Project MVTec Anomaly Detection Report 2

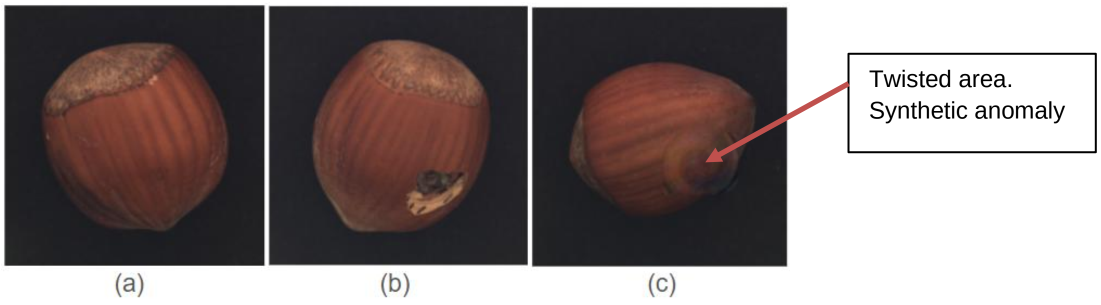
Table of Contents

Machine Learning Models

Data for Training, Testing and (Cross-)Validation

Datasets: Original and Synthetic

The **original data** of the MVTec Anomaly Detection Dataset consists of images with three dimensions of fixed size (`height x width x channels`) for 15 categories. The fact that models have to be trained, tested and evaluated for each of these categories individually, leads to a comparatively small amount of validation data, if different sets for cross validation have to be split from the available test data.
Therefore, in addition to the approach of splitting the test data into test and validation sets, a new set of **synthetic validation data** has been created by taking a percentage of the much larger training data of normal images and randomly inducing anomalies into some of them, by twisting small parts of the image.



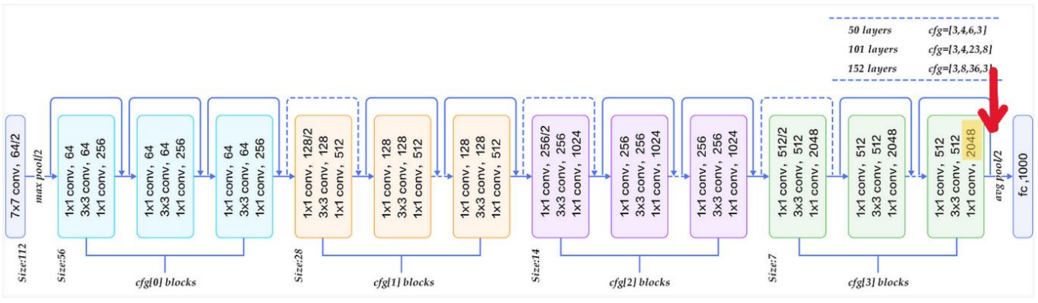
(a) original normal, (b) original anomalous and (c) synthetic anomalous images of category hazelnut

Feature Extraction: Direct and Deep

For use with the various models, features have been extracted from the image databases in two different ways:

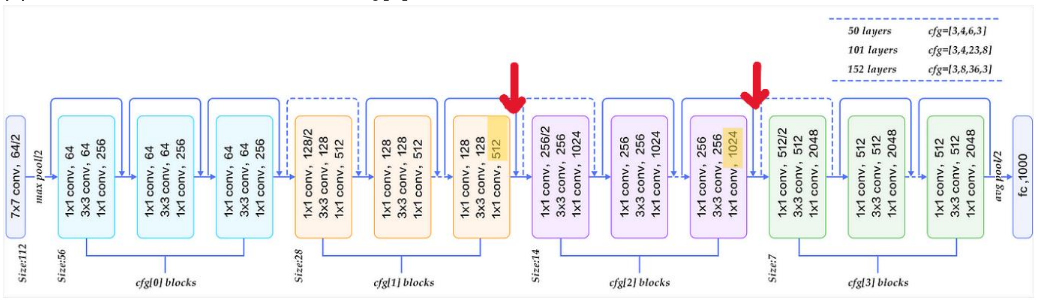
- **directly**, by taking the values for each pixel in each channel, as well as additionally computing various statistical values from the images. A list can be found as an appendix.

- with **deep feature extraction** using transfer learning by feeding the images into ResNet50, a pre-trained net consisting of consecutive blocks of convolutional layers and an output layer that classifies the input. To do this, the convolutional blocks in between extract meaningful features from the input images. Instead of using the classification output of the last, fully connected layer, we use outputs from different convolutional blocks as features to feed into our own models. The following pictures show the ResNet50 architecture as well as the locations after convolutional blocks 1, 2 and 3, where we extract features.



50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 128/2 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
1x1 conv, 256/2 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 512/2 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
2048
avg pool/2
fc, 1000
Size:112
Size:56
Size:28
Size:14
Size:7
cfg[0] blocks
cfg[1] blocks
cfg[2] blocks
cfg[3] blocks

(a) 2048 features extracted from cfg[3] blocks



50 layers cfg=[3,4,6,3]
101 layers cfg=[3,4,23,8]
152 layers cfg=[3,8,36,3]
7x7 conv, 64/2
max pool/2
1x1 conv, 64 3x3 conv, 64 1x1 conv, 256
1x1 conv, 128/2 3x3 conv, 128 1x1 conv, 512
1x1 conv, 128 3x3 conv, 128 1x1 conv, 512
512
1x1 conv, 256/2 3x3 conv, 256 1x1 conv, 1024
1x1 conv, 256 3x3 conv, 256 1x1 conv, 1024
1024
1x1 conv, 512/2 3x3 conv, 512 1x1 conv, 2048
1x1 conv, 512 3x3 conv, 512 1x1 conv, 2048
avg pool/2
fc, 1000
Size:112
Size:56
Size:28
Size:14
Size:7
cfg[0] blocks
cfg[1] blocks
cfg[2] blocks
cfg[3] blocks

(b) 1536 features extracted from cfg[1] and cfg[2] blocks

Two different sets of features are extracted:
- (a) 2048 features from block 3, or
- (b) 512 features from block 1 and 1024 features from block 2 (1536 features in total)

Combinations of different Datasets and Features to compare Model Performance

The above-described datasets and feature extraction methods are combined in different ways to train, test and validate Machine Learning models.
All models are trained on a training set of normal images and tested on different combinations of test and validation sets. The following five combinations have been used:

	feature extraction	**test- / validation- set splitting**
1.	direct	original mvtec (i)
2.	direct	original mvtec test + synthetic validation data (ii)
3.	deep (a)	original mvtec (i)
4.	deep (a)	original mvtec test + synthetic validation data (ii)
5.	deep (b)	original mvtec test + synthetic validation data (ii)

Table 1: feature extraction and data splitting methods

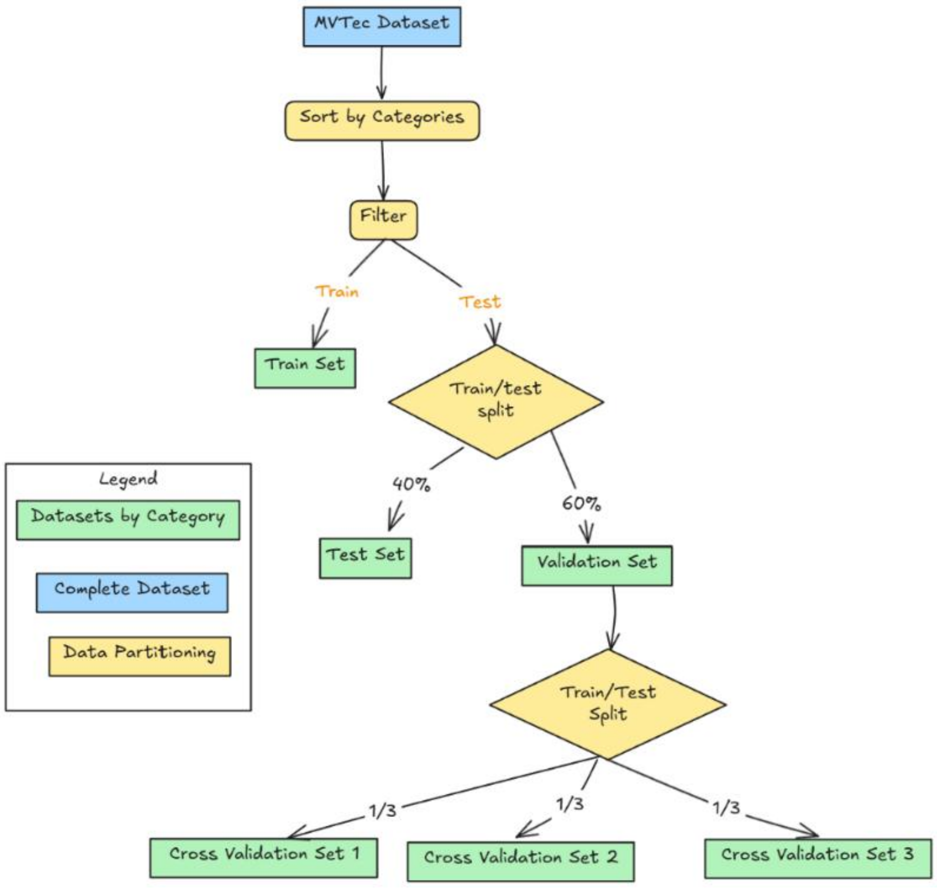
Splitting: Test- / Validation- Sets

While the images to train the different models are always taken from the 'train' portion of the original MVTec Dataset, test and validation sets are built in two ways:

- (i) by splitting the original test-set, or
- (ii) by splitting the original train-set and adding synthetic anomalies to a fixed proportion.

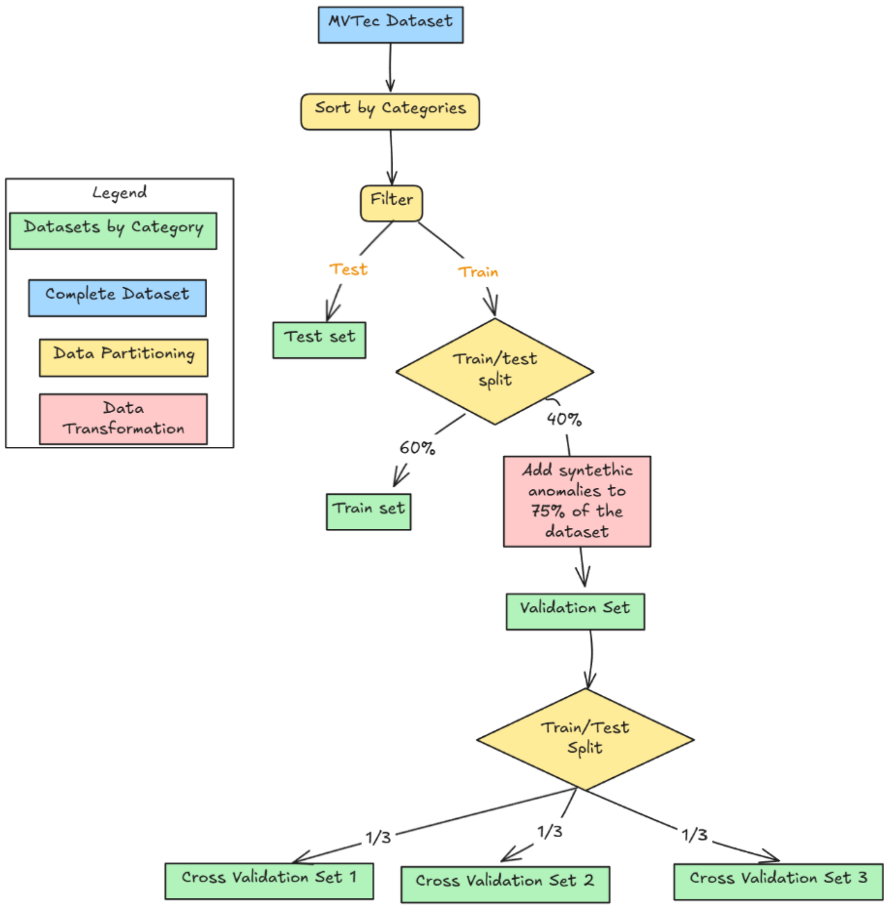
(i) Original MVTec Data

The following chart shows how data from the original MVTec Dataset is sorted by categories and then filtered into different subsets for training and testing.
The original test data is then split into 40% remaining test data and 60% validation data, which is further split evenly into three cross validation sets.



(ii) Original MVTec Test Data + Synthetic Validation Data

The following chart shows how data from the original MVTec Dataset is sorted by categories and then filtered into different subsets for training and testing.
The original training data is then split into 60% remaining training data and 40% validation data, which is further split evenly into three cross validation sets.



Standardization and PCA

The data preprocessing pipeline consists of two main steps: **feature standardization** and **dimensionality reduction using PCA**.

First, feature values are standardized using **Min-Max scaling to the range [0,1]**. This ensures that all features have a uniform scale, preventing those with larger numerical ranges from dominating the analysis.
After standardization, **Principal Component Analysis (PCA)** is applied to reduce dimensionality. The number of principal components is determined by retaining **95% of the variance** in the data. This step helps in reducing computational complexity while preserving the most relevant information.

For each category, the following steps are performed:

1. The training dataset is standardized using **sklearn's StandardScaler**, and the same transformation is applied to the test and validation datasets.
2. PCA is fitted on the standardized training data to determine the principal components.
3. The training, test, and validation datasets are transformed using the trained PCA model.
4. The transformed PCA feature sets are saved for future use.

Model Description

Several anomaly detection models were evaluated to identify deviations from normal patterns in high-dimensional datasets. The models include One-Class Support Vector Machine (One-Class SVM), Isolation Forest, Local Outlier Factor (LOF), and Elliptic Envelope. Each model employs a different methodology to distinguish normal instances from anomalies.

- **One-Class SVM:** This model maps input data into a high-dimensional feature space using a kernel function and identifies a hyperplane that separates normal data from anomalies.
- **Isolation Forest:** It randomly selects features and splits data points into smaller subsets, isolating outliers more efficiently than normal points.
- **Local Outlier Factor (LOF):** This model measures the local density deviation of a given data point compared to its neighbors, identifying instances with significantly lower densities as anomalies.
- **Elliptic Envelope:** Assumes data follows a Gaussian distribution and detects outliers based on Mahalanobis distance.

Hyperparameter Grid

The models were evaluated using the following hyperparameter grids:

- **One-Class SVM**

Hyperparameter	**Values**	**Explanation**
nu	0.001	An upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.
kernel	‘rbf’, ‘linear’, ‘poly’	Specifies the kernel type to be used in the algorithm.
gamma	‘scale’, ‘auto’	Kernel coefficient for ‘rbf’, ‘poly’, and ‘sigmoid’.
degree	2, 3	Degree of the polynomial kernel function (‘poly’).

- **Isolation Forest**

Hyperparameter	**Values**	**Explanation**
n_estimators	100, 200, 300, 400	The number of base estimators in the ensemble.
contamination	0.01	The proportion of outliers in the data set.
max_samples	‘auto’, 1.0	The number of samples to draw from the dataset to train each base estimator.
max_features	0.9, 1.0	The number of features to draw from the dataset to train each base estimator.

- **Local Outlier Factor (LOF)**

Hyperparameter	**Values**	**Explanation**
n_neighbors	10, 20, 30, 50	The number of neighbors to use for k-neighbors queries.

contamination	0.01	The proportion of outliers in the data set.
leaf_size	20, 30, 50, 100	The leaf size passed to the BallTree or KDTree.
metric	‘minkowski’, ‘euclidean’	The distance metric to use for the tree.

- **Elliptic Envelope**

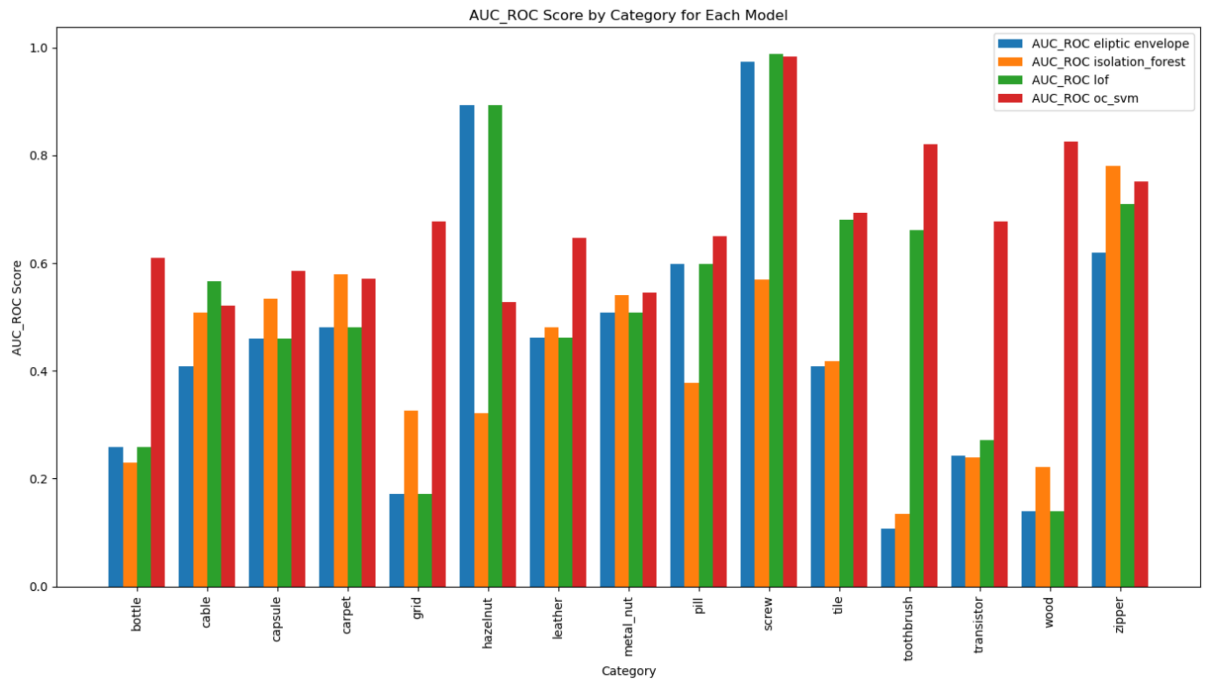
Hyperparameter	**Values**	**Explanation**
contamination	0.01	The proportion of outliers in the data set.
support_fraction	0.9, 1.0	The proportion of points to include in the support of the raw MCD estimate.
assume_centered	True, False	If True, the support of the data is centered.

Model Training, Testing and Validation

The models were trained, tested and evaluated using the five different combinations of data splitting and feature extraction described in *Table 1*. In all cases, a 3-fold cross-validation was applied.

Results

1. **Direct Feature Extraction & Original MVTec Split**

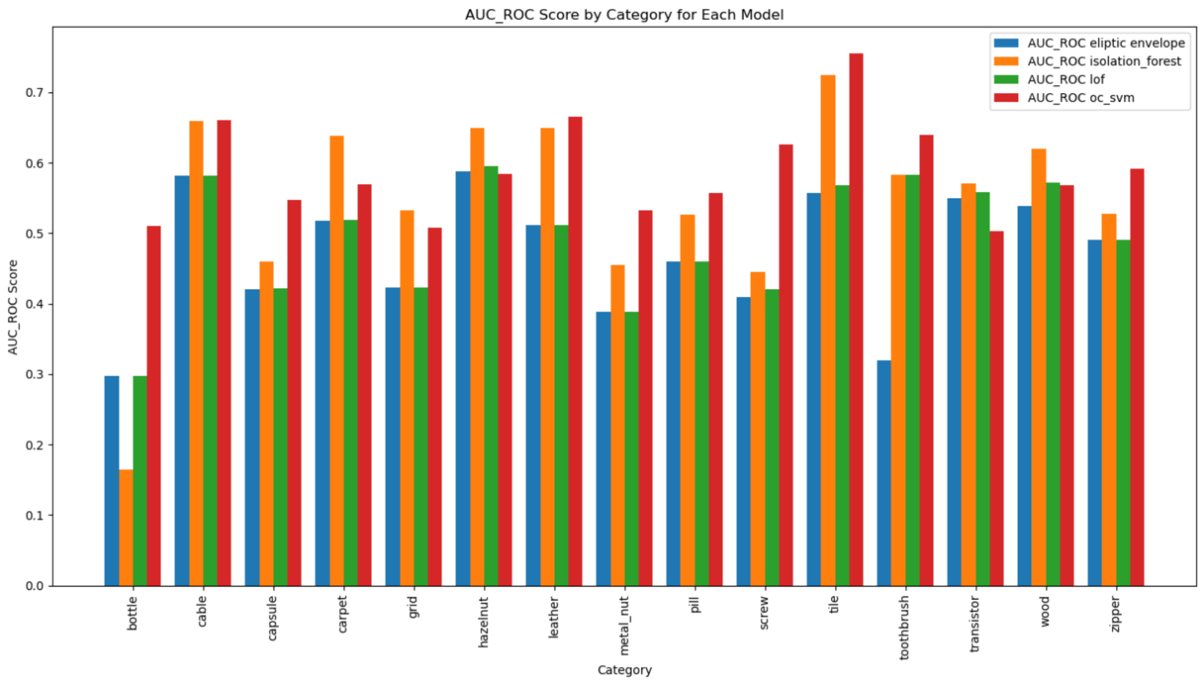


The bar chart represents the **AUC-ROC scores** for different models across various categories. Here are some observations:

- **Overall Performance:**

    - The OC-SVM (red) model appears to perform well across most categories, often scoring higher than other models.
    - The LOF (green) model performs well in some cases but has lower scores in certain categories.
    - Isolation Forest (orange) and Elliptic Envelope (blue) show more variance in performance across different categories.
- **High-Performing Categories:**
    - The "screw" category shows the highest AUC-ROC scores across all models, indicating that all models perform well for this class.
    - The "zipper" and "tile" categories also exhibit strong performance across models.
- **Low-Performing Categories:**
    - The "bottle", "wood", and "grid" categories have relatively low AUC-ROC scores across all models, indicating difficulty in anomaly detection for these objects.
- **Model-Specific Performance:**
    - Elliptic Envelope (blue) generally has lower performance across many categories.
    - Isolation Forest (orange) and LOF (green) show mixed performance, with some high and some low scores.
    - OC-SVM (red) is consistently one of the best-performing models.

2. **Direct Feature Extraction & Synthetic Validation Data**

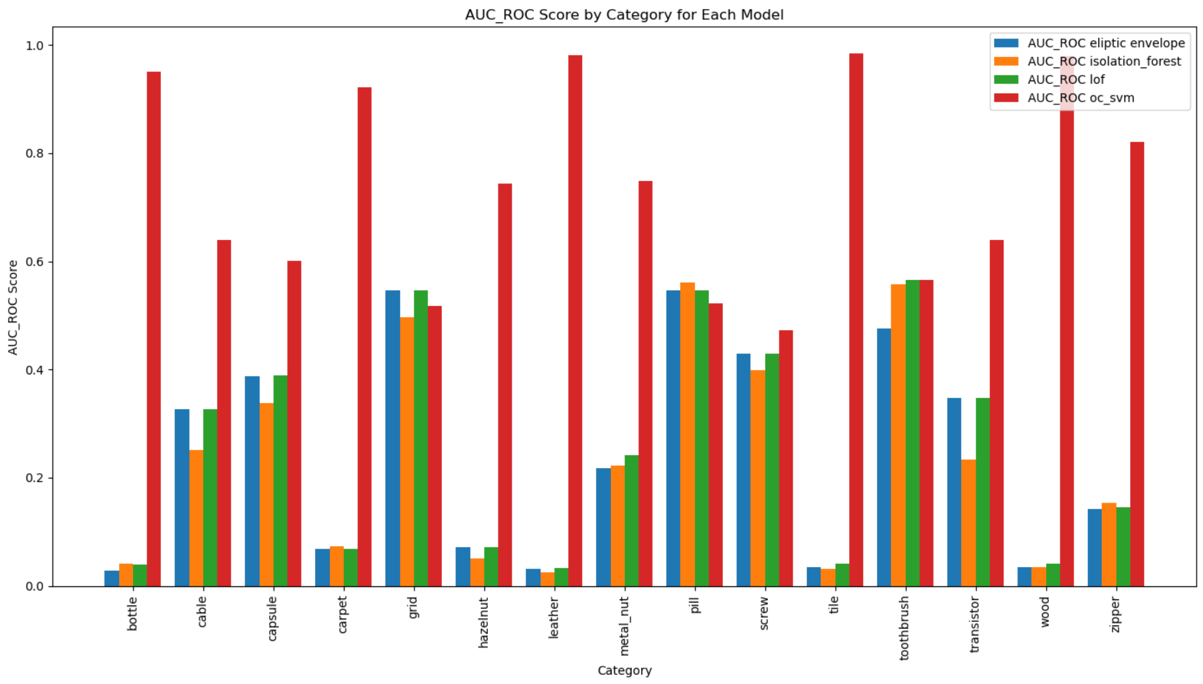


- **Overall Performance:**
    - The OC-SVM (red) model remains the most consistent top performer across most categories.
    - Isolation Forest (orange) also performs well but has some categories where it is slightly lower.

- LOF (green) and Elliptic Envelope (blue) have varying performances across different categories.

- **High-Performing Categories:**
    - The "tile" category has the highest AUC-ROC score, especially for OC-SVM, followed by Isolation Forest.
    - The "cable" and "screw" categories also show strong performance across most models.
    - The "toothbrush" and "transistor" categories have consistently high scores.
- **Low-Performing Categories:**
    - "bottle" has the lowest AUC-ROC scores, with LOF (green) struggling the most.
    - "hazelnut" and "metal_nut" categories have relatively lower scores compared to others.
- **Model-Specific Performance:**
    - OC-SVM (red) remains the strongest model in almost all categories, making it the best choice for overall anomaly detection.
    - Isolation Forest (orange) performs better in categories like cable, tile, and wood.
    - LOF (green) has more variability, with weaker performance in categories like bottle and hazelnut.
    - Elliptic Envelope (blue) shows stable but lower performance across many categories.

3. **Deep Feature Extraction Block 3 & Original MVTec Split**

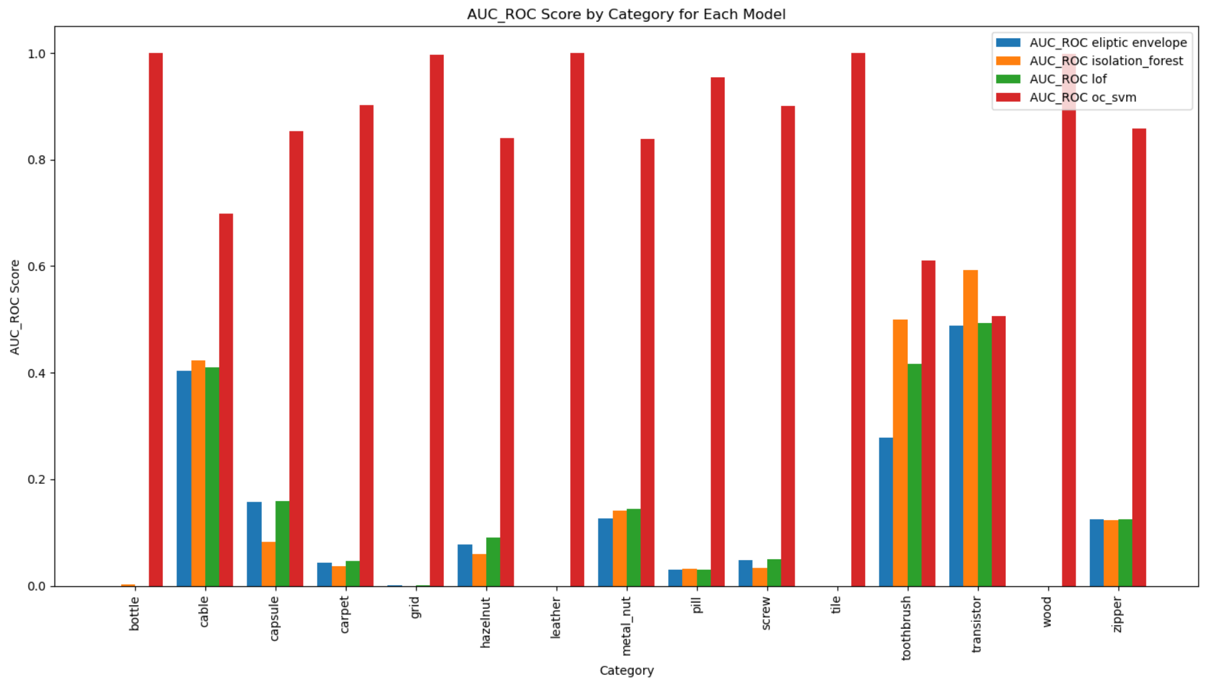


- **Overall Performance:**
    - OC-SVM (red) consistently outperforms all models, often achieving AUC-ROC near 1.0.

- Elliptic Envelope (blue), Isolation Forest (orange), and LOF (green) exhibit lower and more variable scores across categories.
- OC-SVM shows remarkable consistency, making it the most reliable anomaly detection model.

- **High-Performing Categories (OC-SVM Dominance):**
  - OC-SVM achieves near-perfect scores in: Bottle, Carpet, Grid, Screw, Tile, Zipper
  - These categories indicate OC-SVM's superior detection capabilities.
- **Low-Performing Categories:**
  - Bottle, wood, and metal_nut have significantly lower AUC-ROC scores across all models except OC-SVM.
  - Elliptic Envelope, Isolation Forest, and LOF perform poorly in these categories, struggling to detect anomalies effectively.
- **Model-Specific Insights:**
  - OC-SVM (red):
    - Best performance across all categories.
    - Consistently achieves the highest AUC-ROC scores.
    - Most reliable model for anomaly detection.
  - Isolation Forest (orange) & LOF (green):
    - Moderate performance, with some categories showing decent AUC-ROC.
    - Not as stable as OC-SVM, but still viable with tuning.
  - Elliptic Envelope (blue):
    - Weakest model overall.
    - Highly inconsistent performance, often the lowest AUC-ROC scores.

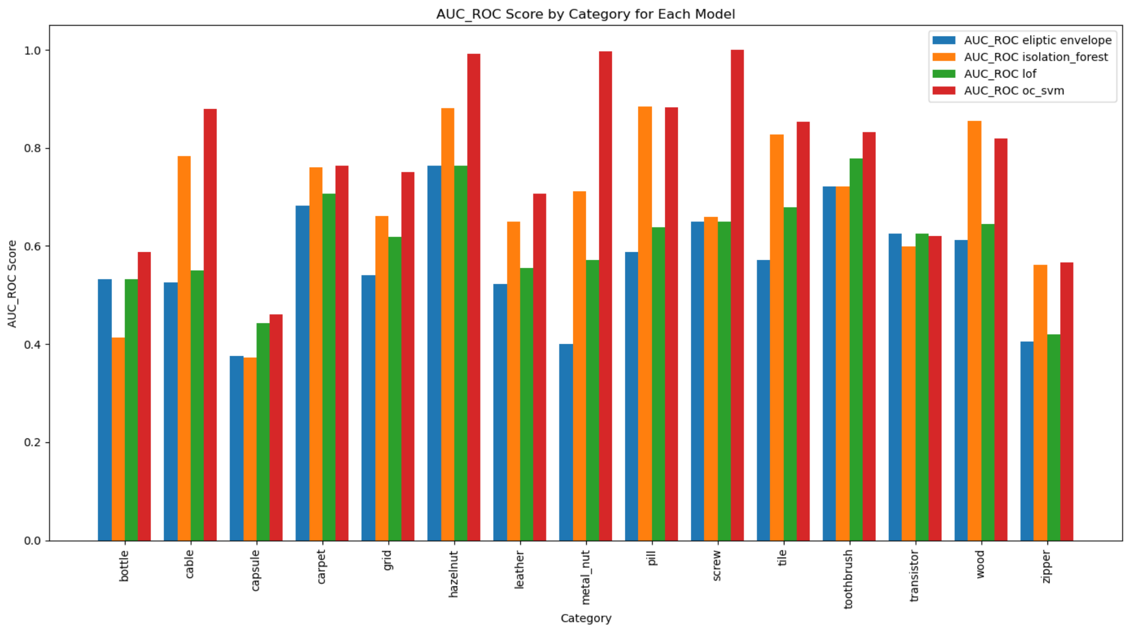
4. **Deep Feature Extraction Block 3 & Synthetic Validation Data**



- **Overall Performance:**
  - OC-SVM (red) consistently outperforms all models, with AUC-ROC scores close to 1.0 across most categories.

    - Elliptic Envelope (blue), Isolation Forest (orange), and LOF (green) exhibit much lower and more variable scores.
    - OC-SVM's dominance indicates it is the most effective anomaly detection model among those tested.
- **High-Performing Categories (OC-SVM Dominance):**
    - OC-SVM achieves near-perfect scores in: bottle, carpet, grid, screw, tile, zipper.
    - These results suggest that OC-SVM is highly reliable in detecting anomalies in these categories.
- **Low-Performing Categories:**
    - Categories with low AUC-ROC scores for all models except OC-SVM: **bottle, wood, metal_nut, hazelnut, pill**.
    - Elliptic Envelope, Isolation Forest, and LOF fail to detect anomalies effectively in these categories.
- **Model-Specific Insights:**
    - **OC-SVM (red)**:
        - Best-performing model across all categories.
        - Achieves the highest AUC-ROC scores, making it the most reliable option for anomaly detection.
    - **Isolation Forest (orange) & LOF (green)**:
        - Show moderate performance but are inconsistent across categories.
        - May require additional tuning or feature engineering for improvement.
    - **Elliptic Envelope (blue)**:
        - Weakest model overall, with very low and inconsistent AUC-ROC scores.
        - Struggles in almost all categories, making it the least effective option.
        -

5. Deep Feature Extraction Block 1&2 & Synthetic Validation Data



- **Overall Performance:**
    - OC-SVM (red) consistently demonstrates superior performance, achieving high AUC-ROC scores across most categories.
    - Isolation Forest (orange) and LOF (green) exhibit moderate and variable performance depending on the category.
    - Elliptic Envelope (blue) tends to underperform in most scenarios, with lower and less consistent scores.
- **High-Performing Categories:**
    - **OC-SVM** achieves the highest scores in categories such as hazelnut, screw, and tile.
    - **Isolation Forest and LOF** also perform well in detecting anomalies in categories like grid, capsule, toothbrush, and transistor.
- **Low-Performing Categories:**
    - Categories with relatively lower AUC-ROC scores across multiple models include bottle, capsule, metal nut, pill, and wood.
    - These categories may benefit from improved feature extraction or alternative anomaly detection methods.
- **Model-Specific Insights:**
    - **OC-SVM (red):**
        - Demonstrates robust performance across nearly all categories.
        - Achieves the highest AUC-ROC scores in most cases, making it the most reliable choice overall.
    - **Isolation Fores**t (orange) & LOF (green):
        - Exhibit good performance in certain categories but show variability.
        - It can serve as alternatives but may require additional fine-tuning for optimal results.
    - **Elliptic Envelope (blue):**
        - The least effective model with highly variable outcomes.
        - Faces difficulties in most categories, rendering it a less dependable option.

Overall Summary of AUC-ROC Score Analysis Across Different Feature Extraction Methods

1. Direct Feature Extraction & Original MVTec Split

- Selected features from the CSV dataset.
- Trained only on normal train data.
- Performance varies across categories, indicating that handcrafted features alone may not generalize well for all cases.

2. Direct Feature Extraction & Synthetic Validation Data

- Selected features from normal data.
- Validated using 75% anomalous synthetic data.
- Performance improves slightly but remains inconsistent across distinct categories.
- The introduction of synthetic anomalies provides a broader test environment but may not always align with real-world anomaly distributions.

3. Deep Feature Extraction Block 3 & Original MVTec Split

- Extracted features from the last layer of ResNet50.
- Trained on normal train data.
- Shows better performance than manual features, indicating the importance of deep learning-based feature extraction.
- Still has some variability in performance, suggesting potential overfitting to normal training data.

4. Deep Feature Extraction Block 3 & Synthetic Validation Data

- Extracted last-layer features from ResNet50.
- Trained on normal data and validated with 75% synthetic anomalous data.
- Higher AUC-ROC scores compared to manual features.
- Demonstrates improved anomaly detection, due to richer feature representations learned by ResNet50.

5. Deep Feature Extraction Block 1&2 & Synthetic Validation Data

- Extracted features from the second and third latent layers of ResNet50.
- Trained on normal data and validated with 75% synthetic anomalous data.
- Shows the best overall performance among all methods.
- Using features from deeper layers of ResNet50 captures more useful information for anomaly detection.

Key Takeaways

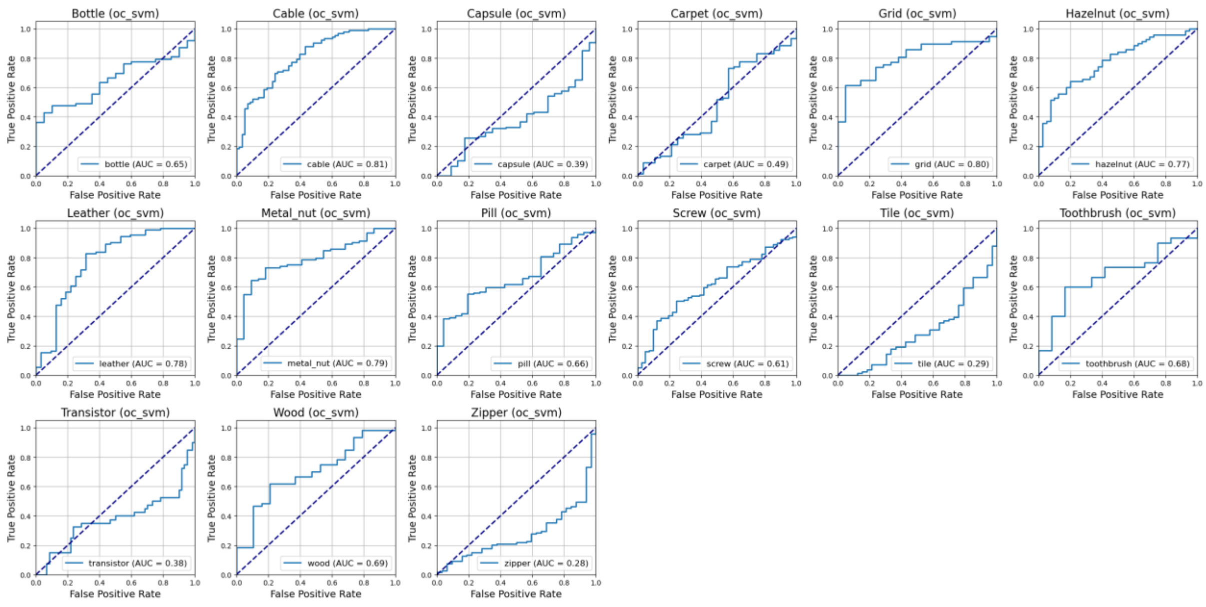
- ResNet50-based feature extraction significantly outperforms manual features in anomaly detection.
- Using multiple latent layers (1st & 2nd block last layers) from ResNet50 provides the best results, suggesting that deeper feature representations are more informative for anomaly detection.
- Synthetic validation helps evaluate model robustness but might introduce distributional mismatches compared to real-world anomalies.
- Manual feature selection struggles with generalization, making deep learning-based approaches more effective.

Best Model

Based on the analysis, Method 5 (ResNet50 - 1st & 2nd block last layers - Synthetic Validation) showed strong performance in anomaly detection. This method achieved high AUC-ROC scores across various categories, indicating the effectiveness of deep feature representations extracted from multiple latent layers.

For clarity and to focus on the effective approach, the AUC-ROC graphs and the confusion matrix for Method 5 will be presented.

AUC-ROC Curves:



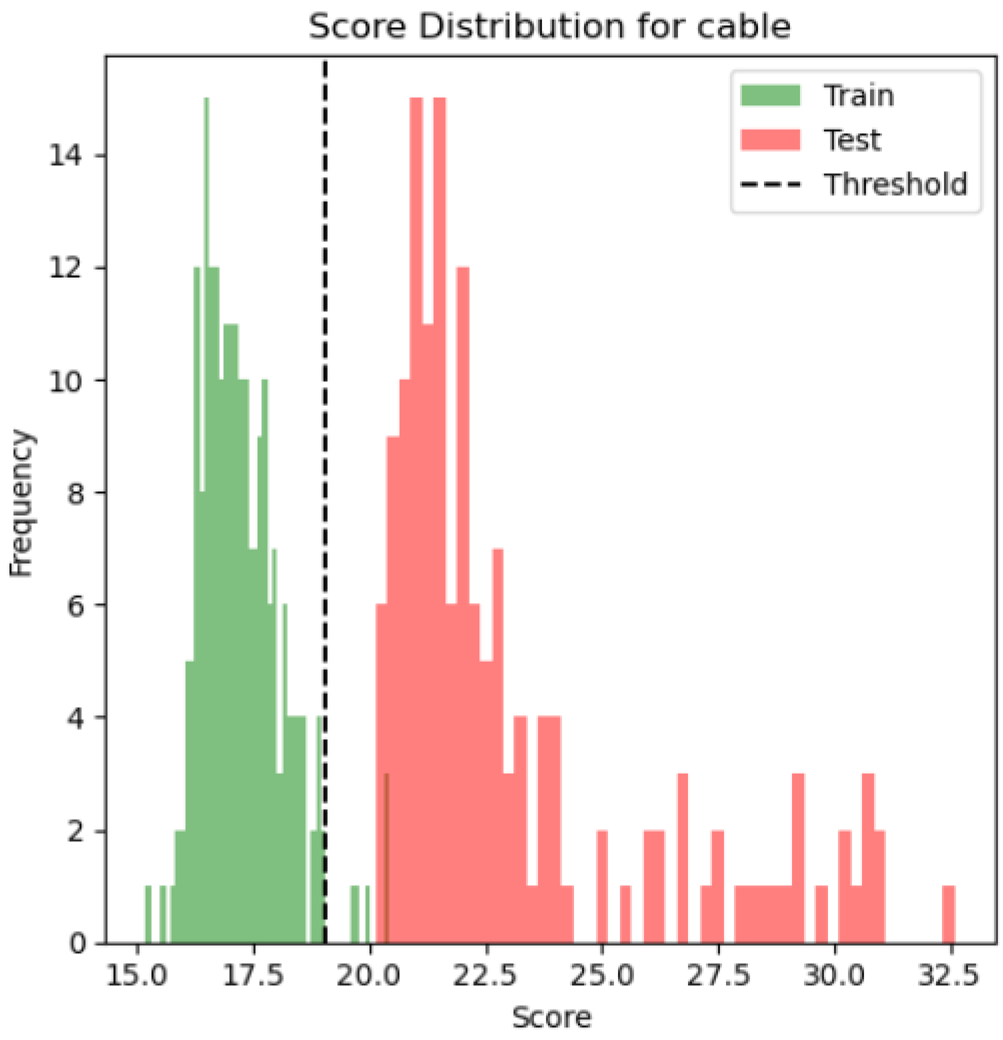
- The AUC-ROC graphs for each category show the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR).
- Higher AUC values indicate better discrimination between normal and anomalous samples.
- Categories such as cable, grid, hazelnut, leather, and metal_nut show strong AUC scores, indicating effective anomaly detection.
- Some categories, like capsule, carpet, tile, and zipper, have lower AUC scores, suggesting that these classes are more challenging for the model.

Threshold Optimization

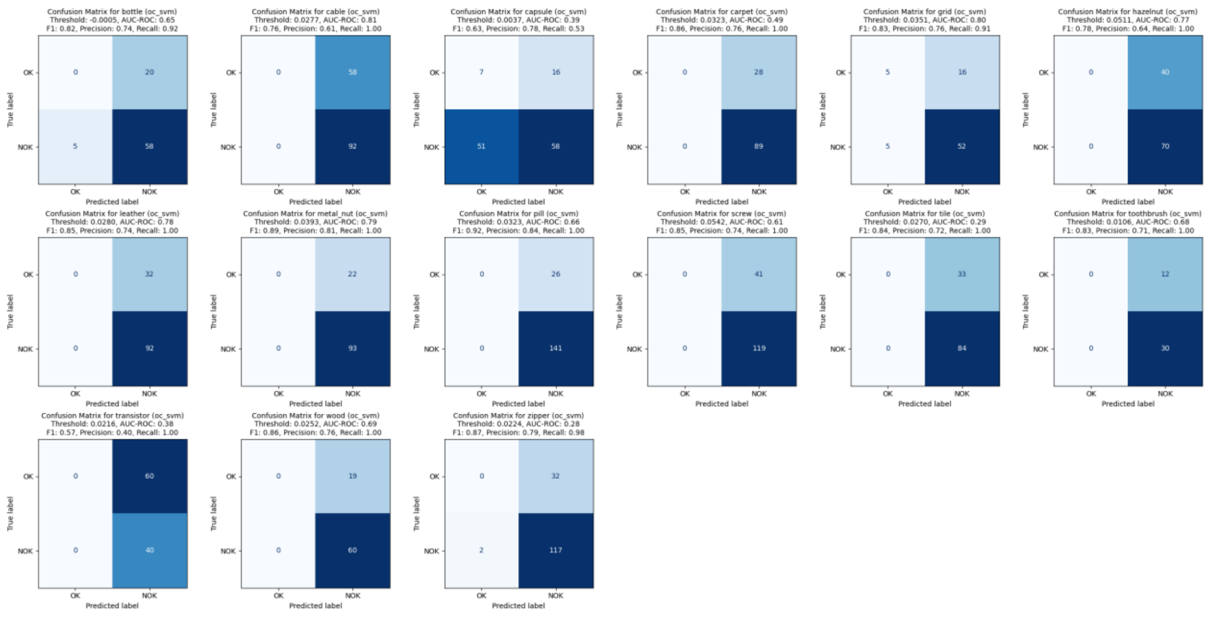
- To achieve the best classification performance, **we conducted a threshold search** to maximize the **F1-score**. This ensures that our model is optimally balancing precision and recall for detecting anomalies.
  To determine the optimal threshold for classifying normal and anomalous data, we analyzed the **score distribution** for both **train (normal) and test (anomalous) samples**.
  The image above illustrates an example of this process for the **"cable"** category:
  - **Green histogram**: Represents the distribution of scores for the normal (train) data.
  - **Red histogram**: Represents the distribution of scores for the anomalous (test) data.
  - **Dashed black line**: Represents the **chosen threshold**, which acts as a decision boundary between normal and anomalous samples.

  Threshold Selection Strategy
  - The threshold is set to **best separate normal and anomalous distributions**, ensuring minimal overlap.
  - **Ideally**, the threshold should:
    - Minimize **false positives** (normal data misclassified as anomalies).
    - Minimize **false negatives** (anomalies misclassified as normal).

    - Maximize the **F1-score**, balancing precision and recall.



- The reported AUC-ROC and F1-score values are from the **test dataset**, which typically has slightly lower performance than the validation dataset used for hyperparameter tuning. This is expected, as the test set represents unseen data.



- The **confusion matrices** provide a direct visualization of model predictions.
- Each matrix includes the **optimized threshold** used for classification, along with the corresponding **AUC-ROC, F1-score, Precision, and Recall** values.
- The **high recall values** in many categories indicate that the model is effective at detecting most anomalies.
- In some cases, precision is lower, suggesting that the model may generate some false positives.

Key Takeaways

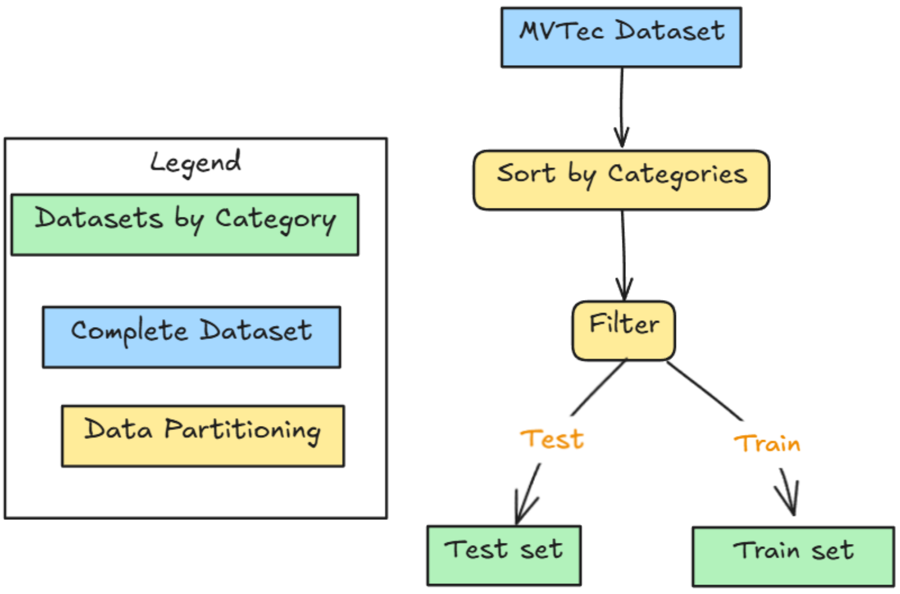
- ResNet50-based feature extraction (1st & 2nd block last layers) is the best-performing method for anomaly detection.
- Test dataset performance is slightly lower than the tuned validation set, which is expected in real-world evaluation.
- AUC-ROC graphs confirm robust performance in several categories, though some remain challenging.
- Confusion matrices highlight the impact of threshold tuning, with an emphasis on balancing recall and precision.

KNN+Resnet50 Model

To implement KNN for anomaly detection, we utilized deep features extracted from ResNet50's block cfg(1) and cfg(2) latent layers (this method is suggested in the paper "DFR: Deep Feature Reconstruction for Unsupervised Anomaly Segmentation").
These features were stored in a memory bank, which allowed for efficient similarity calculations using KNN. This approach follows the similar methodology as the previous models, ensuring consistency in evaluation.

Unlike OC-SVM or Isolation Forest, which are model-based approaches, KNN operates purely on distances.

Dataset

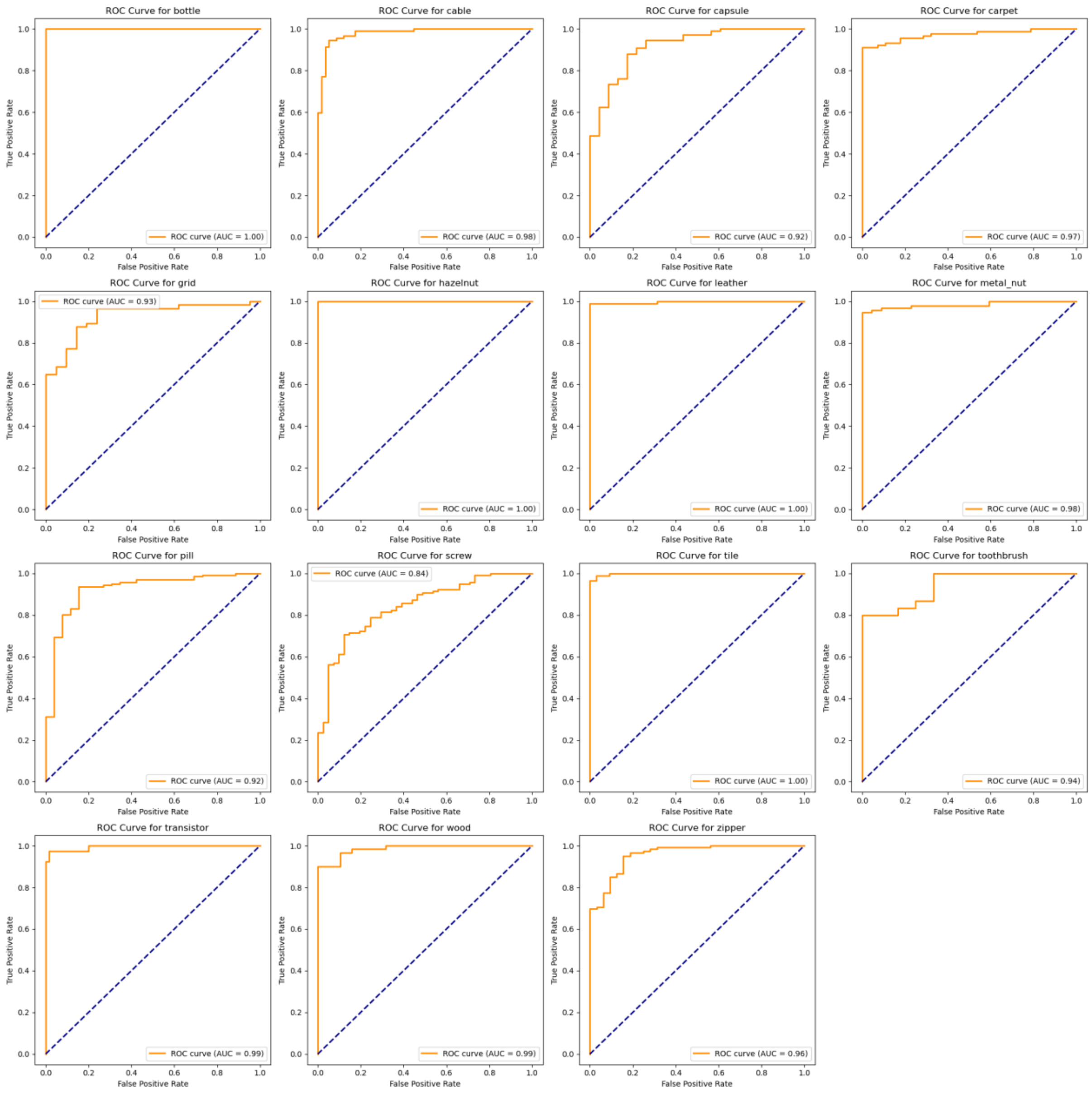


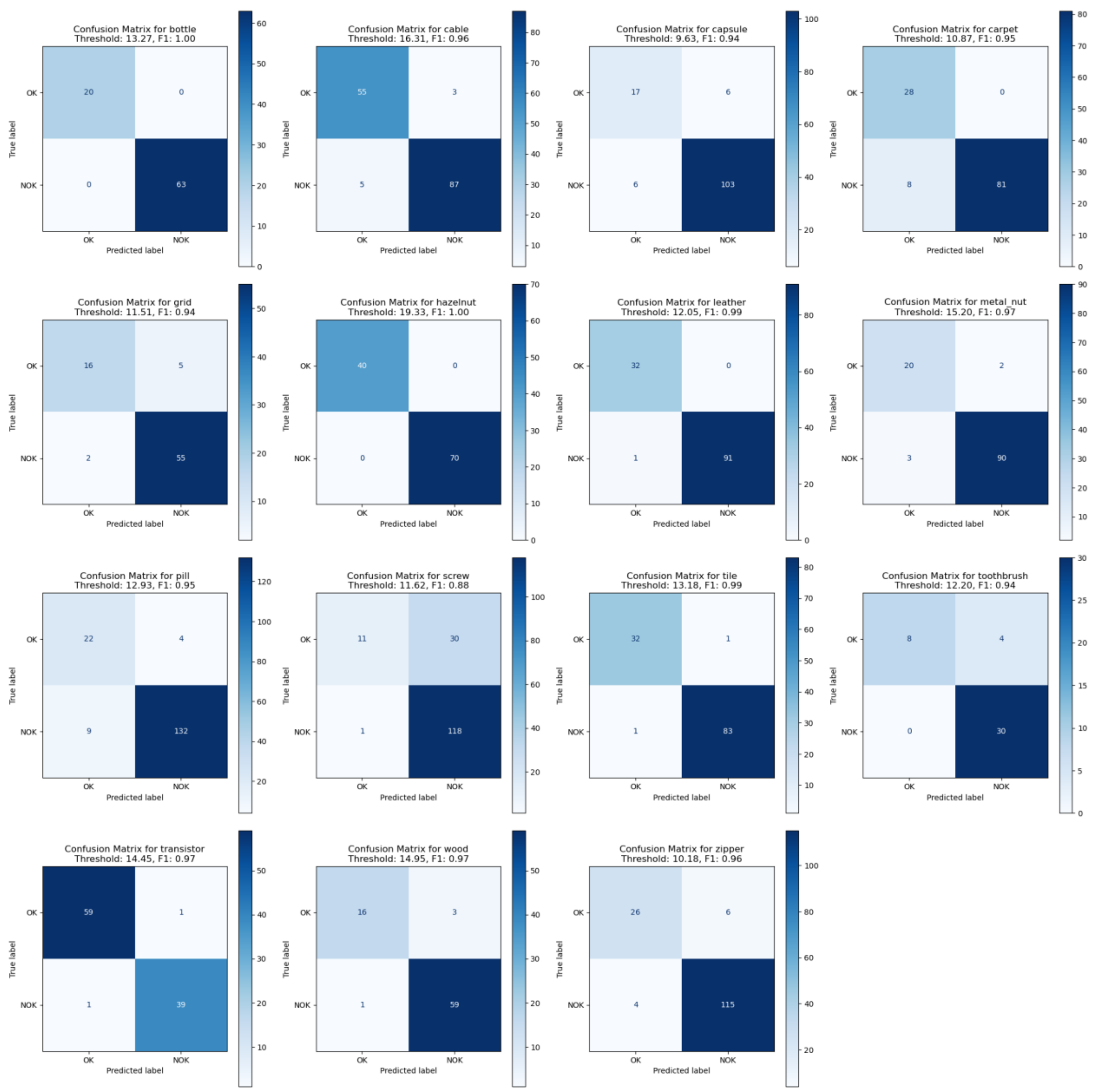
KNN Implementation Details

- **Feature Extraction**:
  - ResNet50's latent layers were used to extract deep feature representations for each sample.
  - The extracted features were stored in a memory bank, acting as a reference set for normal samples.
- **Distance Metric**:
  - Euclidean distance (L2 norm) was chosen to compute similarity between a test sample and normal samples in the memory bank.
  - The lower the distance, the more similar the test sample is to normal data.
- **Number of Neighbors (k)**:
  - k = 5: Each test sample was compared against its 5 nearest neighbors in the memory bank.
  - The anomaly score was computed based on the average distance to these 5 nearest normal samples.
- **Anomaly Score Computation**:
  - For each test sample, distances to all normal samples in the memory bank were computed.
  - The minimum distance (s_star) was selected as the anomaly score.
  - A higher score indicates a sample is more anomalous.
- **Threshold Selection**:
  - To achieve the best separation between normal and anomalous data, we performed a threshold search.

- The threshold was optimized based on the F1-score, balancing precision and recall.

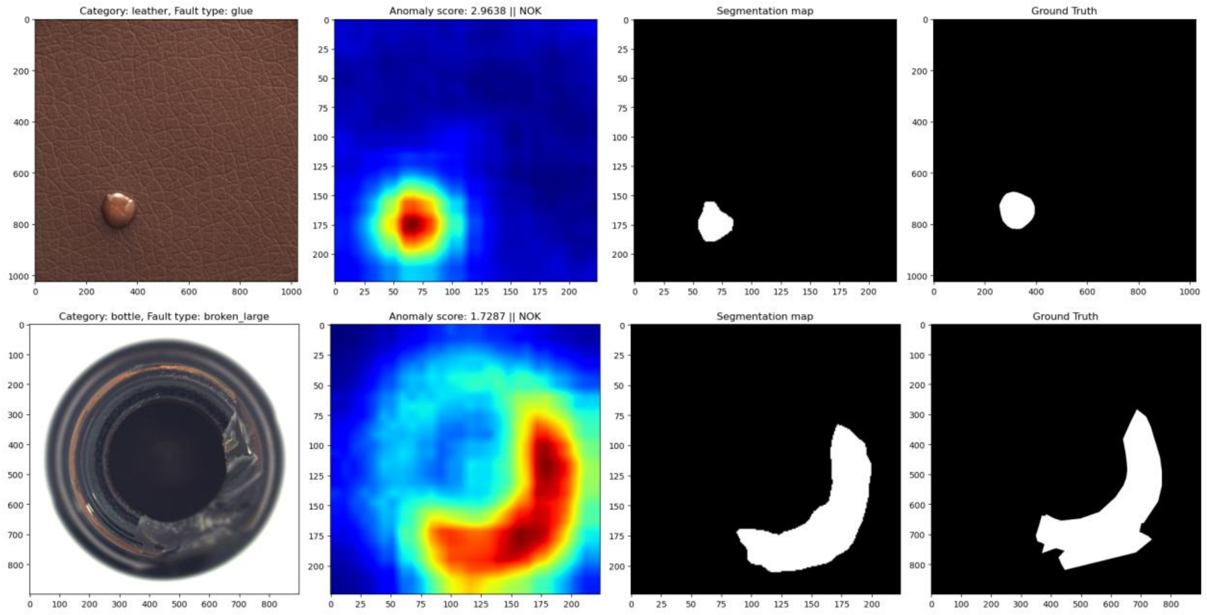
Results





In this k-NN anomaly detection experiment on MVTec, the method demonstrates consistently strong performance across most product categories. Several classes—like bottle, hazelnut, leather, and tile—achieve near-perfect separation between normal and defective samples, as evidenced by AUC values very close to 1.0. Even for categories such as screw and zipper, which have slightly lower AUCs, the model still achieves notably high F1 scores, indicating a solid balance between precision and recall.

A plausible explanation for k-NN's effectiveness is that the chosen feature representations create tight clusters for normal samples, while defective ones end up sufficiently distant in feature space. This makes a local distance-based approach very effective. The primary caveat to keep in mind is that k-NN's performance can be sensitive to the choice of k and the distance metric, and it may become memory-intensive as the dataset grows. Nonetheless, these results show that, under the right conditions, k-NN can serve as an excellent option for industrial anomaly detection tasks.



In the **leather/glue** example (first figure), the method sharply localizes the small glue spot, with a high-intensity region on the heatmap aligning closely to the ground-truth mask. Meanwhile, in the **bottle/broken_large** example (second figure), the algorithm similarly highlights the large broken area in red/yellow, assigning a clearly elevated anomaly score (1.7287) and producing a segmentation that closely corresponds to the ground-truth defect region. Taken together, these examples illustrate robust anomaly localization for both subtle (glue) and more structurally significant (large break) anomalies.

Deep Learning Models

Combinations of different Datasets and Features to compare Model

A Convolutional Neural Network (CNN) is a type of deep learning model designed to process spatially structured data, such as images. CNNs use convolutional layers to extract hierarchical features from input data, enabling them to detect patterns such as edges, textures, and objects. Autoencoders, on the other hand, are neural networks designed for unsupervised learning, primarily used for dimensionality reduction, feature learning, and anomaly detection. A CNN-based autoencoder consists of an encoder and a decoder. The encoder compresses the input into a lower-dimensional latent representation by extracting essential features. The decoder then reconstructs the input from this latent space, ensuring that the meaningful information is retained while discarding noise or redundant data. This structure makes CNN autoencoders highly useful in applications such as image denoising, feature extraction, and anomaly detection.

1. Autoencoder + Custom Resnet3 for Feature Extraction

Why a Custom Architecture?

When each category only provides about 100–200 original images, large networks (like ResNet-50) are likely to overfit, if we want to train from scratch instead of using transfer learning. By creating a smaller “ResNet-3” style encoder—using only a single residual block—you reduce the risk of overfitting, while still benefiting from the improved gradient flow afforded by residual connections.

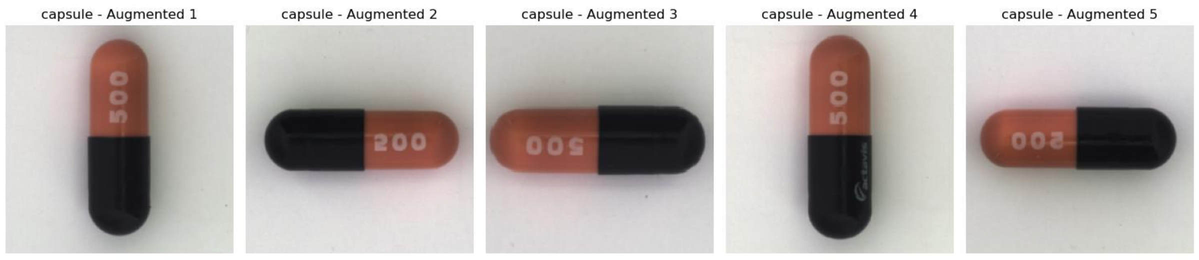
Datasets

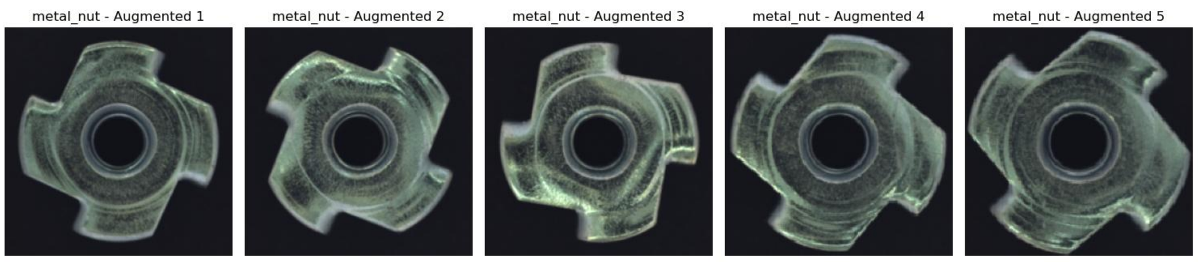
The original database of the MVTec Anomaly Detection Dataset consists of images with three dimensions of fixed size (height * width * channels) for 15 categories. Since models must be trained, tested, and evaluated for each category individually, using separate sets for cross-validation would further reduce the already limited validation data. To address this, instead of splitting the test data for validation, the original training dataset is repurposed as the validation dataset. Meanwhile, a new *augmented dataset* has been created as the training set by applying various augmentation techniques to the original images.

Although each category initially contains a limited number of images, data augmentation has been employed to expand the training set to approximately **1,500** images per category. The augmentation pipeline introduces controlled variations to enhance the model’s generalization ability. The transformations applied include:

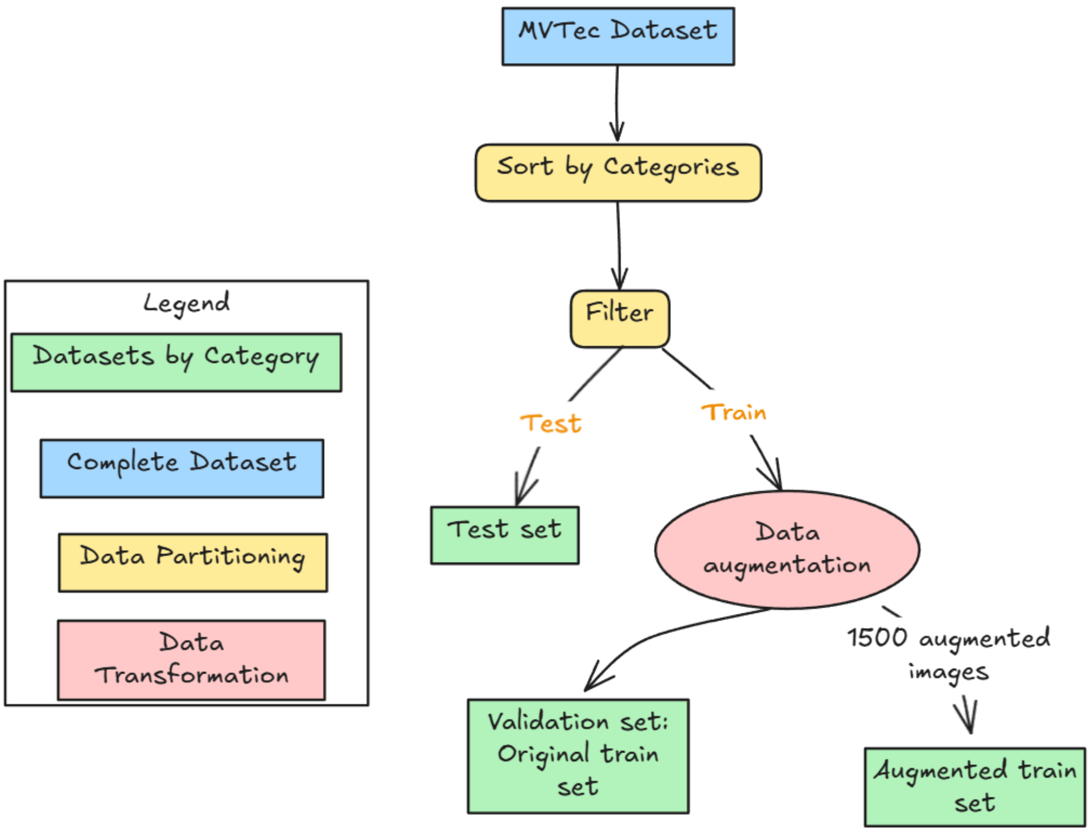
- **Resizing**: Images are resized to a fixed shape of 224×224224 \times 224224×224 pixels.
- **Random Horizontal Flip**: A 50% probability of flipping the image horizontally.
- **Random Scaling & Translation**: A 75% probability of applying random affine transformations with slight translation and scaling variations.
- **Rotation**: A 75% probability of rotating images by -90°, 90°, or 180°.
- **Gaussian Blur**: A slight Gaussian blur with a kernel size of 3 and a randomly chosen sigma between 0.01 and 0.05.
- **Tensor Conversion**: The final step ensures that all images are converted into tensors suitable for deep learning models.

Examples of these augmented images are shown below, demonstrating the various transformations applied (rotations, scaling etc...)

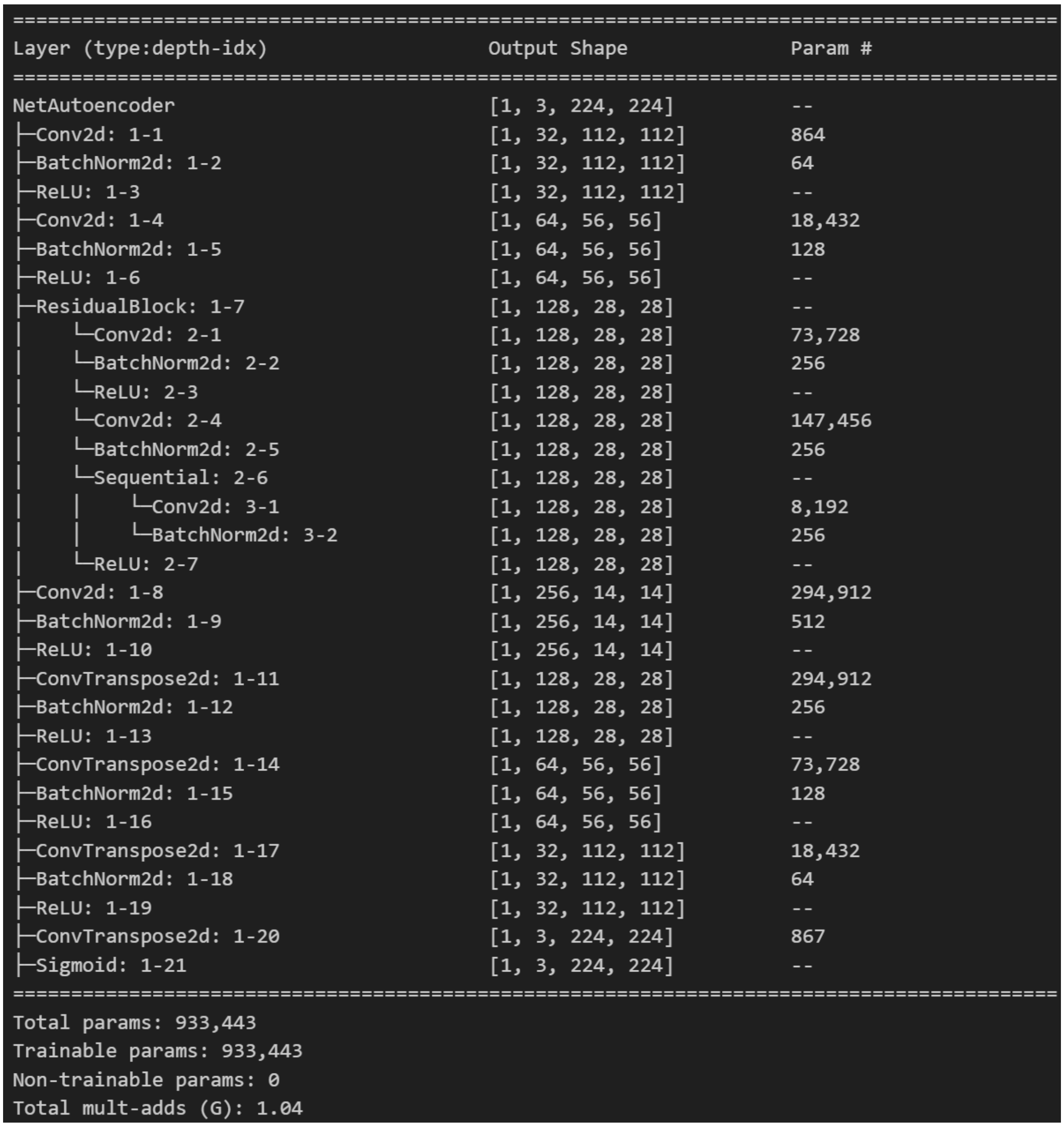




This augmentation strategy increases the diversity of the dataset, allowing the neural network to learn more robust features and improving overall model performance.



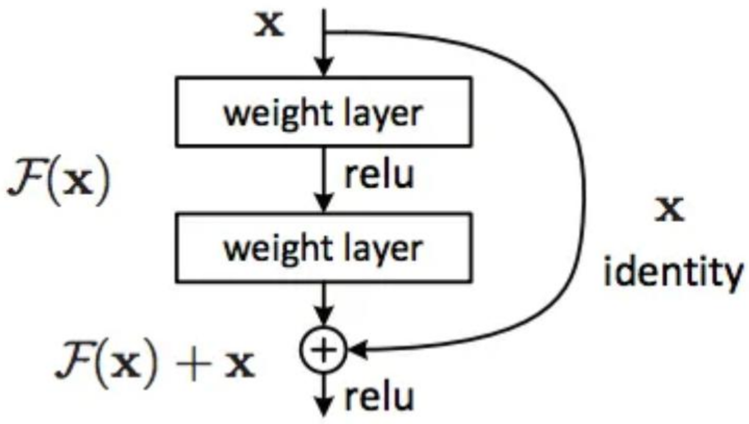
Structure and Number of Layers



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
NetAutoencoder                           [1, 3, 224, 224]          --
├─Conv2d: 1-1                            [1, 32, 112, 112]         864
├─BatchNorm2d: 1-2                       [1, 32, 112, 112]         64
├─ReLU: 1-3                              [1, 32, 112, 112]         --
├─Conv2d: 1-4                            [1, 64, 56, 56]           18,432
├─BatchNorm2d: 1-5                       [1, 64, 56, 56]           128
├─ReLU: 1-6                              [1, 64, 56, 56]           --
├─ResidualBlock: 1-7                     [1, 128, 28, 28]          --
│    └─Conv2d: 2-1                       [1, 128, 28, 28]          73,728
│    └─BatchNorm2d: 2-2                  [1, 128, 28, 28]          256
│    └─ReLU: 2-3                         [1, 128, 28, 28]          --
│    └─Conv2d: 2-4                       [1, 128, 28, 28]          147,456
│    └─BatchNorm2d: 2-5                  [1, 128, 28, 28]          256
│    └─Sequential: 2-6                   [1, 128, 28, 28]          --
│    │    └─Conv2d: 3-1                  [1, 128, 28, 28]          8,192
│    │    └─BatchNorm2d: 3-2             [1, 128, 28, 28]          256
│    └─ReLU: 2-7                         [1, 128, 28, 28]          --
├─Conv2d: 1-8                            [1, 256, 14, 14]          294,912
├─BatchNorm2d: 1-9                       [1, 256, 14, 14]          512
├─ReLU: 1-10                             [1, 256, 14, 14]          --
├─ConvTranspose2d: 1-11                  [1, 128, 28, 28]          294,912
├─BatchNorm2d: 1-12                      [1, 128, 28, 28]          256
├─ReLU: 1-13                             [1, 128, 28, 28]          --
├─ConvTranspose2d: 1-14                  [1, 64, 56, 56]           73,728
├─BatchNorm2d: 1-15                      [1, 64, 56, 56]           128
├─ReLU: 1-16                             [1, 64, 56, 56]           --
├─ConvTranspose2d: 1-17                  [1, 32, 112, 112]         18,432
├─BatchNorm2d: 1-18                      [1, 32, 112, 112]         64
├─ReLU: 1-19                             [1, 32, 112, 112]         --
├─ConvTranspose2d: 1-20                  [1, 3, 224, 224]          867
├─Sigmoid: 1-21                          [1, 3, 224, 224]          --
==========================================================================================
Total params: 933,443
Trainable params: 933,443
Non-trainable params: 0
Total mult-adds (G): 1.04
```

a. **The Single Residual Block**
   - **Residual Connection**
     The core of any ResNet is a residual path that adds the input to the block’s output. This design addresses issues of vanishing or exploding gradients by

making it easier for gradients to backpropagate.


x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
Skip Connection

- **Dimensional Matching**
  When the number of output channels or the stride changes, a 1×1 convolution ensures the input matches the shape required for the final addition step.

b. Encoder (ResNet-3)

- **Basic Convolutions**
  The encoder uses several 3×3 convolutions with stride=2, progressively halving the spatial dimensions.
- **Residual Block**
  A single block is placed in the middle of the encoder to help capture richer features without increasing the network's depth too much.
- **Feature Dimension**
  The final layer maps inputs into a lower-dimensional feature space (e.g., 256 channels), effectively forming the latent representation.

c. Decoder (Upsampling with Transposed Convolutions)

- **Transposed Convolutions**
  The decoder mirrors the encoder by using stride=2 transposed convolutions to gradually restore spatial resolution.
- **Sigmoid Output**
  A final sigmoid activation step restricts the reconstructed image values to the 0–1 range, suitable for image data.

Model Training

a. **Data Splits**
   Train Dataset: Up to 1,500 augmented images per category.
   Validation Dataset: A smaller set of the original, non-augmented training images used to compute the mean squared error (MSE) loss for early stopping.
   Test Dataset: Kept separate; used to assess anomaly detection performance (e.g. via the AUC metric).

b. **Loss and Optimizer**

Loss Function: The network minimizes the MSE between the reconstructed output and the original input.
Cropping: Both reconstructed and original images are cropped slightly (e.g., by a few pixels on each edge) when computing MSE to reduce boundary effects introduced by transposed convolutions.
Optimizer: An adaptive method (e.g., Adam) with a moderate learning rate (like 0.001) helps the model converge without elaborate hyperparameter tuning.

c. **Early Stopping**
Validation Monitoring: After each epoch, the model's performance on the validation set is measured.
Patience Parameter: If no improvement occurs for a specified number of epochs, training halts to prevent overfitting.

d. **Checkpointing**
- **Periodic Saving**: The model's parameters and optimizer states are saved each epoch. If validation performance has improved, the saved checkpoint is labeled as the "best model."
- **Resuming Training**: If needed, training can resume from these checkpoints without losing previous progress.

Testing and AUC Calculation

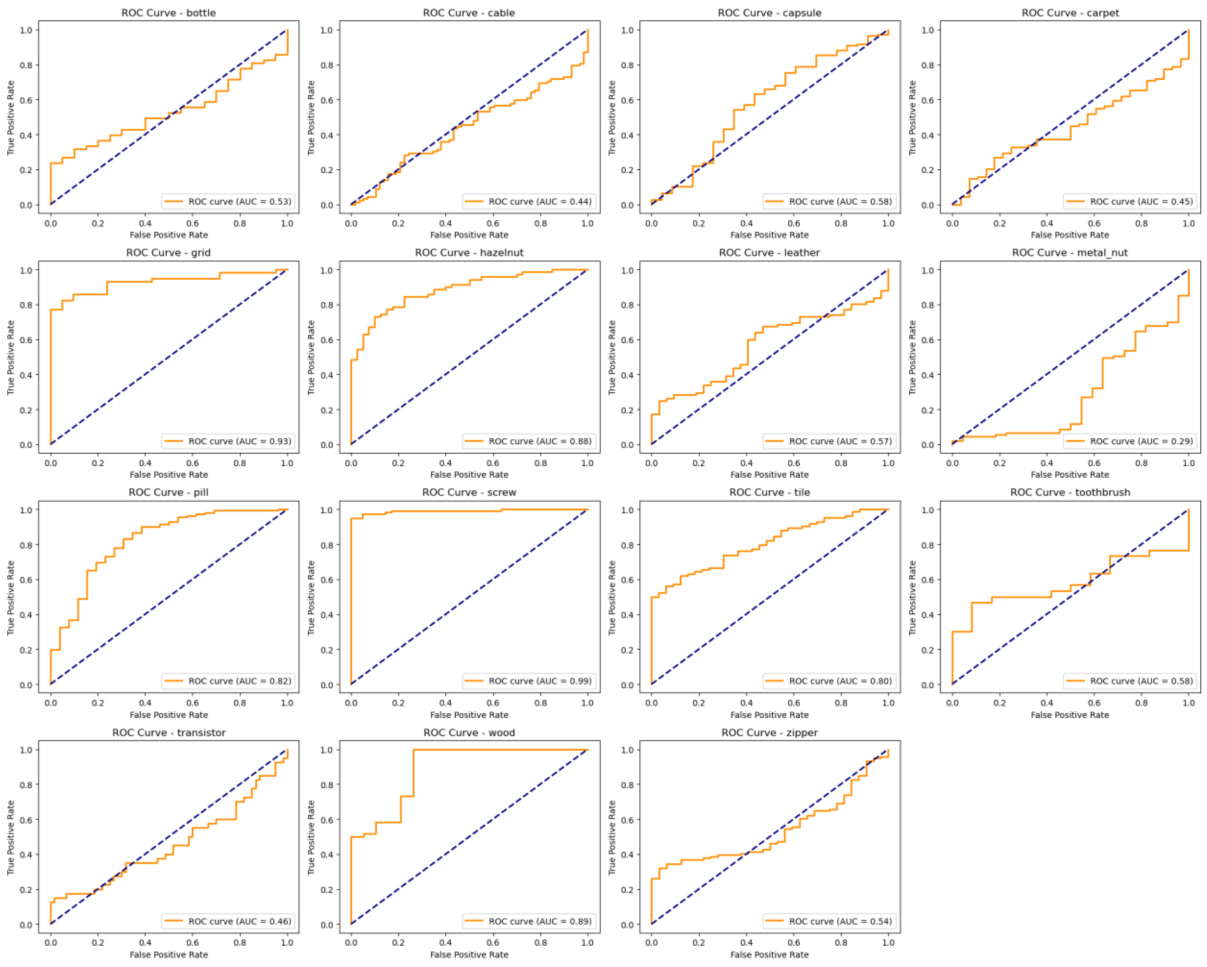
a. **Reconstruction Error as Score**
Each test image's pixel-wise reconstruction error is aggregated into a single anomaly score. Higher reconstruction errors typically indicate more "anomalous" regions.

b. **Thresholding or Ranking**
You can compare the reconstruction error to a threshold or compute a ranking-based metric.

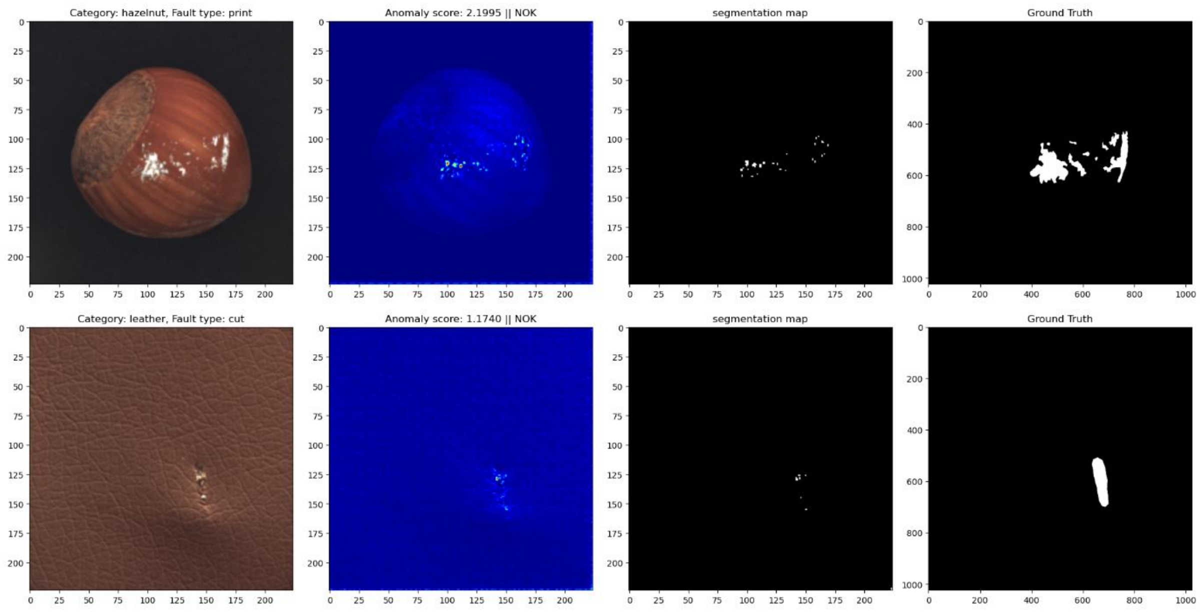
c. **AUC**
By comparing ground-truth labels (normal vs. anomalous) against the reconstruction error scores, you calculate the Area Under the ROC Curve (AUC). This AUC reflects how well the autoencoder distinguishes between normal and anomalous samples.

Results

Overall, the ROC curves show highly variable performance across categories. Some (like **grid**, **hazelnut**, **wood**, and especially **screw**) achieve strong detection capabilities (AUCs above 0.80). Others (such as **metal_nut** and **cable**) are near or below random chance, indicating that the model struggles to differentiate normal from anomalous samples in those classes.



In the example segmentation maps (e.g., **hazelnut** “print” fault, **leather** “cut” fault), the network identifies the rough location of the anomaly regions, but the highlighted areas often only partially align with the true defects. This partial overlap suggests the model is recognizing some—but not all—distinguishing features of the anomalies.

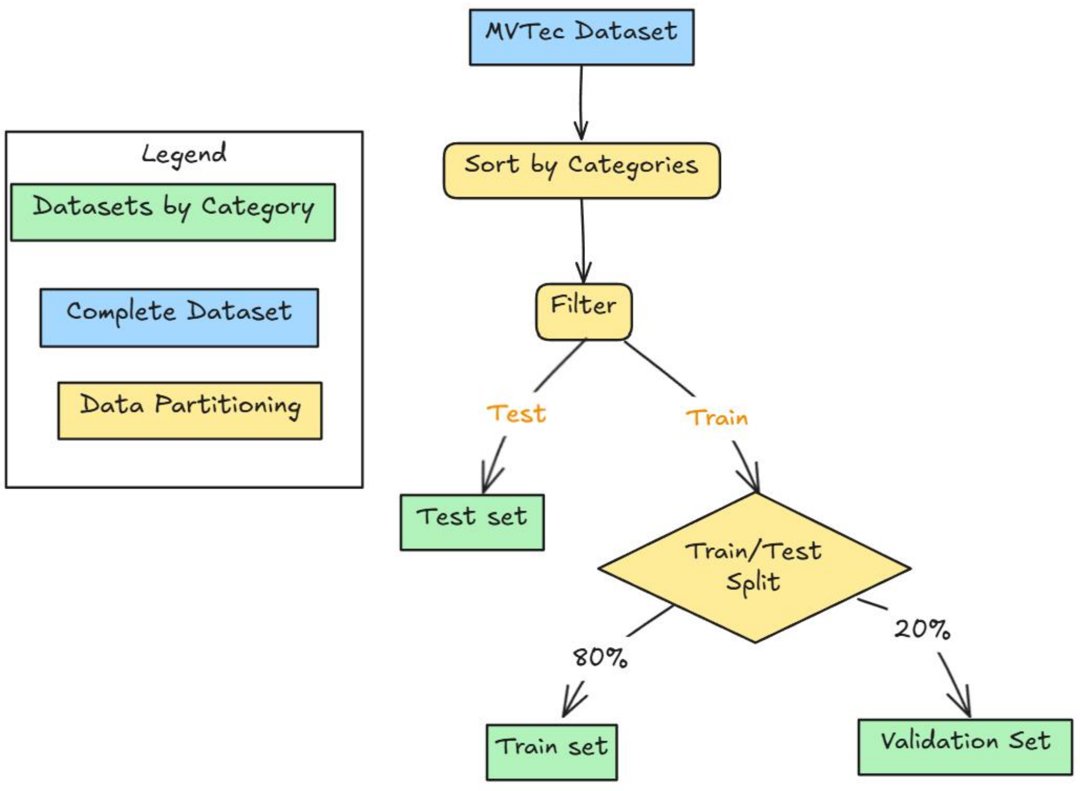

Category: hazelnut, Fault type: print
Anomaly score: 2.1995 || NOK
segmentation map
Ground Truth
Category: leather, Fault type: cut
Anomaly score: 1.1740 || NOK
segmentation map
Ground Truth

Despite a few promising results, the model's overall performance is inconsistent across classes, and it appears to overfit or miss relevant details for certain item types. Ultimately, these results are not satisfactory because the training data is too small to generalize well and capture the full variety of features needed for reliable anomaly detection.

2. Autoencoder + ResNet50 (latent layers) for Feature Extraction

Datasets

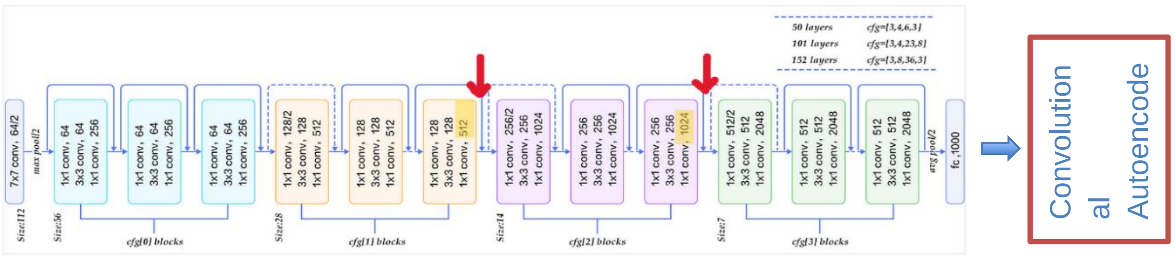
3 datasets have been used:



Feature Extraction using ResNet-50

To extract feature maps from images, a ResNet-50-based feature extractor was employed. ResNet-50 is a deep convolutional neural network known for its ability to learn hierarchical representations through its residual connections, which help mitigate the vanishing gradient problem and enable the training of deeper networks. This architecture has been widely used in computer vision tasks due to its efficiency in capturing both low-level and high-level features. In this work, feature representations were captured from blocks cfg[1] and cfg[2] of ResNet-50. The model was initialized with pre-trained ImageNet weights and set to evaluation mode to prevent updates to the parameters, ensuring that the learned representations remain

unchanged. This approach follows the principles of transfer learning, where a model pre-trained on a large dataset is repurposed for a different but related task, leveraging its learned features without requiring full retraining.
To facilitate feature extraction, hooks were registered on the last blocks of cfg[1] and cfg[2] (see picture below), allowing access to their intermediate feature maps. These extracted feature maps were then processed using average pooling and adaptive resizing to standardize their spatial dimensions. The resulting feature maps were subsequently concatenated along the channel dimension, ensuring a rich and comprehensive representation of the input images. A total of 1536 features were obtained, with 512 features extracted from the block cfg[1] last layer and 1024 features from the block cfg[2] last layer. These feature maps capture different levels of abstraction: earlier layers tend to focus on fundamental image characteristics such as edges and textures, while deeper layers encode more complex semantic information.
The choice of ResNet-50 for feature extraction is motivated by its strong performance on image recognition benchmarks, its efficient residual learning framework, and the availability of pre-trained weights that generalize well across various datasets. By leveraging these pre-trained features through transfer learning, the need for extensive labeled data is reduced, making the model more adaptable to new tasks with minimal fine-tuning.



Convolutional Autoencoder (CAE) for Feature Reconstruction

A Convolutional Autoencoder (CAE) was designed to learn feature representations in a lower-dimensional space and reconstruct them. The encoder progressively reduced input feature map dimensions using convolutional layers, while the decoder mirrored this process to restore the input. Batch normalization and ReLU activation were applied to stabilize activations and introduce non-linearity for better learning.
Feature maps were extracted from intermediate layers, processed with average pooling and adaptive resizing for consistency, and then concatenated for downstream tasks. The model leveraged transfer learning by initializing with pre-trained ImageNet weights, ensuring efficient feature extraction without full retraining.

- **Key Components**
    - **Convolutional Layers**: Extract key spatial features and progressively reduce dimensions to form compact latent representations, capturing both fine and coarse details.
    - **Batch Normalization**: Stabilizes training and reduces internal covariate shifts.
    - **ReLU Activation**: Introduces non-linearity to enable complex pattern learning.
    - **Pooling Operations**: Uses average pooling to maintain feature size consistency and emphasize dominant patterns.
    - **Adaptive Resizing**: Ensures uniform feature dimensions for concatenation.

    - **Transfer Learning**: Utilizes pre-trained weights from ImageNet for enhanced feature extraction.
    - **MSE Loss Function**: Measures reconstruction error to guide model optimization.

- **Architecture**

The CAE was designed with a structured depth to balance feature extraction and reconstruction quality.

Encoder:

- Three convolutional layers progressively reduce spatial dimensions while increasing feature channels.
- The first layer captures low-level features, while deeper layers extract more complex patterns.
- Batch normalization and ReLU activation are applied after each convolution.

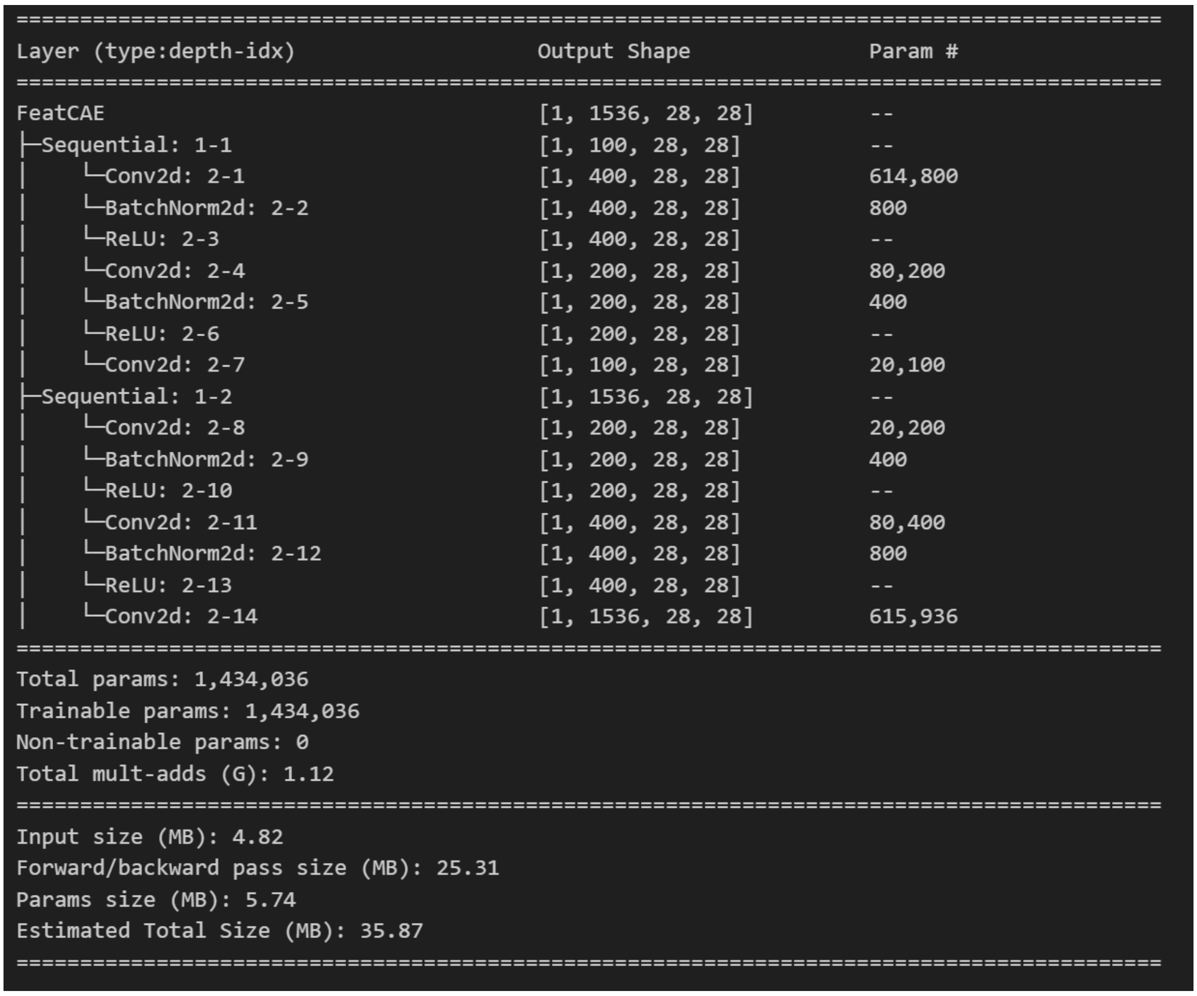
Bottleneck:

- Represents the compressed latent space where essential feature representations are stored with fewer parameters.

Decoder:

- Mirrors the encoder with three transposed convolutional layers to reconstruct the input.
- The first decoder layer expands basic structures, while deeper layers refine details.
- The final layer reconstructs the original feature map dimensions with minimized loss.

This structured approach ensures that the CAE learns meaningful representations efficiently while preserving essential details. The architecture was designed to balance computational efficiency and expressiveness, maintaining high-quality feature extraction and reconstruction. Training aimed to minimize reconstruction error using Mean Squared Error (MSE) loss, optimizing the model for further analysis, classification, or clustering tasks.



```
==========================================================================================
Layer (type:depth-idx)                   Output Shape              Param #
==========================================================================================
FeatCAE                                  [1, 1536, 28, 28]         --
├─Sequential: 1-1                        [1, 100, 28, 28]          --
│    └─Conv2d: 2-1                       [1, 400, 28, 28]          614,800
│    └─BatchNorm2d: 2-2                  [1, 400, 28, 28]          800
│    └─ReLU: 2-3                         [1, 400, 28, 28]          --
│    └─Conv2d: 2-4                       [1, 200, 28, 28]          80,200
│    └─BatchNorm2d: 2-5                  [1, 200, 28, 28]          400
│    └─ReLU: 2-6                         [1, 200, 28, 28]          --
│    └─Conv2d: 2-7                       [1, 100, 28, 28]          20,100
├─Sequential: 1-2                        [1, 1536, 28, 28]         --
│    └─Conv2d: 2-8                       [1, 200, 28, 28]          20,200
│    └─BatchNorm2d: 2-9                  [1, 200, 28, 28]          400
│    └─ReLU: 2-10                        [1, 200, 28, 28]          --
│    └─Conv2d: 2-11                      [1, 400, 28, 28]          80,400
│    └─BatchNorm2d: 2-12                 [1, 400, 28, 28]          800
│    └─ReLU: 2-13                        [1, 400, 28, 28]          --
│    └─Conv2d: 2-14                      [1, 1536, 28, 28]         615,936
==========================================================================================
Total params: 1,434,036
Trainable params: 1,434,036
Non-trainable params: 0
Total mult-adds (G): 1.12
==========================================================================================
Input size (MB): 4.82
Forward/backward pass size (MB): 25.31
Params size (MB): 5.74
Estimated Total Size (MB): 35.87
==========================================================================================
```

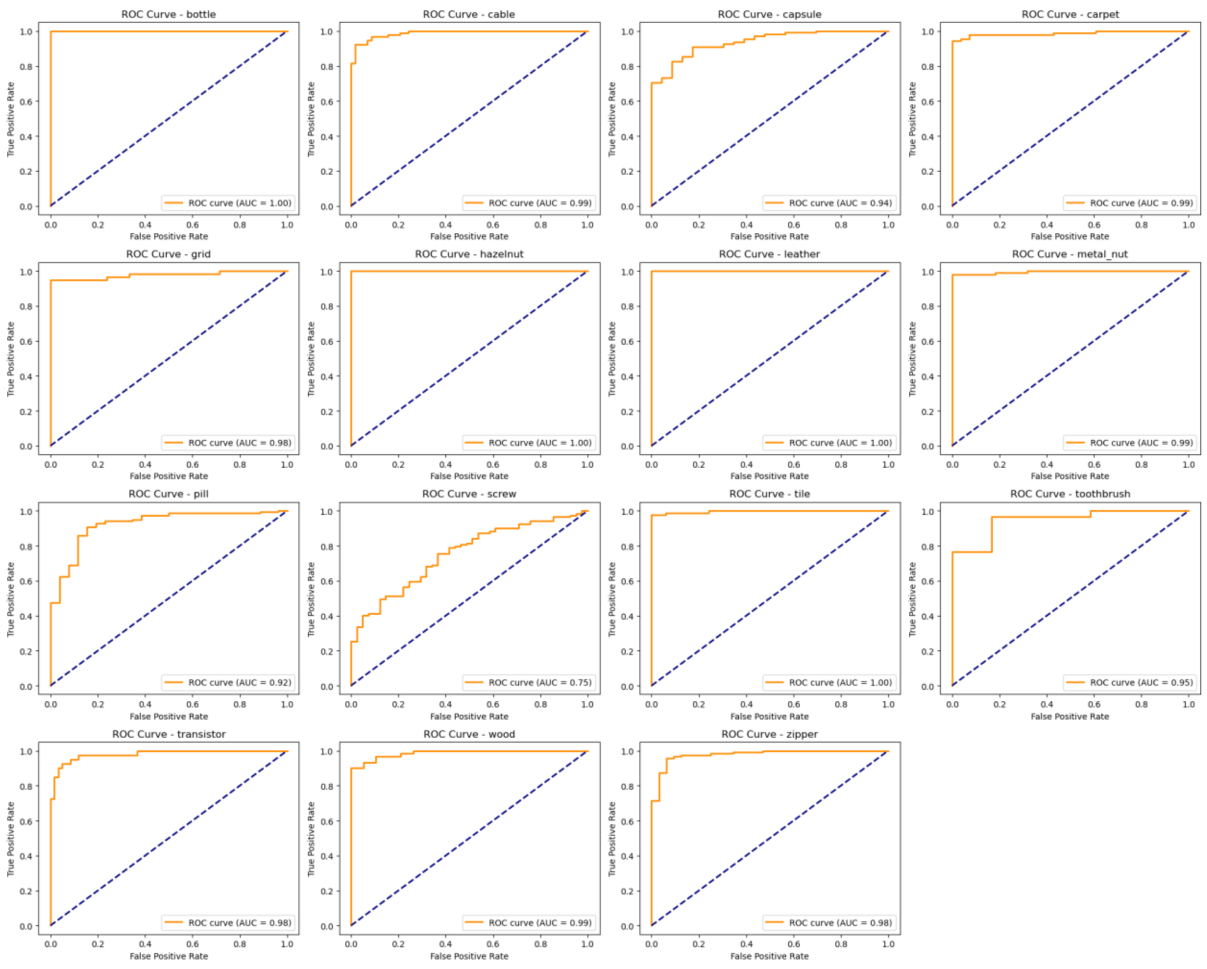
Model Training

The autoencoder was trained using an Adam optimizer with a learning rate of 0.001. The training process included early stopping to prevent overfitting, where validation loss was monitored over epochs. If no improvement in validation loss was observed for two consecutive epochs, training was halted. The training loss and validation loss were recorded for each epoch to track the model's learning process.

Threshold Selection for Anomaly Detection

After training, the reconstruction error was analysed for each category to determine an optimal anomaly detection threshold. The threshold was set based on the mean reconstruction error plus three standard deviations, ensuring that anomalous samples were effectively identified. Histograms of reconstruction errors were plotted, and threshold values were marked to visualize the separation between normal and anomalous data points.

Results

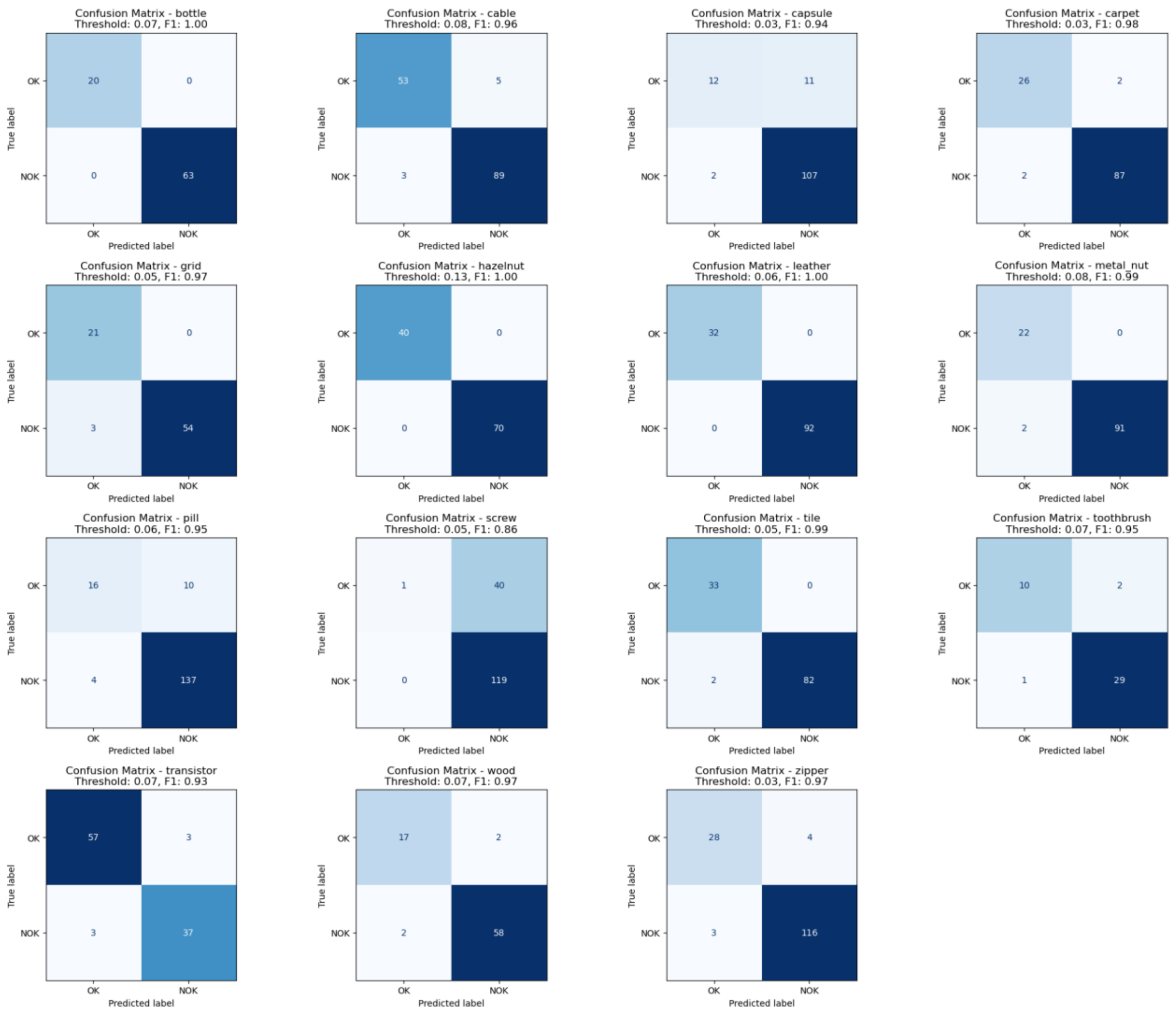


The ROC curves presented in the image provide an evaluation of the model's performance in distinguishing between normal and anomalous samples across different categories. The key observations are:

I. **High AUC Scores:**
   a. Most categories exhibit an Area Under the Curve (AUC) value close to 1.00, indicating excellent performance.
   b. Categories such as *bottle, hazelnut, leather, and tile* have perfect AUC scores of 1.00, suggesting flawless discrimination.

II. **Moderate Performance for Certain Categories:**
   a. The *screw* category has an AUC of 0.75, which is significantly lower than others. This indicates that the model struggles with distinguishing anomalies in screws.
   b. The *capsule* and *pill* categories show slightly lower AUC values (0.94 and 0.92), meaning there is some overlap between false positives and true negatives.

III. **Strong Generalization Across Categories:**
   a. The consistent high AUC scores suggest that the autoencoder-based feature extraction and anomaly detection pipeline generalizes well across various types of industrial components.
   b. This indicates that the chosen feature representation and threshold selection were effective.



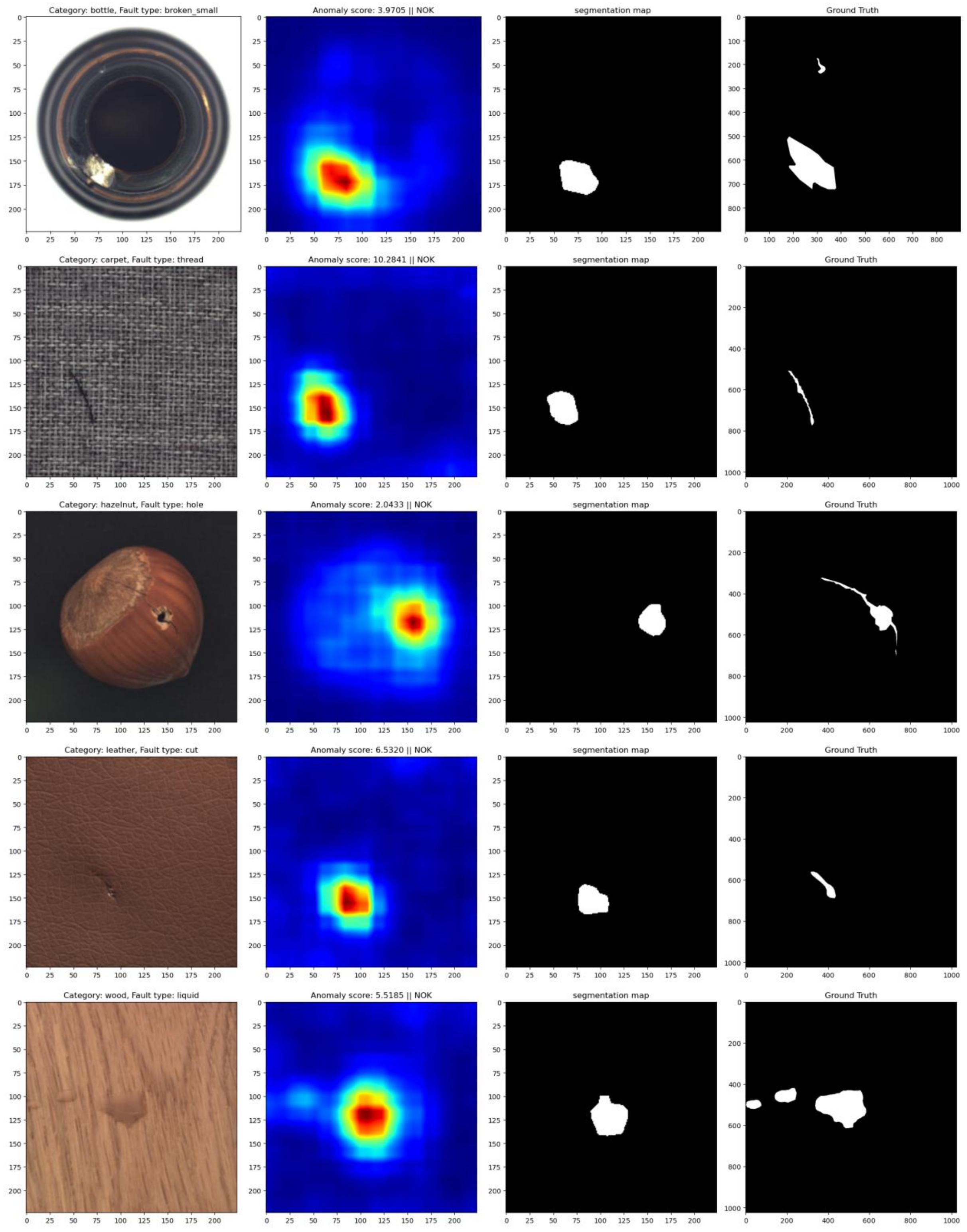
The confusion matrices in the image provide a detailed evaluation of the model's classification performance for different categories. Key observations include:

I. **High Classification Performance for Most Categories:**
   a. The *bottle, hazelnut, leather,* and *tile* categories exhibit perfect classification with an F1-score of 1.00, meaning there were no misclassifications.
   b. Other categories such as *metal nut, carpet, and transistor* also achieved high F1-scores (>0.95), indicating strong generalization.

II. **Moderate Misclassification in Some Categories:**
   a. The *capsule* category has a relatively higher number of false positives (11) compared to the other categories, impacting its F1-score (0.94).
   b. The *screw* category has the lowest F1-score (0.86), showing more significant misclassification errors, particularly with false negatives.

III. **Class Imbalance Considerations:**
   a. Some categories have a lower number of samples (e.g., *capsule, toothbrush*), which could affect generalization.
   b. Categories with more data points (*pill, screw, wood*) still performed well overall.

Anomaly Detection Visualization



To evaluate the model's performance, anomaly detection results were visualized using defect detection heatmaps and segmentation maps. Each row in the visualization represents a different defect category and consists of:

1. The original input image showing the defective object.
2. A heatmap indicating the anomaly score, where red areas correspond to higher detected anomalies.

3. A segmentation map highlighting the detected defective regions.
4. The ground truth mask showing the actual defect locations.

Observations

- The model successfully localized defects such as small breaks in bottles, scratches on carpets, holes in hazelnuts, and cuts on leather.
- The heatmaps strongly correspond to the defective regions, confirming the effectiveness of feature extraction and anomaly segmentation.
- Some minor misalignments between the segmentation maps and the ground truth masks suggest areas for potential refinement.

Overall, the anomaly detection pipeline demonstrated strong localization ability and effective defect segmentation, making it highly suitable for real-world quality control applications.

Conclusion

The evaluation of various models for anomaly detection revealed that ResNet50-based feature extraction using the second and third latent layers, combined with k-Nearest Neighbors (KNN), and deep learning-based ResNet50 with an autoencoder were the two best-performing models. These approaches consistently achieved high AUC-ROC scores across different categories, demonstrating their superior capability to distinguish between normal and anomalous data.

Key takeaways from the study:

- Deep feature extraction significantly outperforms manual features, as seen in the consistently better performance of ResNet50-based methods.
- Combining multiple latent layers (1st & 2nd block last layers) enhances feature representation, leading to improved anomaly detection performance.
- The k-NN approach leverages memory-based similarity comparisons, providing a robust alternative to parametric models.
- Deep learning with an autoencoder provides strong reconstruction-based anomaly detection, ensuring comprehensive feature learning.
- Synthetic validation data introduced diversity but also posed challenges, as some anomalies may not perfectly mimic real-world defects.

Future Improvements

Despite the promising results, several areas could be further improved:

- Data Augmentation Refinement: Exploring more advanced augmentation techniques, such as generative models (e.g., GANs), could enhance training diversity.
- Model Ensembling: Combining multiple anomaly detection models could improve robustness and generalization.
- Hyperparameter Optimization: Fine-tuning model hyperparameters further, using techniques like Bayesian optimization, could boost performance.
- Alternative Architectures: Exploring transformer-based architectures or vision encoders like ViTs for anomaly detection could yield even better results.
- Better Synthetic Data: Refining the process of synthetic anomaly generation to better align with real-world defects.

Overall,

- **“ResNet50 (1st & 2nd block last layers) + KNN”**
- **“ResNet50 (1st & 2nd block last layers) + Autoencoder”**

approaches emerged as the best-performing models, providing a solid foundation for industrial anomaly detection tasks. Further optimizations and explorations in feature extraction and model selection could push performance even higher in future studies.

Appendix

Direct Features computed from Image Values

1. Image Metadata

- category: One of 15 image categories in the MVTec AD Dataset.
- subclass: One of four subclasses that group the categories.
- set_type: Train or test data as provided in the MVTec AD Dataset.
- anomaly_status: Binary variable indicating if the image is anomalous or normal (target variable for anomaly detection).
- anomaly_type: One of several anomaly types specific to each category, or 'none' for normal images.

2. Image Dimensions and Aspect Ratio

- width / height: Image size in pixels.
- aspect_ratio: Proportion of width to height.

3. Pixel Statistics

3.1. Color Channel Summaries

- num_pixels_b / num_pixels_g / num_pixels_r: Sum of pixel values per channel (blue, green, red).
- brightness_b / brightness_g / brightness_r: Mean pixel value per channel.
- contrast_b / contrast_g / contrast_r: Standard deviation of pixel values per channel.
- luminance: Weighted sum of pixel brightness (perceived brightness measure).

4. Structural and Edge Features

- edge_density: Amount of edges detected in the image, normalized by image size.
- h_symmetry / v_symmetry: Symmetry measures along horizontal and vertical axes.

5. Gradient-Based Features

- mean_grad_x / std_grad_x: Mean and standard deviation of gradients along the x-direction.
- mean_grad_y / std_grad_y: Mean and standard deviation of gradients along the y-direction.

6. Object and Contour-Based Features

- object_count: Number of contours found in the image.
- bounding_box_area: Area of the bounding box surrounding the largest detected contour.
- bounding_box_aspect_ratio: Aspect ratio of the bounding box.
- bounding_box_x / bounding_box_y: Coordinates of the bounding box's top-left corner.
- bounding_box_width / bounding_box_height: Width and height of the bounding box.
- perimeter: Perimeter length of the largest detected contour.

7. Intensity and Texture Features

- mean_intensity / variance_intensity: Mean and variance of pixel intensity.
- mode_intensity / median_intensity: Mode and median pixel intensity.
- entropy: Shannon entropy of the grayscale image (measures randomness).

8. Frequency Domain Features (Fourier Transform)

- mean_magnitude / variance_magnitude: Mean and variance of absolute pixel values after Fourier transformation of the grayscale image.

9. Geometric and Topological Features

- line_length: Total length of detected lines in the image.
- blob_count: Number of significantly intense regions detected by the Laplace transformation.
- region_homogeneity: Mean homogeneity in image regions detected using the watershed algorithm.

10. Noise and Signal Quality

- noise_level: Estimated Gaussian noise level in the image.
- signal_to_noise_ratio: Ratio of signal strength to noise level.

11. Intensity Quantiles

- quantiles_intensity_25: 25th percentile of pixel intensity.
- quantiles_intensity_50: 50th percentile (median) of pixel intensity.
- quantiles_intensity_75: 75th percentile of pixel intensity.

12. Raw Pixel Data

- pixel_1 to pixel_150528: All pixel values of an image resized to 224 × 224 × 3 (RGB), stored sequentially.

Bibliography

1. Liu, J., Xie, G., Wang, J., Li, S., Wang, C., Zheng, F., & Jin, Y. (2023). **Deep Industrial Image Anomaly Detection: A Survey**. Springer Nature. Retrieved from arXiv:2301.11514.
2. Yang, J., Shi, Y., & Qi, Z. (2020). **DFR: Deep Feature Reconstruction for Unsupervised Anomaly Segmentation**. Retrieved from [arXiv:2012.07122].
3. Bühler, J., Fehrenbach, J., Steinmann, L., Nauck, C., & Koulakis, M. (2024). **Domain-independent detection of known anomalies**. Karlsruhe Institute of Technology (KIT) & preML GmbH. Retrieved from [arXiv:2407.02910].
4. Heckler, L., & König, R. (2024). **Feature Selection for Unsupervised Anomaly Detection and Localization Using Synthetic Defects**. MVTec Software GmbH & Technical University of Munich. In *Proceedings of the 19th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP 2024)*, 154-165. DOI: 10.5220/0012385500003660.
5. Rippel, O., Mertens, P., & Merhof, D. (2020). **Modeling the Distribution of Normal Data in Pre-Trained Deep Features for Anomaly Detection**. RWTH Aachen University. Retrieved from arXiv:2005.14140.
6. Bergmann, P., Fauser, M., Sattlegger, D., & Steger, C. (2019). **MVTec AD – A Comprehensive Real-World Dataset for Unsupervised Anomaly Detection**. MVTec Software GmbH. Retrieved from www.mvtec.com.
7. Roth, K., Pemula, L., Zepeda, J., Schölkopf, B., Brox, T., & Gehler, P. (2022). **Towards Total Recall in Industrial Anomaly Detection**. University of Tübingen & Amazon AWS. Retrieved from [arXiv:2106.08265].
8. Zheng, Y., Wang, X., Qi, Y., Li, W., & Wu, L. (2022). **Benchmarking Unsupervised Anomaly Detection and Localization**. University of Chinese Academy of Sciences, SenseTime Research, & Tsinghua University. Retrieved from [arXiv:2205.14852].