Kakashi: Generating Dance Choreography from Arbitrary Music Samples

Rudhra Raveendran
Boston University
rooday@bu.edu

Margrit Betke
Boston University
betke@bu.edu

Abstract—This paper presents the preliminary results of Kakashi, a sequence-to-sequence LSTM recurrent neural network aiming to generate dance choreography from arbitrary music samples. Also a method to cheaply generate a dataset of audio features and pose keypoints is outlined, as no public dataset of music mapped to human movement exists at this time. This dataset is also publicly available to assist in future dance synthesis works. As dance is a qualitative activity, the bulk of the work presented pertains to assessing dance outputs (a series of poses) in a quantitative matter that captures the complexities of the art. Initial results show that training samples can be learned effectively, but inference on novel samples fails to produce good output. Included are plans to improve Kakashi with the hopes of reaching a fully generalizable model.

1 INTRODUCTION

Generating realistic-looking human movement is a challenging problem that currently has no satisfactory solutions. Generating artistic works of similar quality to human creations is a similarly difficult problem, with no general method to quantitatively score outputs. Therefore, synthesizing dance — an artistic sequence of human poses — that could pass for a human's movements is a task that combines the difficulty of both domains. Solving the task of dance synthesis could greatly benefit animation and related fields that deal with virtual art, as computer generated movement would be far cheaper and less time-consuming than hand-crafted or motion captured animations, hence the motivation for this project.

Kakashi attempts to provide a solution through sequence-to-sequence learning, a method commonly employed for natural language translation [3, 13]. Much like natural language translation, dance choreography generation can be modelled as transforming an input sequence (audio data) into a feature vector and using those features to generate an output sequence (human poses).

However, unlike natural language translation, dance generation is much less studied and consequently there does not exist a public dataset of human poses mapped to audio, much less dance mapped to music. This proved to be a significant challenge, as all machine learning models are highly dependent on the data used to train them, and this project did not have the resources to fund the generation of a high quality dataset.

Furthermore, with dance being an art form, there are no empirical formulas for measuring the accuracy or quality of Kakashi's output. As a result, the majority of this paper explores different attempts to discover a function to train a generalizable model.

2 RELATED WORKS

While dance generation is not an incredibly popular topic in machine learning, there are still various works relating to gesture and movement synthesis, as well as a few that do focus specifically on dance.

In the realm of motion synthesis, Chiu and Marsella presented a hierarchical factored conditional restricted Boltzmann machine (HFCRBM) model that successfully learned the mapping of conversational gestures to speech prosody [2]. Similarly, Hofer showed that Hidden Markov Models can synthesize head motion from speech input [7]. Both papers prove the feasibility of (albeit specific) human motion generation from audio input.

Focusing on dance synthesis, Li et al. devised a system in which specific poses were represented as "motion textons," as well as a texton distribution that modelled the likelihood of each texton switching to another [9]. Crnkovic-Friis presented a LSTM RNN filtered through a Mixture Density Network to prevent stagnating output, achieving similar results to Li et al. [4]. While both systems output convincing choreography, they were trained on corpuses composed of a single dancer's movement, without any audio data as input.

Moving onto works that do map *audio input to dance output*, Donahue et al. presented *Dance Dance Convolution*, a 2-stage model that generates step charts for the popular arcade game *Dance Dance Revolution* [5]. Recurrent and convolutional neural networks were used to determine where in a song a step should be placed, and a generative LSTM RNN was used to select the step type for each placement. While step chart generation is a much simpler task than full movement synthesis, this paper still proves the feasibility of determining the correct timings and types of movement for a given audio input.

Most relevant to Kakashi however is GrooveNet, presented by Alemi et al. Similar to Chiu and Marsella's model, GrooveNet utilizes Factored Conditional Restricted Boltzmann Machines (FCRBM) to generate convincing human movements from audio inputs [1]. GrooveNet is special however in that it generates dance choreography from music, by extracting a dense 84 dimension feature set from

audio input and training the FCRBM with a combination of audio features and N recently generated poses.

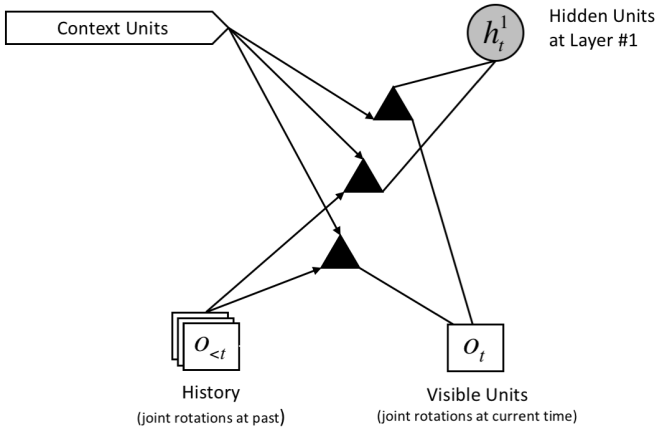

Figure 1. Diagram of an FCRBM used in GrooveNet (audio features would be context units and history would be past pose outputs)

GrooveNet seemed to be the most promising model to study for the development of Kakashi, yet it is not without limitations. Highly specific training data is an issue common among dance generation projects, and GrooveNet was not special in this case. Its custom dataset was composed of four dances to four different songs, danced by a single dancer. Incidentally, while GrooveNet can produce convincing choreography, it can only do so when synthesizing on the songs used in training; novel inputs produce nonsensical movement.

Based on these previous works, a LSTM RNN model trained on audio input to dance output was chosen for the initial version of Kakashi. While other works used iterative RBMs or other complex architectures, Kakashi uses an LSTM due to an assumption about the input data: that audio features contain short *and* long term dependencies, similar to human language. While Kakashi also cannot generalize novel inputs at this time, similar results with a simple model are promising, and the system can be improved in the future by combining more of the techniques mentioned in the above works.

3 DATASET GENERATION

Like the aforementioned related works, Kakashi required a custom dataset as no public dataset of dance movement mapped to audio currently exists. Unlike the related works, no resources or funding were available to generate high quality movement data using motion capture. Furthermore, the hope for Kakashi was to succeed in generating output for novel input. The failure of previous models to do so could have been attributed to a lack of diverse training data, so it was clear that a much larger dataset was needed. Thus, a decision was made to sacrifice the quality of the pose information in exchange for a much greater quantity of data points.

To achieve this, data was gathered through dance videos found on YouTube, with the dance poses generated through pose estimation techniques. However, to ensure that the data was not too random, specific criteria were required for a video to be chosen for the dataset. All the videos had to have:

1) only one dancer visible
2) good lighting on the dancer
3) a stable camera that did not shift too often
4) a similar dance and music style to the other videos

To satisfy all these criteria, videos were selected from the official World of Dance YouTube channel [1]. Choosing videos from a single channel ensured (more or less) consistent lighting and camerawork throughout videos, and videos were then hand-selected when only one dancer was visible and style was similar (hip-hop).

Furthermore, using World of Dance as the "ground truth" source outsourced the difficulty of determining subjectively "good" dances to train on. World of Dance is an international competition, therefore anyone who was showcased was guaranteed to be a good dancer. The selected videos — a total of 188 and containing several different dancers — were saved into the *Kakashi-WOD* playlist [2].

After videos were collected, they had to be trimmed to remove unnecessary information, as each video contained a World of Dance introduction and outro, as well as some videos containing non-dance movement. To this end, a pair of start and end times for dancing was manually collected for each video and saved in a file associated with the playlist. Then a custom script was run to download the playlist using youtube-dl and trim each video according to the recorded start/end times using FFmpeg.

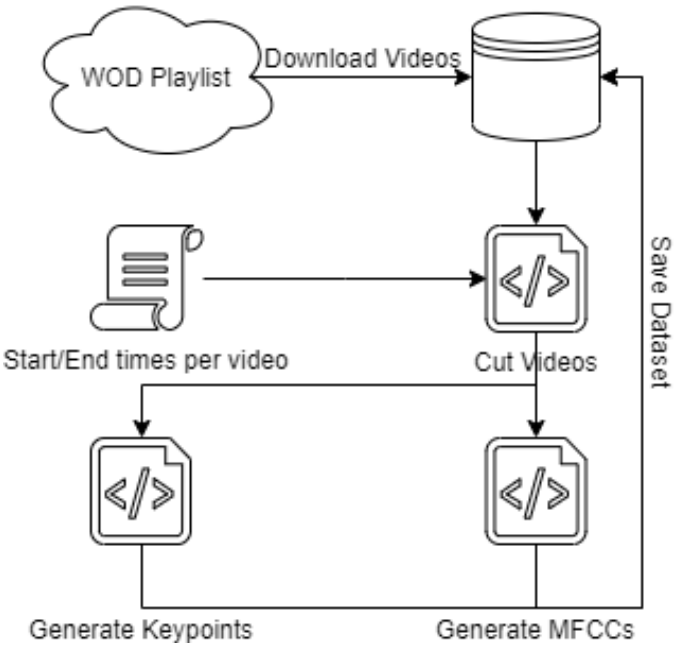

Figure 2. Diagram of Kakashi's dataset generation pipeline

After preprocessing, pose estimation was done using FaceBook's VideoPose3D and Detectron[3] [6, 12] models, as

1. https://www.youtube.com/user/WORLDOFDANCETOUR
2. https://tinyurl.com/kakashi-wod
3. Originally Microsoft Research's Simple Baselines project was used for pose estimation, but was swapped out for VideoPose3D as the latter could accept videos as input.

well as audio feature extraction using LibROSA [10]. The pose estimation models output 17 keypoints (per video frame) in the form of 3D coordinates, saved as NumPy archives for each video. LibROSA was used to extract 20 Mel Frequency Cepstral Coefficients (MFCCs) for the audio accompanying each video frame, also stored in NumPy archives. The MFCCs and keypoints together form the input and output for the Kakashi dataset.[4]

4 MODEL

Based on the related work, it would seem that the leading approaches are to use either iterative Restricted Boltzmann Machines or LSTM RNN architectures. Considering dance as a translation problem of audio sequences to pose sequences, a LSTM RNN architecture was selected, implemented in PyTorch similarly to the system described in Sutskever et al. [13].

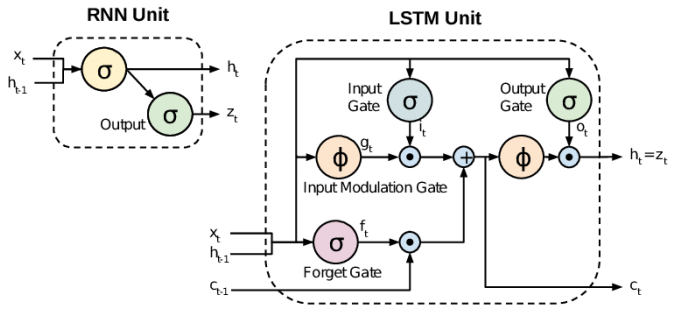

Figure 3. LSTM vs. traditional RNN

Recall the previous assumption that music contains both short and long term dependencies. While songs are composed of repeated patterns, certain embellishments occur sporadically and are interacted with in dance. Under this assumption, it seemed reasonable to build Kakashi on a LSTM RNN. LSTMs also mitigate the problem of vanishing and exploding gradients, which become more of an issue the longer a RNN runs [8]. The songs used as input were much longer sequences than typically seen in natural language translation, hence the motivation for a LSTM.

Specifically, Kakashi is an encoder-decoder model composed of 2 LSTM networks, one to encode the input audio into a hidden state and cell state, and another to decode those states and generate an output sequence of poses. Both the encoder and decoder contain 2-layer LSTMs with a dropout layer to mitigate overfitting.

Audio input is fed to Kakashi in one go, with shape: [sequence length, batch size, feature length]. Pose output is generated one time step at a time (at a frame rate of 24 FPS), with the previously output pose used as input for the next. This is similar to the iterative mechanism used in RCB architectures like Chiu and Marsella and GrooveNet, but this architecture has the added benefit of the entire input's context when generating output.

In an encoder/decoder model, the entirety of the input sequence is encoded into a context vector, allowing the decoder to have an understanding of the whole input sequence when generating output. This was an important factor in choosing to use this architecture for Kakashi, as the entire audio context seemed superior to one time step's worth of audio features at a time.

A caveat to the LSTM is that outputting shapes of [sequence length, batch size, 17, 3] were not allowed, so the model was trained to output [sequence length, batch size, 51], which was simply reshaped accordingly when computing loss.

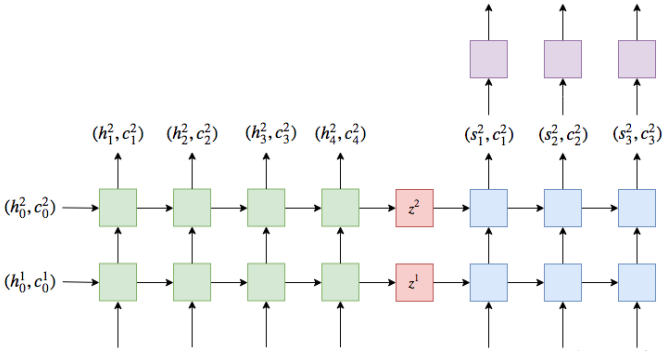

Figure 4. Kakashi's model architecture diagram

Figure 4 shows the 2-Layer LSTM architecture, with green nodes representing the encoder taking in audio input, red nodes representing the context vectors, blue nodes representing the decoder, and purple nodes as final output poses.

5 TRAINING

For training the model, the majority of experiments were run on a 2-layer configuration with a hidden dimensionality of 512, dropout rates of 0.5 for both the encoder and decoder, and an SGD optimizer with learning rate of 0.005 and momentum of 0.9. The configured model was trained for 100 epochs with a gradient clip value of 1.

The dataset went through a final preprocessing step before training where the audio features and pose sequences were cut to a sequence length of 720 frames (30 seconds @ 24 FPS) and randomly batched into groups of 5. This resulted in input data with a shape of [720, 5, 20] and target data with a shape of [720, 5, 17, 3]. The resulting set of batches was then split into training, validation, and test sets at a rate of 0.7, 0.2, and 0.1 respectively.

As the primary problem was attempting to quantitatively assess the model's output, the model architecture and configuration remained constant throughout training rounds, save for the loss function. The underlying assumption was that standard loss functions fail to capture the complexity of dance, which was supported by initial runs of the model.

Early outputs comprised of a single pose repeated for the entire sequence length (with some noise). After further examination, it was clear that in the task of learning which pose to output from a previous input pose, the LSTM was learning to output a pose similar to what was just input. This makes sense, as frame to frame pose within a video changes very slightly. With this discovery, the rest of experimentation composed of trying various PyTorch loss functions, in an

4. Instructions for reproducing the dataset can be found in the Kakashi GitHub repository: https://github.com/ROODAY/Kakashi

attempt to find a function that adequately trained the model to learn not just anatomically correct pose, but how poses transition from one to the next to become choreography.

6 EXPERIMENTAL RESULTS

Various experiments were run on Kakashi, where the independent variable was the loss function. However, unlike other machine learning tasks, an obvious dependent variable like accuracy does not exist for dance. Indeed, desired output is subjectively measured as "good," whereas high accuracy/low loss would mean that it is simply copying what it was trained on.

Instead, experiments were evaluated based on the perceived level of overfitting as epochs progressed. This was done by comparing training loss to validation loss over time: training loss is always expected to go down, but if validation loss did not it would indicate that the model was overfitting and unable to generalize novel input.

The loss functions used in the experiments (in order) are described below. Each experiment generated pretrained models that were used to infer choreography on the same novel music sample[5]. The rendered choreographies are hosted on Google Drive as supplementary material, and are referenced throughout this section.[6]

6.1 Mean absolute percentage error (MAPE)

MAPE is designed to calculate percent error between actual and predicted values, which makes error clearer when the scale of the values changes significantly i.e. the difference between 5 and 6 is much more significant than between 100 and 101. The formula for calculating MAPE is as follows:

$$Loss = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

where A_t is the target value and F_t is the predicted value. The motivation for MAPE was due to the fact that the poses output by VideoPose3D were scaled such that all coordinates were less than one. Therefore, traditional loss functions like MSE would report very low loss (recall $x > x^2$ for $x < 1$), leading the model to believe it was performing well when it was not.

The results for MAPE are shown in Figure 5, where it is clear that overfitting is present. However, it is also evident that validation loss is approaching training loss, as the magnitude of spikes decreases over epochs. Unfortunately, viewing the associated renders shows nonsensical output. Note the Y-axis values and recall that MAPE outputs *percentage* error.

5. https://soundcloud.com/leopop/swet
6. https://tinyurl.com/kakashi-renders

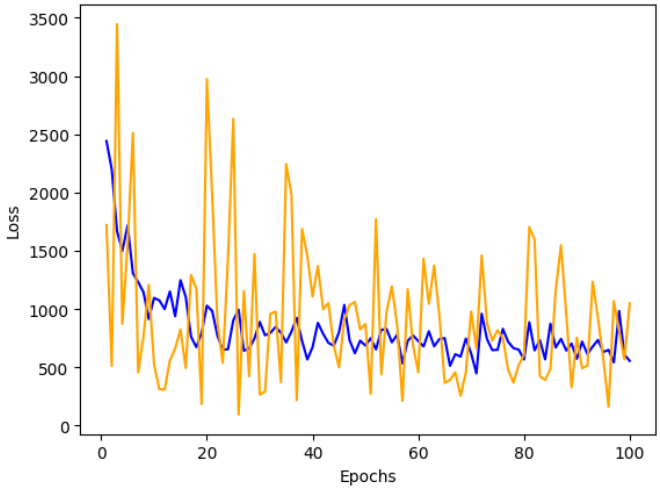

Figure 5. Training (blue) vs. Validation (orange) Loss for MAPE

6.2 Relative Percent Difference (RPD)

RPD, a more general version of percent error, was chosen for a similar reason to MAPE, but the denominator is the average of the target and predicted values. This difference smoothens the error, as the denominator will be closer to the predicted value than in MAPE. RPD is calculated as follows[7]:

$$Loss = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{\frac{(|A_t|+|F_t|)}{2}}\right|$$

The error values from RPD are much smaller than MAPE (due to its formulation), as can be seen in the scale Y-axis of the Training vs. Validation Loss graph (Figure 6). The accompanying rendered output is better than MAPE, as a clear human figure can be seen, but the anatomy of human pose is not learned properly as the coordinate for the hip is projected far off the skeleton. Furthermore, the output sequence is stagnant i.e. a single pose with noise as opposed to choreography.

7. In RPD the denominator is any function of A_t and F_t, such as max, min, etc. For purposes of this experiment only the average was tested.

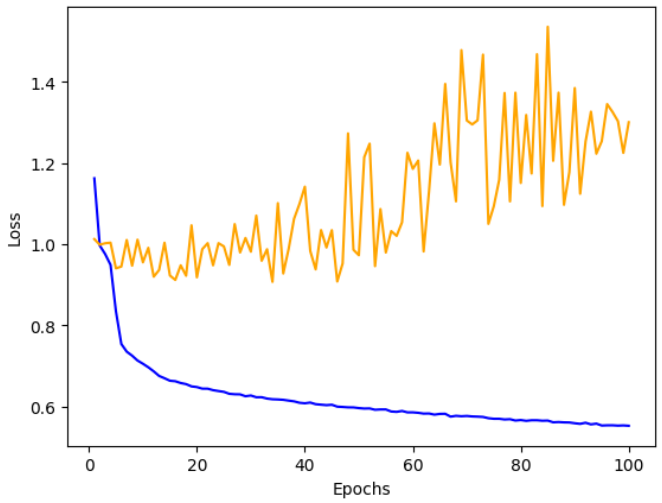

Figure 6. Training (blue) vs. Validation (orange) Loss for RPD

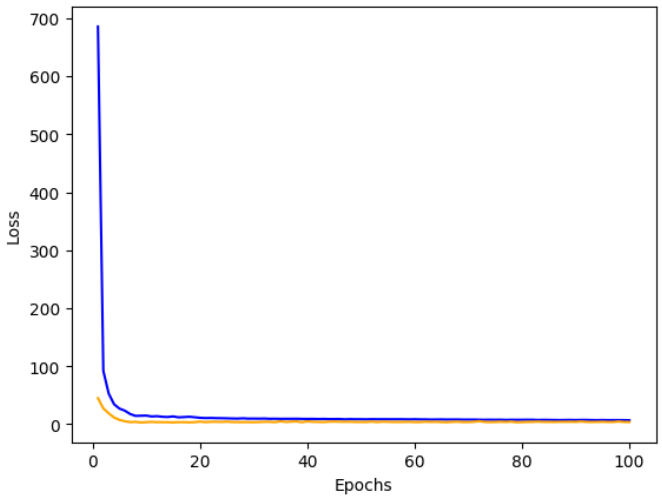

Figure 7. Training (blue) vs. Validation (orange) Loss for MAPE on Velocity

6.3 MAPE on Velocity

This experiment uses the same MAPE calculation as the first experiment, however instead of calculating error on the coordinates themselves, it is calculated on the magnitude of pose velocities. Converting a series of poses of shape [sequence length, batch size, 17, 3] to velocities of shape [sequence length, batch size, 17] was done by taking the difference between each pair of poses ($p_1 - p_0$, $p_2 - p_1$, ...), squaring the elements of the resultant matrix, summing along the third dimension, and taking the square root. This can be seen as a matrix version of Euclidean Distance:

$$D = \sqrt{(x_A - x_F)^2 + (y_A - y_F)^2 + (z_A - z_F)^2}$$

where A represents target coordinates and F represents predicted coordinates.

This is done for both the output and target matrices to calculate their velocities, and MAPE is calculated on those velocities. Note that while $velocity = \frac{distance}{time}$, time is not included in this calculation. That is because the time is the same for every velocity, $\frac{1}{24}$ of a second, which would result in scaling every element in the output and target by 24, which would not change the percent error.

Immediately the results in Figure 7 are suspect, as validation loss should not possibly be so low, especially at such early epochs. The rendered output for the model generated from this experiment confirms this, as the animation shows a human-looking pose, but it is very small compared to the data it was trained on. This indicates that this method learns the anatomy of human pose but not the proper scale.

6.4 RPD on Velocity

This experiment is very similar to the last, as velocity is calculated the same way. The only difference is RPD is used in place of MAPE for calculating the velocity loss.

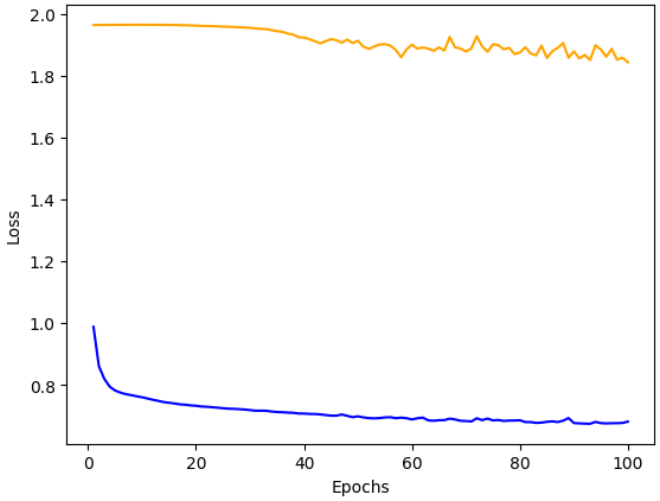

Figure 8. Training (blue) vs. Validation (orange) Loss for RPD on Velocity

Figure 8 shows that while training loss drops at the expected rate, validation loss is resistant to change. Unlike RPD on the coordinates, the rendered output for this experiment did not learn pose anatomy, generating a blob of coordinates in the center of the bounding box.

6.5 Euclidean Distance

Here error is calculated simply as the summed Euclidean distance of target poses from output poses. It is the same calculation as used in experiments 3 and 4 when calculating velocity, however instead of calculating distance of pairs within the target sequence and output sequence, distance is calculated between corresponding target and output poses i.e. $dist(trg_0, out_0) + dist(trg_1, out_1) + ... + dist(trg_n, out_n)$.

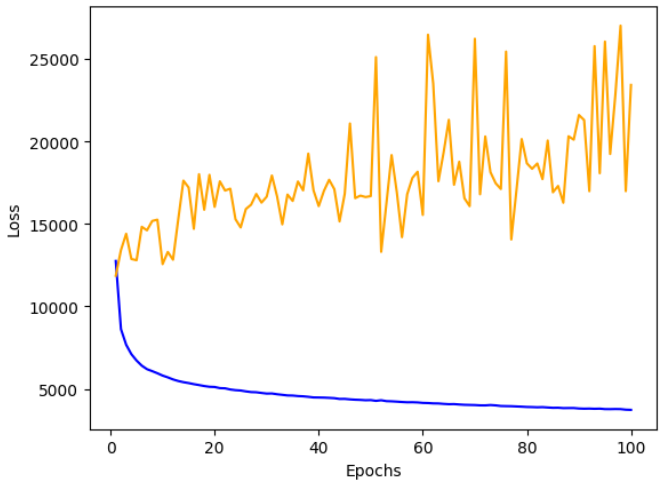

Figure 9. Training (blue) vs. Validation (orange) Loss for Euclidean Distance

The training vs. validation loss in Figure 9 is reminiscent of experiment 2, where RPD on coordinates was used. Viewing the rendered output of the model from this experiment shows a coherent pose with proper anatomy, and although the pose is fairly stagnant, the figure makes small movements synchronized to the beat, suggesting that Euclidean Distance allows the model to associate beat to movement.

However, reconciling this promising result with the Training vs. Validation Loss graph is difficult, indicating that perhaps overfitting is not a good metric to judge the quality of dance synthesis (or rather, high validation loss may not be the right metric to interpret overfitting).

6.6 Sum Root Squared Error (SRSE) on Velocity

This loss function calculates the velocities of the target and output sequences like previous experiments, but instead of taking a percent error, it simply sums the root squared error i.e.

$$Loss = \sum_{t=1}^{n} \sqrt{(A_t - F_t)^2}$$

The reasoning for using velocity was to attempt to break out of stagnant output, as it was believed to be a result of the model not learning how pose moves from one to the next.

Unfortunately, this loss function proved to be unstable, as training failed in epoch 4 due to NaN values showing up in the output (this causes training to halt). These values are likely due to some cases of $A_t - F_t \approx 0$, which aren't valid for the model.

Due to this, a graph of Training vs. Validation Loss was not generated for experiment 6. However, training saves a pretrained model whenever a new best validation loss is achieved, so rendered output was available for this experiment. However, since only 3 epochs were completed, the output is fairly meaningless, and viewing the render confirms this as it is a noisy blob of coordinates centered in the bounding box.

6.7 Ensemble Loss: (Euclidean Distance + SRSE Velocity)

In this experiment, the previous two loss functions were combined, simply by calling each and adding their results. This was done in the hopes that the model would minimize both forms of loss, leading to output that was both anatomically correct and moved like a dance sequence.

However, since all the experiments were run in parallel, the instability of the SRSE Velocity loss function was not caught until both experiment 6 and this experiment failed due to NaN output values. The hope for the Ensemble function was that the Euclidean Distance loss would teach the model pose anatomy and which poses correlate to certain audio features, while the SRSE Velocity loss would teach it to move from one pose to the next.

Due to the SRSE Velocity loss function's instability, experiment 7 failed on epoch 2 (and consequently no Training vs. Validation Loss graph was generated). The rendered output from this experiment's model shows an anatomically correct pose, but the sequence is stagnant and noisy. Furthermore, at certain points the noise reaches levels where the pose undergoes humanly impossible transformations.

6.8 Mean Absolute Error (MAE/L1 Loss)

These next few experiments used common loss functions built into PyTorch to serve as baselines to compare the custom loss functions against. Recall the formula for MAE:

$$Loss = \frac{1}{n} \sum_{t=1}^{n} |A_t - F_t|$$

Looking to the graph in Figure 10, it is clear that just like in previous experiments, it is difficult to generalize the trained model on the validation set. Viewing the associated render for this experiment shows an anatomically correct figure but no significant motion throughout the sequence, save for heavy noise.

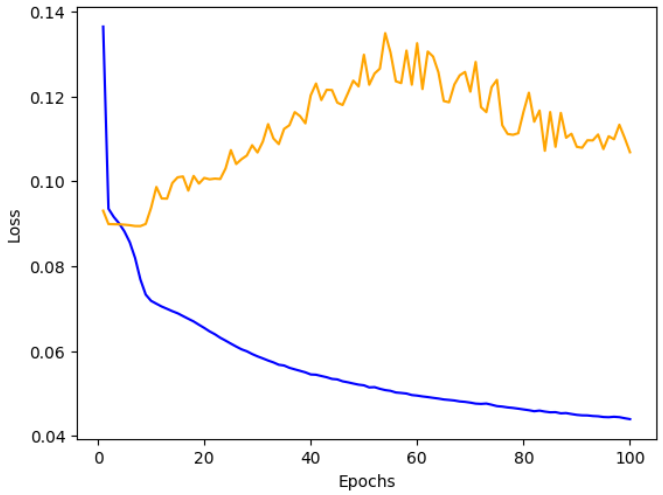

Figure 10. Training (blue) vs. Validation (orange) Loss for MAE

Interestingly, in this experiment there seems to be three distinct stages within training, as evidenced by the different rates of change in training loss (epochs 1-3, 3-10, 10-100). Furthermore, validation loss appears to initially drop, stagnate, then climb during training.

6.9 Mean Squared Error (MSE/L2 Loss)

Here another builtin loss function is used, calculated via the following equation:

$$Loss = \frac{1}{n}\sum_{t=1}^{n}(A_t - F_t)^2$$

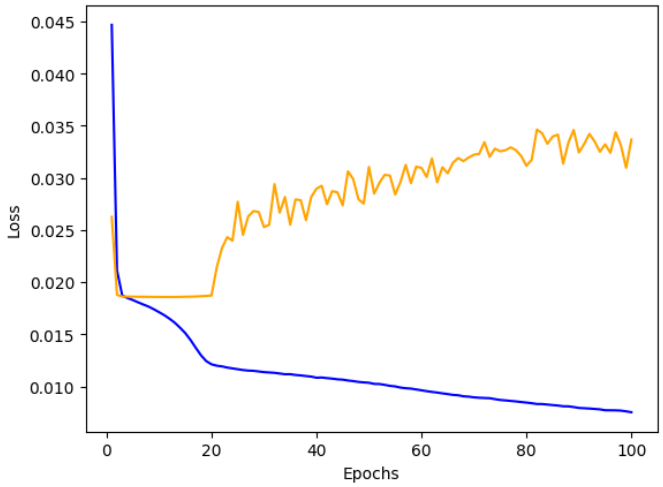

Figure 11. Training (blue) vs. Validation (orange) Loss for MSE

This experiment had similar results to MAE/L1, however the trends in Training vs. Validation Loss are more pronounced. The slower drop in training loss after the initial drop is stretched over nearly twice as many epochs, with the same behavior in the period of stagnant validation loss. The rendered choreography for this experiment had similar properties to that of L1 Loss, however it was far noisier (which is reasonable considering the errors are now squared).

6.10 Smooth L1 Loss (Huber Loss)

This last baseline experiment uses Huber Loss, a variation that combines the sensitivity of L2 loss and the robustness of L1 loss. It is a piecewise function, defined as follows:

$$Loss = \begin{cases} \frac{1}{2}a^2 & |a| \leq \delta \\ \delta(|a| - \frac{1}{2}\delta) & otherwise \end{cases}$$

where a is the residuals at each time step, defined earlier in the paper as $A_t - F_t$. Looking at Figure 12, the period of stagnant validation loss doubles in length again. With this graph it becomes clear that the stagnant validation loss and the change in the derivative of training loss are correlated, as the appear to begin and end at approximately the same epochs.

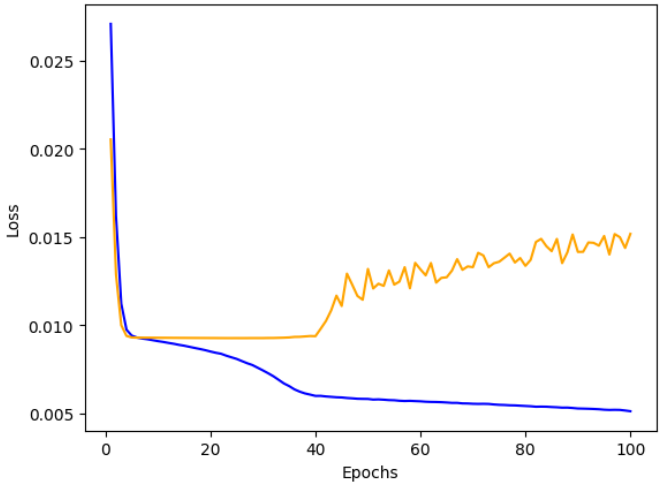

Figure 12. Training (blue) vs. Validation (orange) Loss for Huber Loss

Although Huber loss combines some favorable characteristics of both L1 and L2 loss, the output does not fare much better than either. The render created from this experiment's model is similarly noisy, fluctuating in scale as well throughout the sequence.

7 DISCUSSION

Throughout each of the experiments, save for the outlier of experiment 3 (MAPE on Velocity), validation loss tended to stay high relative to training loss, indicating that in all cases the models were overfitting on training data. However, it is important to note that in the case of dance synthesis, this may not necessarily be undesirable, as output sequences are not judged on their similarity to existing dance but via a subjective measure of quality.

Regardless, viewing the rendered animations that resulted from each of the experiments show unimpressive output, with stagnant poses and heavy noise. In the best case — experiment 5 (Euclidean Distance) — the model adequately learned pose anatomy and contains far less noise than other runs. However, the sequence doesn't contain different poses that would constitute a choreography; it simply shows a figure slightly swaying.[8]

8. The animation did seem to "pop" on the beat, however this could be attributed to noise.

Based on these results, Kakashi encounters the same pitfalls as related works such as GrooveNet: generating dance sequences from novel inputs remains difficult. The task is essentially quantifying and applying a set of rules to creativity i.e. deciding what poses to use for given audio and how to transition between them. Models like vanilla LSTMs simply cannot learn that behavior, as the difference between training and novel data is too great.

8 CONCLUSION

Although this first iteration of Kakashi failed to generalize dance synthesis from novel input, an evaluation of the model and how it compares to related works reveals three key points.

Firstly, a simple encoder/decoder architecture is not sufficient to capture the complexities of dance synthesis, especially when audio input has only 20 features per time step. While modelling the problem in the terms of language translation made it easier to conceptualize, this simplification ignored techniques like Mixture Density Networks or more complicated architectures.

Second, measuring model performance with training vs. validation loss is not necessarily a good metric, as a model could output subjectively good sequences but still have low validation loss. A solution to this is not yet clear and requires further exploration.

Lastly, generating output from a model trained on pose estimated input has inherent limitations. VideoPose3D anchors the estimated pose to the skeleton's hip keypoint, meaning that information like crouching will be represented as legs magically floating up [12]. Furthermore, only 17 keypoints means that a lot of subtle motions inherent in dance are not captured, which means the output will also not include those motions.

Despite this, Kakashi does provide a significant contribution in regards to dataset generation. While no high quality dataset of motion captured dance synchronized to music exists, the dataset created for Kakashi is publicly available[9], and the method to generate similar datasets with different sources is described. This lowers the barrier of entry for new works in the field, and as pose estimation technology improves, the quality of Kakashi-esque datasets will approach that of motion capture.

9 FUTURE WORK

There are a variety of improvements that can be made to Kakashi in the attempts to synthesize choreography from novel input, many of which come from incorporating different aspects of related works.

Recalling the first point from the conclusion, using a richer audio input could allow the model to better learn the association between music and pose. MFCCs are limited in the fact that they represent only timbral features [1]. Using features that describe more aspects of an input sequence could lead to a more general model.

In addition to improving the audio features, acquiring a higher quality dataset with more keypoints could help future models better learn proper pose anatomy (preventing issues like the projected hip keypoint in experiment 2). Either by recording a motion capture dataset, or using more sophisticated pose estimation techniques, a better keypoint dataset would allow for more detailed poses to be learned and generated, better approximating human movement.

The model can also be improved through the use of different architectures. Even in the realm of sequence to sequence models there is variety, with attention models, transformer models, even CNN architectures. Kakashi's LSTM, which was prone to stagnant output, could perhaps benefit from a Mixture Density Network as proposed in Crnkovic-Friis [4] .

Completely different architectures are also planned to be explored, such as Restricted Boltzmann Machines and texton distributions. Perhaps an ensemble architecture, where different models learn different aspects of the task, can utilize these various systems to achieve a better overall result.

By exploring the above possibilities, implementing all or a subset of them, the hope is that a better iteration of Kakashi can be developed, one that can successfully synthesize dance choreography to novel audio input. Further exploration is expected to be completed within the next semester, at which point this paper will be updated with more recent results.

9. https://tinyurl.com/kakashi-dataset

REFERENCES

[1] Omid Alemi, Jules Françoise, and Philippe Pasquier. "GrooveNet: Real-Time Music-Driven Dance Movement Generation using Artificial Neural Networks". In: (Aug. 2017).

[2] Chung-Cheng Chiu and Stacy Marsella. "How to Train Your Avatar: A Data Driven Approach to Gesture Generation". In: (Sept. 2011), pp. 127–140. DOI: 10.1007/978-3-642-23974-8_14.

[3] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). arXiv: 1406.1078. URL: http://arxiv.org/abs/1406.1078.

[4] Luka Crnkovic-Friis and Louise Crnkovic-Friis. "Generative Choreography using Deep Learning". In: *CoRR* abs/1605.06921 (2016). arXiv: 1605.06921. URL: http://arxiv.org/abs/1605.06921.

[5] Chris Donahue, Zachary C. Lipton, and Julian J. McAuley. "Dance Dance Convolution". In: *CoRR* abs/1703.06891 (2017). arXiv: 1703.06891. URL: http://arxiv.org/abs/1703.06891.

[6] Ross Girshick et al. *Detectron*. https://github.com/facebookresearch/detectron. 2018.

[7] Gregor Hofer. "Speech-driven animation using multimodal hidden Markov models". In: (Jan. 2010).

[8] J. F. Kolen and S. C. Kremer. "Gradient Flow in Recurrent Nets: The Difficulty of Learning LongTerm Dependencies". In: *A Field Guide to Dynamical Recurrent Networks*. IEEE, 2001, pp. 237–243. DOI: 10.1109/9780470544037.ch14. URL: https://ieeexplore.ieee.org/document/5264952.

[9] Yan Li, Tianshu Wang, and Heung-Yeung Shum. "Motion Texture: A Two-Level Statistical Model for Character Motion Synthesis". In: *ACM Transactions on Graphics* 21 (July 2002), pp. 465–472. DOI: 10.1145/566570.566604.

[10] Brian McFee et al. "librosa: Audio and Music Signal Analysis in Python". In: (Jan. 2015), pp. 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[11] Dushyant Mehta et al. "Monocular 3D Human Pose Estimation Using Transfer Learning and Improved CNN Supervision". In: *CoRR* abs/1611.09813 (2016). arXiv: 1611.09813. URL: http://arxiv.org/abs/1611.09813.

[12] Dario Pavllo et al. "3D human pose estimation in video with temporal convolutions and semi-supervised training". In: (2019).

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. "Sequence to Sequence Learning with Neural Networks". In: *CoRR* abs/1409.3215 (2014). arXiv: 1409.3215. URL: http://arxiv.org/abs/1409.3215.

[14] Bin Xiao, Haiping Wu, and Yichen Wei. "Simple Baselines for Human Pose Estimation and Tracking". In: *CoRR* abs/1804.06208 (2018). arXiv: 1804.06208. URL: http://arxiv.org/abs/1804.06208.