A. Supplementary material

A.1. Dilated convolutions and information flow

Dilated convolutions are a particular form of convolution with a sparse structure, whose kernel points are spaced uniformly and filled with zeros in between. For instance, the discrete filter $h = [\ 1\ \boxed{2}\ 3\ ]$ (where $\boxed{2}$ is the center) becomes $[\ 1\ 0\ \boxed{2}\ 0\ 3\ ]$ with dilation factor $D = 2$, and $[\ 1\ 0\ 0\ \boxed{2}\ 0\ 0\ 3\ ]$ with $D = 3$. This particular structure enables optimized implementations that skip computations over zero points.

Consider a discrete convolution of two signals f (of length N) and h (zero-centered, of length $2M + 1$), which can be computed as

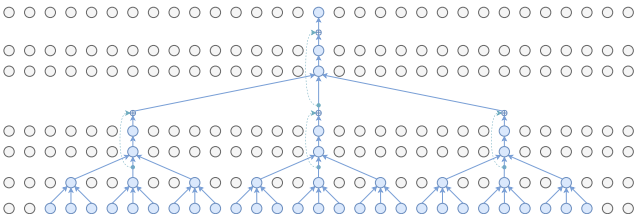
$$(f * h)[n] = \sum_{m=-M}^{M} f[n - m]\, h[m] \tag{2}$$

Instead of spacing the kernel explicitly and applying a regular convolution, a dilated convolution can be computed as

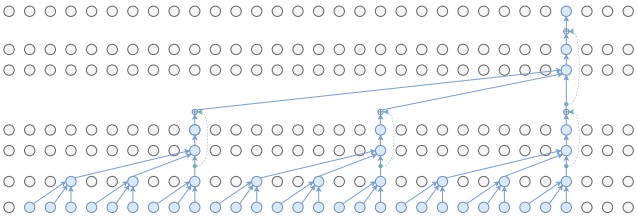
$$(f * h_D)[n] = \sum_{m=-M}^{M} f[D(n - m)]\, h[m] \tag{3}$$

yielding roughly the same computational cost as regular convolutions for the same number of non-zero entries, while increasing the receptive field.

For illustration purposes, in Figure 6 we depict the information flow in our models. We also highlight the difference between symmetric convolutions and causal convolutions.



(a) Symmetric convolutions.



(b) Causal convolutions.

Figure 6: Information flow in our models, from input (bottom layer) to output (top layer). Dashed lines represent skip-connections.

A.2. Computational cost analysis

In this section we show how we computed the computational complexity of our model and that of [16]. The common practice is to consider only matrix multiplications, as other operations (*e.g.* biases, batch normalization, activations) have negligible impact on the final complexity. For [16], we evaluated its reference implementation in TensorFlow. We computed the amortized cost to predict one frame using the TensorFlow profiler, and only counted operations corresponding to matrix multiplications. According to TensorFlow's approximation, multiplying a $N \times M$ matrix by a $M \times K$ matrix has a cost of $2\,NMK$ FLOPs (floating-point operations), which is equivalent to NMK multiply-add operations.

For our model, we adopted the same convention. We provide a sample cost analysis for a model with a receptive field of 27 frames, which consists of 2 residual blocks. Since the matrix multiplications in our model are only due to convolutions, the analysis is straightforward and can be computed by hand.

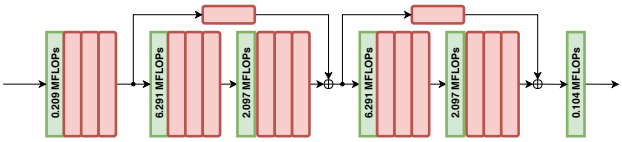


Figure 7: Architecture of a model with a receptive field of 27 frames, with the corresponding amortized cost to predict one frame in convolutional layers.

As can be seen in Figure 7, the model consists of 6 convolutional layers. Disregarding padding (*i.e.* for sequences of length $N \gg 0$), performing a 1D convolution with C_{in} input channels, C_{out} output channels, and width W has a cost of $2\,N\,W\,C_{in}\,C_{out}$ FLOPs, *i.e.* $2\,W\,C_{in}\,C_{out}$ FLOPs per frame. In our 27-frame model, the results can be summarized as follows:

1. $W = 3$, channels $17 \cdot 2 \rightarrow 1024$, cost 0.209 MFLOPs.
2. $W = 3$, channels $1024 \rightarrow 1024$, cost 6.291 MFLOPs.
3. $W = 1$, channels $1024 \rightarrow 1024$, cost 2.097 MFLOPs.
4. $W = 3$, channels $1024 \rightarrow 1024$, cost 6.291 MFLOPs.
5. $W = 1$, channels $1024 \rightarrow 1024$, cost 2.097 MFLOPs.
6. $W = 1$, channels $1024 \rightarrow 17 \cdot 3$, cost 0.104 MFLOPs.

Total: 17.089 MFLOPs per frame.

A.3. Ablation of receptive field and channel size

In Figure 8b we report the test error for different receptive fields, namely 1, 9, 27, 82, and 243 frames. To this end,

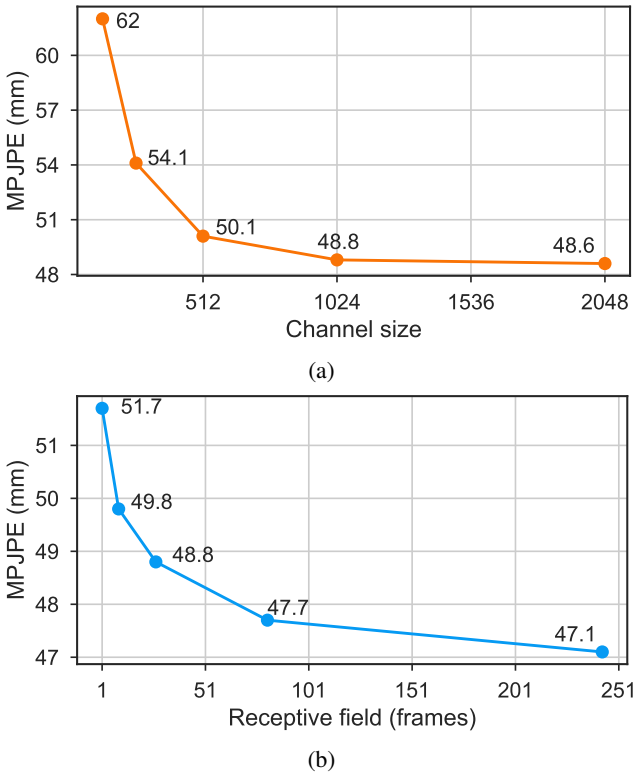


Figure 8: **Top:** Error as a function of the channel size, with a fixed receptive field of 27 frames. **Bottom:** Error as a function of the receptive field, with a fixed channel size of 1024. Fine-tuned CPN detections for both experiments.

we stack a varying number of residual blocks, each of which multiplies the receptive field by 3. In the single-frame scenario, we use 2 blocks and set the convolution widths of all layers to 1, obtaining a model functionally equivalent to [34]. As can be seen, the model does not seem to overfit as the receptive field increases. On the other hand, the error tends to saturate quickly, suggesting that the task of 3D human pose estimation does not require modeling long-term dependencies. Therefore, we generally adopt a receptive field of 243 frames. Similarly, in Figure 8a we vary the channel size between 128 and 2048, with the same findings: the model is not prone to overfitting, but the error saturates past a certain point. Since the computational complexity increases quadratically with respect to the channel size, we adopt $C = 1024$.

A.4. Data augmentation and convolution type

When we remove test-time augmentation, the error increases to 47.7 mm (from 46.8 mm) in our top-performing model. If we also remove train-time augmentation, the error reaches 49.2 mm (another +1.5 mm).

Next, we replace dilated convolutions with regular dense convolutions. In a model with a receptive field of 27 frames and fine-tuned CPN detections, the error increases from 48.8 mm to 50.4 mm (+1.6 mm), while also increasing the number of parameters and computations by a factor of ≈ 3.5. This highlights that dilated convolutions are crucial for efficiency, and that they counteract overfitting.

A.5. Batching strategy

We argue that the reconstruction error is strongly dependent on how the model is trained, and we suggested to generate minibatches in a way that only one output frame at a time is predicted. To show why this is important, we introduce a new hyperparameter – the *chunk size* C (or *step size*), which specifies how many frames are predicted at once per sample. Predicting only one frame, i.e. $C = 1$, requires a full receptive field F as input. Predicting two frames ($C = 2$) requires $F + 1$ frames, and so on. It is evident that predicting multiple frames is computationally more efficient, as the results of intermediate convolutions can be shared among frames – and in fact, we do this at inference. On the other hand, we show that during training this is detrimental to generalization.

Figure 9b illustrates the reconstruction error (as well as the relative speedup in training time) when training a 27-frame model with different step sizes, namely 1, 2, 4, 8, 16, and 32 frames. Since predicting multiple frames is equivalent to increasing the batch size – thus hurting generalization [25] – we make the results comparable by adjusting the batch size so that the model always predicts 1024 frames. Therefore, the 1-frame experiment adopts a batch size of 1024 sequences, which becomes 512 for 2 frames, 256 for 4 frames, and so on. This methodology also ensures the models will be trained with the same number of weight updates.

The results show that the error decreases in conjunction with the step size, at the expense of training speed. The impaired performance of the models trained with high step size is caused by correlated batch statistics [14]. Our implementation optimized for single-frame outputs achieves a speed-up factor of ≈ 2, but this gain is even higher on models with larger receptive fields (*e.g.* ≈ 4 with 81 frames), and enabled us to train the model with 243 frames.

A.6. Optimized training implementation

Figure 10 shows why our implementation tailored for single-frame predictions is important. A regular implementation computes intermediate states layer by layer. This is very efficient for long sequences, as the states computed in layer n can be reused by layer $n + 1$ without recomputing them. However, for short sequences, this approach becomes inefficient because states near boundaries are not used. In the extreme case of single-frame predictions (which we use for training), many intermediate computations are wasted,

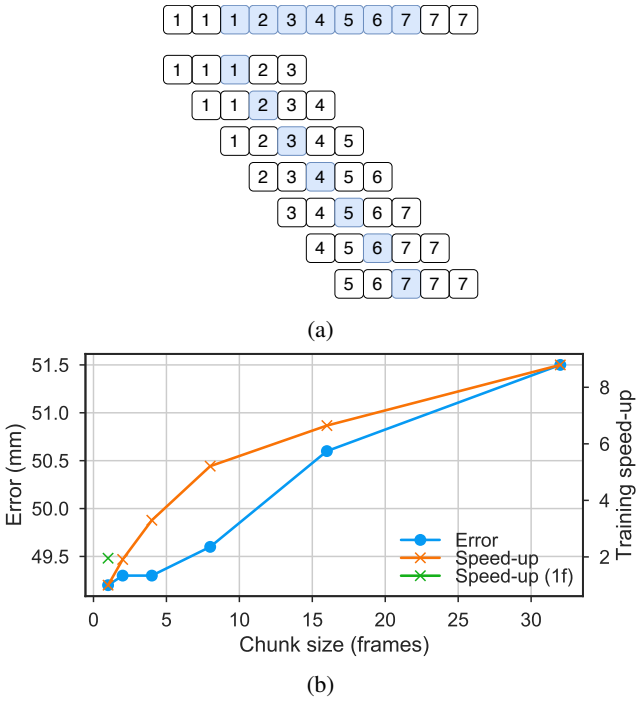


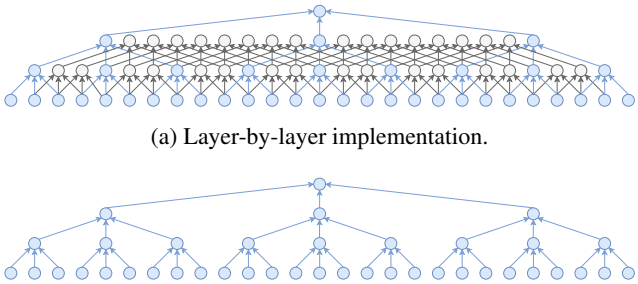
Figure 9: **Top:** batch creation process *for training*. This example shows a video of 7 frames which is used to train a model with a receptive field of 5 frames. We generate a training example for each of the 7 frames, such that only the center frame is predicted. The video is padded by replicating boundary frames. **Bottom:** reconstruction error and training speed-up with different step sizes. The speed-up is relative to $C = 1$. The `1f` variant shows the speed up corresponding to the implementation optimized for single-frame predictions.

as can be seen in Figure 10a. In this case, we replace dilated convolutions with strided convolutions, making sure to obtain a model which is functionally equivalent to the original one (*e.g.* by also adapting skip-connections). This strategy ensures that no intermediate states will be discarded.

As mentioned, at inference we use the regular layer-by-layer implementation since it is more efficient for multi-frame predictions.

A.7. Demo videos

The supplementary material contains several videos highlighting the smoothness of the predictions of our temporal convolutional model compared to the single frame baseline. Specifically, we show side by side the original video sequence, poses predicted by the single-frame baseline, poses from the temporal convolutional model as well as the ground-truth poses. Some demo videos can also be found at https://dariopavllo.github.io/VideoPose3D.



(a) Layer-by-layer implementation.

(b) Implementation optimized for single-frame predictions.

Figure 10: Comparison between two implementations for a single-frame prediction, receptive field of 27 frames. In the layer-by-layer implementation many intermediate states are wasted, whereas the optimized implementation computes only required states. As the length of the sequence increases, the layer-by-layer implementation becomes more efficient.