1 Experiments

The following section describes the training of our models. This includes the the various experiments we perform, the data preprocessing of the data, the configuration details we use, as well as the obtained results.

1.1 Approach

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 performancing well on the ClimbAlong dataset.

1.2 Pretraining

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.

1.2.1 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 start by extracting the bounding-box of the annotated pose in each frame by using the annotated keypoints. Further, we expand each side by 10%, such that no keypoint lies on any of the boundaries of the bounding-box. To ensure that the aspect ratio of the pose is kept later on, we transform the bounding-box into a square by extending the shorter sides, such that they have the same length as the longer sides. Next, we discard everything outside the bounding-box and rescale the bounding-box to have a sidelength of 56, such that it has the same size 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 $c \in \mathbb{R}$ 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 annotation 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 they become similar to the output of the already developed pose-estimator, essentially simulating the output of the already developed pose-estimator. The noise is introduced by randomly shifting each keypoint of each sample and by smearing our each keypoint of each sample by using a Gaussian filter with mean $\mu_{in} = 1$ and some standard deviation $\sigma_{in} \in \mathbb{R}_{>0}$. For the shift-value we use $x \cdot k$, where the **shifting-scalar** $k \in \mathbb{R}_{>0}$ is some fixed positive number and $x \in \mathbb{R}$, is equal to 20% of the mean torso-diameter. We clip the position of the shifted keypoints between 0 and 55, such that they cannot be outside of their corresponding heatmaps.

1.2.2 Training Details

Data Configuration For the data we use a window-size of $k = 5$ 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. We insert the fixed value $c = 255$ in each heatmap at the position of the corresponding keypoint. 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.

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.

Setups For each of the four models we use three different setups. In the first setup 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 of the first setup makes, we fix this standard deviation in our second setup, 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 $k = 5$, 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 our last setup we still use a window-size of 5, however, with half the frame rate. For this setup 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 setup twice for each model. In the first run we use a shifting-scalar of $k = 1$, whereas for the second run we increase this value to $k = 2$, making the data a lot more noisy.

Training Configuration 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. Further, if the lowest validation loss has not been beaten for ten consecutive epochs, we terminate the training of the corresponding model. In case this never happens, we terminate the training of the corresponding model once it has been trained for fifty epochs.

To reduce the likelihood of vanishing or exploding gradients, 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 [2].

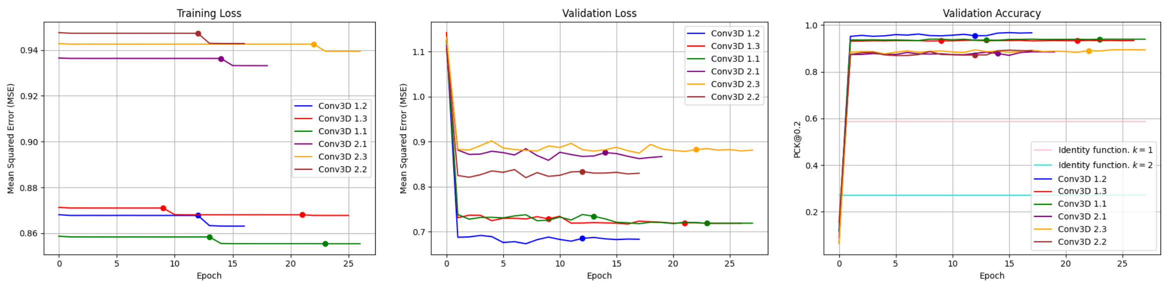
1.2.3 Training and Validation Results

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 experiment and setup it was experimented on. x indicates what shifting-scalar was used when shifting the input keypoints, and y indicates which of the three setups from Section 1.2.2 the experiment 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.

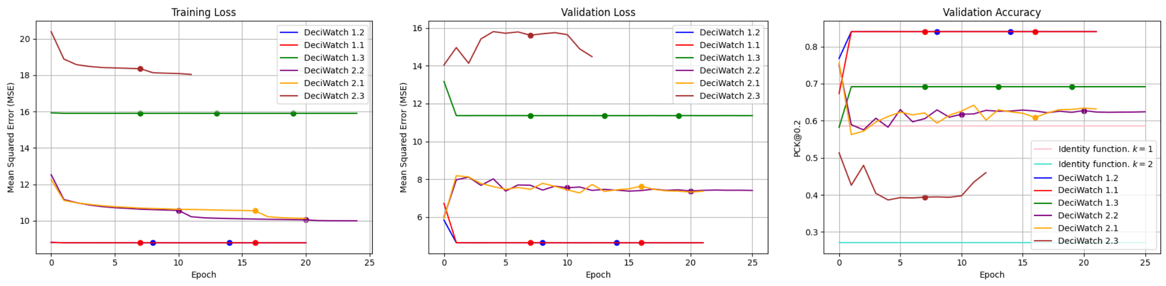
By comparing the validation accuracies of the various models against the identity function we see, that all of the models do learn to somewhat denoise the input data. The simple 3-dimensional convolutional layer seems to be the architecure that generally delivers the greatest results, whereas DeciWatch 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 a pretty substantial effect on the training loss, as 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. Most of the times, 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 and accuracy.

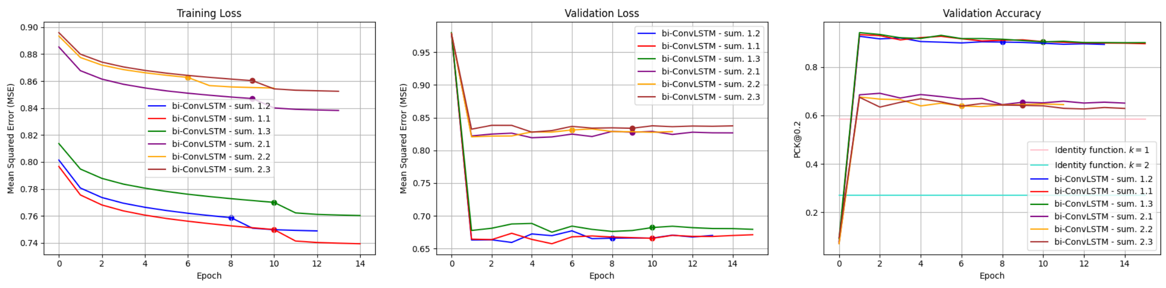
One could look at Figure 1 and argue, that the training of the models generally plateus as soon as after the first epoch, as the evolution of the validation loss and accuracy seems rather flat after the first epoch. However, if we look at when the learning rate reduction happens, we see that more than the five required epochs have passed before the learning rate reduction is taking place, indicating that the learning has not plateued. The evolution of the validation loss and accuracy just seem flat, because the performance difference between the first epoch and the models with their initial weights is so great, that we can have difficuly seeing the performance differences of the following epochs.



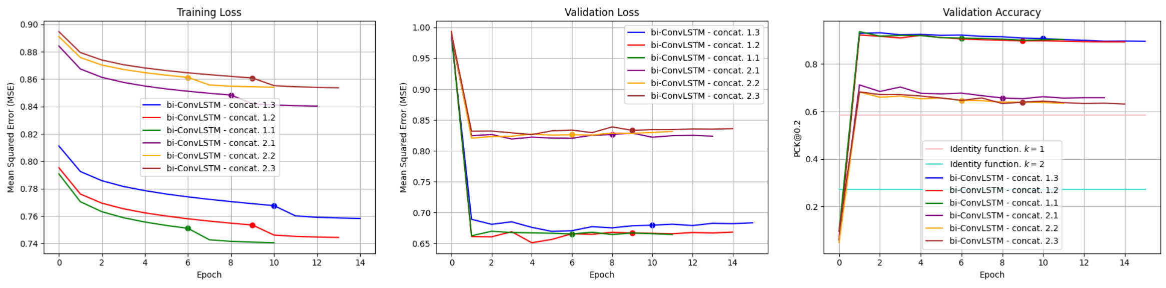
(a) Pretraining results of the 3-dimensional convolutional layer.



(b) Pretraining results of DeciWatch.



(c) Pretraining results of the bidirectional convolutional LSTM with summing.



(d) Pretraining results of the bidirectional convolutional LSTM with concatenation.

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: 3-dimensional convolutional layer. Second row: DeciWatch. Third row: Bidirectional Convolutional LSTM with summing. Fourth row: Bidirectional Convolutional LSTM with concatenation.

Further, we can clearly see the effects of the shifting-scalar on the training of the models, as all of the models peroms 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. However, we have to keep in mind here, that DeciWatch already only considers every fifth frame. This, combined with the fact that setup 3 is only consideren every other frame, makes these two models only consider every tenth frame, which of course makes the prediction very difficult, as these models then have to recover a lot of frames.

1.2.4 Test Results

Accuracy metric	PCK@0.05			PCK@0.1			PCK@0.2		
Threshold distance									
Setup	1.1	1.2	1.3	1.1	1.2	1.3	1.1	1.2	1.3
Identity function									
Conv3D	5.44	33.7	4.61	30.1	60.7	24.1	**94.4**	**97.0**	94.0
DeciWatch	**54.1**	**54.1**	**46.4**	**68.7**	**68.7**	54.8	84.0	84.0	68.9
bi-ConvLSTM - sum.	20.9	21.4	23.5	64.0	61.7	**65.8**	93.8	93.1	**94.6**
bi-ConvLSTM - concat.	21.3	20.7	20.9	62.3	61.1	61.5	94.1	92.7	93.6

Table 1: Testing accuracies of the various developed models for shifting-scalar $k = 1$.

Accuracy metric	PCK@0.05			PCK@0.1			PCK@0.2		
Threshold distance									
Setup	1.1	1.2	1.3	1.1	1.2	1.3	1.1	1.2	1.3
Identity function									
Conv3D	0.80	2.00	0.18	13.7	18.6	9.6	**89.2**	**89.7**	**90.1**
DeciWatch	**8.18**	8.81	5.13	28.7	27.1	18.2	65.1	63.6	48.6
bi-ConvLSTM - sum.	8.05	**9.42**	**7.78**	**30.4**	**31.5**	**29.4**	69.9	68.1	68.2
bi-ConvLSTM - concat.	6.91	8.72	7.77	29.4	31.3	29.0	71.9	68.8	69.1

Table 2: Testing accuracies of the various developed models for shifting-scalar $k = 2$.

We have in Table 1 and Table 2 illustrated the test results of the various models on the testsets for PCK@0.05, PCK@0.1 and PCK@0.2.

For shifting-scalar $k = 1$ we see, that DeciWatch is the architecure that yields the best results for PCK@0.05 and PCK@0.1. However, for PCK@0.2 DeciWatch actually tends to be the worst performing model, which means that most of the predicitons of DeciWatch that are considered correct, are actually very close to the groundtruth, whereas the predictions that are incorrect are pretty far off.

On the other hand, the simple 3-dimensional convolutional layer 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. Thus, the 3-dimensional convolutional layer 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. Similarly to the validation results, the test results do not indicate any major performance differences between the two different types of bidirectional convolutional LSTMs. Likewise, similarly to the validation results, the test results indicate a major effect of a low frame rate for DeciWatch.

If we take a closer look at the four types of architectures we see, that the 3-dimensional convolutional layer generally performs much better on the second setup that it does on the two other setups. As the models of the first and third setup have to learn both translation and scaling, where for the second setup the model only has to learn translation, we can clearly see how big of an impact the task of scaling the data has for this simple model. As for comparison, the performances of the other architectures have generally a lot smaller variance, indicating these models being less impacted by the setup. This is especially visible for DeciWatch, which has no performance differences between the two first setups. However, this is due the architecture working on keypoint-coordinates instead of heatmaps, making it scaling invariant and thus making it, such that there are no differences between the two first setups.

The testing results of shifting-scalar $k = 2$ generally follow the same pattern with some minor differences. In this case, the bidirectioanl convolutional LSTM with summing is actually the model that yields the best results for PCK@0.05 and PCK@0.1 instead of DeciWatch, as it was in the case for $k = 1$. However, DeciWatch does still yied some results that are pretty close to the results of the convolutional LSTM with summing.

Unsurprisingly, all of the models performed worse as the shifting-noise was double. Nonetheless, the 3-dimensional convolutional layer is still the architecture that yields the best results when considering PCK@0.2. However, unlike the previous case, this architecure now completely outperforms the three alternative architectures, when considering PCK@0.2, but 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.

Which architecture is the best at denoising the data depends on one's needs. If one just need a rough estimate of the position of the keypoints, then the 3-dimensional convolutional layer would be the optimal choice, as this model has the highest PCK@0.2-values. However, if one need the model which yields the most exact predictions, then one of the bidirectional convolutional LSTMs would be the optimal choice.

We have further in Table 3 and Table 4 illustrated the keypoint-specific testing PCK@0.2 of the various models.

- $k = 1$
  - Wrist og ankle er de dårligste
  - Ears and Shoulders er de bedste
- $k = 2$
  - Elbow og knee er de dårligste
  - Ears and shoulders er de bedste

Model	Best Epoch	Ear	Shoulder	Elbow	Wrist	Hip	Knee	Ankle
Conv3D 1.1	8	94.3	94.6	94.2	93.9	95.8	94.6	93.6
Conv3D 1.2	15	95.0	**97.5**	**98.7**	**96.6**	**98.3**	**98.6**	**95.1**
Conv3D 1.3	12	94.1	93.7	95.1	92.9	97.0	93.4	92.4
DeciWatch 1.1	3	87.2	88.1	84.1	78.6	86.9	82.7	79.0
DeciWatch 1.2	11	87.2	88.1	84.1	78.6	86.9	82.8	79.0
DeciWatch 1.3	22	73.3	74.9	67.4	61.0	72.3	67.8	63.2
bi-ConvLSTM sum. 1.1	1	95.4	94.4	92.8	91.9	92.4	92.7	92.2
bi-ConvLSTM sum. 1.2	1	94.9	93.4	94.4	93.2	93.5	92.7	94.3
bi-ConvLSTM sum. 1.3	1	95.5	92.2	96.6	**96.6**	92.5	93.6	95.5
bi-ConvLSTM concat. 1.1	1	95.3	95.1	94.1	94.2	93.9	94.4	92.8
bi-ConvLSTM concat. 1.2	1	95.7	93.3	94.6	91.4	90.8	91.5	91.2
bi-ConvLSTM concat. 1.3	2	**95.9**	91.8	93.7	95.4	91.5	94.1	94.0
Total		92.0	91.4	90.8	88.7	91.0	89.9	88.5

Table 3: Keypoint-specific testing PCK@0.2-accuracies of the various models for shiting-scalar $k = 1$.

Model	Best Epoch	Ear	Shoulder	Elbow	Wrist	Hip	Knee	Ankle
Conv3D 1.1	17	85.1	89.5	91.0	92.0	86.0	89.9	91.9
Conv3D 1.2	15	**85.9**	89.7	91.3	**92.2**	88.5	90.8	91.0
Conv3D 1.3	26	85.2	**90.9**	**91.6**	91.9	**88.7**	**91.4**	**92.0**
DeciWatch 1.1	11	77.1	72.6	61.0	57.9	70.4	56.7	58.4
DeciWatch 1.2	5	76.1	71.2	59.6	59.7	68.8	58.3	54.0
DeciWatch 1.3	2	54.8	42.0	41.9	45.3	52.9	40.3	53.6
bi-ConvLSTM sum. 1.1	2	81.0	66.6	66.7	69.2	66.1	72.3	65.3
bi-ConvLSTM sum. 1.2	1	79.9	68.2	66.2	66.6	60.6	68.7	65.1
bi-ConvLSTM sum. 1.3	1	83.8	64.9	63.5	67.3	63.7	65.2	68.5
bi-ConvLSTM concat. 1.1	1	83.9	67.1	66.2	75.0	68.9	67.6	71.7
bi-ConvLSTM concat. 1.2	1	81.7	67.8	66.1	62.7	64.7	65.3	72.0
bi-ConvLSTM concat. 1.3	11	81.8	65.2	65.9	66.2	63.7	67.8	68.7
Total		79.7	72.2	69.2	70.5	70.2	69.5	71.0

Table 4: Keypoint-specific testing PCK@0.2-accuracies of the various models for shiting-scalar $k = 2$.

1.3 Finetuning

In the finetuning stage we will be using the already developed pose-estiamtor to train our temporal-inclusive models. However, we will be freezing the pose-estimator, such that the weights of the model will not change during the training and we will thus only train our temporal-inclusive models. We do this for the following three reasons: (1) the training of the models will be quicker, as we just need to train the tempoeral-inclusive models and not the already developed pose-estimator, (2) we get an greater understanding of the effects of our models when combined with the pose-estimator, as we can clearly see how big of a difference it makes by adding our tempoeral-inclusive models, and (3) we lower the probability of overfitting, as we have less tuneable parameters.

1.3.1 Data Preprocessing

For the ClimbAlong dataset we perform only minor preprocessing. First, the preprocessing of each video is done by having the already developed pose-estimator process the video, such that we have the output heatmaps of the pose-estimator, containing all of the pose-estimations of each video. Next, we preprocess the heatmaps by setting all negative values to 0 and normalizing each heatmap, such that each heatmap sums up to the fixed value $c = 255$ that we used when preprocessing the BRACE and Penn Action datasets, essentially making the heatmaps more similar to the preprocessed heatmaps of BRACE and Penn Action. These heatmaps will then be used as the input for our models.

For the groundtruth heatmaps we create twenty five heatmaps of each frame, similarly to how we did it for the BRACE and Penn Action datasets, however, in this case we use the predicted bounding-box of the pose-estimator as our bounding-box. In cases where the groundtruth keypoint is placed outside of the bounding-box, we place the groundtruth keypoint at the closest border of the bounding-box.

1.3.2 Training Details

Data Configuration Generally, we follow a similar approach to how we did in the pretraining stage. We again use a window-size of $k = 5$ frames, resulting in a total of $9,419$ windows. Also here are we using $c = 255$ as a representation of the placement of each keypoint. We also split the dataset into a non-overlapping and non-repeating training, validation and test set, consisting of 60%, 20% and 20% of the data, respectively. However, we note that one incorrect frame can have a huge impact on the evaluation results, as this frame is used five times during evaluation, due to the small dataset size. For that reason, for the validation and test set we make sure that none of the windows of the same set.

Setups As the finetuning dataset is so small, the fitting of the models is very quick, making us fit all of the developed models from the pretraining stage. For each model we pick the epoch from the pretraining stage, that yielded the highest validation accuracy and use that for finetuning.

Training Configuration The optimization parameters are very similar to the ones from the pretraining stage. We again use the ADAM optimzer with a batch size of 16 and the MSE loss-function. During training, we again keep track of the lowest reached validation loss of an epoch and use learning-rate reduction and early-stopping in a similar manner to how we did in the pretraining stage. However, unlike the pretraining stage, we here use a smaller initial learning rate of 10^{-4}, as the weights only need to be fineadjusted, making us believe that greater learning rate would skew the weights too much.

1.3.3 Results

- Vi skal kun bruge én epoch - nok pga. pretræning

Technical Details