretraining our models detached from the already developed pose-estimator, we will also need some data as input. We acquire this data by adding some noise to the data, such that it becomes similar to the output of the already developed pose-estimator, essentially simulating the output of it. The noise is introduced by randomly shifting and smearing out each keypoint of each frame. For the shift-value we sample from a Gaussian distribution with mean $\mu = 0$ and standard deviation $\sigma = 3k$, where the **shifting-scalar** $k \in \mathbb{R}_{>0}$ is some fixed positive number based on what experiment is run. We clip the position of the shifted keypoints between 0 and 55, such that they cannot be outside of their corresponding heatmap. The smearing is done by using a Gaussian filter with mean $\mu_{in} = 1$ and some standard deviation $\sigma_{in} \in \mathbb{R}_{>0}$

1.2 Training Details

1.2.1 Data Configuration

For the data we use a window-size of $s = 5$ frames, as Artacho *et al.* found this to be the optimal number of frames to use [1], making our dataset consist of $345,120$ windows. Further, we randomly split our dataset into a training, validation and test set, consisting of 60%, 20% and 20% of the data, respectively, without any overlapping or repeating windows among eachother, as this results in minimal evaluation-bias. Lastly, to help the models learn the patterns of the data, we shuffle the three subsets, such that the order of which the data is given to the models does not influence the learning.

Instead of splitting each video sequence into windows of 5 frames, we could have instead just used the whole video sequence at once. However, we see a couple of problems with this. First off, this would require a lot of memory, as we have to store all of these frames at once. Secondly, for models like 3DConv which has a parameter based on the length of the input sequence, this would be a problem, as the video sequences have different lengths and the only solution would be to either pad the shorter video sequences or trim the longer video sequences, such that all video sequences have the same length.

As the datasets for the pretraining stage are missing some keypoints of the ClimbAlong dataset, we have to cancel out the training of these missing keypoints, as this would otherwise result in the models learning to never predict the presence of these keypoints. There are multiple ways to do this. We handled it during training by setting the groundtruth heatmaps of the missing keypoints equal to the corresponding predicted heatmaps, making the loss of these missing heatmaps be zero and thus the weights of the model of these heatmaps would not be adjusted.

1.2.2 Experiments

For each of the four models we run three different experiments. In experiment 1 we uniformly at random sample the standard deviation used by the Gaussian filter at the input data σ_{in} from the set $\{1, 1.5, 2, 2.5, 3\}$. We do this, as the output heatmaps of the already developed pose-estimator does not use a fixed standard deviation, making the data a better representation of the pose-estimator output.

As we find it interesting how big of a difference the randomness experiment 1 makes, we fix this standard deviation in experiment 2, such that we have $\sigma_{in} = 1$, thus, essentially, the models only have to learn to translate the input heatmaps.

Finally, with 30 frames per second and a window-size of $s = 5$, we suspect that the models might be given too little context to actually be able to effectively smooth out the input data.

We could fix this by increasing the window-size, however, we instead chose to make the models work at a lower frame rate, as the increased window-size would also increase the memory usage. Thus, for experiment 3 we still use a window-size of 5, however, with half the frame rate. For this experiment we also sample σ_{in} from the set $\{1, 1.5, 2, 2.5, 3\}$.

As we find it difficult to set the optimal value of the shifting-scalar k, such that the data becomes as close as possible to the finetuning data, we run each experiment twice for each model. In the first run we use a shifting-scalar of $s = 1$, whereas for the second run we increase the value of the shifting-scalar to $s = 2$, making the data a lot more noisy.

1.2.3 Configuration Choices

For optimizing the weights of the models, we use the ADAM optimizer with a batch size of 16, an initial learning rate of 10^{-3} and the MSE loss-function. During training we keep track of the lowest reached validation loss of an epoch. If this lowest validation loss has not been beaten for five consecutive epochs, we reduce the learning rate by a factor of 0.1, essentialy speeding up the training. Further, if the lowest validation loss has not been beaten for ten consecutive epochs, we terminate the training of the corresponding model, as it seems to be overfitting. In case this never happens, we terminate the training of the corresponding model once it has been trained for fifty epochs.

To help the models learn, we initialize the weights of all models by sampling from the Glorot uniform distribution, defined by

$$\mathcal{U}\left(-\frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}, \frac{\sqrt{6}}{\sqrt{n_j + n_{j+1}}}\right), \tag{1}$$

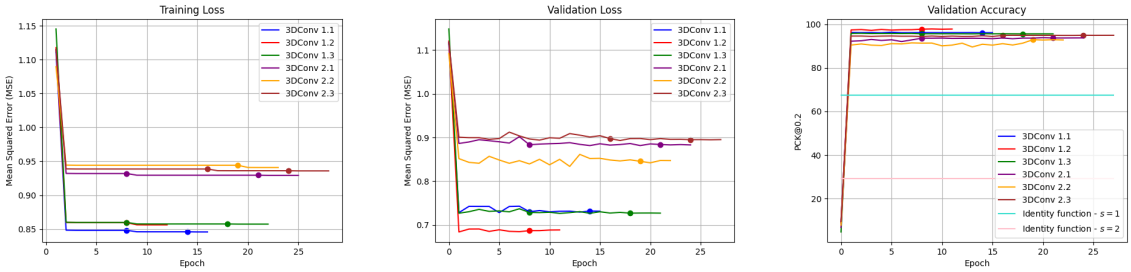
where n_j is the size of the previous layer and n_{j+1} is the size of the layer which is having its weights initialized, as this should decrease the likelihood of **vanishing** or **exploding gradients**, where the steps of gradient are either too short or too long [2].

Further, Zeng *Et al.* [3] have not described in their report their configuration choices for the various parameters of DeciWatch. However, by inspecting the official source code of the paper, we can actually find the configuration of the model, which we will be following [1]. Thus, our implementation of DeciWatch will be using $c = 128$ embedding dimensions, $M = 4$ multi-head Self-Attention blocks, a dropout-rate of 0.1 and five encoder and decoder layers. Zeng *Et al.* [3] actually only sample every 10th frame. However, as the datasets we are using are a lot more fast-paced than the one used by Zeng *Et al.* [3], we have decided to change this sample rate, such that it instead samples every 5th frame, making the model process more frames of the input video.

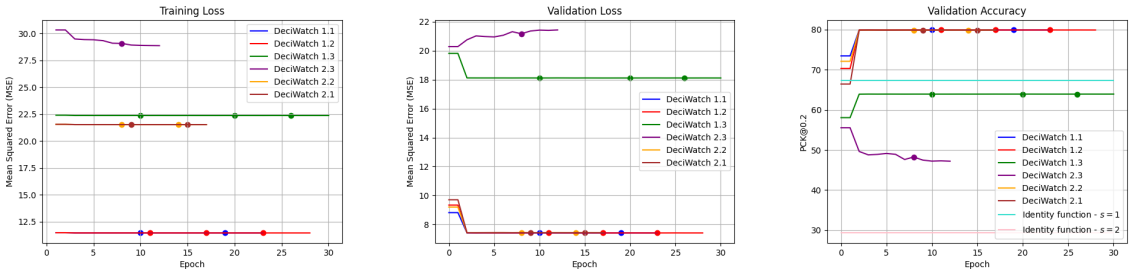
1.3 Training and Validation Results

We have in Figure 1 visualized the evolution of the training loss, validation loss and validation PCK@0.2 accuracy for our 24 models. Each model is delt two numbers seperated by a dot such that it follows a $x.y$-format (for instance, '2.3'), indicating what type of shifing-scalar and experiment it refers to. x indicates what shifting-scalar was used when shifting the input keypoints, and y indicates which of the three experiments from section 1.2 the run belongs to. Each plot has at least one dot, which indicates the reduction of the learning-rate due to five consecutive epochs without beating the minimum validation loss.

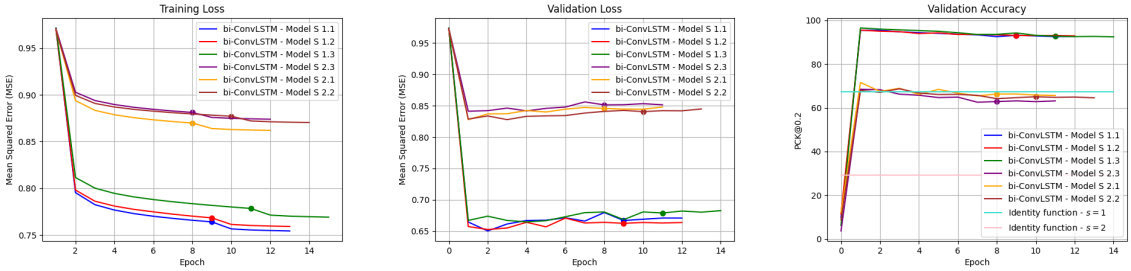
[1] https://github.com/cure-lab/DeciWatch/blob/main/configs/config_jhmdb_simplepose_2D.yaml



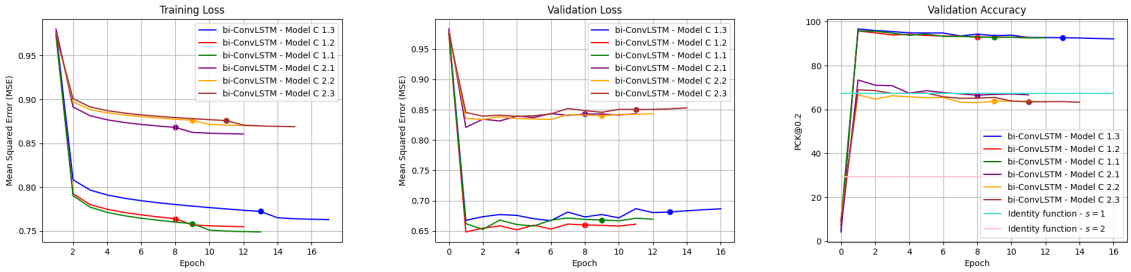
(a) Pretraining results of 3DConv.



(b) Pretraining results of DeciWatch.



(c) Pretraining results of the bi-ConvLSTM Model S.



(d) Pretraining results of the bi-ConvLSTM Model C.

Figure 1: Evolution of the training loss, validation loss and validation PCK@0.2 accuracy of the 24 models during training, as well as the validation PCK@0.2 accuracy of the identity function of the two datasets. The dots indicates a reduction of learning-rate. First row: 3DConv. Second row: DeciWatch. Third row: bi-ConvLSTM Model S. Fourth row: bi-ConvLSTM Model C.

By comparing the validation accuracies of the various models against the identity function we see, that all of the models, except for DeciWatch 1.3, do learn to somewhat denoise the input data. The simple 3DConv seems to be the architecure that generally delivers the greatest results, whereas DeciWatch is generally the architecure that delivers the worst results. Further, the two architectures that are based on bidirectional convolutional LSTMs have close to no difference in their performance, indicating that our raised concern about one of the architectures not being able to prioritize one LSTM-branch over the other is not as great as we hypothesized it to be.

From the figure we can see, that generally the reduction of the learning rate has an effect on the training loss, as generally the training loss is immediately decreased after a reduction of the learning rate. However, the same cannot be said about the validation loss and accuracy, as these do not immediately decrease after the learning rate has been reduced. Often, the training even terminates before a second reduction of the learning rate takes place, indicating the missing effect of the learning rate reduction on the validation loss.

Further, we can clearly see the effects of the shifting-scalar on the training of the models, as all of the models performs better on the dataset with a low shifting-scalar, than on their corresponding counterpart with a higher shifting-sclar. DeciWatch 1.3 and DeciWatch 2.3 do however seems to be struggling quite a bit, as the validation accuracies of these models are much lower than the other DeciWatch models with the same noise-scalar.

1.4 Test Results

Accuracy metric	PCK@0.05			PCK@0.1			PCK@0.2		
Mean threshold distance (px)*	1.11			2.23			4.46		
Experiment	1.1	1.2	1.3	1.1	1.2	1.3	1.1	1.2	1.3
Identity function	6.95	6.95	6.95	25.7	25.7	25.7	67.4	67.4	67.4
3DConv	1.84	20.5	0.67	30.6	58.5	23.8	**96.6**	**98.0**	96.3
DeciWatch	**51.4**	**51.4**	**44.5**	64.6	64.6	51.5	80.2	80.2	64.2
bi-ConvLSTM Model S	27.8	31.1	33.8	68.4	71.0	**72.5**	95.7	95.8	96.7
bi-ConvLSTM Model C	29.5	31.7	31.8	**69.5**	**71.3**	71.8	96.1	96.1	**96.9**

Table 1: Testing accuracies of the various developed models for shifting-scalar $s = 1$. All the accuracies are in percentage. *: The mean maximum distance between the predicted keypoint and corresponding groundtruth keypoint for the prediction to count as being correct, measured in the heatmap coordinate system.

We have in Table 1 and Table 2 illustrated the results of testing the epoch of each model that yielded the best validation PCK@0.2 accuracy.

For shifting-scalar $s = 1$ we see, that DeciWatch is the architecure that yields the best results for PCK@0.05, however, for PCK@0.2 DeciWatch actually tends to be the worst performing model. Further, as stated in section 1.3 DeciWatch 1.3 does perform worse than the identity function when considering PCK@0.2, however, for both PCK@0.05 and PCK@0.1 DeciWatch 1.3 completely outperforms the identity function, hence why it actually does learn to denoise the data, unlike what we stated in section 1.3 where we just looked at the validation PCK@0.2 accuracy. Generally, most of the predicitons of DeciWatch that are considered correct, are actually very close to the groundtruth, however the predictions that are incorrect are pretty far away from their corresponding groundtruth location.

Accuracy metric	PCK@0.05			PCK@0.1			PCK@0.2		
Mean threshold distance (px)*	1.11			2.23			4.46		
Experiment	2.1	2.2	2.3	2.1	2.2	2.3	2.1	2.2	2.3
Identity function	1.84	1.84	1.84	7.75	7.75	7.75	29.3	29.3	29.3
3DConv	0.80	2.40	0.11	21.6	27.6	16.8	**94.2**	**91.8**	**95.5**
DeciWatch	**51.2**	**51.2**	**10.3**	**64.4**	**64.4**	24.4	80.2	80.2	50.3
bi-ConvLSTM Model S	10.5	11.6	10.1	31.1	33.4	29.5	68.5	67.8	69.6
bi-ConvLSTM Model C	12.5	12.0	9.63	36.4	32.5	**29.8**	74.6	65.5	70.0

Table 2: Testing accuracies of the various developed models for shifting-scalar $s = 2$. All the accuracies are in percentage. *: The mean maximum distance between the predicted keypoint and corresponding groundtruth keypoint for the prediction to count as being correct, measured in the heatmap coordinate system.

On the other hand, the simple 3DConv seems to be the architecture that generally performs the worst for PCK@0.05 and PCK@0.1, and actually the architecture that performs the best for PCK@0.2, except for run 1.3, where the two bidirectional convolutional LSTM based models actually outperforms it. Thus, 3DConv is very good at inferencing a rough estimate of the position of the keypoints, however, the exact position of these predictions are generally incorrect.

Generally, the bidirectional convolutional LSTMs yield some decent results, as the predicitons of these models are always pretty far away from the worst-performing model for any of the three used accuracy metrics, and for PCK@0.1 they even seem to yield the best results of the four architectures. Similarly to the validation results, the test results do only indicate a minor performance differences between the two different types of bidirectional convolutional LSTMs in favor of the bi-ConvLSTM Model C. Likewise, similarly to the validation results, the test results indicate a major effect of a low frame rate for DeciWatch.

If we compare the performances of the models on experiment 2 against their performances on experiment 1 we see, that all arhcitectures, except for DeciWatch, performs better on experiment 2 than they do on experiment 1. This is especially true for 3DConv, whose run on experiment 2 completely outperforms its run on experiment 1. On the other hand, if we compare the results of experiment 3 against the results of experiment 1 we see, that both 3DConv and DeciWatch perform worse on experiment 3 than they do on experiment 1. However, the two archtectures that are based on a bidirectional convolution LSTM actually continuously performs better in experiment 3 than they do in experiment 1, making us believe that the added context when decreasing the frame rate, actually help these models.

Generally, the testing results of shifting-scalar $s = 2$ follow the same pattern with some minor differences. If we consider the results of PCK@0.2, 3DConv is now the architecture that always yields the best results - and by a large margin, but it is on the other hand completely outperformed by the other models when considering PCK@0.05. This indicates, that the architecture can still yield decent rough estimations of the keypoints when a lot of noise is added, however, it does also indicate, that the exact position of these keypoints are very much effected by the level of noise. Further, its results in experiment 2 do not any longer outperform its results in experiment 1 when considering PCK@0.2, as it was the case for shifting-scalar $s = 1$.
We have further in Table 3 and Table 4 illustrated the keypoint-specific testing PCK@0.2 of these models.

	3DConv			DeciWatch			bi-ConvLSTM sum.			bi-ConvLSTM concat.			Total
Experiment	1.1	1.2	1.3	1.1	1.2	1.3	1.1	1.2	1.3	1.1	1.2	1.3	
Nose	96.2	97.3	96.3	83.0	83.0	68.0	97.3	94.2	97.8	96.2	96.4	95.9	91.8
Ear	96.3	97.0	96.3	85.1	85.1	70.0	96.8	96.3	96.9	96.3	96.4	96.1	92.4
Shoulder	96.7	98.6	96.7	83.8	83.8	68.6	97.0	96.9	96.0	96.5	97.0	97.2	92.4
Elbow	96.7	98.7	96.4	78.3	78.3	61.8	96.1	97.6	97.6	96.1	96.5	98.5	91.1
Wrist	96.5	97.8	96.2	74.0	74.0	57.7	95.8	95.0	96.2	95.9	95.5	97.0	89.3
Hip	96.8	99.1	96.3	80.3	80.3	64.0	93.5	95.7	96.6	96.8	95.3	96.3	90.1
Knee	96.8	98.5	96.4	75.9	75.9	60.0	95.2	96.0	96.7	95.4	96.4	97.6	90.1
Ankle	96.2	96.9	95.8	72.8	72.8	57.9	95.1	94.1	96.7	95.7	95.6	96.4	88.8
Total	96.6	98.0	96.3	80.2	80.2	64.2	95.7	95.8	96.7	96.1	96.1	96.9	

Table 3: Keypoint-specific testing PCK@0.2-accuracies of the various models for shiting-scalar $s = 1$. All the accuracies are in percentage.

	3DConv			DeciWatch			bi-ConvLSTM sum.			bi-ConvLSTM concat.			Total
Experiment	2.1	2.2	2.3	2.1	2.2	2.3	2.1	2.2	2.3	2.1	2.2	2.3	
Nose	92.5	84.5	94.4	82.9	82.9	53.3	75.8	70.6	73.5	73.9	72.2	73.7	77.5
Ear	92.0	91.1	94.4	85.0	85.0	51.0	80.3	77.7	77.7	80.8	79.2	79.9	81.2
Shoulder	93.7	91.9	95.6	83.8	83.8	40.8	74.7	68.2	68.9	70.4	68.1	68.3	75.7
Elbow	94.8	92.3	95.8	78.3	78.3	39.9	58.2	66.7	64.7	67.0	61.6	63.3	71.7
Wrist	94.8	92.6	96.0	73.9	74.0	56.0	69.4	66.3	69.4	75.6	65.4	69.4	75.2
Hip	94.5	91.6	95.7	80.3	80.3	46.0	64.7	61.9	67.5	80.7	57.3	74.8	74.6
Knee	95.0	92.7	95.7	75.9	75.9	53.3	60.1	65.9	64.7	68.6	57.0	54.5	71.6
Ankle	95.2	94.1	95.6	72.7	72.7	57.7	69.4	67.3	72.5	78.1	67.5	77.5	76.7
Total	94.2	91.8	95.5	80.2	80.2	50.3	68.5	67.8	69.6	74.6	65.5	70.0	

Table 4: Keypoint-specific testing PCK@0.2-accuracies of the various models for shiting-scalar $s = 2$. All the accuracies are in percentage.

Looking at Table 3 we see, that the models with shifting-scalar $s = 1$ generally perform the best on the ears and shoulders, whereas they perform the worst the wrists and ankles. The models in Table 4 tells another story, as instead of performing the worst on the ankles and wrists, the elbows and knees are now the most difficult keypoints, and the ears, nose and ankles are now the least difficult keypoints.

1.5 Technical Details

All models were trained and evaluated on a shared GPU cluster using a 24GB NVIDIA Titan RTX and an Intel(R) Xeon(R) Gold 6248 CPU @ 2.50GHz. All models were implemented in Python version 3.9.9 using PyTorch 2.0.0. 3DConv took about 60 minutes per epoch, DeciWatch about 86 minutes per epoch, the bi-ConvLSTM Model S about 75 minutes per epoch, and the bi-ConvLSTM Model C about 88 minutes per epoch.