CS230

Visual Anomaly Detection in Video

Chris Wu
Department of Electrical Engineering
Stanford University
chris.wu@stanford.edu

Faraz Waseem
Department of Computer Science
Stanford University
faraz333@stanford.edu

Rafael Perez Martinez
Department of Electrical Engineering
Stanford University
rafapm@stanford.edu

Github URL: https://github.com/rafapm/CS_230/

1 Introduction

We demonstrate an unsupervised deep learning model to detect video anomalies for smart surveillance. In the context of our work, we define video anomaly detection as novel objects entering the frame of view, or objects with unusual trajectories and motion patterns. A formal definition was given in Saligrama *et al.* [1], where they defined video anomalies as "the occurrence of unusual appearance or motion attributes or the occurrence of usual appearance or motion attributes in unusual locations or times." Some examples include but are not limited to unattended/abandoned items for long periods of time, car accidents, a person biking through a crowd of pedestrians, or people committing crimes.

In particular, we are focusing on single-scene video anomaly detection since it has the most immediate use in real-world applications (e.g., surveillance camera monitoring of one location for extended periods), and it is also the most common use-case in video anomaly detection. For such applications, it is more time-efficient to have a computer do this task in comparison to a person since there is nothing interesting going on for long periods of time. In fact, this is the driving force behind modern intelligent video surveillance systems. By using computer vision analytics, it will not only increase the efficiency of video monitoring but t will also reduce the burden on live monitoring from humans. Although this problem has been studied thoroughly in previous works [2], it is far from being realized fully since the difficulty lies in modeling anomaly events as well as handling the sparsity of events in data sets. Furthermore, not much work has been done previously on generative approaches for video anomaly detection. We explore two versions of a variational autoencoder approach for anomaly detection in this project.

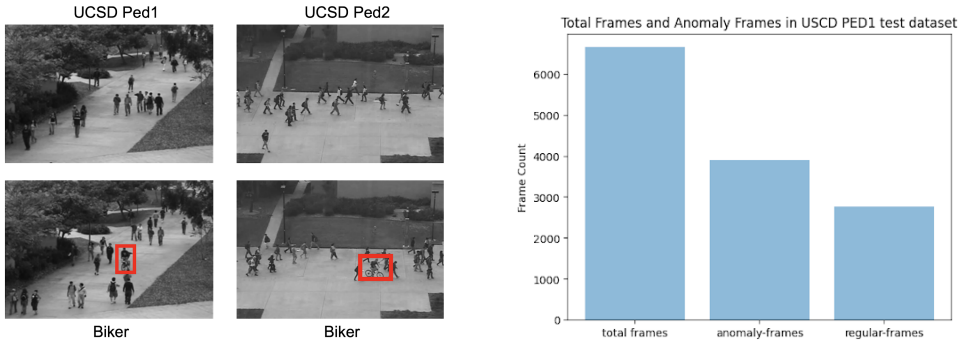
2 Data Description

There are currently several datasets available for single-scene video anomaly detection. These datasets include UCSD Ped1 & Ped2 [3], CUHK Avenue [4], Subway [5], UMN [6], and Street Scene [7]. For our project, we decided to use the UCSD dataset. We chose this dataset since it has been used extensively [8, 9], giving us reference points in terms of the accuracy that can be obtained from other works in literature.

There are important points to note in this data set. The first being that the crowd density does reach a high point that leads to severe occlusions. The second being that the anomalies in both subsets are considered to be any non-pedestrian object such as bikers, skaters, carts, and wheelchairs. Any

unusual motion from the pedestrians is also considered anomalous such as walking in a different pattern from what is considered normal (e.g., people walking across a walkway or at the grass).

The subset Ped1 consists of 34 training videos and 36 testing videos, all of which have a low resolution of 158 x 238 pixels. In the training videos, only normal pedestrian activity is shown. However, in the test dataset, each video clip contains at least some frames with anomalous activity. Notably, as shown in the bar chart of frame counts, the majority of frames in the test video clips contain anomalous activity (such as a biker within the frame for an extended period of time). This is not as ideal as having a more realistic distribution of anomalies in the test dataset since we expect surveillance video to be mostly normal.



(a) (b)

Figure 1: (a) Raw image frame showing a biker on sidewalk for both UCSD Ped1 and Ped2, and (b) Bar chart of total frame counts along with anomaly and regular-frames.

3 Approach and Architectures Explored

3.1 Data Preprocessing

Before we started building and training any models, we first took the training set and divided the training video frames into temporal sequences. These temporal sequences have a size of 10 using the sliding window technique. We then resized each frame to 256x256 to have the same resolution across all input images. The pixels were then scaled between 0 and 1 by dividing each pixel by 256.

Since the number of parameters is huge, we made use of data augmentation in the temporal dimension. More training sequences were generated by concatenating frames with different skipping strides.

3.2 Architectures

Description of layers and optimizers used in both architectures

1) TimeDistributed Layer: A TimeDistributed layer is used to allow us to send multiple input image frames (that are temporally separate) through the network, and produce a single output.

2) ConvLSTM2d Layer: A convolutional LSTM is similar to an LSTM 1D layer, but the input transformations and recurrent transformations are both convolutional. CONVLSTM layers will do a similar task to LSTM but instead of matrix multiplications, they will do convolution operations and retain the input dimensions.

3) Deconvolution Layer (Conv2dTranspose): Deconvolution layers are also called transposed convolution for upsampling and are usually used in autoencoders and GANs for image reconstruction. They usually make the output image larger in size (increase in dimension) by using a kernel which is learned using back-propagation similar to the kernel used in convolution layers.

4) Custom Lambda layer: In the VAE, we used a custom layer to create a representation of the latent distribution mean and standard deviation. In terms of initialization and training for all our architectures, we used the Xavier algorithm to prevent the signal from decaying to zero or "exploding" to a large value. To optimize our first architecture (LSTM Convolutional Autoencoder) we used a

mean squared error loss function, and Adam with a learning rate of 1e-4, decay of 1e-5, and epsilon of 1e-6. For the second and third model versions (LSTM Convolutional Variational Autoencoder), we used a special VAE loss function described later.

3.3 1st architecture: LSTM Convolutional Autoencoder

As a starting point, we first implemented a convolutional LSTM Convolutional autoencoder in Keras similar to the one implemented in [10]. For the autoencoder, we first defined the encoder and the decoder. The encoder takes as input a sequence of frames in chronological order. It was also divided into two parts, the spatial encoder, and the temporal encoder, where the output of the spatial encoder is the input of the temporal encoder to provide motion encoding. The purpose of the decode is to reconstruct the video sequence by mirroring the encoder. Architecture diagram of our baseline LSTM Convolutional Autoencoder:

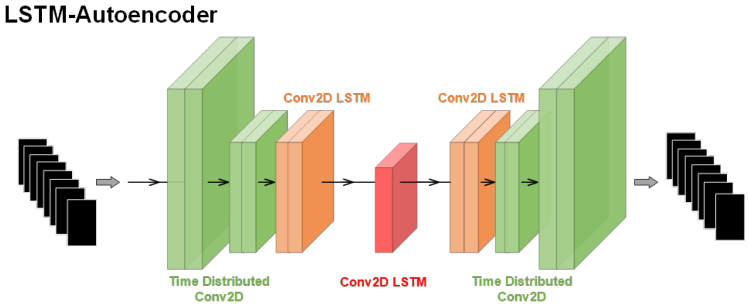


Figure 2: Architecture diagram of our baseline LSTM Convolutional Autoencoder (visual).

Model: "sequential"

Layer (type)	Output Shape	Param #
time_distributed (TimeDistri	(None, 10, 64, 64, 128)	15616
layer_normalization (LayerNo	(None, 10, 64, 64, 128)	256
time_distributed_1 (TimeDist	(None, 10, 32, 32, 64)	204864
layer_normalization_1 (Layer	(None, 10, 32, 32, 64)	128
conv_lst_m2d (ConvLSTM2D)	(None, 10, 32, 32, 64)	295168
layer_normalization_2 (Layer	(None, 10, 32, 32, 64)	128
conv_lst_m2d_1 (ConvLSTM2D)	(None, 10, 32, 32, 32)	110720
layer_normalization_3 (Layer	(None, 10, 32, 32, 32)	64
conv_lst_m2d_2 (ConvLSTM2D)	(None, 10, 32, 32, 64)	221440
layer_normalization_4 (Layer	(None, 10, 32, 32, 64)	128
time_distributed_2 (TimeDist	(None, 10, 64, 64, 64)	102464
layer_normalization_5 (Layer	(None, 10, 64, 64, 64)	128
time_distributed_3 (TimeDist	(None, 10, 256, 256, 128)	991360
layer_normalization_6 (Layer	(None, 10, 256, 256, 128)	256
time_distributed_4 (TimeDist	(None, 10, 256, 256, 1)	15489

Total params: 1,958,209
Trainable params: 1,958,209
Non-trainable params: 0

Figure 3: Architecture diagram of our baseline LSTM Convolutional Autoencoder (table).

3.4 2nd architecture: Variational LSTM Convolutional Autoencoder

To improve our baseline model which was based on a Convolutional autoencoder, we introduced a generative dimension in our architecture. For autoencoder (AEC) the latent space is not continuous and does not allow easy extrapolation. In the case of variational autoencoder (VAE) instead of vectors of single points, it uses mean and standard deviation. This gives the VAE richer representation power and they are much better at mapping input samples to this rich latent space. On other hand, AECs have gaps in the way they construct latent vectors to represent input space [12]. So instead of learning vectors as a bottleneck layer, we learned a Gaussian distribution which consists of both mean and standard deviation. The intuition behind this approach was that learning to minimize the difference

between input and output video distribution in addition to just reconstruction error will be more generic and will add a regularization effect. It will focus less on minor details of input videos and will try to learn a more generic representation of normal data. One intuition is that it will make the model work relatively better with less sensitivity threshold for reconstruction error. VAE are generative models and can be used to generate videos as well.

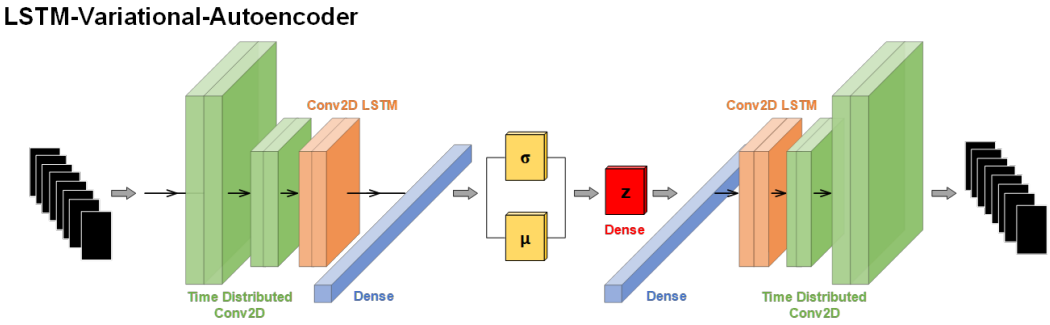


Figure 4: Architecture diagram of our baseline LSTM Variational Autoencoder (visual).

Model: "encoder"

Layer (type)	Output Shape	Param #	Connected to
encoder_input (InputLayer)	[(None, 10, 256, 256	0	
time_distributed_5 (TimeDistrib	(None, 10, 64, 64, 6	1664	encoder_input[0][0]
batch_normalization_8 (BatchNor	(None, 10, 64, 64, 6	256	time_distributed_5[0][0]
conv_lst_m2d_2 (ConvLSTM2D)	(None, 10, 64, 64, 1	46144	batch_normalization_8[0][0]
batch_normalization_9 (BatchNor	(None, 10, 64, 64, 1	64	conv_lst_m2d_2[0][0]
flatten_1 (Flatten)	(None, 655360)	0	batch_normalization_9[0][0]
dense_2 (Dense)	(None, 32)	20971552	flatten_1[0][0]
batch_normalization_10 (BatchNo	(None, 32)	128	dense_2[0][0]
latent_mu (Dense)	(None, 32)	1056	batch_normalization_10[0][0]
latent_sigma (Dense)	(None, 32)	1056	batch_normalization_10[0][0]
z (Lambda)	(None, 32)	0	latent_mu[0][0] latent_sigma[0][0]

Total params: 21,021,920
Trainable params: 21,021,696
Non-trainable params: 224

Model: "decoder"

Layer (type)	Output Shape	Param #
decoder_input (InputLayer)	[(None, 32)]	0
dense_3 (Dense)	(None, 655360)	21626880
batch_normalization_11 (Batc	(None, 655360)	2621440
reshape_1 (Reshape)	(None, 10, 64, 64, 16)	0
conv_lst_m2d_3 (ConvLSTM2D)	(None, 10, 64, 64, 16)	18496
batch_normalization_12 (Batc	(None, 10, 64, 64, 16)	64
time_distributed_6 (TimeDist	(None, 10, 256, 256, 64)	25664
batch_normalization_13 (Batc	(None, 10, 256, 256, 64)	256
time_distributed_7 (TimeDist	(None, 10, 256, 256, 1)	7745

Total params: 24,300,545
Trainable params: 22,989,665
Non-trainable params: 1,310,880

Figure 5: Architecture diagram of our baseline LSTM Variational Autoencoder (table).

One tricky problem we encountered with variational autoencoder is how backpropagation will work with a stochastic sampling layer. The solution is the reparameterization trick which works as follows.

We can compute the gradients of the sampling node with respect to the mean and log-variance vectors (both the mean and log-variance vectors are used in the sampling layer). This is represented in the following figure:

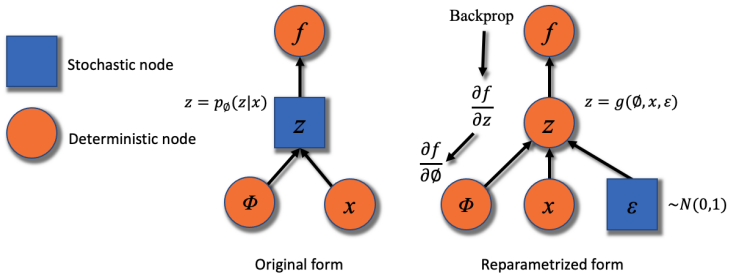


Figure 6: Architecture diagram of our baseline LSTM Variational Autoencoder (table).

Where phi is a learned neural network weight, and epsilon is drawn from a standard normal. We experimented with two versions of this model architecture, by changing the loss functions. Specifically, we modified the weight of the KL-divergence term in the loss function. The Loss function for VAE version 1 contains mean square error loss plus KL-divergence term (where Beta = 1) as shown in (1):

$$\mathcal{L} = \mathbb{E}_{q(z|X)}[\log p(X|z)] - \beta D_{KL}[q(z|X)||p(Z)]. \tag{1}$$

The Loss function for VAE version 2 contains a Beta weight term greater than 1, which has the effect of adding a higher penalty for regularization.

3.5 Experiments Run

We evaluated all three models on the UCSD pedestrian 1 dataset. Each model generated a regularity score output which beyond a preset threshold indicates an anomalous video frame. This regularity score was calculated as follows: To calculate the regularity score, we first computed the reconstruction error of a pixel's intensity with value **I** at the location (x, y) in the frame *t* of the video using L2Norm as shown in (2):

$$e(x, y, t) = \|I(x, y, t) - f_W(I(x, y, t))\|_2 \tag{2}$$

Where F_W is the learned model. The reconstruction error of a frame *t* was then computed by summing all of the pixel-wise errors:

$$e(t) = \sum_{(x,y)} e(x, y, t). \tag{3}$$

We then calculate the reconstruction cost of the 10-frames sequence starting at *t* using (4):

$$\text{sequence\_reconstruction\_cost}(t) = \sum_{t'=t}^{t+10} e(t'). \tag{4}$$

The anomaly score $s_a(t)$ was then computed by scaling the reconstruction cost between 0 and 1 as shown in (5):

$$s_a(t) = \frac{\text{sequence\_reconstruction\_cost}(t) - \text{sequence\_reconstruction\_cost}(t)_{min}}{\text{sequence\_reconstruction\_cost}(t)_{max}}. \tag{5}$$

Finally, the regularity score was obtained by subtracting 1 minus $s_a(t)$:

$$s_r(t) = 1 - s_a(t). \tag{6}$$

Using the regularity scores and preset anomaly thresholds, we computed False Positive Rate, True Positive Rate (Recall), Precision, and F1 score for each model, across a range of regularity score thresholds. This allowed us to compare model performance at different sensitivity thresholds.

4 Results and Analysis

For each input video clip, our models produce regularity scores such as the following (plotted with respect to video frame number):

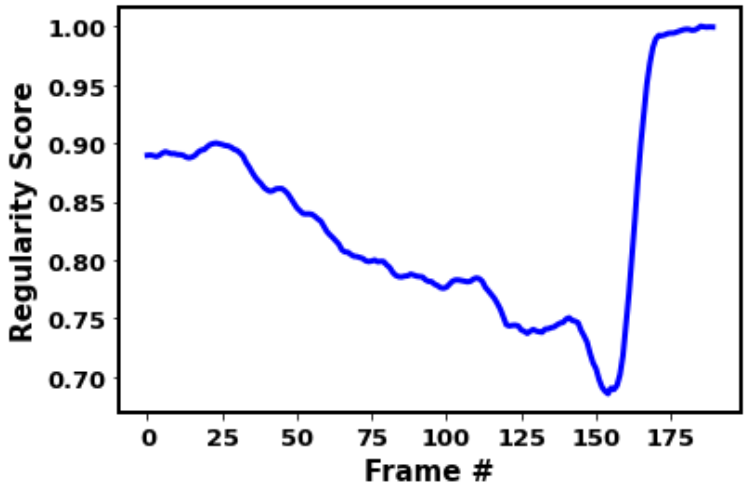


Figure 7: Plot of the regularity score of the stack of video frames containing the anomalous cart driving on the sidewalk.

In this particular example generated by the LSTM autoencoder baseline model, a golf cart drives across the sidewalk (i.e. anomaly); therefore, we see the regularity score decrease as the golf cart drives into the foreground. Lastly, after the golf cart exits the field of view, the regularity score goes back to 1.00 (i.e. normal activity).

Comparing the F1 score vs. regularity threshold for the three models:

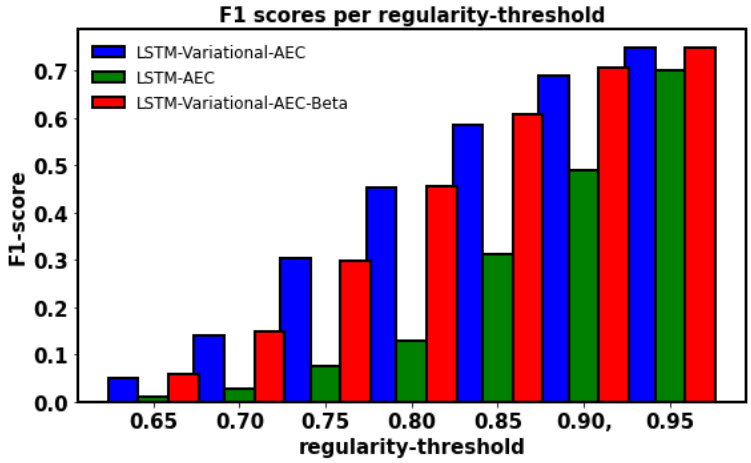


Figure 8: Comparison of the F1 score vs. regularity threshold for the three models.

As we can see in the above plot, the LSTM-VAE-Beta model has consistently the best performance (i.e. highest F1-score) across all the regularity thresholds we tested. Both the VAEs performed similarly and were superior in terms of F1 score compared to the LSTM AE. Second, as the regularity score increases, the performance of the three models all converge to about the same F1 score. This is because as the regularity threshold approaches 1.00, the models become increasingly sensitive to any frames that might be anomalous. Since our test dataset contains a high percentage of anomalous

frames (in fact, anomalous frames are the majority of frames), it makes sense that the hypersensitive model F1 scores will converge and begin to reflect the distribution of anomalous frames in the dataset. This is not practical behavior - in real use-cases, we would set the regularity threshold lower (around 0.85). In future work, it would be interesting to compare model performances on a different dataset.

We also plotted precision-recall curves for the three models:

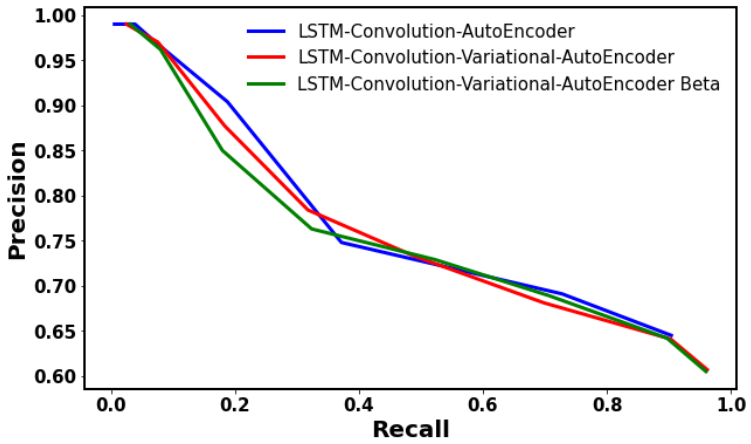


Figure 9: Precision-recall curves for the three models.

We generated the above precision-recall curve by recording precision and recall for each model over a sweep of regularity score thresholds.

5 Conclusion

We found that generative models (VAEs) are better in differentiating normal and anomaly videos by learning richer representations of the input. From plotting F1 scores against different regularity scores, we can see that our performance has increased by focusing more on minimizing KL divergence between input and output videos than just minimizing the mean square error between input and output video sequences (in the LSTM AE case). We can see that variational auto-encoder outperforms vanilla LSTM-Conv Autoencoder and Beta variational autoencoder outperforms variational autoencoder slightly. This suggests that a deeper understanding (through a Gaussian distribution) of the latent space of videos has helped the VAE models with performance on test data. As for the next steps, we will do extensive hyperparameter tuning like playing with the dimension of filters, time window size (we have taken 10 as default), as well as trying different beta factors for the VAE version 2. It would also be interesting to experiment with newer datasets with existing models.

6 Contribution

Faraz Waseem has contributed to original research (papers), data analysis, implementation of all three models and test runs on datasets, results in analysis, and report writing. Chris Wu and Rafael Perez wrote scripts for data analysis and helped with AWS setup and reporting.

7 Appendix

Table with results from experiments:

Models	F1	Precision	Recall/TPR	FPR	Regularity-Threshold
LSTM-Auto-Encoder	0.011	0.99	0.0059	0	0.65
LSTM-Variational-Auto-Encoder	0.049	0.99	0.025	0	0.65
LSTM-Variational-Auto-Encoder-Beta	0.06	0.99	0.0313	0	0.65
LSTM-Auto-Encoder	0.0278	0.99	0.014	0	0.70
LSTM-Variational-Auto-Encoder	0.141	0.97	0.076	0.0032	0.70
LSTM-Variational-Auto-Encoder-Beta	0.148	0.962	0.08	.004	0.70
LSTM-Auto-Encoder	0.076	0.99	0.039	0	0.75
LSTM-Variational-Auto-Encoder	0.304	0.87	0.184	0.036	0.75
LSTM-Variational-Auto-Encoder-Beta	0.298	0.85	0.180	0.043	0.75
LSTM-Auto-Encoder	.13	0.97	.072	0.002	0.80
LSTM-Variational-Auto-Encoder	0.452	0.784	0.318	0.122	0.80
LSTM-Variational-Auto-Encoder-Beta	0.455	0.763	0.324	0.141	0.80
LSTM-Auto-Encoder	.311	0.904	.188	0.027	0.85
LSTM-Variational-Auto-Encoder	0.587	0.733	0.489	0.25	0.85
LSTM-Variational-Auto-Encoder-Beta	0.609	0.729	0.523	0.272	0.85
LSTM-Auto-Encoder	0.49	0.748	.372	0.176	0.90
LSTM-Variational-Auto-Encoder	0.69	0.681	0.699	0.459	0.90
LSTM-Variational-Auto-Encoder-Beta	0.707	0.689	0.707	0.460	0.90
LSTM-Auto-Encoder	0.70	0.691	0.728	0.458	0.95
LSTM-Variational-Auto-Encoder	0.750	0.641	0.90	0.709	0.95
LSTM-Variational-Auto-Encoder-Beta	0.748	0.642	0.897	0.703	0.95
LSTM-Auto-Encoder	0.753	0.645	0.904	0.70	0.975
LSTM-Variational-Auto-Encoder	0.744	0.607	0.962	0.876	0.975
LSTM-Variational-Auto-Encoder-Beta	0.742	0.605	0.960	0.88	0.975
LSTM-Variational-Auto-Encoder	0.740	0.59	0.989	0.960	0.99
LSTM-Variational-Auto-Encoder-Beta-5	0.0398	0.99	0.0203	0	0.65
LSTM-Variational-Auto-Encoder-Beta-5	0.115	0.979	0.0612	0.0018	0.70
LSTM-Variational-Auto-Encoder-Beta-5	0.224	0.923	0.127	0.0148	0.75
LSTM-Variational-Auto-Encoder-Beta-5	0.389	0.804	0.256	0.087	0.80
LSTM-Variational-Auto-Encoder-Beta-5	.556	0.734	0.447	0.228	0.85
LSTM-Variational-Auto-Encoder-Beta-5	0.684	0.689	0.678	0.429	0.90
LSTM-Variational-Auto-Encoder-Beta-5	0.752	0.645	0.902	0.698	0.95
LSTM-Variational-Auto-Encoder-Beta-5					0.975
LSTM-Variational-Auto-Encoder-Beta-5					0.99
LSTM-Variational-Auto-Encoder-Beta-5					0.995

References

[1] V. Saligrama, J. Konrad and P. Jodoin, "Video Anomaly Identification," in IEEE Signal Processing Magazine, vol. 27, no. 5, pp. 18-33, Sept. 2010, doi: 10.1109/MSP.2010.937393.

[2] W. Sultani, C. Chen and M. Shah, "Real-World Anomaly Detection in Surveillance Videos," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 6479-6488, doi: 10.1109/CVPR.2018.00678.

[3] W. Li, V. Mahadevan and N. Vasconcelos, "Anomaly Detection and Localization in Crowded Scenes," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 36, no. 1, pp. 18-32, Jan. 2014, doi: 10.1109/TPAMI.2013.111.

[4] C. Lu, J. Shi, and J. Jia, "Abnormal Event Detection at 150 FPS in MATLAB," in IEEE International Conference on Computer Vision (ICCV). Sydney, Australia: IEEE, Dec. 2013, pp. 2720–2727.

[5] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz, "Robust RealTime Unusual Event Detection using Multiple Fixed-Location Monitors," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 30, no. 3, pp. 555–560, Mar. 2008.

[6] Unusual crowd activity dataset of University of Minnesota, available from http://mha.cs.umn.edu/proj events.shtml.

[7] B. Ramachandra and M. Jones, "Street Scene: A new dataset and evaluation protocol for video anomaly detection," in Winter Conference on Applications of Computer Vision (WACV), 2020.

[8] B. Ramachandra, M. Jones and R. R. Vatsavai, "A Survey of Single-Scene Video Anomaly Detection," in IEEE Transactions on Pattern Analysis and Machine Intelligence, doi: 10.1109/TPAMI.2020.3040591.

[9] S. Zhu, C. Chen, and W. Sultani, "Video anomaly detection for smart surveillance," arXiv preprint arXiv:2004.00222, 2020.

[10] H. Sellat, "Anomaly Detection in Videos using LSTM Convolutional Autoencoder," Medium, 16-Oct-2019. [Online]. Available: https://towardsdatascience.com/prototyping-an-anomaly-detection-system-for-videos-step-by-step-using-lstm-convolutional-4e06b7dcdd29. [Accessed: 17-May-2021].

[11] Abacus.AI, "The Intuition Behind Variational Autoencoders," Medium, 08-Jun-2020. [Online]. Available: https://medium.com/@realityenginesai/understanding-variational-autoencoders-and-their-applications-81a4f99efc0d. [Accessed: 04-Jun-2021].