ASDF-RNN: An Investigation Into Raw Audio Synthesis From Dance Features Using a Probabilistic Deep Encoder-RNN-Decoder Topology

Sophus Bénée Olsen
Sound and Music Computing
Technical Faculty of IT and Design
Aalborg University
sbol13@student.aau.dk

Daniel Michael Woodward
Sound and Music Computing
Technical Faculty of IT and Design
Aalborg University
dwoodw19@student.aau.dk

Juan Alonso Moreno
Sound and Music Computing
Technical Faculty of IT and Design
Aalborg University
jalons19@student.aau.dk

ABSTRACT

This paper describes ASDF-RNN (**A**udio **S**ynthesis from **D**ance **F**eatures), a neural network architecture for synthesising raw audio directly from motion features extracted with a computational model based on Laban Movement Analysis (LMA). The system is composed of a deep recurrent Neural Network that maps LMA features to a mixed gaussian distribution of encoded mel-spectrograms representing classical music. Audio is reconstructed by sampling from the mixed gaussian distribution and decompressing the mel-spectrogram with a dedicated decoder network trained on classical music. The results from this investigation are promising, however the generated audio retains very little resemblance to its original source. Additionally, an informal qualitative assessment made it clear that audio generated by ASDF-RNN from widely different motion had no perceivable differences in melody or harmonic content.

1. INTRODUCTION

Thanks to the apparition of tools that "hide" the most complex mathematical part, libraries that facilitate the use of standard models and the possibility of training the models on servers with great computing power, the discipline of Machine Learning (ML) has made the leap to the public. Among this audience is the artistic community and creative technologists, who have incorporated tools such as RunwayML [1], Magenta [2] or Mimic [3] into their creative process. The aim of this project is to create a musical soundtrack for a video of a single person dancing. There are currently several ML tools which generate dance moves [4, 5], tools to sonify the dance movements [6] or movement in general [7], but there is little research into generating music for a set of dance moves using ML.

Our motivation emerges from the idea that music can be generated directly from low-level or symbolic features extracted from a dance performance. Thus, the problem we seek to circumvent is the arbitrary mapping from gesture to audio that seems to be current state-of-the-art in the extant literature. With this in mind, the authors acknowledge that such approaches are desirable for composers or musicians who would use systems such as the ones developed by [8–10] as extended instruments.

This paper presents ASDF-RNN, a prototype of a full-body motion analysis system for generating classical music. The aim of this system is to analyse dance that has been choreographed to a specific music genre and subsequently generate new music that retains similarity to the original style.

The rest of the paper is organized as follows. Section 2 presents relevant computational models for full-body motion analysis with emphasis on the use of Laban Movement Analysis. Subsequently, we discuss the state-of-the-art ML models for raw audio generation. Section 3 outlines the implementation process of ASDF-RNN. Here we emphasise the training procedure and workflow of our own proposed architecture. Section 4 presents the results from the model training. Informal qualitative assessments of the generated music are also presented. Section 5 discusses the results, and we provide reflections on the hyper-parameters, training workflow and general improvements, while sections 6 and 7 conclude the paper and provide directions for future development, respectively. All the original data, source code and generated content for this project are available online (see table 1):

Table 1. Code and data repositories

Contents	URL
Source code	https://tinyurl.com/SMC7-source
Multimedia examples	https://tinyurl.com/SMC7-media
Full data	https://tinyurl.com/SMC7-data

**all source code and data is freely available to use*

2. BACKGROUND

2.1 Music generation from dance

Some recent advances in the study of generating music from dance rely on an arbitrary mapping (or no mapping at all) from the performed gestures to a corresponding musical output. In [8], no specific movement style was imposed on the dancer. Key points from the movement were de-

Copyright: © 2020 Sophus Bénée Olsen et al. This is an open-access article distributed under the terms of the Creative Commons Attribution 3.0 Unported License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

tected, and lower level features such as the velocity of the key points between frames were directly mapped to a musical output. Other advances have left the musical mapping to the user of the system. The *EyesWeb* project provides an interface with real-time analysis of full-body movement that composers and choreographers can use to make real-time music and visuals to stage performances [9]. A similar tool was developed by [10], who describes their project as an approach to create a real-time motion analysis engine to be used in multimedia dance performances. Other than [8], it is apparent that projects that focus on dance as the interactive mechanism for driving music or visuals, primarily focus on the analysis and extrapolation of full-body motion. They compute low-level features extracted from motion features, used later in higher-level processes to generate symbolic representations of the performed movements. These symbolic representations are the output, to be used as the basis for mapping either music or visuals [9, 10]. These systems have been used in public performances, for example in [5] where the human dancer tries to imitate the (previously trained) AI movements and the AI tries to imitate the human, engaging in a feedback loop / duet.

2.2 Full-body Motion Analysis

At the lowest level of any motion analysis system, some virtual representation of the body has to be acquired. Camurri et al. [9] used a setup of at least two cameras. The signals were preprocessed using classical image processing techniques and propagated further to the MIL (Matrox Imaging Library), which extracts high-level information about movement. An even more extensive setup was used in [10]. A 16-camera marker based motion capture system was used to extract the individual joints of the body. Generally, Motion Capture (MoCap) systems of various sorts are preferred in the analysis of full-body motion due to the extensive information about each joint they provide [11–13]. MoCap setups are, however, both expensive and very time consuming to prepare [12]. To avoid large camera setups, analog sensor based approaches have also been used with some success [14]. Such approaches (e.g. [14]) attach accelerometers to each desired joint of the body under analysis. Modern depth-cameras such as the Kinect is a great alternative to extensive MoCap or analog sensor setups and has been used in various motion analysis projects [15–17]. The Kinect software performs pose estimation and can output a skeleton where prevalent body joints are represented as positions in 3D real-world coordinates in relation to the camera [18]. When real-time acquisition of the body is not important, for example in the training and development phase of motion analysis specific ML projects, online MoCap databases have often been used [12, 19–21]. The problem with online MoCap databases (and databases in general), is that they might not fit within the specific context desired for a ML project [12]. In these scenarios, models as PoseNet [22] e.g. provided through the RunwayML [1] framework can be used by data scientists and ML researchers as a cheap and easily available method of extracting and estimating poses. This approach does not compete with MoCap, analog sensors or the Kinect in real-time full-body acquisition. PoseNet supplies body poses in 2D pixel coordinates only. However, the enormous flexibility provided due to its availability weigh up the necessary trade-offs. When using existing video materials, PoseNet beats all other alternatives.

2.2.1 Laban Movement Analysis

Laban Movement Analysis (LMA) is a comprehensive model describing human motion. Originally developed as a way to qualitatively describe dance, it has since been extended to comprise human movement of any sort [23]. LMA can be considered as high dimensional model. One of these dimensions, denoted the *effort axis*, breaks down movement into four sub-categories: *weight, time, space and flow* [11, 13, 23]:

1. **Weight:** describes the amount of force involved in a movement. It is further sub-divided into two extremes: strong and light.
2. **Time:** describes the degree of *urgency* in the movement. That is, how much acceleration or deceleration is involved. The two extremes of this category are sudden and sustained.
3. **Space:** is sometimes described as the attitude towards a chosen pathway or how that pathway is approached. It can also be described as the *directness* of the movement, relating to how much attention is paid to the surroundings. The two extremes are direct and indirect.
4. **Flow:** is the degree to which the movement is controlled or released. Also described as the continuity of the movement. The two extremes are free and bound.

Another axis of LMA sometimes considered is the *shape-axis* [11, 13]. The *Shape* descriptor describes the way the body sculpts itself in space, or the changes in relationships of the body parts to one another and their surroundings. Only of one of this axis' sub-categories will be considered, namely the *Shape Directional* descriptor. This describes the shape of a path along which a movement is executed. It also has two extremes, arc-like and spoke-like.

2.2.2 LMA Computational Models

LMA is in its essence a purely qualitative model [23], and the above descriptors and their two extremes comprise one of the highest levels of movement representation and description. In actual LMA, practice certified LMA experts (also referred to as Certified Movement Analysis Experts) derive the motion descriptors through repeated observation of the subject in question [13, 14, 23]. Due to its rigidity in describing motion, several computational models have been developed to approximate aspects of LMA [24]. LMA is usually broken down into analysis of smaller segments of the body such as the head, upper limbs (shoulders, elbows, arms), lower limbs (hips, knees, feet/toes) and the torso [23]. As a result, most computational models estimating LMA factors require motion trajectories from a set

of joints. Hence, MoCap-like data is usually the assumption behind most available models. Every model computes a set of *low-level* features based on the motion trajectories. The more advanced models proceed by using these low-level features to estimate high-level LMA dimensions. Most models stay within the effort-axis described above [13, 14, 17]. For the sake of brevity, this paper will not elaborate on high-level LMA feature extraction.

2.2.3 Low Level LMA Features

In their review of computable descriptors of human motion, Larboulette & Gibet [11] provide formulas for estimating motion descriptors from LMA on the effort- and shape axes, as described in section 2.2.1. The first assumption in their model is that the input data is in the form of *k* motion trajectories, where *k* represents a single joint from set *m* of body joints. Each trajectory can be specified in either quaternions or position at time *t*. For example, considering the case where MoCap-like data is provided in positions:

- Given a set of *m* joint positions, a pose can be considered as:

$$x(t_i) = \{x^1, x^2, ..., x^m\}(t_i) \tag{1}$$

  where x^k are positions associated with the k_{th} joint at time index t_i in the motion trajectory. Each pose $x(t_i)$ is encoded by either a $2 \times m$ or $3 \times m$ dimensional vector, corresponding to the depth of the captured data.

In [11]'s formulation, the joint positions of the motion trajectories together with their first, second and third derivatives represents the basis for computing descriptors on the effort- and shape axes. These formulas are outlined from [11, pp. 25-26] as follows:

$$\begin{aligned} E(t_i) &= \sum_{k \epsilon K} \alpha_k v^k(t_i)^2 \\ Weight(T) &= max\, E(t_i), i\, \varepsilon\, [1, T] \end{aligned} \tag{2}$$

$$\begin{aligned} Time^k(T) &= \frac{1}{T} \sum_{i=1}^{T} a^k(t_i) \\ Time(T) &= \sum_{k \epsilon K} \alpha_k\, Time^k(T), i\, \varepsilon\, [1, T] \end{aligned} \tag{3}$$

$$\begin{aligned} Space^k(T) &= \frac{\sum_{i=2}^{T} ||x^k(t_i) - x^k(t_{i-1})||}{||x^k(T) - x^k(t_1)||} \\ Space(T) &= \sum_{k \epsilon K} \alpha_k\, Space^k(T), i\, \varepsilon\, [1, T] \end{aligned} \tag{4}$$

$$\begin{aligned} Flow^k(T) &= \frac{1}{T} \sum_{i=1}^{T} j^k(t_i) \\ Flow(T) &= \sum_{k \epsilon K} \alpha_k\, Flow^k(T), i\, \varepsilon\, [1, T] \end{aligned} \tag{5}$$

$$\begin{aligned} Shape^k(T) &= \frac{1}{T} \sum_{i=1}^{T} C^k(t_i) \\ Shape(T) &= \sum_{k \epsilon K} \alpha_k\, Shape^k(T), i\, \varepsilon\, [1, T] \end{aligned} \tag{6}$$

The variables v^k, a^k and j^k represent the magnitudes of the first, second and third derivatives (velocity, acceleration and jerk, respectively) and C^k the curvature of the motion trajectories associated with each joint *k* at time index *i*. The repeating pattern between the above formulas is that they compute a weighted sum of kinematic descriptors on a per joint basis over a time interval *T*. Subsequently, the contribution from each joint is included in the final estimate by the weight factor α_k. The result is that a small set of *global* feature descriptions of the performed motion for a set of time windows is achieved.

2.2.4 Action Segmentation

The concept of action segmentation in the context of LMA computational models, rely on the notion that *"[...] movement can be rationalized as a series of snapshots that can be ordered, structured, and formalized as the building blocks of a representation"* [24, p. 37]. That is, a particular movement can be regarded as an ordered set of gestures. These gestures can even be subdivided on a per joint basis. On a purely theoretical level, action segmentation is the process of splitting a kinematic signal (e.g. motion trajectories) into smaller windows of time based on some condition. Two LMA feature extraction studies [13, 17] propose a method based on finding the zero-crossings of the scalar acceleration trajectories associated with each joint. In their approach, an action is said to "begin", when the scalar acceleration exceeds a threshold. The action has ended after the second zero-crossing in the trajectory has been passed. The interonset time (in samples) between the beginning and end of the action determines the time window over which the LMA features are computed [13]. An even simpler approach that has been used in an LMA context is uniformly dividing a trajectory in time windows of size *N* [14]. Larboulette et al. [11] proposes three unsupervised clustering methods (e.g. [19–21]) to segment the motion trajectories. These authors stem from the field of kinematics research and proposes algorithms for extracting keyframes from MoCap data. In the context of *action segmentation* they are still relevant, as the keyframe-to-keyframe scenario has to encode the most relevant poses for reconstructing a movement-sequence through subsequent interpolation [20, 21].

2.3 Raw Audio Generation

Currently for raw audio generation ML, SampleRNN and WaveNet are the most ubiquitous tools. Raw audio generation is defined as the synthesis of audio waveforms rather than symbolic representations such as MIDI or sheet music; speech and music synthesis for example. Tools used for speech synthesis are often appropriated for music generation.

2.3.1 *WaveNet*

WaveNet is a fully probabilistic and auto-regressive model developed by Google in order to generate raw audio waveforms. [25]. WaveNet presented a state-of-the-art model in 2016 by incorporating dilated causal convolutions as the "main ingredient". Causal convolutions had previously been used in other contexts such as in signal processing [26] and image segmentation [27] but this was the first time that it had been used in the audio time domain specifically. WaveNet as a result was able to have a very large receptive field allowing it to learn the characteristics of music over large temporal scales.

WaveNet also uses a fully probabilistic model using the softmax function to select the most appropriate value for the next raw audio sample value. Raw audio would normally require 65536 outputs but, by quantising using the μ-law algorithm, this is reduced to 256 values. A typical Neural Network (NN) output results in the conditional average of the presented data where as this probabilistic view presents a condition probability distribution. For tasks where the conditional average is a poor representation of the statistical properties of the data, this probabilistic view often produces better results [28].

2.3.2 *SampleRNN*

SampleRNN proposed a model that utilises a hierarchical structure of modules at different temporal resolutions which allows SampleRNN to learn the data manifold directly from the audio samples [29]. This allows SampleRNN to be computationally much faster than WaveNet. SampleRNN was inspired by a need for a simpler model as WaveNet *"is just really complicated"* [30]. It differs by using a recurrent NN architecture. This helps to model the dependencies in audio data which inspired the use of an RNN architecture in the ML pipeline in this project. Similar to WaveNet, SampleRNN uses a μ-law quantised output to determine output samples.

SampleRNN produces great results for raw audio generation but it appears difficult to find a way to condition SampleRNN on the Laban features. These models alone are not capable of producing music conditioned on another input.

2.3.3 *Tacotron 2*

Tacotron 2 was published by Google Deepmind as a NN architecture designed for text to speech [31]. It uses both convolutional layers and LSTM layers in processing the input data to WaveNet as shown in Figure 1.

The most relevant section is the hidden feature representation which is used to generate a mel-spectrogram. An autoencoder is trained on mel-spectrograms to produce a vector representation of length 1024. The decoder then uses this vector representation in order to predict the mel-spectrogram.

This stage of Tacotron 2 is an example of sequence to sequence learning where the speech and linguistic feature input is translated into a mel-spectrogram.

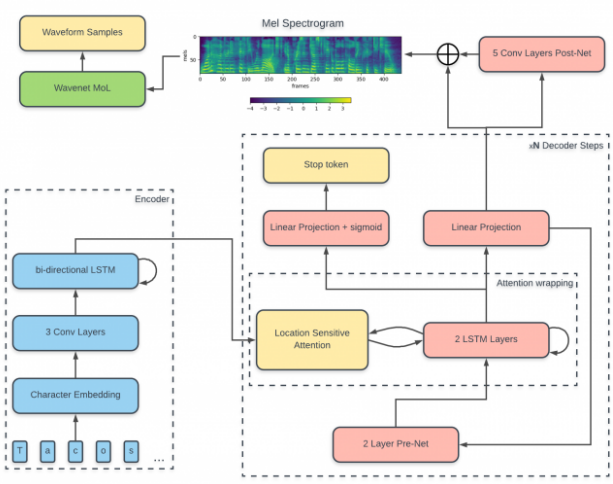


Figure 1. Tacotron2 Architecture [31]

2.4 Mixture Density Networks

Mixture Density Networks aim to model arbitrary conditional probability distributions instead of modelling arbitrary functions as in conventional NNs. This technique is particularly useful in models where an input may map to multiple outputs. One such example presented in the paper is inverse problems such as robot kinematics [28], or in sequence to sequence topologies, such as in [16]. Consider that a dance move has no inherent mapping to its accompanying audio. Hence, when a range of outputs are suitable, a probabilistic model is better suited than a conditional average. The state of the art raw audio generation systems use probabilistic output distribution as seen in 2.3.1 and 2.3.2.

2.5 Spectrogram Representation

One of the problems that arise when dealing with audio is the large amount of raw data to analyze. Therefore, it is common to perform feature extraction on the raw data and achieve a more compact representation. Transforming the signal from the time domain to the frequency domain allows a more compact set of audio features [32].

In fact, the spectral representation of music is so linked to the medium that composers have paid special attention to it, from impressionist musicians (Debussy, Ravel, Delius, Griffes, Varèse, etc.) to the movement of 'spectral composers' that emerged in 1970. In both cases, composers are more interested in the timbre of the works than in the melody or the rhythm itself [33].

Spectrograms, as a lower dimensional set of features, are being used in a wide variety of audio recognition / classification problems: genre recognition, speaker identification, speaker genre classification, artist classification, music transcription, etc. [34–36].

One particular interesting transformation that can be applied to spectrograms are the mel-filters bank to obtain a mel-spectrogram [37], so the linear-frequency scale will be non-linearly transformed into a mel-scale based in the perceptual equidistance of the tones, with fewer components than the original one (the resolution for higher frequencies

is reduced, mimicking human perception) Since this is a perceptual transformation, there is more than one implementation for this conversion. Librosa's implementation is based on the MATLAB Auditory Toolbox by Slaney [38], using a linear scale for frequencies below 1 kHz and logarithmic for frequencies above 1 kHz. The spectrogram is finally converted to dB. As noted, mel-spectrograms are a fundamental part of Tacotron 2 workflow [31], where they are generated in a prediction network and the feed into a modified version of WaveNet, but also are used in disciplines such as sound classification [39].

2.6 Convolutional Autoencoder networks

By using a spectrogram, it is possible to represent the audio as an image and, therefore, apply ML techniques that are used in the field of computer vision. In this case, one of the most common architectures used is a Deep Convolutional Network (DCN), since by its structure -imitating how the human vision works- it lends itself to discovering the structures present in the two-dimensional data, by a series of convolving and downsampling operations. The hidden layers are able to learn and represent the underlying structure of the data. For example, in [40], the authors use a DCN to classify 1.2 million high-resolution images into 1000 categories. The authors cite as a drawback the time needed to train such a network when the images are in high resolution, as well as the need to find datasets large enough to avoid overfitting.

In music and in particular, in automatic music genre classification, DCN are used independently with a three-layer architecture [41] or together with handcrafted features, both from the visual and the acoustic domains [42]. DCN are also used in speech recognition [43] and in music segmentation [44].

In order to further reduce the dimensionality of the data, an autoencoder [45] can be used, a unsupervised multilayer network architecture divided into two parts, the encoder, which reduces the dimensionality of the input data, compressing the input into a vector with a smaller number of elements, called "code vector"; and the decoder, which will reconstruct the original data using the code vector value as input. Autoencoders are used not only to get a more compact set of features (dimensions) but also as generative models [46], image inpainting [47] or anomaly detection [48].

3. DESIGN AND IMPLEMENTATION

The system is subdivided into a series of reusable modules, so we can use them with hardly any modifications both in the training stage and in the usage stage. The most prominent modules are the LMA Feature Extractor for the Laban feature extraction, the Convolutional autoencoder (CAE) for the audio feature extraction and the LSTM Network to generate the coded representation of an audio clip using the Laban features as its input (see fig. 2).

During the training stage, the CAE is trained -using the spectrograms obtained from the same ballet performances- to get a compressed representation of the audio. The LMA

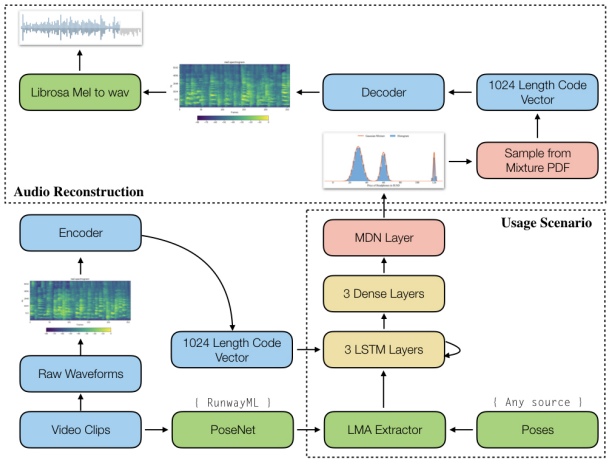


Figure 2. ASDF-RNN architecture

feature extractor is fed with pose data from the ballet performances to get a denser set of (LMA) features. Finally, the LSTM is then trained to generate a compressed audio vector for each element of the LMA features set.

In the usage stage, we present the LSTM network with a set of LMA features, and the output, a compressed representation of the audio, is presented to the decoder half of the CAE to generate the corresponding audio.

3.1 Movement Data Generation

We found no online databases to suit our specific context, hence we opted for a web-scraping approach instead. The data for subsequent training consists of six hours of video of solo classical ballet performances, acquired from various online sources (e.g. YouTube and Vimeo). From a custom made tool we sample each of these videos at a constant framerate of 30fps. Every frame is encoded in .png format and re-scaled to (600×400) pixels. Every frame is analyzed individually in RunwayML [1], using the PoseNet [22] model with the ResNet50 [49] architecture with an OutputStride of 16, prioritizing the quality of the detection over the processing speed. Subsequently, the frame data from RunwayML is grouped in JSON files (240 poses per frame batch). As a pre-filtering condition, we discard files with not enough poses and interpolate frames where the dancer is not detected. After this process, we obtain 1201 JSON files that collect the movements of the dancer for eight seconds each (e.g. 240 samples).

3.2 LMA Feature Extractor

The LMA extractor processes individual JSON files encoding a sequence of poses (as described above). We denote each JSON file a *frame batch*, which consists of 240 samples/poses. The extraction process is subdivided in three stages:

1. Estimate kinematic descriptors on a per joint basis for every pose in a batch.

2. Segment the motion per batch using a keyframe extraction algorithm.

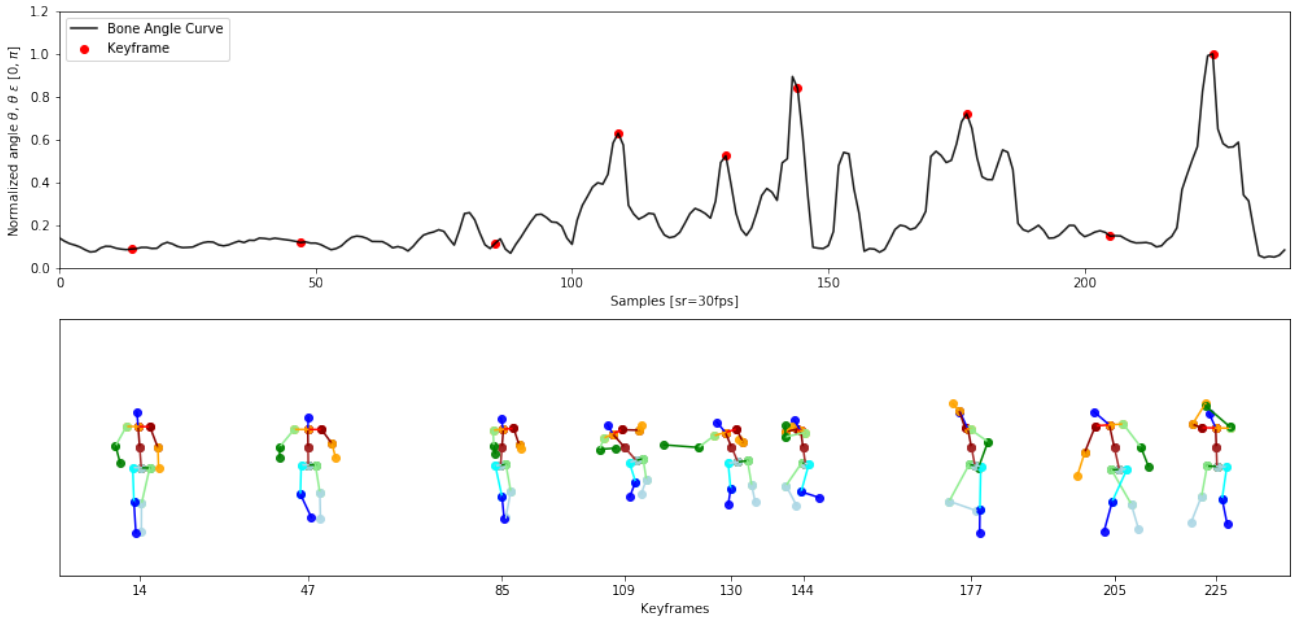


Figure 3. Mean bone angle curve θ, with corresponding keyframes (top). The pose at estimated keyframes (bottom)

3. Compute LMA descriptors per batch over the time windows as estimated by the keyframes.

3.2.1 *Kinematic Descriptors*

The kinematic descriptors; velocity, acceleration, jerk (first, second and third derivatives of the motion trajectory, respectively) and curvature are calculated on a per joint basis over the 240 poses in a frame batch. When estimating higher discrete derivatives, noise in the motion signal is gradually amplified. Therefore, the motion signal is preprocessed by being passed through a low-pass filter (here, the 2D-signal has been split into separate 1D-signals along each dimension). The low-pass filter is a 5th order IIR-filter with a cut-off frequency at 3Hz (cf. [50]). The formulas used for estimating the discrete derivatives are derived from [11]. For example, we estimate velocity as:

$$\boldsymbol{v}^k(t_i) = \frac{x^k(t_{i+1}) - x^k(t_{i-1})}{2\delta t} \tag{7}$$

As usual, the superscript k refers to the k_{th} joint in the pose at time t_i. δt is the sampling frequency, which in our case is $\frac{1}{30}$. The higher derivatives are estimated in a similar fashion to (7). All transients in the kinematic signals are simply estimated using an *edge* wrapping scheme, i.e. $t_0 = t_1$ and $t_{n+1} = t_n$. The scalar equivalents of the motion derivative signals are simply the magnitude of the vector quantity. Since the motion trajectories in our case are derived using PoseNet [22], they are all encoded in 2-dimensional cartesian coordinates. Hence, curvature is calculated using the 2D estimation:

$$C^k(t_i) = \frac{|detA|}{[\boldsymbol{v}^k(t_i) \bullet \boldsymbol{v}^k(t_i)]^{\frac{3}{2}}}, A = \begin{bmatrix} \boldsymbol{a}^k(t_i) \\ \boldsymbol{v}^k(t_i) \end{bmatrix}^T \tag{8}$$

where $\boldsymbol{v}^k$ and $\boldsymbol{a}^k$ are 2-dimensional row vectors and $\bullet$ represents the dot product.

3.2.2 *Action Segmentation*

To segment the motion trajectories into smaller time windows, we use a variation of the keyframe extraction algorithms described in [20] and [21]. First, the motion trajectories are reorganised. Each joint is connected with its neighbour joint (following the hierarchy described in [21]), thus representing the pose as a set of vectors, or "limb bones", with the k_{th} bone at time t_i denoted $\boldsymbol{B}^k(t_i)$. For each limb bone connecting two joints, we calculate an angle with a reference limb bone in the pose. That is,

$$\theta^k(t_i) = \arccos \frac{\boldsymbol{B}^k(t_i) \bullet \boldsymbol{B}^{ref}(t_i)}{|\boldsymbol{B}^k(t_i)||\boldsymbol{B}^{ref}(t_i)|} \tag{9}$$

For each curve represented by bone angle $\theta^k(t_i), i = 1, 2, ..., N$, we find the time indices t_i which represents the highest *curve salience*. Salient points are where the curve has high fluctuation with respect to neighboring points. In other words, where there are occurrences of drastic changes in the bone angle. Finding these salient points is a three-step process inspired by [20]:

1. An absolute difference of gaussians is computed for each bone angle curve θ^k. That is, we convolve θ^k individually with two gaussian kernels of varying sigma parameters. The absolute value of the difference between the two curves after the convolution we denote a *salience curve*, S^k:

$$S^k = |\theta^k * G(\mu, 2\sigma) - \theta^k * G(\mu, \sigma)| \tag{10}$$

In the above equation $\mu = 0$. The specific choice for σ and the window length of the gaussian kernel can be arbitrarily chosen [20]. In our case we chose a window length of size 5 and $\sigma = 0.75$.

2. The time indices t_i representing salient points are the points $S^k(t_i)$ greater than the mean of all points in S^k. We can formulate these time indices as the set F^k of *candidate keyframes*, f_i, for a curve associated with limb bone *k*:

$$F^k = \{f_i\,; \forall t_i : S^k(t_i) > mean\, S^k\} \tag{11}$$

3. The candidate keyframes in F^k close together in time are separated into their own clusters. The separation condition is simply that frames (i.e. time indices) within their cluster differ by no more than 5 samples in relation to their neighbors. For each cluster, we only retain the keyframe which maximises or minimises the original bone angle curve θ^k.

At this stage we have a subset $F_s^k \subseteq F^k$ of keyframes for each limb bone *k*. To achieve a representation of keyframes for the whole movement, we take the union of every F_s^k to get the set F of unique keyframes for every limb bone $k\,\varepsilon\,B$:

$$F = \bigcup_{k=1}^{|B|} F_s^k \tag{12}$$

Again, the keyframes are separated into clusters. However, the amount of clusters at this stage determines the eventual number of time windows that the motion will be segmented into. Since we do not want this number to be variable, we use the k-means algorithm to compute the clusters. A condition for k-means is that you supply how many clusters you expect. We take advantage of this property to get the same amount of clusters for each frame batch we process. Again, we only retain the keyframe of each cluster which maximises or minimises the angle curve θ, where we in this case estimate θ to be the mean of all bone angle curves θ^k:

$$\theta = \frac{1}{|B|} \sum_{k=1}^{|B|} \theta^k \tag{13}$$

The assumption here, is that common high value peaks in the individual bone angle curves will be retained due to constructive interference while opposing peaks will cancel each other out. The result after running the keyframe extraction procedure set to finding 9 keyframes can be seen in fig. 3.

3.2.3 Computing LMA Features

From the subset of unique keyframes $F_s \subseteq F$, we define the number of time windows T_i for a frame batch to be the interonset time between consecutive keyframes:

$$T_i = f_{i+1} - f_i,\; i\,\varepsilon\,[1,\, |F_s| - 1] \tag{14}$$

We compute every LMA feature along the effort- and shape axes as outlined from the equations (eqs. (2) to (6)) in section 2.2.3 for every time window T_i. Recall that the contribution from each joint is combined through a weighted average with the weight factor α_k. In our case, we chose to emphasise the ankle- and wrist joints, since these joints see the most change over time in a ballet performance.

3.3 Audio Feature Encoding

Before the CAE can be trained, the data should be extracted from the performance videos and converted to spectrograms. This process consists of two steps:

1. **Extraction** of 21825 audio fragments of 8 seconds of duration, downsampled to 22050Hz, 16 bit, mono. The audio fragments are overlapped 7 seconds (one-second hop between fragments). Shorter duration fragments are discarded during the extraction process.

2. **Generation** of 128 band mel-spectrograms, using librosa [51], with a FFT window of 4096 samples (185.7ms) and a hop between frames of 690 samples (31.3ms) to obtain a representation of (128×256) elements per spectrogram. Phase data is discarded. The spectrogram (power) is then transformed to dB.

The convolutional autoencoder is trained to extract a set of 1024 features from the mel-spectrograms. This reduced set will be used in the LSTM network. The autoencoder is not symmetrical: the encoder is a deep convolutional network, and the decoder is a shallow network.

The encoder and decoder are created as separate models. A third model linking the encoder and decoder is also created. The full model is used during the training stage: the encoder is used to generate the training set for the LSTM network and the decoder model is used for reconstructing the spectrograms from the LSTM-generated code vectors.

The autoencoder has the following architecture (Figure 4):

- **Input layer**: 32768 (128×256) neurons.
- **Convolutional layers**: four two-layer stages, a fully-connected layer with 32768 neurons and the code layer, with 1024 neurons. The first layer of each two-layer stage is a convolutional layer, in charge of stacking filters (32, 64, 128 and 256) using a $(3, 3)$ kernel and a stride of 1. The second layer will reduce the dimensionality of the previous layer to (64×128), (32×64), (16×32) and (8×16) respectively by applying a pooling operation.
- **Dense layer**: the final convolutional layer is flattened and fully-connected to the code vector.
- **Code vector**: this layer is common to the encoder and the decoder. It has 1024 neurons and is fully-connected to the output layer.
- **Output layer**: 32768 (128×256) neurons.

We use ReLU [52] as the activation function (faster to train than $tanh$ [40]) in all layers except the last one, where we use a sigmoid function.

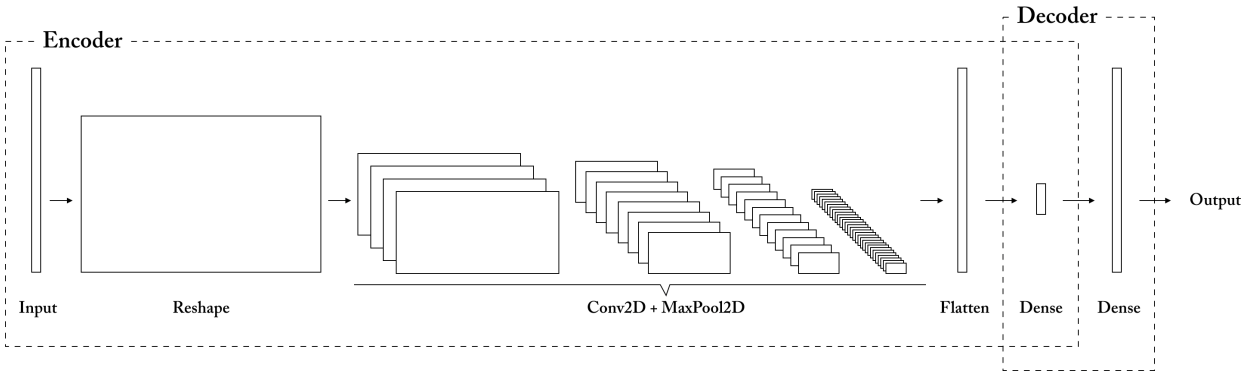


Figure 4. CAE structure.

3.4 LSTM Network

A RNN (Figure 5) is trained to produce the vector of 1024 elements from the Laban features and loss is calculated against the vector representation of the accompanying spectrogram as produced by the CAE. The output from this RNN will be used in the final pipeline as input to the decoder from the previous section.

- **LSTM layers**: The network consists of 2 LSTM layers of 512 and 1024 units with an input of 9x5 corresponding to the 5 Laban features for 9 detected dance moves. The first LSTM uses return sequences as true while the second is false as the dimensional for the spectrogram must be reduced from 3 to 2. The activation function for both these layers is sigmoid.
- **Dense layers**: Next there is a series of Dense and Dropout layers of size 2048, 4096, 2048. The first layer uses a sigmoid activation function with the next two layers being ReLU.
- **MDN layer**: Finally, a Mixture Density Network (MDN) layer is used as the final output. The MDN is set up with a final dimension output of 1024 and the number of distributions is set to 10.

4. RESULTS

4.1 Convolutional autoencoder

4.1.1 Training parameters

For the training process, the data is split 80-20 into training and testing (17460 and 4365 spectrograms). The CAE is trained for 400 epochs with a batch size of 128, using adam [53] as the optimizer with a fixed learning rate of 0.001 and minimum square error as the error function.

Using the final architecture, the autoencoder training takes 3 hours on a Titan X. At the end of that period, we got a training loss of 0.0019 and a validation loss of 0.0031.

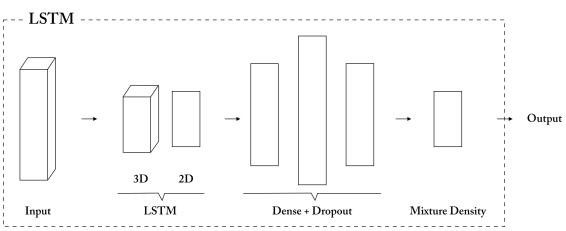


Figure 5. LSTM structure.

4.1.2 Evaluation

We have found that the error must be kept below 0.0040 (ideally below 0.0035) for the CAE to produce recognizable audio. In another case, we will get sounds that go from noise to some kind of audio reconstruction (e.g. table 2), but we need to stay below that threshold to get recognizable musical fragments.

Table 2. Loss vs. perceived quality of sound

Loss	Results
> 0.0080	Noise
0.0080 - 0.0060	Amplitude modulated noise
0.0060 - 0.0050	Amplitude modulated timbre
0.0050 - 0.0035	Blurred notes
< 0.0035	Recognizable melody

To achieve these values we started with approx. 2700 non-overlapping sound fragments, and a simple autoencoder architecture, with no convolutional layers. The autoencoder learned all the examples from the training set, but was overfitting severely, and the validation loss was over 0.0060, way above the threshold of "musical intelligibility" we were aiming for.

To avoid this situation, we augmented the data, generating overlapping spectrograms. We tried several dataset sizes generating fragments with 4-second hop, a 2-second hop, and lastly, 1-second hop. Data augmentation reduced overfitting, but increased the training loss, which forced us to add more complexity to the architecture of the network, introducing the convolutional layers, and increased the dimensionality of the code vector from 256 to 1024 (still a 172-to-1 compression). After trying several architectures, with additional dense layers before and after the code vector, we have obtained the best results flattening the last convolutional layer (256 8×16 filters) which gives us again a layer with 32768 neurons (the same number as

pixels in the image) fully connecting that layer to the code vector and making the decoder shallow. With that architecture (as shown in Figure 4) we were able to obtain both a validation and training loss below our required threshold.

Figure 6 shows the relationship between dataset size and training and validation losses and the intelligibility threshold.

4.2 LSTM Network

4.2.1 Training parameters

The LSTM Network was trained for 3750 epochs taking approximately 15 hours on a Titan X. This resulted in a training loss of -1084.61 and a validation loss of -1124.99 as seen in Figure 7. During training the batch size was set to 8, and during predicting and sampling the output from the MDN layer the temperature is set to 1.

When normalising the Laban input data, some features have large extreme values over different time-steps which results in very small, similar values. When each feature is scaled with respect to the other features in that series of time-steps it yields better results.

After preprocessing, the dataset contains 1201 total spectrogram representations with 9x5 Laban features. Using a 80-20 split for the data set this corresponds to 960 features for training and 241 features for validation. The optimizer is RMSProp with a learning rate of 0.0001 and the ρ parameter is set to 0.9.

The output of the LSTM Network is a combination of distributions that are sampled to obtain the 1024 length vector. The sampling is dependent on temperature. For the temperature value we found that 1.0 provides the best results and even varying this to 0.7 or 1.5 causes the quality of the final audio to decrease significantly to the point where it is indistinguishable from noise.

4.2.2 Evaluation

The Network appears to converge for training and validation, but during training the validation loss follows a trend of long periods of non-decreasing validation before finding a new minima as shown in Figure 7.

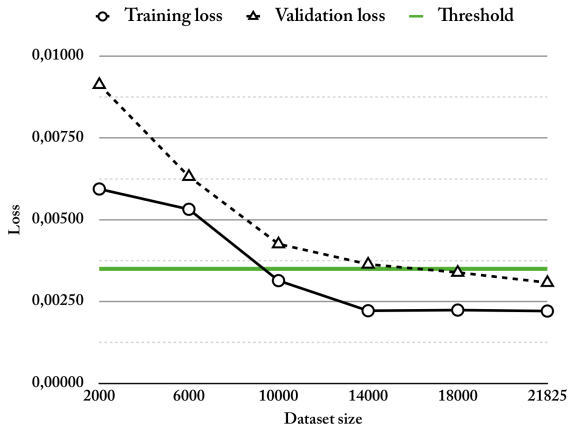


Figure 6. Relationship between number of audio fragments used and training and validation loss.

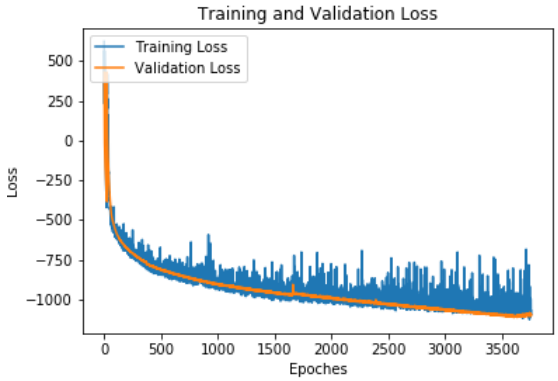


Figure 7. LSTM RNN Loss.

4.3 ASDF Audio Evaluation

To evaluate the performance of ASDF-RNN a series of videos have been chosen including two charleston numbers, the comic sketch Ministry of Silly Walks (Monty Python's Flying Circus, 1970), a musical number from the film "Singin' in the Rain" (Stanley Donen, Gene Kelly, 1952) and different poses by the SMC-2019 student team. Using the same tools as in the training stage, 123 fragments are obtained. Spectrograms are presented to the CAE, and Laban features are presented to the LSTM Network getting a total of 123 audio fragments.

Table 3 shows two generated wav files. These spectrograms and waveforms have been generated only using Laban dance features and ASDF-RNN at different stages of training. Wav10 and Wav50 are examples of the training and validation the output of ASDF-RNN respectively. ASDF-RNN generalises well with comparable sounding results found for both the training and the validation examples but there is a recurring issue of unwanted noise that is present throughout all the generated audio.

Table 4 shows a comparison of audio for 000.wav. The generated audio uses ASDF-RNN system to generate audio from Laban features and the decoded audio is the ideal case with the latent space 1024 vector generated from its corresponding encoder. There are some similarities in the audio with issues stemming from the reconstruction of the phase information but the ASDF-RNN has clearly learned how the lower frequencies generally have a higher power. The trend of this attenuation of power with frequency has also been modelled somewhat correctly by the LSTM Network and CAE. The audio for these examples can be found in the examples from Table 1.

4.4 Qualitative Assessment of Generated Audio

We did an informal qualitative assessment on the quality of the generated audio from ASDF-RNN, through a very brief listening test with subsequent scoring. The assessment was simply carried out by colleagues from the Sound and Music Computing education at Aalborg University. All our colleagues have previous musical experience, and one had a background in non-western music. For the informal procedure, We generated three fragments of audio, presenting

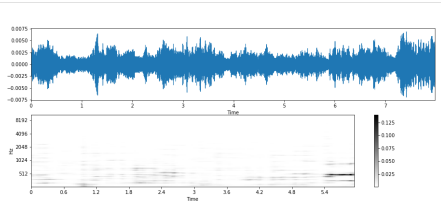
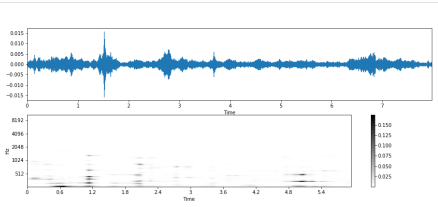
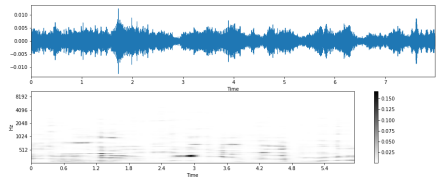
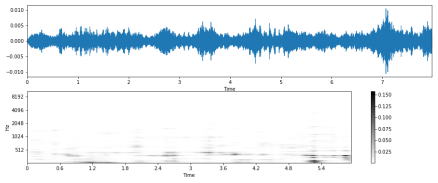
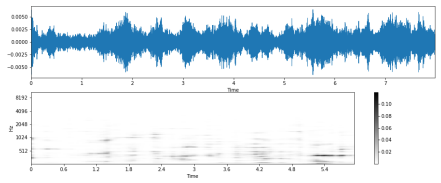
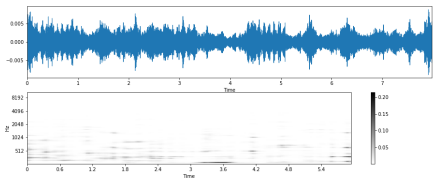
Epochs	Wav 10	Wav 50
	Spectrogram & Waveforms	
100		
1000		
3000		

Table 3. ASDF-RNN Waveform and Spectrogram examples

sequentially the same three fragments to the five subjects. We were curious whether they could perceive significant differences in melodic or harmonic content of the fragments. As such, we asked them to rate on a 10-point likert-type scale if they perceived the fragments to be different, with 1 being very similar and 10 very different. Four rated the fragments to be very similar (between 2-3 points on the scale). The fifth subject, the only one versed in non-western music, rated the fragments as "somewhat different" (5 points on the scale).

5. DISCUSSION

5.1 Audio

ASDF-RNN audio generation is a two-stage process. The LSTM generates a code vector and the CAE renders it into sound. The CAE is robust enough to encode unheard fragments of classical music (specially strings and piano). Testing with other music styles (Charleston, pop, spoken word) and other timbres (wind, percussion, choir, etc.) the results are poor but promising. The CAE tries to reproduce those unheard sounds and is able to generate a similar amplitude envelope, but can not reproduce the timbre. Although the CAE is not intended to be used as a end-to-end music compressor, since once trained we only use the decoder layers, training the CAE using classical pieces with different orchestrations should improve its response and get better audio reconstructions.

ASDF-RNN produces progressively better sound as the number of epochs increase for the LSTM Network. In both examples presented in Table 3, there are unwanted audio artifacts heard in the audio. As the number of epochs increase this sound has become less obvious leading to better quality audio.

The audio produced by the LSTM has a high signal-to-noise ratio, but has no clear tonal center and the sounds are merged in a continuous glissando, producing an overall feeling of unpleasantness and uneasiness. This is due the lack of consistent rhythm and a steady melodic reference. One suggestion for why we interpret this audio as unpleasant is an uncanny valley for audio. Mori [54] suggests that there is a link between uncanniness and fear and before the point of complete familiarity (In our case realistic sounding music) there is a point of maximum unfamiliarity. While there is no literature currently directly supporting an uncanny valley for musical audio, Avdeeff [55] and Grimshaw [56] have suggested links between uncanniness and fear and unfamiliarity and between randomness and urgency for audio in other contexts.

Because of the lack of tonal center (the original pieces being written in different keys and) and the oscilating tones, the resulting audio sounds random, and all the fragments are perceived as very similar sounding, despite presenting clear differences when represented in the time domain as a waveform or in the frequency domain as a spectrogram. The CAE should learn what cadences and sounds are allowed. However, right now, the NN does not have enough long-term information to understand the rhythmic, harmonic and melodic relationships among notes.

Pitch adjusting software (Propellerheads' Neptune pitch adjuster) had problems detecting the pitch of the generated audio. Also, Adobe's Audition was not able to obtain coherent results when showing the spectral pitch of the fragment. However, using the generated audio as the modulator wave in a vocoder (Propellerheads' BV512 Digital Vocoder) the resulting audio sounded like a string instru-

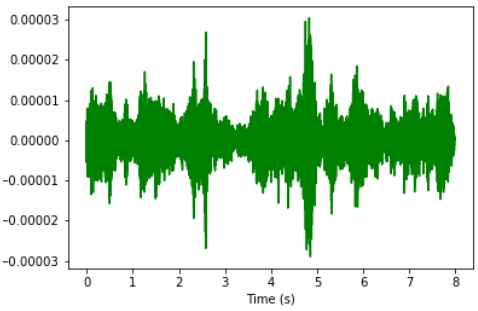
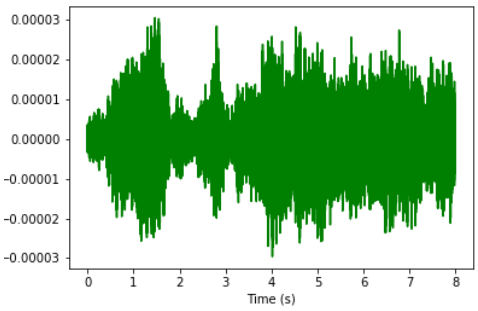
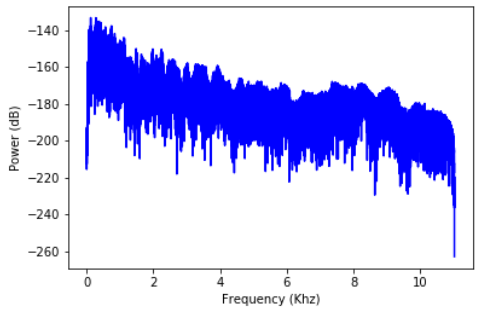
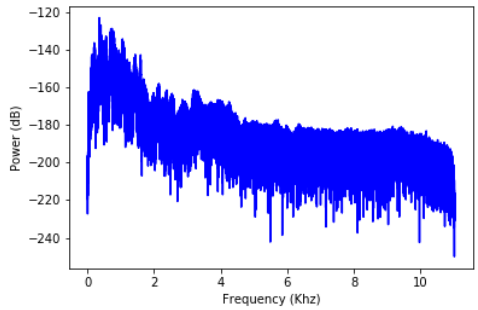
	Generated Audio	Reconstructed Audio
Time Domain		
Frequency Domain		

Table 4. Amplitude & Frequency Comparison Table for 000.wav

ment. At this point, ASDF-RNN has learned the amplitude modulation of the audio and the expected timbre, but it still needs to learn how to produce tones and music.

From examining the spectrograms we can see the prominence of audio at low frequencies. When listening to the audio we can hear beating low frequencies which indicate that there are at least two frequencies within their respective critical band. This could be learned from an overuse of thrills in the training dataset but it possible that this could be an effect from the NN performing poorly at selecting specific frequencies for low register instruments and clustering them too close together. The low frequencies may be attenuated if desired using a 12dB per octave high-pass filter (e.g. MEqualizer by Melda Productions).

During the development of this system the authors were required to listen attentively to the out audio to evaluate the results. Evaluation may have been hindered by the occurrence of listening fatigue. Symptoms can include tiredness and loss of sensitivity and thus, it becomes harder to evaluate audio accurately. As the listener becomes desensitised to artifacts the quality of the output can be misinterpreted.

6. CONCLUSIONS

In this paper we have taken initial steps in the investigation towards raw audio synthesis directly from motion features, in our case derived from classical ballet dance. We have provided a thorough explanation of acquiring these motion features using our own adaptation of a Laban Movement Analysis computational model. Inspired by WaveNet and Tacotron2, we used an Encoder-RNN-Decoder topology to infer raw audio from motion. We have split the implementation of the CAE (Convolutional autoencoder) and the RNN (Deep LSTM network) into separate modules. The CAE and the RNN were trained separately. The CAE was used to produce the target values for training the RNN. Only the decoder module of the CAE is considered when using and evaluating ASDF-RNN. Feeding new motion features through the pipeline produces relatively clear audio, however the relation to its original source (a classical music piece) is extremely small on a perceptual level. The generated audio appear virtually identical, retaining only small fragments of perceivable melody. The results are still promising however, and should be seen as a proof concept in an otherwise sparse area of research.

7. FUTURE WORK

Future work will mainly focus on improving both audio synthesis and sound quality. Increasing the size of the dataset will help both the CAE and the LSTM Network. The data used to train the CAE consists of 6 hours of video and augmented audio data. We could use additional fragments from ballet performances or incorporate a library such as Google Audioset [57] that contains over a million musical audio clips. A post-processing stage can also be useful to filter out some of the aural artifacts the CAE generates: choppy sound, flanger-like sounds, noise, etc.

ASDF-RNN is trained only on ballet music. Alternative versions could be trained in different dance styles and use the LMA Feature Extractor to detect the style and use the adequate model. Also, new CAE and LSTM architectures can be tested. By adding a Variational layer [58] that encodes probabilities instead of points, we can get a decoder that responds better to an unknown input. Final audio results are showing significant capacity for improvement that

can be achieved by improving the training and by testing new architectures.

As the network was trained on a batch size of 8, it requires that the input to the model be a series of 8 9x5 Laban features. A person dancing may not produce the ideal input data size for the architecture. Hence the ability to be able to use different input parameters is worth investigating. An ideal model would be able to take a variable number of dancing fragments instead of the 8 required and would not rely on a static number of Laban dance features.

The training and usage stages are composed by a group of reusable but disjointed scripts (Processing, bash scripts, Python notebooks...) The pipeline could be more agile by integrating some of the tools used. For example, instead of pregenerating spectrograms, they can be created on the fly in Keras using an user made extension [59]. By improving the workflow we can obtain a continuous stream of audio, but with an 8 second delay. Real-time feedback would need additional research.

8. REFERENCES

[1] C. Valenzuela, A. Matamala, and A. Germanidis, "Runway: Adding artificial intelligence capabilities to design and creative platforms," 2018. [Online]. Available: http://nips2018creativity.github.io/doc/runway.pdf

[2] [Online]. Available: https://magenta.tensorflow.org/

[3] [Online]. Available: https://mimicproject.com/

[4] N. Yalta, S. Watanabe, K. Nakadai, and T. Ogata, "Weakly-supervised deep recurrent neural networks for basic dance step generation," in *2019 International Joint Conference on Neural Networks (IJCNN)*, vol. 2019-. IEEE, 2019-07, pp. 1,8.

[5] K. McDonald, "Dance x machine learning: First steps," 2018. [Online]. Available: https://medium.com/@kcimc/discrete-figures-7d9e9c275c47

[6] H. Guðmundsson, K. Ólafsson, and K. Ravnhøj, "CALMUS wave," 2015. [Online]. Available: http://www.calmus.is/projects

[7] [Online]. Available: https://soundmoovz.dmetproducts.co.jp/

[8] M.-A. Weibezahn, "Eins, zwei - an interactive space for rhythm and movement," Master's thesis, Hochschule für Künste Bremenn, 07 2019.

[9] A. Camurri, M. Ricchetti, and R. Trocca, "Eyesweb-toward gesture and affect recognition in dance/music interactive systems," *Proceedings IEEE International Conference on Multimedia Computing and Systems*, vol. 1, pp. 643–648 vol.1, 1999.

[10] J. James, T. Ingalls, G. Qian, L. Olsen, D. Whiteley, S. Wong, and T. Rikakis, "Movement-based interactive dance performance," in *Proceedings of the 14th ACM International Conference on Multimedia*, ser. MM '06. New York, NY, USA: ACM, 2006, pp. 470–480. [Online]. Available: http://doi.acm.org/10.1145/1180639.1180733

[11] C. Larboulette and S. Gibet, "A review of computable expressive descriptors of human motion," in *MOCO*, 2015.

[12] O. Alemi and P. Pasquier, "Machine learning for data-driven movement generation: a review of the state of the art," *ArXiv*, vol. abs/1903.08356, 2019.

[13] L. Zhao and N. I. Badler, "Acquiring and validating motion qualities from live limb gestures," *Graphical Models*, vol. 67, pp. 1–16, 2005.

[14] D. S. Maranan, S. F. Alaoui, T. Schiphorst, P. Pasquier, P. Subyen, and L. Bartram, "Designing for movement: evaluating computational models using lma effort qualities," in *CHI*, 2014.

[15] A. Solander, L. Lessel, S. B. Olsen, and N.-F. Christiansen, "Towards an AI approach for Interactive Digital Theatre," Bachelor's Thesis, Aalborg University Copenhagen, 05 2018.

[16] L. Crnkovic-Friis and L. Crnkovic-Friis, "Generative choreography using deep learning," *CoRR*, vol. abs/1605.06921, 2016. [Online]. Available: http://arxiv.org/abs/1605.06921

[17] L. Souza and S. Freire, "Towards an interactive tool for music and dance: gestures, laban movement analysis and spectromorphology," *Revista Música Hodie*, vol. 18, no. 1, pp. 117–131, 2018.

[18] D. Pagliari and L. Pinto, "Calibration of kinect for xbox one and comparison between the two generations of microsoft sensors," *Sensors*, vol. 15, no. 11, pp. 27 569–27 589, 2015.

[19] Q. Zhang, S.-P. Yu, D. sheng Zhou, and X. peng Wei, "An efficient method of key-frame extraction based on a cluster algorithm," in *Journal of human kinetics*, 2013.

[20] E. Bulut and T. Capin, "Key frame extraction from motion capture data by curve saliency," in *Computer animation and social agents*, 2007, p. 119.

[21] J. Xiao, Y. Zhuang, T. Yang, and F. Wu, "An efficient keyframe extraction from motion capture data," in *Computer Graphics International*, 2006.

[22] A. Kendall, M. K. Grimes, and R. Cipolla, "Posenet: A convolutional network for real-time 6-dof camera relocalization," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 2938–2946, 2015.

[23] J. Levy and M. Duke, "The use of laban movement analysis in the study of personality, emotional state and movement style: An exploratory investigation of the veridicality of "body language"," *Individual Differences Research*, vol. 1, pp. 39–63, 04 2003.

[24] D. Glowinski, S. Y. Coll, N. Baron, M. Sanchez, S. Schaerlaeken, and D. M. Grandjean, "Body, space, and emotion: A perceptual study," *Human technology*, vol. 13, no. 1, pp. 32–57, 2017.

[25] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *SSW*, 2016.

[26] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," *Wavelets, Time-Frequency Methods and Phase Space*, vol. -1, p. 286, 01 1989.

[27] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," *CoRR*, vol. abs/1412.7062, 2014.

[28] C. M. Bishop, "Mixture density networks," 1994, forthcoming.

[29] S. Mehri, K. Kumar, I. Gulrajani, R. Kumar, S. Jain, J. Sotelo, A. C. Courville, and Y. Bengio, "SampleRNN: An unconditional end-to-end neural audio generation model," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*, 2017. [Online]. Available: https://openreview.net/forum?id=SkxKPDv5xl

[30] P. Kozakowski and B. Michalak, "Deepsound," Feb 2017. [Online]. Available: http://deepsound.io/samplernn_first.html

[31] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerry-Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," 2017.

[32] M. French and R. Handy, "Spectrograms: Turning signals into pictures," *Journal of Engineering Technology*, vol. 24, pp. 32–35, 03 2007.

[33] B. Mabury, "An investigation into the spectral music idiom and its association with visual imagery, particularly that of film and video." 2006.

[34] Y. M. Costa, L. S. Oliveira, A. L. Koericb, and F. Gouyon, "Music genre recognition using spectrograms," in *2011 18th International Conference on Systems, Signals and Image Processing*. IEEE, 2011, pp. 1–4.

[35] H. Lee, P. Pham, Y. Largman, and A. Y. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. D. Lafferty, C. K. I. Williams, and A. Culotta, Eds. Curran Associates, Inc., 2009, pp. 1096–1104.

[36] S. Böck and M. Schedl, "Polyphonic piano note transcription with recurrent neural networks," in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012, pp. 121–124.

[37] S. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE transactions on acoustics, speech, and signal processing*, vol. 28, no. 4, pp. 357–366, 1980.

[38] M. Slaney, "Auditory toolbox: A matlab toolbox for auditory modeling work," Interval Research Corporation, 1998.

[39] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *2015 IEEE 25th International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, 2015, pp. 1–6.

[40] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[41] T. L. Li, A. B. Chan, and A. H. Chun, "Automatic musical pattern feature extraction using convolutional neural network," in *In Proc. IMECS*, 2010.

[42] Y. M. Costa, L. S. Oliveira, and C. N. Silla Jr, "An evaluation of convolutional neural networks for music classification using spectrograms," *Applied soft computing*, vol. 52, pp. 28–38, 2017.

[43] T. N. Sainath, A.-r. Mohamed, B. Kingsbury, and B. Ramabhadran, "Deep convolutional neural networks for lvcsr," in *2013 IEEE international conference on acoustics, speech and signal processing*. IEEE, 2013, pp. 8614–8618.

[44] K. Ullrich, J. Schlüter, and T. Grill, "Boundary detection in music structure analysis using convolutional neural networks." in *ISMIR*, 2014, pp. 417–422.

[45] G. E. Hinton and R. R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *science*, vol. 313, no. 5786, pp. 504–507, 2006.

[46] D. P. Kingma and M. Welling, "An introduction to variational autoencoders," *Foundations and Trends® in Machine Learning*, vol. 12, no. 4, p. 307–392, 2019. [Online]. Available: http://dx.doi.org/10.1561/2200000056

[47] O. Shcherbakov and V. Batishcheva, "Image inpainting based on stacked autoencoders," in *Journal of Physics: Conference Series*, vol. 536, no. 1. IOP Publishing, 2014, p. 012020.

[48] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*. ACM, 2014, p. 4.

[49] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016, pp. 770–778.

[50] S. A. Skogstad, "Filtering motion capture data for real-time applications," in *NIME*, 2013.

[51] B. McFee, C. Raffel, D. Liang, D. Ellis, M. Mcvicar, E. Battenberg, and O. Nieto, "librosa: Audio and music signal analysis in python," in *Proc. of the 14th Python in Science Conf. (scipy 2015)*, 01 2015, pp. 18–24.

[52] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 807–814.

[53] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[54] M. Mori, K. F. Macdorman, and N. Kageki, "The uncanny valley [from the field]," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 98,100, 2012-06.

[55] M. Avdeeff, "Artificial intelligence & popular music: SKYGGE, flow machines, and the audio uncanny valley," *Arts*, vol. 8, no. 4, 2019-10-01. [Online]. Available: https://doaj.org/article/3d70e023753042d4bc5ea273459f04a2

[56] M. Grimshaw-Aagaard, "The audio uncanny valley: Sound, fear and the horror game." *Games Computing and Creative Technologies: Conference Papers (Peer-Reviewed)*, 01 2009.

[57] "Audioset." [Online]. Available: https://research.google.com/audioset/dataset/index.html

[58] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[59] K. Choi, D. Joo, and J. Kim, "Kapre: On-gpu audio preprocessing layers for a quick implementation of deep neural network models with keras," *ArXiv*, vol. abs/1706.05781, 2017.