Deep Learning for Anomaly Detection in Rate Gyro Assembly Data

Edward Chau

Marshall Space Flight Center

August 2019

Reviewed By:

Brandon Steele (MSFC-ES52)

Wade Chatam (MSFC-ES52)

Deep Learning for Anomaly Detection in Rate Gyro Assembly Data

Edward Chau

NASA Marshall Space Flight Center (Huntsville, AL) | ES52 Software Development Intern |
edwardchau@berkeley.edu

Abstract: The prominence of neural networks in areas for automation has led to machine learning (ML) applications in data-rich industries. Gyroscopic sensors generate an abundance of data generated from simulations that can be used to train an artificial neural model for anomaly detection. This paper illustrates the use of deep learning networks including Long Short-Term Memory (LSTM), autoencoders, and LSTM-autoencoders to detect faults in time series data. The experiment involves testing on injected faults in nominal Rate Gyro Assembly (RGA) data provided by the Guidance Navigation and Mission Analysis Team (EV42) and calculating the reconstruction error in order to locate the exact occurrences of anomalies. The LSTM-Autoencoder was the most successful neural architecture in analyzing faults and producing the least amount of false positives. A 1D Convolutional Neural Network (CNN) was also explored for anomaly classification.

Keywords: Machine Learning, LSTM, Autoencoder, RGA, Anomaly Detection

I. Introduction

The Guidance Navigation and Mission Analysis team utilizes Sensor Data Quality (SDQ), their software for fault detection checks, on RGA[1] data. A decision manager then examines error counters generated from SDQ checks involving Data Quality Indicators (DQI), time tags, redundancy, and box comparisons. DQI evaluates the health and status of RGA through internal hardware and software tests. Time tag checks ensures timesteps are 50 Hz. Redundancy checks monitor rate differences between each channel of an RGA sensor. Box comparisons are similar to redundancy checks but differences between sensor boxes are monitored. In collaboration with EV42, the ML Technical Excellence (TE) team (ES52) is developing a deep learning approach to automate fault detection in RGA data.

RGA fault detection is one focus; the ML TE team is exploring several other routes for machine learning applications in the Space Launch System (SLS). The goal of this TE is to develop experience in machine learning at Marshall Space Flight Center and cooperate with other sectors to minimize the amount of human analysis. Machine learning provides a general robust framework that can find complex patterns humans may overlook and reduces the amount of test cases for sensor analysis. This paper covers the use of LSTM's, Autoencoders, LSTM-Autoencoders, and Convolutional Neural Networks (CNN) for detection and classification of faults in FWD RGA y-axis data.

[1] RGA is a 3-channel gyroscopic sensor part of the Core Stage Avionics existing in the FWD Skirt and AFT Skirt to sense SLS vehicular motion

II. Neural Network Architectures

A. Recurrent Neural Networks

Recurrent neural networks (RNN) are optimal for detecting features in sequential data because they memorize inputs in order to make predictions on subsequent events. Unlike other neural networks, RNN's have loops that take in previous events as inputs, giving them a temporal dimension [7]. With a deficit of off-nominal RGA sensor data, RNN's are able to utilize unsupervised learning with unlabeled data and reconstruct nominal RGA curves.

B. LSTM

Although RNN's are able to take into account past information to make a prediction, they are limited in handling long-term dependencies [9]. As the gap between information increases, RNN's are unable to connect the outputs together. LSTM networks overcome the long-term dependency issue with a chain of repeating modules. Gates in an LSTM model regulates cell states and determine which information passes [6]. With this property, LSTM's are useful for learning long-term information in sequential data.

C. Autoencoder

Autoencoders, used for dimensionality and noise reduction, attempt to mimic inputs during training. The encoding layers maps the input into code while the decoding layers translates that code into a reconstruction. This approach is unsupervised and uses the nominal RGA data to adapt weights and biases for nominal reconstruction. Thus, when presented with anomalous sequences, the autoencoder would result in a failed reconstruction since it has never encountered faults during training.

D. LSTM-Autoencoder

Taking into account the fact that there is only nominal data available to the Machine Learning TE team, an autoencoder seems appropriate for this scenario. However, because the data is sequential, temporal features need to be considered, which LSTM layers are most efficient for [2]. Incorporating both architectures into one results in an LSTM-Autoencoder[2], allowing for the reconstruction of sequential nominal RGA data.

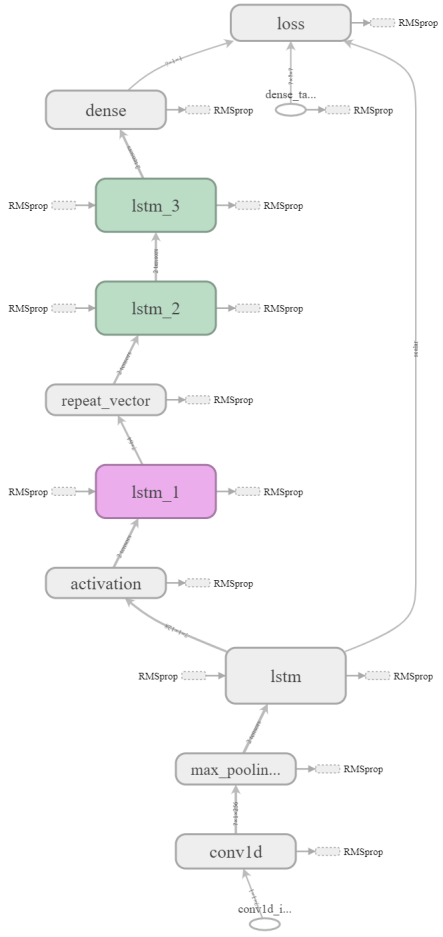

Figure 1. *LSTM Autoencoder model*

[2] LSTM-Autoencoders function similarly to Autoencoders but contain LSTM layers over Dense layers

E. CNN

CNN's are widely applied in computer vision for image analysis and are made up of convolution and pooling layers. The combination of these two layers allow the CNN to extract significant features in data and assign importance (weights and biases) to patterns it finds. In addition to computer vision, CNN's can be used to categorize types of time-series data [1]—in this scenario, classifying between nominal and off-nominal.

III. Neural Network Layers

See https://www.tensorflow.org/api_docs/python/tf/keras/layers for more information on layers

A. Convolution

Convolution is an effective means for deriving the high-level features (based on the filter) in datasets. By performing a series of dot products between the inputs and kernels, convolution allows the neural network to perform efficiently with the reduction of parameters. One-dimensional convolution was used in the LSTM, LSTM-Autoencoder, and CNN.

B. Pooling

Pooling reduces spatial size of convolved features for dimensionality reduction and extraction of prominent features by taking the maximum value of the kernel. One-dimensional max pooling was used in LSTM, LSTM-Autoencoder, and CNN for noise suppression since average pooling does not have this ability.

C. Activity Regularization

Activity regularization acts as a wrapper around a layer to induce learning sparse features and reduce overfitting of the model [4]. An L1 regularizer was used under the Keras module that calculates the sum of absolute values in the output of the first LSTM layer. Applying Rectified Linear Units (ReLU)[3] after regularization allows the neural network to bring the values to a true zero in cohesion with the activation function (since ReLU returns a value between 0 and 1), increasing sparsity[4] compared to applying ReLU before regularization (See Section VII). Activity Regularization was used in the LSTM-Autoencoder.

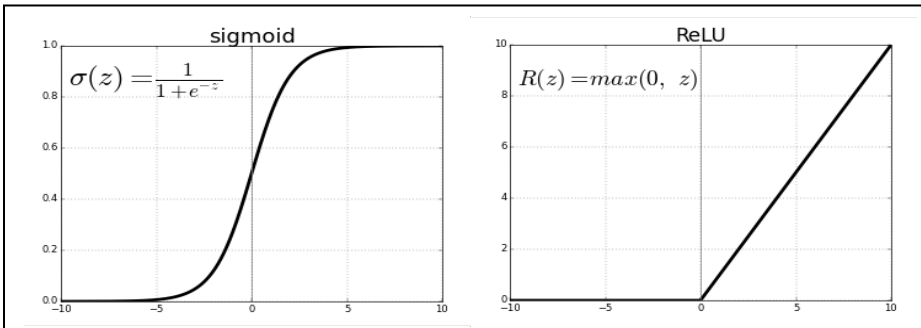

Figure 2. *Sigmoid vs ReLU Activation Functions (Image Downloaded from* https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6)

[3] ReLU is the preferred activation function because it overcomes the vanishing gradient problem where the gradient decreases at an exponential rate during propagation (weights and biases will not be updated effectively)

[4] Sparsity in neural networks simply means most weights are set to 0 for efficiency

D. LSTM

The core idea behind an LSTM is mentioned in Section II (b). Stacked LSTM layers were used in order to allow for greater model complexity in terms of representing features. Return sequences must be set to True when stacking LSTMs in order for the second LSTM layer to have a 3D sequence input. Having a fully connected layer does not allow the neural network to utilize information from a previous time step, missing certain patterns unlike LSTM layers. LSTM layers were used in the LSTM and LSTM-Autoencoder.

E. Repeat Vector

Repeat vectors used to piece together the encoder and decoder paths of the neural network by allowing the decoder to use the same representation when creating each output timestep. It is used to repeat the fixed length vector a given number of times in order to put reshape the 2D output of the encoder into a 3D input for the decoder. Repeat vectors were used in the LSTM-Autoencoder.

F. Dense

Dense layers are used as a fully-connected layer to output predictions based on the input it receives from the previous layer. The LSTM, LSTM-Autoencoder, and CNN use this as a fully-connected layer. However, the autoencoder layers are made up of simply dense layers since the encoder and decoders are just fully-connected feedforward neural networks.

G. Time Distributed

Time distributed wrappers apply a layer to each temporal slice (timestep) of the input independently. This keeps each timestep value separate instead of being flattened (one-to-one relation) [3]. The LSTM-Autoencoder tested was a one to one relationship so Time Distributed wrappers were unnecessary. However, for other types of input-output relationships, time distributed wrappers may be needed (See Figure 3). Adjusting timesteps may benefit accuracy and efficiency of the current LSTM-Autoencoder.

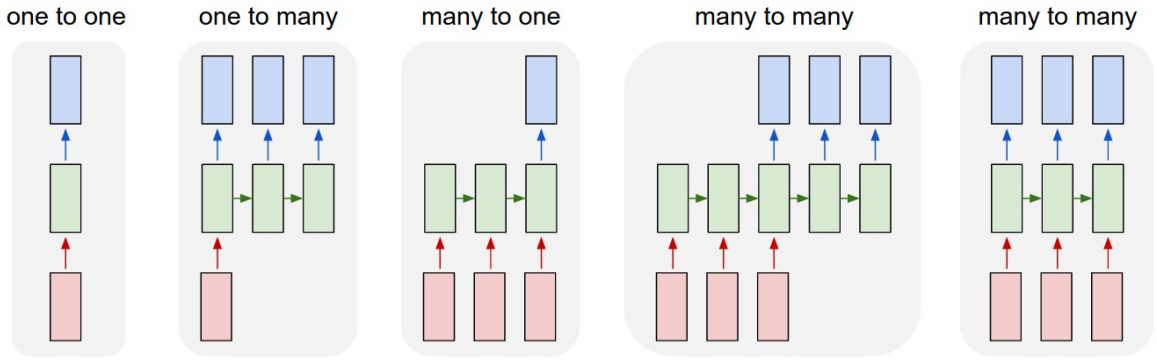

Figure 3. *Input Output combinations of RNN (Image Downloaded from* https://github.com/keras-team/keras/issues/1029)

IV. Reconstruction Error

In order to detect faults in data, the unsupervised model needs to learn the features that are considered nominal. This allows the neural network to reconstruct what it is trained on. Therefore, it will fail to reconstruct off-nominal points. The mean squared error (MSE) is taken between the prediction and input. There will be a high error rate in the reconstruction, indicating faults [10]. Arbitrary thresholds are set to determine which points fall under anomalous.

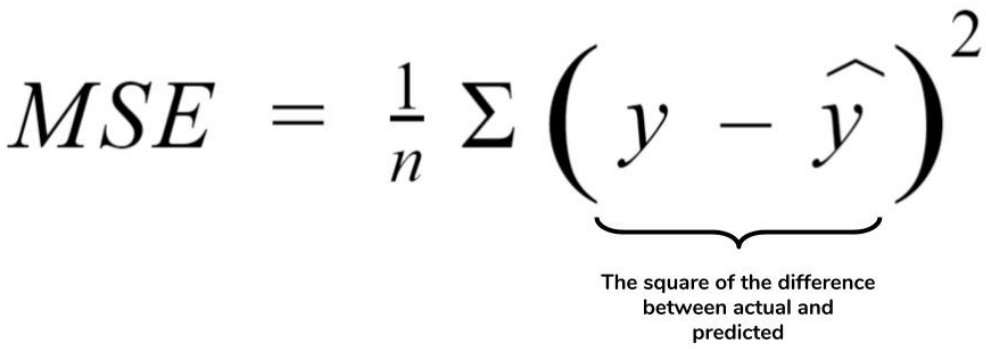

Figure 4. *Mean Square Error Formula (Image Downloaded from* https://www.dataquest.io/blog/understanding-regression-error-metrics/*)*

V. Model Compilation and Training

Each RNN model was compiled using an RMSprop optimizer and MSE as loss. The models were trained on 150 epochs with a batch size of 25. Since autoencoders are used, the training input is a copy of the training labels. There was an issue with the model's ability to detect small outliers so several models had to be used to minimize false positives and obtain significantly more accurate predictions (See Section VII). The CNN model was compiled with the adam optimizer and categorical cross entropy as loss. In order to use categorical cross entropy, the columns have to be one-hot encoded so the column will be created for each output category and a binary variable is inputted for each category.

VI. Testing Procedures

Randomized peaks were added to simulate anomalies that were out of range of the nominal RGA data, as described to by the EV42 team to be one type of faults (See Section VII). Only the Y-Axis was used for simplicity. For the CNN model, 147 columns of nominal FWD RGA y-axis data and 147 columns containing randomly injected peaks were used. Multivariate analysis can be applied in the future through manipulation of input shape to have several features (columns).

VII. Data and Analysis

A. Autoencoder, LSTM, LSTM-Autoencoder Results for FWD RGA Y-Axis Dataset

Figure 4 shows the autoencoder, LSTM, and LSTM-Autoencoder results from training on the entire mission elapsed time (MET) of an FWD RGA Y-Axis dataset. The models failed to predict most anomalies and even contains some false positives. This was primarily due to the two large peaks in between an MET of 100 to 200 and 200 to 300. One method (used in this generation of models) that overcame the reconstruction issue was to subsection the data (See Section VII C) and train 3 models on the subsets. Nevertheless, a more practical method is to manually subsection the data and train a single model with multiple features[5].

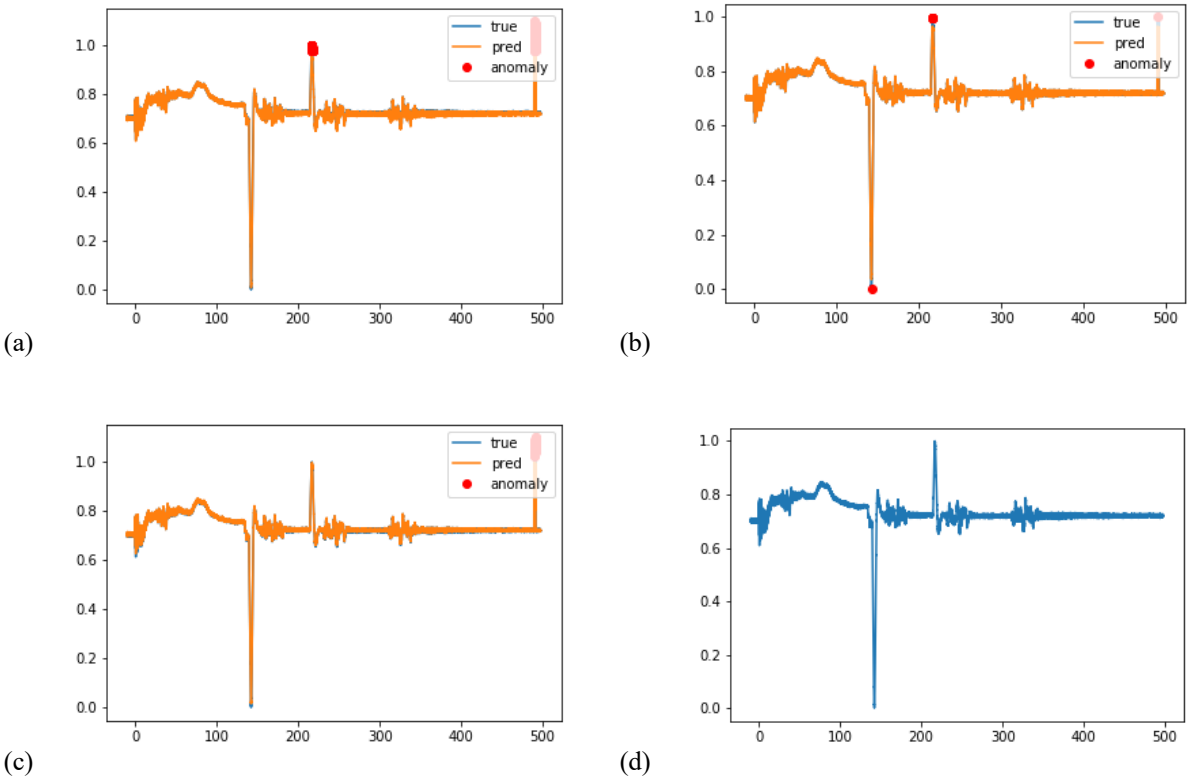

Figure 5. *Initial Prediction Results without Subsections of RGA Y-Axis Data.*

(a) Autoencoder (b) Helmerich's LSTM (Trained like Autoencoder) (c) LSTM-Autoencoder (d) Nominal Data

[5] Unaccomplished due to time limitations

B. LSTM

Helmerich (ES51 Summer Intern 2018) utilizes an LSTM model trained for prediction, not anomaly detection [5]. The method for anomaly detection here is inconsistent with how real data is obtained. In the paper, an outlier point was injected and considered to be anomalous. However, the same nominal dataset (pre-injection) was used in order to find the off-nominal point using the square difference (See Figure 6). In reality, sensors do not provide both anomalous and nominal datasets for the same run. This Siamese twin method that takes in nominal and test data for the input [5], narrowing the problem to a point where the model cannot be reused for other applications and defeating the purpose of a general framework. Helmerich's architecture would be optimal for prediction but should be disregarded for fault detection. When training this LSTM model like an autoencoder for experimental purposes, it predicted many false positives (See Figure 5) and was discontinued in further testing.

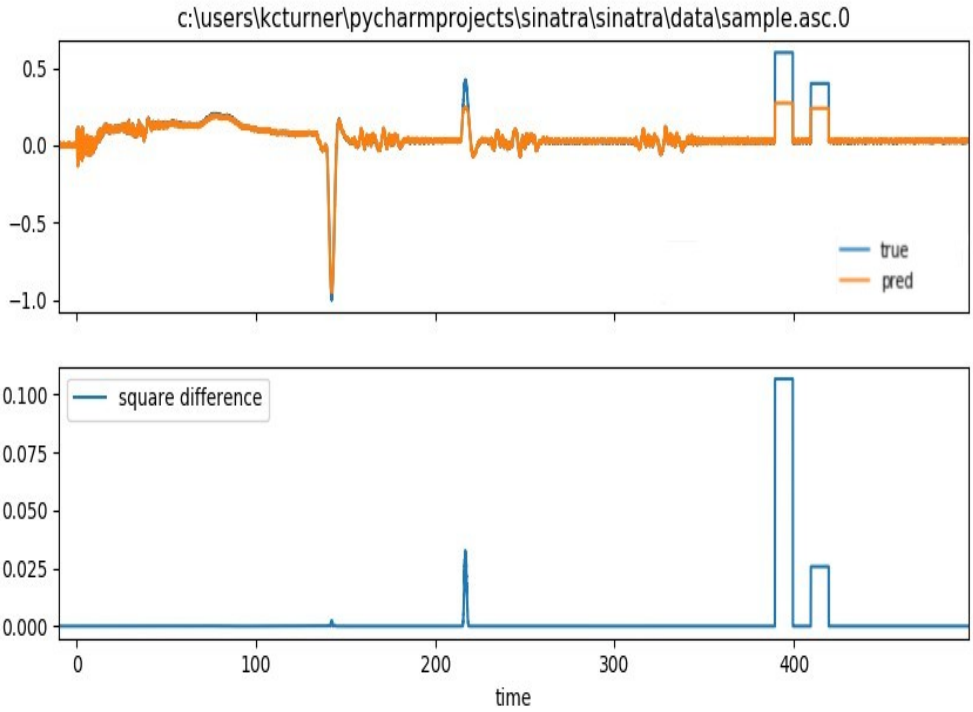

Figure 6. *Helmerich's LSTM results on Subset of FWD RGA Y-Axis data*

C. Subsection Data

The data is split into 5 subsets (Indices: 0:7500, 7500:7800, 7800:11200, 11200:11600, 11600:~25000). All nominal FWD RGA Y-Axis data are analogous and the peaks occur at around the same points. This is where the use of multiple models is introduced. If all these 5 subsets were trained on one model, results would be similar to training the whole dataset at once. Accordingly, 3 separate models of the same architectures were used to analyze one dataset. One model was trained on the minimum peak (7500:7800), one model was trained on the maximum peak (11200:11600), and the last model was trained on the other three data subsets that remained around a fixed range (0:7500, 7800:11200, 11600:).

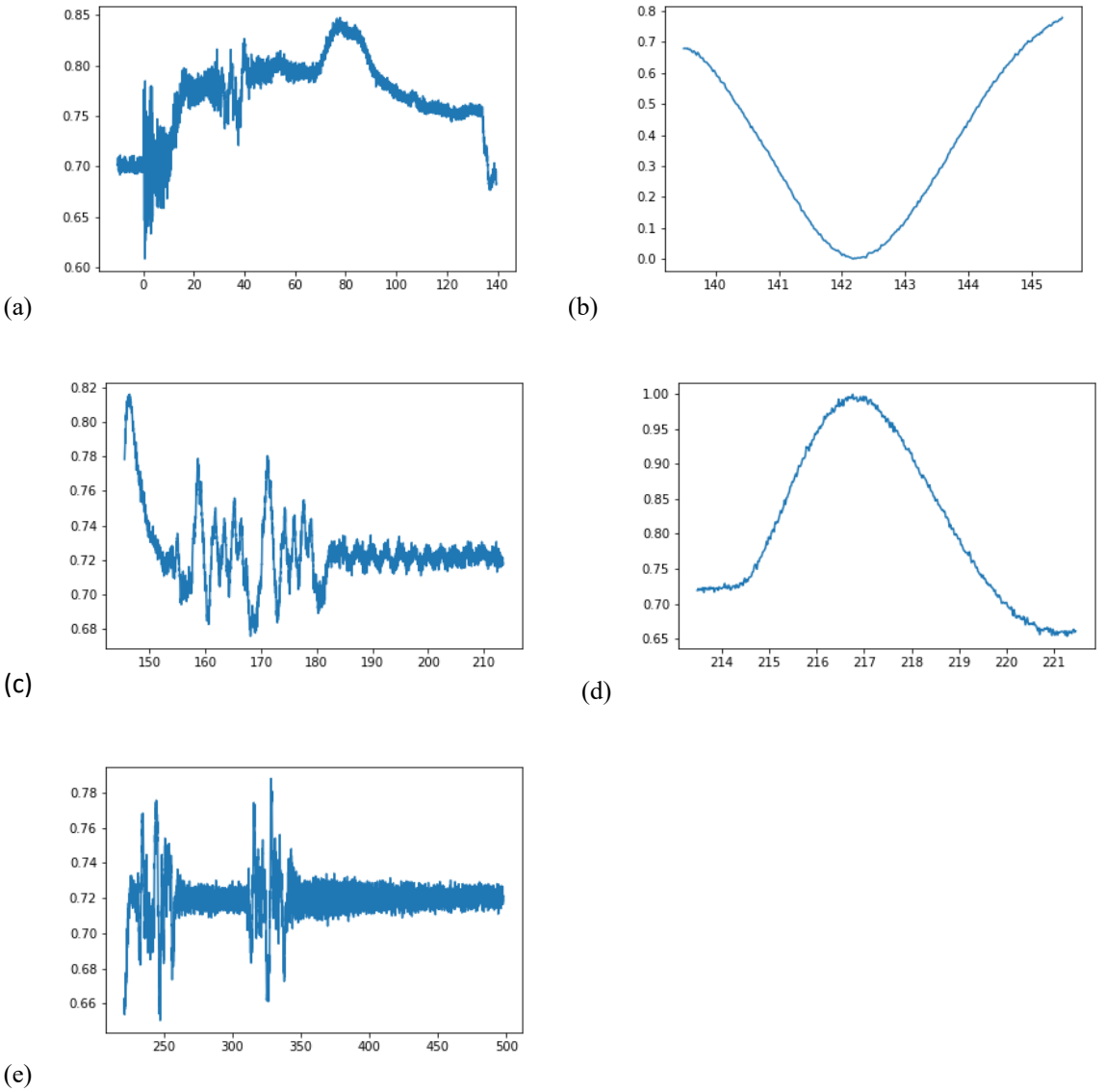

Figure 7. *Subsectioned Nominal FWD RGA Y-Axis Data*

(a) Indices 0:7500 (b) Indices 7500:7800 (c) Indices 7800:11200 (d) Indices 11200:11600 (e) Indices 11600:

D. Autoencoder

Autoencoders are optimal for representation learning. With the limitations in data, autoencoders are able to learn in an unsupervised manner and discover features on its own. The autoencoder resulted in false negatives (See Figure 8e). Because a simple autoencoder does not take into account past information in its predications, LSTM layers were adapted to this model in order to convert it into an LSTM-Autoencoder.

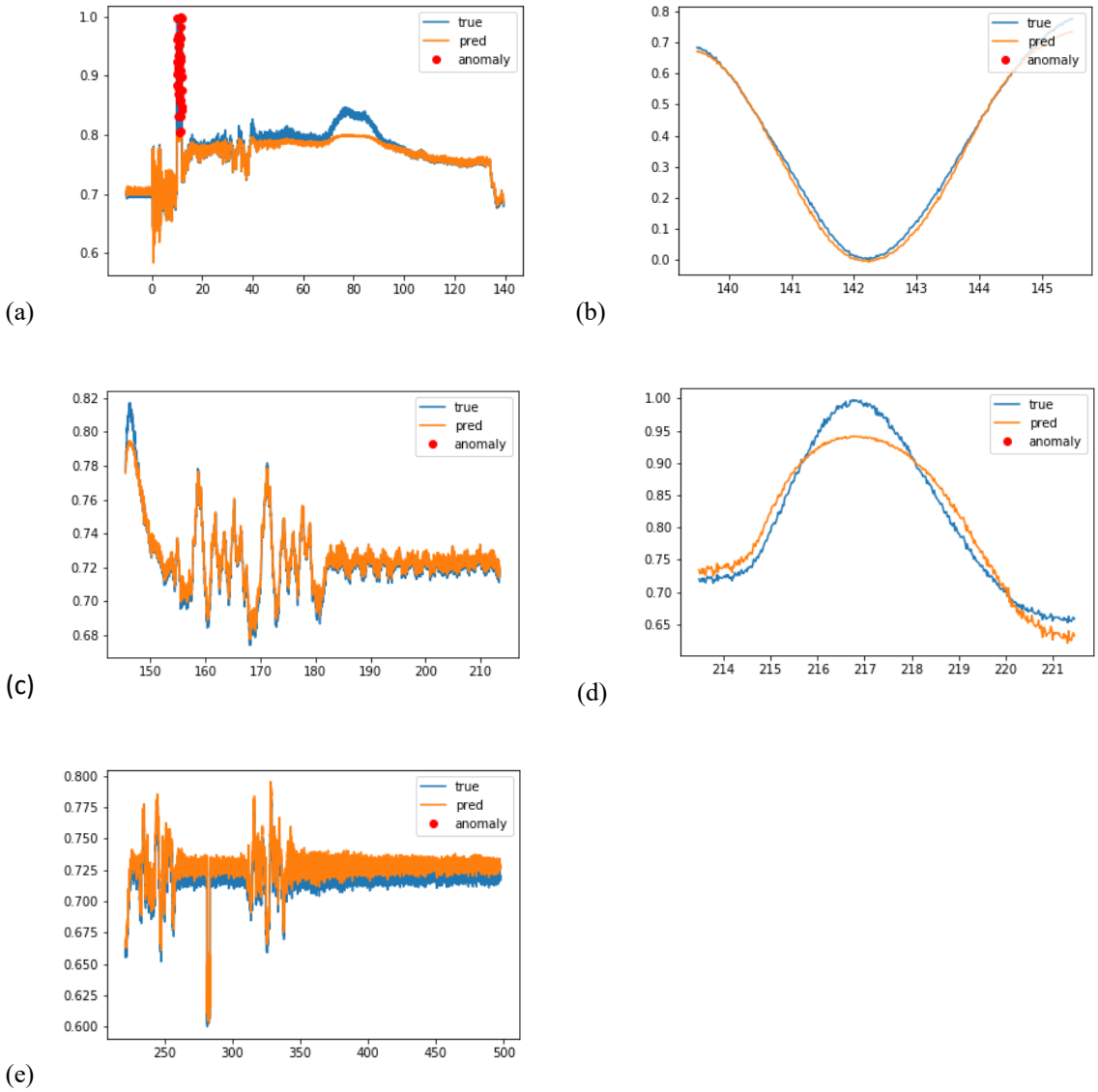

Figure 8. *Autoencoder Reconstruction Results on 5 Subsets of FWD RGA Y-Axis Data*

(a) Indices 0:7500 (b) Indices 7500:7800 (c) Indices 7800:11200 (d) Indices 11200:11600 (e) Indices 11600:

E. LSTM-Autoencoder

LSTM-Autoencoders reconstruct the training data but take into account the temporal dimension unlike regular autoencoders. The LSTM-Autoencoder that executed regularization after ReLU performed more accurately than the LSTM-Autoencoder with regularization before ReLU. Regularizers are highly dependent on the problem. For reconstruction of FWD RGA, because the shape of the curve is about the same, it is preferred for the neural network over fit to the data. Since regularization was performed after ReLU, it is less sparse so more fit to the data it is trained on [4].

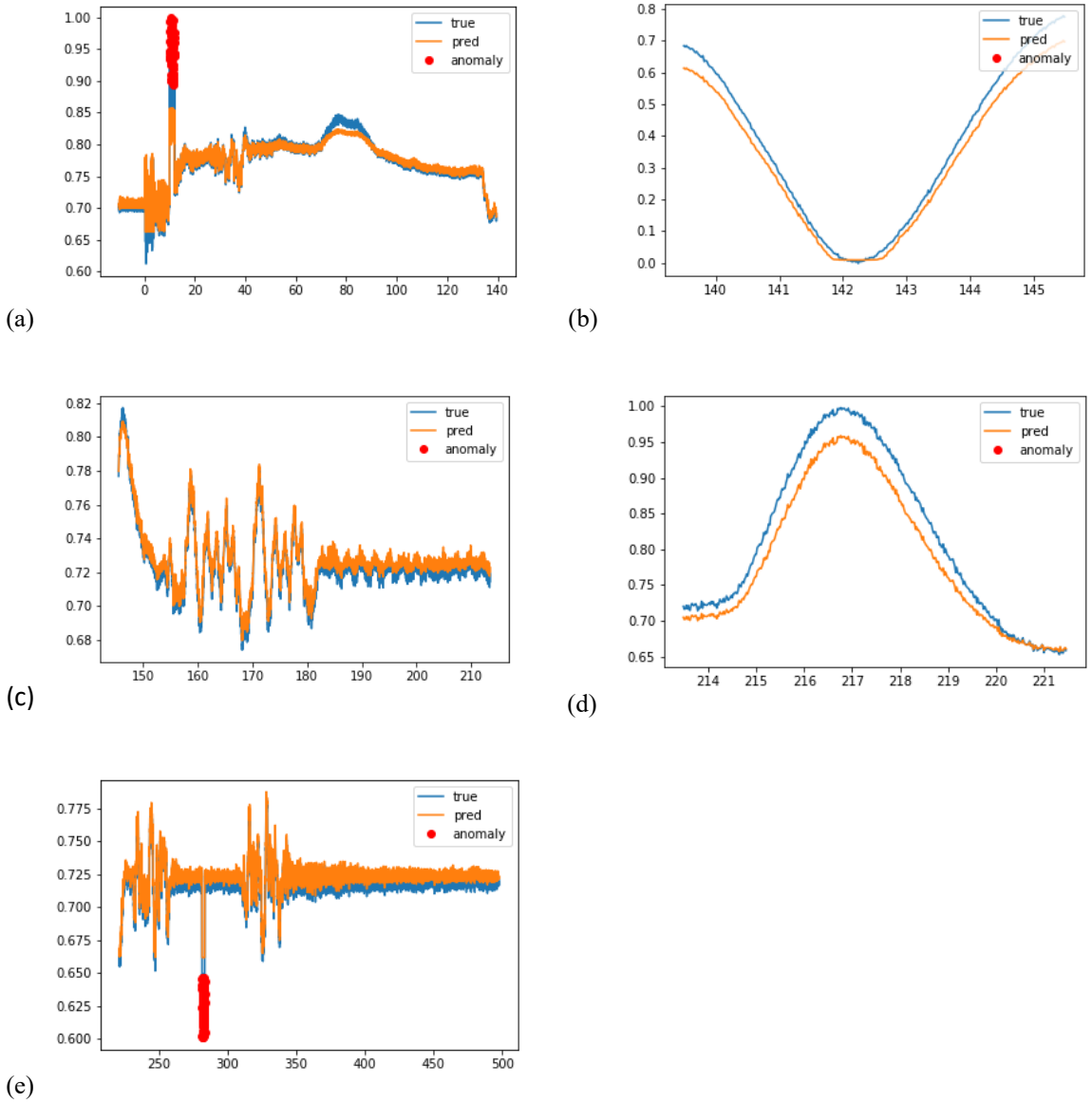

Figure 9. *LSTM-Autoencoder Regularization After ReLU Reconstruction Results on 5 Subsets of FWD RGA Y-Axis Data*

The LSTM-Autoencoder with regularization after ReLU successfully detected 156/200 faults without any false positives and got the location of both anomalous peaks.

(a)

```
pred = model.predict(X_test_scaled_res)
mse = np.mean(np.power(X_test_scaled_res - pred, 2), axis=1)
print(np.where(mse > .003)[0])
print(len(np.where(mse > .003)[0]))
```

```
[1000 1001 1003 1004 1005 1006 1008 1010 1013 1014 1017 1018 1020 1022
 1023 1024 1025 1026 1027 1028 1031 1033 1034 1035 1036 1038 1041 1042
 1043 1044 1046 1049 1050 1051 1052 1053 1054 1055 1056 1057 1060 1062
 1063 1064 1065 1067 1070 1071 1072 1074 1075 1076 1078 1080 1081 1082
 1084 1085 1086 1087 1088 1090 1091 1092 1093 1094 1095 1096 1097 1099]
70
```

(b)

```
pred5 = model.predict(X_test_scaled_res5)
mse5 = np.mean(np.power(X_test_scaled_res5 - pred5, 2), axis=1)
print(np.where(mse5 > .0002)[0])
print(len(np.where(mse5 > .0002)[0]))
```

```
[3001 3002 3003 3004 3005 3006 3007 3008 3009 3010 3012 3013 3014 3015
 3016 3017 3018 3019 3020 3021 3023 3024 3025 3026 3027 3028 3029 3030
 3031 3032 3033 3034 3035 3036 3037 3038 3039 3040 3041 3043 3044 3046
 3047 3048 3049 3050 3052 3053 3054 3055 3056 3057 3059 3060 3061 3062
 3063 3064 3065 3066 3067 3068 3070 3071 3072 3073 3074 3075 3076 3077
 3078 3079 3081 3082 3083 3085 3086 3088 3089 3090 3091 3092 3093 3095
 3096 3097]
86
```

Figure 10. *Prediction of Anomalous FWD RGA Y-Axis from LSTM-Autoencoder with Regularization After ReLU*

(a) Indices 0:7500 (b) Indices 11600:

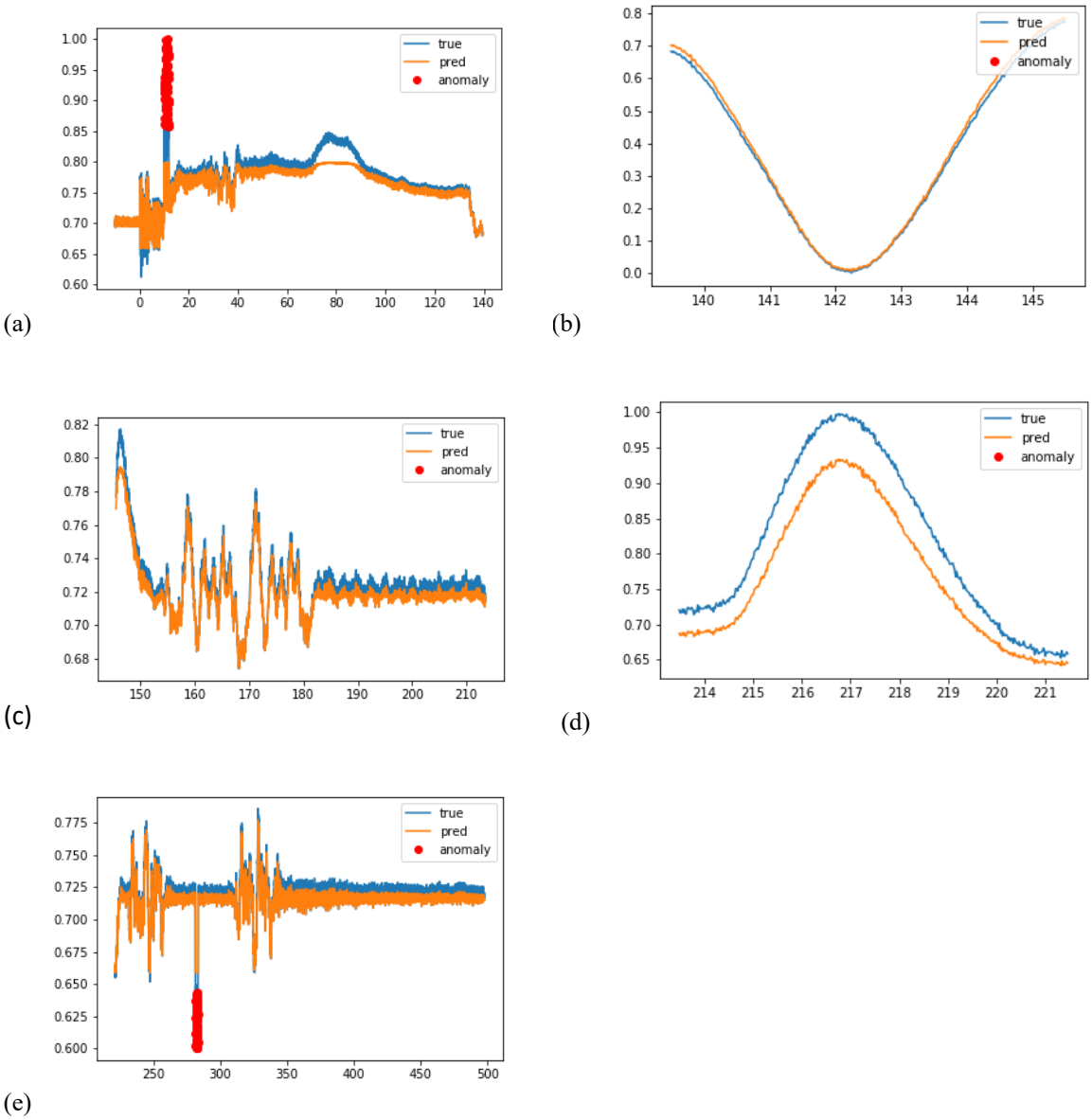

Figure 11. *LSTM-Autoencoder Regularization Before ReLU Reconstruction Results on 5 Subsets of FWD RGA Y-Axis Data*

(a) Indices 0:7500 (b) Indices 7500:7800 (c) Indices 7800:11200 (d) Indices 11200:11600 (e) Indices 11600:

The LSTM-Autoencoder with regularization before ReLU successfully detected 152/200 faults without any false positives and got the location of both anomalous peaks.

```
pred = model.predict(X_test_scaled_res)
mse = np.mean(np.power(X_test_scaled_res - pred, 2), axis=1)
print(np.where(mse > .003)[0])
print(len(np.where(mse > .003)[0]))
```

```
[1000 1003 1004 1008 1009 1010 1011 1013 1016 1017 1018 1019 1020 1022
 1027 1028 1029 1030 1031 1034 1035 1037 1038 1039 1044 1046 1047 1051
 1052 1054 1058 1059 1061 1062 1063 1065 1066 1067 1069 1070 1073 1074
 1076 1078 1079 1080 1081 1082 1083 1085 1087 1088 1089 1092 1093 1095
 1096 1097 1099]
59
```

(a)

```
pred5 = model.predict(X_test_scaled_res5)
mse5 = np.mean(np.power(X_test_scaled_res5 - pred5, 2), axis=1)
print(np.where(mse5 > .0002)[0])
print(len(np.where(mse5 > .0002)[0]))
```

```
[3000 3001 3002 3003 3004 3005 3007 3008 3009 3010 3011 3012 3013 3014
 3015 3016 3017 3018 3019 3020 3021 3022 3023 3024 3025 3026 3027 3028
 3029 3030 3031 3032 3033 3034 3035 3036 3037 3039 3040 3041 3042 3043
 3044 3045 3046 3047 3048 3049 3050 3051 3052 3054 3055 3056 3057 3058
 3060 3061 3062 3063 3064 3065 3066 3067 3068 3069 3070 3071 3072 3073
 3074 3075 3076 3077 3078 3079 3080 3082 3083 3084 3086 3087 3088 3089
 3090 3091 3092 3093 3095 3096 3097 3098 3099]
93
```

(b)

Figure 10. *Prediction of Anomalous FWD RGA Y-Axis from LSTM-Autoencoder with Regularization Before ReLU*

(a) Indices 0:7500 (b) Indices 11600:

F. CNN

The results of the CNN were negligible. Two different architectures were used (See Appendix (6) and Reference [1]) The CNN was tested on 25000 points to keep to datasets uniform and also 2000 points to see if there was an issue with noise but the model still classified every dataset as anomalous. Once EV42 generates datasets with their definition of faults, the CNN model may be able to classify these anomalies. However, this model should be discontinued until labeled data is provided.

VIII. Discussion

The LSTM-Autoencoder was the most accurate out of the 3 RNN's tested, detecting 156/200 of the anomalous injected points. However, the LSTM and Autoencoder models should not be disregarded in future applications. LSTMs can be used for prediction purposes while simple autoencoders could be used to reconstruct non-sequential data. Furthermore, CNN's could be applied to a multitude of classification problems. Though the CNN trained was defective, architecture adjustments and additional anomalous data provided by EV42 may allow for a valid results.

Machine learning introduces an additional layer of complexity in pattern analysis that may be overlooked by humans. Unsupervised models are able to discover features and patterns on its own in order to make a prediction. One deep learning model could also be used for different types of datasets, reducing the amount of test cases required across SLS sensor testing. This report covers only one axis of RGA data for simplicity but the capacity of deep learning has transcended to the point where other sensor data can be trained on the same LSTM-Autoencoder to detect anomalies, allowing for a general framework that encompasses different types of data. Continual experimentation with multivariate fault detection, architecture adjustments, and hyper-parameter tuning would allow the machine learning model to far exceed its current abilities.

Appendix

1) Lstm (See Helmerich Paper)

```
model = Sequential()
model.add(Conv1D(filters=256, kernel_size=10, padding='same',
                                    activation='relu', input_shape=(1, 1)))
model.add(LSTM(128, return_sequences=True))
model.add(LSTM(128,return_sequences=True))
model.add(Dense(1, activation='sigmoid'))
model.summary()
```

2) Autoencoder

```
input_dim = X_train.shape[1]
encoding_dim = 6

input_layer = Input(shape=(input_dim, ))
encoder = Dense(encoding_dim, activation="tanh",
                activity_regularizer=regularizers.l1(10e-5))(input_layer)
encoder = Dense(int(encoding_dim / 2), activation="tanh")(encoder)
encoder = Dense(int(2), activation="tanh")(encoder)
decoder = Dense(int(encoding_dim/ 2), activation='tanh')(encoder)
decoder = Dense(int(encoding_dim), activation='tanh')(decoder)
decoder = Dense(input_dim, activation='tanh')(decoder)
autoencoder = Model(inputs=input_layer, outputs=decoder)
autoencoder.summary()
```

3) Lstm-Autoencoder with ReLU Before Regularization

```
model = Sequential()
model.add(Conv1D(filters=256, kernel_size=10, padding='same',
                                     activation='relu', input_shape=(1, 1)))
model.add(MaxPooling1D(pool_size=1, strides=1, padding='valid'))
model.add(LSTM(128, activation='relu', input_shape=(1,1),
                return_sequences=True,activity_regularizer=l1(0.0000001)))
model.add(LSTM(64, activation='relu', return_sequences=False))
model.add(RepeatVector(1))
model.add(LSTM(64, activation='relu', return_sequences=True))
model.add(LSTM(128, activation='relu', return_sequences=True))
model.add((Dense(1)))
model.summary()
```

4) **Lstm-Autoencoder with ReLU After Regularization**

```
model = Sequential()
model.add(Conv1D(filters=256, kernel_size=10, padding='same',
                                 activation='relu', input_shape=(1, 1)))
model.add(MaxPooling1D(pool_size=1, strides=1, padding='valid'))
model.add(LSTM(128, activation='linear', input_shape=(1,1),
               return_sequences=True,activity_regularizer=l1(0.0000001)))
model.add(Activation('relu'))
model.add(LSTM(64, activation='relu', return_sequences=False))
model.add(RepeatVector(1))
model.add(LSTM(64, activation='relu', return_sequences=True))
model.add(LSTM(128, activation='relu', return_sequences=True))
model.add((Dense(1)))
model.summary()
```

5) **Reconstruction Error**

```
pred = model.predict(X_test_scaled_res)
mse = np.mean(np.power(X_test_scaled_res - pred, 2), axis=1)
print(np.where(mse > .003)[0])
```

6) **CNN**

```
model.add(Conv1D(256, 10, activation='relu', padding = 'same', input_shape=(25000, 1)))
model.add(Conv1D(256, 10, activation='relu', padding = 'same'))
model.add(MaxPooling1D(3))
model.add(Conv1D(128, 10, activation='relu', padding = 'same'))
model.add(Conv1D(128, 10, activation='relu', padding = 'same'))
model.add(MaxPooling1D(3))
model.add(Dropout(0.3))
model.add(Flatten())
model.add(Dense(num_classes, activation='softmax'))
print(model.summary())
```

References

[1] Ackermann, N. (2018, September 4). Introduction to 1D Convolutional Neural Networks in Keras for Time Sequences. *Medium*. Retrieved from https://blog.goodaudience.com/introduction-to-1d-convolutional-neural-networks-in-keras-for-time-sequences-3a7ff801a2cf

[2] Brownlee, J. (2018, November 5). A Gentle Introduction to LSTM Autoencoders**.** *Machine Learning Mastery.* Retrieved from https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/

[3] Brownlee, J. (2017, May 17). How to Use the TimeDistributed Layer for Long Short-Term Memory Networks in Python**.** *Machine Learning Mastery.* Retrieved from https://machinelearningmastery.com/timedistributed-layer-for-long-short-term-memory-networks-in-python/

[4] Brownlee, J. (2018, November 30). How to Reduce Generalization Error With Activity Regularization in Keras. *Machine Learning Mastery.* Retrieved from https://machinelearningmastery.com/how-to-reduce-generalization-error-in-deep-neural-networks-with-activity-regularization-in-keras/

[5] Helmerich, C. (2018 August). Machine Learning for SLS Flight Software Anomaly Detection and Analysis. *NASA*.

[6] Korneev, E. (2018, December 20). LSTM Neural Networks for Anomaly Detection. *Medium*. Retrieved from https://medium.com/datadriveninvestor/lstm-neural-networks-for-anomaly-detection-4328cb9b6e27

[7] Kostadinov, S. (2017, December 6). How Recurrent Neural Networks Work. *Towards Data Science*. Retrieved from https://towardsdatascience.com/learn-how-recurrent-neural-networks-work-84e975feaaf7

[8] Malhotra, P., Ramakrishnan, A., Anand, G., Vig, L., Agarwal, P., & Shroff, G. (2016 July). LSTM-based Encoder-Decoder for Multi-sensor Anomaly Detection. *Tata Consultancy Services Ltd.* Retrieved from https://arxiv.org/pdf/1607.00148.pdf

[9] Olah, C. (2015, August 27). Understanding LSTM Networks. *Github.io.* Retrieved from https://colah.github.io/posts/2015-08-Understanding-LSTMs/

[10] Rajaratne, M. (2018, November 17). Credit Card Fraud Detection using Autoencoders in H2O. *Towards Data Science.* Retrieved from https://towardsdatascience.com/credit-card-fraud-detection-using-autoencoders-in-h2o-399cbb7ae4f1

[11] Ranjan, C (2019, June 4) Step-by-step understanding LSTM Autoencoder Layers. *Medium.* Retrieved from https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352