Introduction:

Music is a great art to connect people and a medium to express feelings. So many people spend their leisure time watching movies and listening to songs during boring times. But music composers and singers generate themes. What do you think if you can generate music by yourself with your own choice? AI machine learning helps you to develop your music. We implement an AI system which able to create your music. For this purpose, we will show you a short film in silence and describe the movie with your facial expression and hand gesture what you watched in the film. At first, our system detects and collects human emotions and then converts these emotions into a piece of music.

Methodology:

As this project generates music from human body movement and facial expression (emotion detection), we organized our system into two phases. At first, we collect emotion from body movement and then generate AI music, and at last, we combine two steps for our final result.

Detect Emotion:

As we collect emotion from visitors in live streams, we use Mediapipe, which gives us good performance. It is used for different purposes like face, hand, pose, iris, eyes, and hair detection. Among all of them, we only use face and hand gesture detection. It is detected face and hand gestures with different landmarks points in face and hands (see figure-1)

Figure-1

We collect facial (468 landmark points), right (21 landmark points), hand and left (21 landmark points) hand landmarks data in different positions and give them other emotional names like happy, sad, joy, and so on (see figure-2). As I mentioned, 468 landmark points have two coordinates (x and y) on a scale of [0,1]. This is a normalized coordinate value. The same things also applied to the right and left hands. We collect facial, left, and right hands landmarks at the same time with different positions and store them in a list that will be used for training and an interface session. With the help of a webcam, we collect twelve positions concerning the relevant emotions.

Figure-2

Now, it's time to train the Machine Learning model, and we used 1400 images and 107900 coordinates for each emotion. We used the train test split method to split collected data into train and test data sets. We trained on a particular portion of our data and tested it on a separate partition to see how well it typically performs in the real world. So ideally, if we get a good accuracy metric when we test out our model, this should indicate that it will perform well when we go and make our detections. We next want to set up our features and the target variable. So, our parts will be all our coordinate values. Those will be

the features we've got to pass to our classification model, and our class will be our target variable. So, we're effectively trying to train a multi-class classification model to understand the relationship between this class and these coordinates. So ideally, we should be able to posture new coordinates, and our models can then predict what class that is. So, it's going to be then able to decode our body language and determine whether or not whether happy, whether or not sad, whether or not Joyful, and again good add in a whole bunch of additional classes to the end of this to be able to train up and classify different poses. We used four classification models to train our data (Logistic Regression, Ridge Classifier, Random Forest classifier, and Gradient Boosting classifier) to define the best model for our data set. After training our four classification models with our data, we got the following table (Table -1). It performed, and in this case, it performed pretty well, so our accuracy.

Model	Accuracy
Logistic Regression	100%
Random Forest	100%
Ridge Classifier	100%
Gradient Boosting	98%

For three of our models was about 100%. So, this is very high and gives us a bit of anxiety because nothing is ever perfect, but in this case, we'd got 100%. We used the random forest model we called the emotion detection model (Figure 3), which is suitable for big data (in data science). We can also use it for a vast data set later.

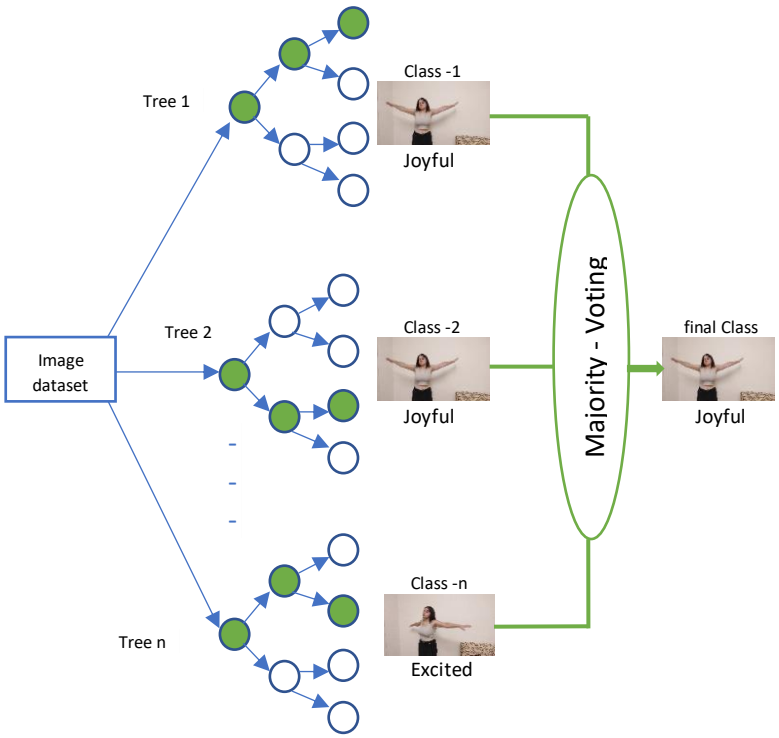

Figure-3: Random Forest Classifier

How to work the Random Forest algorithm?

Random Forest is a well-known ML (Machine Learning) algorithm for supervised learning methods. It is not only used in classification but also regression problems. This model can be used properly to solve a complex issue and improve the performance of machine learning models.

In layman's terms, Random Forest is a classifier that contains several decision trees on various subsets of a given dataset and takes the average to enhance the predicted accuracy of that dataset. Instead of relying on a single decision tree, the random forest collects the result from each tree and expects the final output based on the majority votes of predictions.

Now that you know what Random Forest Classifier is and why it is one of the most used classification algorithms in machine learning, let's dive into a real-life analogy to understand it better.
The Working of the Random Forest Algorithm is quite intuitive. It is implemented in two phases: The first is to combine N decision trees with building the random forest, and the second is to make predictions for each tree created in the first phase.
The following steps can be used to demonstrate the working process (Figure 4):
Step 1: Pick M data points at random from the training set.
Step 2: Create decision trees for your chosen data points (Subsets).
Step 3: Each decision tree will produce a result. Analyze it.
Step 4: For classification and regression, accordingly, the final output is based on Majority Voting or Averaging, accordingly.

Generate Music:
Now it's time to Generate Music. Musical sequences are fundamentally high-dimensional. For example, consider the space of all possible monophonic piano melodies. At any given time, exactly one of the 88 keys can be pressed down or released, or the player may rest. We can represent this as 90 events (88 key presses, one release, one rest). If we ignore tempo and quantize time down to 16^{th} notes, two music measures (bars) in 4/4 time will have 90^{32} possible sequences. If we extend this to 16 bars, it will be 90^{256} possible sequences, which is many times greater than the number of atoms in the Universe!
Exploring melodies by enumerating all possible variations is not feasible and would result in many unmusical sequences that sound random. For example, we show "piano rolls" of samples randomly chosen from the 90^{32} possible 2-bar lines. The vertical axis represents the notes on the piano, and the horizontal axis represents time in the 16th note steps. We also included the synthesized audio for one of the samples.
Many models can learn latent representations, each having various tradeoffs concerning the three properties we desire. One such model is called an autoencoder (AE). An autoencoder builds a latent space of a dataset by learning to compress (encode) each example into a vector of numbers (latent code, or *z)* and then reproduce (decode) the same example from that vector of numbers.

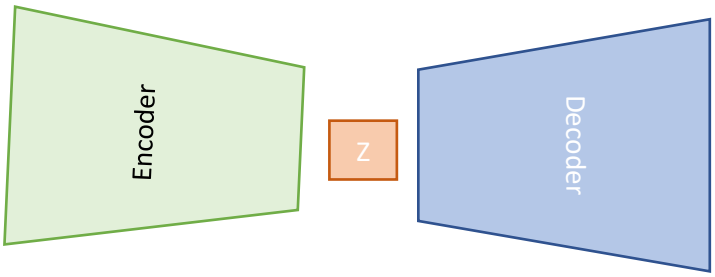

Figure 4: VAE model

A vital component of an AE (Figure 4) is the bottleneck introduced by making the vector have fewer dimensions than the data itself, which forces the model to learn a compression scheme. In the process, the autoencoder distills the dataset's common qualities. NSynth is an example of an autoencoder that has learned a latent space of timbre in the audio of musical notes.

I used a famous pre-train model for AI music generation with machine learning called MusicVAE (magenta). At a high level, this model maintains some prime structure rules used in VAEs for sequential data [17]. However, this paper [17] introduced a novel hierarchical decoder, which is demonstrated to produce a substantially better performance on long sequences. A schematic of this model, dubbed "MusicVAE," is shown in Figure 5.

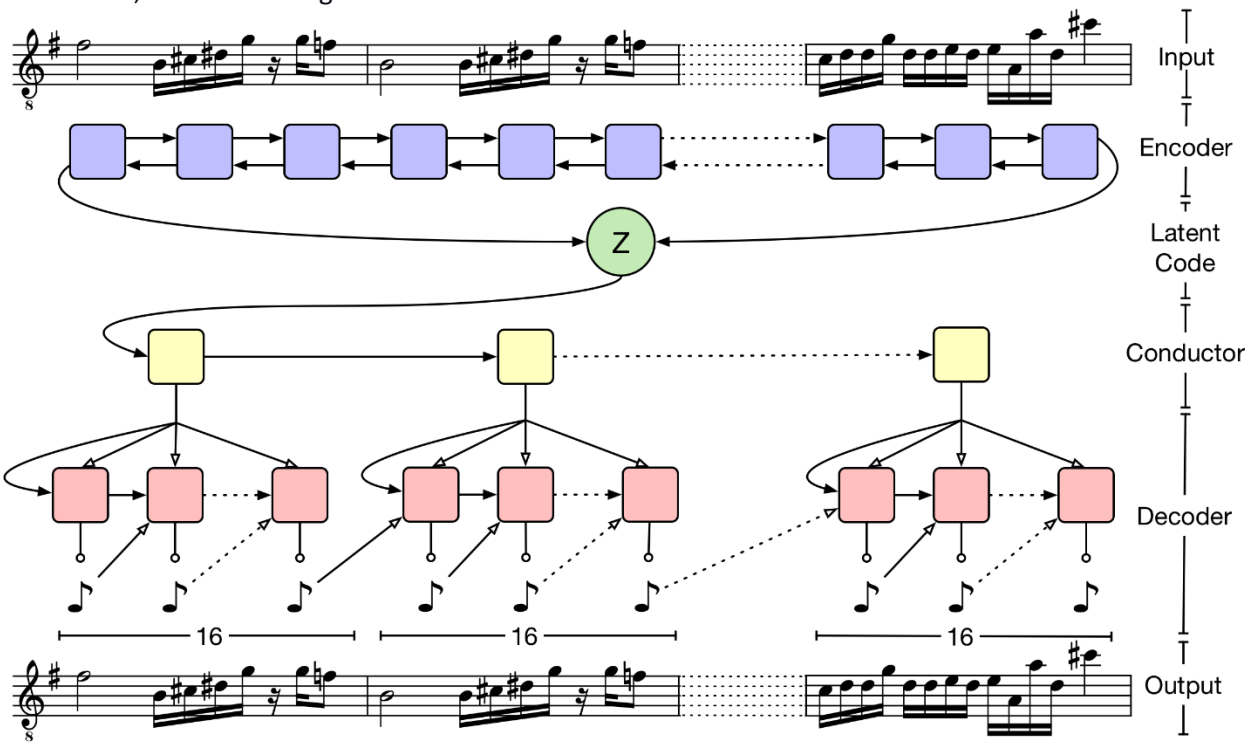

Figure 5: Schematic of hierarchical recurrent Variational Autoencoder model, **MusicVAE**.

Bidirectional Encoder: I use a two-layer bidirectional LSTM network [14] for the encoder. I processed an input sequence to obtain the final state vectors from the second bidirectional LSTM layer. These are then concatenated to produce state vectors and fed into two fully connected layers to produce the latent distribution parameters weight matrices and basis vectors. In my experiments, I used an LSTM state size of 2048 for all layers and 512 latent dimensions. A bidirectional recurrent encoder ideally gives the parametrization of the latent distribution longer-term context about the input sequence.

Hierarchical Decoder: In prior work, the decoder in a recurrent VAE is typically a simple stacked RNN. The decoder RNN uses the latent vector to set its initial state and proceeds to generate the output sequence autoregressively. In preliminary experiments, we found that using a simple RNN as the decoder resulted in poor sampling and reconstruction for long sequences. We believe this is caused by the vanishing influence of the latent state as the output sequence is generated. To mitigate this issue, we propose a novel hierarchical RNN for the decoder. Assume that the input sequence can be segmented into nonoverlapping subsequences with endpoints. Then the latent vector is passed through a fully connected layer followed by a tanh activation to get the initial state of a "conductor" RNN. The conductor RNN

produces embedding vectors, one for each subsequence. In our experiments, we use a two-layer unidirectional LSTM for the conductor with a hidden state size of 1024 and 512 output dimensions. Once the conductor has produced the sequence of embedding vectors, each one is individually passed through a shared fully connected layer, followed by a tanh activation to produce initial states for a final bottom-layer decoder RNN. The decoder RNN then autoregressively produces a sequence of distributions over output tokens for each subsequence via a softmax output layer. At each step of the bottom-level decoder, the current conductor embedding is concatenated with the previous output token to be used as the input. In our experiments, we used a 2-layer LSTM with 1024 units per layer for the decoder RNN. In principle, our use of an autoregressive RNN decoder still allows for the "posterior collapse" problem where the model effectively learns to ignore the latent state. Like [3], it is essential to limit the scope of the decoder to force it to use the latent code to model long-term structure. For a CNN decoder, this is as simple as reducing the receptive field, but no direct analogy exists for RNNs, which in principle, have an unlimited temporal receptive field. To get around this, we reduce the practical scope of the bottom-level RNN in the decoder by only allowing it to propagate a state within an output subsequence. As described above, we initialize each subsequence RNN state with the corresponding embedding passed down by the conductor. This implies that the only way for the decoder to get longer-term context is by using the embeddings produced by the conductor, which in turn depend solely on the latent code. We experimented with an autoregressive version of the conductor where the decoder state was passed back to the conductor at the end of each subsequence but found it exhibited worse performance. We believe that these combined constraints effectively force the model to utilize the conductor embeddings and, by extension, the latent vector to correctly autoencoder the sequence.

Generate Music from human body movement: I generate unique music for every emotion with our MusicVAE model. Save this music in the same folder where I already saved images of emotions from human body movement.

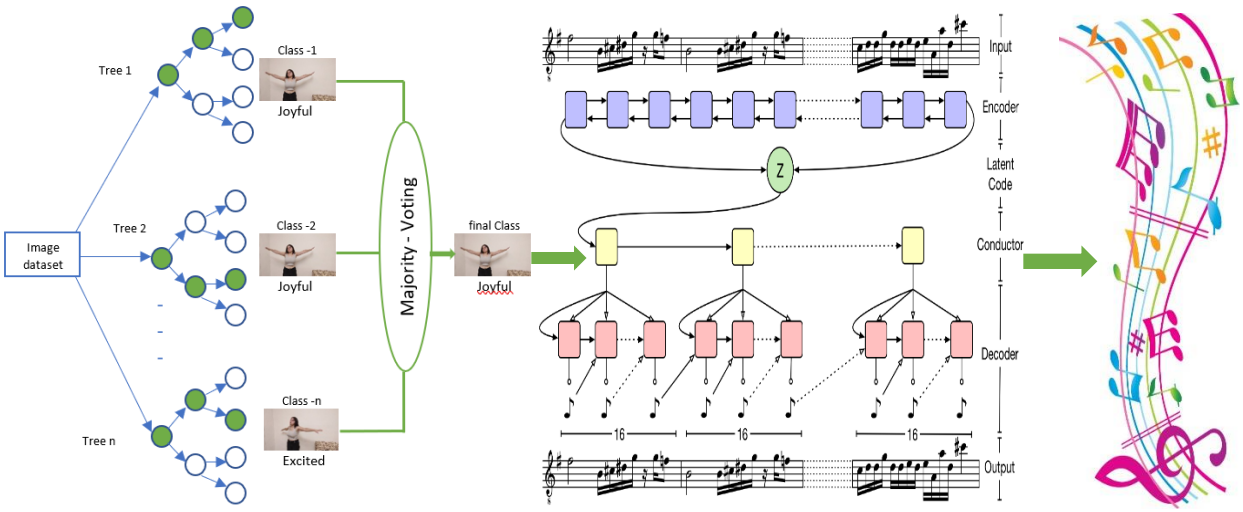

Figure 6: Generate AI music from human body movement.

Now it is time to conduct experiments to generate music from human body movement. For example, a visitor stands in front of my system camera and tries to explain a silent movie that has been watched by the audience before. The visitor should give at least 60 of the same positions (like happy, sad, and so on body position, which I mentioned in figure 2). After explaining the movie with body movements by the

visitor, my system collected emotions and stored them. And find out emotion-related music, which I generated before by the MusicVAE model and combine this music and create one unique piece of music (Figure 6) related to visitor body movement.

References:

[1] Tamara Berg, Debaleena Chattopadhyay, Margaret Schedel, and Timothy Vallier. 2012. Interactive music: Human motion initiated music generation using skeletal tracking by kinect. In Proc. Conf. Soc. Electro-Acoustic Music United States.

[2] Hsuan-Kai Kao and Li Su. Temporally Guided Music-to-Body-Movement Generation, ACM Multimedia 2020

[3] Chen, X., Kingma, D. P., Salimans, T., Duan, Y., Dhariwal, P., Schulman, J., Sutskever, I., and Abbeel, P. Variational lossy autoencoder. In Fifth International Conference on Learning Representations, 2017.

[4] Jane W Davidson. 2012. Bodily movement and facial actions in expressive musical performance by solo and duo instrumentalists: Two distinctive case studies. Psychology of Music 40, 5 (2012), 595–633.

[5] Shiry Ginosar, Amir Bar, Gefen Kohavi, Caroline Chan, Andrew Owens, and Jitendra Malik. 2019. Learning Individual Styles of Conversational Gesture. In IEEE Conference on Computer Vision and Pattern Recognition. 3497–3506.

[6] Facial emotion recognition using deep learning: review and insights. The 2nd InternationalAugust Workshop 9-12,on 2020, the Future Leuven,ofBelgium Internet of Everything (FIoE) August 9-12, 2020, Leuven, Belgium

[7] E. Sariyanidi, H. Gunes, et A. Cavallaro, « Automatic Analysis of Facial Affect: A Survey of Registration, Representation, and Recognition, IEEE Trans. Pattern Anal. Mach. Intell., oct. 2014, doi: 10.1109/TPAMI.2014.2366127

[8] C. Marechal et al., « Survey on AI-Based Multimodal Methods for Emotion Detection, in High-Performance Modelling and Simulation for Big Data Applications: Selected Results of the COST Action IC1406 cHiPSet, J. Kołodziej et H. González-Vélez, Éd. Cham: Springer International Publishing, 2019, p. 307-324.

[9] C. Shan, S. Gong, et P. W. McOwan, « Facial expression recognition based on Local Binary Patterns: A comprehensive study », Image Vis. Comput., vol. 27, no 6, p. 803-816, mai 2009, doi: 10.1016/j.imavis.2008.08.005.

[10] R. Gross, I. Matthews, J. Cohn, T. Kanade, et S. Baker, « Multi-PIE », Proc. Int. Conf. Autom. Face Gesture Recognit. Int. Conf. Autom. Face Gesture Recognit., vol. 28, no 5, p. 807-813, mai 2010, doi: 10.1016/j.imavis.2009.08.002.

[11] Differential Music: Automated Music Generation Using LSTM Networks with Representation Based on Melodic and Harmonic Intervals. Rafraf, Hooman. University of Florida ProQuest Dissertations Publishing, 2022. 29066282.

[12] Choi, Keunwoo, George Fazekas, and Mark Sandler. 2016. “Text-Based LSTM Networks for Automatic Music Composition.” ArXiv:1604.05358 [Cs], April. http://arxiv.org/abs/1604.05358.

[13] Kumar, Harish, and Balaraman Ravindran. 2019. “Polyphonic Music Composition with LSTM Neural Networks and Reinforcement Learning.” ArXiv:1902.01973 [Cs, Eess, Stat], March. http://arxiv.org/abs/1902.01973.

[14] Schuster, M. and Paliwal, K. K. Bidirectional recurrent neural networks. IEEE Transactions on Signal Processing, 45(11):2673–2681, 1997.

[15] “Magenta.” https://magenta.tensor□ow.org/music-vae.

[16] “Musicvae: Creating a pale□e for musical scores with machine learning.” https://magenta.tensor□ow.org/music-vae

[17] Bowman, S., Vilnis, L., Vinyals, O., Dai, A., Jozefowicz, R., and Bengio, S. Generating sentences from a continuous space. In Proceedings of the Twentieth Conference on Computational Natural Language Learning, 2016.