TEACH AN AI TO DANCE USING RECURRENT NEURAL NETWORK.

Nguyen Tan Hoang,
Students,
Bachelor of Engineer, Robotics and Artificial Intelligence, HCMC University of Technology and Education, HCM, Viet Nam 20134015@student.hcmute.edu.vn

Abstract: This paper proposes a technique for teaching AI dancing using regressive neural networks (RNN). In order to discover underlying patterns and relationships in movements, the method is based on training an RNN model on a dataset of dance sequences. We test our approach using a dataset of several dances and show that our AI can produce new dance sequences that are just like those made by humans.
Index terms — Deep Learning, RNN, AI dance

I. INTRODUCTION

The intricate and dynamic activity of dancing calls for the perfect timing of many different body parts. More and more individuals are becoming interested in using AI to create virtual dancers and choreography in recent years. But it's difficult to develop an AI that can dance as well as a human. This paper proposes a technique for training AI to dance using recurrent neural networks (RNN).

The RNN model is trained using a dataset of dance sequences as one of the methods. To extract features that describe the motion of each body part at each time step, the dataset is first processed. The RNN model is then trained to discover the underlying dependencies and patterns in the movements. By using the cross-validation technique, the RNN model's parameters, including the number of hidden layers and units, are tuned.

Part 1-Video Preprocessing: The frames will be taken from a dance video of silhouettes, preprocess them to smaller and simpler, and add them to a zip file in sequence.

Part 2-Autoencoder Compression: To save even more memory, an Autoencoder will be used to divide into a much smaller numpy array.

Part 3-Train model: We will put these compressed frames into sequences and train a model.

Using these techniques for a dancing AI opens up the groundwork for AI to predict on and create all types of different videos.

II. *VIDEO PREPROCESSING*

Processing will be done with a YouTube instructional video. It was a silhouette of a woman dancing. This is perfect because the majority of other dance films are sloppy and difficult to pre-process. This dance does not repeat, albeit it is not extremely varied.

In this phase, every frame from the video will be extracted and put in sequential order to the zip file. Additionally, in order to conserve space and facilitate later processing by models, frames will also undergo the following preprocessing:

- Take every other frame only: Since we don't need every frame, this will lower the amount of frames required later when using the RNN model to view further into the past.

- Grayscale the image: Shadows can be captured without the need of any color channels.

- Image resized to 64 by 96 pixels Because the file is significantly smaller and 64 x 96 can be readily divided, it will be simpler to structure the autoencoder later on without any data being lost.

- Make it binary by changing the lighter and darker pixels to white and black, respectively. The input for the most crucial pixels is made simpler as a result.

```
for n in range(5,15):
    imshow(X_train[n].reshape(IMG_HEIGHT,IMG_WIDTH))
    plt.show()
```

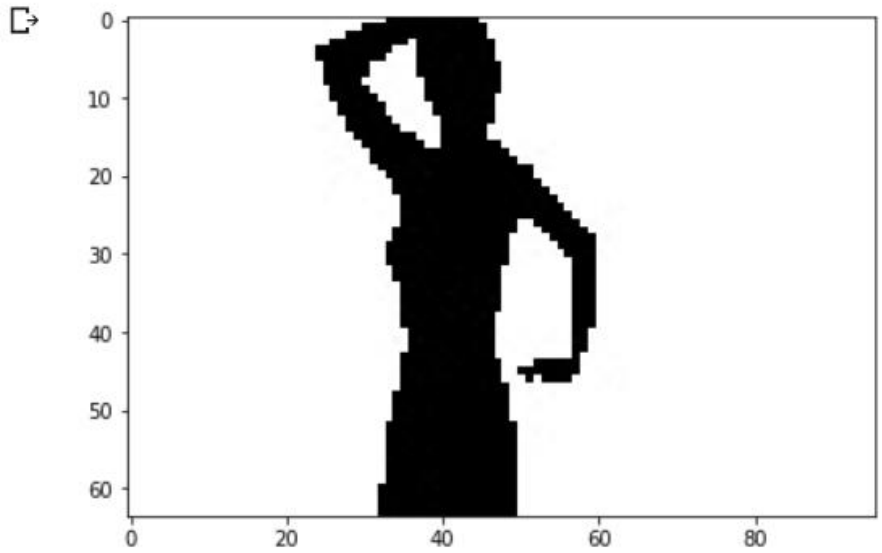

Fig 1 : Original data

III. AUTOENCODER COMPRESSION

The frames from the shadow dancer video must still be further compressed after being pre-processed in order for the frames to suit the RNN model. The creation of specific compression models is one of the many applications for autoencoders. In this step, we'll utilize the dance photos to train an autoencoder, which we'll then employ to compress the images into a much smaller array.

To aid in compression and noise reduction, the autoencoder is designed to recreate the input.
The Input() function is used to define the model's input shape at the beginning of the function. Then it uses Conv2D to produce several convolutional layers (). In order to extract features, this function applies a number of filters to the input. Elu (Exponential Linear Unit), which is used in these layers as an activation function, is comparable to relu but increases the risk of neuronal death and negative input. Finally, using Dense(), which applies a linear modification to the input, the output of the convolution is flattened and passed through a number of dense layers. The input, which consists of 128 units, is encoded in the final dense layer.

After that, a number of additional dense layers are applied to the encoded representation in order to rebuild the original input. To restore the encoded representation to its original input shape, use the Reshape() function. Conv2DTranspose() is used to apply the deconvolution layers in order to make the feature maps more spatially expansive. The learning process is then configured by compiling the model using the compile() function.

```
def Autoencoder():
    inp = Input(shape=INPUT_SHAPE)
    x = Conv2D(128, (4, 4), activation='elu', padding='same',name='encode1')(inp)
    x = Conv2D(64, (3, 3), activation='elu', padding='same',name='encode2')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='elu', padding='same',name='encode3')(x)
    x = Conv2D(32, (2, 2), activation='elu', padding='same',name='encode4')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='elu', padding='same',name='encode5')(x)
    x = Conv2D(32, (2, 2), activation='elu', padding='same',name='encode6')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(64, (3, 3), activation='elu', padding='same',name='encode7')(x)
    x = Conv2D(32, (2, 2), activation='elu', padding='same',name='encode8')(x)
    x = MaxPooling2D((2, 2), padding='same')(x)
    x = Conv2D(32, (3, 3), activation='elu', padding='same',name='encode9')(x)
    x = Flatten()(x)
    x = Dense(256, activation='elu',name='encode10')(x)
    encoded = Dense(128, activation='sigmoid',name='encode11')(x)
    x = Dense(256, activation='elu', name='decode1')(encoded)
    x = Dense(768, activation='elu', name='decode2')(x)
    x = Reshape((4, 6, 32))(x)
    x = Conv2D(32, (2, 2), activation='elu', padding='same', name='decode3')(x)
    x = Conv2D(64, (3, 3), activation='elu', padding='same', name='decode4')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(32, (2, 2), activation='elu', padding='same', name='decode5')(x)
    x = Conv2D(64, (3, 3), activation='elu', padding='same', name='decode6')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (2, 2), activation='elu', padding='same', name='decode7')(x)
    x = Conv2D(128, (3, 3), activation='elu', padding='same', name='decode8')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (2, 2), activation='elu', padding='same', name='decode9')(x)
    x = Conv2D(64, (4, 4), activation='elu', padding='same', name='decode10')(x)
    x = Conv2D(128, (3, 3), activation='elu', padding='same', name='decode11')(x)
    x = UpSampling2D((2, 2))(x)
    x = Conv2D(64, (4, 4), activation='elu', padding='same', name='decode12')(x)
    x = Conv2D(32, (3, 3), activation='elu', padding='same', name='decode13')(x)
    x = Conv2D(16, (2, 2), activation='elu', padding='same', name='decode14')(x)
    decoded = Conv2D(1, (2, 2), activation='sigmoid', padding='same', name='decode15')(x)
    return Model(inp, decoded)
```

Fig 2: Create the autoencoder code.

IV. TRAIN MODEL AND RESULTS

A dataset of several dancing moves was used to assess this strategy. The results demonstrate that the AI can produce new dance sequences that are comparable to those made by humans after little over two hours of training with 20 training sessions. The accuracy of the model is measured as another way in which the article assesses the effectiveness of its artificial intelligence. The motions that result are also incredibly fluid, comparable to those of a dancer.

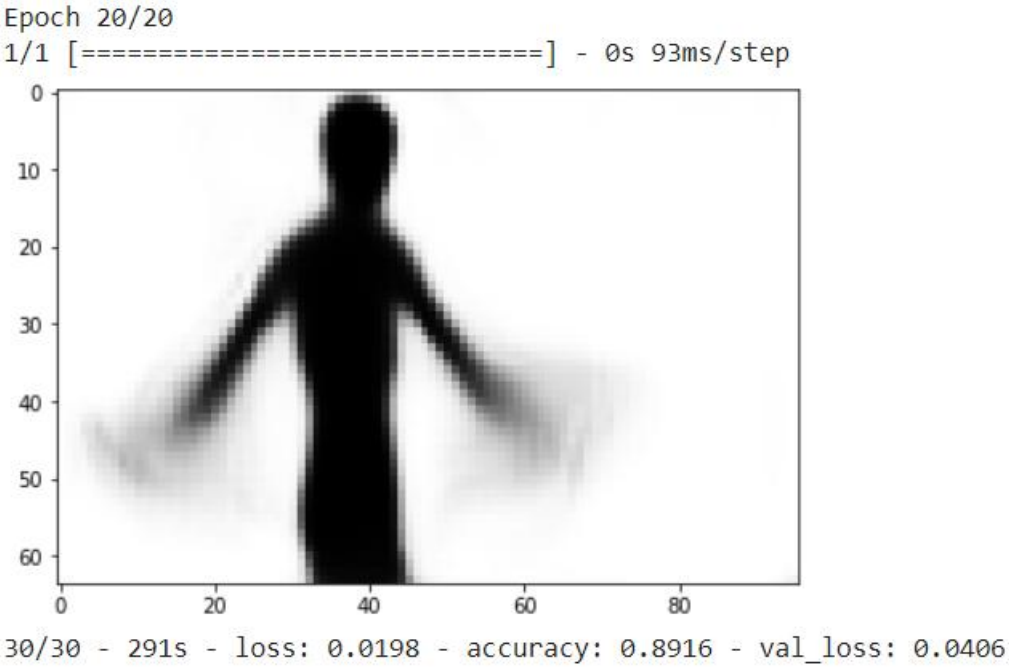

Fig 3: The model's final output.

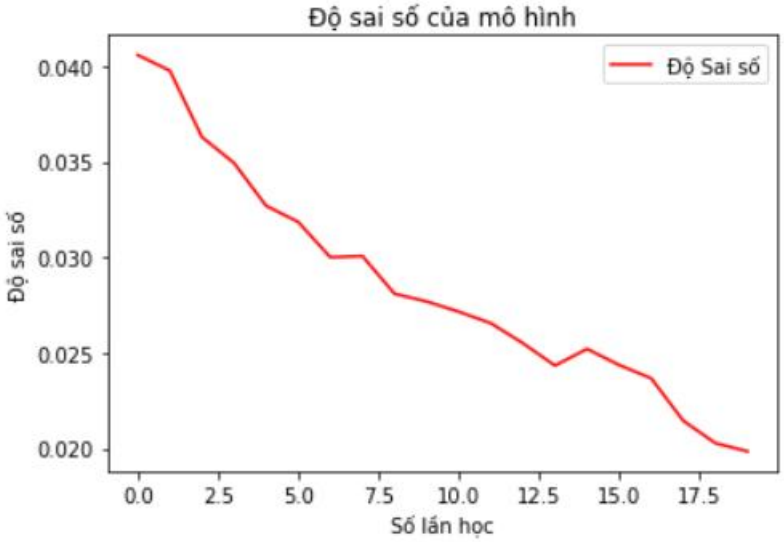

Fig 4: The graph displays the model's error

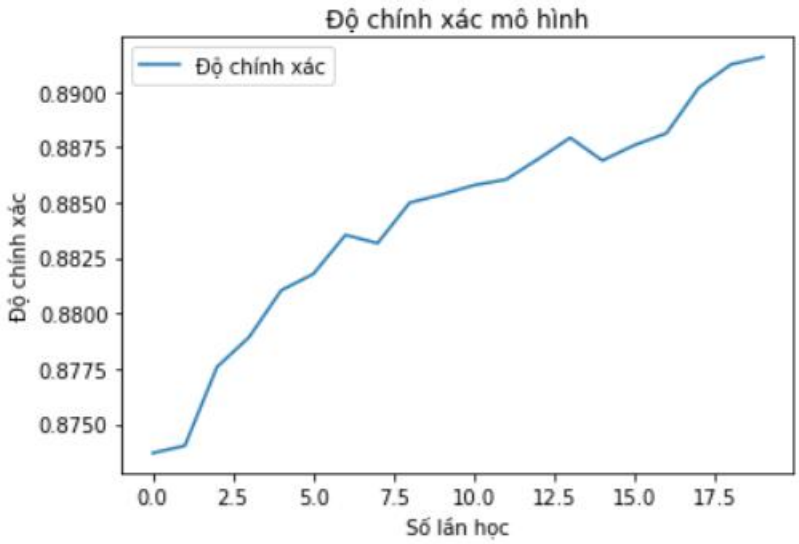

Fig 5: The graph displays the model's accuracy.

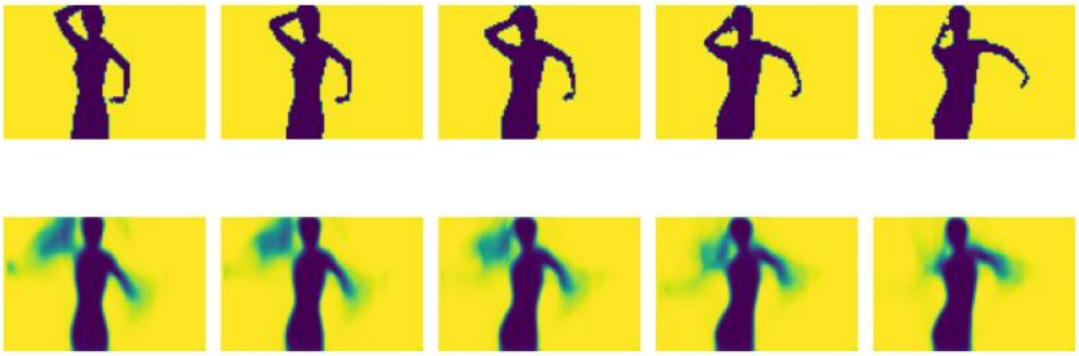

Fig 6: The model's forecast is the outcome of the provided 5 pictures

V. CONCLUSION

In this paper, recurrent neural networks are used to construct a way of teaching AI to dance (RNN). The approach is based on learning underlying patterns and dependencies in movement by training an RNN model on a dataset of dance sequences. We have shown that our AI can develop new dance sequences that are comparable to human-made sequences and perform well in a range of dance genres. In the future, fresh and innovative choreography can be made using the suggested method.

References:

1. Xin Liu, and Young Chun Ko. ”The use of deep learning technology indance movement generation” https://doi.org/10.3389/fnbot.2022.911469.
2. Xuan Ma, and Kai Wang . “Dance Action Generation Model Based on Recurrent Neural Network” https://doi.org/10.1155/2022/8455961.