Data Anomaly Detector - CS 7643

Lily Chebotarova, Raga Lasya Munagala, Ligeng Peng, Brian Zhang
Georgia Institute of Technology
lily.chebotarova@gmail.com, rmungala6@gatech.edu, kevinplg@gatech.edu, brianxicheng@gmail.com

Abstract

Anomaly detection is an important and prevalent problem across multiple domains, such as credit card fraud, medical diagnostics, and many others. Existing common approaches applied to anomaly detection often consist of supervised machine learning such as clustering or ensemble methods like random forest. These approaches often have limitations in their ability to learn complexities of anomalous data structures. In this project, we examine leading papers describing novel methods in the field of anomaly detection for timeseries data. We reproduce the results of these papers while ekeing out additional performance by fine-tuning of the models when applied to an established labelled dataset for anomaly detection, "S5- A Labeled Anomaly Detection Dataset". The first method examined is a Convolutonal-Long-Short-Term-Memory (C-LSTM) neural network, as described in the paper by Tae-Young Kim and Sung-Bae Cho [3]. Model performance similar to that of the authors is demonstrated using a pytorch implementation of the model architecture described. We also achieve similar performance with an autoencoder architecture proposed by several papers in the anomaly detection field [2, 5].

1. Introduction/Background/Motivation

Anomalies represent some sort of irregularity in the normal pattern or the prevailing trend of data, or in other words they are data points that stand out from the others and deviate from observed and expected behavior in the data. Anomalies can include outliers, drifts in data or other changes in the system. Perhaps the most common type of anomaly detection problem, and the type addressed by this paper, occurs in time series data. Anomaly detection is a common problem across many domains, with potentially profound implications if addressed effectively. The ability to accurately identify anomalies enables proactive detection of irregular patterns, diagnostic of potential issues, and enhancement of the quality and reliability of various services. Anomalies in web-traffic data may indicate potential issues such as unexpected spikes in user traffic which may be due to cyber-attacks, system failures, or viral trends.

The current state of the art in timeseries anomaly detection primarily consists of deep learning methods, which together with the high prevalence and importance of this problem, makes it an ideal candidate for study in this paper. Deep Learning methods have been found to significantly outperform the more traditional methods, as they are able to learn hierarchical features and capture complexities in the data and perform better at large scale [1]. Common deep learning methods employed in this field include Long Short-Term Memory (LSTM) networks and autoencoders, both of which we examine in this paper. RNN-based methods such as these are particularly well-suited for timeseries problems which inherently consist of sequential data [1]. Autoencoders have also been successfully applied to timeseries anomaly detection in unsupervised scenarios and performed better than methods such as Local Outlier Factor and Principal Component analysis [2].

Our paper examines leading novel approaches in this field of anomaly detection and compares the relative performance of these methods. Our paper examines leading novel approaches in this field of anomaly detection and compares the relative performance of these methods. The first method examined is C-LSTM neural network applied to time series web traffic data, as described in the paper by Tae-Young Kim and Sung-Bae Cho [3]. Model performance similar to that of the authors is demonstrated using a pytorch implementation of the model architecture described. The second method reproduced is an autoencoder approach, as described in the paper by Lawrence Wong, et al [5].

The paper by Tae-Young Kim and Sun-Bae Cho [3] offers an improvment over conventional LSTM approaches by integrating a convolutional layer as the input for LSTM layer, reducing temporal variations and improivng overall performance. Similarly, the paper by Lawerence Wong, et al. [5] offers an improvment over conventional approaches by combining separate autoencoder and LSTM approaches models into a single composite model optimizing a joint objective function, and able to make bi-directional predictions. We will compare the effectiveness of the two approaches in-

vestigated in this paper to see which offers the best performance on the anomaly detection dataset selected.

We will compare the effectiveness of the two approaches investigated in this paper to see which offers the best performance on the anomaly detection dataset selected. Timely detection of these anomalies is essential for organzing a prompt response and minimizing the impact to system performance and user experience. In short, a robust and effective anomaly detection method for a given problem should allow us to proactively detect irregular patterns and diagnose potential issues, thus enhancing the quality and reliability of the services tracked by this timeseries data.

Our paper uses the dataset, S5 A Labeled Anomaly Detection Dataset [4], which offers a diverse range of real and synthetic time-series data specifically curated for the purpose of anomaly detection. The dataset is publicly available via Yahoo Research [4]. This dataset was chosen because it offers a diverse set of edge cases for testing our models, including outliers and various change-points. The time series data also includes varying trend, noise, and seasonality which enables us to better test the robustness of our model implementations. The real portion of the dataset is made up of the web traffic metrics of various yahoo services, alongside synthetic data added to the dataset.

2. Approach

All models, data, and code are contained in the team git repo at https://github.com/brian-x-zhang/CS7643-Anamoly.

2.1. Pre-processing

First, we applied pre-processing steps and exploratory analysis to the data. We focused on the A1 part of the Yahoo dataset that consists of the real web traffic with manual human-applied labels for anomalies. It is presented in 67 separate files with traffic measures from various Yahoo web services and anomaly data. The data in this dataset is represented by a time series of traffic measurement values from actual web series in one hour units. Since the labeling was manually done, there is potential for some inconsistency in data.The traffic also presents a high level of variability. The data from 67 files was concatenated for training, values were normalized and any values in traffic data that were equal to zero were dropped. Normalizing data was important as it ensured that we didn't have features with larger scales or variances that could dominate the learning process. The values were normalized using the Euclidean norm. This was achieved using the preprocessing.normalize function available in scikit-learn in Python. The resulting normalized has each row scaled such that the Euclidean norm (the square root of the sum of squares of each element) of each row is 1. The normalized web-traffic timeseries data is shown in Figure 1.

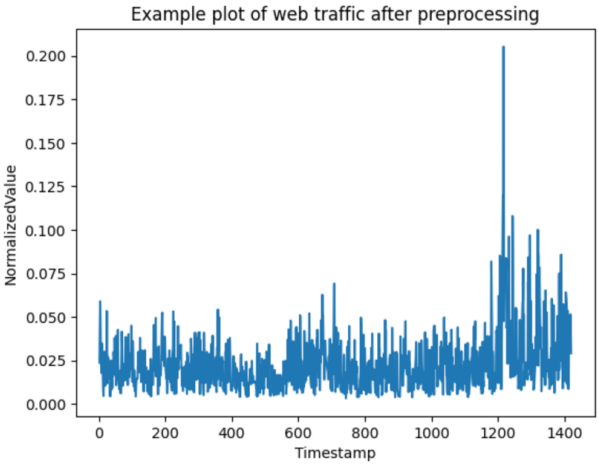

Figure 1. Pre-processed Web Traffic Timeseries Data

The structure of the problem is characterized by a highly imbalanced dataset, with normal network activity dominating and anomalies representing a minority class. Considering these characteristics of the data, we employed the sliding window method to the data. In this sliding window technique, we created windows of data by sliding a fixed-size window across the data samples. For the data presented in this dataset, we created a sliding window of length 60.

To perform anomaly detection on the preprocessed dataset, we trained and evaluated an Autoencoder and C-LSTM network. We also further compared the performance to traditional machine learning methods such as Random Forest.

In response to addressing the highly imbalanced nature of the dataset, we have innovatively incorporated a sliding window methodology as part of our preprocessing strategy. This approach entails carefully segmenting the dataset into distinct windows, allowing for a more balanced representation of both normal and anomaly instances. By implementing this sliding window technique, we aim to mitigate the inherent imbalance and create a more equitable distribution of data points, fostering a robust learning environment for our model. This comprehensive evaluation involves comparing the performance of our model across different preprocessing strategies to discern the most effective approach for anomaly detection.

2.2. Autoencoder

The structure of the autoencoder model was designed to reflect the underlying imbalanced makeup of the dataset. We start the problem with 70% training dataset and 30% testing dataset. The encoder is trained on a subset of the training dataset, capturing the essential characteristics of the network traffic, before being trained per the following steps:

1. Created an autoencoder with an encoder and a decoder

in PyTorch.

2. Trained the autoencoder using the training dataset.
3. Extracted the encoder from the trained autoencoder.
4. Built a classifier using the extracted encoder for feature representation.
5. Trained the classifier using the features from the training set.
6. Evaluated the model on a separate test set, passing inputs through the autoencoder and then through the classifier, calculating accuracy for anomaly detection.

In the autoencoder model, the parts with learned parameters were primarily associated with the encoder and decoder components. These components consisted of linear layers (fully connected) and activation functions, such as ReLU, designed to transform and compress the input data. Specifically, the encoder learned to map the input features to a lower-dimensional representation, capturing essential information about normal network behavior. Conversely, the decoder learned to reconstruct the input data from this compressed representation 2

The post-processing classifier, which was applied for decision-making or anomaly identification, involved a separate linear layer followed by a sigmoid activation function. This part of the model facilitated the conversion of the encoded features into decision probabilities, determining whether a given input represented normal or anomalous network activity. While the weights of the linear layer in the post-processing step were learned during training, this part of the model served more as a decision-making component than a feature extraction one.

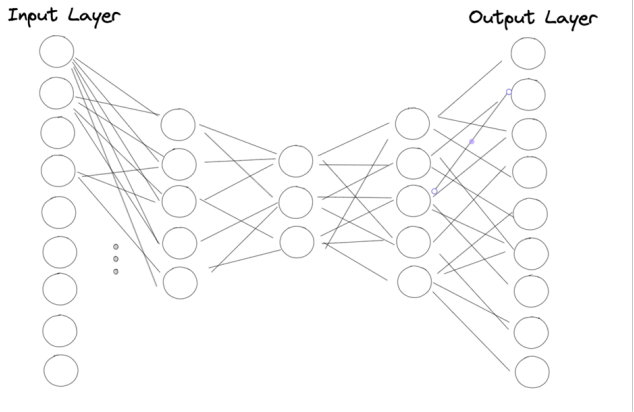

Figure 2. Autoencoder-Decoder Diagram

In the quest to optimize the autoencoder model, several hyperparameters were explored and fine-tuned to enhance overall performance. The initial architecture, characterized by three linear layers in both the encoder and decoder with hidden dimensions (16, 8, 4), underwent significant adjustments. The optimized architecture featured larger hidden dimensions in the encoder (64, 32, 16) and a mirrored structure in the decoder (16, 32, 64). This architectural refinement aimed to empower the model with the capacity to capture more intricate patterns within the data. Moreover, the learning rate, a critical factor influencing training dynamics, was systematically tested at values of 0.001, 0.005, and 0.01. Through careful evaluation, a learning rate of 0.005 emerged as the most effective in striking a balance between convergence speed and stability.

Batch size, another crucial hyperparameter, was subjected to experimentation with various sizes. The optimal batch size was identified as 512, showcasing its pivotal role in facilitating efficient model training and convergence. Furthermore, the choice of optimizer played a significant role in the training process. The Adam optimizer was selected for its effectiveness in optimizing neural networks, contributing to the overall success of the model. In our initial attempt to model the data directly, we encountered a significant challenge due to the highly imbalanced nature of the dataset. Despite achieving an impressive overall accuracy of 98.15%, the F1 score, a critical metric for evaluating model performance, remained notably low at 13%. This low F1 score suggested that the model struggled to discern and learn the essential patterns or features crucial for accurate predictions. Recognizing the limitations of our first approach, we pivoted to employ the sliding window method. This strategic adjustment proved instrumental in addressing the imbalance issue and yielded tangible improvements, particularly in the F1 score. The sliding window method allowed for a more nuanced analysis of the data, contributing to a refined understanding of the underlying patterns. Consequently, our model exhibited enhanced performance, showcasing the importance of tailored preprocessing techniques in handling imbalanced datasets and improving overall predictive accuracy.

2.3. C-LSTM

The C-LSTM model is particularly well-suited for anomaly detection in time series data due to its unique architecture that combines the strengths of convolutional neural networks (CNNs) and long short-term memory networks (LSTMs). Real web traffic data with manual anomaly labels presented in the Yahoo data set has patterns of spatial and temporal information. LSTMs are traditionally an effective choice for anomaly detection especially on sequential data due to their ability to extract temporal features of the data. Additionally, the ability of these models to capture long-term dependencies in time-series data also makes them more adaptable to evolving data patterns, which is crucial for accurately identifying anomalies over time. Further, the

CNN layers provide the benefit of reduction of variation in spatial information. This combination allows the model to effectively capture both spatial and temporal features in the data, which are crucial for accurately identifying anomalies [3] to achieve higher performance for anomaly detection in time series. We attempt to replicate the results of the model in the paper by Tae Young Kim, Sung Bae Cho [3].

The data was pre-processed using the sliding window approach, with the length of the web traffic window set to 60.Our model was built in PyTorch as opposed to Keras model presented in the paper, with architecture presented in Table 1. First, there are 2 convolutional layers, each followed by the tanh activation and max pooling. Then the LSTM is applied to extract the temporal features from the data. Finally, two fully-connected linear layers and softmax are applied for the final classification task and output layer. We noticed differences in behavior of the model in Pytorch as compared to Keras model, which can potentially be explained by the underlying differences in how these frameworks handle certain factors in the model architecture and training, like weight initialization, loss functions etc. Due to some of these differences, the total number of parameters and certain attributes have been modified for Pytorch implementation. Additionally, gradient clipping was implemented to prevent exploding gradients.

2.4. Hyperparameter Tuning

As part of optimizing the C-LSTM and autoencoder models, we employed a detailed hyperparameter tuning framework designed to optimize the performance of both Convolutional Long Short-Term Memory (C-LSTM) and Autoencoder models. This framework is pivotal in determining the most effective configurations for our models, ensuring they are finely tuned for the specific tasks of feature representation and anomaly detection. We adopted a systematic approach to explore a wide range of hyperparameters, including but not limited to, layer dimensions, learning rates, and activation functions. The range of tuning parameters for the C-LSTM and Autoencoder models are shown is shown in Figure 1 and Figure 2, respectively.

Table 1. Hyperparameter tuning grid for C-LSTM model

Hyperparameter	**Values**
Conv1 Output Channels	16, 32
Kernel Size	3, 5
LSTM Hidden Size	16, 32
FC1 Output Features	4, 16, 32
Learning Rate	0.001, 0.01
Optimizer	Adam
Criterion	BCELoss
Epochs	200

For the C-LSTM model, hyperparameter tuning focused on its unique architecture that combines convolutional layers with LSTM units. Key hyperparameters include the number and size of convolutional filters, kernel sizes, the number of LSTM units, and the architecture of fully connected layers. Given the sequential nature of our data, special attention was given to the size of the LSTM units, which play a crucial role in capturing temporal dependencies. The convolutional layers were fine-tuned to effectively extract spatial features before they were fed into the LSTM units. The learning rate was also adjusted to balance the speed and stability of the training process.

Table 2. Hyperparameter tuning grid for Autoencoder model

Hyperparameter	**Values**
Encoder Layer Sizes	[32,16,8],[64,32,16],[128,64,32]
Autoencoder Learning Rate	0.001, 0.01, 0.05
Classifier Learning Rate	0.0005, 0.001, 0.005
Activation	LeakyReLU, ReLU
Optimizer	Adam
Criterion	MSELoss
Epochs	200

For the autoencoder tuning focused on the encoder and decoder's layer sizes and the activation functions used. The depth and breadth of these layers were especially important in determining the model's ability to compress and reconstruct the input data effectively. The leaky ReLU actiation function was tested in addition to the original ReLU activation, as it was hypothesized that outliers we are seeking to focus on in an anomaly detection problem may be better represented by allowing a small gradient for negative values (over the dead-neurons at the zero-cutoff for ReLU).

3. Experiments and Results

3.1. Experiments

Throughout the development of the autoencoder and C-LSTM models for anomaly detection, a series of experiments were systematically conducted to address various challenges and enhance the model's overall performance. Given the highly imbalanced nature of the initial dataset, the effectiveness of different techniques, such as the sliding window method and subsampling, was thoroughly explored to balance the dataset and improve the model's ability to detect anomalies. Additionally, hyperparameter tuning played a crucial role in optimizing the model. The architecture of the autoencoder, including the number of layers and neurons per layer, as well as hyperparameters like the learning rate and batch size, were systematically adjusted and tested to identify the most effective configuration. The training process involved monitoring the loss function over 100 epochs, ensuring convergence and assessing the model's learning progress.

In conjunction with the autoencoder, a classifier was trained on the encoded features to perform binary classification. The training of this classifier was evaluated over multiple epochs to gauge its effectiveness in distinguishing between normal and anomalous instances. Quantitative evaluation experiments included the calculation of various metrics such as accuracy, precision, recall, and F1-score on a separate test set. These metrics provided valuable insights into the model's ability to correctly classify instances and handle class imbalances effectively. Finally, the performance of the autoencoder model was compared with other models, such as a Random Forest classifier, to assess its relative effectiveness in anomaly detection. Through these comprehensive experiments, the iterative refinement of the model aimed to overcome challenges, optimize its architecture, and achieve accurate anomaly detection.

3.2. Results

The dataset we had is highly imbalanced. It's important to note that while the accuracy metric is commonly used, its reliability can be compromised in the presence of imbalanced datasets. For instance, if anomalies are significantly outnumbered by normal cases, a high overall accuracy may be misleading. In the context of anomaly detection, the actual detection of anomalies becomes crucial. In this scenario, precision, recall, and F1-score metrics can provide a more nuanced evaluation of the model's performance. Precision evaluates the accuracy of positive predictions, recall assesses the model's ability to capture all actual positives, and the F1-score balances precision and recall. These metrics are particularly useful for addressing class imbalance and providing a more comprehensive understanding of the model's strengths and weaknesses.

3.2.1 C-LSTM

To train the C-LSTM model, Adam Optimizer and Binary Cross entropy were used. Binary Cross entropy is appropriate for this task since our task is binary classification. An important consideration in using this loss function for anomaly detection is that it may favor the majority class. However, since we performed the sliding window transformation for the input data, it offers some compensation for the class imbalances. After preprocessing with a sliding window of length 60, we ended up with 92619 windows, of which 10194 were windows with anomaly. Compared to the initial dataset, where only 0.02% The initial learning rate of 0.01 provides a desirable convergence within the 100 epochs used and with batch size of 512. We experienced instability in the model with training loss spiking at around 60-70 epochs before decreasing and leveling off again. Implementing gradient clipping has resulted in more stable gradients and improved performance. See Appendix 2 for training loss curve and confusion matrix results. Recall of 62.21% achieved by C-LSTM is not extremely high, but indicates that the model sensitivity is capturing a reasonable proportion of actual anomalies, while The f1 score of 64.27% further suggests that the model is also capable of identifying anomalies with some degree of accuracy, balancing precision 66.46% and recall. Overall accuracy achieved was 91.78%.

3.2.2 Autoencoder

While the quantitative results indicate a decrease in the reconstruction loss, a qualitative analysis involves inspecting the reconstructed outputs visually or through other means. The loss function used in the provided code is the Mean Squared Error (MSE) loss. The MSE loss is computed between the reconstructed outputs (outputs) generated by the autoencoder and the original inputs (inputs). It measures the average squared difference between corresponding elements of these two tensors. The criterion variable, which is used to calculate the loss, is initialized as nn.MSELoss() earlier in the code. The model then minimizes this loss during training to improve the reconstruction performance and learn a meaningful representation of the input data.

The progression from an initial loss of 0.4728 to a final loss of 0.1211 highlights the effectiveness of the Autoencoder in minimizing the dissimilarity between the original and reconstructed data. This trend suggests that the model has successfully captured meaningful representations of the input features, resulting in a more refined and optimized reconstruction of the data. Such a quantitative improvement supports the overall success of the Autoencoder in its learning objectives. The success of the Autoencoder model can be assessed based on the performance metrics summarized in the results. While the Autoencoder achieved a high accuracy of 92.54%, indicating its ability to make correct predictions on the overall dataset, it is crucial to delve deeper into other metrics for a more comprehensive evaluation. Precision, recall, and F1-Score provide insights into the model's performance in detecting anomalies specifically. The Autoencoder demonstrates a precision of 68.80%, indicating that when it predicts an anomaly, it is correct approximately 68.80% of the time. However, the recall is 34.16%, implying that the model may not be capturing all actual anomalies, missing a significant portion. The F1-Score, which balances precision and recall, is 45.65%, suggesting a trade-off between correctly identifying anomalies and minimizing false positives. In summary, while the Autoencoder shows a high overall accuracy, its success in anomaly detection is tempered by a lower recall and F1-Score, indicating room for improvement in capturing true anomalies.

3.2.3 Final Models - Post-Hyperparameter-Tuning

After training and initial tuning of the autoencoder and C-LSTM models, a more detailed search through a finer hyperparamter space around the initial model parameters was undertaken to eke out additional performance. The search space and results of this tuning are provided in Appendix A. As previously discussed, the anomaly detection problem requires that the model balances the accuracy of positive predictions and the ability to identify all actual sparse anomalies. This is balance that depends on the exact goals of the model, (i.e. how important finding anomalies is vs. how damaging false-negatives are). The best model according to the F1-score, the harmonic mean of precision and recall, offers a simple metric by which we select the best model in our paper. The resulting best models according to F1-score are described in Table 3 and Table 4 below for the C-LSTM and Autoencoder models, respectively.

Table 3. C-LSTM Best Model Paramters

Hyperparameter	**Values**
Conv1 Output Channels	16
Kernel Size	3
LSTM Hidden Size	16
FC1 Output Features	4
Learning Rate	0.001
Optimizer	Adam
Criterion	BCELoss

Table 4. Autoencoder Best Model Paramters

Hyperparameter	**Values**
Encoder Layer Sizes	[128,64,32]
Autoencoder Learning Rate	0.01
Classifier Learning Rate	0.005
Activation	ReLU
Optimizer	Adam
Criterion	MSELoss

The overall performance of these models are described in Table 5 below.

Table 5. Model Performance Summary

Model	Accuracy	Precision	Recall	F1-Score
Random Forest	92.5%	61.2%	50.7%	55.5%
Autoencoder	90.3%	71.9%	41.8%	52.9%
C-LSTM	91.5%	69.9%	76.2%	64.6%

For the Autoencoder, the adjustment of layer sizes and learning rates was particularly impactful in improving the model's ability to effectively capture less certain anomalies, as indicated by the increase in recall over the baseline autoencoder model trained previously. This increase in recall was the primary source of improvement in the overall F1-score for the autoencoder modle. The C-LSTM model, benefitted from the fine-tuning of its convolutional and LSTM units, with similar improvement in the recall contributing most strongly to the overall improved performance in F1-score.

Note that the accuracy of these models does not improve over the baseline autoencoder and C-LSTM models. However, the uniquely sparse nature of the anomaly detection problem incentivies us to focus on these other metrics which more closely track the expected goals of the model. Fine-tuning of the model hyperparameters around the initially explored space resulted in a moderate increase in these more important dimensions of model performance of recall and F1-score.

4. Potential Areas for Improvement

The success of the autoencoder model hinges on its performance in identifying anomalies during the subsequent evaluation phase. The effectiveness of anomaly detection will be assessed in conjunction with the classifier's performance. The dataset exhibits a significant class imbalance, with the majority being normal network instances, while accurate anomaly detection remains crucial. Despite our attempt to address this through the window sliding method for balancing the dataset, an alternative strategy involves selectively omitting some normal data. This approach aims to enhance accuracy in identifying anomaly data points by mitigating the impact of the dominant normal class.

A crucial next step not explored by this paper is the application of our models to unseen web-traffic data. While we have demonstrated that the fundamental model architecture of both the C-LSTM and Autoencoder approaches is valid, we have not seen how these perform on new real-world data. The testing and model tuning performed in this paper has resulted in models that out-perform conventional machine learning approaches such as Random Forest. Models have also been optimized through a hyperparameter search validated against a held-out validaiton set of data. However, the next step to truly understand the effectiveness of these models is application to an un-seen test set of new web-traffic data with and without actual anomalies.

References

[1] R. Chalapathy and S. Chawla. Deep learning for anomaly detection: A survey. *arXiv*, Jan 2019. 1

[2] T. Hagemann and K. Katsarou. Reconstruction-based anomaly detection for the cloud: A comparison on the yahoo! webscope s5 dataset. In *Proceedings of the 2020 4th International Conference on Cloud and Big Data Computing*, pages 68–75, New York, NY, USA, Sep 2020. Association for Computing Machinery. 1

[3] T.-Y. Kim and S.-B. Cho. Web traffic anomaly detection using c-lstm neural networks. *Expert Systems with Applications*, 106:66–76, Sep 2018. 1, 4

[4] Yahoo Research. S5 - a labeled anomaly detection dataset. Available at: https://webscope.sandbox.yahoo.com/catalog.php?datatype=s&did=70. 2

[5] L. Wong, D. Liu, L. Berti-Equille, S. Alnegheimish, and K. Veeramachaneni. Aer: Auto-encoder with regression for time series anomaly detection. *arXiv*, Dec 2022. 1

Student Name	Contributed Aspects	Details
Lily Chebotarova	C-LSTM and exploratory data	Researched labelled anomaly detection datasets and found selected Yahoo dataset for project. Developed C-LSTM model implementation code $anomalydetection.ipynb$ and respective report sections for the model approach and training..
Raga Lasya Munagala	Pre-processing	Developed data pre-processing code $data\_preprocessing.ipynb$ and report section on -reprocessing.
Ligeng Peng	Autoencoder	Developed autoencoder with regression model implementation code $anomaly\_detection\_encoder.ipynb$ and report sections for Autoencoder model approach, training and experiments..
Brian Zhang	Hyperparamater Tuning, Overall Report and LaTeX	Performed hyperparameter tuning of both models $non - notebookfilesingitrepo$. Main author for abstract, introduction, and hyperparamter tuning write-ups in report. Responsible for LaTeX compilation and overall report format/organization. Created and maintained team git repo: https://github.com/brian-x-zhang/CS7643-Anamoly.

Table 6. Contributions of team members

A. Hyperparameter Tuning Results

type	encoder_layer_sizes	learning_rate	learning_rate_classifier	activation	optimizer	criterion	epochs	F1	Accuracy	Recall	Precision
autoencoder	[128, 64, 32]	0.010000	0.005000	ReLU()	Adam	MSELoss	200	0.528876	0.903117	0.418212	0.719181
autoencoder	[32, 16, 8]	0.001000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.514819	0.904556	0.389427	0.759309
autoencoder	[32, 16, 8]	0.001000	0.001000	ReLU()	Adam	MSELoss	200	0.501046	0.905600	0.364517	0.801095
autoencoder	[32, 16, 8]	0.001000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.478569	0.899302	0.355383	0.732459
autoencoder	[128, 64, 32]	0.001000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.447730	0.904556	0.297537	0.904121
autoencoder	[128, 64, 32]	0.010000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.436072	0.898726	0.301135	0.790123
autoencoder	[32, 16, 8]	0.010000	0.001000	ReLU()	Adam	MSELoss	200	0.432758	0.898294	0.298367	0.787436
autoencoder	[32, 16, 8]	0.010000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.423208	0.896603	0.291724	0.770468
autoencoder	[128, 64, 32]	0.010000	0.001000	ReLU()	Adam	MSELoss	200	0.406534	0.896711	0.272073	0.803761
autoencoder	[128, 64, 32]	0.010000	0.000500	ReLU()	Adam	MSELoss	200	0.332598	0.891096	0.208691	0.818675
autoencoder	[64, 32, 16]	0.001000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.313806	0.891240	0.191254	0.873578
autoencoder	[64, 32, 16]	0.010000	0.000500	ReLU()	Adam	MSELoss	200	0.309910	0.889729	0.190423	0.831923
autoencoder	[64, 32, 16]	0.010000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.304126	0.598143	0.675339	0.196252
autoencoder	[32, 16, 8]	0.001000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.260186	0.888253	0.151121	0.934932
autoencoder	[64, 32, 16]	0.010000	0.001000	ReLU()	Adam	MSELoss	200	0.259613	0.885662	0.154166	0.821534
autoencoder	[64, 32, 16]	0.010000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.245145	0.385842	0.766953	0.145888
autoencoder	[128, 64, 32]	0.010000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.238841	0.202188	0.962635	0.136333
autoencoder	[32, 16, 8]	0.050000	0.000500	ReLU()	Adam	MSELoss	200	0.230135	0.130030	1.000000	0.130030
autoencoder	[64, 32, 16]	0.050000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.230135	0.130030	1.000000	0.130030
autoencoder	[64, 32, 16]	0.050000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.230135	0.130030	1.000000	0.130030
autoencoder	[64, 32, 16]	0.050000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.230135	0.130030	1.000000	0.130030
autoencoder	[128, 64, 32]	0.050000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.230135	0.130030	1.000000	0.130030
autoencoder	[128, 64, 32]	0.010000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.227303	0.540236	0.520066	0.145433
autoencoder	[32, 16, 8]	0.010000	0.005000	ReLU()	Adam	MSELoss	200	0.202166	0.443245	0.542485	0.124231
autoencoder	[64, 32, 16]	0.001000	0.001000	ReLU()	Adam	MSELoss	200	0.198317	0.670913	0.313036	0.145130
autoencoder	[32, 16, 8]	0.010000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.189957	0.400849	0.540271	0.115237
autoencoder	[64, 32, 16]	0.001000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.168347	0.881235	0.092444	0.940845
autoencoder	[64, 32, 16]	0.001000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.151977	0.485100	0.354830	0.096696
autoencoder	[64, 32, 16]	0.010000	0.005000	ReLU()	Adam	MSELoss	200	0.147761	0.519254	0.320509	0.096012
autoencoder	[32, 16, 8]	0.001000	0.005000	ReLU()	Adam	MSELoss	200	0.132365	0.548478	0.264877	0.088227
autoencoder	[32, 16, 8]	0.010000	0.000500	ReLU()	Adam	MSELoss	200	0.056959	0.873677	0.029338	0.972477
autoencoder	[64, 32, 16]	0.001000	0.005000	ReLU()	Adam	MSELoss	200	0.028868	0.527820	0.053972	0.019703
autoencoder	[64, 32, 16]	0.001000	0.000500	ReLU()	Adam	MSELoss	200	0.015359	0.870798	0.007750	0.848485
autoencoder	[128, 64, 32]	0.001000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.002764	0.870150	0.001384	1.000000
autoencoder	[32, 16, 8]	0.001000	0.000500	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[32, 16, 8]	0.010000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[32, 16, 8]	0.050000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[32, 16, 8]	0.050000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[32, 16, 8]	0.050000	0.001000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[32, 16, 8]	0.050000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[32, 16, 8]	0.050000	0.005000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[64, 32, 16]	0.010000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[64, 32, 16]	0.050000	0.000500	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[64, 32, 16]	0.050000	0.001000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[64, 32, 16]	0.050000	0.005000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.001000	0.000500	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.001000	0.001000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.001000	0.005000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.001000	0.005000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.050000	0.000500	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.050000	0.000500	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.050000	0.001000	LeakyReLU(negative_slope=0.01)	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.050000	0.001000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000
autoencoder	[128, 64, 32]	0.050000	0.005000	ReLU()	Adam	MSELoss	200	0.000000	0.869970	0.000000	0.000000

type	conv1_out_channels	kernel_size	lstm_hidden_size	fc1_out_features	learning_rate	optimizer	criterion	epochs	F1	Accuracy	Recall	Precision
clstm	16	3	16	4	0.001000	Adam	BCELoss	200	0.609093	0.898510	0.608082	0.610108
clstm	16	3	16	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	3	16	16	0.001000	Adam	BCELoss	200	0.599922	0.889261	0.638528	0.565718
clstm	16	3	16	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	3	16	32	0.001000	Adam	BCELoss	200	0.596846	0.882243	0.670357	0.537864
clstm	16	3	16	32	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	3	32	4	0.001000	Adam	BCELoss	200	0.615864	0.897862	0.629671	0.602649
clstm	16	3	32	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	3	32	16	0.001000	Adam	BCELoss	200	0.615240	0.896962	0.633546	0.597962
clstm	16	3	32	16	0.010000	Adam	BCELoss	200	0.680727	0.913374	0.710213	0.653591
clstm	16	3	32	32	0.001000	Adam	BCELoss	200	0.624002	0.894911	0.670634	0.583434
clstm	16	3	32	32	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	5	16	4	0.001000	Adam	BCELoss	200	0.645031	0.905132	0.662884	0.628114
clstm	16	5	16	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	5	16	16	0.001000	Adam	BCELoss	200	0.599451	0.889765	0.634376	0.568171
clstm	16	5	16	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	5	16	32	0.001000	Adam	BCELoss	200	0.614854	0.895487	0.641572	0.590272
clstm	16	5	16	32	0.010000	Adam	BCELoss	200	0.699175	0.914741	0.761971	0.645941
clstm	16	5	32	4	0.001000	Adam	BCELoss	200	0.607578	0.893400	0.634653	0.582719
clstm	16	5	32	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	5	32	16	0.001000	Adam	BCELoss	200	0.582443	0.884618	0.618876	0.550062
clstm	16	5	32	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	16	5	32	32	0.001000	Adam	BCELoss	200	0.634360	0.900885	0.661223	0.609594
clstm	16	5	32	32	0.010000	Adam	BCELoss	200	0.674369	0.911754	0.702740	0.648200
clstm	32	3	16	4	0.001000	Adam	BCELoss	200	0.638962	0.900885	0.674509	0.606974
clstm	32	3	16	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	3	16	16	0.001000	Adam	BCELoss	200	0.650304	0.902577	0.696651	0.609738
clstm	32	3	16	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	3	16	32	0.001000	Adam	BCELoss	200	0.643100	0.897898	0.707445	0.589483
clstm	32	3	16	32	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	3	32	4	0.001000	Adam	BCELoss	200	0.626528	0.897754	0.659563	0.596645
clstm	32	3	32	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	3	32	16	0.001000	Adam	BCELoss	200	0.646150	0.900274	0.700249	0.599810
clstm	32	3	32	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	3	32	32	0.001000	Adam	BCELoss	200	0.622211	0.892752	0.679214	0.574035
clstm	32	3	32	32	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	5	16	4	0.001000	Adam	BCELoss	200	0.636823	0.901605	0.663438	0.612261
clstm	32	5	16	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	5	16	16	0.001000	Adam	BCELoss	200	0.635312	0.896747	0.691669	0.587447
clstm	32	5	16	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	5	16	32	0.001000	Adam	BCELoss	200	0.627376	0.894191	0.685026	0.578677
clstm	32	5	16	32	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	5	32	4	0.001000	Adam	BCELoss	200	0.628121	0.898150	0.661500	0.597948
clstm	32	5	32	4	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	5	32	16	0.001000	Adam	BCELoss	200	0.622782	0.894407	0.670357	0.581513
clstm	32	5	32	16	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000
clstm	32	5	32	32	0.001000	Adam	BCELoss	200	0.648635	0.903225	0.686964	0.614356
clstm	32	5	32	32	0.010000	Adam	BCELoss	200	0.000000	0.869970	0.000000	0.000000

B. Loss Curves

Loss curves for the C-LSTM and autoenocder models prior to hyperparameter tuning are shown below.

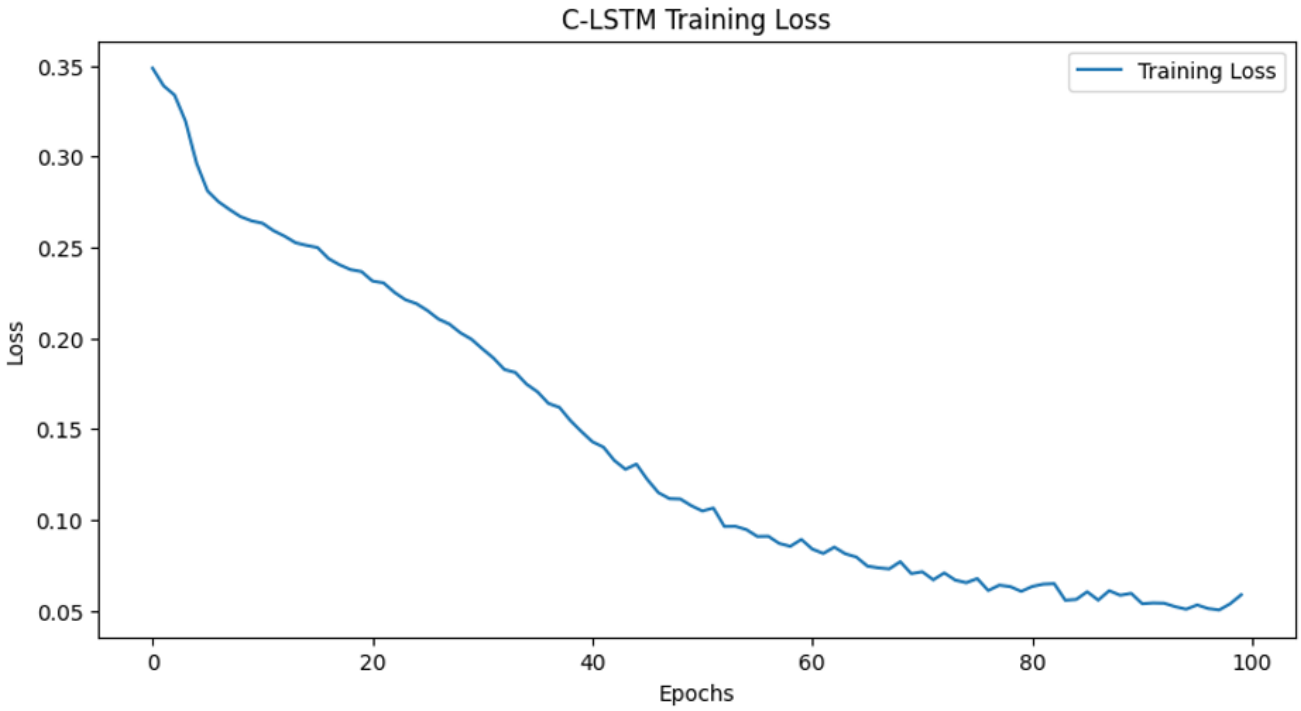

Figure 3. Training Loss Over Time (C-LSTM)

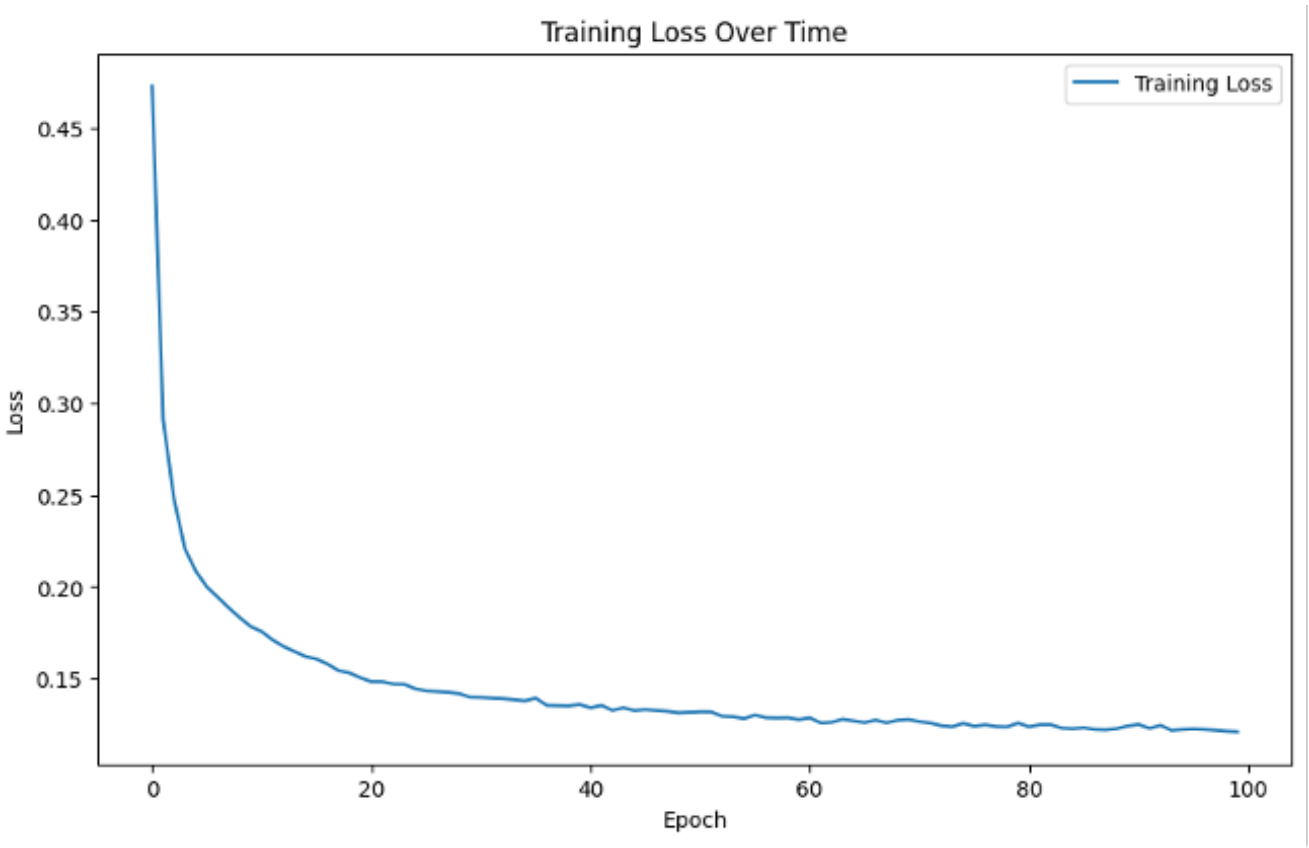

Figure 4. Training Loss Over Time (Autoencoder)

C. C-LSTM Model Architecture

Type	Filter	Kernel size	Stride	# Param
Convolution	64	5	1	385
Activation (tanh)				0
Pooling		2	2	0
Convolution	64	5	1	20544
Activation (tanh)				0
Pooling		2	2	0
LSTM (64)				262656
Dense (32)				2080
Activation (tanh)				0
Dense (2)				66
Softmax				0
Total parameters	285730			