DEEP DANCE (SHOWCASING THE CAPABILITIES OF RNN AND CNN ON A VIDEO DATASET)

Akash Raj
Student, Department of Computer Science and Engineering, Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi
Saurav Sharma
Student, Department of Computer Science and Engineering, Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi
Yogesh Rauthan
Student, Department of Computer Science and Engineering, Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi
Sagarika Debnath
Student, Department of Computer Science and Engineering, Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi
Nishi Sharma
Assistant professor, Department of Computer Science and Engineering, Dr. Akhilesh Das Gupta Institute of Technology & Management, New Delhi

Abstract

Using a combination of Convolution Neural Network and Recurrent Neural Network to generate new dance moves to showcase the capabilities of CNN and RNN on a video dataset. Simple RNN used with combination of a CNN can be used to predict the next frame of video given few frames of video. The technique used in this research/project can be used to extend an incomplete video or to generate videos of similar type, given that the video is not very complex. This research showcases a simple Machine Learning architecture that can be used to achieve some interesting results on a simple video based dataset.

Keywords– Deep Learning, RNN, CNN, Autoencoders, Video

Introduction

Machine learning has many applications, one of which is to forecast time series, and videos are nothing but a series of frames arranged in order of time. The idea behind this project/research is

to showcase the capabilities of CNN[2][4][5] and RNN[3][5] using a simple video based dataset. The main motive of this project the same as any time series prediction project but instead of predicting numbers we will predict the next frame if given a sequence of frames from a video. Dataset used in this project was a 1hr20min video of Silhouette dance which was used to train an autoencoder and RNN. The trained model was then used to generate dance videos of similar type. The model used in this project can be used to generate videos of similar type as that of the training dataset which in this case was a Silhouette dance. In this project we used the trained model to continue a dance video if given a few frames of the video and also generated new dance videos from scratch.

Implementation

Technical Preparation for Autoencoder

Creating Training/Testing Data

Training data is generated from a 1hr20min long video of Silhouette/Shadow Dancing 3 frames are extracted every second using FFmpeg[8] utility and hence giving us 14400(80x60x3) image sequence of dance movements which we would then clean up to create our dataset.

Cleaning Data

The images extracted from the video have a very dynamic changing background that has nothing to do with the dancer and doesn’t carry any relevant information, so we'll go ahead and removed the background from each frame and made the background white such that only dancer is present in the frame. Then we resize the image to 480x800 for easier calculation. We further reduced the image size by a ratio of 4 just to make things simpler, and hence, we end up with 14400 images of size 120x200. The 14400 cleaned images with the removed background will act as our Training data.

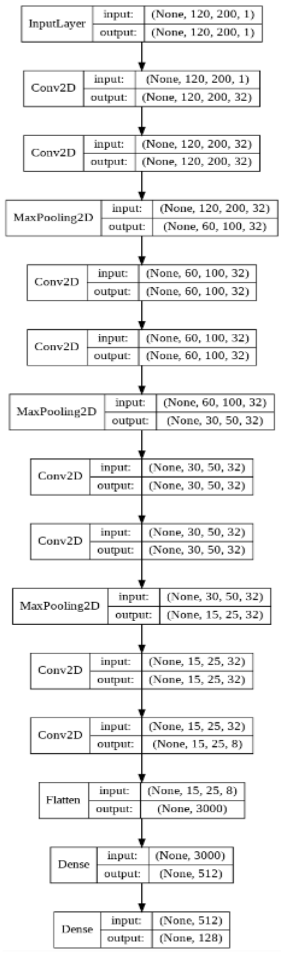
Model Creation

Models were created and trained using Keras with TensorFlow backend. We created two models for this project. The first model is an autoencoder which is a combination of encoder and decoder network. Autoencoder consists of 8 Convolution operation and 9 Deconvolution operations with relu[1] activation function used for activation.

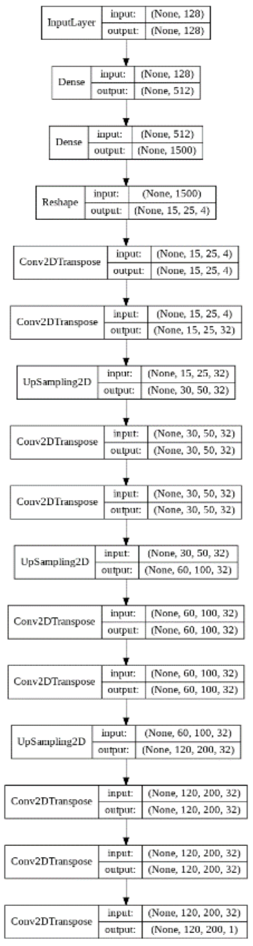
Encoder was designed to reduce 120x200image into a 128x1 vector (ref Fig 1.1 for encoder architecture). Decoder was designed to recreate image from 128x1 vector to a size of 120x200.(ref Fig 1.2 for decoder architecture).

Training

Autoencoder model was trained on 14401 images of dimension 120x200 and only after 27 epochs we achieved an accuracy of 96.4% with a loss of 0.7%



InputLayer input: (None, 120, 200, 1) output: (None, 120, 200, 1)
Conv2D input: (None, 120, 200, 1) output: (None, 120, 200, 32)
Conv2D input: (None, 120, 200, 32) output: (None, 120, 200, 32)
MaxPooling2D input: (None, 120, 200, 32) output: (None, 60, 100, 32)
Conv2D input: (None, 60, 100, 32) output: (None, 60, 100, 32)
Conv2D input: (None, 60, 100, 32) output: (None, 60, 100, 32)
MaxPooling2D input: (None, 60, 100, 32) output: (None, 30, 50, 32)
Conv2D input: (None, 30, 50, 32) output: (None, 30, 50, 32)
Conv2D input: (None, 30, 50, 32) output: (None, 30, 50, 32)
MaxPooling2D input: (None, 30, 50, 32) output: (None, 15, 25, 32)
Conv2D input: (None, 15, 25, 32) output: (None, 15, 25, 32)
Conv2D input: (None, 15, 25, 32) output: (None, 15, 25, 8)
Flatten input: (None, 15, 25, 8) output: (None, 3000)
Dense input: (None, 3000) output: (None, 512)
Dense input: (None, 512) output: (None, 128)



InputLayer input: (None, 128) output: (None, 128)
Dense input: (None, 128) output: (None, 512)
Dense input: (None, 512) output: (None, 1500)
Reshape input: (None, 1500) output: (None, 15, 25, 4)
Conv2DTranspose input: (None, 15, 25, 4) output: (None, 15, 25, 4)
Conv2DTranspose input: (None, 15, 25, 4) output: (None, 15, 25, 32)
UpSampling2D input: (None, 15, 25, 32) output: (None, 30, 50, 32)
Conv2DTranspose input: (None, 30, 50, 32) output: (None, 30, 50, 32)
Conv2DTranspose input: (None, 30, 50, 32) output: (None, 30, 50, 32)
UpSampling2D input: (None, 30, 50, 32) output: (None, 60, 100, 32)
Conv2DTranspose input: (None, 60, 100, 32) output: (None, 60, 100, 32)
Conv2DTranspose input: (None, 60, 100, 32) output: (None, 60, 100, 32)
UpSampling2D input: (None, 60, 100, 32) output: (None, 120, 200, 32)
Conv2DTranspose input: (None, 120, 200, 32) output: (None, 120, 200, 32)
Conv2DTranspose input: (None, 120, 200, 32) output: (None, 120, 200, 32)
Conv2DTranspose input: (None, 120, 200, 32) output: (None, 120, 200, 1)

Fig 1.1. Encoder Architecture, Fig 1.2 Decoder Architecture

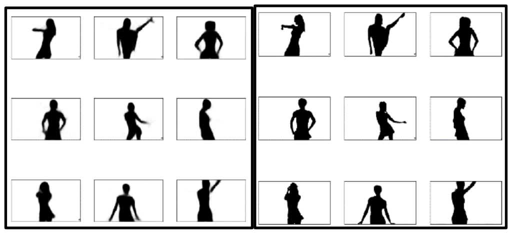


Fig 2. Original Image, Fig 3. Recreated image by autoencoder

Technical Preparation for RNN

Creating Training/Testing Data

We will use 14401 samples of 128 dims output from the encoder network as training data for our RNN model.

Arranging Data

The 14401x128 data needs to be pre-processed and to make it ready to feed into RNN. To do that, we will make our input data as batches of 10 frames and the corresponding output data will be the 11th frame. After rearranging, we will get the input data dims as 14401x10x128 and output data dims as 14401x128.

Model Creation

Models were created and trained using Keras with TensorFlow backend. The second Model is an RNN, which is used to learn the pattern between those 10x128 vectors(10 frames of video) and then create a new sequence of 128 dims vectors(11th frame of video), which can be transformed back into images using the decoder network. RNN consists of 4 LSTM[3][7] layers with dropout of 20% to prevent overfitting[6] .(ref Fig 2 for RNN architecture).

Training

The RNN model was trained over an input of shape 14401x10x128 and achieved an accuracy of 80.0% in just 853 epochs.

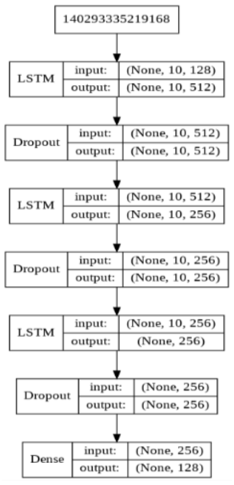

140293335219168
LSTM | input: (None, 10, 128) | output: (None, 10, 512)
Dropout | input: (None, 10, 512) | output: (None, 10, 512)
LSTM | input: (None, 10, 512) | output: (None, 10, 256)
Dropout | input: (None, 10, 256) | output: (None, 10, 256)
LSTM | input: (None, 10, 256) | output: (None, 256)
Dropout | input: (None, 256) | output: (None, 256)
Dense | input: (None, 256) | output: (None, 128)

Fig 4. RNN architecture

Procedure

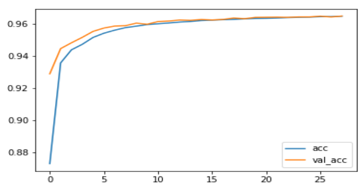
The result is generated by either using a random seed of 1x10x128 dimensions fed to RNN, or by using any 10 continuous sequences of frames (in dense format of 128 dims vector) from the training dataset, which is reshaped to 1x10x128 as an input to RNN.

If we wish to create a dance video from scratch we use the random seed of 1x10x128, and if we wish to continue an existing dance video then we must use the last 10 frames of video (in dense format) as the seed.[4]

The 128 dims output of RNN is appended to the previous input, and the first sequence is popped from previous input. The new modified input is then passed to RNN and the cycle continues. The 128 dims output is the new sequence for a new dance.

The new 128 dims output from RNN is passed through the decoder to generate new image frames of size 120x200, which are then stitched together to make a new video.

Metrics



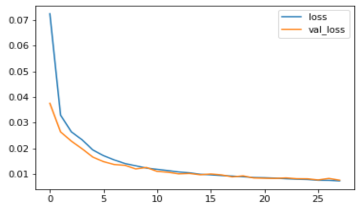


Fig 5.Autoencoder accuracy and loss respectively

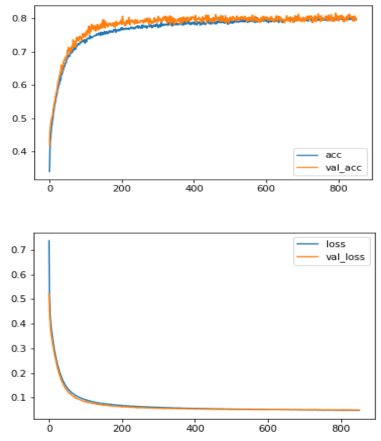


Fig 6.RNN accuracy and loss respectively

Result and discussion

The autoencoder model achieved an accuracy of 96.4% and a loss of 0.7% in just 27 epochs.

The RNN model achieved an accuracy of 80.02% and a loss of 4.8% in 853 epochs.

The output dance generated using this method were convincingly real and this model can be used with different dataset to get different interesting results.

Model	**epoch**	**acc**	**loss**	**val_acc**	**val_loss**
Autoencoder	27	0.964	0.007	0.964	0.007
RNN	853	0.800	0.048	0.799	0.049

Dataset used :Techno/Tech House Mix by DJ Haluk Arslan with Shadow Dancershttps://www.youtube.com/watch?v=NdSqAAT28v0

The code and the result is publicly available at :https://github.com/akashraj9828/Deep-Dance

Output can be viewed at :

https://github.com/akashraj9828/Deep-Dance/blob/master/result.gif?raw=true

Conclusion

Using a combination of CNN and RNN we achieved reasonable well results in generating dance moves using artificial intelligence.

We achieved 96% accuracy with auto-encoder model and 80% accuracy with the recurrent neural network model. The output produced was realistic and, this project showcased the possibilities of CNN and RNN on a video dataset.

References

1. Romanuke, Vadim (2017). "Appropriate number and allocation of ReLUs in convolutional neural networks" (PDF). Research Bulletin of NTUU "Kyiv Polytechnic Institute".
2. Deshpande, Adit. "The 9 Deep Learning Papers You Need To Know About (Understanding CNNs Part 3)". adeshpande3.github.io. Retrieved 2018-12-04.
3. A. Sherstinsky, "Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network", CoRR, vol. abs/1808.03314, 2018.
4. Borovykh, Anastasia; Bohte, Sander; Oosterlee, Cornelis W. (2018-09-17). "Conditional Time Series Forecasting with Convolutional Neural Networks". arXiv:1703.04691[stat.ML].
5. Bai, Shaojie; Kolter, J. Zico; Koltun, Vladlen (2018-04-19). "An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling". arXiv:1803.01271 [cs.LG].
6. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". jmlr.org. Retrieved 2015-12-17.
7. Sepp Hochreiter; Jürgen Schmidhuber (1997). "Long short-term memory".
8. "ffmpeg Documentation". https://ffmpeg.org/ffmpeg.html