National College of Ireland

Configuration Manual

MSc Research Project

MSc in Data Analytics

Sharik Arif Sayyad

Student ID: X22210253

School of Computing

National College of Ireland

Supervisor:Teerath Kumar Menghwar

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……Sharik Arif Sayyad……		
Student ID:	X22210253……		
Programme:	M.Sc in Data Analytics	**Year:**	2023-2024.
Module:	…M.Sc Research Project ……		
Lecturer:	…Mr teerath Kumar Menghwar ……		
Submission Due Date:	…16-09-2024……		
Project Title:	Variational AutoEncoder(VAE) for Anomaly Detection in Network Traffic ……		
Word Count:	……853…… **Page Count:** ……7……		

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	……Sharik Sayyad……
Date:	……10 – 09-2024 ……

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on the computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Variational AutoEncoder(VAE) for Anomaly Detection in Network Traffic

SHARIK SAYYAD
Student ID:x22210253

1. Introduction

This project identifies network intrusions using the latest machine learning techniques in the rapidly changing cyber security domain. We have used several deep learning models, including Autoencoders and Variational Autoencoders, to build an effective Intrusion Detection System. By training on robust datasets like CICIDS 2017 and KDD99, these models are equipped to efficiently flag anomalous activities in network traffic.

The project is more than just modeling; it's about how these models handle the complexities of network data to identify known and emerging threats in real time. Ready? Let's set up your environment.

2. Setting Up the Environment

2.1 Connecting to Google Colab

Google Colab allows you to run Python notebooks right in a cloud environment, which is really similar to the setup of certain machine learning projects. The only difference is the setting up of the environment.

Open Google Colab:

Open your web browser and go to Google Colab.
Sign in with your Google Account.

Create New Notebook:

Open a new Python notebook for your project by the menu commands File > New Notebook.

Set Up the Runtime:

Go to Runtime > Change runtime type.
Select Hardware accelerator from the drop-down menu, and then click on GPU. It will make sure that your deep learning models train faster.

2.2 Installing Necessary Libraries

The latter would need a few basic Python libraries to carry on the process for model building, fitting, and other visitors. Run the following commands lines inside a code cell in Colab to install the libraries:

```
!pip install numpy pandas matplotlib scikit-learn tensorflow imbalanced-learn seaborn
```

This command installs:

numpy: For numerical operations.
pandas: To manipulate and analyze data.
matplotlib & seaborn: To be used for visualization purposes.
scikit-learn: Data preprocessing and model evaluation.
TensorFlow: Develop and train deep learning models.
imbalanced-learn: For dealing with imbalanced datasets.

3. Add and Check Data

The detailed information for the datasets used for the project includes CICIDS 2017 and KDD99; you can get the simple download from the following links:

CICIDS 2017 Dataset: https://paperswithcode.com/dataset/cicids2017
KDD99 Dataset: http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html

After downloading, you can upload the downloaded datasets in your Colab environment by clicking on the file upload feature.

If the data would be fed into a machine learning model, then as a quality preparation of the data, a preprocessing step would have to be conducted. The steps involved are evident in this process:

Treating Missing Values and Infinities:
Replace infinity values with the mean of the pertaining feature.

Feature Scaling:
Use StandardScaler to scale the features so that they become comparable.

SMOTE for Imbalanced Data:
Apply the SMOTE technique to balance the dataset.

Example pre-processing code:

```
# Apply SMOTE to balance the datasets
smote = SMOTE()
X_smote1, y_smote1 = smote.fit_resample(processed_data1, labels1)
X_smote2, y_smote2 = smote.fit_resample(processed_data2, labels2)

# Split the data into training and testing sets
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_smote1, y_smote1, test_size=0.2, random_state=42)
X_train2, X_test2, y_train2, y_test2 = train_test_split(X_smote2, y_smote2, test_size=0.2, random_state=42)
```

Fig 1. Sample Pre processing code

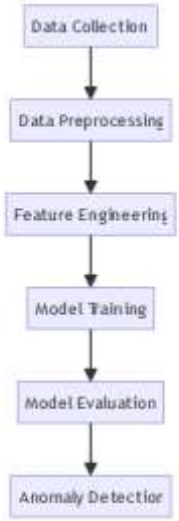

Fig 2. Data Flow Diagram

4. Developing and Training the Model

The following model architectures were put to use in this project:

Autoencoder with Integrated Classifier:

The autoencoder learns to compress and reconstruct input data. Then, enhance this with an integrated classifier layer for the prediction of anomalies.

Sample code:

from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model

```
# Build autoencoder classifier model
def build_autoencoder_classifier(input_dim, encoding_dim=32, num_classes=1):
    input_layer = Input(shape=(input_dim,))
    encoded = Dense(encoding_dim, activation='relu')(input_layer)
    decoded = Dense(input_dim, activation='sigmoid', name='decoded')(encoded)

    classifier = ClassifierLayer(num_outputs=num_classes, name='classifier_layer')(encoded)
    autoencoder = Model(inputs=input_layer, outputs=[decoded, classifier])
    autoencoder.compile(optimizer='adam',
                        loss={'decoded': MeanSquaredError(), 'classifier_layer': hinge_loss},
                        metrics={'decoded': 'mse', 'classifier_layer': 'accuracy'})

    return autoencoder
```

Fig 3. Sample Code

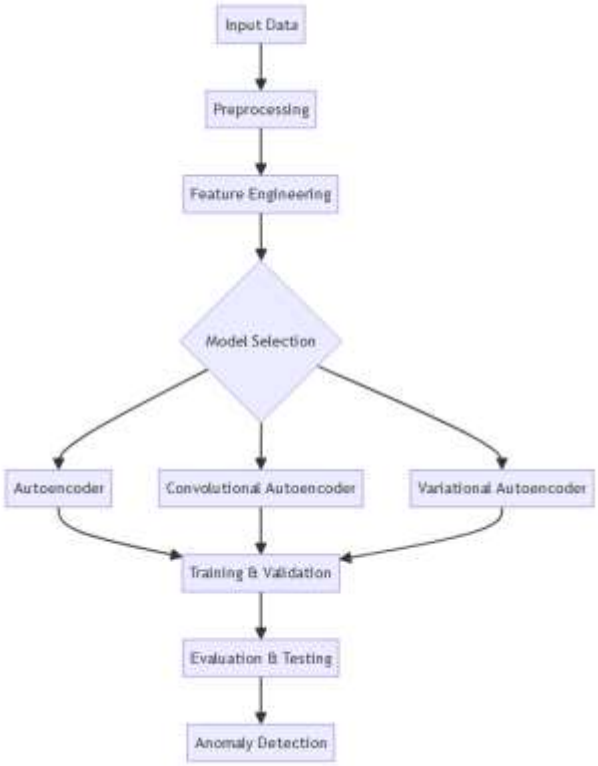

Fig 4. System Architecture Diagram

Convolutional Autoencoder:

It contains convolutional layers capable of capturing data's spatial hierarchies, therefore proving very helpful in network intrusion identification.

Variational Autoencoder (VAE):

It is a generative model that probabilistically maps input data into the latent space.

Conditional Variational Autoencoder (CVAE):

It extends VAE to be able to condition the generative process with additional labels, further improving the model with a fine reconstruction of the various classes.

4.1 Model Training

For each one of the models, the following procedures train these on the pre-trained data. Here is an example of how to train an autoencoder with a classifier:

```
model = create_autoencoder_classifier(input_dim=X_train.shape[1])
history = model.fit(X_train, [X_train, y_train], epochs=15, batch_size=256, validation_data=(X_test, [X_test, y_test]))
```

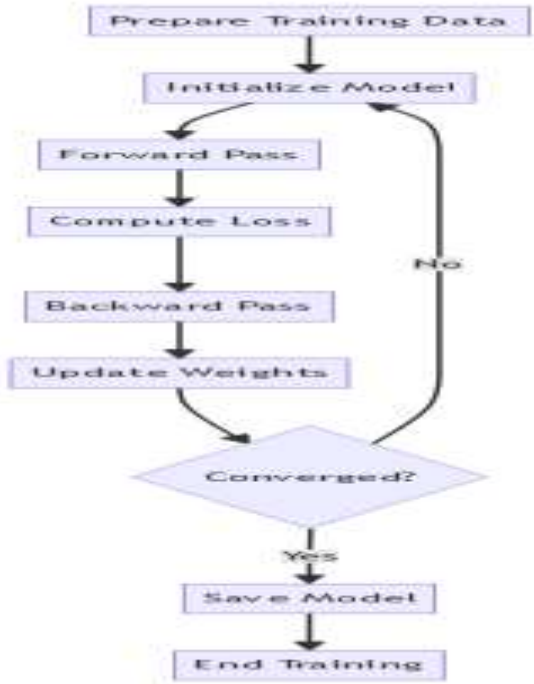

Fig 5. Model Training Process Diagram

4.2 Model Validation

The models' evaluation metrics are accuracy, precision, recall, F1-score, and ROC-AUC. These give results illustrated either in a confusion matrix or ROC–AUC curve.

Sample Evaluation Code:

from sklearn.metrics import confusion_matrix, accuracy_score, roc_curve, auc
import matplotlib.pyplot as plt

```
# Evaluate the models on the test sets
def evaluate_model(model, X, y_true, threshold):
    decoded_output, classifier_output = model.predict(X)
    y_pred = (classifier_output > threshold).astype(int).flatten()

    cm = confusion_matrix(y_true, y_pred)
    accuracy = accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
```

Fig 6. Sample Evaluation Code

5. Troubleshooting and Hints

Common Issues and Solutions:

File Path Errors:

The file paths should be properly specified. Use an absolute path to avoid any kind of errors in Colab.

Memory Issues:

If you run into memory problems, try decreasing your batch size, or just work on a smaller subsample of your data.

Model Overfitting:

Overfitting could be controlled by techniques like dropout, early stopping, and regularization.

Additional Tips:

Check Status of Training:

Keep checking the training and validation loss throughout the training process for signs of overfitting or underfitting.

Experiment with Hyperparameters:

Try with varying hyperparameters—such as learning rate, batch size, and the number of epochs—to fine-tune the model performance.

6. Conclusion

This is a configuration guide for setting up, running, and evaluating an Intrusion Detection System using deep learning models. This will take you through the steps so that you can replicate the results of the project and make inferences around which models worked best with network intrusions detection. The project results assuredly portray the potential in employing machine-learning techniques in improving the security of network systems.

References

Google. (n.d.). Google Colaboratory. Retrieved August 11, 2024, from https://colab.research.google.com/

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... & Oliphant, T. E. (2020). Array programming with NumPy. Nature, 585(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

McKinney, W. (2010). Data Structures for Statistical Computing in Python. In S. van der Walt & J. Millman (Eds.), Proceedings of the 9th Python in Science Conference (pp. 51-56). https://doi.org/10.25080/Majora-92bf1922-00a

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. Computing in Science & Engineering, 9(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

Waskom, M. (2021). Seaborn: Statistical Data Visualization. Journal of Open Source Software, 6(60), 3021. https://doi.org/10.21105/joss.03021

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... & Duchesnay, É. (2011). Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 12, 2825-2830. Retrieved from https://scikit-learn.org/

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). TensorFlow: A System for Large-Scale Machine Learning. In 12th USENIX Symposium on

Operating Systems Design and Implementation (OSDI 16) (pp. 265-283). USENIX Association. Retrieved from https://www.tensorflow.org/

Lemaître, G., Nogueira, F., & Aridas, C. K. (2017). Imbalanced-learn: A Python Toolbox to Tackle the Curse of Imbalanced Datasets in Machine Learning. Journal of Machine Learning Research, 18(1), 559-563. Retrieved from https://imbalanced-learn.org/

Sharafaldin, I., Lashkari, A. H., & Ghorbani, A. A. (2018). Toward Generating a New Intrusion Detection Dataset and Intrusion Traffic Characterization. In Proceedings of the 4th International Conference on Information Systems Security and Privacy (ICISSP) (pp. 108-116). https://doi.org/10.5220/0006639801080116

Hettich, S., & Bay, S. D. (1999). The UCI KDD Archive. University of California, Irvine, School of Information and Computer Sciences. Retrieved from http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html