**Enhancing Smart Grid Security: Anomaly and Novelty Detection for Communication Threats**

**Overview:**

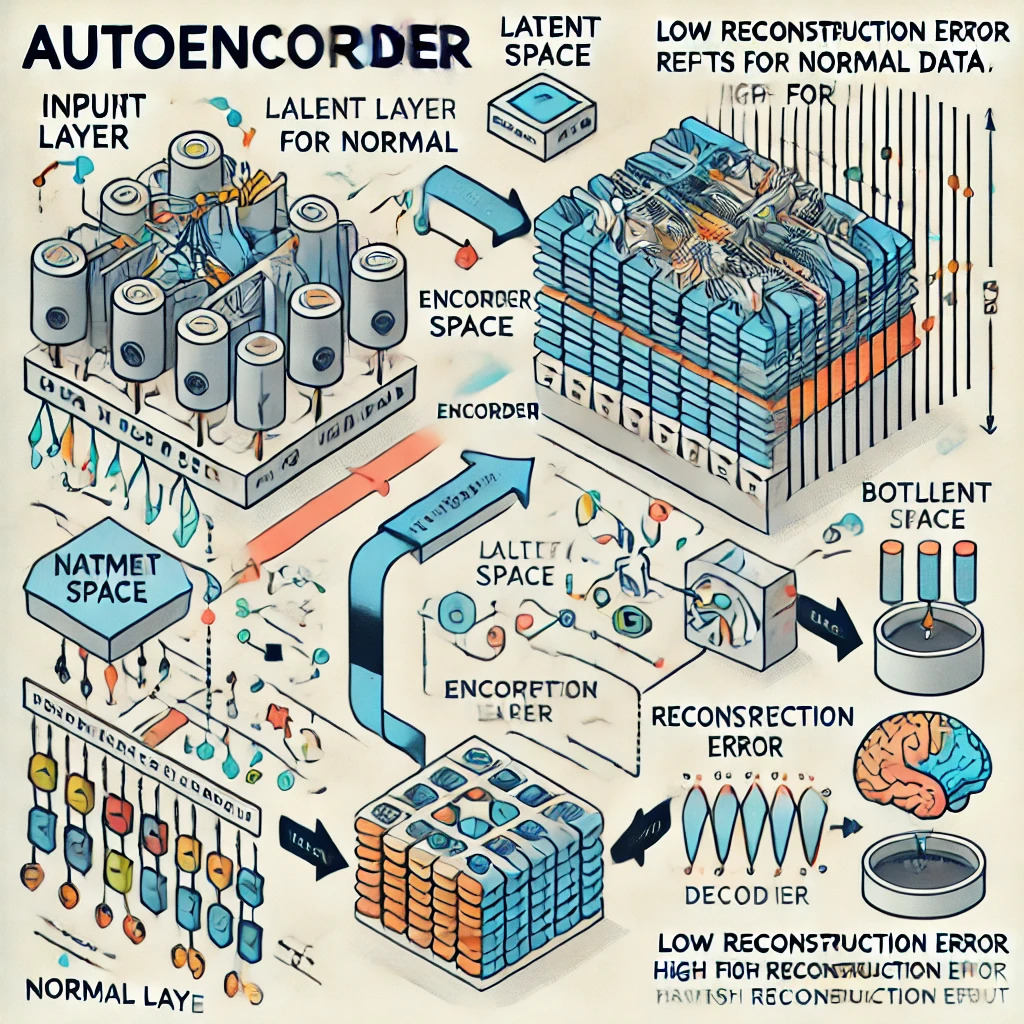
This project focuses on using machine learning to detect anomalies and novel patterns in data. Two approaches are employed:

* **Local Outlier Factor (LOF)** for anomaly detection, and
* **Autoencoder** (a type of neural network) for novelty detection.

Anomalies refer to unusual observations, while novelty detection identifies data patterns previously unseen by the model. We’ll use a dataset containing both normal and anomalous samples and aim to detect these with high accuracy.

**How Autoencoders Work for Anomaly Detection?**

1. **Training Phase**:
   * During training, the Autoencoder learns to reconstruct only the normal data (FLAG=0). It minimizes reconstruction error by adjusting weights to capture the features of normal data.
2. **Inference Phase**:
   * When presented with anomalous data (FLAG=1), the model is unable to reconstruct it well, leading to a high reconstruction error.
   * By setting a threshold, we classify samples with reconstruction errors above this threshold as anomalies.



## **2. Code Walkthrough and Explanation**

### **Step 1: Load and Clean Data**

# Load the dataset

file\_path = '/content/drive/MyDrive/archive/data set.csv'

data = pd.read\_csv(file\_path)

# Overview of the dataset

print("Data shape:", data.shape)

print("First few rows of the data:\n", data.head())

* **Description**: Here, the dataset is loaded from the specified path. It gives an initial shape of the data and displays the first few rows for an overview.

### **Step 2: Data Preprocessing**

# Convert non-numeric columns (e.g., dates) to numeric if possible

data = data.apply(pd.to\_numeric, errors='coerce')

# Drop columns with a high percentage of NaN values

nan\_threshold = 0.5

data = data.loc[:, data.isnull().mean() < nan\_threshold]

# Drop rows with NaN values

data = data.dropna()

print("Data shape after cleaning:", data.shape)

* **Purpose**: Ensures data consistency by converting non-numeric columns to numeric, dropping columns with too many missing values, and removing any remaining rows with NaN values.

### **Step 3: Feature and Target Definition**

# Check if the FLAG column exists

if 'FLAG' not in data.columns:

print("Error: 'FLAG' column not found in the dataset.")

else:

# Define features and target variable

X = data.drop(columns=['FLAG'])

y = data['FLAG']

* **Explanation**: FLAG column is assumed to be the target variable (indicating normal vs. anomalous data points). If FLAG is absent, an error message is displayed.

### **Step 4: Data Splitting and Scaling**

# Split into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Scale the data

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

* **Description**: The dataset is split into training and testing sets. Scaling is performed to normalize the data, which improves model performance.

### **5. Anomaly Detection with Local Outlier Factor (LOF)**

lof = LocalOutlierFactor(n\_neighbors=20, contamination=0.1, novelty=True)

lof.fit(X\_train)

# Predict on test set

y\_pred\_lof = lof.predict(X\_test)

y\_pred\_lof = np.where(y\_pred\_lof == -1, 1, 0)

* **Explanation**: The LOF model identifies data points that deviate from the norm based on their relative densities.
* **Working**: LOF detects anomalies by comparing local density deviations, labeling points with lower densities than their neighbors as anomalies.

**Evaluation and Results**

print("LOF Classification Report:\n", classification\_report(y\_test, y\_pred\_lof))

print("LOF Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_lof))

* **Confusion Matrix and Report**: Provides precision, recall, and F1 scores for LOF’s predictions, helping assess its accuracy.

### **6. Novelty Detection with Autoencoder**

#### **Building and Training the Autoencoder**

Autoencoders are neural networks designed to learn an efficient representation of the input data. By training on normal data, they learn to reconstruct it well, but produce higher reconstruction errors for anomalous data.

model = Sequential([

Dense(128, activation='relu', input\_shape=(X\_train\_normal.shape[1],)),

Dense(64, activation='relu'),

Dense(32, activation='relu'),

Dense(64, activation='relu'),

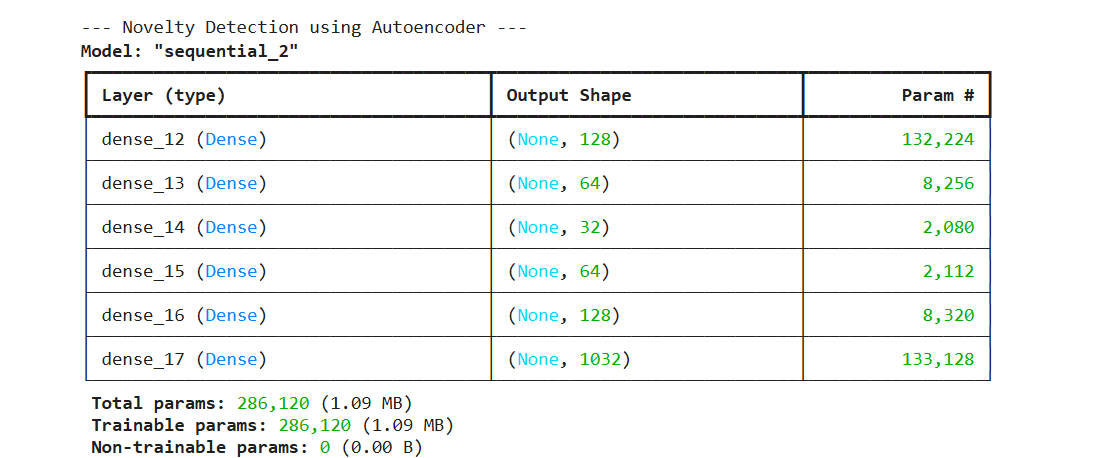
Dense(128, activation='relu'),

Dense(X\_train\_normal.shape[1], activation='sigmoid')

])

model.compile(optimizer='adam', loss='mse')

model.summary()



* **Structure**: The encoder compresses the input to a smaller latent representation, and the decoder reconstructs it back. The neural network minimizes reconstruction errors, making it effective for novelty detection.

#### **Training and Validation**

history = model.fit(X\_train\_normal, X\_train\_normal, epochs=50, batch\_size=256, validation\_split=0.2, verbose=1)

* **Explanation**: Only normal data (X\_train\_normal) is used to train the Autoencoder, allowing it to learn standard patterns.

#### **Setting a Threshold for Novelty Detection**

# Calculate reconstruction error and set threshold

reconstructions = model.predict(X\_train\_normal)

train\_loss = np.mean(np.square(reconstructions - X\_train\_normal), axis=1)

threshold = np.percentile(train\_loss, 95)

print("Reconstruction error threshold:", threshold)

* **Threshold**: The 95th percentile of training loss errors is chosen as the threshold. Any test sample exceeding this error is flagged as an anomaly.

#### **Testing the Autoencoder**

# Test the model on test data

reconstructions = model.predict(X\_test)

test\_loss = np.mean(np.square(reconstructions - X\_test), axis=1)

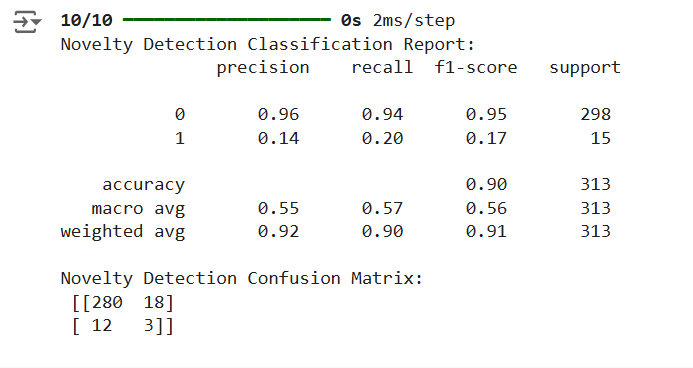
y\_pred\_novelty = np.where(test\_loss > threshold, 1, 0)

* **Detection**: After calculating the reconstruction error on X\_test, the model classifies samples with errors above the threshold as anomalies.

**Evaluation**

print("Novelty Detection Classification Report:\n", classification\_report(y\_test, y\_pred\_novelty))

print("Novelty Detection Confusion Matrix:\n", confusion\_matrix(y\_test, y\_pred\_novelty))



### **Training Loss Visualization**

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.legend()

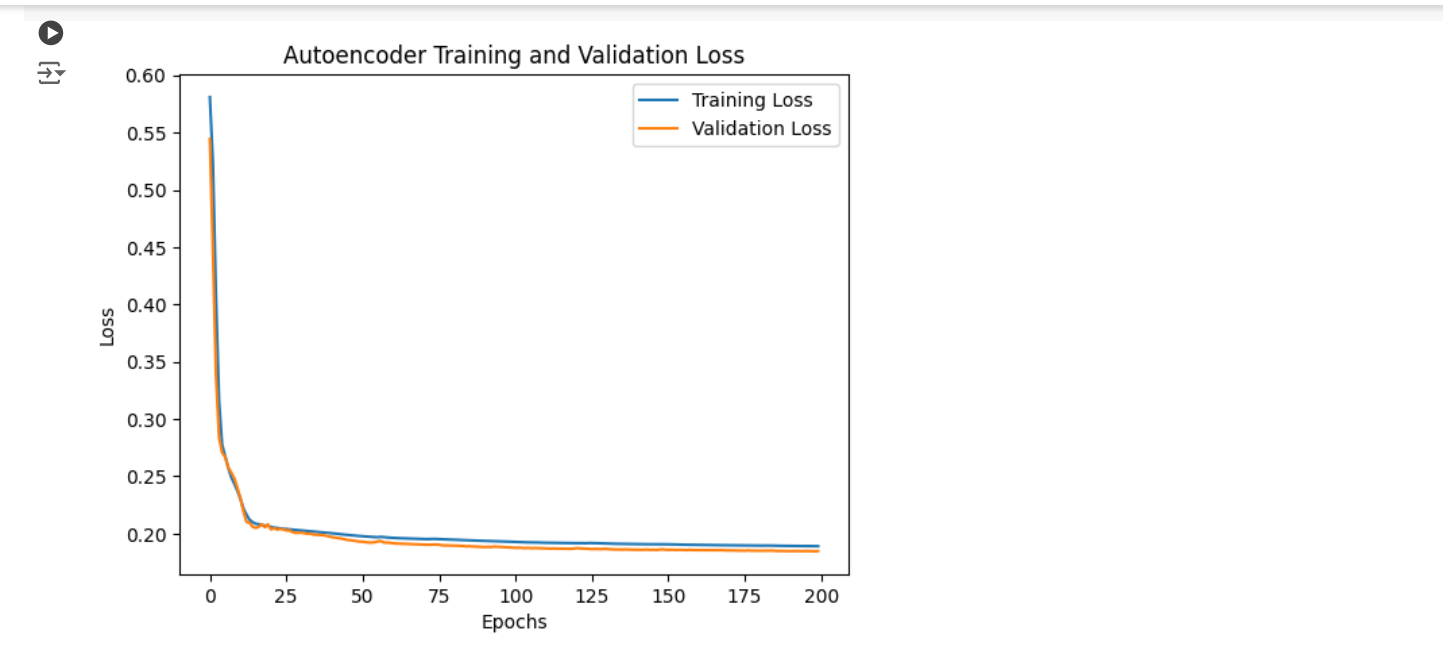
plt.title("Autoencoder Training and Validation Loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.show()

* **Visualization**: Shows the learning progress of the Autoencoder, helping assess if it’s learning effectively and not overfitting.



## **7. Summary of Results**

* **Local Outlier Factor (LOF)**: Efficient for basic anomaly detection.
* **Autoencoder**: Effective for novelty detection as it reconstructs known patterns accurately and detects unknown patterns.