# RSGAN: Recurrent Stacked Generative Adversarial Network for Conditional Video Generation

**Shujon Naha, Khandokar Md. Nayem, Md. Lisul Islam**
School of Informatics and Computing
Indiana University
Bloomington, IN
{snaha, knayem, islammdl}@iu.edu

## Abstract

Generating video frames based on a pre-condition is a challenging problem and requires understanding of per frame contents and visual dynamics and their relevance to the pre-condition. In this paper, we propose a novel Recurrent Stacked Generative Adversarial Network (RSGAN) based model to generate video frames based on a given pre-condition. The pre-condition can be anything related to the generated video, like- action classes, sentence descriptor, fMRI signal, etc. In our knowledge, this is the first work to address the problem of conditional video generation using adversarial network.

**Index Terms:** rsgan, recurrent netwok, stacked generative adversarial network, adversarial network, conditional video generation, video generation.

## 1 Introduction

Generative adversarial networks have shown incredible results to produce images from pre-conditions such as text, attributes etc [36, 34]. In these works, random noises incorporated with a semantic vector representation of the prec-condition is given as input to the generator network to produce images relevant to the pre-condition. The discriminator network then learns to distinguish between the images generated by the generator and the real image from the database. A min-max learning algorithm is used to train both the model where the generator tries to continously fool the discriminator by producing better images similar to the original one and the discriminator learns to make the job harder for the generative network by getting better at distinguishing between real and fake images. Most of the times, the generative network is a convolutional neural network which produces image from a single vector by using several deconvolution steps. The discriminator network is also a convolutional neural network which takes the image from the generator network output and the corresponding original image from the database and tells the similarity between the generated image and the real image.

Generative adversarial networks have been also used for predicting future frames from a video sequence and generate videos with scene dynamics [16, 31]. In this works, multiple frames are combined together and 3D convolution is used in the domains of space and time to predict the next frames. These works have shown the capability of adversarial networks to capture the scene dynamics although for small temporal intervals.

In this paper, we address the problem of generating videos based on pre-conditions such as action classes, fMRI signals and sentence descriptions using adversarial network. Generating videos based on pre-conditions pose a unique set of challenges than the conditional image generation and unconditional video generation problem. In our case, each of the frames will be generated based on the previous frames and the given pre-condition such as in Figure 1. The numbers of previous frames

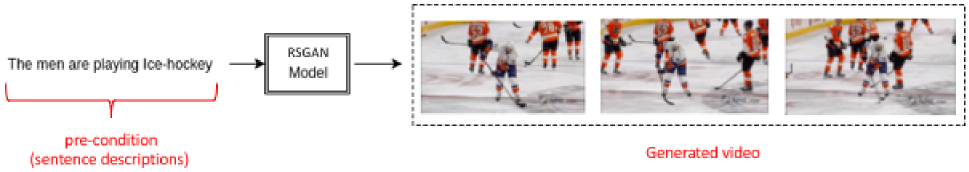

The men are playing Ice-hockey
RSGAN Model
pre-condition (sentence descriptions)
Generated video

Figure 1: Generating video sequence based on a given pre-condition (sentence description).

can vary from zero to a maximum number. Thus the usual approach of using 3D convolution will not be applicable in our case. Moreover, we need to make the pre-condition available to the system at each time frame so the whole generated video is consistent with the pre-condition.

## 2 Background

In this section will review FCN, RNN and GAN which will be repeatedly referred throughout the paper.

### 2.1 Fully Convolutional Networks (FCN)

For classification task using convolutional neural networks, the few last fully connected layers are responsible for the classification part. But with pixel-wise labelling, there is a need for dense predictions on all the pixels. In [14] the idea of using a fully convolutional neural network that is trained for pixel-wise semantic segmentation is presented. It is shown that it surpasses the state of the art in semantic segmentation on PASCAL VOC, NUYDv2, and SIFT Flow datasets. The FCN method is briefly discussed in what follows.

FCN architecture is based on VGG [25] architecture due to its success in classification tasks. However, due to the fully connected layers that these networks have, they can only accept fixed size input and produce a classification label. To overcome this problem, it is possible to convert a fully connected layer into a convolutional layer. Accordingly, this network can yield coarse maps pixel wise prediction instead of one classification output.

In order to have dense prediction from this coarse map, it needs to be up-sampled to the original size of the input image. The up-sampling method can be a simple bi-linear interpolation. But in [14] a new layer that applies upsampling within the network was presented. It makes it efficient to learn the up-sampling weights within the network using back-propagation. The filters of the deconvolution layer act as the basis to reconstruct the input image. Another idea for up-sampling is to stitch together output maps from shifted version of the input. But it was mentioned in [14] that using up-sampling with deconvolution is more effective. In [18] the idea of having a full deconvolution network with both deconvolution layers and unpooling layers is presented.

The FCN architecture has been tried in different applications. In [9] it is used for object localization. In [32] modified architecture was used for visual object tracking. Finally for semantic segmentation in [19] a full deconvolution network is presented with stacked deconvolution layers.

### 2.2 Recurrent Neural Networks (RNN)

Recurrent Neural Networks [30] are designed to incorporate sequential information into a neural network framework. These networks are capable of learning complex dynamics by utilizing a hidden unit in each recurrent cell. This unit works like a dynamic memory that can be changed based on the state that the unit is in. Accordingly, the process of each unit yields to two outcomes. Firstly, an output is computed from the current input and the hidden units values (the networks memory). Secondly, the network updates its memory based on, again, current input and hidden units value. The simplest recurrent unit can be modeled as,

$$h_t = \theta\phi(h_{t-1}) + \theta_x x_t$$

$$y_t = \theta_y \phi(h_t)$$

Here, $h$ is the hidden layer, $x$ is the input layer and $y$ is the output layer and $\phi$ is the activation function.

Recurrent networks were successful in many tasks in speech recognition and text understanding [28] but they come with their challenges. Unrestricted data flow between units causes problems with vanishing and exploding gradients [1]. During the back propagation through recurrent units, the derivative of each node is dependent of all the nodes which processed earlier. This is shown in following equations where $E$ is the loss of the layer. To compute $\frac{\partial h_t}{\partial h_k}$ a series of multiplication from $k = 1$ to $k = t - 1$ is required. Assume that $\phi$ is bounded by $\alpha$ then $||\frac{\partial h_t}{\partial h_k}|| < \alpha^{t-k}$

$$\frac{\partial E}{\partial \theta} = \sum_{t=1}^{t=S} \frac{\partial E_t}{\partial \theta}$$

$$\frac{\partial E_t}{\partial \theta} = \sum_{k=1}^{k=t} \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial \theta}$$

$$\frac{\partial h_t}{\partial h_k} = \prod_{i=k+1}^{t} \frac{\partial h_i}{\partial h_{i-1}} = \prod_{i=k+1}^{t} \theta^T diag[\phi(h_{i-1})]$$

A solution to this problem is to use gated structures. The gates can control back propagation flow between each node. Long-Short Term Memory [8] is the first such proposed architecture and it is still popular. A more recent architecture is Gated Recurrent Unit [2] which has simpler cells yet with competent performance [3].

### 2.2.1 Long Short Term Memory (LSTM)

As mentioned, LSTM uses a gated structure where each gate controls the flow of a particular signal. Each LSTM node has three gates that are input, output and forget gate each with weights that can be learnt. These gates can learn the optimal way to remember useful information from previous states and decide the current state.
In the following equations, the procedure of computing different gates and hidden states is shown, where $i_t$, $f_t$ and $o_t$ are input, forget and output gates respectively. While $c_t$ denote the cell internal state, and $h_t$ is the hidden state.

$$i_t = \sigma(W_{x_i} x_t + W_{h_i} h_{t-1} + b_i)$$
$$f_t = \sigma(W_{x_f} x_t + W_{h_f} h_{t-1} + b_f)$$
$$o_t = \sigma(W_{x_o} x_t + W_{h_o} h_{t-1} + b_o)$$
$$g_t = \sigma(W_{x_c} x_t + W_{h_c} h_{t-1} + b_c)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot \phi(c_t)$$

### 2.2.2 Gated Recurrent Unit (GRU)

The Gated Recurrent Unit, similar to LSTM, utilizes a gated structure for flow-control. However, it has a simpler architecture which makes it both faster and less memory consuming. The model is shown in Figure 2, where $r_t$, $z_t$ is the reset and update gate respectively, while $h_t$ is the hidden state.

$$z_t = \sigma(W_{hz} h_{t-1} + W_{xz} x_t + +b_z)$$
$$r_t = \sigma(W_{hr} h_{t-1} + W_{xr} x_t + +b_r)$$
$$\hat{h_t} = \Phi(W_h(r_t \odot h_{t-1}) + W_x x_t + +b)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z \odot \hat{h_t}$$

GRU does not have direct control over memory content exposure while LSTM has it by having an output gate. These two are also different in the way that they update the memory nodes. LSTM updates its hidden state by summation over flow after input gate and forget gate. GRU however, assumes a correlation between how much to keep from the current state and how much to get from the previous state and it models this with the $z_t$ gate.

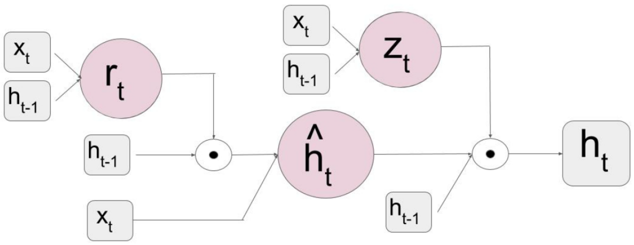


Figure 2: GRU Architecture.

### 2.3 Generative Adversarial Networks (GAN)

Generative Adversarial Networks (GAN) [6] are composed of two models that are alternatively trained to compete with each other. The generator $G$ is optimized to reproduce the true data distribution $p_{data}$ by generating images that are difficult for the discriminator $D$ to differentiate from real images. Meanwhile, $D$ is optimized to distinguish real images and synthetic images generated by $G$. Overall, the training procedure is similar to a two-player min-max game with the following objective function,

$$\underset{G}{min}\,\underset{D}{min}\, V(D,G) = \mathbb{E}_{x\ p_{data}}[logD(x)] + \mathbb{E}_{z\ p_z}[log(1 - G(z))]$$

where $x$ is a real image from the true data distribution $p_{data}$, and $z$ is a noise vector sampled from distribution $p_z$ (e.g., uniform or Gaussian distribution).

Conditional GAN [5, 17] is an extension of GAN where both the generator and discriminator receive additional conditioning variables $c$, yielding $G(z, c)$ and $D(x, c)$. This formulation allows $G$ to generate images conditioned on variables $c$.

## 3 Our Approach

In our problem, we are given a pre-condition either in the form of an action class name, fMRI signal or textual description and we need to generate a sequence of video frames which will be coherent with the given pre-condition. First, we will discuss how we can generate a high resolution frame independently based on the pre-condition using a stacked adversarial network. Then we will discuss how we can make a recurrent fully convolutional network so we can propagate the context to generate the next frame. Finally, we will describe our recurrent stacked adversarial network to generate videos based on pre-conditions.

### 3.1 Stacked Adversarial Network

To generate a video first, we need to learn to generate a single frame from the pre-condition. We have adapted the StackGAN adversarial network model [36] for this problem, as it can generate photo-realistic images from sentence descriptions. At first stage, this model generates an image based on the encoded text and a random vector. Then the generated image in the first stage is used as the input with the encoded text vector to the second stage adversarial network to generate a photo-realistic image. The architecture of the model can be seen in Figure 3.

#### 3.1.1 Stage-I GAN

As shown in Figure 3, the conditioning text description $t$ is first encoded by an encoder, yielding a text embedding $\varphi_t$. In previous works [12, 20], the text embedding is nonlinearly transformed to generate conditioning latent variables for the generator. However, latent space conditioned on text is usually high dimensional (> 100 dimensions). With limited amount of data, it usually causes discontinuity in the latent data manifold, which is not desirable for learning the generator.
To mitigate this problem, we introduce a conditioning augmentation technique to produce more conditioning variables for the generator. We randomly sample latent variables from an independent Gaussian distribution $N(\mu(\varphi_t), \Sigma(\varphi_t))$, where the mean $\mu(\varphi_t)$ and diagonal covariance matrix $\Sigma(\varphi_t)$ are functions of the text embedding $\varphi_t$. The proposed formulation encourages robustness to small

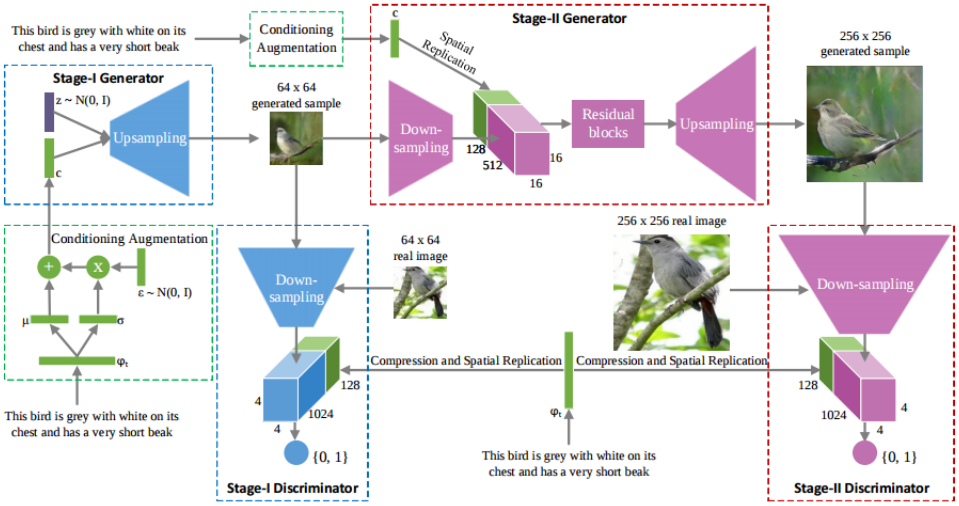

This bird is grey with white on its chest and has a very short beak
Conditioning Augmentation
Stage-II Generator
c
Spatial Replication
256 x 256 generated sample
Stage-I Generator
z ~ N(0, I)
c
Upsampling
64 x 64 generated sample
Down-sampling
128
512
16
16
Residual blocks
Upsampling
256 x 256 real image
Conditioning Augmentation
ε ~ N(0, I)
μ
σ
φt
This bird is grey with white on its chest and has a very short beak
Down-sampling
64 x 64 real image
Compression and Spatial Replication
Compression and Spatial Replication
Down-sampling
128
4
1024
4
{0, 1}
Stage-I Discriminator
φt
This bird is grey with white on its chest and has a very short beak
128
1024
4
4
{0, 1}
Stage-II Discriminator

Figure 3: The architecture of the StackGAN from [36]. The Stage-I generator draws a low resolution image by sketching rough shape and basic colors of the object from the given text and painting the background from a random noise vector. The Stage-II generator generates a high resolution image with photo-realistic details by conditioning on both the Stage-I result and the text again.

perturbations along the conditioning manifold, and thus yields more training pairs given a small number of image-text pairs. To further enforce the smoothness over the conditioning manifold and avoid overfitting [4, 13], we add the following regularization term to the objective of the generator during training,

$$D_{KL}(N(\mu(\varphi_t), \Sigma(\varphi_t))||N(0, I))$$

which is the Kullback-Leibler divergence (KL divergence) between the standard Gaussian distribution and the conditioning Gaussian distribution.

Conditioned on Gaussian latent variables $c_0$, Stage-I RSGAN trains discriminator $D_0$ and generator $G_0$ by alternatively maximizing $\mathcal{L}_{D_o}$ and minimizing $\mathcal{L}_{G_o}$.

$$\mathcal{L}_{D_o} = \mathbb{E}_{(I_o,t)\sim p_{data}}[logD_0(I_0, \varphi_t)] + \mathbb{E}_{z\sim p_z, t\sim p_{data}}[log(1 - D_0(G_0(z, c_0), \varphi_t))]$$
$$\mathcal{L}_{G_o} = \mathbb{E}_{z\sim p_z, t\sim p_{data}}[log(1 - D_0(G_0(z, c_0), \varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu_0(\varphi_t), \Sigma_0(\varphi_t))||\mathcal{N}(0, I))$$

where the real image $I_0$ and the text description $t$ are from the true data distribution $p_{data}$ $z$ is a noise vector randomly sampled from a given distribution $p_z$ (e.g., Gaussian distribution used in this paper). $\lambda$ is a regularization parameter that controls the balance between the two terms of $\mathcal{L}_{G_o}$. We use $\lambda = 1$ for all our experiments. $\varphi_t$ is the text embedding, which is generated by a pre-trained encoder [21] in this paper. Gaussian conditioning variables $c_0$ are sampled from $N(\mu_0(\varphi_t), \Sigma_0(\varphi_t))$ to reflect the text description. Using the reparameterization trick introduced in [22], both $\mu_0(\varphi_t)$ and $\Sigma_0(\varphi_t)$ are learned jointly with the rest of the network.

**Model Architecture** For the generator, the text embedding $\varphi_t$ is fed into a fully connected layer to generate $\mu_0$ and $\sigma_0$ ($\sigma_0$are the values in the diagonal of $\Sigma_0$) for Gaussian distribution $N(\mu(\varphi_t), \Sigma(\varphi_t))$. Our $N_g$ dimensional conditioning vector $c_0$ is computed by $c_0 = \mu_0 + \sigma_0 \odot \epsilon$ (where $\odot$ is the element-wise multiplication, $\epsilon$ $N(0, I)$). Then, $c_0$ is concatenated with a $N_z$ dimensional noise vector to generate a $W_0 \times H_0$ image by a series of up-sampling blocks.
For the discriminator, the text embedding $\varphi_t$ is first compressed to $N_d$ dimensions using a fully-connected layer and then spatially replicated to form a $M_d \times M_d \times N_d$ tensor. Meanwhile, the image is fed through a series of downsampling blocks until it has $M_d \times M_d$ spatial dimension.Then, the image filter map is concatenated along the channel dimension with the text tensor. The resulting tensor is further fed to a $1 \times 1$ convolutional layer to jointly learn features across the image and the text. Finally, a fully connected layer with one node is used to produce the decision score.

### 3.1.2 Stage-II GAN

Low resolution images generated by Stage-I GAN lack vivid object parts and might also contain shape distortions. In addition, some details in the text might be omitted in the first stage. This is important information needed to generate a photo-realistic image. Stage-II GAN is built upon Stage-I GAN to generate photo-realistic high resolution images. It conditions on low resolution images generated by the previous stage, and also the text embedding again to correct defects in Stage-I results and encourage the model to extract previously ignored information in the text to generate more photo-realistic details.

Conditioned on the low resulation sample $s_0$ and Gaussian latent variables $c$, discriminator $D$ and generator $G$ in Stage-II RSGAN is trained by alternatively maximizing $\mathcal{L}_D$ and minimizing $\mathcal{L}_G$.

$$\mathcal{L}_D = \mathbb{E}_{(I,t)\sim p_{data}}[logD(I,\varphi_t)] + \mathbb{E}_{s_\mathrm{o}\sim pG_\mathrm{o},t\sim p_{data}}[log(1 - D(G(s_0,c),\varphi_t))]$$
$$\mathcal{L}_G = \mathbb{E}_{s_\mathrm{o}\sim pG_\mathrm{o},t\sim p_{data}}[log(1 - D(G(s_0,c),\varphi_t))] + \lambda D_{KL}(\mathcal{N}(\mu(\varphi_t),\Sigma(\varphi_t))||\mathcal{N}(0,I))$$

where $s0 = G0(z, c_0)$ is generated by Stage-I GAN. Different from the original GAN formulation, the random noise $z$ is not used in this stage with the assumption that the randomness has already been preserved in $s_0$. Gaussian conditioning variables $c$ used in this stage and c0 used in Stage-I GAN share the same pre-trained text encoder, generating the same text embedding $\varphi_t$. But, they utilize different fully connected layers for generating different means and standard deviations. In this way, Stage-II GAN learns to capture useful information in the text embedding that is omitted by Stage-I GAN.

**Model Architecture** For the generator, similar to the previous stage, $\varphi_t$ is used to generate our $N_g$ dimensional Gaussian conditioning vector $c$, which is spatially replicated to form a $M_g \times M_g \times N_g$ tensor. Meanwhile, the sample $s_0$ generated by Stage-I GAN is fed into several downsampling blocks until it has a spatial size of $M_g \times M_g$. Then, the image filter map and the text tensor are concatenated along the channel dimension. The resulting tensor is fed into several residual blocks [11, 7] to jointly encode the image and text features, and finally a series of up-sampling blocks are used to generate a $W \times H$ image.
For the discriminator, its structure is similar to that of Stage-I discriminator with only extra downsampling blocks since the image size is larger in this stage. To explicitly enforce GAN to learn better alignment between the image and the conditioning text, rather than using the naive discriminator, we adopt the matching-aware discriminator proposed by Reed et al. [22] for both stages. During training, the discriminator takes real images and their corresponding text descriptions as positive sample pairs, whereas negative sample pairs consist of two groups. The first is real images with mismatched text embeddings, while the second is synthetic images with conditioning text embeddings.

## 3.2 Recurrent Fully Convolutional Network

Now to generate a video, we need to pass the contextual information from previous frames to the current frame so the frames are coherent to each other. For the discriminator network, we can do that using a regular LSTM network. The discriminator network generates a vector from the input images which can be transfered to the next instance of the discriminator. But the generator network is mostly a fully convolutional neural network, it is not straightforward to create a recursive model for the generator. We have considered the recurrent fully convolutional network [29] to solve this problem. The model uses convolutional gated recurrent units which establishes recurrent connections between the colvolutional layers. This model preserves the spatial information while passing context to the next LSTM unit and reduces the number of learned parameters as well. The model is described in Figure 4.

### 3.2.1 Convolutional Gated Recurrent Unit (Conv-GRU)

Conventional recurrent units are capable of processing temporal data however, their architecture is not suitable for working on images/feature maps for two reasons. 1) weights matrix size, 2) ignoring spatial connectivity. Assume a case where a recurrent unit is placed after a feature map with the spatial size of $h \times w$ and have a number of channels $c$. After flattening, it will turn into

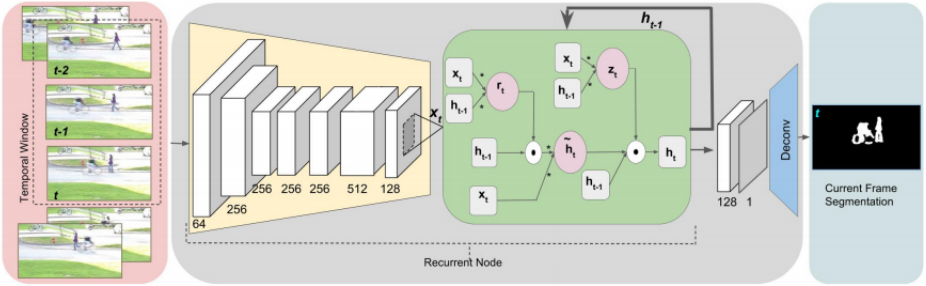


Figure 4: The architecture of RFC-VGG from [29]. Images are fed frame by frame into a recurrent FCN. A Conv-GRU layer is applied on the feature maps produced by the preceding network at each frame. The output of this layer goes to one more convolutional layer to generate heat maps. Finally, a deconvolution layer up-samples the heat map to the desired spatial size.

a $c \times h.w$ long matrix. Therefore, weights of the recurrent unit will be of size $c \times (h.w)^2$ which is power four of spatial dimension. These matrices for weights can only be maintained for small feature maps. Even if the computation was not an issue, such design introduces too much variance in the network which prevents generalization. In Convolutional recurrent units, similar to regular convolutional layer,weights are three dimensional and they convolve with the input instead of dot product. Accordingly, the cells model, in the case of a GRU architecture, will turn into equations below where the dot products are replaced with convolutions. In this design, weights matrices are of size $k_h \times k_w \times c \times f$ where $k_h$, $k_w$, $c$ and $f$ are kernels height, kernels width, number of input channels, and number of filters, respectively. In Figure 2 the operations applied on the input and the previous step will all be convolutions instead. Since we can assume spatial connectivity in feature maps, kernel size can be very small compared to feature maps spatial size. Therefore, this architecture is much more efficient and weights are easier to learn due to smaller search space.

$$z_t = \sigma(W_{hz} * h_{t-1} + W_{xz} * x_t + +b_z)$$
$$r_t = \sigma(W_{hr} * h_{t-1} + W_{xr} * x_t + +b_r)$$
$$\hat{h_t} = \Phi(W_h * (r_t \odot h_{t-1}) + W_x * x_t + +b)$$
$$h_t = (1 - z_t) \odot h_{t-1} + z \odot \hat{h_t}$$

We employ this approach for segmentation in a fully convolutional network. It is possible to apply this layer on either heat maps or feature maps. In the first case, the output of this layer will directly feed into the deconvolution layer and produces the pixel-wise probability map. In the latter case, at least one CNN layer needs to be used after the recurrent layer to convert its output feature maps to a heat map.

### 3.3 Recurrent Stacked Generative Adversarial Network (RSGAN)

Now, we have the models to generate individual images from the pre-condition and also we can connect this individual adversarial networks using recurrent connection. We combine these two models and propose the Recurrent Stacked Generative network (RSGAN) for conditional video generation. The model expands both in the temporal and spatial dimension. Each module in the first stage takes the encoded pre-condition and a random vector as input and passes a contextual matrix to the next module to generate a low resolution frame sequence. Then the modules in the second stage takes the encoded pre-condition and the low resolution output from the previous stage to generate a high resolution video sequence. The model is described in Figure 5.

## 4 Experiments

This section presents our experiments and results. First, we describe the datasets that we used then, we discuss our training methods and implemented hyper-parameters settings. Finally, quantitative and qualitative results are shown.

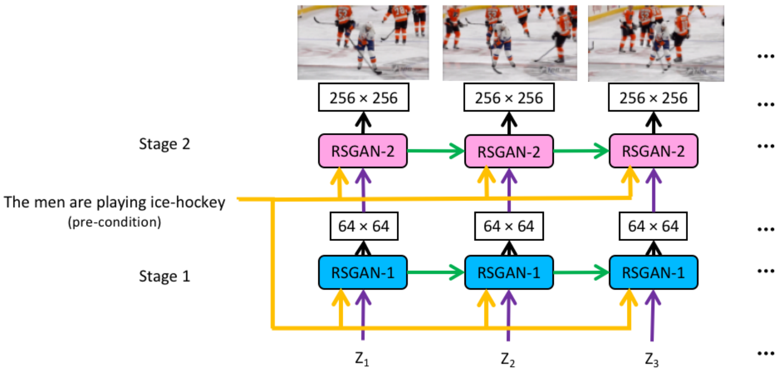


Figure 5: The proposed Recurrent Stacked Generative network (RSGAN). The recurrent adversarial network modules at the first stage (RGAN-1) take the encoded pre-condition and a random vector $z_t$ and then produces a low resolution ($64 \times 64$) size image. The modules at stage-2 (RGAN-2) takes the generated image at stage-1 and the encoded pre-condition to generate a high resolution image ($256 \times 256$).

## 4.1 Datasets

In this paper four datasets are used: 1) NTU RGB+D Action Recognition Dataset, 2) UCF-101 Dataset.

### 4.1.1 NTU RGB+D Action Recognition Dataset

NTU RGB+D action recognition dataset consists of 56,880 action samples containing RGB videos, depth map sequences, 3D skeletal data, and infrared videos for each sample [24]. This dataset is captured by 3 Microsoft Kinect v.2 cameras concurrently. The resolution of RGB videos are $1920 \times 1080$, depth maps and IR videos are all in $512 \times 424$, and 3D skeletal data contains the three dimensional locations of 25 major body joints, at each frame.

### 4.1.2 UCF-101 Dataset

UCF101 [27] dataset is an action recognition data set of realistic action videos, collected from YouTube, having 101 action categories. With 13320 videos from 101 action categories, UCF101 gives the largest diversity in terms of actions and with the presence of large variations in camera motion, object appearance and pose, object scale, viewpoint, cluttered background, illumination conditions, etc. The videos in 101 action categories are grouped into 25 groups, where each group can consist of 4-7 videos of an action. The videos from the same group may share some common features, such as similar background, similar viewpoint, etc. The action categories can be divided into five types: 1)Human-Object Interaction 2) Body-Motion Only 3) Human-Human Interaction 4) Playing Musical Instruments 5) Sports.

## 4.2 Evaluation Metrics

To our knowledge, there is no existing published paper or method that accommodates RSGAN architecture and its performace. open source framework that accommodates RFCNN architecture. So as the baseline, we utilize only Stage-I GAN of our Stack-GAN for generating $64 \times 64$ images to investigate whether the stack structure is beneficial. Then we modify our Stack-GAN to generate $128 \times 128$ images to investigate whether larger images by our method results in clearer images. And we input text at both stages for generating images of better quality.

It is difficult to evaluate the performance of generative models (e.g., GAN). Asking human annotators to determine the visual quality of samples is most intuitive and reliable. We also choose a recently proposed numerical assessment approach *inception score* [23] for quantitative evaluation,

$$I = exp\Big(\mathbb{E}_x D_{KL}\big(p(y|\mathbf{x})||p(y))\big)\Big)$$

where $x$ denotes one generated sample, and $y$ is the label predicted by the Inception model [28]. The intuition behind this metric is that good models should generate diverse but meaningful images. Therefore, the KL divergence between the marginal distribution $p(y)$ and the conditional distribution $p(y|\mathbf{x})$ should be large.

### 4.3 Training Methods and Implementation

From NTU RGB+D Dataset, we use only the 3D skeletal videos which are 2-5s long each and each video is labeled by an action label. Skeletal video frames are based on the locations of the detected body joints, and join them by a line. For good performance, we use mask to remove the background and less important parts of the depth maps and to improve the compression rate.

**Stangan Implementaion** The up-sampling blocks consist of the nearest-neighbor upsampling followed by a $3 \times 3$ stride 1 convolution. Batch normalization [10] and ReLU activation are applied after every convolution except the last one. The residual blocks consist of $3 \times 3$ stride 1 convolutions, Batch normalization and ReLU. Two residual blocks are used in $128 \times 128$ StackGAN models while four are used in $256 \times 256$ models. The down-sampling blocks consist of $4 \times 4$ stride 2 convolutions, Batch normalization and LeakyReLU [15, 33], except that the first one does not have Batch normalization.
By default, $N_g = 128, N_z = 100, M_g = 16, M_d = 4, N_d = 128, W_0 = H_0 = 64$ and $W = H = 256$. For training, we first iteratively train $D_0$ and $G_0$ of Stage-I GAN for 600 epochs by fixing Stage-II GAN. Then we iteratively train D and G of Stage-II GAN for another 600 epochs by fixing Stage-I GAN. All networks are trained using ADAM solver with batch size 64 and an initial learning rate of 0.0002. The learning rate is decayed to $\frac{1}{2}$ of its previous value every 100 epochs.

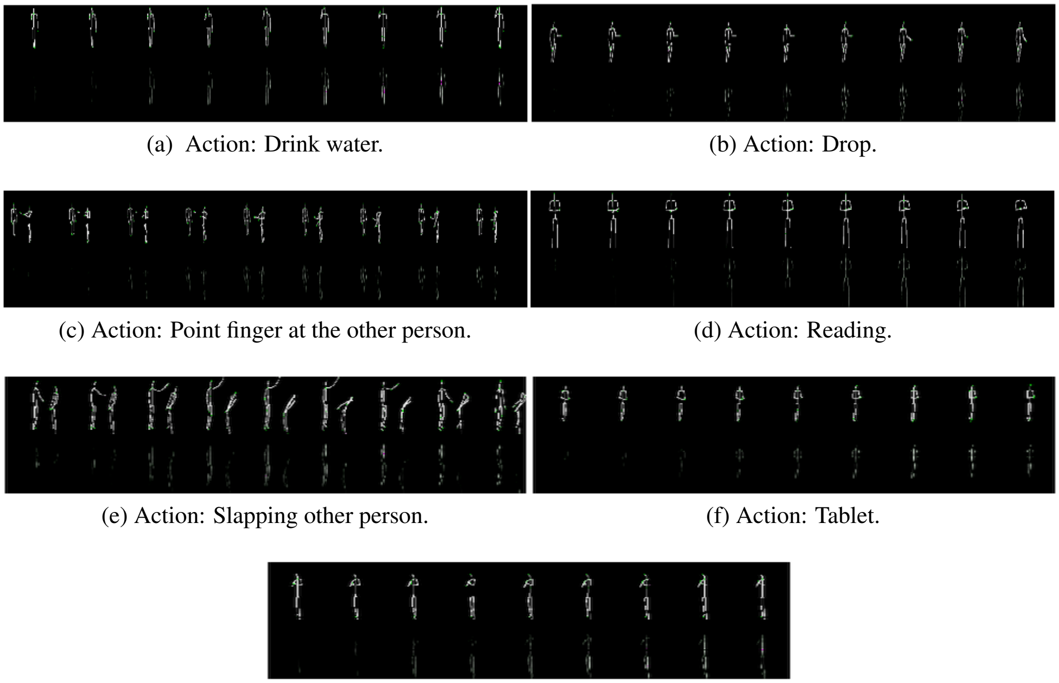
**Conv-GRU Implementaion** RFC-VGG in figure 6 is based on VGG-F [26] network. Initializing weights of our filters by VGG-F trained weights, alleviates over-fitting problems as these weights are the result of extensive training on the imagenet. The network is cast to a fully convolutional one by replacing the fully connected layers with convolutional layers. The last two pooling layers are dropped from VGG-F to allow a finer frame. Then a convolutional gated recurrent unit is used followed by one convolutional layer and then deconvolution for up-sampling. Figure 4 shows the detailed architecture of RFC-VGG.

| RFC-VGG | |
|---|---|
| input: 240×360 | |
| Recurrent Node | Conv: F(11), S(4), P(40), D(64) |
| | Relu |
| | Pool 3×3 |
| | Conv: F(5), P(2) D(256) |
| | Relu |
| | Pool(3×3) |
| | Conv: F(3), P(1) D(256) |
| | Relu |
| | Conv: F(3), P(1) D(256) |
| | Relu |
| | Conv: F(3), P(1) D(256) |
| | Relu |
| | Conv: F(3), D(512) |
| | Conv: F(3), D(128) |
| | ConvGRU: F(3), D(128) |
| ▨ | Conv: F(1), D(1) |
| | DeConv: F(20), S(8) |

Figure 6: Proposed Conv-GRU networks. $F(n)$ denotes filter size of $n \times n$. $P(n)$ denotes total of $n$ zero padding around the feature map. $S(n)$ denotes stride of length $n$ for the convolution. $D(n)$ denotes number of output feature maps from a particular layer $n$ for a layer (number of feature maps is same as previous layer if $D$ is not mentioned).

### 4.4 Qualitative Results

**NTU RGB+D Action Recognition Dataset** The main experiments are conducted using Adadelta [35] for optimization that practically gave much faster convergence than standard stochastic gradient descent. The logistic loss function is used and the maximum number of epochs used for the training is 500.


(a) Action: Drink water.

(b) Action: Drop.

(c) Action: Point finger at the other person.

(d) Action: Reading.

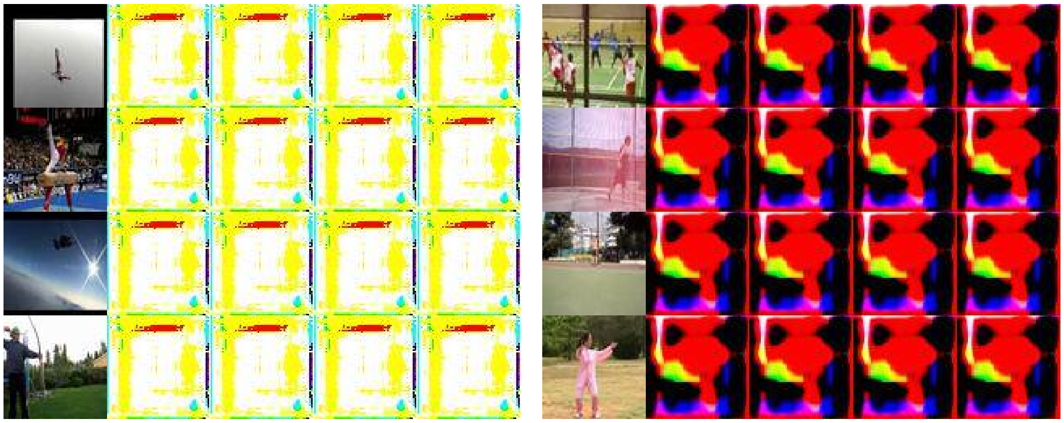
(e) Action: Slapping other person.

(f) Action: Tablet.

(g) Action: Writing.

Figure 7: Example results by our proposed RSGAN. The first row is the ground truth, and the second row is the output frame of RSGAN.

**UCF-101 Dataset** UCF-101 videos are more realistic and, has complex texture and sturctural relations among various objects. We train our RSGAN using first 90 action classes and later try to generate videos for other 11 classes. We input the word-vector representation of the action class names and use those as the pre-conditions to generate the videos.


(a)

(b)

Figure 8: Example results by our proposed RSGAN. The first coulmn is a ground truth frame of the input action class, and the other columns are the output frames of RSGAN.

## 5 Conclusion and Future Works

We have proposed a novel adversarial network based model to generate videos based on a given condition. Our model can generate video frames which are coherent and consistent to the given condition. Since no published work is still avaiable to address this pre-conditioned video, it is very difficult to compare performance with. Right now, for simple detailed video like, NTU RGB+D Action Recognition Dataset, RSGAN is the only and most likely model to generate somewhat consistant video frames. But for complex scene of UCF-101 Dataset, the result is very poor. In future, we would like to investigate whether applying batch normalization in GRU can imporve the result like it does in StanGan module. And if we get any success in that, then we will extend our pre-condition to VIM-2 dataset [18]. In this dataset, fMRI BOLD signals from human brain will be used to simulate video frames.

## References

[1] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions* , 5(2):157–166, 1994.

[2] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] C. Doersch. Tutorial on variational autoencoders. *arXiv:1606.05908*, 2016.

[5] J. Gauthier. Conditional generative adversarial networks for convolutional face generation. Technical report, Tech Report, 2015.

[6] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio. Nips. In *Generative adversarial nets*, 2014.

[7] K. He, X. Zhang, S. Ren, and J. Sun. Cvpr. In *Deep residual learning for image recognition*, 2016.

[8] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[9] L. Huang, Y. Yang, Y. Deng, and Y. Yu. Densebox: Unifying landmark localization with end to end object detection. *arXiv preprint arXiv:1509.04874*, 2015.

[10] S. Ioffe and C. Szegedy. Icml. In *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015.

[11] J. Johnson, A. Alahi, and L. Fei-Fei. Eccv. In *Perceptual losses for real-time style transfer and super-resolution*, 2016.

[12] D. P. Kingma and M. Welling. Iclr. In *Auto-encoding variational bayes*, 2014.

[13] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Icml. In *Autoencoding beyond pixels using a learned similarity metric*, 2016.

[14] G. Lin, C. Shen, I. Reid, and et al. Efficient piecewise training of deep structured models for semantic segmentation. *arXiv preprint arXiv:1504.01013*, 2015.

[15] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Icml. In *Rectifier nonlinearities improve neural network acoustic models*, 2013.

[16] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. *arXiv preprint arXiv:1511.05440*, 2015.

[17] M. Mirza and S. Osindero. Conditional generative adversarial nets. *arXiv:1411.1784*, 2014.

[18] S. Nishimoto, A. T. Vu, T. Naselaris, Y. Benjamini, B. Yu, and J. L. Gallant. Reconstructing visual experiences from brain activity evoked by natural movies. In *Current Biology*, volume 21(19), pages 1641–1646, 2011.

[19] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *Proceedings of the IEEE International Conference on Computer Vision*, page 1520–1528, 2015.

[20] S. Reed, Z. Akata, S. Mohan, S. Tenka, B. Schiele, and H. Lee. Nips. In *Learning what and where to draw*, 2016.

[21] S. Reed, Z. Akata, B. Schiele, and H. Lee. Cvpr. In *Learning deep representations of fine-grained visual descriptions*, 2016.

[22] Z. A. S. Reed, X. Yan, L. Logeswaran, B. Schiele, and H. Lee. Icml. In *Generative adversarial text-to-image synthesis*, 2014.

[23] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *NIPS*, 2016.

[24] A. Shahroudy, J. Liu, T.-T. Ng, and G. Wang. NTU RGB+D: A Large Scale Dataset for 3D Human Activity Analysis. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[25] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[26] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[27] K. Soomro, A. R. Zamir, and M. Shah. UCF101: A Dataset of 101 Human Action Classes From Videos in The Wild. *CRCV-TR-12-01*, 2012.

[28] I. Sutskever, J. Martens, and G. E. Hinton. Proceedings of the 28th International Conference on Machine Learning (ICML-11). In *Generating text with recurrent neural networks*, page 1017–1024, 2011.

[29] S. Valipour, M. Siam, M. Jagersand, and N. Ray. Recurrent Fully Convolutional Networks for Video Segmentation. *arXiv preprint arXiv:1611.09904*, 2016.

[30] O. Vinyals, S. V. Ravuri, and D. Povey. Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference. In *Revisiting recurrent neural networks forrobust asr*, page 4085–4088, 2012.

[31] C. Vondrick, H. Pirsiavash, and A. Torralba. Generating videos with scene dynamics. In *In Advances In Neural Information Processing Systems*, pages 613–621, 2016.

[32] L. Wang, W. Ouyang, X. Wang, and H. Lu. Visual tracking with fully convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, page 3119–3127, 2015.

[33] B. Xu, N. Wang, T. Chen, and M. Li. Icml workshop. In *Empirical evaluation of rectified activations in convolutional network*, 2015.

[34] X. Yan, J. Yang, K. Sohn, and H. Lee. Attribute2image: Conditional image generation from visual attributes. In *In European Conference on Computer Vision*, pages 776–791. Springer International, October 2016.

[35] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[36] H. Zhang, T. Xu, H. Li, S. Zhang, X. Huang, X. Wang, and D. Metaxas. StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks. *arXiv preprint arXiv:1612.03242*, 2016.