Tesnsorflow & Keras INFO-6146-(01)-24F Group 9 – Project 2

Anomaly Detection Using Autoencoder with Reconstruction Error Analysis

GROUP MEMBER: AHKAR HTET
SU MON HPAN
THAI KIEN NGUYEN
THIRI THAN MYINT

1001 Fanshawe College Boulevard London, Ontario N5Y 5R6

Contents

1.Objective

This project develops for image preprocessing and anomaly detection using autoencoders, focusing on the Caltech-101 dataset. It involves loading and normalizing images, building a convolutional autoencoder for image reconstruction, and identifying anomalies based on reconstruction errors with optimized thresholds. The approach emphasizes effective visualization, robust model evaluation, and actionable recommendations for improving dataset design to enhance detection accuracy.

2.Caltech101 Dataset Description:

Overview:

The Caltech-101 dataset, which consists of diverse images across 101 categories, was used to train and evaluate the model. This dataset is well-suited for learning robust feature representations

Key Features:

- **Number of Categories:** 101 object classes + 1 background class.
- **Total Images:** Approximately 9,146 images.
- **Image Dimensions:** Images vary in size but are generally medium-resolution (e.g., 300 x 200 pixels on average).
- **File Formats:** Images are in JPEG format.

Classes:

- Each class contains images of a single object type (e.g., airplanes, cars, flowers, faces).
- Categories include:
    - Animals (e.g., "elephant," "leopard").
    - Objects (e.g., "accordion," "camera").
    - Miscellaneous (e.g., "faces," "background").

- **Source:** Caltech101 (via TensorFlow Datasets)

3.Approach

3.1 Loading and preprocess dataset

- The dataset is loaded using TensorFlow Datasets (tfds)
- Split the Caltech-101 dataset into training, validation, and testing subsets, with splits of 70%, 15%, and 15%, respectively.

3.2 Building Unsupervised Autoencoder Model

- Refer to the detail model architecture in selection 5

3.3 Anomaly Detection

- Refer to the detail implementation in selection 6

3.4 Evaluation and Visualization

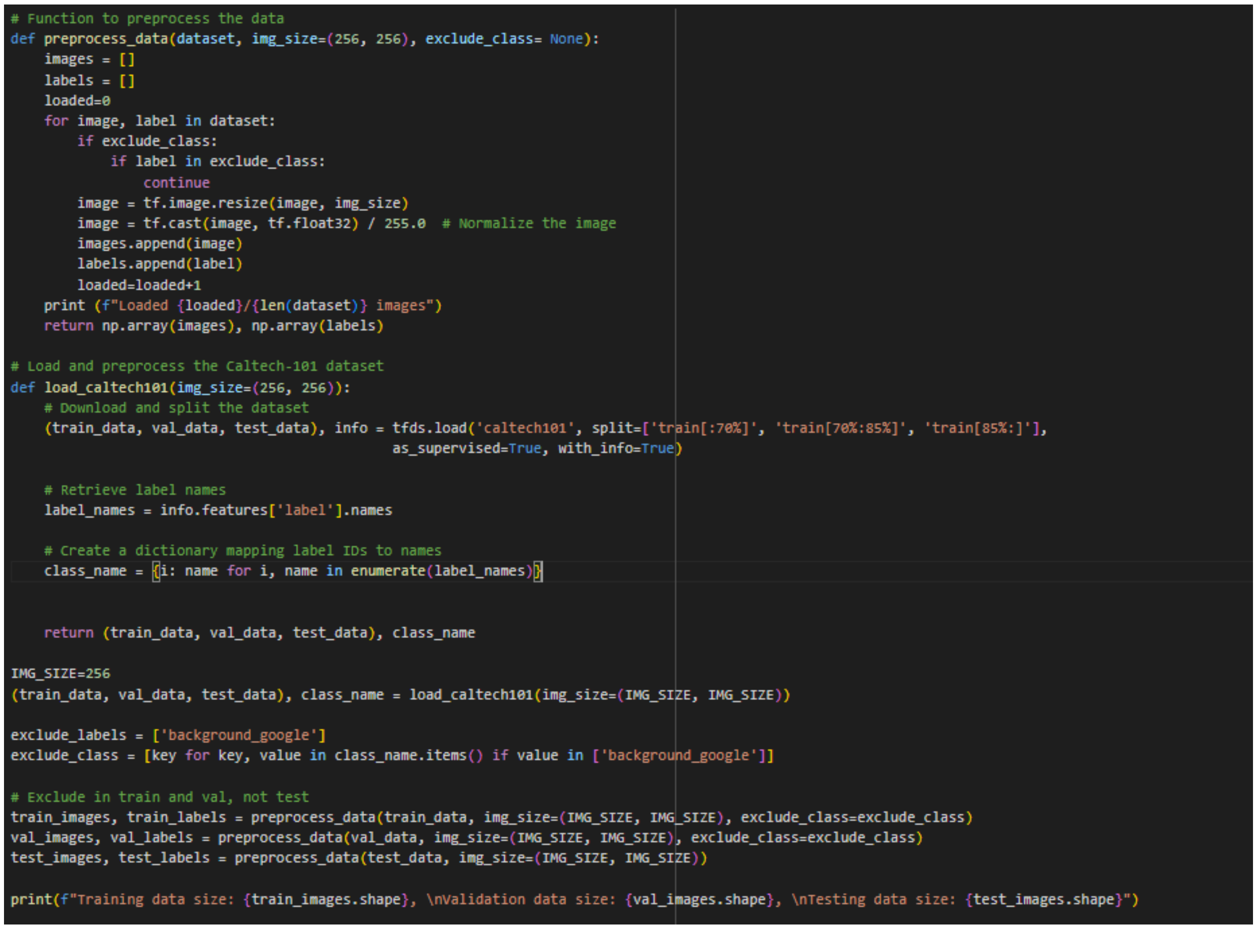
- Visualize the anomaly detection model’s performance using ROC Cure, Confusion Matrix and dimensionality reduction (PCA scatterplot)

3.5 Model Refinement and Optimization

- Refer to the detail experiment in selection 8

4.Loading and preprocess dataset

Load the Caltech-101 dataset using TensorFlow Datasets (tfds) and preprocessed for image classification tasks. The preprocessing involves resizing images to a specified size (default: 256x256) and normalizing pixel values to the range [0, 1]. Then, download and split the dataset into training, validation, and testing subsets with proportions of 70%, 15%, and 15%, respectively. Additionally, specific classes (e.g., background_google) can be excluded from the training and validation datasets while retaining them in the test dataset. The preprocessed data is then stored as NumPy arrays for further use.



```python
# Function to preprocess the data
def preprocess_data(dataset, img_size=(256, 256), exclude_class= None):
    images = []
    labels = []
    loaded=0
    for image, label in dataset:
        if exclude_class:
            if label in exclude_class:
                continue
        image = tf.image.resize(image, img_size)
        image = tf.cast(image, tf.float32) / 255.0  # Normalize the image
        images.append(image)
        labels.append(label)
        loaded=loaded+1
    print (f"Loaded {loaded}/{len(dataset)} images")
    return np.array(images), np.array(labels)

# Load and preprocess the Caltech-101 dataset
def load_caltech101(img_size=(256, 256)):
    # Download and split the dataset
    (train_data, val_data, test_data), info = tfds.load('caltech101', split=['train[:70%]', 'train[70%:85%]', 'train[85%:]'],
                                                      as_supervised=True, with_info=True)

    # Retrieve label names
    label_names = info.features['label'].names

    # Create a dictionary mapping label IDs to names
    class_name = {i: name for i, name in enumerate(label_names)}


    return (train_data, val_data, test_data), class_name

IMG_SIZE=256
(train_data, val_data, test_data), class_name = load_caltech101(img_size=(IMG_SIZE, IMG_SIZE))

exclude_labels = ['background_google']
exclude_class = [key for key, value in class_name.items() if value in ['background_google']]

# Exclude in train and val, not test
train_images, train_labels = preprocess_data(train_data, img_size=(IMG_SIZE, IMG_SIZE), exclude_class=exclude_class)
val_images, val_labels = preprocess_data(val_data, img_size=(IMG_SIZE, IMG_SIZE), exclude_class=exclude_class)
test_images, test_labels = preprocess_data(test_data, img_size=(IMG_SIZE, IMG_SIZE))

print(f"Training data size: {train_images.shape}, \nValidation data size: {val_images.shape}, \nTesting data size: {test_images.shape}")
```

Preprocessing Images:

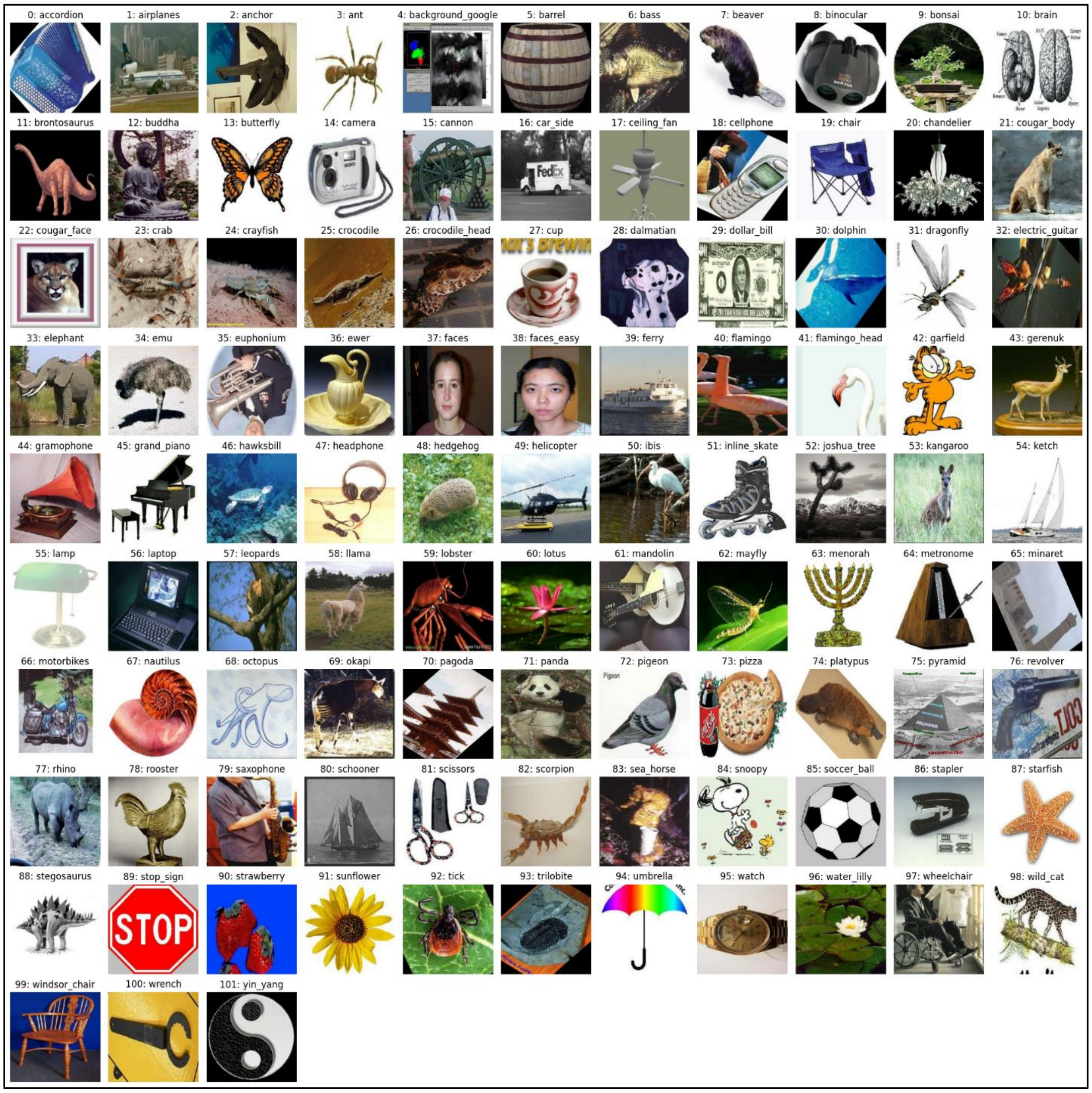
Each image is resized to a consistent size (default: 256x256) to ensure uniformity in input dimensions. The pixel values are normalized by dividing them by 255.0, scaling them to a range between 0 and 1, which helps the model learn more effectively.

Filtering by Class:

The function iterates over the dataset and checks the label of each data point. If the label matches the specified class (class_to_get), the image and its label are added to the output list.

Merging into Test Dataset:

Once the data for the target class is extracted from both the **training and validation** datasets, it is combined with the test dataset. This is done by concatenating the images and labels of the target class to the existing test data.



Overview of Class Labels and Categories

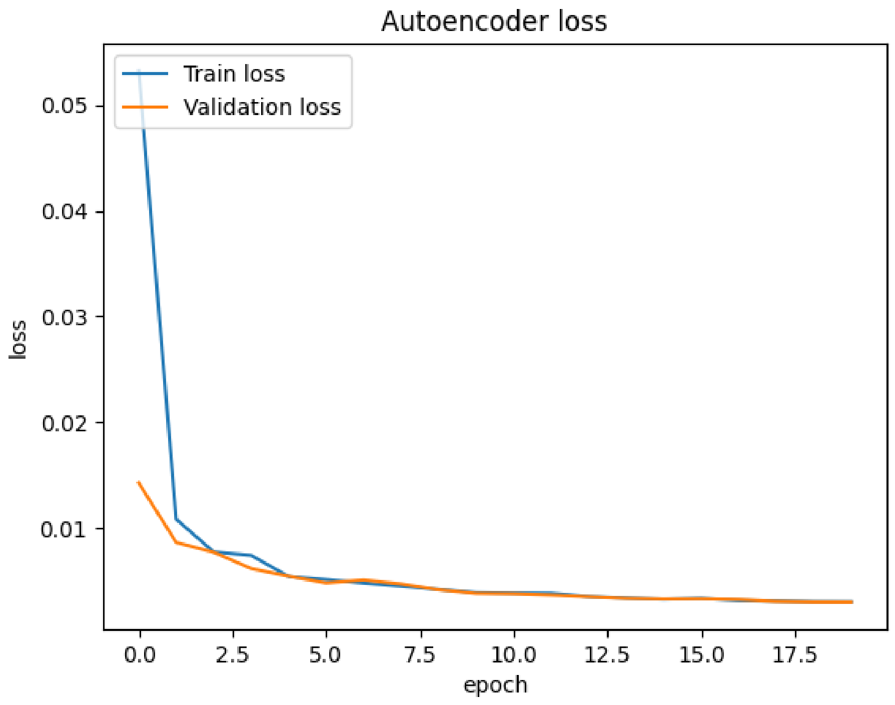
Background Google Class Samples

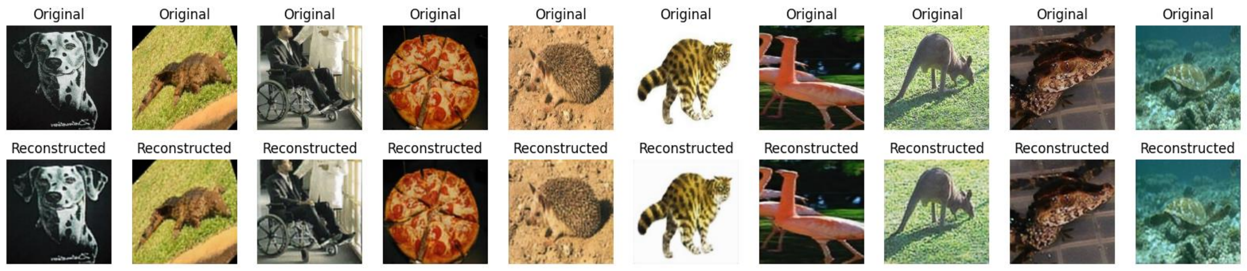
5. Autoencoder Model Architecture

The Autoencoder component serves as a key element in image reconstruction tasks, offering functionality across various stages of the process. The **create_autoencoder** function defines and compiles a convolutional autoencoder, which leverages convolutional layers for encoding and decoding images. This model requires parameters such as the input_shape (e.g., (256, 256, 3) for RGB images) and the encoding_dim, representing the dimensionality of the encoding layer.

After creating the model then **train**_autoencoder function trains the model on the provided training data while evaluating its performance on a validation set.

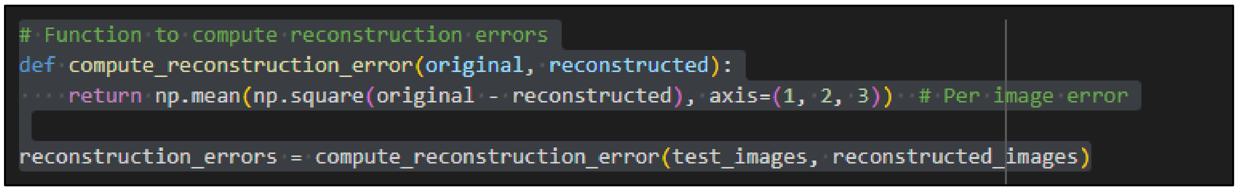
The training process involves fitting the autoencoder to the training dataset (train_images), with validation performed on a separate validation set (val_images). The model trains for up to **20 epochs** with a batch size of 64. After training, the function evaluates the best model on a test dataset (test_images) and predicts reconstructed versions of these test images for further analysis. The below image show the loss during training and validation.





Comparison of Original and Reconstructed Image

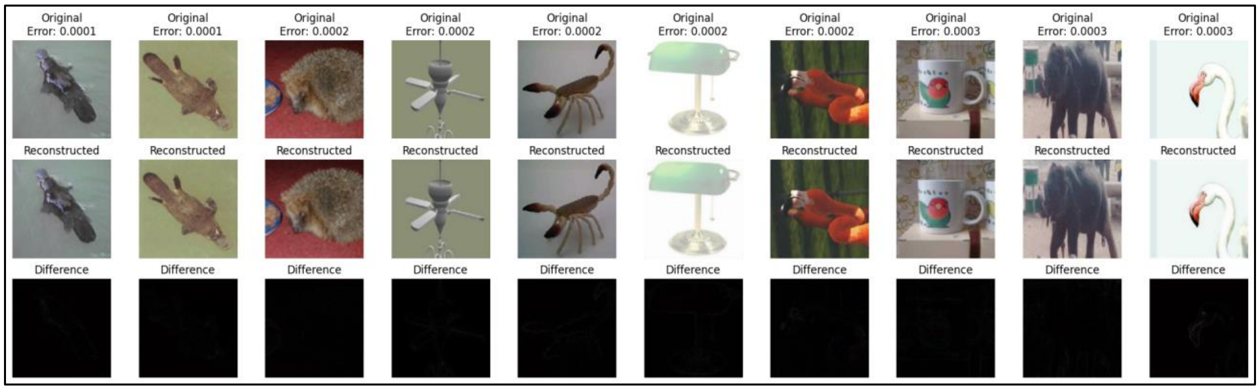
Lastly, the compute_reconstruction_error function calculates the mean squared error (MSE) between the original and reconstructed images to assess the model's accuracy. It takes arrays of original and reconstructed images as input and returns the per-image reconstruction errors, which are crucial for identifying anomalies or evaluating the model's effectiveness. This comprehensive architecture ensures a robust pipeline for training, evaluation, and visualization of the autoencoder's performance.



```
# Function to compute reconstruction errors
def compute_reconstruction_error(original, reconstructed):
    return np.mean(np.square(original - reconstructed), axis=(1, 2, 3))  # Per image error

reconstruction_errors = compute_reconstruction_error(test_images, reconstructed_images)
```

The image below displays the original, reconstructed, and error-difference images, arranged in descending order of reconstruction error. This highlights anomalies or instances with significant reconstruction errors.



Comparison of Original, Reconstructed, and Error Difference Images

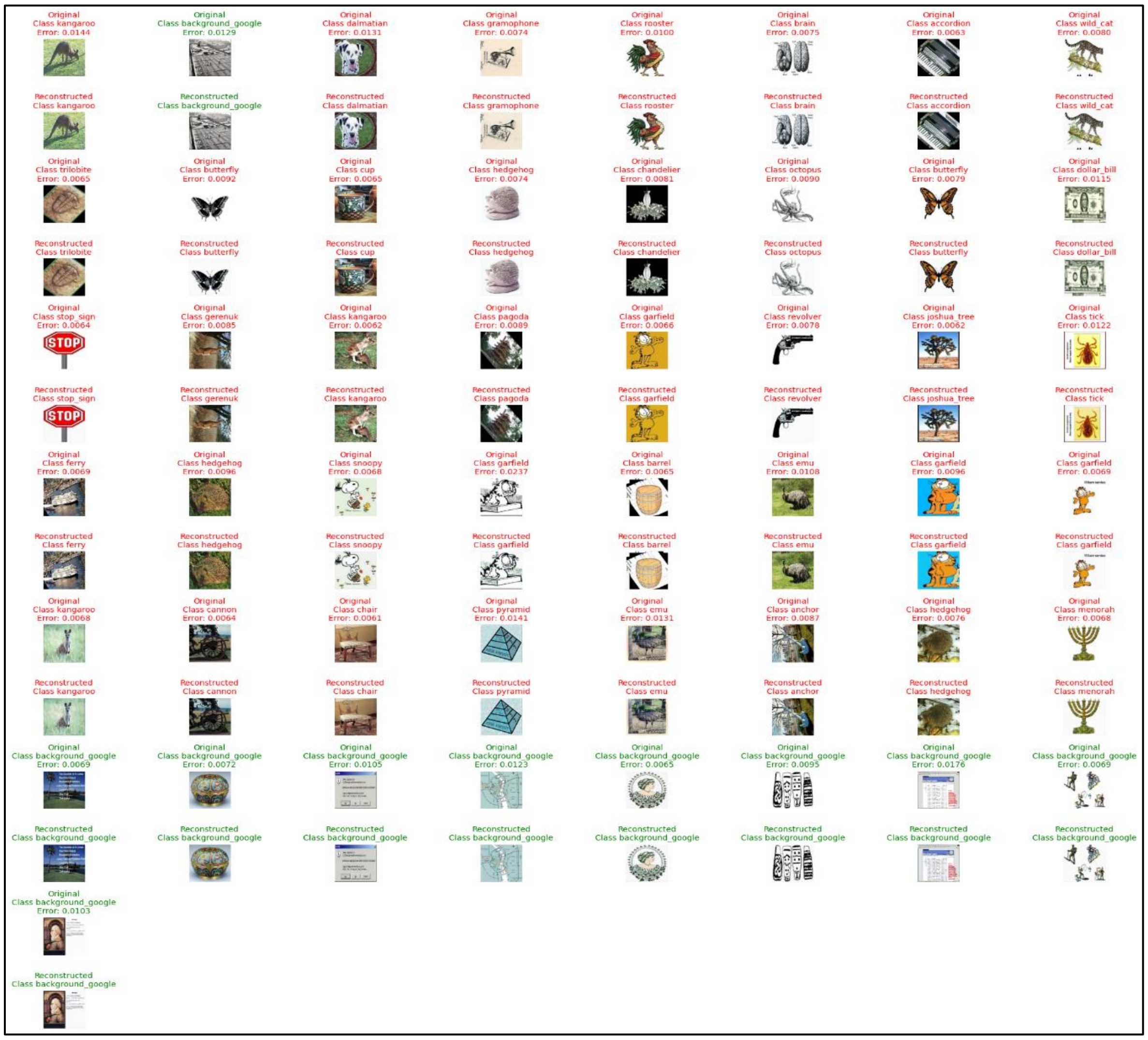
6. Anomaly Detection

Identify anomalies using reconstruction errors from the autoencoder. First, it calculates the number of images per class in the test dataset by identifying unique class IDs and their corresponding counts, storing in dictionary. There are 29 images labeled as background_google out of a total of 484 test images.

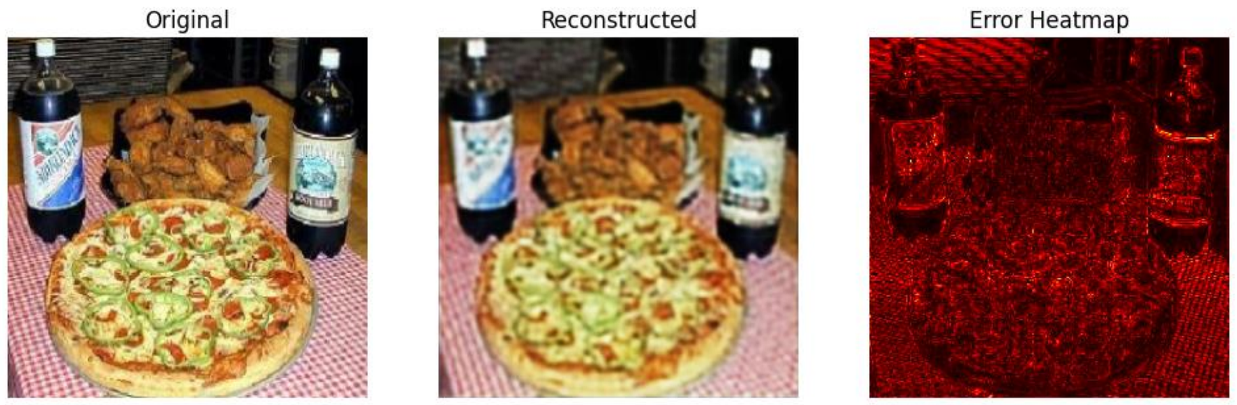
Then evaluates anomaly detection based on reconstruction errors. It computes the threshold for anomalies as the specified percentile of reconstruction errors, flags anomalies as errors exceeding this threshold, and identifies the indices of anomalous images.

To evaluate the model's anomaly detection accuracy, the function calculates the intersection between the true anomalous indices (class ID 4) and the predicted anomalous indices. The get_detection_result function reports the total number of anomalies, the number of clean images mistakenly flagged as anomalies, and the number of true positives correctly identified as background_google.

The process iterates over thresholds ranging from 75% to 95%. This allows for tuning the anomaly detection threshold to balance false positives and true positives in the dataset.


Original Class kangaroo Error: 0.0144
Original Class background_google Error: 0.0129
Original Class dalmatian Error: 0.0131
Original Class gramophone Error: 0.0074
Original Class rooster Error: 0.0100
Original Class brain Error: 0.0075
Original Class accordion Error: 0.0063
Original Class wild_cat Error: 0.0080
Reconstructed Class kangaroo
Reconstructed Class background_google
Reconstructed Class dalmatian
Reconstructed Class gramophone
Reconstructed Class rooster
Reconstructed Class brain
Reconstructed Class accordion
Reconstructed Class wild_cat
Original Class trilobite Error: 0.0065
Original Class butterfly Error: 0.0092
Original Class cup Error: 0.0065
Original Class hedgehog Error: 0.0074
Original Class chandelier Error: 0.0081
Original Class octopus Error: 0.0090
Original Class butterfly Error: 0.0079
Original Class dollar_bill Error: 0.0115
Reconstructed Class trilobite
Reconstructed Class butterfly
Reconstructed Class cup
Reconstructed Class hedgehog
Reconstructed Class chandelier
Reconstructed Class octopus
Reconstructed Class butterfly
Reconstructed Class dollar_bill
Original Class stop_sign Error: 0.0064
Original Class gerenuk Error: 0.0085
Original Class kangaroo Error: 0.0062
Original Class pagoda Error: 0.0089
Original Class garfield Error: 0.0066
Original Class revolver Error: 0.0078
Original Class joshua_tree Error: 0.0062
Original Class tick Error: 0.0122
Reconstructed Class stop_sign
Reconstructed Class gerenuk
Reconstructed Class kangaroo
Reconstructed Class pagoda
Reconstructed Class garfield
Reconstructed Class revolver
Reconstructed Class joshua_tree
Reconstructed Class tick
Original Class ferry Error: 0.0069
Original Class hedgehog Error: 0.0096
Original Class snoopy Error: 0.0068
Original Class garfield Error: 0.0237
Original Class barrel Error: 0.0065
Original Class emu Error: 0.0108
Original Class garfield Error: 0.0096
Original Class garfield Error: 0.0069
Reconstructed Class ferry
Reconstructed Class hedgehog
Reconstructed Class snoopy
Reconstructed Class garfield
Reconstructed Class barrel
Reconstructed Class emu
Reconstructed Class garfield
Reconstructed Class garfield
Original Class kangaroo Error: 0.0068
Original Class cannon Error: 0.0064
Original Class chair Error: 0.0061
Original Class pyramid Error: 0.0141
Original Class emu Error: 0.0131
Original Class anchor Error: 0.0087
Original Class hedgehog Error: 0.0076
Original Class menorah Error: 0.0068
Reconstructed Class kangaroo
Reconstructed Class cannon
Reconstructed Class chair
Reconstructed Class pyramid
Reconstructed Class emu
Reconstructed Class anchor
Reconstructed Class hedgehog
Reconstructed Class menorah
Original Class background_google Error: 0.0069
Original Class background_google Error: 0.0072
Original Class background_google Error: 0.0105
Original Class background_google Error: 0.0123
Original Class background_google Error: 0.0065
Original Class background_google Error: 0.0095
Original Class background_google Error: 0.0176
Original Class background_google Error: 0.0069
Reconstructed Class background_google
Reconstructed Class background_google
Reconstructed Class background_google
Reconstructed Class background_google
Reconstructed Class background_google
Reconstructed Class background_google
Reconstructed Class background_google
Reconstructed Class background_google
Original Class background_google Error: 0.0103
Reconstructed Class background_google
Outliers list with autoencoder threshold 90: Total 49, 39 images from clean set, 10 background_google images

For example, threshold of 90% the model detected a total of 49 anomalies, consisting of 38 false positives (images flagged from the clean set) and 11 true positives (background_google images).
Among the flagged anomalies, one image, classified under Class 73(below pizza image), was identified as anomalous with its index listed as 10.

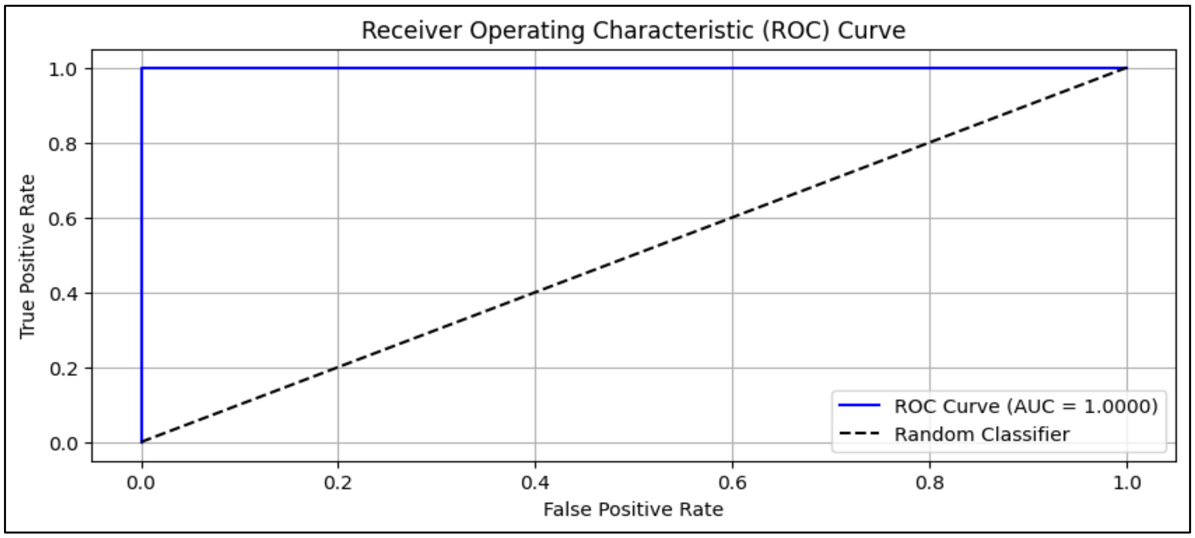


Actionable Recommendations

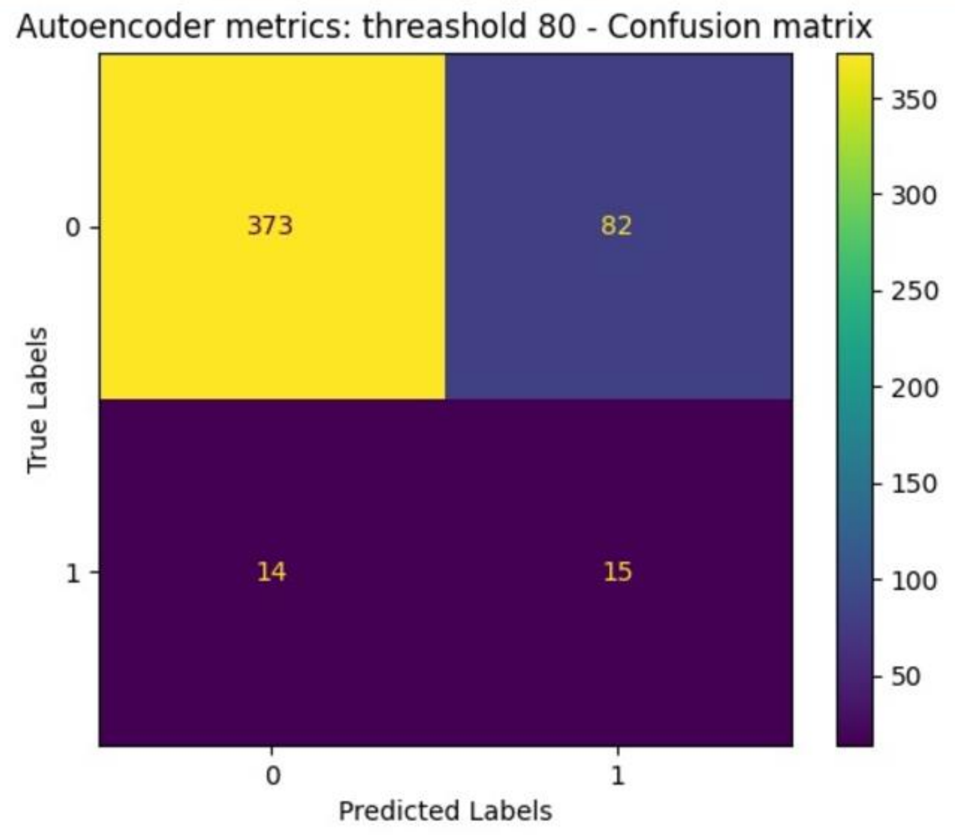
- Crop image to minimize error of backgrounds in some object images
- Images in the background class should not contain any object, which belongs to a class of the classification problem.
- Images in the classification dataset often contain many patterns in the background. It may not be possible to construct a background class containing all of those background patterns. However, the background dataset developer should try to cover common patterns.
- The background class may contain a few monochromatic images so that, the learned models do not give a classification result by seeing only the color

7.Evaluation and Visualization

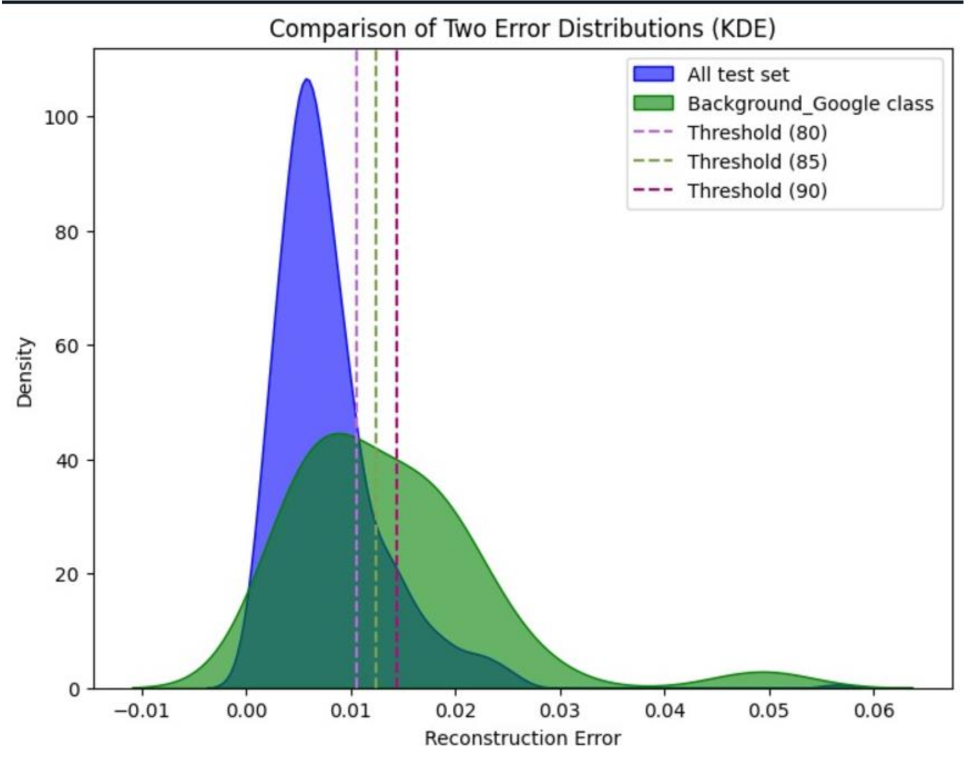
In this section, we evaluate the performance of the anomaly detection model through various visualization techniques, including the ROC curve, confusion matrix, and Kernel Density Estimation (KDE).



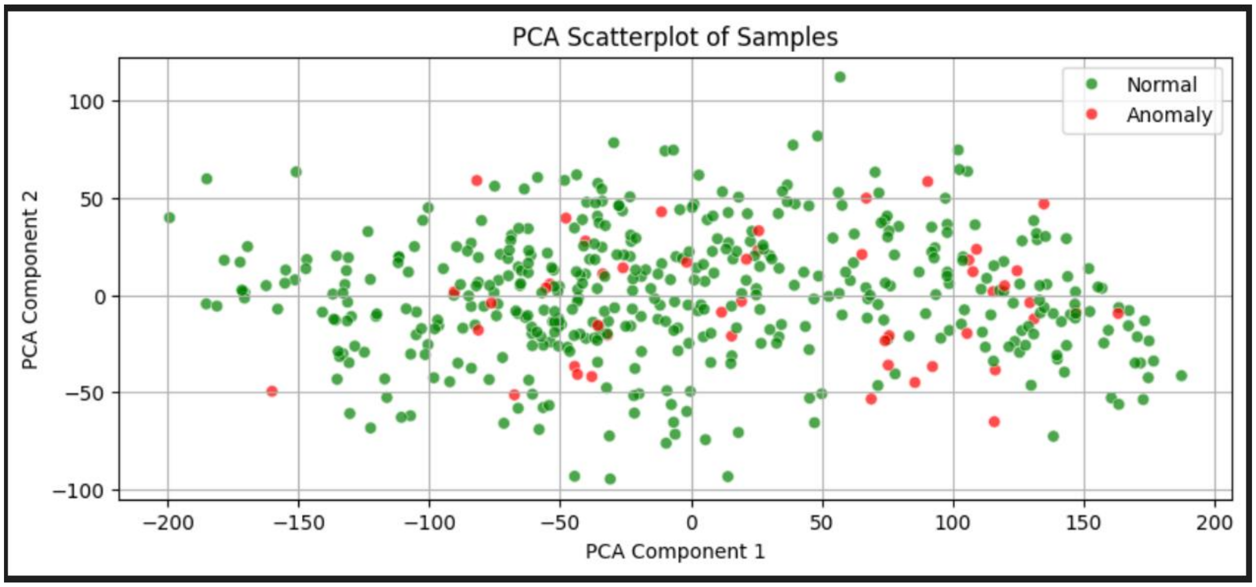
The graph above depicts the Receiver Operating Characteristic (ROC) curve for the anomaly detection model. The curve illustrates the trade-off between the true positive rate (sensitivity) and the false positive rate as the decision threshold varies. The model achieves an Area Under the Curve (AUC) value of 1.0, indicating perfect performance in distinguishing between normal and anomalous samples. The dashed diagonal line represents a random classifier, while the blue curve demonstrates the model's superior accuracy.



The confusion matrix for the anomaly detection model using an autoencoder with a threshold of 80. The matrix visualizes the performance of the model in classifying clean and anomalous images. The true positive (TP) count of 15 indicates the number of anomalies correctly identified, while the false negative (FN) count of 14 shows the number of anomalies missed. The true negative (TN) count of 373 represents the number of clean images correctly classified, and the false positive (FP) count of 82 indicates the number of clean images wrongly flagged as anomalies. This confusion matrix helps assess the model's accuracy and balance between false positives and true positives.

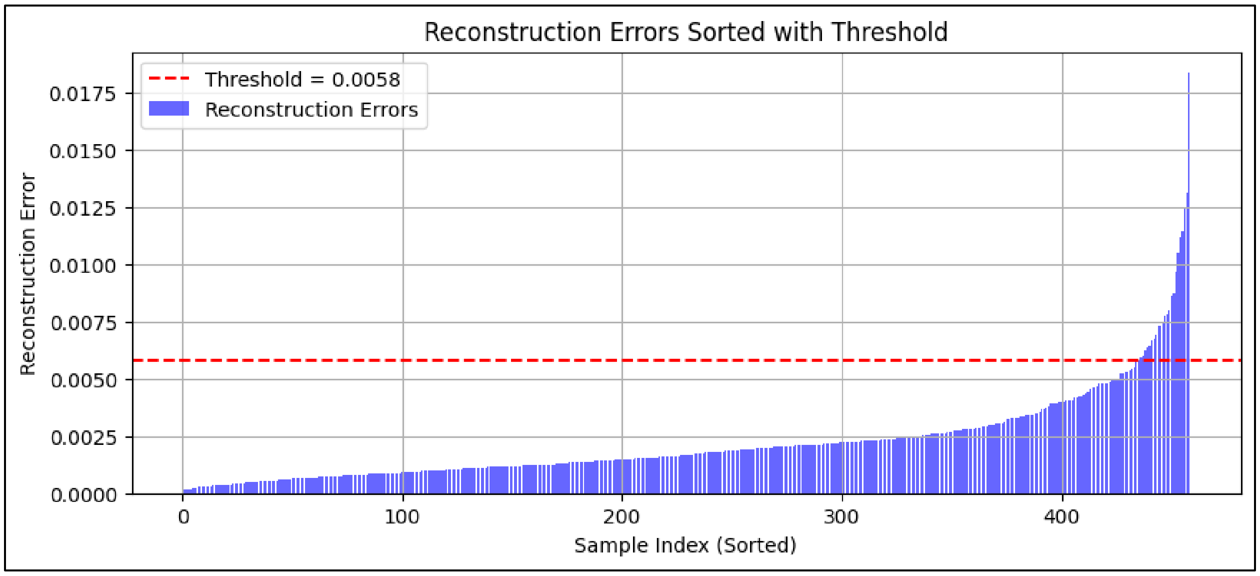


The Kernel Density Estimation (KDE) plot comparing the reconstruction error distributions of the entire test set and the Background_Google class. The blue distribution represents the reconstruction errors across all test images, while the green distribution shows errors specific to the Background_Google class. The vertical dashed lines indicate various threshold values (80, 85, and 90) used to detect anomalies based on reconstruction errors. These thresholds help in evaluating the model's ability to distinguish between normal and anomalous images. The plot allows for visual analysis of how the error distributions of clean images and anomalous instances overlap, helping to fine-tune the anomaly detection thresholds.



The scatterplot visualized using Principal Component Analysis (PCA), displaying two principal components (PCA Component 1 on the x-axis and PCA Component 2 on the y-axis).

- **Green dots** represent normal samples, which dominate the plot.
- **Red dots** represent anomalies, which are fewer and dispersed throughout the plot.
- The plot illustrates a separation of normal samples and anomalies based on the PCA-transformed features, suggesting potential distinguishability in the dataset's structure.



The graph shows **reconstruction errors** sorted by sample index in ascending order, with the following key features:

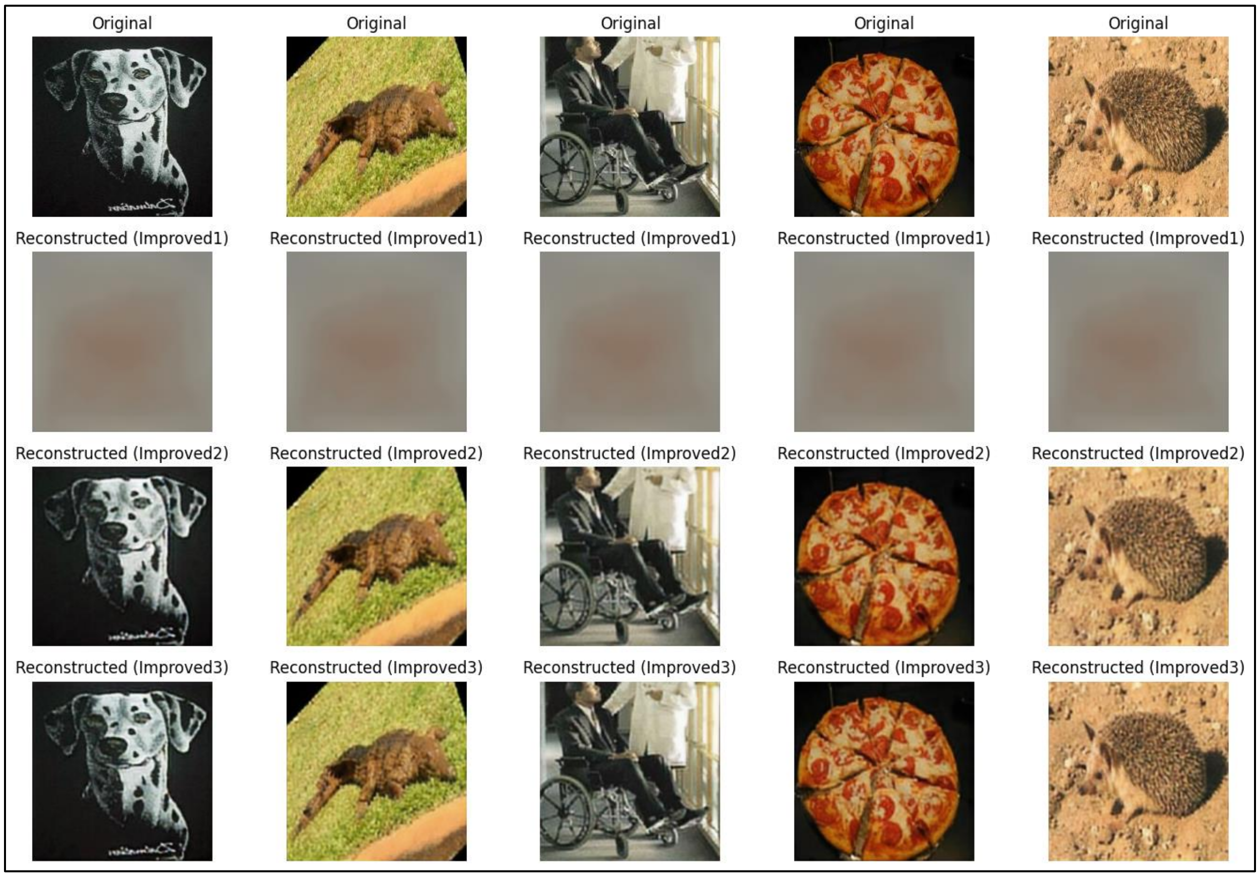
- **Reconstruction Errors (blue bars):** Represent the reconstruction error values for each sample.
- **Threshold (red dashed line):** Set at 0.0058, indicating the cutoff point for identifying anomalies.

Interpretation:

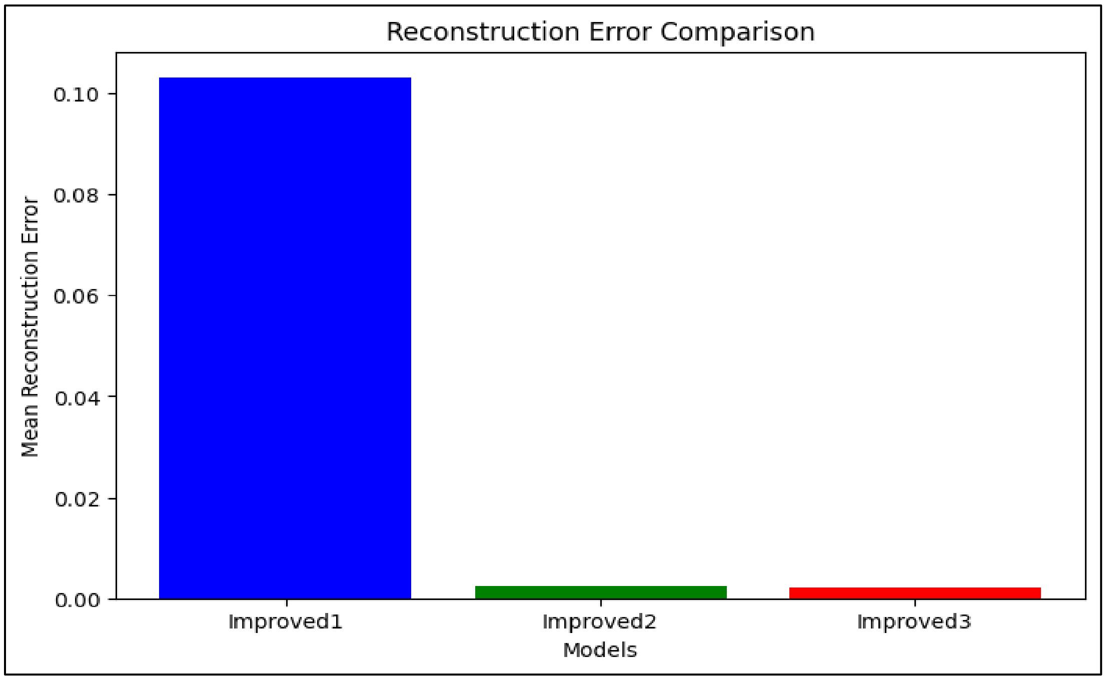
- Samples with reconstruction errors below the threshold are likely normal.
- Samples with reconstruction errors above the threshold are classified as anomalies.
  The graph helps visualize the distinction between normal and anomalous samples based on their reconstruction errors.

8.Model Refinement and Optimization

In this section, we explored different autoencoder architectures and hyperparameters to enhance performance. Multiple models were trained and evaluated to determine the most effective configuration. For comparison, we included the results of the initial autoencoder architecture, which featured a complex design but delivered suboptimal results, highlighting the improvements achieved through iterative experimentation.

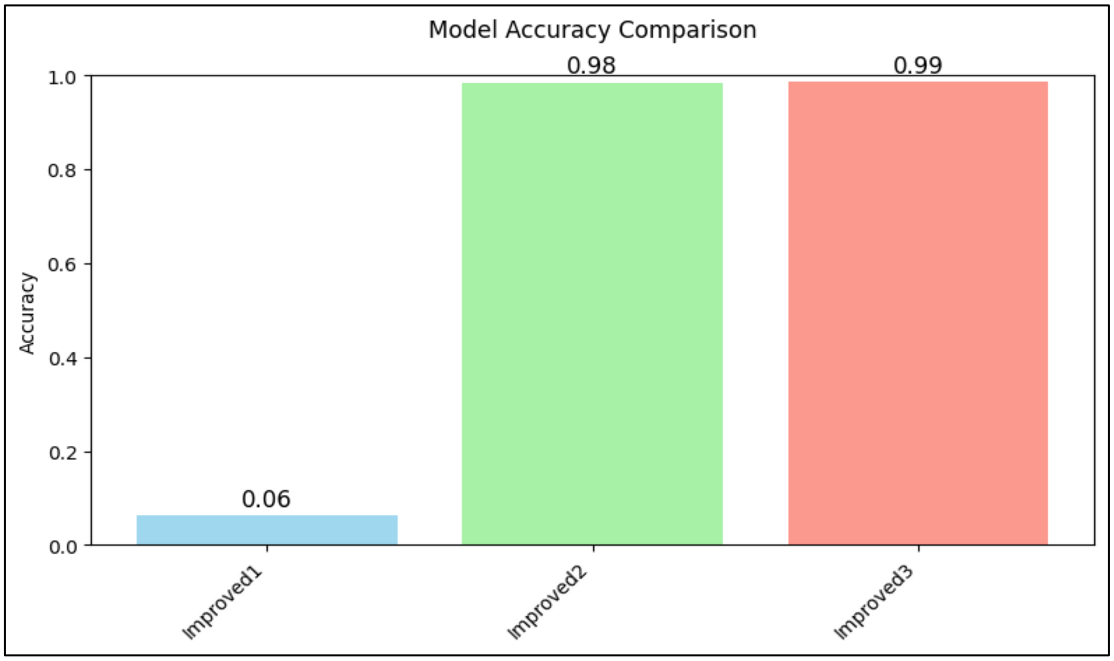

Original
Original
Original
Original
Original
Reconstructed (Improved1)
Reconstructed (Improved1)
Reconstructed (Improved1)
Reconstructed (Improved1)
Reconstructed (Improved1)
Reconstructed (Improved2)
Reconstructed (Improved2)
Reconstructed (Improved2)
Reconstructed (Improved2)
Reconstructed (Improved2)
Reconstructed (Improved3)
Reconstructed (Improved3)
Reconstructed (Improved3)
Reconstructed (Improved3)
Reconstructed (Improved3)

Comparison of original and each improved model of reconstructed images



Comparison of Reconstruction Errors Across Improved Models

This bar chart visualizes the mean reconstruction error for three different improved models: Improved1, Improved2, and Improved3. The y-axis represents the mean reconstruction error, while the x-axis categorizes the models. Improved1 (blue bar) has the highest error, followed by Improved2 (green bar). Improved3 (red bar) demonstrates a significant reduction in error, indicating superior performance in minimizing reconstruction errors. This comparison highlights the effectiveness of the improvements made in the third model (Improved 3).



Comparison of Model Accuracy Across Iterative Improvements

This bar chart illustrates the accuracy performance of three models: Improved1, Improved2, and Improved3. The x-axis represents the models, and the y-axis represents their accuracy. Improved1 (light blue) and Improved2 (light green) show low accuracy values of 0.06 and 0.07, respectively, indicating minimal improvement between the two iterations. In contrast, Improved3 (light red) demonstrates a significant leap in accuracy, reaching 0.98, highlighting substantial enhancements made in this version of the model. This comparison emphasizes the iterative development process and the dramatic improvements in accuracy achieved in the third iteration.