Human Motion Prediction: With Great Power Comes Great Res-*pose*-ability

Abhinandan Krishnan
akrishnan331@gatech.edu

Prajwal Bhaskar Bharadwaj
pbharadwaj33@gatech.edu

Pranay Mathur
pranay.mathur@gatech.edu

Surya Prakash Senthil Kumar
skumar671@gatech.edu

Abstract

Human motion prediction is a vital task for various applications involving human-machine interaction and scene understanding and can be formulated as a time-series generative modelling task. We explore standard baseline implementations to identify which of these can best model spatio-temporal relations of our problem. We attempt to provide comparable performance at reduced computational requirements and analyze under-represented architectures, by proposing two novel architectures inspired by convolutional sequence-to-sequence models and temporal convolutional networks which outperform similarly sized baselines. We demonstrate our results on the AMASS-ACCAD dataset and support design considerations with rigorous analysis shown in Appendix 5. Our implementation can be found on GitHub

1. Introduction and Motivation

In this paper, we tackle the problem of human-motion prediction with a focus on how different architectures affect performance. The problem can be defined as predicting a sequence of the most likely future poses of a human given the history of poses.

Predicting human motion is a critically important task in the field of human-robot interaction, tracking humans through successive time frames and motion generation in computer graphics. Humans are pre-disposed to predicting future human motions in tasks such as crowd navigation, playing sports in teams, or even regular social interaction with other humans. Giving robots and self-driving cars this ability makes their operation safer and easier to work with as shown by previous research [12, 4, 9]. We argue that, in most robotic applications, computational resources are not abundantly available and applications like self-driving cars require faster inference and excellent real-time performance, making our research impact both of these fields.

Current practices include using methods that are based on RNNs, Convolutions and Transformers. RNN-based methods have been previously implemented and surveyed since they are good at handling sequential data [13]. However, the performance is short-sighted producing subpar results to go along with longer training times. Seq2Seq models perform well while handling inputs that they are trained upon but have difficulties generalizing to rare and unseen inputs. They also struggle with capturing long-term dependencies due to vanishing gradients. On the other hand, transformers [18] are robust, faster and have non-recurrent behaviour but they possess a limited ability in handing variable-length input sequences.

In this study, we propose the development and analysis of two novel architectures inspired by convolutional networks and show that they use a small parameter space in comparison to the transformers without significant degradation in performance. For our analysis, we use the open-sourced AMASS [15] dataset and the ACCAD dataset, in particular, [1] due to its size. AMASS unifies multiple parameterizations used by different datasets into a single representation. Every joint in the data is represented as a set of 3**x**3 rotation matrices, $R_{matrix} \in \mathbb{R}^9$ of their respective angles, and is flattened to a vector. The **input representation** of our data consists of vectors of human-poses, concatenated over 120 time frames and parameterized by 24 joints. Thus the input is a 2D matrix of size $120\mathbf{x}24\mathbf{x}9 = 120\mathbf{x}216$ and the size of the **output representation** is $24\mathbf{x}24\mathbf{x}9 = 24\mathbf{x}216$, where 24 is the number of future predictions.

2. Approach

We used the Fairmotion library from Facebook [6]. No pre-processing or post-processing was required and data-loading was handled by the library with minor changes for analysis. The library provides the implementation of several baselines which we trained, hyper-paramater tuned and analyzed to compare the **two novel models** inspired by convolutional sequence-to-sequence models[11] and temporal convolutional networks [3]. To the best of our knowledge,

these are the first implementations in **PyTorch**. For the purposes of evaluation and analysis, we made several changes to the Fairmotion library which we list below.

1. Contributed two novel implementations, the Convolutional Sequence-to-Sequence model and Temporal Convolutional Network to the Fairmotion library.
2. Changes to log model metrics, visualize weights, loss values and MAE values.
3. Modified training scripts to load and train our own model architectures.
4. Changed optimizers and loss functions used, apart from trivial changes to hyper-parameters and architectures

A formal notation of the problem is that we are given a series of seed human poses $\mathbf{X_{1:t}} = [x_1, x_2, \ldots, x_t]$, where each $x_i \in \mathbb{R}^{JXA}$ is a parameterization of a human pose with J joints with each joint angle representation A. And our goal given these sequence of poses $\mathbf{X_{1:t}}$ is predicting the next T set of poses, $\mathbf{\hat{X}_{t+1:t+T}}$. We also attempt to compare the computational load of architectures and propose two novel architectures that address this since the main problem that we forecasted was the availability of computational resources, specifically in terms of the availability of memory to train larger models. As expected, we faced the Out-of-Memory (OOM) error and had to solve it by increasing swap-space in our SSD and reducing model parameters.

2.1. RNN

The RNN model consists of a single LSTM module to take an input pose and generate a pose prediction for the next time step. It involves a forward pass of the LSTM module on the given inputs and generates the outputs for each time step. It further converts the source and target tensors to the (seqlen, batchsize, inputdim) format, applies dropout to the source tensor, and generates as many poses as in the target during training. Further, it generates the output sequence and during training, concatenates the encoder and decoder outputs. Finally, it transposes the output tensor to the (batchsize, seqlen, inputdim) format and returns it. This architecture is tested for varying values of hidden size and choice of optimizer.

2.2. Seq2Seq

The task of a sequence-to-sequence model is to generate a target sequence from a given seed sequence[11]. Most sequence-to-sequence models consist of two parts, an encoder that encodes the seed sequence into a hidden variable and a decoder that generates the target sequence from the hidden variable. The seq2seq model uses an encoder-decoder architecture to generate a target sequence from a given seed sequence. The encoder encodes the seed sequence into a hidden variable and the decoder generates the target sequence from the hidden variable.

We performed multiple tuning experiments on the seq2seq model like varying the hidden dimensions and teacher forcing ratio in an attempt to test which aspect of its design would allow us to get optimal performance with the fewest parameters.

2.3. Tied Seq2Seq

In a traditional Seq2Seq model, the encoder and decoder are separate networks, with their own sets of weights and biases. A tied sequence-to-sequence (Seq2Seq) model is a type of neural network architecture where the encoder and decoder share some or all of their parameters. In contrast, a tied Seq2Seq model has a shared embedding layer or a shared output layer between the encoder and decoder.

By sharing parameters between the encoder and decoder, tied Seq2Seq models can be more efficient in terms of model size and computation, as they have fewer parameters to learn. This can also lead to better generalization performance and can help prevent overfitting, as the shared parameters can enforce consistency between the encoder and decoder. The performances of the tied seq2seq and the traditional seq2seq are analysed in the subsequent sections and shown in Table 3.2.

2.4. Transformer

Transformers [18] are one of the most successful architectures for tasks that involve sequence-to-sequence modelling and predicting data with long-range dependencies. It consists of an encoder which takes input embeddings augmented with positional encodings. The encoder contains multi-head attention mechanisms where each head receives different linearly projected versions of the queries, keys, and values. Each of these heads produces an output in parallel. These are followed by a fully connected feed-forward network consisting of two linear layers with Rectified Linear Unit (ReLU) activation in between. These also contain a residual connection around them. The output of this is then passed to a decoder which receives the previous output with positional embeddings and implements a multi-head self-attention mechanism on it. The decoder is only allowed to attend to tokens prior to the current one in the sequence through a process of masking. In the next layer of the decoder, it is concatenated with the output of the encoder where the decoder can attend to all input tokens of the sequence. This is then passed to a fully connected layer to generate the next prediction of the sequence. We tuned multiple aspects of the architecture and hyper-parameters and have analyzed the results in the next section

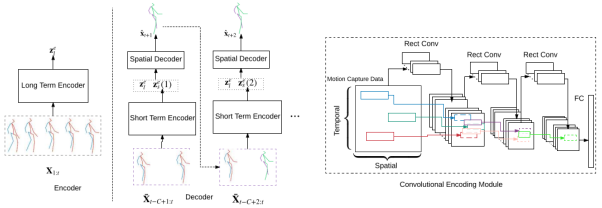

Figure 1. Convolutional Sequence-To-Sequence Model Architecture [11]

2.5. Convolutional Sequence to Sequence Model

We propose an auto-regressive model based on convolutional sequence-to-sequence architecture for human motion prediction [11]. The structure of convolutional networks allows them to intrinsically model and learn both spatial dependencies as well as long-term temporal dependencies[5]. The architecture consists of a convolutional long-term encoder which is used to encode the entire sequence of motion $\mathbf{X_{1:t}}$ into a latent variable, z_l^e, encoding the entire sequence, which is then used by a decoder to predict the sequence over the next T frames $\hat{\mathbf{X}}_{\mathbf{t+1:t+T}}$. Prior to being passed to the decoder the long-term encoded latent variable z_l^e is concatenated with a short-term hidden variable, z_s^e, by a short-term encoder encoding a shorter sequence. These are then passed to a spatial decoder which maps the long and short-term hidden variables, z_l^e and z_s^e to motion predictions $\hat{\mathbf{X}}_{\mathbf{t+1:t+T}}$. The intuition behind using a structure lies in the ability of CNNs to capture both long-horizon invariant information and short-term dynamic information of human motion. The input consists of each input sequence $\mathbf{X_{1:t}}$, acting as a row forming the spatial domain and successive frames below this forming the temporal axis. To ensure the receptive field of the encoder can capture correlations of joints from different joints a rectangular kernel with size $2\mathbf{x}7$, as in the original paper, is used.

Our proposed architecture uses 3 convolutional layers with rectangular kernels and ReLU activation in between followed by a single fully connected layer for the long and short-term encoders, that create z_l^e and z_s^e respectively. In the interest of keeping model size small, our short-term encoder only encodes the last two data frames. For the decoder, we use two fully-connected layers with the first mapping the concatenated hidden variables to a lower dimension followed by the second layer which maps the output to a single frame of human poses $\hat{\mathbf{X}}_{\mathbf{t+1:t+T}}$. The entire architecture is shown in **Figure** 1.

A detailed analysis of the model with experiments indicating why certain design decisions were made, along with ablation studies has been presented in the next section.

2.6. Temporal Convolutional Network

Research on CSS showed that convolutional kernels can capture long-term dependencies and can model spatiotemporal dependencies without needing to depend on too many steps ($O(logn)$ steps) in the past to determine the future joint angle estimates. The reason for its low validation MAE is that the sliding of CSS kernels through different time frames tends to peek into the future as shown in Fig. 4 which produces sub-optimal outputs during inference. The number of parameters needed to train the CSS is almost 4x a conventional transformer to achieve a similar MAE. The disadvantage of RNNs is the number of steps needed to predict the future time frames is of the order $O(n)$ for RNN and hence takes a long time to train. Hence our temporary problem statement is as follows.

- Build a convolution kernel which prohibits looking into the future (temporal masking) so as to improve the validation MAE.

- Improve the receptive field of the network without increasing the number of learnable model parameters.

This motivated us to move in the direction of 1D convolutions as they can effectively model temporal dependencies [17], especially in the domains of action segmentation, trajectory prediction and machine translation. Temporal Convolutional Network (TCN) was originally proposed by [10] for human action segmentation in videos. It uses a temporal encoder-decoder architecture that consists of 1D temporal convolutions (TC) layers to encode temporal dynamics from the video frames and provide a suitable action prediction. TCN modules use a hierarchical architecture, similar to conventional 2D CNNs, to increase receptive fields and efficiently model the long-term dependencies of sequential data. However, the use of TCNs has been very limited in predicting human motion, owing to the outstanding performance of transformers and generative adversarial networks in this area in recent years. Thus, we study the effect and performance of TCNs for human motion prediction by using a slightly modified version of the architecture proposed by [3] for our experiments. To prevent the kernels from learning by peeking into the future, we have used *Causal Convolutions*, which zero out all future instances by appropriately padding the input before convolution. By keeping a unity dilation factor and kernel size, our network's behaviour converges into a Hidden Markov Model. Our proposed architecture, is shown in Fig. 2.

To prevent vanishing gradients, we have a residual connection with an identity mapping of the input at the end of every block. At each TC layer following dilated-causal convolution, the output is then batch normalised followed by an activation with LeakyReLU.

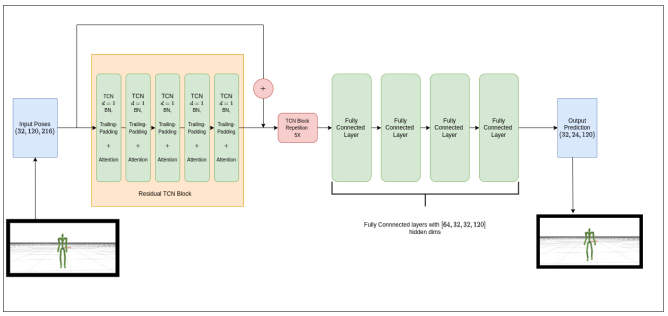

Figure 2. TCN Architecture

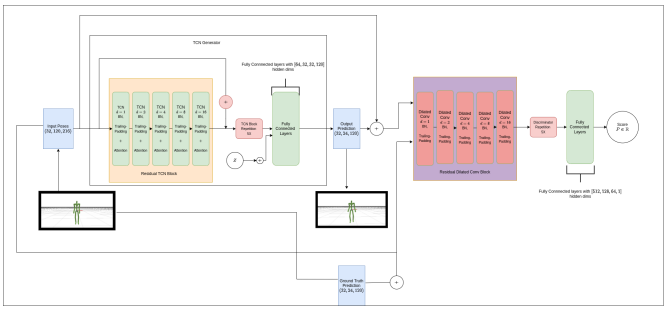

Figure 3. TCN-GAN Architecture

2.7. GANs for Motion Prediction

2.7.1 Architecture

We study the performance of Generative Adversarial Networks to generate a set of distinct and kinematically valid future poses given an input pose conditioned on a latent embedding $Z \sim N(0, 1)$. Since the TCN outperforms all the other networks that we researched, it forms the generator of the GAN. The discriminator follows a similar structure as proposed by [3] and [2] except for the final output layer where we use a ReLU activation instead of Sigmoid. The proposed architecture is shown in Fig. 3

2.7.2 Loss Functions

Training of GANs can be quite cumbersome due to the min-max nature of the generator G and the discriminator D objectives. Thus we need to be careful about picking up appropriate loss functions and carefully monitor the flow of gradients into the individual networks for stable training. The discriminator is trained in order to minimize the following objective,

$$\mathbf{L_{discrim}} = \mathbf{E}[\mathbf{D}(\mathbf{X_{1:t}} : \mathbf{G}(\mathbf{X_{1:t}}, \mathbf{Z}))] - \mathbf{E}[\mathbf{D}_\theta(\mathbf{X_{1:t}} : \mathbf{Y})] + \lambda_{\mathbf{reg}} ||\mathbf{W_{discrim}}||^{\mathbf{2}}$$

where : represents the concatenation of two sequences about the time dimension and λ_{reg} which is the regularisation factor is kept as 1. Similarly, the generator loss function is given by,

$$L_{gen} = L_{recon} + L_{adv} + L_{reg} \tag{1}$$

$$L_{recon} = \frac{1}{\mathbf{T}} \sum_{t'=t+1}^{t'=t+T} |||\hat{x}'_t - x'_t|||^2 \tag{2}$$

$$L_{adv} = \mathbf{E}[\mathbf{D}(\mathbf{G}(\mathbf{X_{1:t}}, \mathbf{Z}))] \tag{3}$$

$$L_{reg} = \lambda_{\mathbf{reg}} ||\mathbf{W_{gen}}||^{\mathbf{2}}, \lambda_{reg} = 1 \tag{4}$$

3. Experiments and Results

3.1. Evaluation Policy

To compare the performance of our proposed models we experimented with different versions of baseline implementations, and minor changes to architectures such as hidden sizes, number of attention heads, number of layers, optimizers, learning rate schedulers, teacher-forcing ratios and ablation studies. We highlight that not all of these hyper-parameters are applicable to every architecture and so a standard policy was used for changing the hyper-parameter to evaluate baselines. Hidden sizes were increased from 128 to 512 in factors of two, performance with both SGD and Adam[8] **optimizers** were checked, the effect of teacher-forcing was evaluated, the number of layers was varied beginning from 1 to the maximum value that could be tested given GPU memory constraints, and for proposed architectures, network blocks were removed to examine how performance varied in their absence. All parameters were left **trainable and no pre-trained weights were used**. Concatenation layers were the only non-learnable layers. All architectures were run for 100 epochs and training losses, validation losses, GPU memory usage, number of parameters and MAE over 6,12,18 and 24 frame predictions were recorded. All architectures were trained on 4GB Nvidia RTX 3050 and 3050Ti GPU's using CUDA [16].

3.2. Performance Metrics and Loss Function

We measured the success and performance of our algorithm using the Mean Angle Error (MAE) which is widely used and accepted metric for the task of human-motion prediction [14]. Given the Euler angles $\hat{x}_{n,k}$, MAE can be defined as shown below.

$$MAE = (N_a K)^{-1} \sum_{n=1}^{N_a} \sum_{k=1}^{K} |\hat{x}_{k,n} - x_{k,n}| \tag{5}$$

where $\hat{x}_{k,n}$ denotes the predicted k^{th} angle in frame n and $x_{k,n}$ is the ground truth. We chose this metric instead of other metrics such as Mean Per Joint Position Error (MPJPE) and Normalized Power Spectrum Similarity

(NPSS) in the interest of simplicity, the horizon of our predictions and its widespread acceptance.

During training, we use the mean squared error of the predicted poses as the loss function:

$$\mathbf{Loss}_{model} = \frac{1}{\mathbf{T}} |||\hat{\mathbf{X}}_{\mathbf{t+1:t+T}} - \mathbf{X}_{\mathbf{t+1:t+T}}|||^2 \quad (6)$$

$$= \frac{1}{\mathbf{T}} \sum_{t'=t+1}^{t'=t+T} |||\hat{x}'_t - x'_t|||^2 \quad (7)$$

Two different types of regularizing techniques are used to **prevent overfitting** - a dropout layer and $l2$ regularizer. Our validation graphs indicate that the model was able to generalize well to unseen input sequences.

3.3. Baseline Analysis

We now discuss the results of hyper-parameter tuning the baseline models i.e., RNN, Seq2Seq, and Transformers. Table 3.2 shows the performance comparison between the baselines. Tied seq2seq performs better than the seq2seq model while using significantly fewer parameters compared to latter with the same number of layers. We observed the tied seq2seq model converged better than the Transformer over time 6. We believe this is because of the fact that the tied seq2seq model uses a shared LSTM layer with the same bases and weights and hence is able to capture the long-term joint angle dependency better. The tied seq2seq was trained for 3 layers which gave an optimal performance in terms of a number of parameters and the MAE values at different frames. As we expected increasing the hidden size lead to an improvement in performance except in Seq2Seq where it began to overfit. With transformers, a major improvement was seen with an increase in the number of attention heads. We argue that these improvements can be attributed to the model being able to capture the dependencies between joints of the sequence due to more heads processing the sequence in parallel. Due to regularization and dropout of 0.2, no overfitting occurred with increasing hidden size but the validation loss decrease was not observed to be smooth. Initially, we attributed this to the learning rate being too high and but an additional factor was that we used only dropout while training the transformer and not $l2$ regularization. RNN and Seq2Seq models were run with a batch size of 32, but we had to reduce our batch size to 4 to get the transformer model to fit on the GPU. We postulate that a higher batch size will lead to smoother batch gradients resulting in lower fluctuation. We believe that the drop in performance when the number of layers is increased can be attributed to a similar reason. Training the data on a higher batch size with more instances and over more epochs will result in better learning. We also believe this will ensure a lower overall validation loss with less fluctuation than seen in **Figure** 7. We use Student-Teacher Curriculum-Learning (CL) where initially the input to the model is the original input sequence with weight 1 but this gradually drops to 0 over time. In transformers, we observed that the training loss varies smoothly when the student-teacher curriculum learning is used in comparison to when the weight of teacher forcing is set constantly to 1. This explains the variation in loss seen at 50 epochs when the teacher forcing drops to 0. The losses can be seen in Figure 7.

3.4. CSS Analysis

The experiments conducted include the effect of the optimizer, hidden size of the encoder and decoder, loss function, convolutional kernel sizes and ablation studies keeping only the long-term encoder and subsequently only the short-term encoder. All experiments were run for 100 epochs with a learning rate scheduler that reduces the learning rate when validation plateaus out. A general trend observed was increasing validation MAE with increasing hidden dimensions of the encoder and decoder. This however peaked at a hidden size of 512 where slight overfitting was observed since training loss decreased but validation loss increased along with MAE values. A hidden size of 256 was an optimal balance. Change in optimizer from Adam to SGD resulted in a trivial improvement and resulted in the best performance 5. The major difference was observed with convolutional kernel sizes. We tested results with kernel sizes of $2\mathbf{x}7$, $4\mathbf{x}4$ and $7\mathbf{x}2$ and obtained the best results with a kernel size of $2\mathbf{x}7$. We argue that this is because performing a rectangular convolution kernel over the spatial axis captures the dependencies between different body parts most effectively [11]. This is illustrated in **Figure** 4. Our ablation studies indicate that the long-term encoder was an extremely crucial aspect of the architecture as performance deteriorated significantly without it and improved significantly with it. We argue that this was crucial in capturing long-term dependencies and helps the decoder take into account the history of poses rather than just the immediate history of the last two poses. To prevent overfitting we used dropout in the encoders and obtain smooth losses as seen in **Figure** 7.

3.5. TCN Analysis

Similar to CSS, the experiments conducted on TCN were run for 100 epochs with the learning rate scheduler fixed from above and a batch size of 32 was used throughout. The number of hidden dimensions was set as 64 instead of 128 as proposed by [3] and the number of attention heads were fixed as 1 due to CUDA compute constraints. Owing to better performance over SGD on CSS, Adam was the choice of optimizer for these experiments. The ablation study was conducted to understand the importance of temporal attention for TCN and our results are tabulated in **Table** 5. For our experiments, we used a temporal attention head that computes scores between the input poses using

Model	Num. Params	GPU Memory	Num. Layers	Hidden Dim.	Training Loss	Val. loss	MAE @6	MAE @12	MAE @18	MAE @24
RNN	1,605,848	1.35 GB	1	512	1.70E-02	1.79E-02	12.8180	26.0632	39.1999	52.4308
Seq2Seq	356,424	1.10 GB	1	256	2.45E-03	6.63E-03	4.8862	11.5677	19.2466	27.3418
Tied-Seq2Seq	694,344	1.20 GB	3	256	1.04E-03	5.88E-03	4.6842	10.7567	17.4765	25.1659
TF	4,431,576	3.16 GB	1	512	5.97E-02	4.38E-02	4.2841	12.5953	22.3869	32.8969
CSS	15,033,688	2.40 GB	-	256	7.95E-03	2.05E-02	8.1264	15.6363	23.6327	32.1456
TCN-A	1,967,288	2.40 GB	-	64	5.8E-03	3.8E-03	2.0112	6.8700	13.3982	20.8592
GAN-WHR	1,206,369	1.8 GB	-	64	**2.29**	**3.710E-03**	**1.9010**	**6.5865**	**12.9655**	**20.2525**

Table 1. Comparative results of all architectures with the best results highlighted in bold. TF-Transformer. CSS-Convolutional Sequence-to-Sequence. TCN-A Temporal Convolutional Network with Attention. GAN-WHR (With Hinge Loss and ReLU)

scaled dot product and Softmax. The training time was relatively small for TCN when compared to other networks due to its reduced number of learnable parameters. We can observe that, the TCN w/attention is able to perform relatively better when compared to its counterpart. This can be attributed to the fact that the temporal attention head allows the network to focus more on certain frames that provide important contextual information for motion prediction. Another ablation study is the influence of residual connections between the blocks and it can be clearly seen that without the residual connection, the MAE is reduced. This is because that the model fails to capture the long-term temporal dependencies and thus the problem of vanishing gradients.

3.6. TCN-GAN Analysis

The generator of the TCN-GAN has a similar architecture to that of proposed TCN model and number of discriminator layers is fixed as 5 with the last layer being mapped with Sigmoid activation. The training of the GAN was carried out in a conventional fashion where the discriminator and the generator are updated at every step and all the hyperparameters were the same as above. After training with the above settings, We observed that the discriminator saturated quickly without affecting the MAE even though the generator loss decreased consistently. We attributed the reason to be due to the fact that the quality of generated samples was good enough to effectively fool the discriminator but was not good enough to match the real data. Also, we reasoned that there wasn't a continuous gradient flow within the network due to the vanishing gradient problem caused by the Sigmoid activation. We came to this conclusion after executing multiple trials with loss functions like the one proposed by [7] to enforce the ρ-Lipschitz constraint on the gradient for a stable GAN training. GAN validation loss is higher than training loss since the former only considers reconstruction loss 1.

To ameliorate the issue, we used an SVM-like Hinge Loss formulation for the discriminator. In addition to that, we replaced the terminal Sigmoid with a ReLU activation to prevent gradient saturation. This approach turned out to be successful in reducing the MAE as it can be seen in Table 3.2. Other methods like infusing arbitrary noise in the discriminator's linear layers and training the discriminator with half the speed as the generator were carried out but their performance was sub-optimal. For space constraints, we will be attaching only the successful trials from the approaches taken. and the rest of the results will be attached in this sheet.

3.7. Results

Having discussed the intricacies of individual model performance we now present a comparison of different architectures. The results of our runs can be seen in **Table** 3.2. Standard RNN implementations contain fewer parameters than other models but fail to perform effectively where predictions over a longer horizon is required. We argue that this is because there is no aspect of the model that helps it capture long-term dependencies effectively. Our proposed model Temporal Convolutional Networks (TCN) clearly outperforms all other architectures while the Convolutional Sequence-to-Sequence model (CSS), due to its receptive field was able to capture long-term dependencies better, producing better results than a transformer as seen in the MAE@24 metric even though the transformer performs better at MAE@6 as seen in **Figure** 6. Additionally, we found that all these models had comparable GPU utilization and TCN required significantly fewer parameters. The large number of parameters in the Convolutional Sequence-to-Sequence Model can be attributed to the fully connected layers in its decoder. Despite this GPU utilization was capped at 2.4 GB. Since we were able to obtain better MAE results with fewer parameters we **succeeded** in our endeavour. Our detailed results with raw data can be found in the sheet here.

4. Conclusion and Future Work

Human motion prediction and trajectory estimation is critical when it comes to autonomous navigation. In this work, we proposed two novel implementations, CSS and TCN-GAN that use a small parameter space and convolutions to perform motion prediction and have presented their performance against the baseline sequential models. Future work could include, exploring convolutional networks along with robust generators and discriminators since our analysis shows this is a promising future direction.

References

[1] Advanced Computing Center for the Arts and Design. ACCAD MoCap Dataset. 1

[2] Emad Barsoum, John Kender, and Zicheng Liu. Hp-gan: Probabilistic 3d human motion prediction via gan, 2017. 4

[3] Qiongjie Cui, Huaijiang Sun, Yue Kong, Xiaoqian Zhang, and Yanmeng Li. Efficient human motion prediction using temporal convolutional generative adversarial network. *Information Sciences*, 545:427–447, 2021. 1, 3, 4, 5

[4] Zan Gao, Leming Guo, Weili Guan, An-An Liu, Tongwei Ren, and Shengyong Chen. A pairwise attentive adversarial spatiotemporal network for cross-domain few-shot action recognition-r2. *IEEE Transactions on Image Processing*, 30:767–782, 2021. 1

[5] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. Convolutional sequence to sequence learning. 05 2017. 3

[6] Deepak Gopinath and Jungdam Won. fairmotion - tools to load, process and visualize motion capture data. Github, 2020. 1

[7] Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron Courville. Improved training of wasserstein gans, 2017. 6

[8] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[9] H.S. Koppula and A. Saxena. Learning spatio-temporal structure from rgb-d videos for human activity detection and anticipation. *30th International Conference on Machine Learning, ICML 2013*, pages 1829–1837, 01 2013. 1

[10] Colin Lea, Rene Vidal, Austin Reiter, and Gregory D. Hager. Temporal convolutional networks: A unified approach to action segmentation, 2016. 3

[11] Chen Li, Zhen Zhang, Wee Lee, and Gim Lee. Convolutional sequence to sequence model for human dynamics. pages 5226–5234, 06 2018. 1, 2, 3, 5

[12] Quan Liu, Zhihao Liu, Bo Xiong, Wenjun Xu, and Yang Liu. Deep reinforcement learning-based safe interaction for industrial human-robot collaboration using intrinsic reward function. *Advanced Engineering Informatics*, 49:101360, 2021. 1

[13] Kedi Lyu, Haipeng Chen, Zhenguang Liu, Beiqi Zhang, and Ruili Wang. 3d human motion prediction: A survey. *arXiv preprint arXiv:2106.03305*, 2021. 1

[14] Kedi Lyu, Haipeng Chen, Zhenguang Liu, Beiqi Zhang, and Ruili Wang. 3d human motion prediction: A survey, 2022. 4

[15] Naureen Mahmood, Nima Ghorbani, Nikolaus F. Troje, Gerard Pons-Moll, and Michael J. Black. AMASS: Archive of motion capture as surface shapes. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 5441–5450, Oct. 2019. 1

[16] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020. 4

[17] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio, 2016. 3

[18] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017. 1, 2

5. Work Division and Appendix

Student Name	Contributed Aspects	Details
Abhinandan Krishnan	Seq2Seq and Tied Seq2Seq Report	Analysis, Hyper-parameter tuning, Minor Modifications in Architecture. Generated plots used to compare performance of models Analysis of Seq2Seq based models
Prajwal Bhaskar Bharadwaj	RNN and Optimizer study Report	Hyper-parameter tuning, Execution time and memory comparison, CUDA and swap space setup for reduced runtime, Analysis of RNN models
Pranay Mathur	Convolutional Sequence-to-Sequence Models Transformers Report	Implementation, Analysis, Hyper-parameter Tuning, Ablation Studies of CSS. See css.py, css_ablation.py, css_encoder_decoder.py, Table 5 Analysis, Hyper-parameter tuning, Minor Modifications in Architecture. See Table 5 Analysis of Transformers, CSS, Overall Results Analysis
Surya Prakash	TCN TCN-GAN Report	Complete implementation, analysis and ablation studies for both the networks. Implemented loss functions and modified training scripts for TCN-GAN. Check code for more details

Table 2. Contributions of team members.

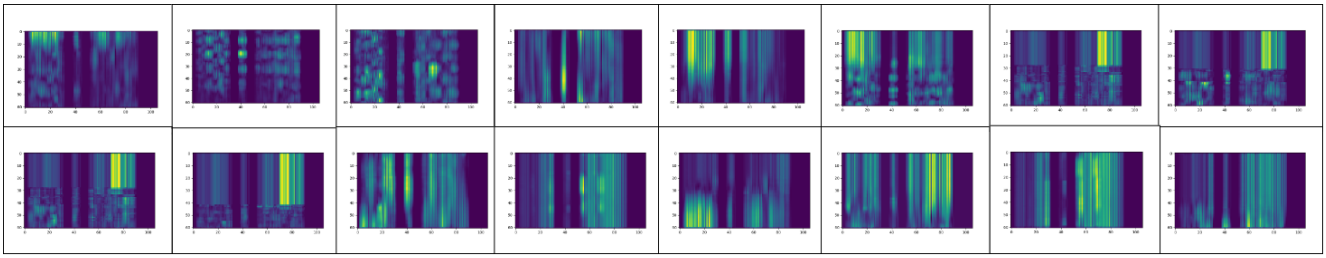

Figure 4. Weights of the long-term encoder for CSS visualized: The x-axis is joint angles and the y-axis is temporally stacked sequences. Due to the rectangular kernel for convolution, joints that are spatially distant and sequences that are temporally close are involved in the final prediction. The convolutional networks progressively pay more attention to previous time instances as weights get brighter in the lower part of the y-axis at later times

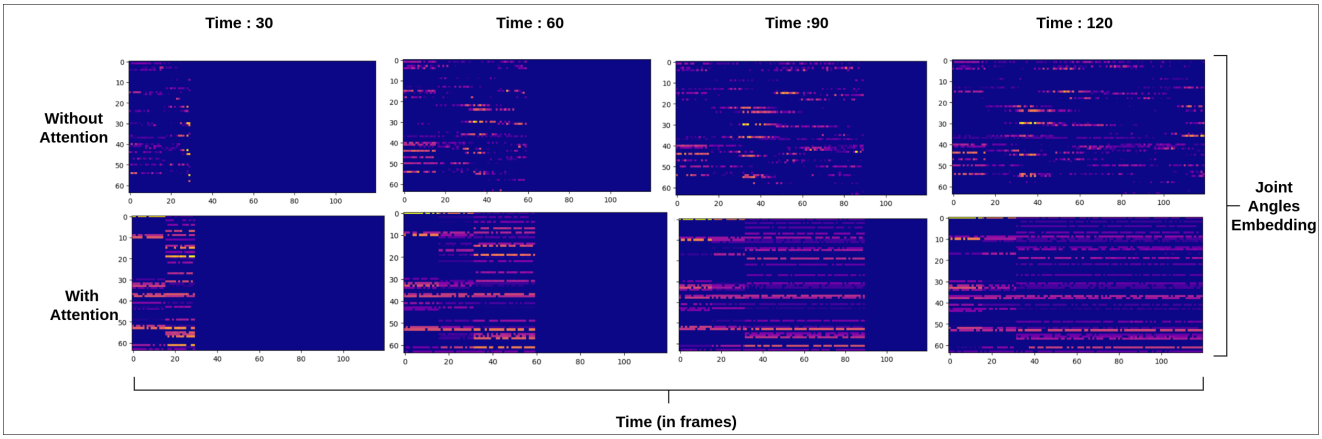

Figure 5. Weights of the TCN architecture with and without attention are visualised. We can see that the network never peeks into the future because of temporal masking. TCN with attention learns information about specific joint angles that affect the future poses and hence the weights are more concentrated in contrast to TCN without attention.

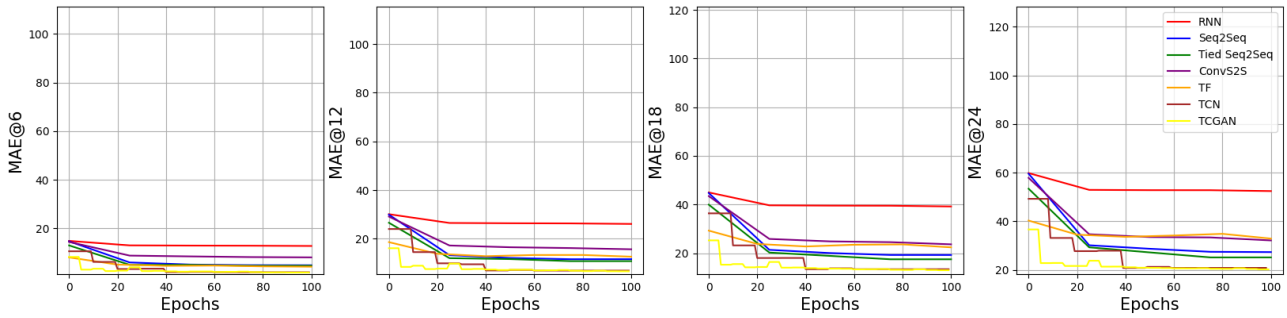

Figure 6. MAE values for different model Architectures

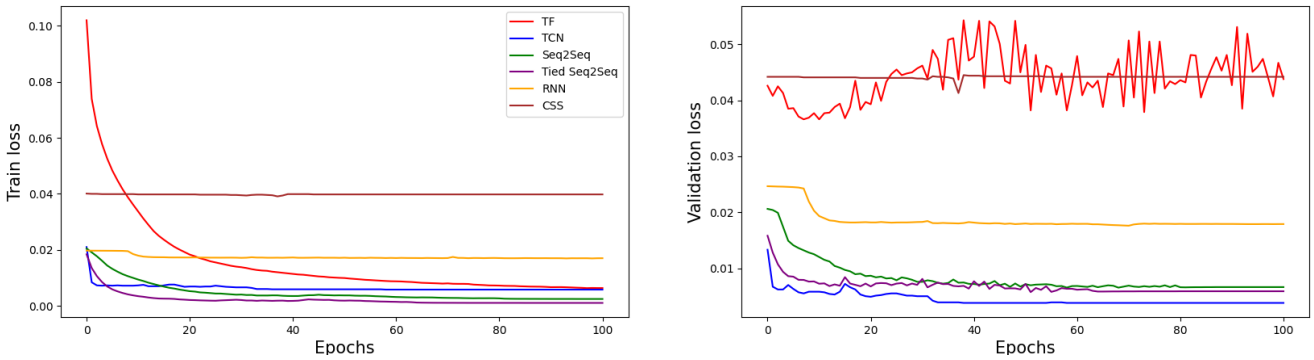

Figure 7. Losses of different model architectures trained over 100 epochs

Model	Optimizer	Hidden Dim	Training Loss	Val loss	MAE@6	MAE@12	MAE@18	MAE@24
CSS-128	Adam	128	9.15E-03	2.12E-02	8.4702	16.3423	24.6866	33.4429
CSS-256	Adam	256	4.98E-03	1.94E-02	8.2684	16.0081	24.1368	32.6177
CSS-512	Adam	512	**7.83E-03**	2.22E-02	8.7596	16.5657	24.7893	33.4861
CSS-256	SGD	256	7.95E-03	**2.05E-02**	**8.1264**	**15.6363**	**23.6327**	**32.1456**
CSS-NLE	Adam	256	3.98E-02	4.42E-02	14.7198	29.6122	44.3474	59.1356
CSS-NSE	Adam	256	1.08E-02	2.19E-02	8.4779	16.3185	24.6078	33.3461

Table 3. Results of the Convolutional Sequence-to-Sequence Model with the best results have been highlighted in bold. CSS-NLE and CSS-NCE are the models without the Long-term encoder and short-term encoder respectively

Model	Num. Params	GPU Memory	Hidden Dim.	Training Loss	Val. loss	MAE @6	MAE @12	MAE @18	MAE @24
TCN-NA	1,471,928	2.0 GB	64	4.80E-03	4.20E-03	2.5318	7.4823	14.0253	21.5182
TCN-NR	1,471,928	2.0 GB	64	2.08E-03	1.98E-03	15.0200	30.2650	45.2600	60.233
GAN-NHS	1,206,369	1.8 GB	64	4.88	4.933E-03	4.329	9.828	16.498	24.311

Table 4. Comparative results of TCN and TCGAN architectures TCN-NA Temporal Convolutional Network with no Attention. TCN-NR Temporal Convolutional Network with no residual connections between blocks. GAN-NHS (No Hinge loss and Sigmoid))

Model	Optimizer	Num. Layers	Num. Heads	Hidden Dim	Training Loss	Val loss	MAE@6	MAE@12	MAE@18	MAE@24
TF-128	Adam	1	4	128	1.08E-01	3.72E-02	5.6048	14.3658	23.6565	33.1254
TF-256	Adam	1	4	256	**5.44E-02**	5.17E-02	4.8089	13.9666	24.5204	35.7751
TF-512	Adam	1	4	512	5.97E-02	**4.38E-02**	**4.2841**	**12.5953**	**22.3869**	**32.8969**
TF-512	SGD	1	4	512	1.58E-02	5.11E-02	7.2788	17.5394	28.5313	39.8764
TF-512	Adam	2	4	512	7.86E-02	4.92E-02	6.2024	16.2695	27.4354	38.9860
TF-512	Adam	1	2	512	1.86E-01	1.22E-01	5.2564	17.9348	28.7984	39.9487

Table 5. Results of the Transformers with the best results have been highlighted in bold

Model	GPU Memory	GFLOPS	MAE @6	MAE @12	MAE @18	MAE @24
RNN	1.35 GB	0.15652	12.8180	26.0632	39.1999	52.4308
Seq2Seq	1.10 GB	0.14320	4.8862	11.5677	19.2466	27.3418
Tied-Seq2Seq	1.20 GB	0.89300	4.6842	10.7567	17.4765	25.1659
TF	3.16 GB	6.3000	4.2841	12.5953	22.3869	32.8969
CSS	2.40 GB	**1.71552**	8.1264	15.6363	23.6327	32.1456
TCN-A	2.40 GB	**0.22976**	2.0112	6.8700	13.3982	20.8592
GAN-WHR	1.8 GB	**0.17655**	1.9010	6.5865	12.9655	20.2525

Table 6. Shows the number of floating point operations (GFLOPS) required for each model with the transformer requiring the highest. Our proposed architectures require very small FLOPS to produce a better MAE on the validation dataset. Our proposed networks are highlighted in bold