A Report on LSTM and Convolutional Autoencoder based Abnormality Detection for Heterogeneous Autonomous Systems

Himaddri Roy, Shafin Bin Hamid, Munshi Sanowar Raihan, Prasun Datta, Ashiqur Rasul,
Md. Mushfiqur Rahman, K M Naimul Hassan, Mohammad Ariful Haque*
Department of Electrical and Electronic Engineering
Bangladesh University of Engineering and Technology
*arifulhoque@eee.buet.ac.bd

CONTENTS

List of Figures

List of Tables

A Report on LSTM and Convolutional Autoencoder based Abnormality Detection for Heterogeneous Autonomous Systems

***Abstract*—Automatic self-abnormality detection in autonomous systems is becoming increasingly important. We systematically study the anomalous behavior of autonomous systems and reformulate the task as detecting instances that do not fit seamlessly into the existing distribution. We propose an LSTM (Long Term Short Term Memory)-Autoencoder based approach that captures the sequential nature of the multivariate time series data generated by the Inertial Measurement Unit (IMU) sensor. Our LSTM model is capable of making diagnosis and pinpointing the features responsible for abnormality. We also propose a Convolutional-Autoencoder model that operates on the optical flow map of the images captured by the system camera. We utilize on the optical flow features instead of directly using raw images because they better represent the motion state of the system. We have found both our models to show promising results.**

I. Introduction

Given the nature of intelligent autonomous systems, their anomalous behavior do not have a consistent structure in the feature space. This makes it a hard task for binary classification algorithms to learn a class boundary. Therefore, a better way to map out the problem is to use an unsupervised learning algorithm that attempts to model normal behavior in order to classify new examples as either or abnormal. Most of the current unsupervised methods are not directly applicable for anomaly detection in autonomous systems because of two reasons.

1) There exists a temporal dependency in multivariate time series data. As a result, one class classification algorithms e.g., One-Class Support Vector Machine, Isolation Forest, density estimation methods e.g.,Multivariate Gaussian Estimation, representation learning methods e.g., fully connected autoencoders may not perform well since they can not capture temporal dependencies across different time steps.
2) In a real world application, it is meaningful to provide a continuous measure of abnormality based upon severity of incidents instead of a binary normal or abnormal prediction. But the output of systems like One-Class Support Vector Machine only has a binary interpretation. On the other hand, fully connected autoencoder has an unbounded reconstruction loss that makes it hard to interpret the severity.

An accurate detection is crucial to avoid serious failures in autonomous systems. In addition to that, pinpointing the root causes, i.e., identifying which sensors are referring to anomaly, can be helpful to perform a system diagnosis. To overcome the shortcomings of current methods, we propose an LSTM based autoencoder that is capable of both anomaly detection and diagnosis. It addresses three issues jointly: anomaly detection, root cause identification and anomaly severity interpretation in continuous scale. The encoder LSTM takes a vector valued input and encodes it to a fixed feature vector. This encoded representation is then repeated and used as input to a decoder LSTM. The output is used to define reconstruction loss per feature. These losses for every feature are indications of source of anomaly because features responsible for anomaly produce high reconstruction loss whereas reconstruction losses for other features are below a certain threshold. We define a compound function parameterized by the training statistics to map the reconstruction losses to continuous anomaly score which represents the acuteness of abnormality.

For detecting anomalies from images captured by the autonomous system we define a separate model. Our problem is unique in the sense that, from the video feed, we are looking for system anomaly (e.g.,whether a drone is imbalanced or jerking) whereas current methods opt to recognize anomaly in the environment (e.g., appearance of an intruder at night). A lot of recent works have been done in finding environmental anomaly from the image's semantic content. But finding system anomaly from video feed is not a well-studied problem. This is a particularly hard task because no matter what the semantic content of the image, we are only interested in the motion state of the drone capturing the video. So, to capture the motion state, we use an optical flow estimation algorithm that returns a dense flow map representing the flow vectors of each pixel. Then we train a Convolutional-Autoencoder only on the normal flow frames. While testing, the reconstruction losses are mapped to an anomaly score using the same transformation used in LSTM-Autoencoder. Both sensor and image models are tested to demonstrate superior performance over classical and hybrid models (deep learning models paired with one class classification algorithms)

II. Problem Definition

In data science, anomaly detection is the identification of rare items, events or observations which raise suspicions by differing significantly from the majority of the data. These anomalous samples or items result in problems like system defects, bank fraud, erroneous word in texts . Unsupervised anomaly detection techniques detect anomalies in an unlabeled test data set under the assumption that the majority of the instances in the data set are normal by looking for instances that seem to fit least to the remainder of the dataset. [1]

By definition, a heterogeneous autonomous system operates without any manual control and has a multi-core architecture as well as multiple ways of accumulating data. The application of this type of system has gained significant momentum across many domains such as for real-time monitoring, remote sensing, inspecting disaster area, delivery of goods, surveillance, agriculture and several other disciplines. This type of system has the potential of acquiring data with fast pace. (Unsupervised anomaly detection in unmanned aerial vehicles,Author:Samir Khan a, Chun Fui Liew a, Takehisa Yairi a, Richard McWilliam).For instance, unmanned ground vehicles (UGVs) are very useful for various tasks of reconnaissance and transportation. In this case, human operators are greatly supported and do not have to enter possibly dangerous areas personally. [2]

An embedded sensor network is a network of embedded computers placed in the physical world that interacts with the environment. These embedded computers, or sensor nodes, are often physically small, relatively inexpensive computers, each with some set of sensors or actuators. Some of the embedded sensors used to create the dataset for this problem include Inertial Measurement Unit (IMU) sensors and digital camera placed on the system. [3]

The problem requires us to build a model that can detect abnormalities from unlabeled data collected from embedded sensors placed on a heterogeneous autonomous system that is in motion. The model can be trained only on normal data that is known to contain no abnormality. Once the model has been trained, it should be able to declare abnormality in every timestamp of IMU and camera synchronized data in real time. The model should preferably determine the reason behind the abnormality and also give a continuous score indicating the degree of abnormality.

Before we move on to our proposed solutions for this problem, we believe now is a good time to introduce the reader with the dataset used in this work. The details on which tools were used to extract data will be discussed in the later sections, but here we provide a general description of the provided dataset.

The dataset for this work consisted of rosbag files containing several topics. The topics comprised of messages that represent time series data. There were 6 such rosbag files that were known to contain no abnormality and 6 that contained both normal and abnormal data. The various topics of messages were IMU data, magnetic field data, velocity of the drone, corresponding odometer data of the drone, different sensor values and also corresponding image data captured by the drone camera. But for our work, we only utilized the IMU data synchronized with image data from camera.

III. Proposed Solutions

We concurred that the two broadly defined classes of data, IMU sensor data and images from camera were, although synchronized and aligned in time, but very different in nature. We pursued several methods to train on both types of data separately. Following is a discussion on what those methods are and what motivated us to choose them to find a solution to our problem and the results obtained by these methods are summarized in the next section:

A. Feedforward Autoencoder for IMU Sensor Data

Autoencoder is a good choice for learning efficient data coding of the normal examples in an unsupervised manner. An autoencoder learns how to regenerate the same examples that have been fed to it. By training on normal data and reducing the feature dimension, the autoencoder learns a consistent representation of normal behavior of the drone motion and ignores the signal "noise". Then, it can easily separate abnormal instances as those that do not fall under its own description of normal instances. Since we are working in the field of unsupervised learning, this simple feedforward model, shown in Fig. 11 seemed a good fit and a place where we can begin.

For the sensor measures, we considered ten features under the topic 'IMU data, orientation computed by FCU' that were fed to the input of the encoder (Orientation in X, Y, Z and W direction, linear acceleration in X, Y, Z direction, angular velocity in X, Y, Z direction). The linear velocity data was not used because it fell under the topic of 'Drone Velocity' and the timestamps of linear velocity did not coincide with that of the other messages under the topic of IMU data indicating linear velocity is not a part of IMU measurement.

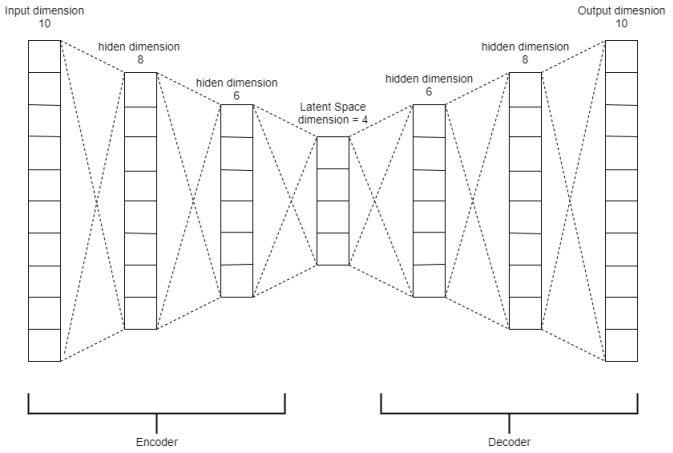


Fig. 1: The Feedforward Autoencoder architecture. The encoder stage compresses the input vector by a factor of 2.5. This enforced bottleneck in the network ensures compressed knowledge representation of the original input.

As has been previously mentioned, the normal instances are reconstructed at the end of the decoder stage using only the information from the latent variables. So, it is expected that these latent space variables themselves capture the structure useful for discriminating abnormal instances from the normal ones. We also test this hypothesis using our hybrid model, shown in Fig. 2

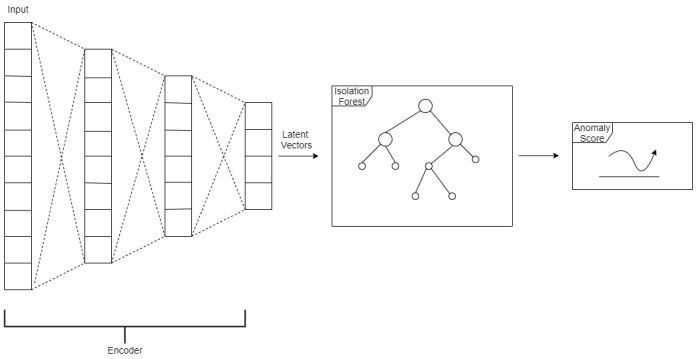


Fig. 2: Our hybrid model uses latent variables of the autoencoder as features for the Isolation Forest algorithm.

*B. LSTM Autoencoder for Sensor Data**

While dealing with sequential data, i.e. time series data, the most popular application is Long Short-Term Memory networks, or in short, LSTM's. LSTM's are one type of RNN that can solve the vanishing gradient problem while learning the temporal relationship between data.Initially, we wanted to implement a prediction type LSTM model that would take fixed length sequences generated from time series data and try to predict the data of subsequent timestamp. But our problem description facilitated the use of reconstruction type LSTM network. The idea was to incorporate an Autoencoder with an LSTM network. This is more commonly referred to as an encoder-decoder LSTM, where the encoder part takes in time series data as fixed length of sequences and the decoder part tries to reconstruct the same sequences, instead of predicting data from the timestamp following the sequence. The architecture of the network and working principle is presented hereafter.It needs to be mentioned that the following description regarding the encoder-decoder LSTM model has also been described in the paper associated with this work. The initial dataset of shape ($M_{sequences}$, Timesteps, $N_{features}$) was fed into the encoder of the LSTM block (one sequence as one sample) that has 64 LSTM Units. Each of these 64 units take in a timestep of sequence data that is a fixed length vector of size 10 and produces a short-term state and makes some changes to the existing long-term state. The output after the final timestep of the encoder is a vector of the same size as the number of LSTM units, in this case 64. Before passing this onto the decoder LSTM, the output needs to be resized so that it has the shape (Timesteps, $N_{features}$). To do that, the output after the final timestep of the encoder is repeated the same number of times as the number of timesteps in a sequence. The decoder LSTM has the same structure with 64 units and takes in a sequence of encoded data. Then, it produces an output of similar shape ($M_{sequences}$, Timesteps, $N_{features}$) by returning all previous outputs for each timestep as a sequence. Finally, a dense layer is applied to the output of LSTM units one timestep at a time. The weights associated with the dense layer are same for every timestep. The purpose of the dense layer is to transform the output of the decoder LSTM to have the same shape as the initial shape of the sequence (Timesteps, $N_{features}$). The output of the dense layers together now represent the reconstructed sequence. The model architecture is shown in Fig. 3.

TABLE I: Model Architecture of LSTM model

Layer	Input Shape	Output Shape	Number of Parameters
Encoder LSTM	(None,5,10)	(None,64)	19200
Dropout 1(20%)	(None.64)	(None,64)	0
Repeat Vector	(None,64)	(None,5,64)	0
Decoder LSTM	(None,5,64)	(None, 5,64)	33024
Dropout 2(20%)	(None,5,64)	(None,5,64)	0
TimeDistributedDense	(None,5,64)	(None,5,10)	650

After training the model with extracted sensor data, the encoder part of the model was used to compress time series data that could be used for visualisation purposes or as inputs in the form of feature descriptors to a classifier. Although in our implementation, since we wanted to declare abnormality as a probability between 0 to 1, instead of using the feature space to a binary classifier, we calculated the error in reconstruction between the decoded output sequences and original input sequences and used them as the reference for abnormal situation. The results are described in the next section.

C. DenseNet Autoencoder for Image Data

DenseNet [5] is the state of the art method for image classification. The success of DenseNet comes from its densely connected layers. Every dense blocks in the model is concatenated with the next one, so that features can be transferred from block to block, solving the problem of vanishing gradient. Because of unsupervised nature of the dataset, the idea is to extract important features from the frames using an DenseNet autoencoder structure. The model learns the regularity of video sequences and make predictions based on them. The encoder part is implemented by an DenseNet model, extracting feature space from fully connected layer and decoding the feature space using a series of transpose convolution and batch normalization layers. As the model is trained on only normal frames, abnormal frames will produce high reconstruction loss comparing to normal frames. Encoder part of the model is consisted of 4 dense blocks with transition layers. Dense blocks are again consisted of concatenated dense layers. Structure and number of dense layers in each dense block are determined based on DenseNet 121 model architecture. At the end of the dense blocks, flattening layer and fully connected layer of dimension 32 is connected to reduce dimension and extract the latent space from the frames. In the decoder section, a dropout layer of dropout rate of 40% followed by a series of 5 transpose convolution layer is connected. Transpose convolution layers have increasing filter size to match the size of input image and reconstructed image. The Dense connection between Dense blocks and Densnet Autoencoder architecture are shown in Fig. 4 and Fig. 5 respectively.

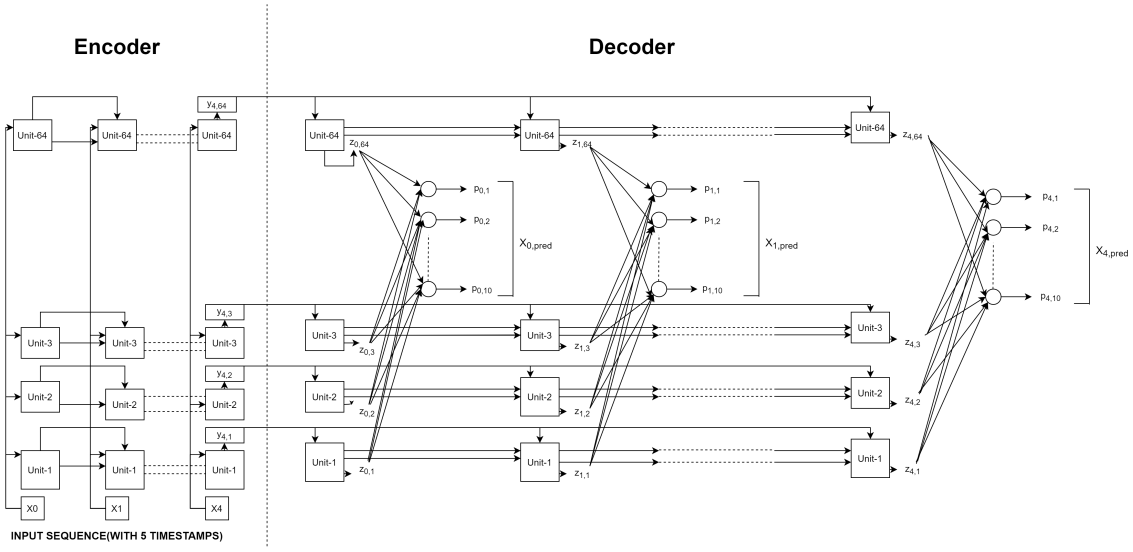


Fig. 3: Encoder-Decoder LSTM Architecture, Number of LSTM Units in each layer=64

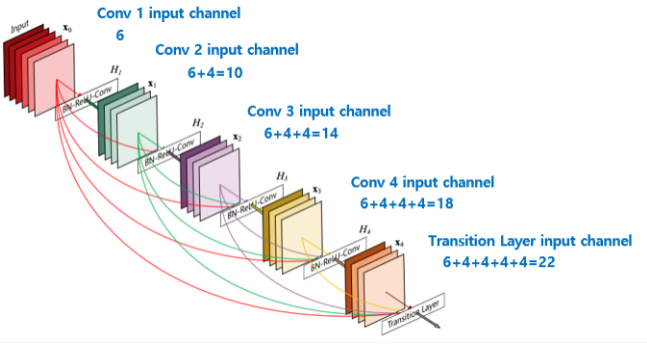


Fig. 4: Dense connections between dense blocks

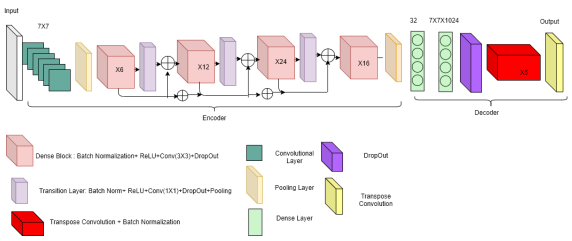


Fig. 5: DenseNet autoencoder model

D. Convolutional LSTM Model for Image Data

Recurrent autoencoder structure was proposed to extract the spatio temporal features of the input space which could be helpful for differentiating between normal and abnormal behavior. Among different recurrent network architecture Long Term Short Term Memory (LSTM) [6] was proposed in this model which is a popular model for analyzing sequential data.

For training purpose, total training frames were divided into

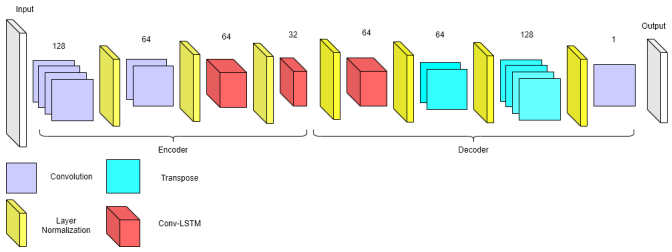


Fig. 6: Model architecture of conv-lstm autoencoder

temporal sequences, each consisting of 10 frames, using sliding window technique. Furthermore, frames were converted to grayscale and reshaped to 128X128 size to enhance computation efficiency along with scaling each pixel value between 0 to 1 by diving each pixel value with 255. To augment data in the temporal dimension, frames were concatenated with skipping strides. In case of making predictions, reconstruction loss among normal and abnormal frames were considered but this feature failed to provide any clear distinction between the two class. [7] [8]

E. Background Subtraction Using Google Deeplab V3+

To make a generalized anomaly detection model, it should be independent of the peripherals. The main objective is to detect abnormality of the surveillance system from the frames it captures. Therefore it is wise to focus on some distinct structure or object and discard the background objects. There are various traditional methods to remove background, but they are strictly parameter dependent and success rate is low. Hence a deep learning background detection model was adopted to discard unimportant features from the image. To accomplish

this task google deeplab v3+ [9] model using Xception [10] architecture was proposed, as shown in Fig. 7. The model uses

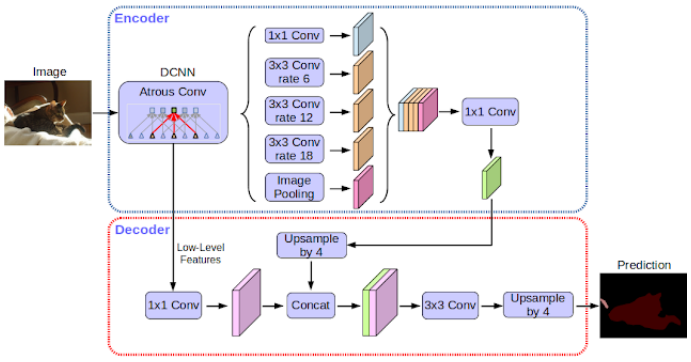


Fig. 7: Deeplab V3+ model using modified Xception architecture

pretrained modified Xception as feature extractor with atrous convolution with different sampling rate. In the decoder part, it has spatial pyramid pooling along with biliear upsampling on top of the Xception network. Notable modifications of the Xception model are:

1) All max pooling operations are replaced by depth wise separable convolution with striding
2) Extra batch normalization and ReLU activations are added after each 3X3 convolution.
3) Depth of the model is increased without changing the entry flow of the network.

Encoder section is based on output stride of 16. But instead of using bilinear upsampling with a factor of 16, the encoded features are first upsampled with a factor of 4 and concatenated with corresponding low level features from the encoder module having the same spatial dimensions. Before concatenation, 1X1 convolutions are applied on the low level features to reduce the number of channels and after that 3X3 convolutions are applied and the features are upsampled by a factor of 4. This method produces same size for both input and output images. [11]

*F. Optical Flow Based Convolutional Autoencoder for Images**

One obvious way to detect anomaly from video is to capture the orientation of the images. If the images highly rotated or vertically flipped, we can infer that the drone is jerking too much or it might be upside down. But modern Image models (like Convolutional Networks) are inherently invariant to translation, rotation and viewpoint; that makes them ill-suited for this problem. This concept is also echoed by our work using Dense-net. This problem is largely attributed to the translation invariance of modern Conv-Nets.

Intuitively, motion content of subsequent image frames should be more informative of system anomaly than the semantic content. One way to encode motion information from video data is to use optical flow. Optical flow captures the relative movement between objects and the camera. The intuition of using optical flow as a basis for system anomaly detection is verified by the visualization of the flow fields shown in Fig. 8. The Optical flow during the anomaly state of the drone is very noisy and occluded, reflecting the fact that the drone is moving at a very random pattern.

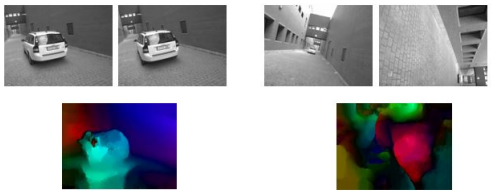


Fig. 8: Optical Flow visualization of normal and abnormal frames (left and right columns respectively). In the flow visualization, 'hue' channel represents the direction of the optical flow, 'value' channel represents the magnitude and the saturation channel was set to maximum for better visibility. These visualizations reflect the randomness in the motion between anomalous frames, essential for detecting anomalous state.

Considering the speed and accuracy tradeoff between popular optical flow estimation algorithms, we chose DualTVL1 as our preferred method (Table II). Although, Ferneback and Coarse2Fine algorithms are appealing choices for their low latency, but because of very low fps rating (2Fps) of our video feed, the estimated flow is very noisy and unreliable. Recent works on deep learning-based flow estimation have also been studied for our problem. But, the memory footprints and flow estimation latency of are too high to justify their real time use. For example, the state-of-the art LiteflowNet is an expensive 5.37 million parameter model that takes almost 3s to estimate one flow tensor form a pair of images. As an optimal choice, the DualTVL1 algorithm has an acceptable flow estimation time. It also preserves discontinuities in the flow field and offers an increased robustness against illumination changes, occlusions and noise.

The original images are of size 2048x1536. But calculating optical flow from these large size images are very costly. So, they were resized to 256x192 size keeping the aspect ratio. For resizing operation, we used inter-area interpolation that resizes the image using pixel area relation. Inter-area interpolation was preferred over linear or bilinear interpolation, as it gives moire-effect free results for image decimation. Although it might be tempting to reduce the image size even further for faster flow estimation, our experiments suggest that reducing the image size can heavily affect the anomaly prediction performance. Also, the images were converted to grayscale, since the color information is not essential for motion estimation.

Next for every pair of images, our dense optical flow method returns 2 flow vectors for each image pixel (one horizontal and one vertical). As a result, for 256x192 sized images it returns a tensor of shape 256x192x2. These flow tensors are the basis for detecting anomaly in video feed.

For an unsupervised representation learning of normal flow

TABLE II: Latency Comparisons of different optical flow methods

Flow Algorithm	Latency Per flow of Normal Images (ms)	Latency Per flow of Abnormal Images (ms)
Ferneback	25	120
Coarse2Fine	113	350
LiteFlowNet	3000	5000
DualTVL1	400	1000

vectors, a Convolutional- Autoencoder was trained using only the normal video data. The encoder compresses the input flow vectors by a factor of 256; as a result, the network is forced to learn a summarized knowledge representation of the normal flow vectors. To restrict the Conv-Autoencoder from learning an identity function, we implemented a denoising autoencoder with a noise ratio of 10%. Another approach we implemented was to incorporate KL divergence loss penalty on the hidden layer activations to introduce sparsity; the results were identical to that of denoising autoencoder. Our Conv-Autoencoder architecture is presented in Fig. 9 and Table III demonstrates the model structure.

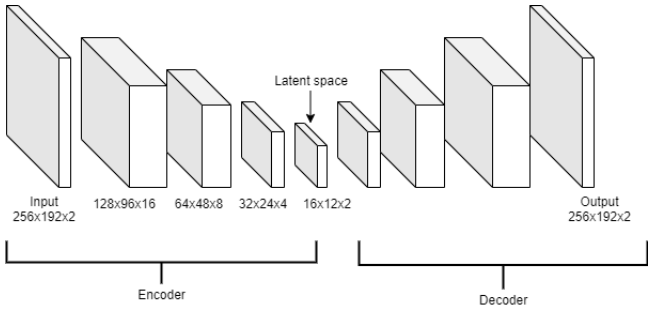


Fig. 9: Architecture of the Conv-Autoencoder used for optical flow.

The architecture follows some simple rules:

1) In the encoder part, every Conv layer compresses the channel dimension by a factor of 2. And, every Maxpool layer compresses the spatial dimension by a factor of 2.
2) In the decoder part, every conv layer expands the channel dimension by a factor of 2. And every Upsampling layer expands the spatial dimension by a factor of 2.
3) Every conv layer is followed by a batch-normalization layer and ReLU activation except the last one. This allows the Autoencoder to completely reconstruct the input.

For the upsampling layer, one popular choice is to use a transposed convolution. But transposed convolution is inherently prone to aliasing and causes "checkerboard artifact" [12]. Following the recommendation of Odena et al. we used nearest neighbor interpolation as our upsampling mechanism, followed by a conv layer. This upsampling method shows much better reconstruction performance.

TABLE III: Model Structure of Convolutional Autoencoder

	Layer Description	Output Shape
Input flow tensor		256x192x2
Encoder	Conv3, 16	256x192x16
	Maxpool	128x96x16
	Conv3, 8	128x96x8
	Maxpool	64x48x8
	Conv3, 4	64x48x4
	Maxpool	32x24x4
	Conv3, 2	32x24x2
	Maxpool	16x12x2
Decoder	Upsample	32x24x2
	Conv3, 4	32x24x4
	Upsample	64x48x4
	Conv3, 8	64x48x8
	Upsample	128x96x8
	Conv3, 16	128x96x16
	Upsample	256x192x16
	Conv3, 2	256x192x2

Similar to what we did with feedforward Autoencoder, we tried to use the latent variables of our Conv-Autoencoder as discriminative features. Isolation forest algorithm was used to map the latent features to an anomaly score, as shown in Fig. 10.

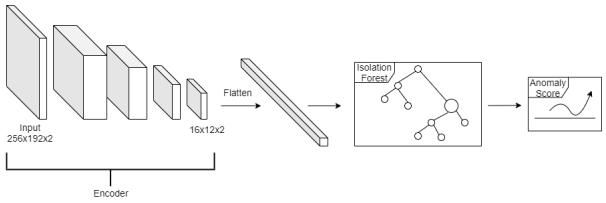


Fig. 10: Hybrid optical flow model flattens the latent space of the Conv-Autoencoder and uses these transformed vectors as features for Isolation forest.

G. GAN-based Anomaly Detection Tactic

Long Short Term-Recurrent Neural Networks (LSTM-RNN) had been shown to be capable of learning complex time series by taking the information in backward (or even forward) time steps with memorising cells. For handling the time-series data, both the generator (G) and discriminator (D) of GAN are substituted by LSTM-RNN.

Specifically, the generator G,which is an LSTM-RNN model, defines a probability distribution for the generated samples, from the latent space. The discriminator, which is another LSTM-RNN model, is then trained to minimise the average negative cross entropy between its predictions and sequence

labels (e.g., train D to recognize as many training samples as real as possible, and recognize as many generated samples as false as possible). Simultaneously, the generator is trained to confuse the discriminator in order that the discriminator would recognize as many generated samples as real as possible. As a newly developed research interest to detect anomaly using GAN, both the generator and discriminator part is trained to highlight normal variability, helpful for detecting anomalies. The anomaly detection for time series data consists of the following two parts.

1) **Anomaly Detection with Discrimination:** Theoretically the discriminator model after sufficient number of adversarial training is a tool for detecting anomaly as it can differentiate fake samples from real with high sensitivity.
2) **Anomaly Detection with Residuals:** The trained generator G, which is capable of generating realistic samples, is actually a mapping from the latent space to real data space and can be viewed as an inexplicit system model that reflects the normal data's distribution. Due to the smooth transitions of latent space, the generator outputs similar samples if the inputs in the latent space are close. Indeed, residuals between these latent space could be utilized for identifying anomalies in testing data.
   Then the samples from the latent space could be updated with the gradients obtained from the error function. If after enough iteration rounds the error is small enough, the samples are recorded as the corresponding mapping in the latent space for the testing samples. An anomaly is detected if the cross entropy error H for the anomaly score is higher than a certain value [13]

Many GAN models suffer from the following major problems which led us to believe that autoencoders are the better suit for this problem:

1) **Mode Collapse:** Usually the generator model fails to produce a variety of samples which leads to mode collapse problem commonly encountered in training GAN.
2) **Diminished Gradient:** It's a common thing that while training GAN,the discriminator model becomes too successful.Eventually, the generator gradient tends to vanish and thereby learns nothing.
3) **Non-convergence:** Parameter selection is quite crucial in training GAN.The inappropriate selection of parameters leads to destabilize the model and consequently the model never converges.
4) **Sensitivity:** The GAN model is largely sensitive to hyperparameter selection.

H. Deep Clustering Approach For Anomaly Detection

Clustering is one of the fundamental unsupervised method of knowledge discovery. The goal is to group similar data points together without supervision or prior knowledge of nature of the clusters. Various aspects of clustering such as distance metrics, feature selection, grouping methods etc. have been extensively studied. [14]

Deep Embedded Clustering (DEC), is a method that simultaneously learns feature representations and cluster assignments using deep neural networks. DEC learns a mapping from the data space to a lower-dimensional feature space in which it iteratively optimizes a clustering objective. [15]

Optimizing DEC is challenging. We wanted to simultaneously solve for cluster assignment and the underlying feature representation. However, unlike in supervised learning, we cannot train our deep network with labeled data. Instead we have to iteratively refine clusters with an auxiliary target distribution derived from the current soft cluster assignment. This process increases the training difficulty in a much greater way. Hence we had to discard this method.

IV. Criteria for Assessing Solutions

In the previous section, we discussed at length what the motivation was behind our proposed methods. In this section, we move on to describe the results obtained by those methods and infer why one proved to better than the other. We also introduce a scoring function that we have used to provide abnormality score on given data.

A. Feedforward Autoencoder for IMU Sensor Data

We trained our feedforward autoencoder end to end with mean squared error using only normal sensor data from the last 5 experiments. The remaining 1 normal experiment data and all abnormal experiment data were used for testing. Fig. 11 shows the reconstruction error of the autoencoder output:

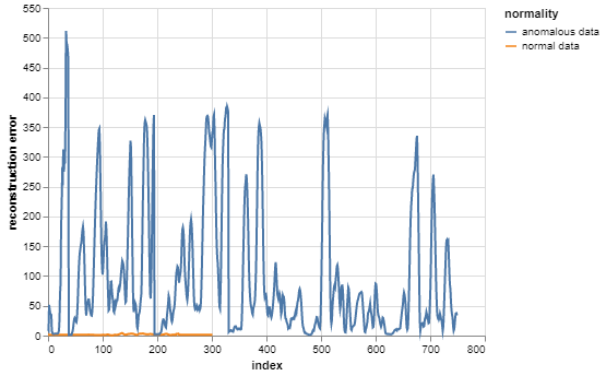


Fig. 11: We see that the autoencoder produces low reconstruction error for normal data and high reconstruction error for anomalous samples. The momentary low reconstruction loss for anomalous data means that the drone was behaving normally for that brief period.

To test our hypothesis about the latent space learning important feature descriptors, we discard the decoder part. Then only the encoder stage can be used as a fixed feature extractor for any One-class classification algorithm. Our choice of one-class classification algorithm was Isolation Forest [16]. The Isolation Forest 'isolates' observations by randomly selecting a feature and then selecting a split value between the maximum

and minimum values of the selected feature. Since recursive partitioning can be represented by a tree structure, the number of splits required to isolate a sample is equivalent to the path length from the root node to the terminating node. Random partitioning produces noticeable shorter paths for anomalies. Fig. 12 shows the anomaly score of isolation forest.

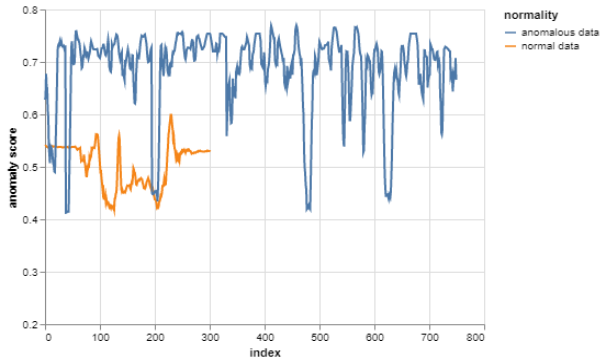


Fig. 12: Anomaly prediction of the hybrid model (between 0 and 1). Higher score means more anomalous.

These experiments indicate that using hybrid models like this could be a good starting point for our solution. But if we look closely, the anomaly score for normal instances are heavily concentrated around 0.5. This is clearly ambiguous and gives us no useful information about the drone's regularity. We hypothesize that, this is because these models completely ignore the continuous nature of the drone motion. They treat sensor measures coming from drone IMU as temporally independent from each other.

*B. LSTM Autoencoder for IMU Sensor Data**

Although the result obtained from the implementation of Linear Autoencoder was proof that autoencoders can be useful in this domain, the feature space learned by the encoder was not able to capture the temporal dependencies between the timestamps of sensor data. So, a better direction is to use a network such as the encoder-decoder LSTM that inherently captures the sequential nature of the sensor data. At train time, our model tries to reconstruct the same sequences that are fed to it. In doing so, it generated some losses per feature per sequence. The model was trained such that the reconstruction losses for the sequences were minimized. Now since the model was trained only on normal time-series data, if it were to reconstruct sequences from abnormal time-series data, it should yield a higher reconstruction error. This again is analogous to the motivation behind Linear Autoencoder. The difference is here is that the LSTM model tries to learn the temporal dependencies between the sensor measurements since it is reconstructing sequences generated from fixed lengths of timestamps of sensor data, so the abnormality score on a timestamp encompasses the effect of the sensor data from previous timestamps.

Fig. 13 shows the comparison between reconstruction losses for angular velocity in y-direction between normal and abnormal sequences for test data. The model was trained on the final 5 normal experiments. The remaining normal dataset was used for testing and all abnormal datasets were concatenated for visualization purposes.

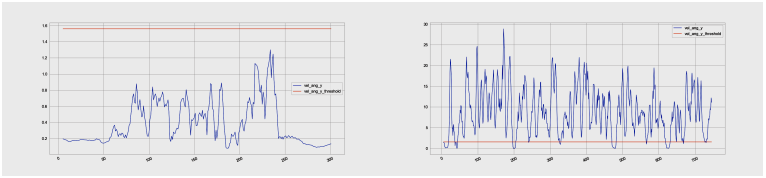


Fig. 13: Angular Velocity losses in the y direction across sequences for normal bag files(left) and abnormal bag files(right). The horizontal line is the maximum loss for angular velocity learned from the normal bag files used for training

The horizontal line is the maximum loss for the corresponding feature learned from the normal bag files used for training. It can be thought of as a threshold on abnormality. If the loss exceeds this threshold, we know that sequence is abnormal. It can be seen that the reconstruction losses for normal sequences fall below the threshold and the losses for abnormal sequences exceed the threshold in some sequences. Therefore we came to a conclusion that these losses per feature can be used as the abnormality measures. The higher the loss per feature, the more it will contribute to that sequence being abnormal. It also enables our model to determine the reason behind the abnormality because the feature that has the higher reconstruction loss should ideally have the higher contribution.

C. DenseNet Autoencoder for Image Data

DenseNet autoencoder is successful to differentiate normal and abnormal frame in some extent. From the reconstructed images, it is clear from Fig. 15 that normal images are reconstructed successfully whereas the model fails to reconstruct abnormal frames completely. But by observing the reconstruction loss curve in Fig. 14, it is very difficult to define any threshold that can differentiate normal and abnormal frames. The main reason behind that is the extraction of peripheral features by the model. But the model is supposed to learn normality of the heterogeneous system based on the behavior of objects from the frames, which it fails completely.

D. ConvLSTM Autoencoder for Image Data

As can be seen from the loss curve shown in Fig. 16, there is no real distinction between the two distributions and no effective threshold can be calculated on which abnormality score can be predicted. The main reason behind the failure is scarcity of data. Proposed model had around 1.9 million trainable parameters whereas available frames for training purposes were only 420. Because the model makes predictions on sequence of frames, producing abnormality score on every frame based on this model is not effective procedure.

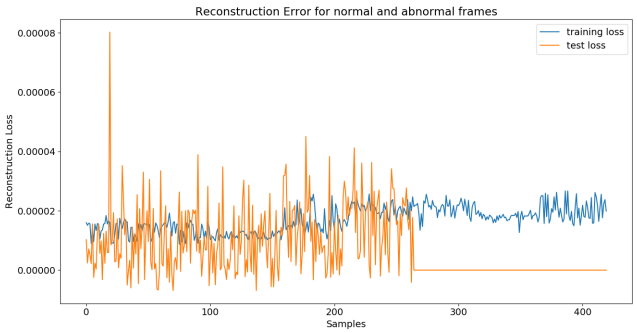


Fig. 14: Reconstruction Loss for normal and abnormal frames.

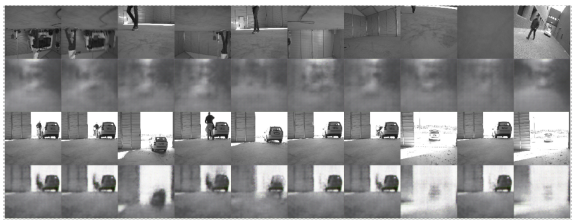


Fig. 15: Input (abnormal frame on 1st row and normal frame on 3rd row) and Reconstructed Image (abnormal reconstructed frame on 2nd row and normal reconstructed frame on 4th row).

E. Background Subtraction

Segmented Images:

After background subtraction, the abnormal images look as shown in Fig. 17(b) and normal images look as shown in Fig. 18(b). The idea was to segment important features from the frames and feed the segmented images to autoencoder so that prediction is independent from involvement from environment. Originally the model was designed for semantic segmentation, detecting objects in an image and labeling it. Therefore the model is trained to segment some specific objects, for which the model works best. Hence this method is not generalized. Another major drawback of this method is the case where no foreground can be detected and the whole segmented image remains dark. Again the model fails to remove background in blurry images. Considering these points, background subtraction method was discarded.

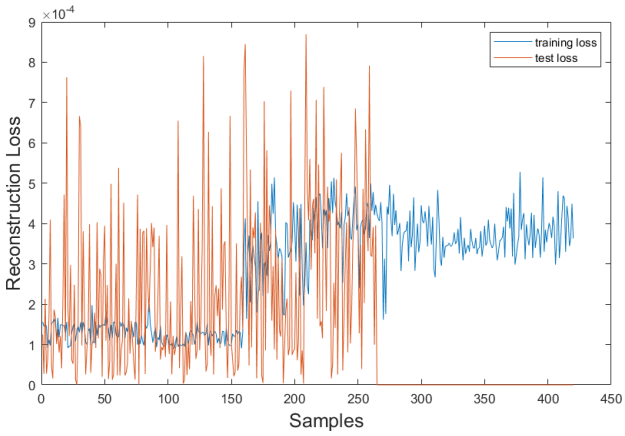
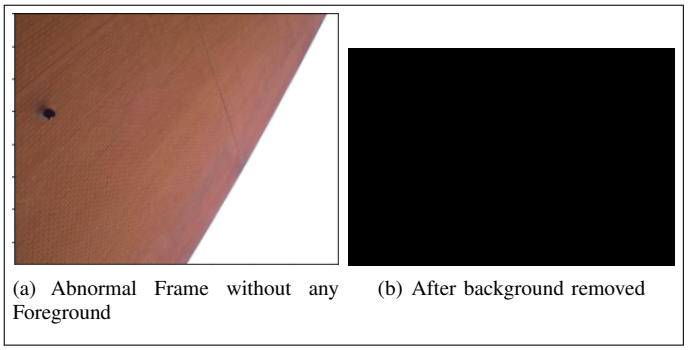
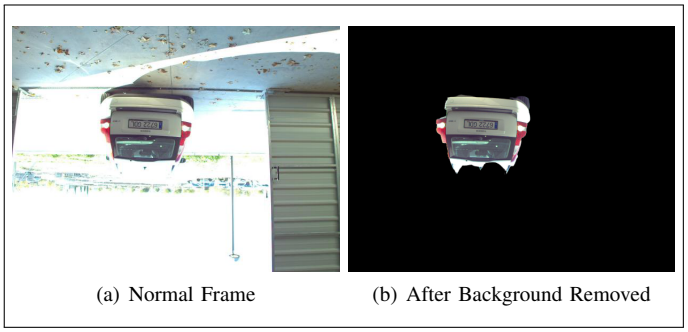


Fig. 16: Reconstruction Loss for normal and abnormal frames.



(a) Abnormal Frame without any Foreground (b) After background removed

Fig. 17: Background Subtraction for Abnormal Image



(a) Normal Frame (b) After Background Removed

Fig. 18: Background Subtraction for Normal Image

*F. Optical Flow Based Convolutional Autoencoder for Image data**

Since the optical flow inherently captures the temporal dependency of consecutive image frames, anomaly score of the hybrid optical flow model is a good indicator of the drone regularity. But similar to our hybrid sensor model, we can see from Fig. 20 all the anomaly scores are heavily concentrated around 0.5. Although the above experiments show that the latent space of the Conv-Autoencoder can be a good indicator of the anomalous state, we find that using the reconstruction loss of the decoder output is much more reliable and informative.

The reconstruction loss curve, shown in Fig. 20 is indicative that the model learns to successfully distinguish between normal and abnormal images, unlike the methods described previously where there was no clear distinction between both. These results convinced us to use the Optical-Flow based approach in our final submission.

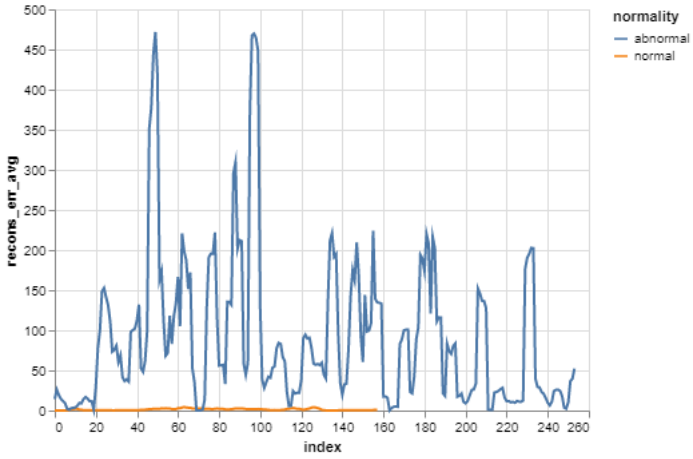


Fig. 19: Anomaly score of our hybrid optical flow model (between 0 and 1). Higher means more anomalous. Score from all 5 abnormal datasets are concatenated for better visualization.

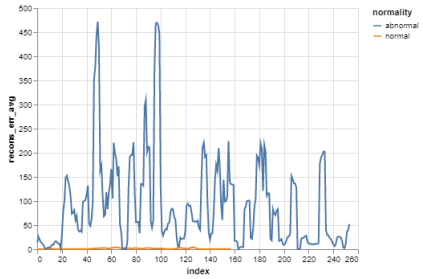


Fig. 20: The convolutional-autoencoder produces low reconstruction error for normal data and high reconstruction error for anomalous samples. The momentary low reconstruction loss for anomalous data means that the drone was behaving normally for that brief period. Scores from all 5 abnormal datasets are concatenated for better visualization.

G. Abnormality Scoring

Now that we have established we will be using the reconstruction loss between the decoded outputs and original inputs as our measures for the degree of abnormality for both our image and sensor model, we now need to define a scoring function that would map the reconstruction losses to a range of 0 and 1 to give a continuous abnormality score. To do so, we first used regenerated the training sequences and plotted a histogram of the losses, shown in Fig. 21. Naturally, it appeared that the frequency of low values of losses was considerably higher for the training data.
Fig. 21 reveals that, density of reconstruction loss can be assumed as a Gaussian distribution around a mean (μ) and a standard deviation (σ) for every specific feature. The intuition is, if abnormal-data is passed through the model reconstruction loss will not follow the normal distribution range.

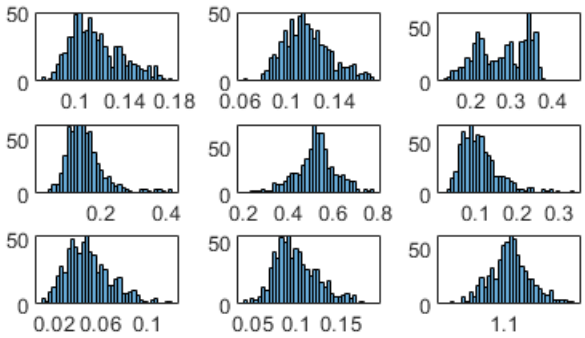
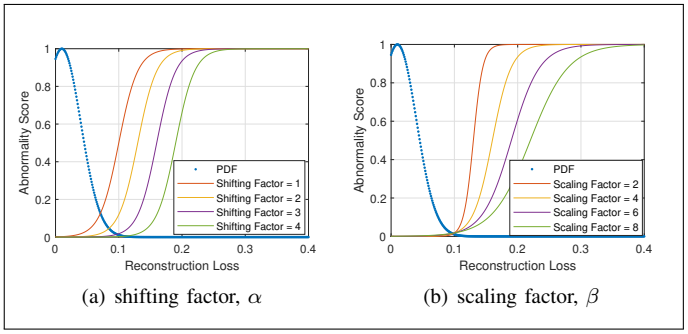


Fig. 21: Density of reconstruction loss for different features by LSTM Autoencoder.

To get abnormality score, based on re-construction loss we propose a mathematical model with two parameters: 1) shift factor, α and 2) scale factor, β. Fig 22 demonstrates the result of α and β for our proposed mathematical model of abnormality scoring.



(a) shifting factor, α (b) scaling factor, β

Fig. 22: Effect of shifting and scaling factor

To get abnormality score of a feature with normal reconstruction loss range μ and σ, first we calculate new reconstruction loss from original reconstruction loss:

$$Newloss = \frac{8}{(\beta * \sigma)} * (loss - \mu - \alpha * \sigma) - 4$$

New loss ranges from $-\infty$ to $+\infty$, we apply logistic function to get abnormality score from new reconstruction loss.

$$Abnormalityscore = \frac{1}{1 - e^{-Newloss}}$$

Thus we get abnormality score between 0 to 1 and finally we take additive mean of all individual feature's abnormality score to get final abnormality score for sensor model. As we used different models for image and sensor data, α and β parameter were also different. For sensor model we used α = 3, β =6 and for image model we used α = 1, β = 10.

V. Research Methodology

A. Experimental Setup

- Our LSTM Autoencoder model on IMU sensor data was implemented using the Tensorflow framework by Google. [17]
- For our Optical Flow based Conv-Autoencoder model that provides results on image data, we used PyTorch, an open source machine learning library based on the Torch framework. [18]
- The user interface for this work was developed using PyQt5, a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in.
- The optical flow between a pair of images was calculated using functions from OpenCV library for Python. [19]
- Our implementation was developed and tested on windows platform and MATLAB is used to extract necessary IMU sensor data and images.
- To fasten training process we used NVIDIA GPU (RTX 2070 Super), but this was not mandatory.
- To extract data from different topics in the rosbag file, we initially used ROS software. We used rqt_bag module of ROS to visualize the data. Through this module we observed the data in every timestamp with the corresponding images. The images were part of a 2 frames per second video feed. Afterwards for the purpose of training, we extracted time-series data in csv files. We executed the extraction with python scripts and extracted the sensor data and images separately. However, to keep a harmony with the GUI that we have designed we finally decided to extract the data with the help of MATLAB. For that we have used the rosbag functionality of MATLAB.

B. Flow of Code

In this part, we provide a flow diagram of our code for this work. This will help the reader visualize how and when several functions are called inside our code and what they do.

1) Training Phase: Fig. 23 is a walkthrough of the training process. First we select the normal datasets to train the model with and those bag files are then extracted by manually running a MATLAB code. After extraction, a new file that has the same name as that of the bag file is created inside the 'data' folder that contains the merged csv of sensor data, images and another csv file with the path information of images. The data folder remains in the same directory as the UI and contains information for each bag file.
Next, when we press on 'TRAIN' from the UI, a function from the script '$train\_sensor$' is run. It reads in the real time sensor data from sensor.csv and sends it to '$create\_dataset$' that generates sequences from it. For each bag file, the sequences are appended to the last. Then, the merged training dataset consisting of sequences from all bag files are feature scaled using another function, '$featureScale\_train$'. The scaling parameters are then saved for scaling data at test time. Now we have the dataset ready for training, before we train the model, we purposefully introduce noise to our data by using DataGenerator. Then our LSTM model is trained and saved. The mean and standard deviation of reconstruction losses are also stored for our scoring function. Next, a function from '$train\_flows$' is run and starts calculating optical flow between two consecutive frames for each bag file unless it has been previously calculated. Once we have the optical flows for each frame, our convolutional autoencoder model is then trained and saved. The mean and standard deviation of reconstruction losses are also stored for our scoring function.

2) Testing Phase: Fig. 24 demonstrates the testing. First we select from UI which bag file we want to use for testing. Then similar to the training process, a function from '$test\_sensor$' is called and sequences are generated using '$create\_dataset$' The data are then scaled using stored parameters from training inside the function '$featureScale\_test$'. Once the sequences are scaled, we use the saved model to reconstruct the sequences. Then we get the loss per feature per sequence which we send to our scoring function '$normal\_dist$'. The function outputs a probability score of abnormality on a given sequence. We take that score for the timestamp in the middle of the sequence. For an even length sequence, the score is for the left middle timestamp. Now, for image data, similar to training images, first optical flow is calculated between two consecutive frames. Then the Conv-autoencoder model is used to reconstruct the flow vectors which generate losses. We send the losses to our scoring function '$normal_dist$', but in this case we use a fixed scaling factor of 10 and a fixed shifting factor of 1. The function gives a probability score of abnormality on the timestamp of the first frame.

C. Synchronization of IMU and camera data

In our final solution, we decided to keep the results from both models separate. Initially, we wanted to fuse the results at decision level. But, although both data are aligned in time, the timestamps and sampling rate are different for each. Going from one image to the next in the video feed, we get multiple timestamps of sensor data in between. At the same time, this number of timestamps of sensor data in between two consecutive frames is not uniform. So, there is no straightforward way to tell the correspondence between both. We first tried to find the correlation between the two results. To match their length, we used the resample function from MATLAB. But the results were not convincing. It was obvious that the function tried to correlate the result from one sensor timestamp to the result from another image timestamp and in reality they are not related to each other.

Then we attempted to use nearest neighbour interpolation for image data. In this process, we go over all the sensor timestamps, copy the flow vectors that is nearest to that timestamp and assign those flow vectors to that sensor timestamp. This way, the same flow vector is repeated multiple times. But our Conv-autoencoder model could not

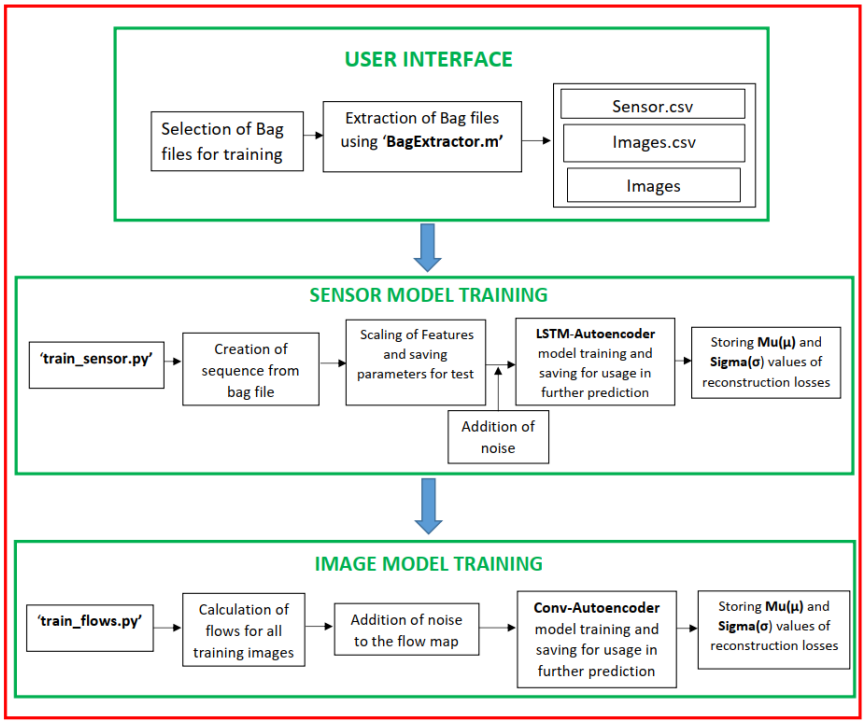


Fig. 23: A visual demonstration of training LSTM and Conv Autoencoder Mode

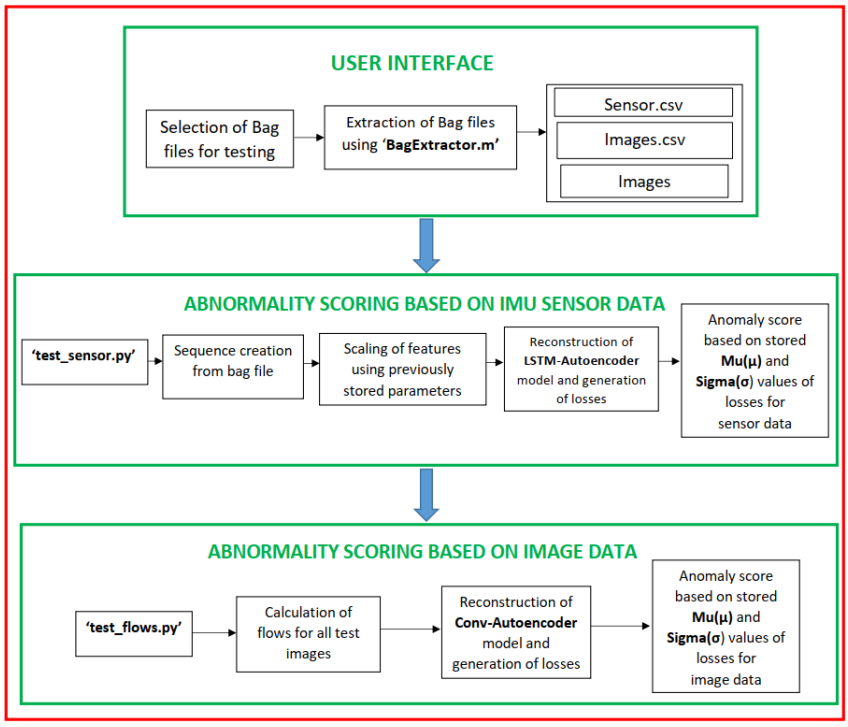


Fig. 24: A visual demonstration of testing phase of LSTM and Conv Autoencoder Mode

be trained properly using these flow vectors and provided unreliable results.

Then we decided to use the flows as they were and used our Conv-autoencoder model to give abnormality scores on timestamps of images. Similarly we applied nearest neighbour for those scores and aligned them with scores predicted by LSTM autoencoder model.

Although the results were better than the previous attempts, we believed that it was in our best interest to keep the results separate. The main objective of our problem is to reliably detect abnormalities in real time data. Before fusing, we were confident that our LSTM Autoencoder model was showing stable and reliable results on all the datasets. We were able to draw analogies between the abnormalities observed in the video feed and the prediction of our model. To reiterate, our Conv Autoencoder model takes in flow vectors between two consecutive frames and is trained on that. Now, if we interpolate the results obtained from this to nearest sensor timestamps, they fail to convey any meaning since the whole idea is to take two frames as fixed inputs and see if any abnormality appeared. The inference that abnormality remained the same even at sensor timestamps cannot be made here because the model is not trained in that manner. This is why we kept both results as they were. We can now use the LSTM autoencoder model to give stable predictions at all timestamps. Our Conv-autoencoder model will give predictions at timestamps in between and can be used as a sanity check for whether our LSTM autoencoder model is working properly.

VI. Analysis and Interpretation

A. Dataset

Originally, the datasets released by the organizers of "IEEE SP CUP 2020: Unsupervised abnormality detection by using intelligent and heterogeneous autonomous systems" were named according to their release dates. We have named them in a short way so that we can refer to them easily in this paper. The nomenclature is shown in Tab. IV.

TABLE IV: Data Nomenclature

Experiments	Bag files	Nomenclature
Normal	First dataset released on 22 Nov 2019	N_0
	2020-01-17-11-32-12.bag	N_1
	2020-01-17-11-32-49.bag	N_2
	2020-01-17-11-33-26.bag	N_3
	2020-01-17-11-34-08.bag	N_4
	2020-01-17-11-34-08.bag	N_5
Abnormal	First dataset released on 2 Dec 2019	A_0
	2020-01-17-11-35-27.bag	A_1
	2020-01-17-11-36-03.bag	A_2
	2020-01-17-11-36-43.bag	A_3
	2020-01-17-11-37-25.bag	A_4
	2020-01-17-11-38-07.bag	A_5

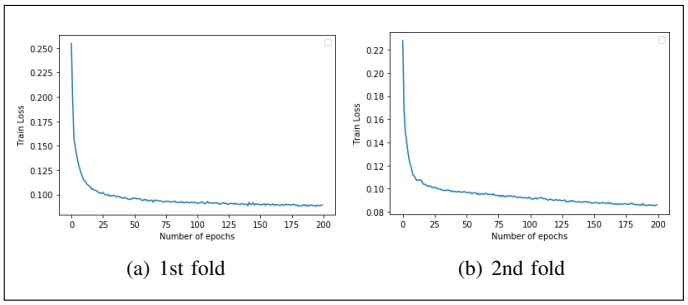
TABLE V: Folding of the dataset

Fold	Train Set	Test Set
1st	N_0	N_1-N_5, A_0-A_5
2nd	N_1-N_5	N_0, A_0-A_5

B. Train

The training was done two fold which is shown in Tab. V.

To train the sensor data with our LSTM Autoencoder, we used Adam optimizer with a learning rate of 0.001 and an MAE loss function and a batch size of 32.As working with small dataset can lead to overfitting, for training robust model and reducing generalization error we added small Gaussian noise to input of the sensor model which actually acts as data augmentation. The loss vs epoch curves are shown in Fig. 25



(a) 1st fold (b) 2nd fold

Fig. 25: Loss vs Number of epochs during training for LSTM Autoencoder

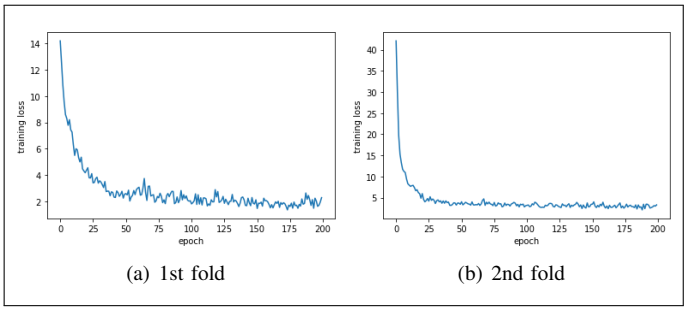
As working with small dataset can lead to overfitting, we added small Gaussian noise to input of the LSTM Autoencoder model for training robust model and reducing generalization error which basically works as data augmentation.

For the optical flow obtained from the image data, a Conv-Autoencoder was trained using SGD optimizer with momentum 0.9, weight decay 1e-5 and a learning rate of 0.001. The loss function used is Mean Squared Error (MSE). Batch normalization is added after every convolution layer but with no dropout. Our model was trained with a batch size of 32 on a single GPU for 200 epochs. Notably, 10% input noise during training acts as a form of regularization for our autoencoder model.The loss vs epoch curves are shown in Fig. 26

C. Test

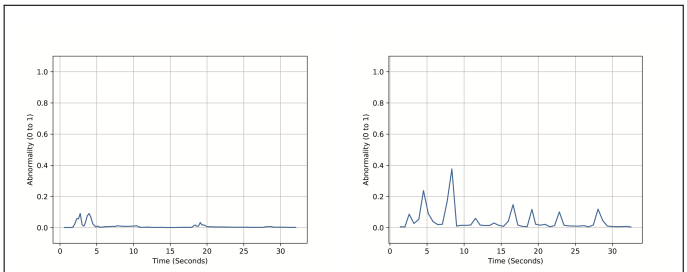
1) 1st fold:

a) Dataset with normal experiments: The results on the normal experiment dataset, N1 are shown in Fig. 27. The figures show flat results after testing on dataset of normal experiments indicating very low abnormality score.



(a) 1st fold (b) 2nd fold

Fig. 26: Loss vs Number of epochs during training for Conv-Autoencoder

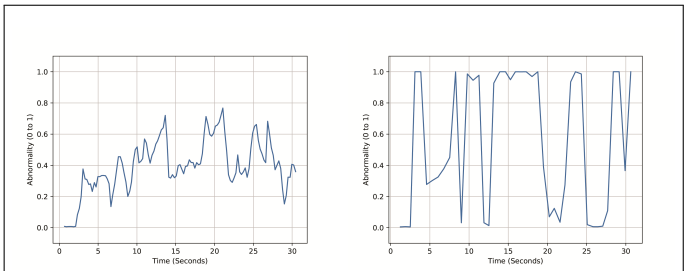


(a) Abnormality score of LSTM Autoencoder on sensor data (b) Abnormality score of Conv-Autoencoder on image data

Fig. 27: Abnormality score on N_1 dataset

b) Dataset with abnormal experiments: The abnormality score on the abnormal experiment dataset, A1 are shown in Fig 28

The models predict high abnormality scores for this dataset as compared to N1. Source of abnormality can be explained from category-wise abnormality which will be discussed later in section VI-D

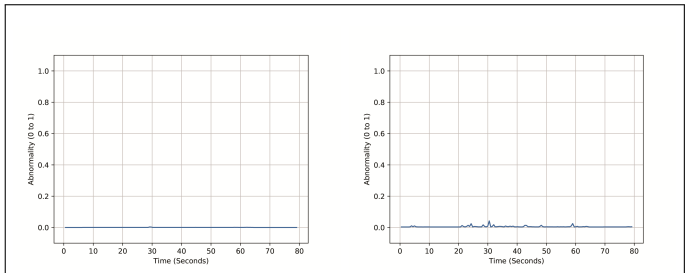


(a) Abnormality score of LSTM Autoencoder on sensor data (b) Test results of Conv-Autoencoder on image data

Fig. 28: Abnormality score on A_1 dataset

2) 2nd fold:

a) Dataset with normal experiments: The abnormality score on the normal experiment dataset, N_0 are shown in Fig. 29

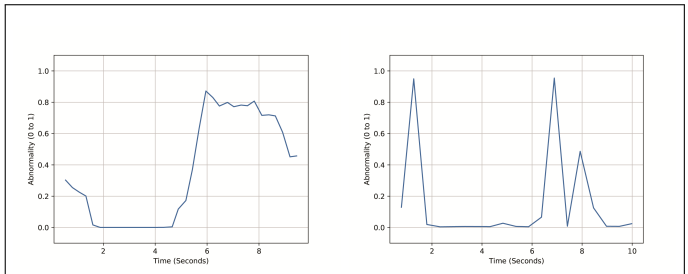


(a) Abnormality score of LSTM Autoencoder on sensor data (b) Abnormality score of Conv-Autoencoder on image data

Fig. 29: Abnormality score on N_0 dataset

The results show that the ensor and image results corresponded well with each other and showed no visible sign of abnormality.

b) Dataset with abnormal experiments: The abnormality score on abnormal dataset, A_0 is shown in Fig. 30



(a) Abnormality score of LSTM Autoencoder on sensor data (b) Test results of Conv-Autoencoder on image data

Fig. 30: Abnormality score on A_0 dataset

The results of the LSTM Autoencoder model shows that there is a change in the system at around the 6th second mark. Now playing the original video sequence, shown in Fig. 31 was a reconfirmation since we could see that the drone suddenly flipped with respect to the vertical axis at around that particular point in time indicating an abnormality. The Conv-Autoencoder model can also detect this abnormality due to the sudden abnormal jerk faced by the system and is evident from the second plot. The more fluctuating result of the Conv-Autoencoder model can help us determine whether the system has undergone severe change in a small duration. The results of the LSTM Autoencoder model can back it up by providing more continuous and stable results over that same duration.

D. Category-wise Abnormality detection

Our LSTM Autoencoder model is capable of determining the reason behind the abnormality by giving separate abnormality scores for separate features, such as orientation, angular

(a) Stable system (b) System undergoing sudden change (c) Flipped system

Fig. 31: Video Sequence Feed around the point of abnormality sensed by model

Velocity and linear Acceleration. These scores represent the contribution towards the final abnormality score for a certain timestamp. The category-wise abnormality results on abnormal dataset A_1 are shown in Fig. 32.

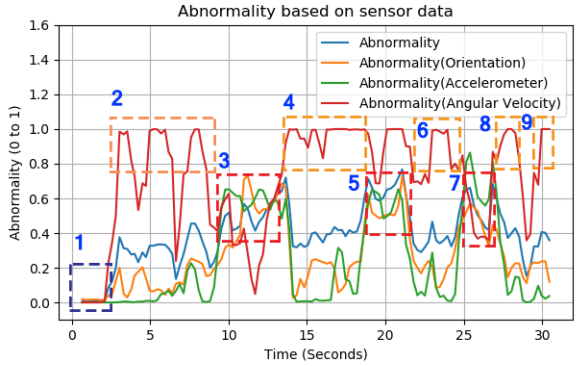


Fig. 32: Category-wise abnormality for A_1

We can divide the portions in some categories:

a) Normal: This is indicated in blue rectangle (labelled 1) in the figure. No abnormality from any source is visible in this portion.

b) Abnormal due to high angular velocity: This is indicated by dashed orange rectangle (labelled 2,4,6,8 and 9). It is obvious that angular velocity is abnormal in these portions. Portion 2, 4 and 6 are of most significance. Prior to 2, 4 and 6 abnormality due to orientation and accelerometer was low. But after high abnormal angular speed, we can see portion 3, 5 and 7 where orientation and accelerometer is found most abnormal in the total time-span. On the other hand in portion 8 (high angular velocity) the abnormality related to orientation is reduced.

c) Abnormality due to orientation and accelerometer: These two sources are found inter-related and are most obvious in portion 3, 5 and 7 after high angular speed. These portions are indicated by red dashed rectangle.

E. Summary of results

From the table shown in Tab. VI, mean and standard deviation of abnormality scores for normal datasets are low as compared to mean and standard deviation of abnormality scores in the abnormal datasets. The higher standard deviation of abnormality scores as predicted by image model can be attributed to the fact that it compares two consecutive frames and shows a large spike when there is significant change going from one frame to the next.

TABLE VI: Summary of results

Experiments	Bag files	Sensor mean	Sensor std	Image mean	Image std
Normal	N_0	0.001	0.008	0.005	0.002
	N_1	0.0005	0.0007	0.019	0.047
	N_2	0.0004	0.0001	0.009	0.005
	N_3	0.001	0.004	0.028	0.131
	N_4	0.0004	0.005	0.007	0.004
	N_5	0.0004	0.004	0.008	0.006
Abnormal	A_0	0.511	0.412	0.494	0.296
	A_1	0.426	0.231	0.381	0.413
	A_2	0.496	0.241	0.469	0.463
	A_3	0.556	0.258	0.547	0.445
	A_4	0.656	0.247	0.567	0.451
	A_5	0.711	0.202	0.778	0.369

VII. Conclusions and Recommendations

In this work, we systematically study both deep and hybrid networks (deep model paired with classical one-class classification methods) for anomaly detection in autonomous systems. We identify that end to end deep networks provide a much more reliable anomaly score. To address the heterogeneous nature of the data coming from the autonomous systems, we propose two different networks that handle IMU sensor values and the Camera feed separately. An LSTM-Autoencoder is proposed to handle the sequential nature of the IMU sensory values. We also identify that regardless of the semantic content, motion information from consecutive video frames are much more informative of the system regularity and we propose a Convolutional-Autoencoder operating on Optical-Flow maps as our 2nd model. A unique solution is also reported to transform the unbounded autoencoder loss to a normalized anomaly score. Extensive experimental analysis demonstrates the efficacy of our design. Measuring dense optical flow from video frames is a computationally expensive task. Current CPU implementations are not fast enough for real-time flow estimation. Our next attempt would be a vectorized implementation of dense flow methods that can take advantage of modern Graphics Processors to accelerate the computation.

References

[1] https://en.wikipedia.org/wiki/Anomaly_detection

[2] Unsupervised anomaly detection in unmanned aerial vehicles,Author:Samir Khan, Chun Fui Liew , Takehisa Yairi , Richard McWilliam

[3] Control Strategies for Heterogeneous, Autonomous Robot Swarms.Author:Stefan Thamke1, Markus Ax1, Lars Kuhnert1

[4] An Overview of Embedded Sensor Networks, ISI TR-2004-594, John Heidemann and Ramesh Govindan

[5] Densely Connected Convolutional Networks Gao Huang , Zhuang Liu , Laurens van der Maaten , Kilian Q. Weinberger

[6] LONG SHORT-TERM MEMORY Neural Computation, Sepp Hochreiter

[7] https://towardsdatascience.com/prototyping-an-anomaly-detection-system-for-videos-step-by-step-using-lstm-convolutional-4e06b7dcdd29

[8] Deep convolutional recurrent autoencoders for learning low-dimensional feature dynamics of fluid systems, Francisco J. Gonzalez, Maciej Balajewicz

[9] DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille

[10] Xception: Deep Learning with Depthwise Separable Convolutions François Chollet

[11] https://www.analyticsvidhya.com/blog/2019/02/tutorial-semantic-segmentation-google-deeplab/

[12] Deconvolution and Checkerboard Artifacts", Distill, 2016. http://doi.org/10.23915/distill

[13] Anomaly Detection with Generative Adversarial Networks for Multivariate Time Series. Author: Dan Li, Dacheng Chen, Jonathan Goh, and See-Kiong Ng

[14] https://deepnotes.io/deep-clustering

[15] Unsupervised Deep Embedding for Clustering Analysis.Author: Junyuan Xie ,Ross Girshick, Ali Farhadi ALI

[16] Liu, Fei Tony and Ting, Kai Ming and Zhou, Zhi-Hua, Isolation Forest, Proceedings of the 2008 Eighth IEEE International Conference on Data Mining)

[17] [Martin Abadi, Ashish Agarwal, Paul Barham, et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, Software available from tensorflow.org.] running behind the wrapper library Keras [Francois Chollet et al.,"Keras," 2015.

[18] H. Wallach and H. Larochelle and A. Beygelzimer and F. de-Buc and E. Fox and R. Garnett, "PyTorch: An Imperative Style, HighPerformance Deep Learning Library", Advances in Neural Information Processing Systems, 2019]

[19] G. Bradski, "The opencv library," Dr. Dobb's Journal of Software Tools, 2000