1 Discussion

The following section discusses our obtained results from the pretraining and finetuning stages, some concerns regarding possible suboptimal choices we have made, as well as suggests some possible future work. The section begins in section 1.1, where we discuss the obtain results. Then, in section 1.2, we discuss why the models performed much better during the finetuning stage, than during the pretraining-stage. Section 1.3 then discusses which model is the optimal choice. In section 1.4 we then discuss some pitfalls of our approach and how we could have made some better choices. Lastly, in section 1.5 we present some possible future work possibilities.

1.1 Results

In section **??** and section **??** we successfully implemented and tested the four architectures in various experiments. We will in the following be analysing and discussing the potential reasons for these obtained results.

1.1.1 Pretraining

We saw in section **??** how 3DConv tend to perform much better in experiment 2, than it does in experiment 1 (with the exception of run 2.1 and 2.2). As the goal of experiment 1 is for the models to both learn translation and scaling, whereas the goal of experiment 2 is just for the models to learn translation, we can clearly see here how big of a difference the task of learning to also scale the data has on the results for 3DConv. Of course, some randomness does also play in here, as the models for the two experiments were initialized with different weights. However, as the weights were sampled from the same distribution, we find it hard to believe that this randomness has such a big impact on the results. Thus, we claim that the performance differences between the two experiments is due to the task of scaling the heatmaps. For the two architectures that are based on a bidirectional convolutional LSTM, a similar pattern is observed for the shifting scalar, however, this performance difference is much smaller than it was in the case for 3DConv. We simply believe, that this improved performance is due to the bidirectional convolutional LSTMs predicting the position of the keypoints in a more sophisticated manner, making them better than 3DConv at scaling the heatmaps. For DeciWatch there are no performance differences between experiment 1 and experiment 2. However, this is expected, as unlike the other architectures, DeciWatch works directly on keypoint-coordinates instead of the heatmaps, making it scaling invariant and thus making the two experiments equivalent to each other.

We further saw in section **??** the effects on the models of halfing the frame rate. For both the 3-dimensional convolutional LSTM and DeciWatch this had a negative effect, whereas it actually had a positive effect for the two models that are based on a bidirectional LSTM. For DeciWatch we have to keep in mind, that it is already only considering every fifth frame, so by halfing the frame rate it is actually only considering every tenth frame, removing a lot of the context. For 3DConv and the two models based on a bidirectional convolutional LSTM, we again suggest that the performance difference is due to the later models yielding their predictions in a more sophisticated manner than the simple 3DConv. In the case of when the shifting-scalar is $s = 2$, all models generally perform worse in experiment 3 than they do in experiment 1, which we simply believe is due to the data being too noisy, making the finer/local context more important. As the distance-threshold for PCK is increased however, 3DConv actually perform better in experiment 3 than it does in experiment 1, contradicting this belief. However, we have to note here, that the performance difference is rather small and that it is in the case where the the distance-threshold is at its greatest. Thus, we believe this performance difference be due to the

random initialization of the models or the models learning a rough estimate of the keypoints, instead of learning a fine estimation, as this is more difficult when more noise is added to the data. Further, 3DConv is the lowest performing model when considering PCK@0.05 but the highest performing model when considering PCK@0.2, meaning that it generally does seem to learn a rough estimate of the position of the keypoints, strengthening our last belief.

We finally saw in **??** how the models for shifting-scalar $s = 1$ had the most difficulty with the wrists and ankles, whereas they performed the best on the ears and shoulders. However, we find this very expected, as the wrists and ankles tend to be the joints that have the highest amount of movements, especially in sports which our pretraining dataset is centered around, and thus are much more difficult to denoise than more static joints such as the ears and shoulders. On the other hand, in the case of using the shifting-scalar $s = 2$, we saw how the models struggled the most with the elbows and knees, and performed the best on the ears, nose and the ankles. At first glance, one would find this rather unexpecting as the wrists and ankles are no longer the most difficult keypoints. However, we suggest that this could be caused by these keypoints being too close to the heatmap borders, such that when they are shifted a lot towards any of these borders, they will end up at the border, instead of exceeding it. Thus, the wrists and ankles are not the most shifted keypoints in the cases of the shifting-scalar $s = 2$. Instead, the next most moving keypoints, the knees and elbows, are the most difficult keypoints, as these keypoints are both shifted a lot and the corresponding joints contain a decent amount of movement.

1.1.2 Finetuning

In section **??** we saw how the models from the experiments with shifting-scalar $s = 1$ tend to yield better results than the models trained on data with shifting-scalar $s = 2$. We further saw, how the models from experiment 2 generally performs better than the models from experiment 1. Generally however, these performance differences are very minor, which makes sense as the finetuning-data is the same for the two groups. We do note the minor performance differences, which could mean, that (1) the noise in the finetuning data is more similar to the data with shifting-scalar $s = 1$ than the data with shifting-scalar $s = 2$, and (2) the standard deviation of the peaks of the finetuning data is not changing as much throughout the dataset, as we made them do in experiment 2 during the pretraining-stage.

One could look at Figure **??** and claim that the pretraining-stage has had no effect, as the validation accuracy starts off very low. However, we disagree for two reasons. First off, we just noted that the choice of shifting-scalar from the pretraining-stage had an effect on the final results, hence why the pretraining-stage must have had an effect. Secondly, most of the models actually reach a very high validation accuracy after just a couple of epochs, which we believe is due to the pretraining. One could argue, that a similar pattern was observed during the pretraining-stage and is thus just a general pattern. However, we have to remember, that the pretraining-dataset is much bigger than the finetuning-dataset and thus just after one epoch in the pretraining-stage, the models have already been trained many more samples, which explains this major performance-jump during the pretraining-stage.

Lastly, one saw in section **??** how the models tend to perform the worst on keypoints related to the thumbs, pinkies and index fingers. We see two reasons for this observation. First off, these keypoints were not included in the pretraining dataset, hence why the models have learned to infer the position of these keypoints purely from the finetuning dataset. We believe this reason to only have a minor effect, as if we look at the performance of the models on the keypoints related to the heels and the toes, which were neither included in the pretraining dataset, we

see, that the performance differences between these keypoints and the remaining keypoins is only minor, hence why the missing of finetuning-keypoints in the pretraining-dataset is not the major explanations of the bad performance of the finger keypoints. The second reason is, that the finger keypoints just have a lot of movement, as these joints are the most used joint in bouldering and are thus the joints that moves the most, hence why they are difficult to predict the position off.

1.2 Why did the Models Perform Better during Finetuning than during Pretraining?

All of the finetuned models outperform performs better on the finetuning test dataset, than their pretraining counterpart on the pretraining test dataset. We see a couple of reasons for this.

First off, the models have been trained om more data, as they have been trained on both the pretraining data and the finetuning data. Secondly, the various videos of the ClimbAlong dataset are very semantically similar. This does not hold for the pretraining dataset, as this dataset consists of people breakdancing, throwing a baseball, bench pressing and performing sit ups, which are all very semantically different from each other, which could make it hard for the models to learn to generalize.

Further, the BRACE dataset was not fully manually annotated, but instead annotated using a pose estimator and only certain incorrectly predicted poses were manually annotated, which were detected by a machine learning model. However, if either the machine learning model for detecting incorrectly predicted poses or the pose estimator delivered suboptimal results, then the annotations would also be suboptimal and thus contain some noise, which is very difficult for our models to replicate. The finetuning dataset on the other hand was completely fully annotated by humans, making the annotations much less noisy and thus easier for the models to replicate.

For the BRACE dataset, it is clearly documented, that the video sequences are filmed using 30 frames per second. For Penn Action, on the other hand, a similar information is not stated anywhere and the video sequences come as individual frames without the duration of the video sequences noted, hence why we cannot neither compute the frame rate of these video sequences. This could cause some problems, as the windows from the two datasets, could span two different durations and thus capture two different amount of context. On the other hand, all of the video sequences of the finetuning dataset are all filmed using 30 frames per second, making the dataset much more consistent, which makes the fitting easier.

Lastly, one major reason behind the performance differences of the models on the pretraining dataset and the finetuning dataset is the performance of the identity function. If we compare the performance of the identity function on the pretraining dataset in Table **??** and Table **??** against the performance of the identity function on the finetuning dataset in Table **??** and Table **??**, we clearly see, that the identity function performs much better on the finetuning dataset than on the pretraining dataset. Hence why our models have to perform way less denoising on the finetuning dataset than on the pretraining dataset, making the task on the finetuning dataset a lot easier than on the pretraining dataset.

1.3 Which Model is the Best for Temporal Smoothing?

As we have now fully developed and tested all of our models on all three experiments, we can decide which models yield the best results for temporal smoothing in bouldering.

	1.1	1.2	1.3	2.1	2.2	2.3	Total
3DConv	81.5	82.4	82.7	80.4	82.1	**80.8**	81.7
DeciWatch	**90.1**	**90.1**	**83.8**	**89.8**	**89.8**	68.2	**85.3**
bi-Conv LSTM Model S	76.3	75.6	76.8	77.0	76.0	75.5	76.2
bi-Conv LSTM Model C	76.3	77.4	77.3	77.1	77.2	76.4	77.0

Table 1: Mean of testing PCK@0.05, PCK@0.1, and PCK@0.2 of each finetuning run.

	Mean prediction time (ms)	Standard deviation of prediction time (ms)	Number of parameters
3DConv	0.39	9.29×10^{-2}	78,150
DeciWatch	14.2	0.11	1,708,388
bi-Conv LSTM Model S	10.6	1.68×10^{-2}	1,875,666
bi-Conv LSTM Model C	20.1	6.93×10^{-2}	1,872313

Table 2: Mean prediction time of each architecture on a single window from the whole finetune dataset, as well as the total number of parameters of each architecture. The time measurements are based on five runs and were measure on the machine described in section **??**.

We have in Figure 1 noted the mean of testing PCK@0.05, PCK@0.1, and PCK@0.2 of each finetuning run. Based on these numbers it seems like DeciWatch 1.1/1.2 would be the models to pick, as these yield the greatest average testing accuracy. However, as we know from section **??**, DeciWatch 1.1/1.2 delivers some of the least optimal results when considering PCK@0.2. On the other hand, if we instead consider PCK@0.05, then DeciWatch 1.1/1.2 are the models that deliver the greatest results. Thus, if one is looking for the run that on average delivers the greatest results, then DeciWatch 1.1 or 1.2 would be the ideal choice.

On the other hand, if the requirement is just to get a rough estimation of the position of the keypoints, then bi-ConvLSTM Model C 1.1 would be the ideal choice, that this run delivers the highest PCK@0.2 testing accuracy, at the cost of one of the lowest PCK@0.05 testing accuracies.

One could also consider the prediction time and the size of each model. We have in Table 2 noted the mean prediction time of each model on a single window, as well as the size of each model. From the table we can clearly see, that 3DConv is by far the fastest and smallest architecture. This, combined with the general decent results of 3DConv in Table 1, could mean, that 3DConv would be the optimal choice if one is concerned about the prediction time or size of the model.

1.4 General Reflections

Generally, we believe that we made the correct choices throughout the execution of the project. However, looking back throughout the project, we do find some bad decision made by us, that could have been avoided or made in a better way.

1.4.1 Pretraining

For the preprocessing of the pretraining dataset, the goal was to preprocess the data, such that it simulated the output of the keypoint detector. This was done through multiple steps, where we used multiple values. For instance, we used a standard deviation in the range $\{1, 1.5, 2, 2.5, 3\}$ for the Gaussian filter on the input data and we shifted the input data by

sampling from a normal distribution with mean $\mu_{in} = 1$ and standard deviation 3 or 6, depending on the experiment. These values were all some values that we came up with. We could instead have picked these values in a more sophisticated manner such that the data would be more similar to the ClimbAlong data. This could for instance have been done by approximating them from the ClimbAlong dataset. We could further have made the shifting of the keypoints, keypoint-dependent, such that it would have modelled the level of movement of each keypoint. We are however, not sure how beneficial this would have been, as (1) we generally already receive some great results which are difficult to beat, and (2) for the experiments where we did experiment with increasing the noise, we did not see any improvement on keypoints with a high amount of movement.

For the last part of the preprocessing of the pretraining dataset, we split the data into a training, validation and test set, consisting of 60%, 20% and 20% of the data respectively. This was done by making sure that none of the windows were repeated and that none of the windows were overlapping between the three datasets. By doing so we ensured that the evaluation carried minimal bias, as the models had not seen any of the frames in the validation and test set during its training. However, different frames of the same video sequence could appear across the three sets, which could potentially carry some bias, as the same person appear across multiple sets. Instead, it would most likely have been better if we made sure that the sets had no overlapping video sequences, as this would lower the likelihood of introducing any evaluation bias.

1.4.2 Finetuning

When creating the target heatmaps for the finetuning data we occasionally stumbled upon the ground truth keypoints being suppoed to be placed outside of the predicted bounding-box, which we used as the target bounding-box. As the goal of our models is not to correct the predicted bounding-box, but instead just to correct the position of the predicted keypoints, we had to make a choice of what to do for the keypoints that were supposed to be placed outside of the bounding-box. We generally saw two solutions, as we could either (1) completely remove these keypoints and just leave the heatmap empty, or we could (2) move these keypoints such that they were somewhere inside the bounding-box. As we expected the first solution to have a negative impact on the results, as the models would learn that these keypoint were missing, we simply went with the second solution. However, one could argue that the first solution would be the optimal choice, as the models, with the second solution, are essentially learning the incorrect position of the keypoints.

1.5 Future Work

If we were to work further with this project, we first find it interesting to experiment with other machine learning methods. We would for instance find it very interesting to test the effects of letting DeciWatch process all frames, instead of only processing every fifth frame. Further, we find it interesting whether or not it would improve the results if DeciWatch was adapted, such that it made use of Vision Transformers as introduced by Dosovitskiy *Et al*. [1].

Further, we see some potential work in trying to avoid the overfitting during the finetuning-stage that we experienced in section **??** and find it interesting how much this would improve the final results. We did attempt some of this in section **??**, however, without any luck. One could for instance further try to avoid the overfitting by (1) increasing the variance of the dataset by incorporating even more data augmentation, for instance by adding some noise to the data, or (2) decrease the complexity of the models, such that they have less tuneable parameters

Lastly, we suggest that the models could be retrained multiple times. In section 1.1 we blaimed some of the results on the random initialization of the weights of the models. By retraining the models multiple times, the likelihood of this randomness being the reasoning for these results decreases and thus we will be sure whether or not our results are due to the randomness of the weights.