Vision and Language Group Recruitment Challenge 2026 Report

Jaikaran Singh
Indian Institute of Technology, Roorkee

1. Introduction

Abnormal event detection is the task of identifying events or patterns in video streams that deviate from normal, expected behavior. Video Anomaly Detection (VAD) poses great challenges due to the vast number and variety of abnormal events that can occur in videos in different contexts. For example, a child running in a playground is not an anomalous activity in the context of a playground, however a man running through a crowd of people walking slowly is anomalous in the context of the speed of the crowd.

The project aims to solve the problem of manually going through hours and hours of video surveillance footage to find rare abnormal events, which is time consuming for a human. The project treats video anomaly detection as a computer vision task that aims to automatically flag events that deviate from normal behavior by assigning an 'anomaly score' to each frame of the video. The model is trained on the CUHK Avenue Dataset [1] which is famous for having difficult to predict anomalous events. The state-of-the-art methods of video anomaly detection are able to achieve just over 90% AUC score on this dataset. There are many other datasets useful for experiments related to video anomaly detection:- ShanghaiTech, UCSD Ped1, UCSD Ped2 [2][3] etc. but those are out of the scope of this project. The focus is on generalization and the ability to distinguish between standard background activity and significant deviations in a real-world environment.

Throughout the two weeks assigned for this challenge, I experimented with a lot of different architectures and techniques of solving detection problems in videos by taking inspiration from existing research on the topic of Video Anomaly Detection. In this process I implemented three research papers [4][5][6] with different architectures. I fully document the entire process of implementation, testing, tackling logical errors, failed attempts, challenges faced, and the final results along with inferences of each experiment through this report.

2. Dataset

All models were trained and tested on the CUHK Avenue Dataset. The dataset contains 16 training and 21 testing videos. The videos were captured in the CUHK campus with 30652 (15328 training, 15324 testing) frames in total.

Normal Events	Abnormal Events
	
	

Each video frame is coloured and has a resolution of 640x360.

The dataset provided for the challenge has slight variations. There are a total of only 9204 frames for training videos and 11706 frames for testing videos i.e. a total of 20910 frames, which is approximately $2/3^{rd}$ of the actual dataset. As a result, some frames are missing between a series of consecutive frames. Additionally, approximately 1195 images of the test video clips are vertically flipped. Many of the training as well as testing videos have grainy footage (random noise in pixel values). There are occasional camera shakes in a few videos as well. Hence the name 'Corrupted Avenue Dataset'. These variations pose challenges in accurate inference of the model however I've tried various techniques to minimize the effect of these differences.

3. Model Development

3.1 Data Preprocessing

Each frame of size 640x360 is resized to 128x128 to decrease processing times. The frames (RGB) are converted to grayscale (1 channel) and the pixel values are normalised to a scale of 0-1. In order to incorporate the occasional camera shakes and footage blur, frames are randomly blurred by applying Gaussian Blur and or rotated clockwise/anticlockwise by 2 degrees with probabilities 0.2 and 0.3 respectively. This ensures that the model learns to see these camera shakes as normal behavior. This prevents false positives.

3.2 Data Restructuring and Handling

The entire training dataset is partitioned into clips of 16 consecutive frames by running a sliding window over the frames with stride 2. Thus, adjacent clips have 14 overlapping frames. Stride 2 is chosen to generate enough clips for training the model. All in all, there are 4486 clips of 16 frames each. Each clip is a 4D tensor of shape 16x128x128x1.

3.3 Model Architecture

The network architecture is a reproduction of that proposed by the paper Spatio-Temporal AutoEncoder for Video Anomaly Detection (*Zhao* et al., 2017) [4] as shown in Fig 1. The model consists of a video encoder that takes in input video clips of 16 frames. The encoder has 4 3D convolutional layers to extract spatio-temporal features of the input clip. Each layer consists of 3D Convolution with kernel size 3x3x3 and stride 1x1x1. The input channel size is 1 for grayscale images. The output channel size of each layer is 32, 48, 64 and 64. The bottleneck is of the shape 64x2x16x16. Each 3DConv step is followed by 3D batch normalisation. A LeakyReLU activation function is applied after each batch normalization with a negative slope of 0.01. At the end of each layer, 3D max pooling with kernel size 2x2x2 and stride 2x2x2 is applied. The bottleneck containing an encoded form of spatio-temporal features is then passed through 2 decoder branches. Both branches have an identical architecture but with different weights and biases. The first decoder's job is to reconstruct the original input video clip, while the second decoder's job is to predict the next video clip following the input video clip. This ensures the model learns the spatial (appearance) features of the video clips as well as the temporal (motion) features. Each decoder branch has three 3D Deconvolutions of kernel size 3x3x3 and stride 2x2x2, where each deconvolution is followed by 3D batch normalisation and

LeakyReLU. At the end, a convolutional layer is applied to reduce the number of channels to 1, followed by a sigmoid activation function. This gives the reconstructed and predicted video clips at the end.

16 frames, it is harder to predict for example the 15th frame as compared to predicting the next or the 2nd frame after the input video clip. Hence the paper introduced time weights in the prediction loss.

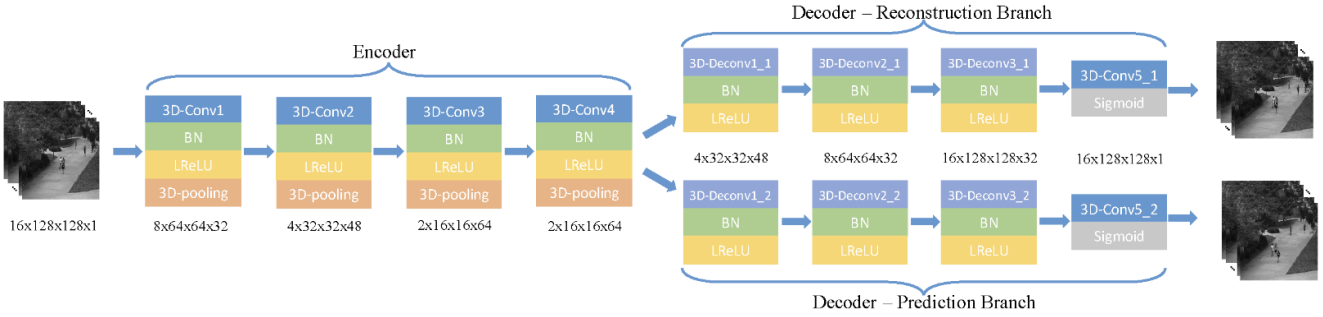


Fig 1: Spatio-temporal autoencoder architecture (reproduced from [4]).

3.4 Training

Loss Calculation: The two decoder branches use different losses during training. The reconstruction branch uses a pixel-wise Mean Squared Error (MSE) loss. The goal is to minimize this reconstruction loss so that the model learns to reconstruct spatial features like the background.

$$\mathcal{L}_{\text{rec}} = \frac{1}{h \cdot w \cdot n \cdot b \cdot t} \sum_{i=1}^{B} \|X_i - f_{\text{rec}}(X_i)\|_2^2$$

Reconstruction loss is the squared L-2 Norm of the difference between the input batch of video clips and the reconstructed batch of video clips. Here h and w represent the dimension of each frame, n is the number of channels, b is the number of clips in each batch and t is the number of frames in each clip. B represents the total number of batches.

The prediction branch uses a different kind of loss, called the **Weight-decreasing Prediction Loss** formulated in Spatio-Temporal AutoEncoder for Video Anomaly Detection (*Zhao* et al., 2017) [4, Sec 3.3]. The prediction loss guides the model to learn the trajectory of normal moving objects. While predicting the next

The paper formulates the loss as

$$\mathcal{L}_{\text{pred}} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{T^2} \sum_{t=1}^{T} (T-t) \left\| X_{i+T}^{t} - f_{\text{pred}}(X_i)^t \right\|_2^2$$

for T=16 and N video clips. This gives a higher weightage to the first few predicted frames compared to the end of the predicted video clip.

Time weights are only used while training, and are removed during inference to give a simple pixel-wise MSE loss.

While training, the objective is to try to minimize the sum of the reconstruction and prediction loss along with regularisation. [4, Sec 3.4]

$$\min_{W} \; L_{rec} + L_{pred} + \lambda \|W\|_2^2$$

Adam Optimizer is used with a weight decay of 1e-5 for L-2 regularisation.

A **mini-batch size** of **4** is used during training. This stacks 4 video clips of 16 frames each. A total of 1122 batches are thus trained.

The model was trained for **25** epochs on a learning rate of 1e-4 followed by another **15** epochs at a learning rate of 1e-5. Thus a total of **40** epochs. The model was trained using T4 GPU on Google Colab with an

average training time of 11 minutes per epoch.

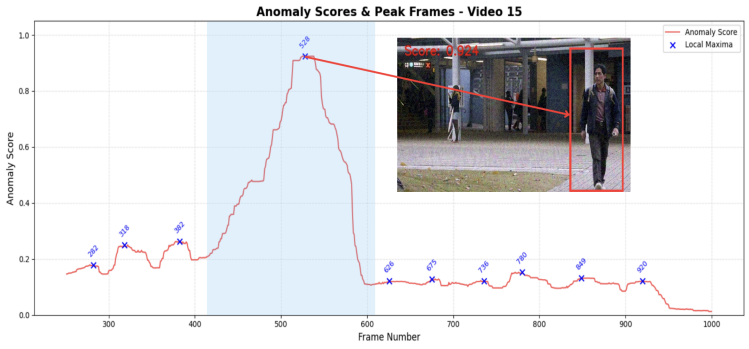
3.5 Inference and Results

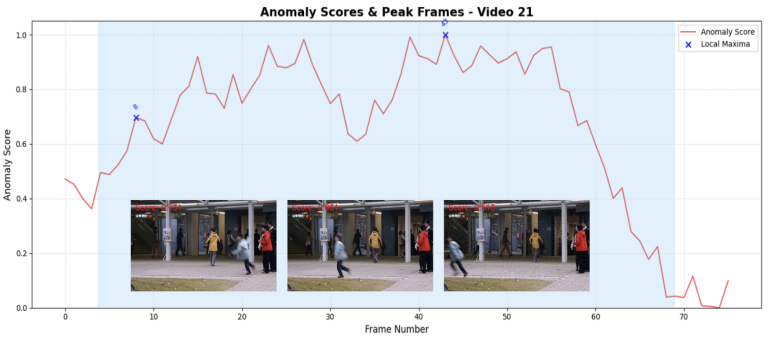
To handle the flipped frames in testing videos, these frames are unflipped before running inference. This is done by calculating the 'median' background of the test dataset. Each frame is then compared with this median background and with the flipped version of the median background. If the frame resembles the flipped version more, the frame is unflipped.

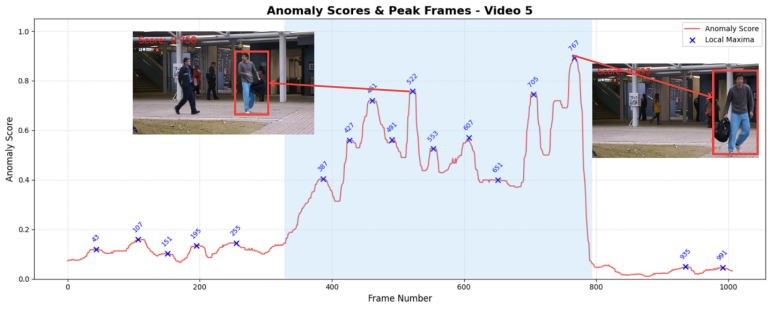
During inference, we recalculate the reconstruction and prediction loss without time weights. Then we assign an anomaly score to each frame. This is done by normalising the total loss using a min-max normalisation technique. This normalised loss is the anomaly score. Note that this normalisation is done separately for every video (1-21) not for the entire test set. This ensures that anomaly scores are assigned as per the context of each video since different anomalous events have a different anomaly 'influence'. A median filter with kernel size 17 is applied to these scores to prevent abrupt maximums or minimas in the scores. The figures below show the inference results of the anomaly detection model on the test videos.

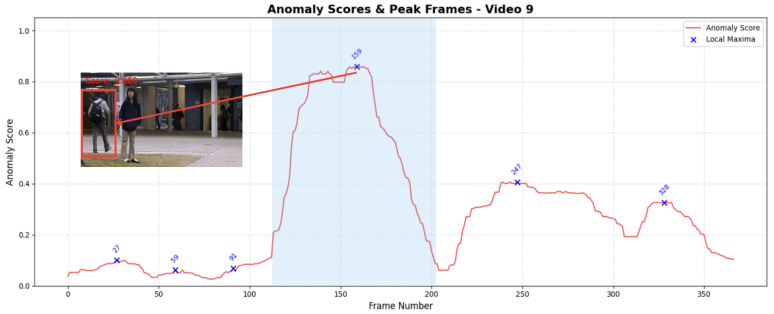
Final Loss: 0.01535

Some Examples of Successful Anomaly Prediction









Detects Small Object Anomalies: Falling Bag

Lower Anomaly Scores For Objects/People That Are Stationary For A Long Time

Stationary Bag on the ground

Girl Standing in the foreground

Carrybag on the ground, Women in Red

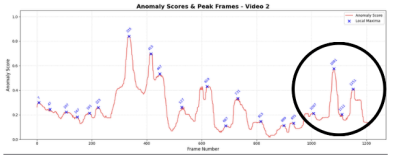
False Negatives

Man Throwing Bag

Man Entering For A Brief Moment

Video 13

False Positives



Slight Spike in Anomaly Score When Camera Shakes

4. Methods I've Tried and Why

4.1 A Simple 3D Convolutional AutoEncoder

When the challenge began, I learnt about autoencoders and how they work. The goal was to encode the input into a smaller latent space, and then make a reconstruction of the input by using a decoder on that latent space. By minimising the reconstruction loss, we can train the model to reconstruct the input accurately. Since the entire training dataset contains videos of 'normal' behavior, the model only learns to reconstruct normal video frames containing the background features, people in the background and the appearance of the foreground. The model would have a hard time reconstructing anomalous frames since it doesn't recognise them, thus giving them a higher anomaly score.

The first model I trained was based on this idea. It was a simple 3D Convolutional AutoEncoder with 4 encoding layers and 4 decoding layers. At that time I was using 128x128 RGB resized frames (3 input channels) output channels being 32, 48, 64, 64. Each convolution was followed by Batch Norm, LeakyReLU and 3D max pooling. The last layer of the decoder was a convolutional layer followed by a sigmoid activation. The model had just one decoder, which was the reconstruction decoder. I trained the model for around 7 epochs and started to notice that the model was not only able to reconstruct 'normal' frames but also the 'anomalous' frames! And it got better and better at doing that when I trained more. Early stopping was also not beneficial either because the model was learning to reconstruct normal and abnormal frames at almost the same pace whereas it was supposed to learn to reconstruct only the normal frames.

Fig 2 shows Row 1 with a set of anomalous input frames and Row 2 with the reconstruction of these anomalous input frames. There's quite some resemblance between the two. This led to a very low AUC score when I first made a submission (barely above the baseline score).

Fig 2: AutoEncoder able to reconstruct even abnormal frames

I soon realised that such a simple model would never actually learn the different types of anomalous events no matter how I tune the hyperparameters. This was because the model was learning only spatial features (appearance of objects). It didn't care about how the object moved, just how it looked. And that's exactly why it was reconstructing even the anomalous frames. Then I stumbled across the paper by *Zhao* et al., 2017 [4] which introduced the prediction decoder branch alongside the reconstruction branch which would help in capturing the motion of objects more accurately since it's harder to predict moving objects.

4.2 First Attempt At Implementing the 3D Convolutional Spatio-Temporal AutoEncoder (STAE)

With the introduction of the prediction decoder branch, I thought things would improve. Which they did! My AUC score increased to more than double my first score on the leaderboard. It seemed that the model was actually not able to perform well on anomalous frames. Which is

evident in Fig 3. But as I trained the model more, there were many problems that led to the saturation of my model.

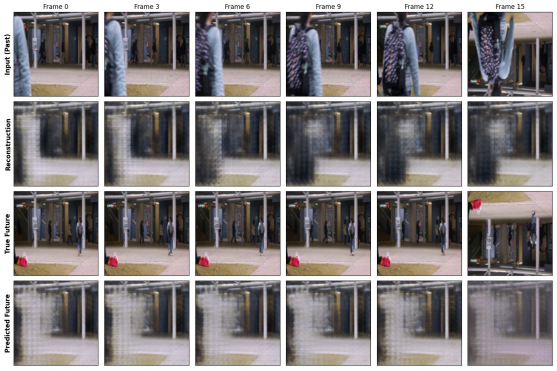


Fig 3: Row 1 has few frames of the input video clip. Row 2 has the reconstructed video clip. Row 3 has the actual future video clip (In this case, a bug in the dataset class, because the clip is not the immediate future, rather the distant future). Row 4 has the predicted future clip

After training the model long enough, I thought that by now the model would have learnt to reconstruct people in the background (no longer ghost like objects but a more detailed reconstruction of people). To my surprise, the model failed to reconstruct or predict the background people at all!

This was partly because I tried to make training faster by reducing the image size to 64x64 which already made the background people blurry. The main culprit was a huge logical flaw in my dataset classes related to the stride of the sliding window and how the future target clip was being selected! In Fig 3. The 3rd row is the distant future rather than the immediate future. Turns out my model was trained to predict the distant future rather than the immediate future due to the bug.

Input:

Reconstructed (No background people):

In addition to this, I used a stride of 4 in the sliding window, generating only about 1100 video clips for training. That's quite less, especially for RGB images with more channels. Thus the model quickly learnt to reconstruct the buildings, the grass, the pillars, the walls of the background. The model completely ignored the people in the background because it already performed well on the static objects. By this time I also realised about the flipped frames in the test set, so I unflipped those first before running inference.

Due to the model not learning about background motion, it failed to detect anomalies in the background completely. For instance, the man running in the background was considered normal.

The moving object was also noisy and blurry, resulting in the model not being able to detect it. However, the model was able to detect the running man in the foreground (bright pathway). This proved that the model had only learnt to reconstruct the foreground properly and not the background (the dark hallway)

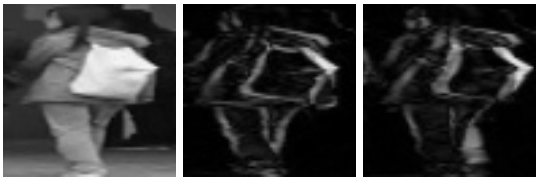
And in general, it was only able to detect anomalies in the foreground. (Scores: 0.782 Left, 0.664 Right).

My model was saturated, and I decided to come back to this method after trying some other model architectures.

4.3 Object-Centric AutoEncoders and Dummy Anomalies

Thanks to a github repository of awesome video anomaly detection methods [7] with tons of papers related to video anomaly detection, I came across the paper *Object-centric Auto-encoders and Dummy Anomalies for Abnormal Event Detection in Video* by Ionescu *et al.* (2019) [5], which introduces an object-centric convolutional auto-encoder framework and a one-versus-rest abnormal event classifier for detecting anomalies in video sequences.

The paper was originally implemented in TensorFlow, however I decided to use PyTorch. For this method, I first pre-processed the training data by running a pre-trained object detection model (Faster R-CNN model with a MobileNetV3-Large FPN backbone) on all frames of the training videos. This helped focus only on objects that may have a tendency to move (potential anomalies) rather than reconstructing large amounts of static pixels (static background). Hence adapting to the 'object-centric' approach, hoping to solve the problem of not being able to detect background people.



The image patch contained in the bounding box of each object was resized to 64x64 and converted to grayscale. In order to capture the motion of each object, the same bounding box of frame 't' was used on frame 't-3' and 't+3' and then subtracting patches 't' and 't-3', 't+3' and 't' gave 2 gradients which represented the motion of the object in the past and future of frame 't'.

The proposed architecture has 3 autoencoders. One processes the appearance of the object and the other two process the motion (gradients) of the object. I trained these auto-encoders at a learning rate of 1e-4 for 100 epochs followed by another 100 epochs at a learning rate of 1e-5. Using a pixel-wise MSE loss function. *[5, 1. Introduction, Figure 1]*

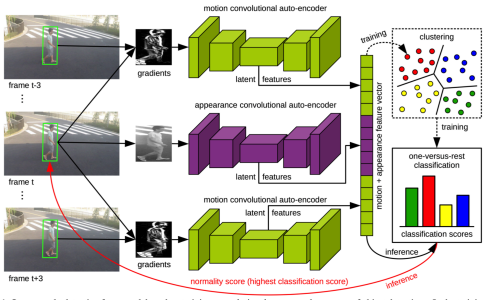


Figure 1. Our anomaly detection framework based on training convolutional auto-encoders on top of object detections. In the training phase (represented in dashed lines), the concatenated motion and appearance latent representations are clustered and a one-versus-rest classifier is trained to discriminate between the formed clusters. In the inference phase, we label a test sample as abnormal if the highest classification score is negative, i.e. the sample is not attributed to any class. Best viewed in color.

After the autoencoders were done learning spatial and temporal features of the objects. They were capable of accurately encoding the inputs to a latent space of size 16x8x8. These latent features were then flattened and stacked to form a vector of 3072 dimensions. There would thus be a unique feature vector with 3072 features

for every object (like bags, persons etc.) in all frames. In the end there were about 90,000 such feature vectors.

Next I ran k-means clustering (k=10) on these 90,000 feature vectors to make 10 different clusters. These clusters represent 10 different types of 'normal' behavior of objects. With the help of this method, I was able to create synthetic labels for 10 types of normal events. This adds some extent of supervision to the model.

Next I trained 10 different binary classifiers, one for each synthetic label. The idea was that for 1 normal behavior, the other 9 could be considered as dummy anomalies for training these binary classifiers i.e. Normal of one kind or not normal.

By running all these binary classifiers on a particular object, scores for each label were obtained. If all scores were negative, then the object doesn't belong to any type of normal behavior i.e. an anomaly. The max of these scores were chosen and negated. Where 'g(x)' is a binary classifier with x as an input sample (3072 dim feature vector). [5, 3. Method, (3)]

$$s(x) = -\max_i \{g_i(x)\}, \forall i \in \{1, 2, ..., k\}.$$

This was done for every object in a frame. And then the max of all object anomaly scores was chosen to be the anomaly score for that frame. This was repeated for every frame in the test data.

At the end, a min-max normalisation on the anomaly scores was done to bring the scores to a scale of 0-1.

It was expected that this method would work as expected. But unfortunately it did not! When I visualised the graphs corresponding to anomaly scores of each video, I noticed that it was just random noise. This was disappointing but after a lot of attempts trying to debug the problem, I figured that it would not work.

Since the dataset is already 'corrupted' with noise, motion blur, camera shakes etc. It would be almost impossible to encode object patches into meaningful latent spaces no matter how long I train the autoencoders. And because of this, it becomes difficult to separate out latent spaces into meaningful clusters. The t-SNE plot of these clusters also confirmed this. The plot contains 5000 samples.

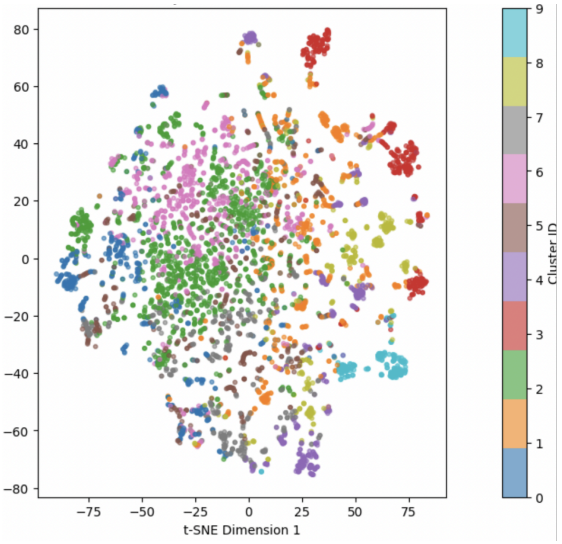


Fig 4: Generated t-SNE Plot of Clustered Latent Spaces

The plot shows a lot of imperfections in clustering which are difficult to overcome. There clusters didn't seem to form islands, many clusters were intersecting each other, some clusters had scattered samples. Thus I decided to move on from this method.

4.4 Spatio-Temporal AutoEncoder with ConvLSTM Layers

I came across the paper *Abnormal Event Detection in Videos using Spatiotemporal Autoencoder* by Yong Shean Chong and Yong Haur Tay [6], which proposes an unsupervised spatiotemporal autoencoder architecture for video anomaly detection. The model learns regular motion and

appearance patterns and detects anomalies based on reconstruction error, where events that deviate from learned patterns yield higher error scores.

This was aligned with the Spatio-Temporal AutoEncoder (STAE) that I had previously [4.2] trained and got a decent AUC on.

In the previous STAE [4.2] that I trained, the temporal features were learnt using a prediction decoder branch which was trained by minimising a weight-decreasing prediction loss. This paper [6], on the other hand, suggested a different approach. The autoencoder starts off with a spatial-encoder followed by a temporal encoder. Then a temporal decoder followed by a spatial decoder. The preprocessing was quite similar to the previous STAE.

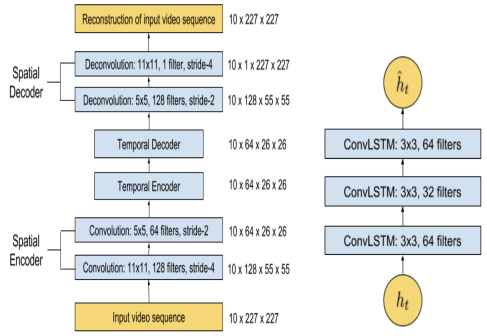


Fig 5: Left: Model architecture; Right: Temporal Encoder

The model takes in an input video clip of 10 frames. These frames are sampled by running a sliding window over the training dataset with a dilation of 1, 2 and 3. It is passed through 2 layers of convolution and the Tanh activation function.

The temporal encoder has 3 convolutional LSTM layers for learning motion features inside the input video clip. LSTMs are good at processing time-series data, thus ConvLSTMs are quite effective in handling videos.

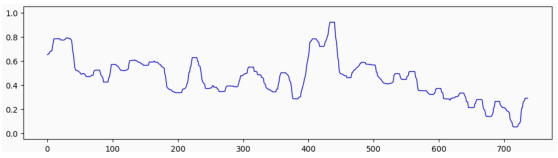
The decoders mirror the encoders in reconstructing the input video clip.

The reconstruction loss was a pixel-wise MSE loss. I used the Adam optimizer.

After training for 30-35 epochs, the loss was almost saturated. Inference was done in the same way as the previous STAE. The final result was slightly worse than the previous STAE [4.2] that I implemented.

I noticed that the model was performing poorly on a few videos of the test dataset, where everything was somehow being detected as an anomaly, video 03 was one such video.

Jagged Anomaly Score Plot; Video 03



While in a lot of the videos, anomalies were being detected at the right times. The exact underlying cause of why this model didn't perform well is still left to be explored.

After this attempt, I went back to my previous STAE implementation and improved it by first converting RGB images to grayscale and performing data augmentation by adding blur and rotations to ensure that the model learns to see through motion blur and prevent camera shakes from being flagged as anomalies.

4.5 Attention-based Residual AutoEncoder for Video Anomaly Detection

I figured that if I really want to ensure that the model focuses on moving objects rather than static background objects (motivated by my previous attempt at object-centric autoencoders), I would have to use attention in my model.

Le and Kim (2023) proposed an *Attention-based Residual Autoncoder for Video Anomaly Detection* [8]**.** The model's goal is to predict the next video frame given a set of consecutive video frames. The architecture consists of a convolutional encoder that extracts spatio-temporal features from consecutive video frames using a dual-branch design: a spatial branch for appearance modeling and a temporal branch based on a **temporal shift** operation to capture motion efficiently. These features are fused and passed to a decoder that predicts the next frame, where **residual channel attention** blocks are applied at multiple stages to emphasize informative feature channels and suppress irrelevant ones.

Due to time constraints and also limited GPU resources, I could not implement the methods proposed by this paper. However it was an informative read, given that I'm still learning more about the attention mechanism.

5. Conclusion

This two-week long contest offered an intensive experience of video anomaly detection problems, emphasizing the challenges of learning normal spatio-temporal patterns and reliably identifying rare abnormal events. Through rigorous experimentation, I was able to build a model capable of detecting different types of abnormal events accurately in various video footage. Achieving an inference time of 5 minutes for all 21 videos (11706 frames). That roughly equates to 40 frames per second when run on T4 GPU (Google Colab). There were many challenges throughout the competition, but even more learning outcomes.

5.1 Challenges

5.1.1. Because of the wide variety of abnormal events that occur in the Avenue dataset, it was difficult to predict each and every anomaly perfectly. There still exist quite a few false positives and false negatives during model inference.

5.1.2. Limited GPU resources slowed down training times and posed limits to how well I could experiment with different architectures and hyperparameters of the model. In order to reduce computation power usage, I had to resize images to a smaller resolution of 128x128, convert them to grayscale equivalent and use smaller convolutional layers in my network.

5.1.3. The 'corrupted' dataset posed many difficulties during inference. Flipped frames had to be unflipped. Occasional camera shakes and motion blur had to be taken into account by introducing small amounts of blur and rotation to the training examples so that the model learns all varieties of camera behavior. One thing that I perhaps failed to tackle was noise (grain) in video frames. I tried Gaussian filters and Bilateral filters but I could have used better noise reduction techniques like Noise2Void [9].

5.1.4. Assigning anomaly scores was a difficult task in itself. There were many choices to try out. For example min-max normalisation of reconstruction+prediction losses, this could be done either on the entire test set scores or separately for each video. I chose to do it for each video separately since every video has a different level of reconstruction+prediction loss for different severities of abnormal behavior. Thus it was more intuitive to choose min

and max values of losses for every video rather than the global min and global max. It was also a challenge to choose a hyperparameter that controls how much of the prediction loss is to be taken into account for training and inference. Since the prediction loss is usually much higher than the reconstruction loss, it was important to set a weightage for how much the prediction loss would contribute to the total loss. However, I decided to go with equal weightage for both reconstruction and prediction loss i.e. just summing both losses to get the total loss.

5.2 Learning Outcomes

5.2.1 This is one of the first convolutional neural networks that I have trained for a computer vision task. Through this competition I was able to get experience of practical applications of 3D convolution operations in neural networks for learning features from video data. I used PyTorch for this project.

5.2.2 Explored unsupervised approaches of video anomaly detection using auto-encoders, ConvLSTMs etc. as well as a semi-supervised approach using autoencoders, k-means clustering and binary classifiers.

5.2.3 Learned how to tune different hyperparameters for attaining the best model convergence and analyse the loss curve.

5.2.4 I learned how to handle image data efficiently using the pillow library (Python Imaging Library), handling csv files with pandas, and debugging visual features and graphs using matplotlib in Python.

5.2.5 Learned to analyze and mitigate common challenges such as false positives, training plateaus, and sensitivity to scene-specific motion patterns.

6. References

[1] Avenue Dataset for Abnormal Event Detection https://www.cse.cuhk.edu.hk/leojia/projects/detectabnormal/dataset.html

[2] Liu, W., Luo, W., Lian, D., & Gao, S. (2018). *ShanghaiTech Campus dataset (Anomaly Detection)*. Provided by SVIP-Lab. Based on videos from ShanghaiTech University. https://svip-lab.github.io/dataset/campus_dataset.html

[3] University of California, San Diego, *UCSD Anomaly Detection Dataset*, Statistical Visual Computing Lab. http://www.svcl.ucsd.edu/projects/anomaly/dataset.htm

[4] Y. Zhao, B. Deng, C. Shen, Y. Liu, H. Lu and X.-S. Hua, "Spatio-Temporal AutoEncoder for Video Anomaly Detection," in *Proceedings of the 25th ACM International Conference on Multimedia (MM '17)*, Mountain View, CA, USA, Oct. 2017, pp. 1933–1941, doi: 10.1145/3123266.3123451.

[5] R. T. Ionescu, F. Shahbaz Khan, M.-I. Georgescu, and L. Shao, "Object-centric Auto-encoders and Dummy Anomalies for Abnormal Event Detection in Video," *arXiv*, Dec. 11, 2018. [Online]. Available: https://arxiv.org/abs/1812.04960

[6] Y. S. Chong and Y. H. Tay, "Abnormal Event Detection in Videos using Spatiotemporal Autoencoder," *arXiv*, Jan. 6, 2017. [Online]. Available: https://arxiv.org/abs/1701.01546

[7] vt-le, "Video-Anomaly-Detection," GitHub repositoryhttps://github.com/vt-le/Video-Anomaly-Detection/

[8] V.-T. Le and Y.-G. Kim, "Attention-based residual autoencoder for video anomaly detection," *Applied Intelligence*, vol. 53, no. 3, pp. 3240–3254, 2023, doi: 10.1007/s10489-022-03613-1

[9] A. Krull, T.-O. Buchholz, and F. Jug, "Noise2Void – Learning denoising from single noisy images," *arXiv*, Nov. 26, 2018. [Online]. Available: https://arxiv.org/pdf/1811.10980