Video Predictions

A Project Report Submitted
in Partial Fulfillment of the Requirements
for the Degree of

Bachelor of Technology

by

Rakesh Kumar
(111701024)

COMPUTER SCIENCE AND ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY PALAKKAD

CERTIFICATE

*This is to certify that the work contained in the project entitled "**Video Predictions**" is a bonafide work of **Rakesh Kumar (Roll No. 111701024)**, carried out in the Department of Computer Science and Engineering, Indian Institute of Technology Palakkad under my guidance and that it has not been submitted elsewhere for a degree.*

Dr. Chandra Shekar Lakshminarayanan

Assistant Professor

Department of Computer Science & Engineering

Indian Institute of Technology Palakkad

Contents

List of Figures

Chapter 1

Introduction

The ability to predict or anticipate about future outcomes from video predictions can have many applications in several domains. For eg., Videos of radar images can be used for weather nowcasting, Action predictions from videos, etc. Developing a model that identifies the moving objects, understands their correlation from visual scenes and predicts their future trajectories in an image scene is challenging. Deep learning based video prediction is an interesting research problem. Video prediction can also be seen as a sequence-to-sequence problem where the sequence consists of image frames.

Recent advances in deep learning, especially RNNs (more precisely LSTMs) are used for various sequence-to-sequence prediction problem. The LSTM encoder-decoder architecture is a general framework used in various sequence-to-sequence learning problems such as machine translation, etc. The NeurIPS 2015 [2] paper provides the ConvLSTM (Convolutional LSTM) Encoder-Decoder Network based model for this video predictions problem. LSTMs based models work well but the requirement of processing data in sequential order doesn't give much scope for parallelization during training.

A recent deep learning model, **Transformers** [3], utilizes the mechanism of attention (similar to cognitive attention, will be explained later) have rapidly become the model of choice for NLP problems. This model provide scope for parallelization and thus reduces training time. In this project, I am trying to modify the existing Transformers model to make it work for the video prediction problem as well as compare its performance with the ConvLSTM based models.

1.1 Organization of The Report

This chapter provides a brief introduction about the importance of video predictions and tells about some deep learning models currently in use for the same. Chapter 3 explains the Transformers model for natural language processing related tasks. The Chapter 4 talks about the modification needed in the Transformers model for NLP so that it can be used for the video predictions by drawing analogy between LSTMs and ConvLSTMs. Chapter 5 compares the performance of the ConvLSTM model with the transformer model on the Moving MNIST dataset. Chapter 6 mainly shows the results of the predictions on the UCF101 dataset by the ConvLSTM model. The final chapter tells about the work going on now and what should be done in the future.

Chapter 2

Review of Prior Works

2.1 Unsupervised Learning of Video Representations using LSTMs [6]

The paper proposes variants of LSTM Encoder-Decoder model : LSTM Autoencoder Model, LSTM Future Predictor Model, Composite Model etc. The paper compares the performance of various models on moving MNIST dataset, UCF-101 dataset, etc.

2.2 Deep Learning for Weather and Climate Science [1]

This was my project in BTP-I. It was an implementation of Encoder-Decoder Network consisting of Convolutional LSTM layers as described in the paper : **Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting** [2].

Encoder-Decoder network is a general model used for sequence-to-sequence prediction problems (eg., Machine translation). Encoder-Decoder network consists of two networks: Encoder and Decoder Network. These networks may be composed of several layers of other networks such as RNN based network (LSTMs), etc. Encoder processes the input sequence and returns vectors also known as states. Decoder uses the states returned by the Encoder and generates the output sequence of required length.

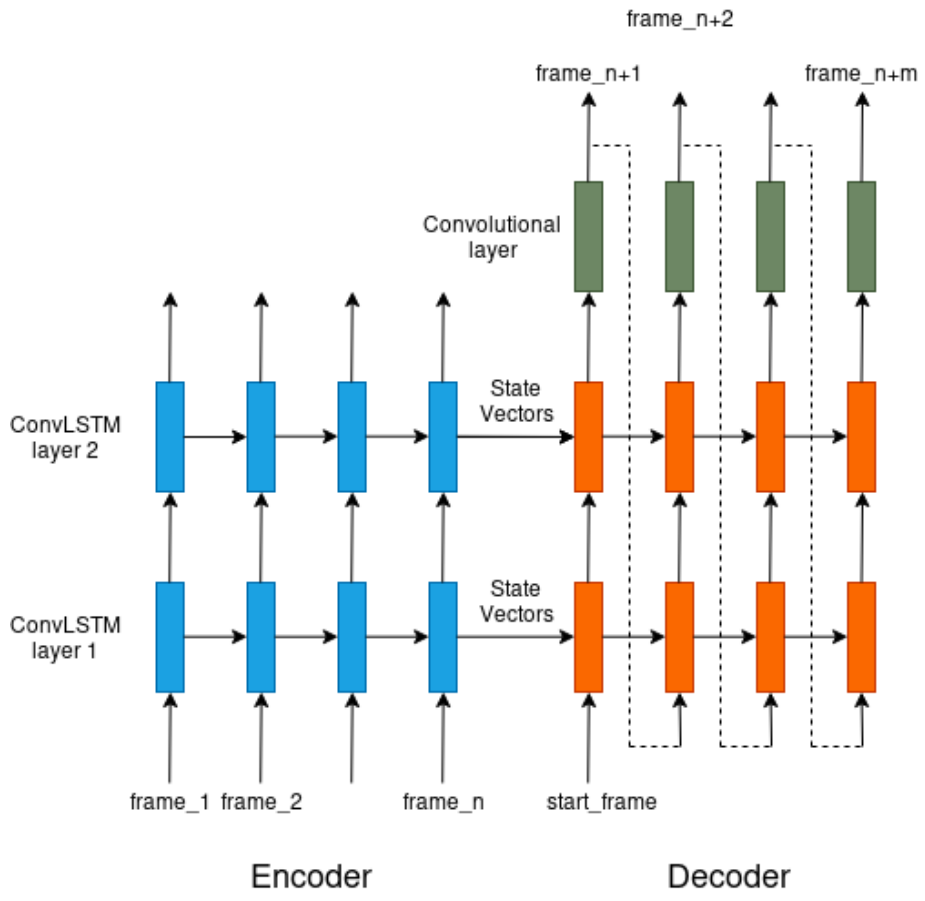

Fig. 2.1 2 layer Encoder-Decoder Structure

In this project, Encoder and Decoder Networks consisted of ConvLSTMs layers. This Encoder-Decoder Network was fed a sequence of NEXRAD radar images and future predicted

radar images were generated. The input radar images were collected from a short duration (around 1 hour) and the predicted radar images helped to analyze the future weather conditions for a short duration (around 1 hour). The model was also tested on some artificial datasets like Moving-MNIST dataset. Though the model required very high GPU memory and CPU time, it worked well on the above datasets.

This model is also used for Video Predictions. Due to its sequential nature, it can't be trained parallely to improve the training time. Also, it can only be used for low resolution images as on higher resolution, the model consumes very high GPU memory which is not suitable for normal machines.

Chapter 3

Transformer [3]

3.1 Background

Recurrent Neural Networks (RNNs) are well suitable for classifying, processing and making predictions based on time series data. In theory, RNNs can learn long-term dependencies in sequence-to-sequence problems (eg., Natural Language Processing) but in practice it doesn't seem to be able to learn. Later, LSTM Networks were introduced which is a special kind of RNNs capable to learn long-term dependencies much better than RNNs. Though LSTM Networks work well in practice, it requires the data to be processed in sequential order which gives no scope for parallelization to reduce the training times.

The Transformer is a deep learning model that utilizes the mechanism of attention. In the context of neural networks, **attention** is a technique that mimics cognitive attention. Cognitive attention [4] is the behavioral and cognitive process of concentrating on a certain aspect of information while ignoring other percievable information. The attention mechanism in neural networks, enhances the relevant and important parts of the input and fades out the unrelevant parts. The attention mechanism learns which parts are important through training.

The advantage of Transformers over the Recurrent Neural Networks is that it doesn't require the sequential data to be processed in order. This feature gives more scope for parallelization during training than RNNs and therefore reduces the training time. There are networks where attentions are used along with LSTMs but in practice it is found that attention mechanisms is sufficient alone.

3.2 Model Architecture

Like the other LSTM based models, the Transformer is also an Encoder-Decoder Architecture. The Encoder consists of several layers, one on top of another. Each encoder layer processes its input to generate the encodings, containing information about which parts of inputs are relevant to each other. These encodings are passed to the next encoder layer as inputs for further processings. The final encodings from the top layer is passed to the Decoder network. The Decoder also has several layers and each layer takes encoder outputs along with its inputs. The Decoder processes the encodings and using their contextual information, generates an output sequence. To find out the relevance of each part of the input with every other parts in the input, each encoder and decoder layer uses the attention mechanism. For eg., *He went to the doctor because **he** was unwell.* Here, ***he*** refers to the person not the doctor. So, the word ***he*** is much more relevant to ***He*** than the ***doctor*** in the sentence. The attention mechanism finds out this and other types of relevance for each part of the inputs.

3.2.1 Attention

Attention mechanism relates different positions of a sequence in order to compute a mathematical representation of the sequence. An attention function is a mapping of a query and a set of key-value pairs to an output. These query, keys, values and output are vectors. The output is computed as a weighted sum of the values, where the weight multiplied to each value depends on the relevance of the query with the corresponding key. The intuition is that attention finds the relevant values of the given query by comparing it with the given list of keys and from the key, the associated value is obtained.

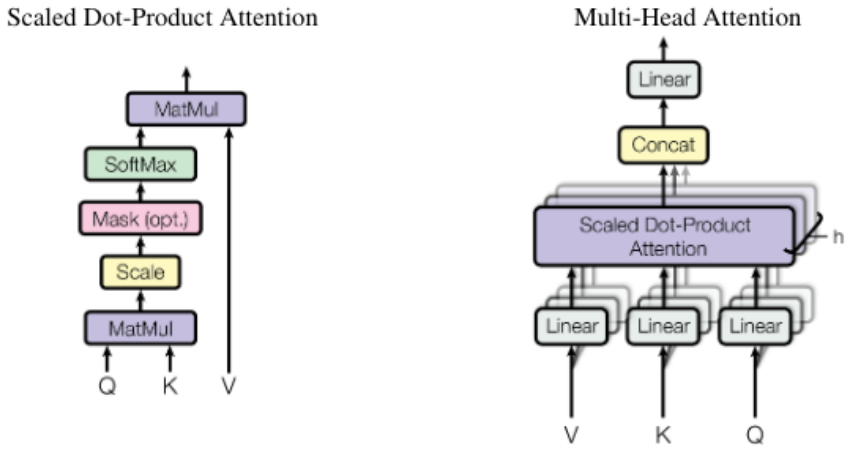

Fig. 3.1 (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several ttention layers running in parallel. [3]

3.2.2 Scaled Dot-Product Attention [5]

It is the basic building blocks of the Transformer. It performs a dot product between each pair of query and the key and finds out the relevant part or information for each query.

Let's consider an example from Natural Language Processing. The inputs to the Transformer is a sequence of word embeddings x_i. Each attention unit learns 3 weight matrices, the query weight matrix W_Q, the key weight matrix W_K and the value weight matrix W_V. Using the weight matrices, for each input word x_i, a query vector $q_i = x_i W_Q$, a key vector $k_i = x_i W_K$ and a value vector $v_i = x_i W_V$ are computed. For a q_i vector, the attention weights for each value v_j is computed by taking the dot product of q_i with every other k_i vector. These dot products or attention weights are divided by $\sqrt{d_k}$ (d_k is the dimension of the key vector) to stabilize the gradient while training. These products are further normalized using softmax function and the final output for ith input word is computed by taking the weighted sum of all the value vectors weighted by their attention weight.

The attention calculation for all tokens or words can be expressed as one large matrix calculation. The same above computation is expressed as matrix multiplication below. The matrices Q, K and V have q_i, k_i and v_i in their ith row respectively. The matrix calculations are useful for training due to computational matrix optimizations.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{3.1}$$

3.2.3 Multi-Head Attention

One set of (W_Q, W_K, W_V) matrices is called an attention head and in multi-head attention, there are multiple such heads or scaled dot-product attention. The output from each head is finally concatenated and multiplied with a learnable matrix W^O. In practice, multi-head performs better as the model can now learn for each input, the different definitions of "relevance" compared to single definition in the case of single head which results in significant improve in performance.

$$\begin{aligned} MultiHead(Q, K, V) &= Concat(head_1, .., head_h)W^O \\ \text{where } head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{3.2}$$

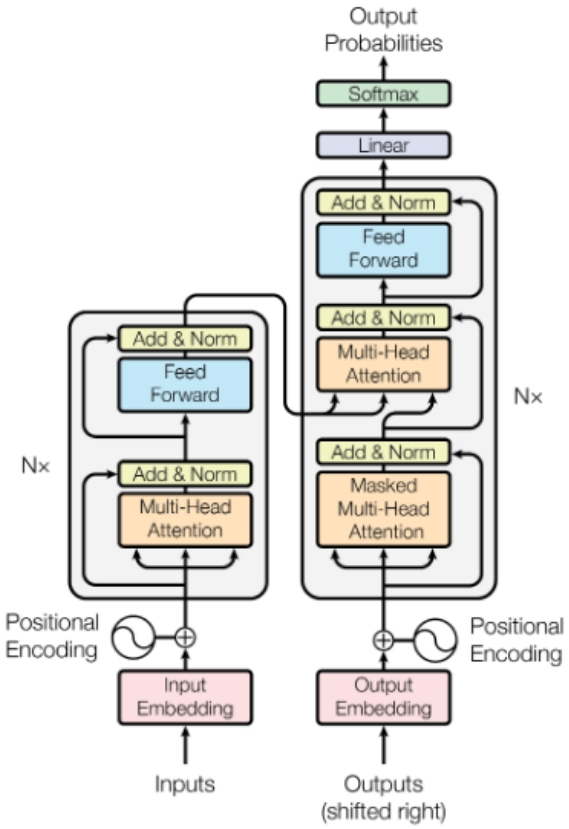

Fig. 3.2 One layer Transformer - model architecture [3]

3.2.4 Encoder

Each layer in Encoder is build from 2 major components : a multi-head self-attention mechanism, and a feed-forward neural network. The self-attention mechanism takes a set of input encodings from the previous encoder layer and weighs their relevance to each other to produce a set output encodings. The feed-forward neural network further processes each output encoding and passes it to the next encoder as its input. The final encodings from the top layer is passed to each decoder layer.

To make use of the order of the inputs in the sequence, the first encoder layer takes the positional information along with input embeddings. Positional encodings injects the relative or absolute position of the tokens in the sequence. In many work, each dimension of the postional encodings corresponds to a sinusoid. For eg.,

$$\begin{aligned} PE(pos, 2i) &= \sin(\frac{pos}{10000^{2i/d_{model}}}) \\ PE(pos, 2i+1) &= \cos(\frac{pos}{10000^{2i/d_{model}}}) \end{aligned} \tag{3.3}$$

where pos is the position, i is the dimension and d_{model} is the dimension of a final output from the encoder encodings. This positional encodings vectors are added to the input embedding to inject the positional information. The authors of the paper used the above positional encodings.

3.2.5 Decoder

Each layer in Decoder is consists of 3 major components : a multi-head self-attention mechanism (over decoder inputs), a multi-head attention (over encoder and decoder encodings)and a feed-

forward neural network. The decoder layer functions similarly like the encoder layer except it has one more attention mechanism which draws relevant information from encoder encodings as well.

The final output from the top decoder layer is followed by linear transformation and softmax layer to produce proabilites over the vocabulary. Since, the outputs of decoder is also fed to the decoder, positional encodings is added into the outputs of the decoder before feeding again to the decoder network.

Chapter 4

Transformers for Video Predictions

4.1 Background

Video is just a sequence of image frames. Video predictions comes under the sequence-to-sequence problems where the input is a sequence of continuous image frames and the expected outcome is also a sequence of image frames or video for some short duration. There are some models used for Video Predictions and most of them uses LSTM based models. In my BTP-I, we used ConvLSTM based Encoder-Decoder Model. The model worked well but as noted earlier, it consumed a lot of memory and time during training. The memory consumption was due to weights in the ConvLSTMs and the sequential nature of the input didn't allow much parallelization to reduce the training time. Here, we are trying to use the transformers which have rapidly become a popular model of choice for sequence-to-sequence problems such as NLP.

4.2 Modifications to the Transformers

Drawing the analogy between the LSTMs and ConvLSTMs, the LSTMs are suitable for sequential problems and to incorporate the spatial information among the images input, the equations of the LSTMs were modified. The major change was the replacement of matrix multiplications with the convolutional operations.

4.2.1 Image Representation

In the original transformer model used in natural language processing, the input words are embedded into d_{model} dimensional vectors before being feed to the network. These word embeddings have similar representation for same meaning words.

Here, in the video prediction model, the input images ($\in \mathbb{R}^{rows \times cols \times channels}$) are transformed to images with the third dimension equal to d_{model} ($\in \mathbb{R}^{rows \times cols \times d_{model}}$). The images are transformed by performing convolution operations on the images with d_{model} filters or kernels.

In the implementation, a one layer CNN with d_{model} filters and $relu$ activation is used for the transformation.

4.2.2 Scaled Dot-Product Attention

The inputs to the Transformer is a sequence of image frames f_i. Each attention unit learns 3 set of convolution kernels, the query kernel W_Q, the key kernel W_K and the value kernel W_V. Using the kernels, for each input frame f_i, a query vector $q_i = f_i * W_Q$, a key vector $k_i = f_i * W_K$ and a value vector $v_i = f_i * W_V$ are computed. Here, $'*'$ is the convolutional operator. The vectors, q_i, v_i, and k_i, are 3D vectors with last two dimensions being rows and columns ($\in \mathbb{R}^{rows \times cols \times h}$, the other dimension h depends on the number of kernels). For a q_i vector, the attention weights for each value v_j is computed by taking the dot product of q_i with every other k_i vector. These dot products or attention weights are divided by $\sqrt{d_k}$ (d_k is the channel dimension of the key vector) to stabilize the gradient while training. These products are further normalized using softmax function and the final output for ith input frame is computed by taking the weighted sum of all the value vectors weighted by their attention weight.

4.2.3 Feed Forward Networks

As a convolutional neural networks successfully capture the spatial information in an image compared to the fully connected neural networks due to the reduction in the number of parameters and reusability of weights. Therefore, the feed forward network is replaced by the convolutional neural networks. The final layer of each feed forward network have d_{model} number of filters to keep the 3rd dimension of the image consistent after each encoder or decoder layer.

4.2.4 Final Layer

The last decoder layer is now just followed by a 1 x 1 CNN with sigmoid activation. This layer generates the final predicted image.

4.2.5 Positional Encoding

Similar to positional encodings used by the authors of the original paper [3] , we also use sine and cosine functions of different frequencies:

$$\begin{aligned} PE(pos, row, col, 2i) &= \sin(\frac{pos}{10000^{2i/d_{model}}}) \\ PE(pos, row, col, 2i+1) &= \cos(\frac{pos}{10000^{2i/d_{model}}}) \end{aligned} \tag{4.1}$$

For a given position pos in the image sequence, this positional encodings vector has the dimensions $rows \times cols \times d_{model}$ ($\in \mathbb{R}^{rows \times cols \times d_{model}}$). Since the sine and cosine functions are independent of row and col, each pixel in a channel of the image will have the same value.

The rest model i.e., the number of components in Encoder / Decoder, multi-head attention, etc. everything remains the same.

4.3 Experiment on an Artificial Dataset

The Keras next-frame prediction tutorial [7] uses a very simple artifically generated dataset for testing the video prediction model. One sample of the dataset is a sequence of image frames. Each frame is 40 $\times$ 40 in size and has 3 - 7 squares moving inside the frame with a constant velocity. The squares are of shapes 1 $\times$ 1 or 2 $\times$ 2 and have intensity value as 1 while background has 0 intensity value. Since it's a very simple dataset, it helps to understand whether the model can work on a simplest dataset.

A training dataset of 1000 image sequences was created with each sequence having 20 frames (10 input frames + 10 output frames). A 3 layer Encoder-Decoder Transformer model was trained for 50 iterations to minimze average binary crossentropy loss using back- progation through time and RMSProp optimizer(learning rate = 0.001 and decay rate = 0.9) was used. In the model, d_{model} = 64 and number of heads was set to 16. In the feed forward network inside each encoder or decoder layer, 2 layers of convolutional neural network was used. 1st CNN layer had 128 filters and the number of filters in the 2nd layer was d_{model}(= 64). The filters size was 3 $\times$ 3 in every CNN layer. The final layer was also a CNN layer with 1 filter and sigmoid activation. The total training loss was 0.0002.

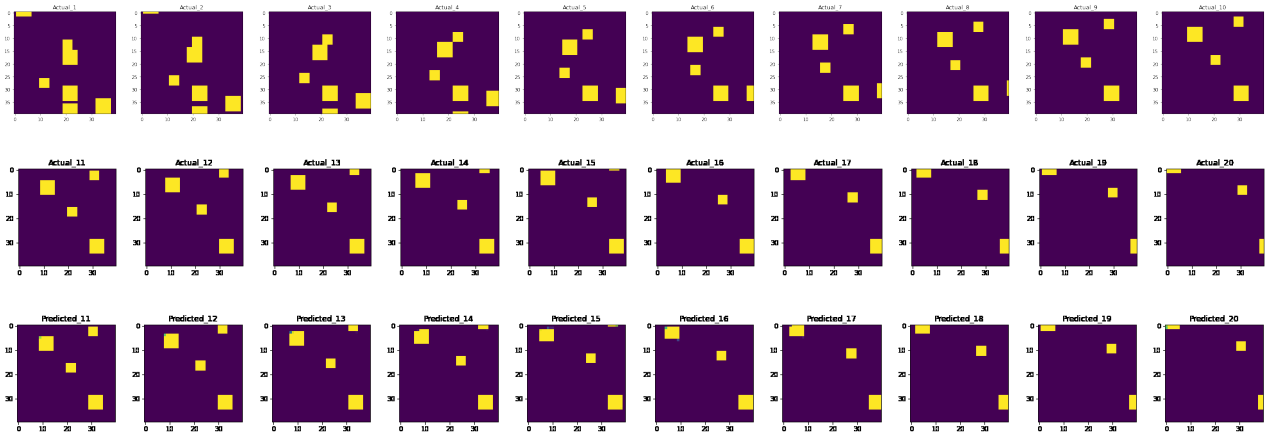

Fig. 4.1 An example from the test set. From left to right: input frames; ground truth; prediction by the model

The above figure is a sample from test dataset. Image frames in the 1st row is the input (total 10 frames), 2nd row is the ground truth or sequence of next image frames and 3rd row is predicted image frames.

Chapter 5

Moving MNIST Dataset

Moving MNIST Dataset consists of sequences of 20 image frames showing 2 handwritten digits from the MNIST database moving in the 64 x 64 frame. Each sequence is generated by randomly choosing 2 digits from the MNIST database and each of them is assigned a random velocity in the beginning. The digits move inside the frame and bounce back when they reaches the edge of the frame. Moving MNIST dataset of 10,000 sequences can also be found at `http://www.cs.toronto.edu/~nitish/unsupervised_video/`

The Moving MNIST dataset frame has 256 intensity levels (0 - 255). The frames were thresholded to have binary intensities (0 and 1). Each 20 frames sequence was splitted into two sequence of 10 frames (first 10 frames for input and other 10 for output). The models were trained to minimize the binary crossentropy loss and RMSProp optimizer (learning rate = 0.001 and rho = 0.9) was used during the training. Also, early stopping was performed on the validation set.

5.1 Image Reshaping

All the models, listed below, were trained on NVidia K80 GPU provided by Kaggle. These models contain many convolutional neural networks (in Transformer model) or components using a lot of convolutional operations (in Convolutional LSTMs). The memory allocated for the weights or paramenters of these models are mostly dominated by the convolutional operations parameters. These weights (or parameters in convolutional operations) depends on `batch_size x image_rows x image_cols x filters`. Here, the number of filters vary from one convolutional component to other. Training deeper models will require a good memory space. Here, deeper models means models having many layers.

Directly training the models with image frames of size 64×64 restricted us to built not sufficiently deep network which resulted in very less accurate predictions. Since the memory allocated depends mostly on `batch_size x image_rows x image_cols x filters`, we can reduce the memory usuage by shrinking the base dimension of the images i.e., `image_size (rows x cols)`. Therefore, all 64 x 64 frames were reshaped into 16 x 16 x 16 frames (increasing the channel dimension) using 4 x 4 patch. This allowed us to train deeper networks. The batch sizes for the following models were 16.

5.2 Convolutional LSTM Encoder-Decoder Networks [1]

Two Encoder-Decoder networks were trained on the MNIST image sequences (7,000 training sequences, 1,500 validation sequences and 1,500 test sequences).

1. 2 Layers with 128, 128 hidden units and (5 x 5) filter size in each layer. The average binary crossentropy loss was 0.1868.

2. 3 Layers with 128, 64, 64 hidden units and (5 x 5) filter size in each layer. The average binary crossentropy loss was 0.1675.

5.3 Transformer Encoder-Decoder Networks

For training the transformer model, a dataset of 1000 MNIST image sequences was used. A 5 layer Encoder-Decoder Transformer model was trained for 50 iterations to minimze average binary crossentropy loss using back- progation through time and RMSProp optimizer(learning

rate = 0.001 and decay rate = 0.9) was used. In the network, $d_{model} = 64$ and number of heads was set to 16. The feed forward networks inside each encoder or decoder layer, constituted of 3 layers of convolutional neural networks. 1st and 2nd CNN layer had 128 filters while the number of filters in the 3rd layer was $d_{model}(= 64)$. The filters size was 3×3 in every CNN layer. The final layer was also a CNN layer with 1 filter and sigmoid activation. The model was tested on a test dataset of 300 image sequences and the average binary crossentropy loss was 0.5380.

Fig. 5.1 An example from the test set. From left to right: (i) Row 1 : input frames; (ii) Row 2 : ground truth; (iii) Row 3 : prediction by 128-128 layers ConvLSTM model (iv) Row 4 : prediction by 128-64-64 layers ConvLSTM model (v) Row 5 : prediction by the 5 layers Transformer model

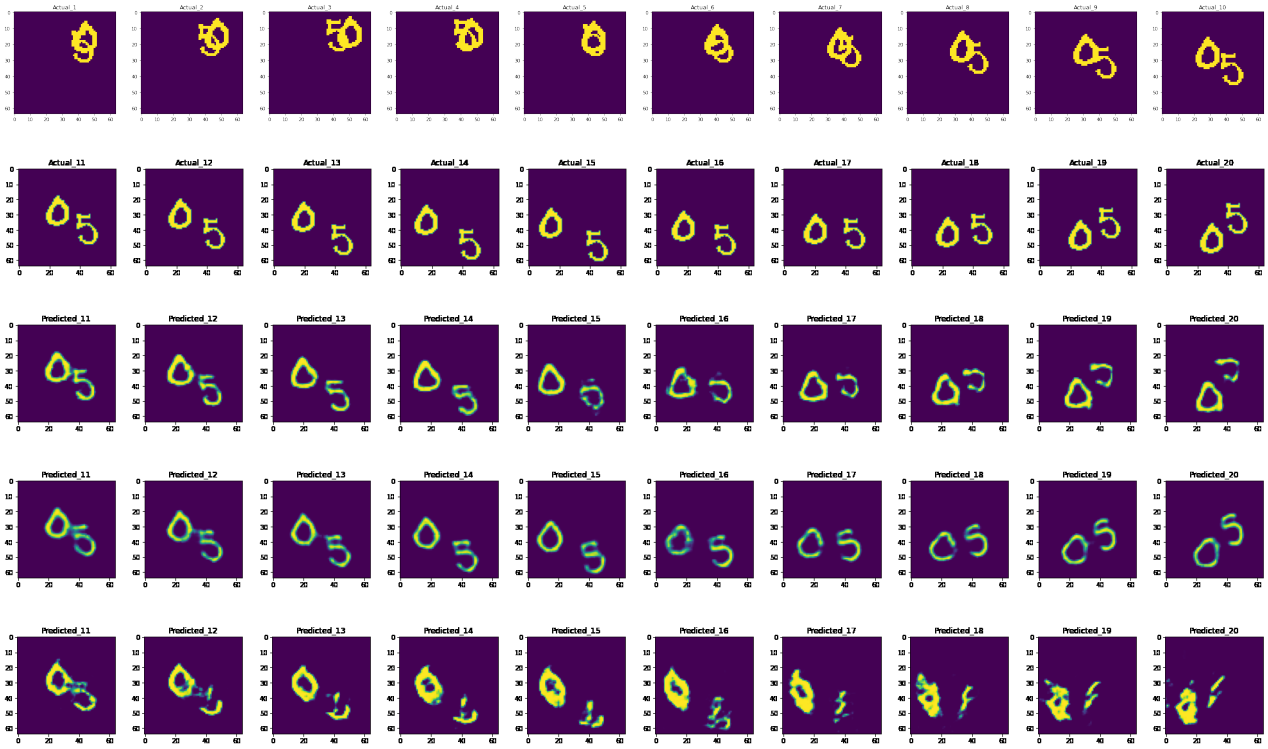

Fig. 5.2 An example from the test set. From left to right: (i) Row 1 : input frames; (ii) Row 2 : ground truth; (iii) Row 3 : prediction by 128-128 layers ConvLSTM model (iv) Row 4 : prediction by 128-64-64 layers ConvLSTM model (v) Row 5 : prediction by the 5 layers Transformer model

In the above figures, 128-128 layers ConvLSTM model refers to the Convolutional Encoder-Decoder Network having 2 layers with 128 hidden units in each layer. 128-64-64 layers ConvLSTM models can also be interpreted in similar manner. 5 layer Transformer model refers to the model described in the above section.

The Convolutinal LSTM based models performed better compared to the Transformer based model. But the performance of the transformer based model can be improved as during the training the ConvLSTM based consumed around 90 % of the 16 Gigabytes GPU memory while the above transformer model just consumed around 15 % of the 16 Gigabytes GPU memory. The transformer model can be made more complex by increasing the layers, higher d_{model} value or by increasing the feed forward networks compexity. I have tried to train the transformer model with more layers, higher d_{model}, etc. but I could only achieve best performance with the parameter used above till now.

The reason for using 1000 training sequences in the transformer model compared to the 7000 sequences is the training time. Since the transformer model was 5 layer model, it took around twice the more time to train compared to the 3 layer ConvLSTM model.

Chapter 6

UCF101 Dataset [8]

UCF101 is an action recognition dataset of realistic action videos, collected from YouTube, having 101 action categories. It has a total of 13320 videos from 101 action categories. Each action has 25 different videos with different scene or person. Each of these 25 videos, are further splitted into 4-7 videos in avi format.

Before training, the videos had to be processed. All those videos (the above 25 videos) under an action which are splitted into 4-7 videos, were merged back together. Then, under every action, each of the 25 videos were converted to a sequence of image frame using open-cv library. These images had 240 x 320 x 3 as its dimensions. These images were first converted to grayscale images and then resized to 90 x 120 using interpolation based on pixel area relation (`INTER_AREA` in open-cv). Also, these images were normalized to have intensities in $[0, 1]$. From these images, 10 length sequences (5 inputs + 5 outputs) were created, reshaped using 3 x 4 patch into 30 x 30 x 12 images and these sequences were used in the training and test datasets. The models were trained for each action set separately.

(a) (b) (c)

Fig. 6.1 Images from UCF101 dataset: (a) Biking; (b) Bowling; (c) Basketball

6.1 Basketball Action Data

964 10-length sequences were created from the basketball action videos. These were further splitted into 600 training sets, 150 validation sets and 214 test sets. The batch size was fixed to 8. RMSProp optimizer with learning rate = 0.001 and decay rate = 0.9 was used. The models were trained to minimize the average mean squared error.

6.1.1 Convolutional LSTM Encoder-Decoder Networks

1. A 2 Layers with 256, 256 hidden units and (3 x 3) filter size in each layer was trained for 125 iterations. The average mean squared loss was equal to 0.0124.

2. A 3 Layers with 160, 160, 160 hidden units and (3 x 3) filter size in each layer was trained for 150 iterations and the average mean squared loss was 0.0123.

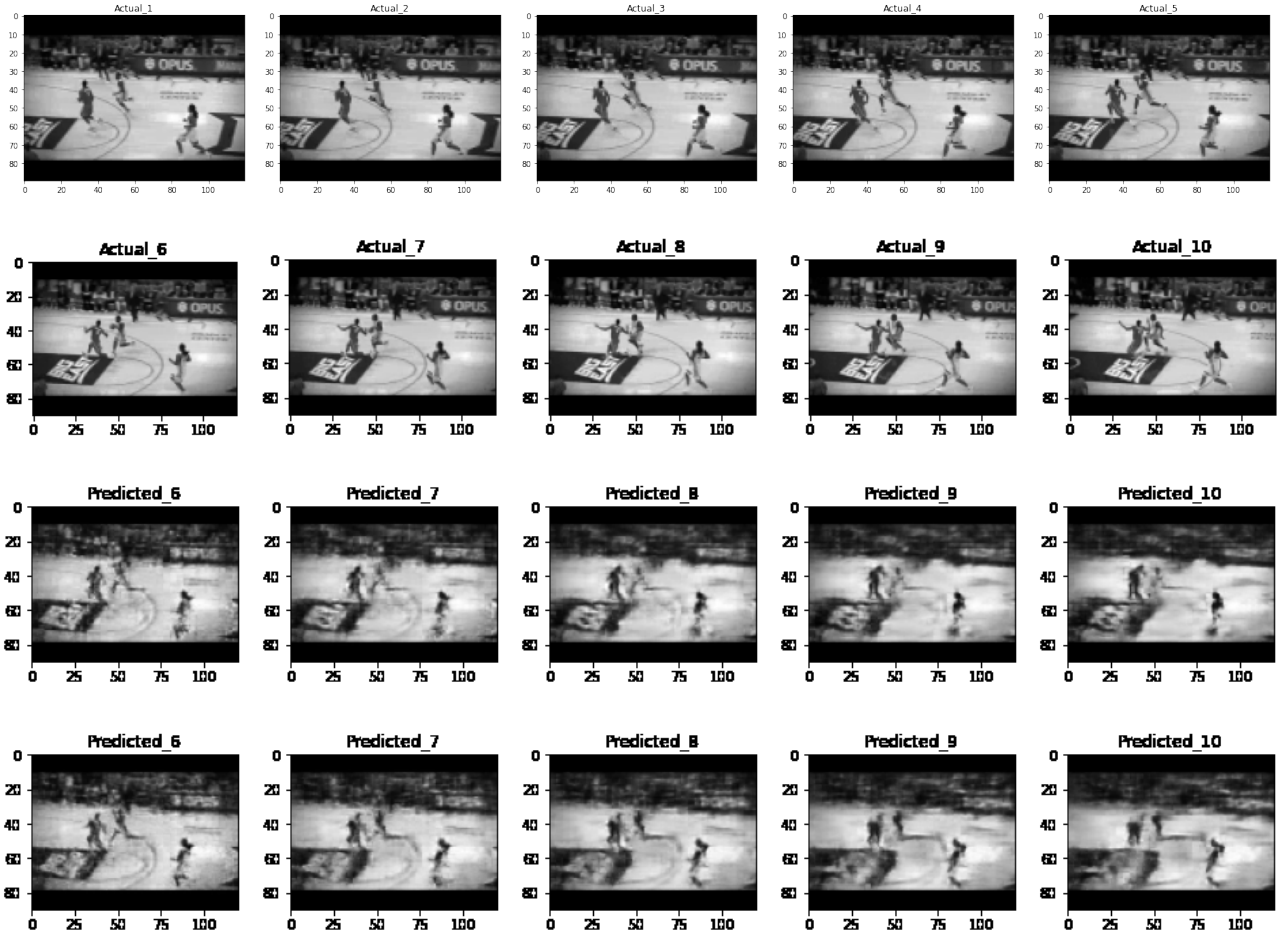

Fig. 6.2 An example from the test set. From left to right: (i) Row 1 : input frames; (ii) Row 2 : ground truth; (iii) Row 3 : prediction by the above 2 layers ConvLSTM model (iii) Row 4 : prediction by the above 3 layers ConvLSTM model; Click here to see the gif version

In the above figure, the models couldn't predict the leg movements and other things but it could predict the camera and players motion.

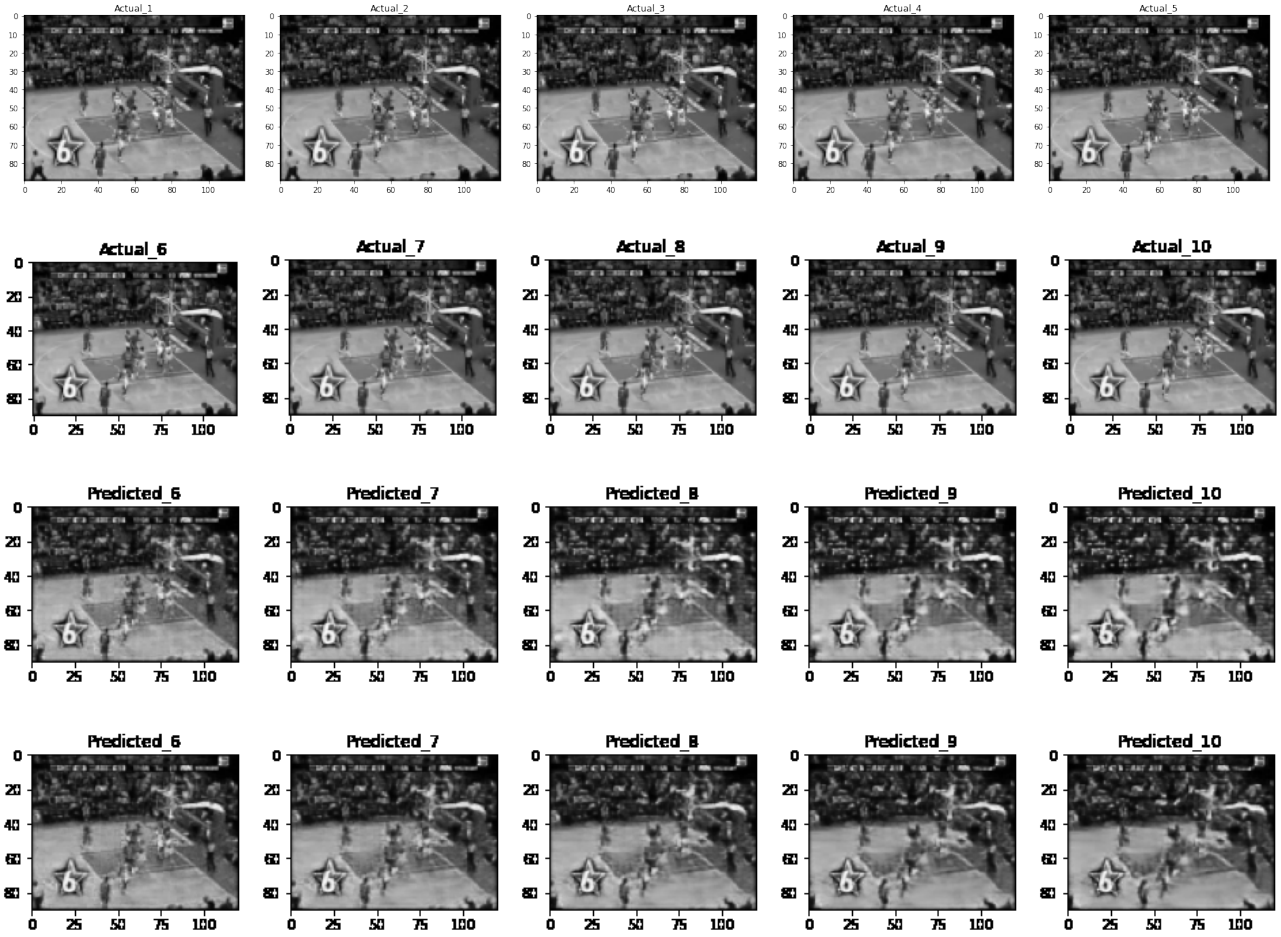

Fig. 6.3 An example from the test set. From left to right: (i) Row 1 : input frames; (ii) Row 2 : ground truth; (iii) Row 3 : prediction by the above 2 layers ConvLSTM model (iii) Row 4 : prediction by the above 3 layers ConvLSTM model; Click here to see the gif version

In the above figure, the models could predict some of the motions of few persons / players like a person moving out the image in the bottommost left corner.

6.1.2 Transformer Encoder-Decoder Networks

For training the transformer model, the 600 10-length was used. A 5 layer Encoder-Decoder Transformer model was trained for 150 iterations to minimze average mean squared error loss and RMSProp optimizer(learning rate = 0.001 and decay rate = 0.9) was used. In the network, $d_{model} = 64$ and number of heads was set to 16. The feed forward networks inside each encoder or decoder layer, constituted of 2 layers of convolutional neural networks. 1st CNN layer had 128 filters while the number of filters in the last layer was $d_{model}(= 64)$. The filters size was 3×3 in every CNN layer. The final layer was also a CNN layer with 1 filter and sigmoid activation. The model was tested on a test dataset of 364 image sequences and the average mean squared error loss was 0.0168. The model doesn't seem to understand the motions from the inputs and predict the future motions. It is still being tuned to get better result (no images for this model's prediction is shown in this chapter). We get a low error value as the intensities values are in $[0, 1]$ and many pixels remain stationary in the images.

6.2 Bowling Action Data

There were 1930 10-length sequences of bowling action data. These were splitted into 1200 training set, 300 validation set and 430 test set. The model was trained to minimize the average mean squared error loss and RMSProp optimizer (learning rate = 0.001 and decay rate = 0.9) was used. The batch size was equal to 8.

1. A 3 layer ConvLSTM Encoder-Decoder Network with 256, 128 and 128 hidden units in the 1st, 2nd and 3rd layer respectively was trained for 150 iterations. The filter sizes were 3 x 3 in each layer. The average mean squared error loss was 0.0081.

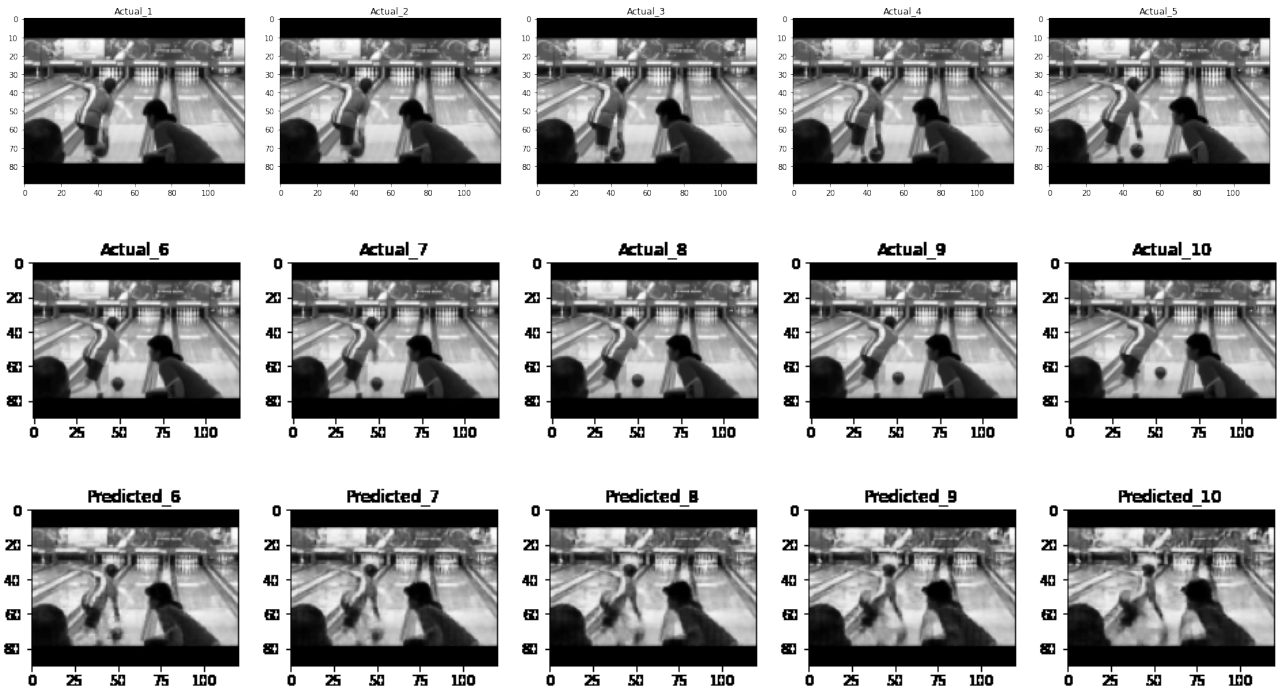

Fig. 6.4 An example from the test set. From left to right: (i) Row 1 : input frames; (ii) Row 2 : ground truth; (iii) Row 3 : prediction by the 256-128-128 layers ConvLSTM model; Click here to see the gif version

In the above figure, the model could somewhat predicts the motion of few things such as ball, hand, head movements, etc for the initial few frames.

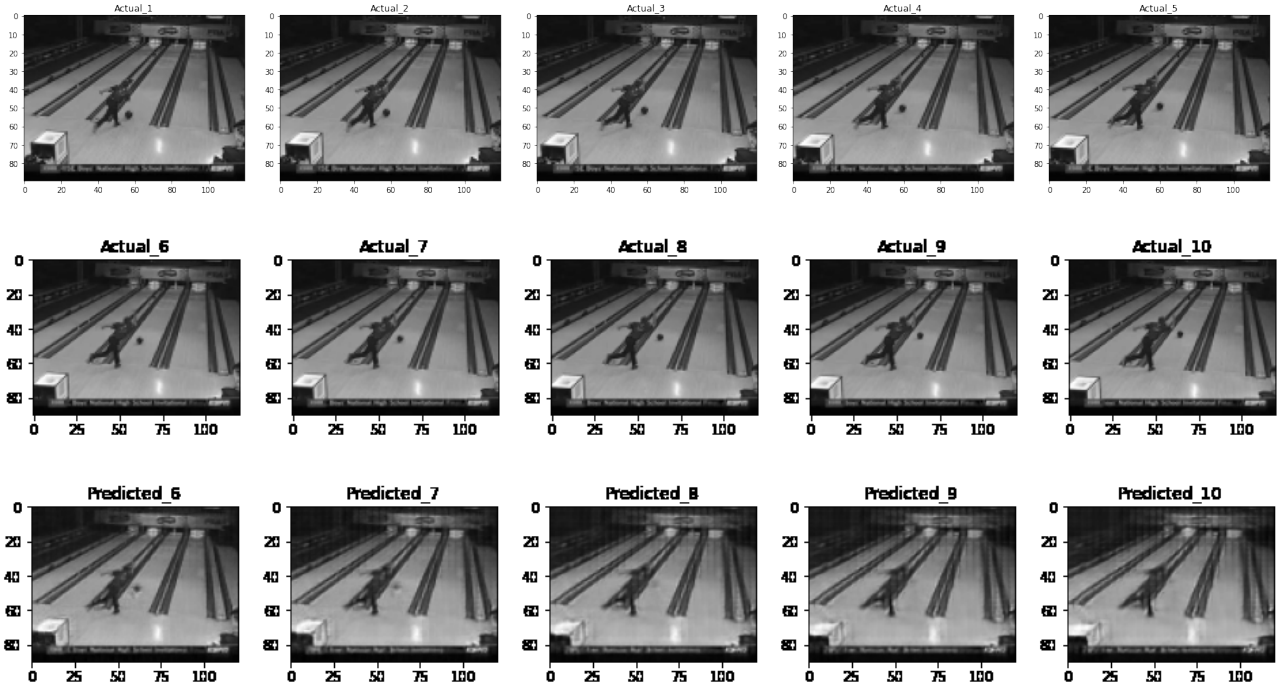

Fig. 6.5 An example from the test set. From left to right: (i) Row 1 : input frames; (ii) Row 2 : ground truth; (iii) Row 3 : prediction by the 256-128-128 layers ConvLSTM model; Click here to see the gif version

In the above figure, the model couldn't fully predict his hand movement after throwing but it predicts the could motion of the ball for few frames before the ball disappears in the predicted images.

The prediction made by the Convolutional Encoder-Decoder Networks is not that good but it's fine. It is difficult for the model to understand the exact smaller movements such as hand motion, legs movement, etc but it atleast understands the motion and predicts it. The Convolution model was almost full to its capacity for a 16 GB GPU.

The transformer based model is not performing well for the complex data like the real world videos. Some parameters on which I tried to train model consumed about 1 / 4th of the GPU memory compared to the ConvLSTM based model. So, there is a definitely scope for improvement. One thing to note is that higher layers consumes more running time. So, training with less layers but more filters in CNN layers and d_{model} can perform well while consuming lesser time during the run. d_{model} and the number of heads are important parameters as they find relevant information from the previous sequences and it is important to choose them properly.

Chapter 7

Repository, Conclusion and Future Work

7.1 Conclusion and Future Work

Though the transformer model is not performing satisfactorily for the complex real world videos, it has a scope for the improvement because of its low memory consumption compared to the ConvLSTM based models. Currently, I'm training and testing the model on different values of various parameters (d_{model}, filters, etc). If the model performs well on a particular action of UCF101 dataset then it's robustness can be tested by training and testing on the combined actions dataset. Further, it can also be tested on other datasets like kitti (videos captured by driving cars), etc. and could be used for other predictions.

7.2 Repository

The Github repository of this project can be found at :

`https://github.com/iamrakesh28/Video-Prediction/`. It contains the implementation of the transformer model for the video predictions.

The code for the Convolutional Encoder-Decoder Networks can be found at : `https://github.com/iamrakesh28/Deep-Learning-for-Weather-and-Climate-Science/`. It is also the repository for the BTP-I project.

References

[1] https://github.com/iamrakesh28/Deep-Learning-for-Weather-and-Climate-Science/

[2] Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting

[3] Attention Is All You Need

[4] https://en.wikipedia.org/wiki/Attention

[5] https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)

[6] Unsupervised Learning of Video Representations using LSTMs

[7] https://keras.io/examples/vision/conv_lstm/

[8] https://www.crcv.ucf.edu/data/UCF101.php