technische universität
dortmund

Arbeitsgebiet
Bildsignalverarbeitung

Master Thesis

Anomaly Detection on Diverse Kitchen Surfaces: A Case Study Using Industry Data

Shakib Neman

Student Nr.: 236762

Degree: Master of Science in Automation and Robotics

Supervisors:

Prof. Dr. rer. nat. habil. Christian Wöhler

M.Sc. Tom Sander

Technische Universität Dortmund

AG Bildsignalverarbeitung

Prof. Dr. rer. nat. habil. Christian Wöhler

Abstract

Anomaly detection is crucial for maintaining quality control in industrial manufacturing, ensuring that defective products are identified before reaching consumers. Deep learning-based unsupervised anomaly detection methods, such as PatchCore by Amazon AWS and CutPaste by Google Cloud AI Research, have demonstrated strong performance on benchmark datasets like MVTec. However, their effectiveness in real-world industrial settings remains an open challenge. This thesis evaluates and compares these two methods for detecting surface defects in kitchen boards using high-resolution images from an industrial production environment.

The dataset consists of images from a kitchen carpentry production facility. The analysis focuses on two categories, White and White with Edges. To process the data efficiently, a patch-based approach was implemented, extracting and normalizing image patches to enable localized anomaly detection. PatchCore utilizes a memory bank approach for anomaly detection, while CutPaste employs self-supervised learning with synthetic anomaly generation. Both methods were modified to accommodate the industrial dataset, incorporating custom data loaders, augmented training strategies, and visualization techniques for qualitative analysis.

Experiments were conducted using ResNet18 and ResNet50 backbones with and without blur preprocessing. Results indicate that PatchCore achieved a higher ROC-AUC (up to 89.06%) and precision, making it preferable for high-precision defect detection. Additionally, PatchCore produced clearer and more interpretable anomaly maps compared to CutPaste.

This work bridges the gap between cutting-edge anomaly detection research and real-world industrial applications, providing insights into the deployment of deep learning-based quality control solutions in manufacturing.

Contents

CHAPTER 1

Introduction

Anomaly detection using deep learning is a widely applied area of artificial intelligence, with applications spanning various fields such as medical diagnosis, fraud detection in cybersecurity, and identifying defects in manufacturing. In the context of manufacturing, anomaly detection is crucial for ensuring product and material integrity before products reach customers. It involves identifying deviations from expected patterns, detecting the ”unnatural” within the ”natural,” which helps prevent defects and maintain high-quality standards.

This thesis explores two state of the art, unsupervised deep learning-based anomaly detection algorithms: PatchCore (Roth et al., 2022), developed by Amazon AWS, and CutPaste (Li et al., 2021), introduced by Google Cloud AI Research. These algorithms were presented at the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) in 2022 and 2021, respectively. These algorithms will not be applied to benchmarking data sets like MVtec, where they already proved their performance, as the data set is perfectly configured for anomaly detection, but to images taken from industry captured in a kitchen carpentry production factory with the goal of detecting anomalies in the boards surfaces. The anomalies in question are often very small relative to the size of the boards and are not easily detectable by the naked eye. The aim of this thesis is first to evaluate the effectiveness of these algorithms in identifying these subtle defects, as well as satisfying the need for producing high quality defect free surfaces, thereby maintaining the high-quality standard in manufacturing.

By addressing these goals, this work aims to bridge the gap between cutting-edge research and practical industrial applications, thereby advancing the capabilities of anomaly detection in manufacturing.

CHAPTER 2

Theoretical Background

To be able to understand and follow up with all the information and techniques mentioned, we provide a short background information that are essential to understand the further work in the thesis.

2.1 Learning Approaches in Machine Learning

Machine Learning (ML) involves different types of learning methods, depending on the availability and nature of labeled data. In this section, we discuss three types of learning: **Supervised Learning**, **Unsupervised Learning**, and **Self-Supervised Learning**.

2.1.1 Supervised Learning

Supervised learning relies on training a model using a dataset where each input is paired with a corresponding label. The goal is for the model to learn a relationship between the inputs and the labels so it can make predictions on new, unseen data. For example, in image classification, the model might be trained on labeled examples (e.g., images labeled as cat, dog, or car) and then used to classify new images.

K-Nearest Neighbors

An example of supervised learning is K-Nearest Neighbor (KNN) (Fix and Hodges, 1951), which is a simple, non-parametric, and instance-based learning algorithm commonly used for classification and regression tasks. It is highly intuitive and relies on the proximity of data points in feature space to make predictions.
The KNN algorithm assumes that similar instances exist in close proximity to each other in the feature space. Given a data point, KNN identifies its k nearest neighbors and makes predictions based on their labels (for classification) or values (for

regression).
For a given input sample x, the algorithm works as follows:

1. Compute the distance between x and all points in the training dataset (which is very computationally intensive). Common distance metrics include:
   - Euclidean distance:

$$d(x, x_i) = \sqrt{\sum_{j=1}^{n} (x_j - x_{ij})^2}$$

2. Identify the k nearest neighbors to x.
3. Assign the majority class label of the k neighbors to x.
4. For regression: Compute the average (or weighted average) of the values of the k neighbors.

Key Parameters of KNN

- **k**: Number of neighbors considered. A smaller k captures local structures but may be sensitive to noise, while a larger k generalizes better at the cost of increased computational complexity.
- **Distance Metric**: Defines how similarity is measured. Euclidean distance is the most common, but others like cosine similarity or Mahalanobis distance can be used depending on the data.
- **Weighting Scheme**: Neighbors can be uniformly weighted or weighted inversely to their distance from the query point.

2.1.2 Unsupervised Learning

Unsupervised learning works with data that does not have explicit labels. The main goal is to uncover hidden patterns, structures, or distributions within the data. This type of learning is commonly used for clustering and dimensionality reduction.
For instance, clustering algorithms such as K-Means can group similar data points together based on their features without needing predefined labels.

Kernel Density Estimation

Kernel Density Estimation (KDE) (Chen, 2017) is a non-parametric Unsupervised technique for estimating the probability density function (PDF) of a dataset. Unlike parametric methods that assume a specific underlying distribution, KDE provides a flexible way to infer the data distribution directly from the sample points.

KDE works by summing kernel functions K centered at each data point. The estimated PDF at a point x is given by:

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

where:

- $\hat{f}(x)$: Estimated PDF at x,
- n: Number of data points,
- h: Bandwidth (a smoothing parameter),
- x_i: Data points,
- K: Kernel function (e.g., Gaussian kernel, Epanechnikov kernel).

The choice of kernel K and bandwidth h significantly affects the smoothness and accuracy of the estimation.

KDE is widely used in anomaly detection, clustering, and other tasks where understanding the underlying data distribution is critical.

Gaussian Density Estimation

Gaussian Density Estimation (GDE) (Rippel et al., 2020) is a parametric method for estimating the PDF of a dataset, assuming the data follows a Gaussian (normal) distribution.

GDE models the data distribution using the mean vector μ and covariance matrix Σ. The PDF for a point x in d-dimensional space is given by:

$$f(x) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu)^\top \Sigma^{-1}(x - \mu)\right)$$

where:

- μ: Mean vector,
- Σ: Covariance matrix,
- $|\Sigma|$: Determinant of the covariance matrix,
- Σ^{-1}: Inverse of the covariance matrix.

GDE assumes that the data follows a Gaussian distribution, which may not be valid for complex datasets. As a result, its performance can degrade significantly when the underlying data distribution is non-Gaussian or multimodal, limiting its flexibility and applicability in certain scenarios.

GDE is commonly used in anomaly detection, where anomalies are identified as points with low probability under the estimated Gaussian distribution.

2.1.3 Self-Supervised Learning

Self-Supervised Learning (SSL) trains models to discover patterns in unlabeled data by solving synthetic problems derived from the data itself. These problems generate pseudo-labels, enabling the model to learn representations that can be applied to downstream tasks like classification or anomaly detection.
For example, in anomaly detection, a model could be trained to reconstruct images from corrupted versions (e.g., with random noise or missing sections). By learning how to reconstruct normal patterns, the model becomes sensitive to irregularities, which helps it identify anomalies in unseen data.

2.2 Artificial Neural Networks

Artificial Neural Network (ANN) (Goodfellow et al., 2016) are computational models inspired by the biological neural networks of the human brain. They consist of interconnected layers of nodes, called neurons, which process input data through mathematical transformations to produce outputs. This capability to model complex relationships makes ANNs highly versatile for various machine learning ML tasks, including classification, regression, and anomaly detection.

2.2.1 Structure of ANNs

ANNs are composed of three main types of layers:

- **Input Layer**: Receives raw input data. Each neuron corresponds to a feature in the input dataset.
- **Hidden Layers**: Process the data using weights, biases, and activation functions to capture patterns and representations. The depth and size of these layers determine how much the network can learn and represent complex relationships in the data.
- **Output Layer**: Produces the final prediction, such as class probabilities or regression outputs.

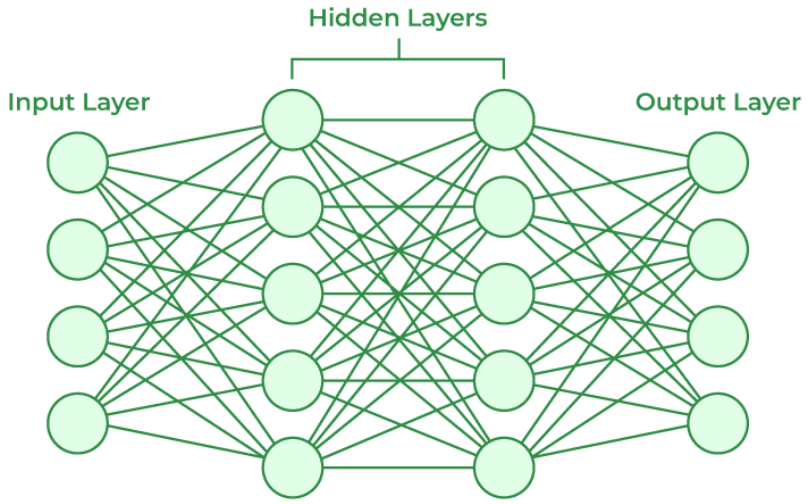


Figure 2.1: Structure of an Artificial Neural Network (GeeksforGeeks, 2024)

Each neuron in the network computes a weighted sum of its inputs, adds a bias term, and applies an activation function to introduce non-linearity. This transformation enables the network to approximate complex functions (Hornik et al., 1989). Mathematically, the output of a neuron j in layer l is calculated as:

$$z_j^{(l)} = \sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)}$$

$$a_j^{(l)} = \sigma\left(z_j^{(l)}\right)$$

where:

- $w_{ij}^{(l)}$: Weight connecting neuron i in layer $l-1$ to neuron j in layer l,
- $b_j^{(l)}$: Bias of neuron j in layer l,
- σ: Activation function (e.g., Rectified Linear Unit (ReLU), Sigmoid),
- $a_i^{(l-1)}$: Activation of neuron i in the previous layer.

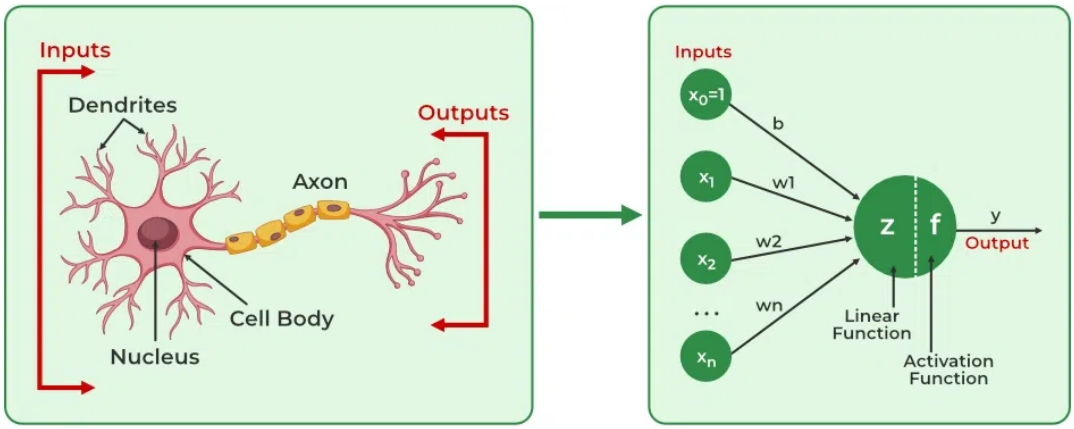


Figure 2.2: Basic Structure of a Neural Network Neuron (GeeksforGeeks, 2024)

2.3 Backpropagation Algorithm

The **backpropagation algorithm** is a key method for training ANNs. It enables the network to iteratively improve its performance by updating its weights and biases to minimize the error between predictions and actual targets (Rumelhart et al., 1986).

2.3.1 Key Steps in Backpropagation

The backpropagation algorithm consists of the following steps:

1. **Forward Propagation**: The input data passes through the network layer by layer, and predictions are generated based on current weights and biases.
2. **Loss Calculation**: A loss function (e.g., Mean Squared Error, Cross-Entropy Loss) quantifies the difference between the predicted output and the true target value.
3. **Backward Propagation**:
   - **Gradient Calculation**: Using the chain rule, gradients of the loss function with respect to weights and biases are computed layer by layer, starting from the output layer and moving backward. For a weight $w_{ij}^{(l)}$, the gradient is computed as:

$$\frac{\partial \mathcal{L}}{\partial w_{ij}^{(l)}} = \delta_j^{(l)} a_i^{(l-1)}$$

where $\delta_j^{(l)}$ is the error term for neuron j in layer l.

- **Weight Updates**: The gradients are used to adjust weights and biases in the opposite direction of the gradient to reduce the loss. This step is governed by an optimization algorithm, such as Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (ADAM) (Kingma and Ba, 2014).

The error term $\delta_j^{(l)}$ is calculated as:

$$\delta_j^{(l)} = \frac{\partial \mathcal{L}}{\partial z_j^{(l)}} = \sigma'\left(z_j^{(l)}\right) \sum_k \delta_k^{(l+1)} w_{jk}^{(l+1)}$$

where:

- $\mathcal{L}$: Loss function,
- $\sigma'\left(z_j^{(l)}\right)$: Derivative of the activation function,
- $\delta_k^{(l+1)}$: Error term of neuron k in the next layer $l+1$,
- $w_{jk}^{(l+1)}$: Weight connecting neuron j in layer l to neuron k in layer $l+1$.

2.4 Optimization Algorithms

Optimization algorithms play a crucial role in training machine learning ML models by minimizing the loss function. These algorithms iteratively update the model's parameters to improve performance. Two popular optimization techniques in ML are Gradient Descent and ADAM.

2.4.1 Gradient Descent

Gradient Descent is a first-order optimization method that minimizes the loss function by iteratively moving in the direction of the steepest descent.

Mathematical Formulation

The parameter update rule for Gradient Descent is expressed as:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t)$$

where:

- θ_t: Model parameter at iteration t,
- η: Learning rate, controlling the step size,

- $\nabla_\theta \mathcal{L}(\theta_t)$: Gradient of the loss function $\mathcal{L}$ with respect to θ at iteration t.

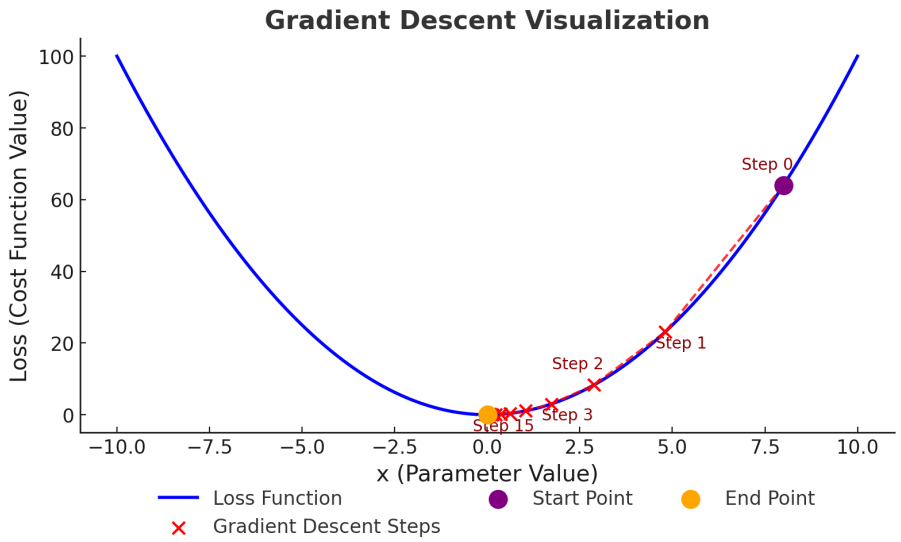


Figure 2.3: Gradient descent in operation

Variants of Gradient Descent

- **Batch Gradient Descent**: Computes the gradient using the entire dataset, leading to stable updates but higher computational cost for large datasets.
- **Stochastic Gradient Descent**: SGD updates model parameters iteratively using the gradient of a single randomly selected data point per step:

$$\theta_{t+1} = \theta_t - \eta \nabla_\theta \mathcal{L}(\theta_t).$$

  Unlike batch gradient descent, which computes the gradient over the entire dataset, SGD is computationally efficient and allows faster updates. However, it introduces noise due to high variance in gradient estimates, leading to oscillations around the optimum rather than smooth convergence. Despite its instability, the noise can help escape sharp local minima, often improving generalization.
- **Mini-Batch Gradient Descent**: A compromise between batch and stochastic methods, it computes the gradient over small data batches, balancing stability and efficiency.

2.4.2 Adam Optimizer

ADAM Kingma and Ba, 2014 extends SGD by incorporating momentum and adaptive learning rates. It maintains exponentially weighted moving averages of the

gradient (m_t) and squared gradient (v_t), with bias correction:

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t}.$$

The update rule is:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}.$$

where η is the learning rate, β_1, β_2 control momentum $(0.9, 0.999)$, and ϵ ensures numerical stability. ADAM's adaptive learning rates improve convergence, making it a widely used optimizer in deep learning.

2.4.3 Learning Rate Schedulers

A learning rate scheduler adjusts the learning rate η during training to improve performance. One example of this is Cosine Annealing (Loshchilov and Hutter, 2016), which gradually decreases η using a cosine function:

$$\eta_t = \eta_{\min} + \frac{1}{2}(\eta_0 - \eta_{\min}) \left(1 + \cos\left(\frac{t}{T}\pi\right)\right).$$

This means the learning rate starts high and smoothly decreases to $\eta_{\min}$ over time. This helps the model learn quickly at the beginning and fine-tune parameters later, improving convergence and stability.

2.4.4 Regularization Techniques in Deep Learning

Regularization techniques are essential in deep learning to prevent overfitting, improve generalization, and enhance model performance on unseen data. Overfitting occurs when a model learns noise or irrelevant patterns in the training data, leading to poor performance on validation or test sets. Regularization techniques impose constraints or add penalties to the learning process to avoid this issue.

L1 and L2 Regularization

L1 (Tibshirani, 1996) and L2 regularization add penalties to the loss function based on the magnitude of model weights:

- **L1 Regularization (Lasso)**: Adds the absolute values of weights to the loss function:

$$\mathcal{L}_{\text{L1}} = \mathcal{L}_{\text{original}} + \lambda \sum_i |w_i|$$

where:

- $\mathcal{L}_{\text{original}}$: Original loss function (e.g., mean squared error or cross-entropy loss).
- w_i: Weight of the i-th parameter in the model.
- λ: Regularization parameter, which controls the strength of the penalty. A higher λ results in greater regularization.

This encourages sparsity, often resulting in some weights becoming exactly zero.

- **L2 Regularization (Ridge)**: Adds the squared values of weights to the loss function:

$$\mathcal{L}_{\text{L2}} = \mathcal{L}_{\text{original}} + \lambda \sum_i w_i^2$$

This penalizes large weights, encouraging smoother and smaller weight values.

Batch Normalization

Batch normalization (Ioffe and Szegedy, 2015) standardizes the inputs to each layer during training, reducing internal covariate shift. It helps stabilize the learning process and can act as a form of regularization by introducing noise through the normalization process. For a mini-batch, the normalized output is:

$$\hat{x} = \frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}}, \quad y = \gamma \hat{x} + \beta$$

where μ and σ^2 are the mini-batch mean and variance, and γ, β are learnable parameters.

Early Stopping

Early stopping is a simple yet effective regularization technique that monitors model performance on a validation set during training. The training process is stopped when the validation performance stops improving, preventing overfitting to the training data. Early stopping ensures the model retains its best generalization performance without explicitly modifying the loss function or adding penalties.

Data Augmentation

Data augmentation artificially increases the size and diversity of the training dataset by applying transformations such as rotation, flipping, scaling, or cropping. This helps the model generalize better by learning invariances to these transformations.

2.5 Convolutional Neural Networks

Convolutional Neural Network (CNN) are specialized ANNs designed to process data with a grid-like structure, such as images. They are particularly effective in tasks involving spatial data because of their ability to learn local patterns, such as edges and textures, through convolutional operations (LeCun et al., 1998).

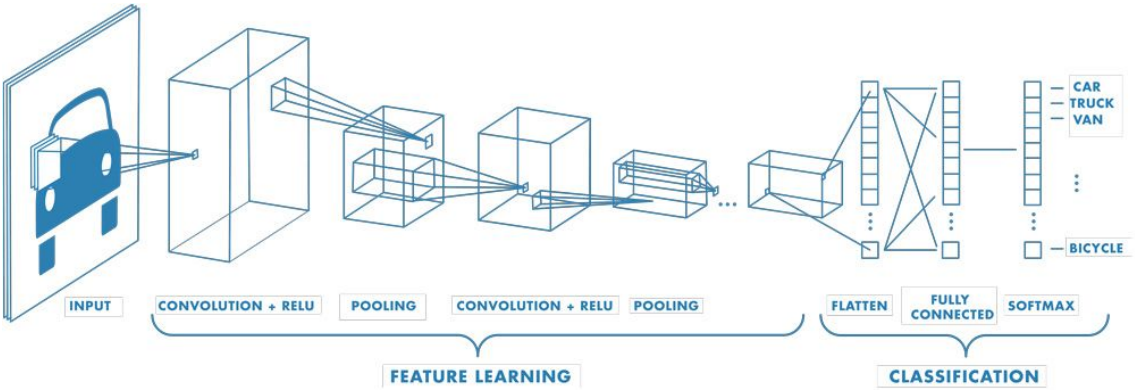


Figure 2.4: Structure of a Convolutional Neural Network (MathWorks, n.d.)

2.5.1 Structure of CNNs

CNNs consist of several key layers, each contributing to feature extraction and decision-making:

Convolutional Layers

Convolutional layers apply convolutional operations to input data using small, learnable filters (kernels). These filters slide across the input to produce feature maps, capturing local spatial patterns (Goodfellow et al., 2016). The output of a convolution operation is given by:

$$f(x, y) = (I * K)(x, y) = \sum_m \sum_n I(x + m, y + n) K(m, n)$$

where:

- $I(x, y)$: Input image,
- $K(m, n)$: Kernel/filter,
- $*$: Convolution operation.

Mathematically, the operation described here is **cross-correlation**, not convolution, as the filter K is not flipped. However, it is often referred to as *"convolution"* in

deep learning contexts for simplicity (Goodfellow et al., 2016).

Pooling Layers

Pooling layers reduce the spatial dimensions of feature maps, making the network computationally efficient and invariant to small translations in the input. This helps retain the most important features while discarding redundant information. Common pooling methods include:

- **Max Pooling**: Selects the maximum value from a region of the feature map, which highlights prominent features.
- **Average Pooling**: Computes the average of values within a region of the feature map, summarizing local information.

Mathematically, for max pooling over a region, the output can be expressed as:

$$P(x, y) = \max_{m,n} (f(x + m, y + n))$$

where $P(x, y)$ is the output of pooling, $f(x+m, y+n)$ is the feature value at position $(x + m, y + n)$ in the input.

Pooling reduces the size of the feature maps when combined with an appropriate stride, which in turn decreases the computational complexity of the subsequent layers (Zeiler and Fergus, 2014).

Fully Connected Layers

After the convolutional and pooling layers, the feature maps are flattened into a one-dimensional vector. This vector serves as input to the fully connected layers, which consist of traditional dense layers from ANNs. These layers combine the extracted features to perform tasks such as classification or regression (Goodfellow et al., 2016).

Mathematically, the output of a fully connected layer is calculated as:

$$a_j^{(l)} = \sigma \left(\sum_i w_{ij}^{(l)} a_i^{(l-1)} + b_j^{(l)} \right)$$

where:

- $w_{ij}^{(l)}$: Weight connecting neuron i in layer $l - 1$ to neuron j in layer l,
- $b_j^{(l)}$: Bias for neuron j in layer l,
- σ: Activation function (e.g., ReLU or Sigmoid).

2.5.2 Activation Functions

Activation functions introduce non-linearity into the network, enabling it to learn complex patterns. Two commonly used activation functions in CNNs are Rectified Linear Unit (ReLU) and Sigmoid.

Rectified Linear Unit:

ReLU is defined as:

$$f(x) = \max(0, x)$$

It outputs x for positive inputs and 0 otherwise. ReLU is computationally efficient, prevents the vanishing gradient problem for positive inputs, and promotes sparsity by deactivating neurons for negative values.

Sigmoid:

Sigmoid maps inputs to the range $[0, 1]$:

$$f(x) = \frac{1}{1 + e^{-x}}$$

This makes it suitable for binary classification outputs. However, Sigmoid can suffer from the vanishing gradient problem, slower convergence due to non-zero-centered outputs, and higher computational cost compared to ReLU.

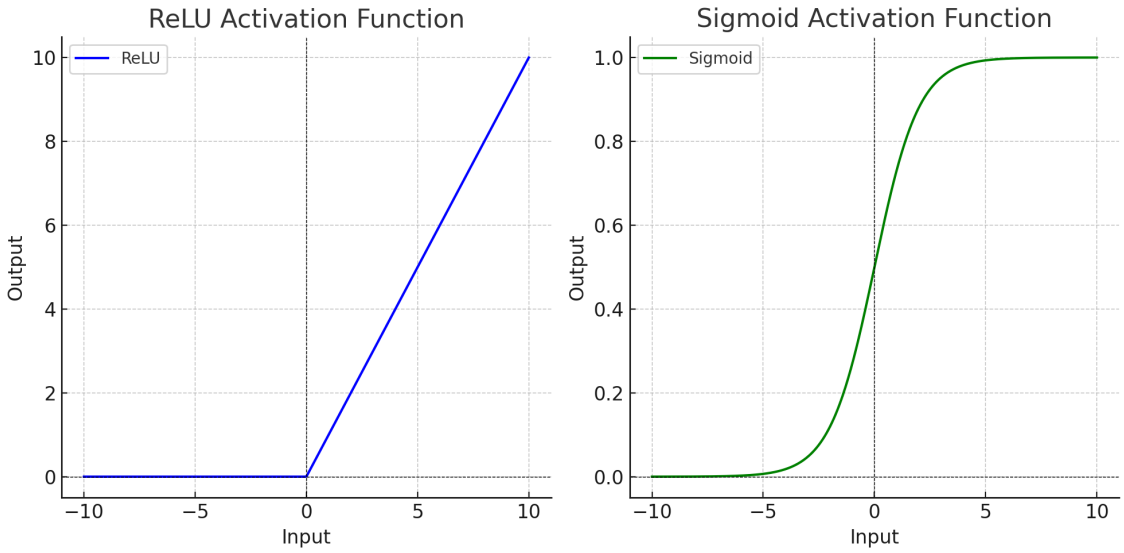


Figure 2.5: ReLU vs Sigmoid activation functions

Vanishing Gradient Problem: The *vanishing gradient problem* occurs when gradients become very small as they are passed backward through the layers of a

deep neural network during training.
This happens because some activation functions, like the Sigmoid, squash input values into a small range (e.g., [0, 1] for Sigmoid). In these ranges, their derivatives are very small. When gradients are multiplied layer by layer during backpropagation, they shrink quickly and can become almost zero for earlier layers.
As a result:

- Earlier layers learn very slowly or stop learning altogether.
- The network struggles to train effectively, especially when it is deep.

Softmax:

Softmax is commonly used in the output layer of classification networks for multi-class problems. It converts raw logits (network outputs) into a probability distribution over C classes:

$$f(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{C} e^{x_j}}, \quad i = 1, \ldots, C$$

where:

- x_i: Logit (raw output) for class i,
- C: Total number of classes.

Softmax ensures that the output probabilities sum to 1, making it suitable for multi-class classification tasks.

2.6 Cross-Entropy Loss

Cross-Entropy Loss is a widely used loss function in ML, particularly for classification tasks. It measures the dissimilarity between the predicted probability distribution and the true distribution of the target labels. Cross-Entropy Loss is derived from information theory and quantifies the uncertainty of predictions when compared to the ground truth.
The Cross-Entropy Loss for a single prediction can be expressed as:

$$\mathcal{L} = -\sum_{i=1}^{C} y_i \log(\hat{y}_i)$$

where:

- C: Number of classes,
- y_i: Ground truth label for class i (1 if i is the true class, otherwise 0),
- $\hat{y}_i$: Predicted probability for class i, obtained from a softmax activation func-

tion.

Applications

- **Multi-Class Classification**: Cross-Entropy Loss is commonly used with the softmax activation function for tasks where the output corresponds to multiple mutually exclusive classes (e.g., image classification).
- **Binary Classification**: In binary classification, Cross-Entropy Loss can be simplified as:

$$\mathcal{L}_{\text{binary}} = -\frac{1}{N}\sum_{n=1}^{N}\left[y^{(n)}\log(\hat{y}^{(n)}) + (1-y^{(n)})\log(1-\hat{y}^{(n)})\right]$$

2.7 Residual Networks

Residual Networks (ResNet) are a family of deep convolutional neural networks introduced by (He et al., 2016) to address the degradation problem in deep networks. As networks become deeper, training becomes more challenging due to vanishing gradients and the potential for accuracy to saturate or degrade. ResNet solves this by introducing **residual learning**, which enables the training of very deep networks without performance degradation.

2.7.1 Core Concept: Residual Learning

The key idea behind ResNet is the use of **residual blocks**. Instead of learning the desired underlying mapping $H(x)$, a residual block learns the residual $F(x) = H(x) - x$. This reformulation allows the network to explicitly pass the input x through an **identity shortcut connection**, effectively bypassing one or more layers.

The shortcut connection ensures that the gradient can flow directly to earlier layers, mitigating the vanishing gradient problem.

2.7.2 ResNet-18 Architecture

ResNet-18 is one of the smallest models in the ResNet family, designed for efficient training and inference. It consists of:

- 18 layers, including 16 convolutional layers and 2 fully connected layers.
- Basic residual blocks with two 3x3 convolutional layers.

- Identity and projection shortcuts for residual learning.

The architecture of ResNet-18 is as follows:

- **Input**: 224x224 RGB image.
- **First layer**: 7x7 convolution, stride 2, followed by max pooling.
- **Convolutional layers**: Four stages of residual blocks, each doubling the number of channels:
  - Stage 1: 2 residual blocks with 64 channels.
  - Stage 2: 2 residual blocks with 128 channels.
  - Stage 3: 2 residual blocks with 256 channels.
  - Stage 4: 2 residual blocks with 512 channels.
- **Output**: Global average pooling and a fully connected layer for classification.

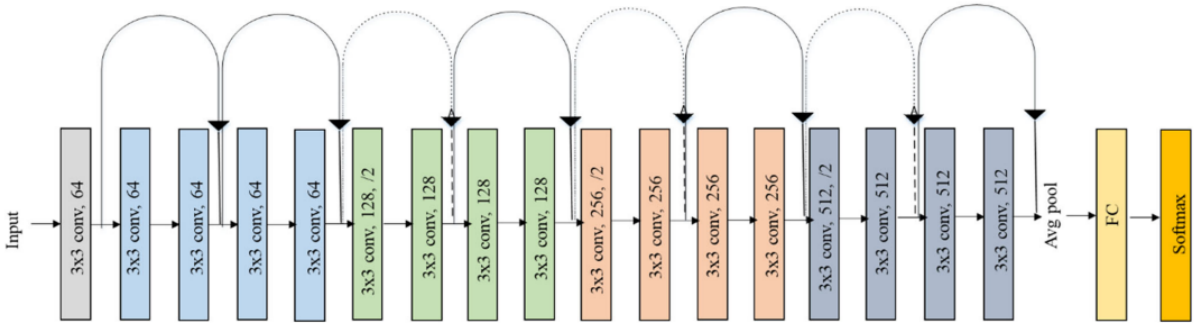


Figure 2.6: ResNet-18 Architecture (Ramzan et al., 2019)

2.7.3 ResNet-50 Architecture

ResNet-50 is a deeper variant designed for more complex tasks, with higher accuracy compared to ResNet-18. It consists of:

- 50 layers, including 48 convolutional layers and 2 fully connected layers.
- Bottleneck residual blocks, which reduce computational cost.
- Identity and projection shortcuts.

The architecture of ResNet-50 is as follows:

- **Input**: 224x224 RGB image.
- **First layer**: 7x7 convolution, stride 2, followed by max pooling.
- **Convolutional layers**: Four stages of bottleneck residual blocks:
  - Stage 1: 3 blocks with 64 channels.
  - Stage 2: 4 blocks with 128 channels.
  - Stage 3: 6 blocks with 256 channels.
  - Stage 4: 3 blocks with 512 channels.
- **Output**: Global average pooling and a fully connected layer for classification.

2.7.4 Comparison of ResNet-18 and ResNet-50

- **Depth and Complexity**: ResNet-50 is deeper and more complex, making it suitable for large-scale datasets. ResNet-18 is simpler and faster to train, making it ideal for smaller datasets or resource-constrained environments.
- **Accuracy**: ResNet-50 generally achieves higher accuracy due to its greater depth and capacity for learning intricate features.
- **Efficiency**: ResNet-18 requires fewer parameters and computational resources, resulting in faster inference.

2.8 PyTorch: A Deep Learning Framework

PyTorch is an open-source deep learning framework developed by Facebook AI Research. It combines Graphics processing Unit (GPU)-accelerated computation with a Pythonic interface, making it ideal for rapid prototyping and model experimentation. PyTorch uses dynamic computation graphs, enabling on-the-fly graph construction, which is particularly useful for research-focused tasks like anomaly detection and deep learning model customization (Paszke et al., 2019).

2.8.1 Relevance to Anomaly Detection

PyTorch's flexibility makes it well-suited for anomaly detection tasks, where custom architectures often need to adapt to diverse data distributions. Key features that align with anomaly detection workflows include:

- **Dynamic Graphs**: Allow experimentation with non-standard models like PatchCore and CutPaste.
- **Extensive Libraries**: Prebuilt modules (`torch.nn`, `torch.optim`) accelerate the development of specialized neural networks.
- **GPU Acceleration**: Handles large datasets and computationally intensive tasks efficiently.
- **Community Support**: A rich ecosystem of tools, tutorials, and active forums facilitates problem-solving and implementation.

PyTorch's focus on usability and flexibility bridges the gap between research and practical applications, ensuring it can handle the iterative and exploratory nature of anomaly detection models.

2.9 Related Work

In this section, we discuss several key papers that either precede PatchCore and CutPaste or build upon their ideas, offering a deeper understanding of anomaly detection methods and their evolution.
According to (Cui et al., 2023), unsupervised deep learning approaches for anomaly detection can be divided into four categories:

1. **Reconstruction-Based Methods:** These methods train models to reproduce normal data. During testing, if the model cannot accurately reconstruct an input, it is considered anomalous. Reconstruction errors highlight patterns the model did not encounter during training. Techniques like autoencoders and Generative Adversarial Network are commonly used in this category.
2. **Normalizing Flow-Based Methods:** These methods use a series of invertible transformations to map complex data distributions into simpler ones. They learn the likelihood of normal data, and anomalies are identified as points with low probability under this learned distribution.
3. **Representation-Based Methods:** This includes PatchCore.
4. **Data Augmentation-Based Methods:** This includes CutPaste.

Representation-Based Methods

The key idea of representation-based methods is to compare embedding features from target and normal images. Where an embedding is a compact and meaningful numerical representation of an image that captures important characteristics and patterns, such as textures, shapes, and semantic content. These embeddings are typically generated using pre-trained deep learning models, such as Convolutional Neural Networks (CNNs), which are trained on large datasets like ImageNet. The embeddings effectively transform high-dimensional image data into a lower-dimensional feature space, where the essential information about the image is preserved.
In anomaly detection, these embeddings allow for comparisons between normal and target (potentially anomalous) images These embeddings are typically derived using pre-trained models trained on large-scale databases such as ImageNet.
An example of this is **Semi-supervised Pseudolabeler Anomaly Detection with Ensembling (SPADE)** (Yoon et al., 2022), an anomaly detection framework that combines feature extraction with unsupervised and supervised learning. SPADE trains an ensemble of One-Class Classifiers (OCCs) on data subsets to assign

pseudo-labels to unlabeled samples when the ensemble agrees. These pseudo-labels, along with any existing labels, are used to train a supervised model to identify anomalies.

Another example is the **Patch Distribution Modeling (PaDiM)** (Defard et al., 2020), an anomaly detection method that uses a pre-trained CNN to extract patch-level features and models normal data with multivariate Gaussian distributions. PaDiM captures detailed relationships across semantic levels of the CNN by leveraging these distributions. Anomalies are detected by computing the Mahalanobis distance between test image features and the Gaussian model, where high scores indicate anomalies.

Among these, **PatchCore** demonstrated superior performance on the MVTec dataset, outperforming both SPADE and PaDiM by efficiently leveraging embeddings and patch-level representations for anomaly detection.

Data Augmentation-Based Methods

Data augmentation-based methods play a crucial role in enhancing the performance of SSL by introducing creative and diverse variations in data, allowing models to learn robust patterns and features without requiring labeled data.

SSL has become a powerful approach for training models without labeled data, relying on diverse variations in data to help the model understand patterns and features. A key part of this process is data augmentation, which introduces variations to make the model more robust. Techniques such as **CutMix**, **Cutout**, and **PatchMix** stand out in this area. CutMix creates new training examples by cutting and pasting parts of one image onto another, blending their labels to encourage the model to learn from mixed patterns. Cutout works by randomly blocking out sections of an image, pushing the model to rely on the remaining parts to make sense of the data. PatchMix extends this idea by creating grids of mixed image patches to add more variety and detail to training.

CutPaste applies targeted transformations, such as cutting and repositioning parts of an image, to create subtle anomalies for training. This approach enhances models' ability to detect irregularities, making it highly effective for anomaly detection.

CHAPTER 3

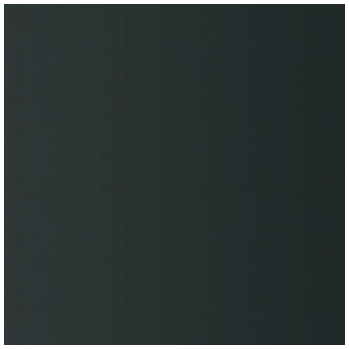
Data and Preprocessing

In this chapter, we describe the dataset used in this thesis, detailing its structure, categorization, and the preprocessing steps applied to prepare the data for analysis. The preprocessing pipeline, tailored to the requirements of PatchCore and CutPaste, includes patch-based analysis, normalization, and the creation of ground truth for testing. These steps ensure the dataset is optimized for robust and efficient anomaly detection.

3.1 Dataset Description and Categorization

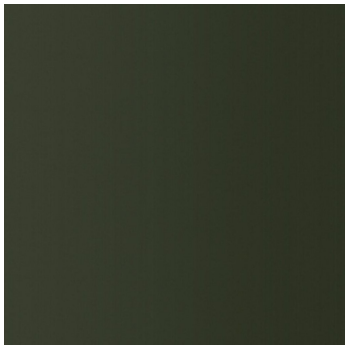
The data in this thesis was provided by "Burger Küchenmöbel GmbH", a company that specialize in kitchen-related carpentry like worktops, kitchen storage spaces, cabinets and drawers.
The dataset comprises 621 high-resolution images of kitchen boards captured using an industrial line scan camera in a factory setting. As a first step, the images were categorized based on properties such as brightness, color, material type, and the presence or absence of edges. The results as can be seen in figure 3.1, are as follows:

- **White**
- **Green**
- **Black**
- **White with edges**
- **Concrete**
- **Dark with edges**
- **Wood**

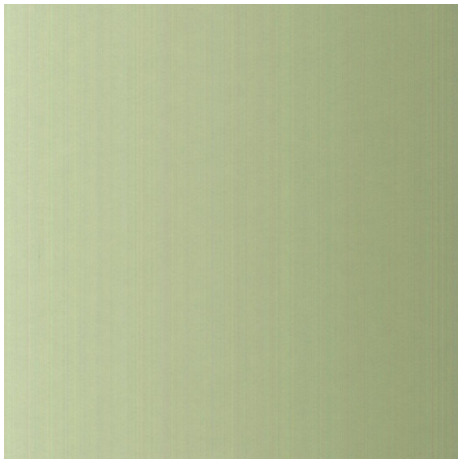


(a) Black

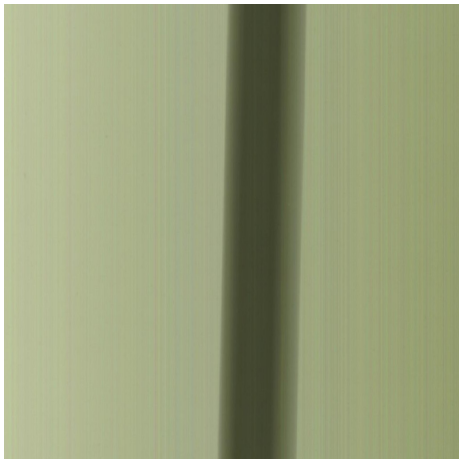
(b) Dark with edges



(c) Green

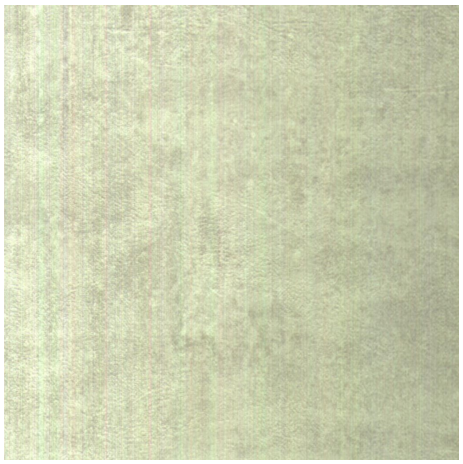


(d) White



(e) White with edges

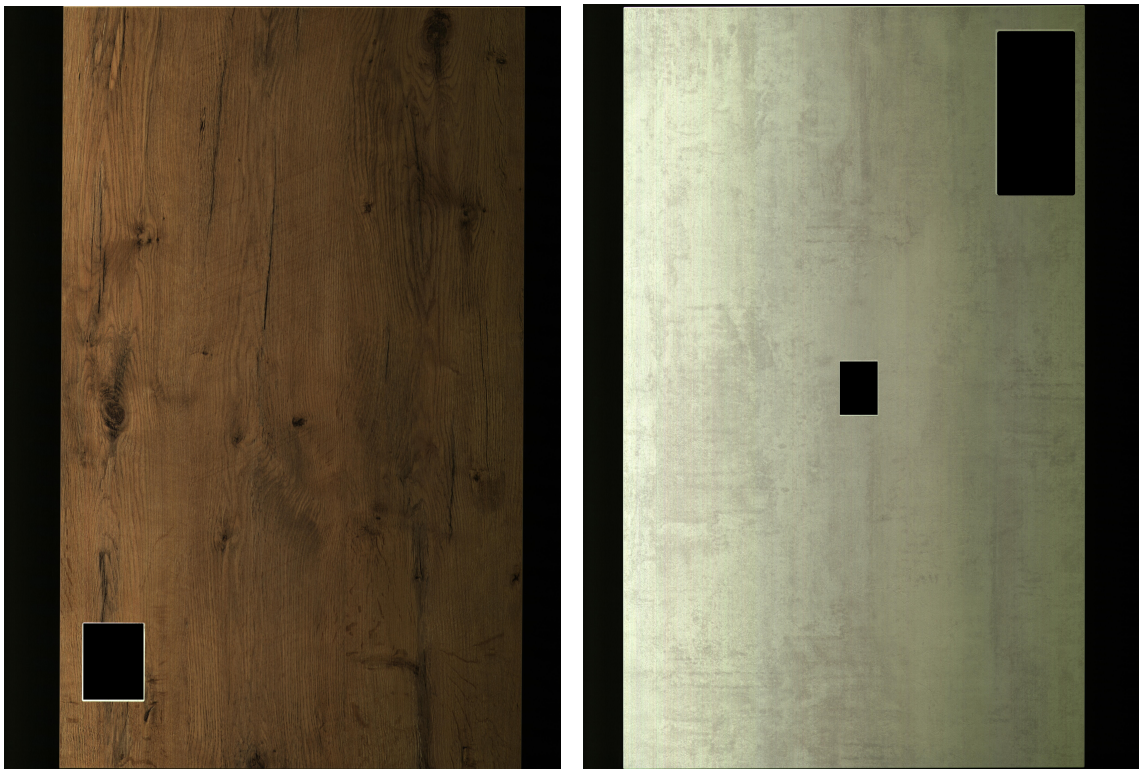
(f) Wood



(g) Concrete

Figure 3.1: Overview of the Categories

The "Wood" category was excluded from analysis because it lacks a clear structure, making it difficult to differentiate anomalies from normal samples. Similarly, "Concrete" was challenging for human inspectors to identify anomalies accurately (figure 3.2). The "Black" and "Green" categories were found to have similar characteristics to "White" boards but appeared darker, making them less distinct. The same observation applies to "Dark with edges", which has similar characteristics to "White with edges".
Given these considerations, the focus of this thesis was narrowed to the "White" and "White with edges" categories. examples of these can be seen in figure 3.3.



(a) an example of wood (b) an example of concrete

Figure 3.2: showcasing the difficultly of distinguishing anomalies in Wood and concrete

The **White** folder contains 254 images, while the **White with edges** folder contains 57 images. Each folder is further divided manually into two subfolders: **anomaly-free** and **with anomaly**. The distribution is as follows:

- **White**:
  - 86 images with anomalies, 11 of these will used for testing.
  - 168 images without anomalies, which will be used for training.
- **White with edges**:

- 20 images with anomalies, 9 of these will used for testing.
- 37 images without anomalies, which will be used for training.

The average dimensions of the images across both categories were analyzed, revealing an average width of 4074 pixels and an average height of 4730 pixels. To better understand the nature of the anomalies present in the dataset, further inspection was conducted. This examination identified several main types of anomalies, as illustrated in figure 6.5. Additionally, it was observed that most images exhibit more than one type of anomaly.

3.2 Preprocessing the Dataset

3.2.1 Image Patch Preparation for Training data

To facilitate localized analysis, the dataset's images are divided into smaller patches. This preprocessing step is essential for models like PatchCore and CutPaste, which rely on detailed, localized features to identify anomalies. The preparation of patches involves several stages, including calculating the number of patches, determining their coordinates, extracting the patches, filtering out irrelevant ones, and finally determining the number of retained patches.

The first step is to calculate the number of patches that can be extracted from an image. This depends on the image dimensions, the patch size, and the stride used for sliding the patch window. The number of patches along each row is determined by dividing the width of the image, adjusted for the patch size and stride. Similarly, the number of patches along the columns depends on the height. The following formulas define these values:

$$\text{patches\_per\_row} = \left\lfloor \frac{\text{width} - \text{Patch Size}}{\text{stride}} \right\rfloor + 1$$

$$\text{patches\_per\_column} = \left\lfloor \frac{\text{height} - \text{Patch Size}}{\text{stride}} \right\rfloor + 1$$

With these calculations, the dimensions of the patch grid are established, ensuring an even distribution of patches across the image.

Next, the coordinates of the top-left corner of each patch are determined. Using the stride, the x- and y-coordinates for each patch can be calculated iteratively. The coordinates along the rows depend on the stride and the index of the patch in the row. The same principle applies to the columns. These coordinates are given by:

$$x = \text{stride} \cdot i \quad \text{where } i = 0, 1, \ldots, (\text{patches\_per\_row} - 1)$$

$$y = \text{stride} \cdot j \quad \text{where } j = 0, 1, \ldots, (\text{patches\_per\_column} - 1)$$

This systematic calculation ensures that patches are extracted consistently and without overlap beyond the intended stride.
Once the coordinates are defined, patches are extracted as sub-images from the original dataset. Each patch is cropped based on its top-left and bottom-right coordinates. The cropping operation is defined as:

$$\text{patch} = \text{crop}(x, y, x + \text{Patch Size}, y + \text{Patch Size})$$

where $\text{crop}(x_1, y_1, x_2, y_2)$ extracts the bounding box of the image enclosed by the specified coordinates.
After extraction, the patches are filtered to remove those that are irrelevant for the analysis such as completely black regions. To determine which patches to keep, the mean intensity of each patch is calculated. Only those patches with a mean intensity above a threshold value of 31 are retained. This threshold was selected empirically to ensure that meaningful content is included while reducing noise in the dataset.
Finally, the total number of retained patches for each image depends on its dimensions, the chosen patch size and stride, and the filtering process. These parameters are chosen carefully during the training and testing phases to align with the requirements of the anomaly detection model.

3.2.2 Normalization

Normalization is applied to ensure consistent scaling of pixel values. The normalization is using the mean and standard deviation. They are calculated based on the RGB channels of all the patches in one category.

3.2.3 Additional Steps for the Testing data

Ground Truth Creation

For testing, a ground truth image is manually created for each test image to evaluate the model's performance. The ground truth consists of black and white pixels: white pixels indicate the location of anomalies in the original image, while black pixels

represent normal regions. The anomalies in the ground truth were marked to the best of my knowledge and two people agreed on it.

Preprocessing with `.json` File

The preprocessing of training data is performed using a `.json` file. This file specifies the coordinates of the active area of the image to exclude the black background. Additionally, it defines the coordinates of labeled regions on the image surface, which are excluded from further processing.
The `.json` file contains a list of test image names. Each image entry includes two keys: `labels`, which defines regions to exclude, and `borders`, which specifies the active area. An example structure is shown below:

```
{
    "231117_0302_1.jpg":
    {
        "labels": [[659, 2231, 600, 1239], [2064, 1792, 404, 294]],
        "borders": [[617, 161, 3279, 3424]]
    }
}
```

Patch Extraction with Defined Borders and Exclusion Areas

Using the information from the `.json` file, patches are extracted only from the active region of the image. Unlike the training set, which considers the entire image for patch extraction, the testing set restricts patch locations to the defined `borders`. To ensure consistency, the patch extraction process in each row and column follows the same approach as in the training set.
The patch extraction process begins by determining the active region from the `borders` field, which provides a clear boundary for valid patch locations. Once the active area is identified, patches are systematically extracted. If necessary, additional patches are added along the right and bottom edges to ensure full coverage of the active area. After extraction, patches are filtered to exclude those that overlap with the `labels` regions. This is achieved by checking if a patch's coordinates intersect with any labeled areas defined in the `labels` field, preventing any interference from predefined excluded regions.

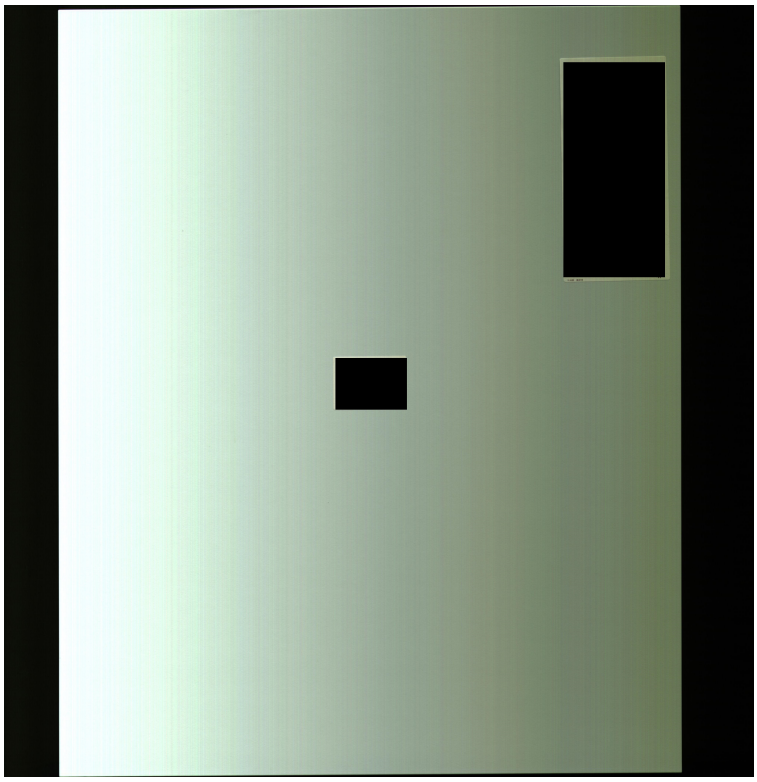
Ground Truth Annotation Integration

To evaluate anomaly detection, the corresponding ground truth mask, stored as a separate image with a `groundtruth.png` suffix, is utilized. For each extracted patch, the corresponding ground truth mask is first loaded to ensure alignment with the test image. Using the same coordinates from the test image, a matching patch is then extracted from the ground truth mask. The extracted ground truth patch is subsequently converted to grayscale to simplify the intensity analysis. If the mean pixel intensity of the ground truth patch is greater than zero, the patch is labeled as containing an anomaly, denoted by label 1. Otherwise, it is classified as normal, with label 0.
This systematic approach ensures that the testing dataset is well-structured for evaluating the anomaly detection model while maintaining consistency with the training process. By applying this method, only relevant patches are included, reducing noise and improving the reliability of the model's evaluation.

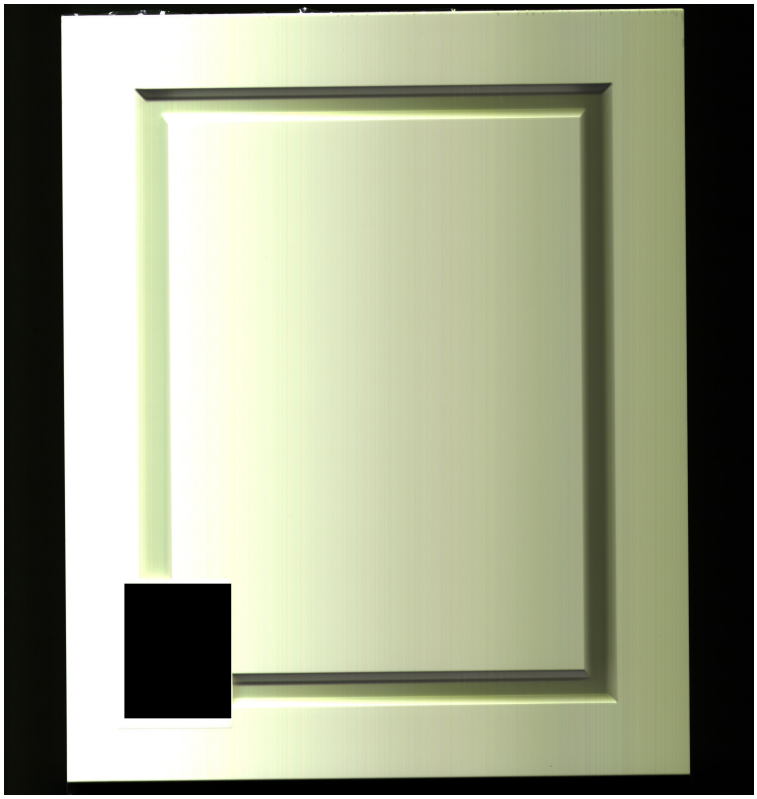
3.2.4 Benefits of the Patch-Based Analysis

This patch-based approach offers several advantages, addressing both computational and practical challenges in image analysis:

- **Enhanced Defect Detection:** This patch-level analysis is particularly useful for detecting small, localized defects that might be missed if the whole image were processed as a single unit.
- **Increased Data Availability:** By cutting large-scale images into smaller patches, the number of training samples increases significantly. For instance, processing only large-scale images might provide fewer than a hundred training examples, whereas dividing them into smaller patches yields thousands of training samples, significantly enriching the dataset.
- **Efficient Training:** Filtering out irrelevant patches (e.g., black regions) optimizes the data pipeline and improves model performance by focusing on meaningful areas of the image.
- **Memory Efficiency:** Large images (e.g., 2000×2000 pixels) exceed GPU memory limits, which typically support 128×128 to 512×512 pixels. Splitting images into patches ensures feasibility without major information loss.
- **Flexible Analysis:** Patch size and stride can be tuned to focus on specific details or broader patterns, allowing the approach to adapt to various image analysis requirements.



(a) Example from the White category.



(b) Example from the White with edges category.

Figure 3.3: Examples of the anomaly-free data

CHAPTER 4

Methodology

In this chapter, we present the theoretical and mathematical foundations of the two primary methods in this thesis: PatchCore and CutPaste. PatchCore combines memory-based anomaly detection with efficient coreset reduction, while CutPaste leverages synthetic anomaly generation for unsupervised learning. Each method is explored in detail, highlighting its principles, methodology, and mathematical framework.

4.1 PatchCore

PatchCore (Roth et al., 2022) leverages the representational power of deep neural networks and the efficiency of memory-based anomaly detection to achieve state-of-the-art performance in detecting anomalies in visual data. The methodology consists of three main stages: local patch feature aggregation into a memory bank, coreset reduction to increase efficiency, and the full algorithm for detection and localization.

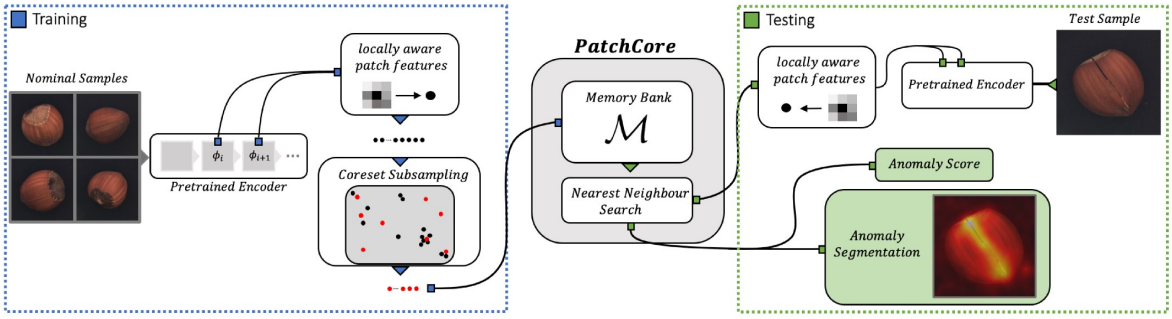


Figure 4.1: General view of PatchCore operation (Roth et al., 2022)

4.1.1 Memory Bank Construction Through Local Patch Features

The first step in PatchCore involves the extraction and aggregation of local patch features into a memory bank. The training process is performed using only non-anomalous images. A pre-trained CNN, trained on the ImageNet dataset (Deng et al., 2009), serves as the backbone for feature extraction. Specifically, intermediate feature maps are extracted from hierarchical levels $j \in [2, 3]$, balancing generality and specificity in the learned representations.

For an input image $x \in \mathbb{R}^{H \times W \times C}$, the CNN extracts feature maps $\phi_{i,j} \in \mathbb{R}^{c \times h \times w}$, where c, h, and w represent the depth, height, and width of the feature maps, respectively. Each spatial position (h, w) in these feature maps corresponds to a patch-level feature $\phi_{i,j}(h, w) \in \mathbb{R}^c$.

To enhance local contextual awareness, features are aggregated over a local neighborhood $N_p(h, w)$ with patch size p, ensuring robustness and capturing meaningful context. This is achieved using adaptive average pooling, refining each patch-level feature:

$$\phi_{i,j}(N_p(h, w)) = \text{Pool}(\phi_{i,j}(h, w)), \tag{4.1}$$

where Pool denotes the pooling operation applied to the neighborhood.

Patch-level features are collected over the spatial grid with a stride s, typically set to 1 unless specified otherwise. The set of locally refined patch features is defined as:

$$P_{s,p}(\phi_{i,j}) = \{\phi_{i,j}(N_p(h, w)) \mid h, w \bmod s = 0, h < h', w < w'\}, \tag{4.2}$$

where h' and w' are the spatial dimensions of the feature map.

To ensure a balance of detail and generality, PatchCore combines features from two intermediate levels, j and $j + 1$, aligning their resolutions using bilinear rescaling. The resulting feature vectors are aggregated into a memory bank M, which serves as a reference for subsequent anomaly detection stages. The memory bank is constructed as:

$$M = \bigcup_{x_i \in X_N} P_{s,p}(\phi_j(x_i)). \tag{4.3}$$

This memory bank captures the localized details and broader context of nominal training samples, providing a robust foundation for anomaly detection while avoiding overfitting to pretraining biases.

4.1.2 Efficient Memory Bank Reduction via Coreset Subsampling

One of the key challenges in PatchCore is managing the size of the memory bank M, which can grow significantly with larger image datasets and higher resolutions. A large memory bank not only increases computational complexity but also poses substantial memory requirements, making it infeasible on most hardware setups. To address this, PatchCore employs a coreset subsampling mechanism to reduce the size of M while maintaining its representational coverage of nominal features.
Coreset subsampling selects a subset $M_C \subseteq M$ that retains the essential characteristics of the original memory bank, making it searchable and efficient for patch-based comparisons. Formally, coreset selection minimizes the maximum distance between any feature $m \in M$ and its nearest neighbor in M_C:

$$M_C = \arg \min_{M_C \subseteq M} \max_{m \in M} \min_{n \in M_C} \|m - n\|_2. \tag{4.4}$$

PatchCore adopts an iterative greedy approximation algorithm (Sener and Savarese, 2018) to construct the coreset. To further reduce the computational cost, dimensionality reduction (Sinha et al., 2019) is applied to $m \in M$ using random linear projections based on the Johnson-Lindenstrauss theorem (Dasgupta and Gupta, 2003). This projection reduces the dimensionality of the feature space while preserving pairwise distances, enabling faster coreset construction. The reduced coreset M_C retains the spatial coverage of nominal features, ensuring effective anomaly detection and segmentation.
By reducing the memory bank to a manageable size, PatchCore achieves significant improvements in inference time while preserving detection and localization accuracy. This enables efficient processing of large datasets without compromising performance.

4.1.3 Anomaly Detection and Segmentation

Given a test image x_{test}, PatchCore computes an anomaly score by comparing its patch-level features to those in the nominal memory bank M. This involves generating feature maps, calculating patch-level anomaly scores, and deriving an image-level anomaly score.

Patch-Level Anomaly Scores

Feature maps are extracted from the test image using a pre-trained network:

$$F_{\text{test}} = \phi(x_{\text{test}}) \in \mathbb{R}^{h \times w \times d}, \tag{4.5}$$

where h, w, and d represent the height, width, and depth of the feature map. These maps are reshaped into $h \times w$ feature vectors:

$$m_{\text{test},i,j} = F_{\text{test}}(i,j), \quad i \in \{1,\ldots,h\},\ j \in \{1,\ldots,w\}. \tag{4.6}$$

For each feature vector $m_{\text{test},i,j}$, the anomaly score is computed as the minimum Euclidean distance to any feature vector $m \in M$ in the memory bank:

$$s_{i,j} = \min_{m \in M} \|m_{\text{test},i,j} - m\|_2. \tag{4.7}$$

The collection of patch-level scores $s_{i,j}$ forms an anomaly map:

$$S_{\text{map}}(x_{\text{test}}) \in \mathbb{R}^{h \times w}, \tag{4.8}$$

where higher values indicate more anomalous patches.

Image-Level Anomaly Scores

To compute an image-level anomaly score, the maximum patch-level score is taken:

$$s = \max_{1 \le i \le h,\ 1 \le j \le w} s_{i,j}. \tag{4.9}$$

This score is refined to account for the rarity of the closest memory bank feature $m \in M$ relative to its neighbors (*omitted for simplicity*). If the feature m is far from other features in the memory bank, it suggests that m represents a rare nominal occurrence. The refinement ensures that the anomaly score reflects both the distance to the closest nominal feature and the relative rarity of that feature within the memory bank.

Classification and Segmentation

If the image-level score s exceeds a predefined threshold τ, the image is classified as anomalous; otherwise, it is classified as normal:

$$\hat{y}_{\text{test}} = \begin{cases} \text{anomalous}, & \text{if } s \geq \tau, \\ \text{normal}, & \text{if } s < \tau. \end{cases} \tag{4.10}$$

For segmentation, the anomaly map $S_{\text{map}}(x_{\text{test}})$ is upscaled to match the original image resolution using bilinear interpolation.

4.2 CutPaste

CutPaste (Li et al., 2021) is an anomaly detection method designed to address challenges in unsupervised settings, particularly when anomaly samples are unavailable for training. It comprises two stages. In the first stage, the model learns features by distinguishing synthetic anomalies, created by cutting and pasting patches within images, from normal ones. In the second stage, a generative one-class classifier models the distribution of normal data to identify deviations as anomalies. This method enables effective anomaly detection and localization without the need for anomalous samples during training, making it ideal for challenging data scenarios.

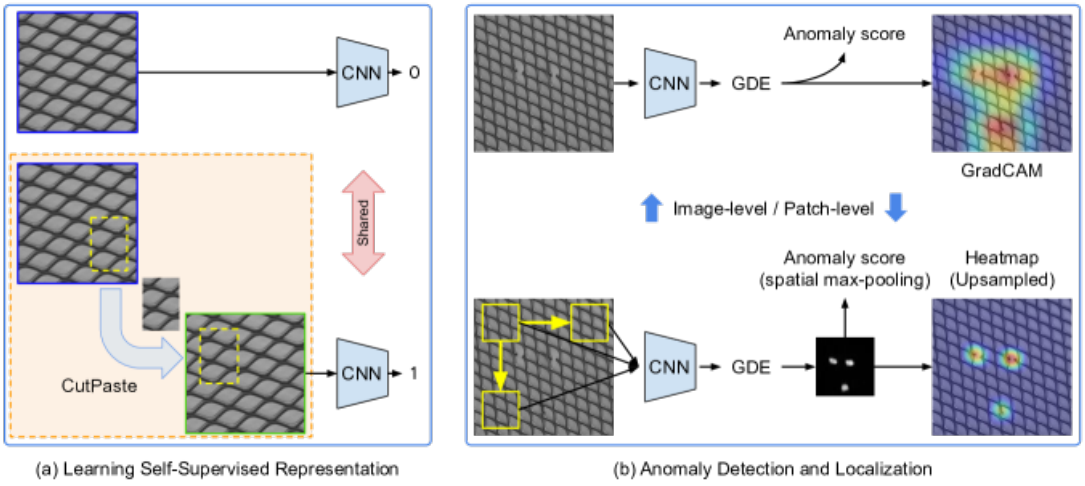


Figure 4.2: General view of CutPaste operation (Li et al., 2021)

4.2.1 Data Augmentation with CutPaste

The first step in the CutPaste methodology involves generating synthetic anomalies through a specialized data augmentation technique. Given a dataset of normal images, the process begins by randomly selecting a small rectangular region within an image. This region is then cut out and pasted onto a different location within the same image, creating a localized distortion that simulates an anomaly.

The key advantage of this approach lies in its simplicity and effectiveness. By altering only a small portion of the image, the augmented data retains the overall structure of the original while introducing realistic anomalies. This process is repeated across the dataset to create a diverse set of training samples.

To train the model, a binary classification objective is used, where the original images are labeled as normal (class 0) and the augmented images are labeled as

anomalies (class 1). The loss function L_{CP} for CutPaste is defined as:

$$L_{CP} = \mathbb{E}_{x \in X} \left[\mathrm{CE}(g(x); 0) + \mathrm{CE}(g(\mathrm{CP}(x)); 1) \right], \tag{4.11}$$

where CE denotes the cross-entropy loss (section 2.6), $g(\cdot)$ represents the model's prediction, and $\mathrm{CP}(x)$ is the CutPaste-augmented version of the image x. This loss function encourages the model to correctly classify both normal and augmented images, thereby improving its ability to detect anomalies.

4.2.2 CutPaste Variants

The authors introduce two variants of CutPaste to enhance anomaly diversity. **CutPaste-Scar** simulates thin, scar-like anomalies, ideal for detecting scratches or cracks. **CutPaste-3Way** adds a third class of unmodified images, refining the model's ability to distinguish between normal and augmented data. These variants improve robustness and adaptability for diverse anomaly detection tasks.

4.2.3 Computing Anomaly Score and Localization

After training the feature extraction model, anomaly scores are computed to detect deviations from normal patterns. One-class classifiers, such as the GDE (Rippel et al., 2020), are applied to the learned feature representations $f(x)$. GDE uses the log-density formula:

$$\log p_{\mathrm{gde}}(x) \propto -\frac{1}{2}(f(x) - \mu)^{\top} \Sigma^{-1} (f(x) - \mu), \tag{4.12}$$

where μ and Σ are the mean and covariance matrix of normal training data. This method provides a computationally efficient way to estimate anomaly scores.
For defect localization, a patch-based approach is employed. During training, patches are randomly cropped and augmented with CutPaste. The training objective is:

$$\mathbb{E}_{x \in X} \left[\mathrm{CE}(g(c(x)); 0) + \mathrm{CE}(g(\mathrm{CP}(c(x))); 1) \right], \tag{4.13}$$

where $c(x)$ denotes the cropping operation. At test time, patch embeddings are extracted, and anomaly scores are computed for each patch. Gaussian smoothing is applied to propagate scores, generating pixel-level heatmaps for precise defect localization.
Additionally, Gradient-weighted Class Activation Mapping (Grad-CAM) (Selvaraju

et al., 2017) can be utilized to visualize regions of high model activation, further enhancing interpretability for detecting and localizing anomalies.

CHAPTER 5

Implementation

In this chapter, we will delve into the practical implementation of the methodologies outlined in the previous chapter. We will discuss how the proposed approaches were applied, highlighting any deviations or adaptations made to suit specific requirements. Furthermore, we will detail the hyperparameters, workflows, and procedures employed for data preparation, training, and evaluation across the two used algorithms.

5.1 Hardware Specifications

The training and experiments for this implementation were conducted on a hardware setup equipped with the following specifications:

Property	**GPU 0**	**GPU 1**
Model	NVIDIA TITAN Xp	NVIDIA TITAN Xp
Total Memory	11.90 GB	11.90 GB
Multiprocessors	30	30
Compute Capability	6.1	6.1

Table 5.1: Specifications of GPUs in the system

5.2 Normalization

This section applies for both algorithms. As discussed in Section 3.2.2, normalization ensures consistent scaling of pixel values by utilizing the mean and standard deviation of the RGB channels. This preprocessing step is crucial for ensuring that the model operates effectively across diverse image inputs.
For this implementation, the mean and standard deviation are computed using an image size of 224 pixels and a stride of 112 pixels. These parameters were selected

for the following reasons:

- An image size of 224 pixels aligns with the input size requirements of ResNet-18 and ResNet-50 (introduced in section 2.7) , which are employed as the backbone for feature extraction (refer to Section 4.1.1).
- A stride of 112 pixels allows comprehensive coverage of the image patches without losing critical features along the edges.

The computed normalization statistics for the datasets are as follows:

- **White category:**
  - Mean: $[0.5815, 0.5940, 0.5015]$
  - Standard deviation: $[0.2716, 0.2812, 0.2710]$
- **White with Edges category:**
  - Mean: $[0.6384, 0.6557, 0.5500]$
  - Standard deviation: $[0.2846, 0.2897, 0.2772]$

These normalization values ensure robust and consistent feature extraction. Since the pre-trained ResNet models were trained on standardized data, normalizing the input ensures compatibility with their training conditions, leading to improved feature quality and model performance.

5.3 PatchCore

The implementation of PatchCore is based on the code provided in the GitHub repository by Raphael Vorias (Vorias, 2021). The original code was modified to fit our approach and a custom data set builder and loader was used to handle our data and to tailor the solution to our patch-based approach. Also, localization, visualization and metrics calculations components were added for the interpretation of results.

The following diagram illustrates the workflow of the PatchCore process:

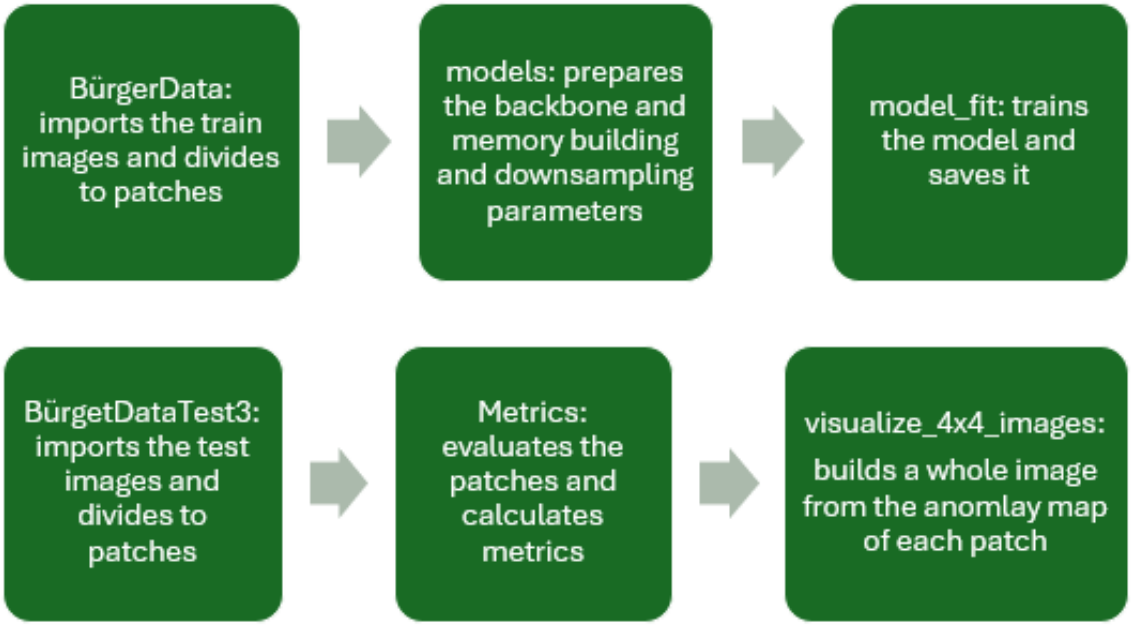


Figure 5.1: PatchCore workflow

5.3.1 Feature Extraction and Pooling

After loading and normalizing the data, PatchCore relies on intermediate representations from a pre-trained ResNet backbone (either ResNet-18 or ResNet-50, both initialized with ImageNet weights). In this implementation, feature maps are extracted from the network at indices $(2, 3)$ as explained in Section 4.1.1, using a pooling mechanism described in Equation 4.1. Each 3×3 patch, aggregated with a stride of 1, captured local context without losing spatial resolution. Attempting earlier indices $(1, 2)$ led to Out-Of-Memory (OOM) issues for all strides noted in Table 5.2. All backbone layers were kept frozen during the implementation.

5.3.2 Memory building

We start building the memory banks using exclusively clean images, we use an image size of 224 and a stride of 512 for white category and a stride of 448 for white with edges category as the computational needs for a lower stride (optimal is 112) are too high for the available hardware as per the table 5.2.

5.3.3 Coreset Subsampling

As described in Section 4.1.2, the memory bank undergoes coreset subsampling to maintain representativeness while alleviating memory usage. Only a fraction of

Image Size	Train Stride	Data Category	Data Length	Training Result (ResNet 18 and ResNet50)
224	112	White	185,875	Failed - OOM
224	224	White	46,821	Failed - OOM
224	448	White	11,899	Failed - OOM
224	512	White	9,254	Worked
244	112	With Edges	42,746	Failed - OOM
224	224	With Edges	10,946	Failed - OOM
224	448	With Edges	2,826	Worked

Table 5.2: Training Results for ResNet18 and ResNet50 Based on Image Size, Train Stride, and Data category

patches, specifically $f_{\text{coreset}} = 0.01$ (1% of patches), is retained. This process is guided by a sparse projection parameter, $\epsilon_{\text{coreset}} = 0.90$, ensuring sufficient representational coverage while maintaining efficiency, and preserving pairwise distances while enabling efficient greedy coreset selection (Equation 4.4).

5.3.4 Anomaly Detection and Scoring

For a given test image x_{test}, patch-level features are extracted and their Euclidean distances to features in the memory bank M_C are computed (Equation 4.7). These distances yield patch-level anomaly scores, combined into an anomaly map $S_{\text{map}}(x_{\text{test}})$ (Equation 4.8). Image-level anomaly scores s are determined as the maximum patch-level score, optionally reweighted using the 3 nearest neighbors. Classification as anomalous or normal is performed by thresholding at τ, and segmentation maps for localization are generated by bilinearly upscaling $S_{\text{map}}(x_{\text{test}})$ to the input resolution. A Gaussian blur with a kernel size of 4 is applied to smooth the anomaly map.

5.4 CutPaste

The implementation of CutPaste is based on the code provided in the Github repository by Lilit Yolyan (Yolyan, 2021). The original code was modified to fit our approach and a custom data set builder and loader was used to handle our data and to tailor the solution to our patch-based approach. In addition, new augmentations

were added which mimic the anomalies in our data as we will see in the next section. The following diagram illustrates the workflow of the CutPaste process:

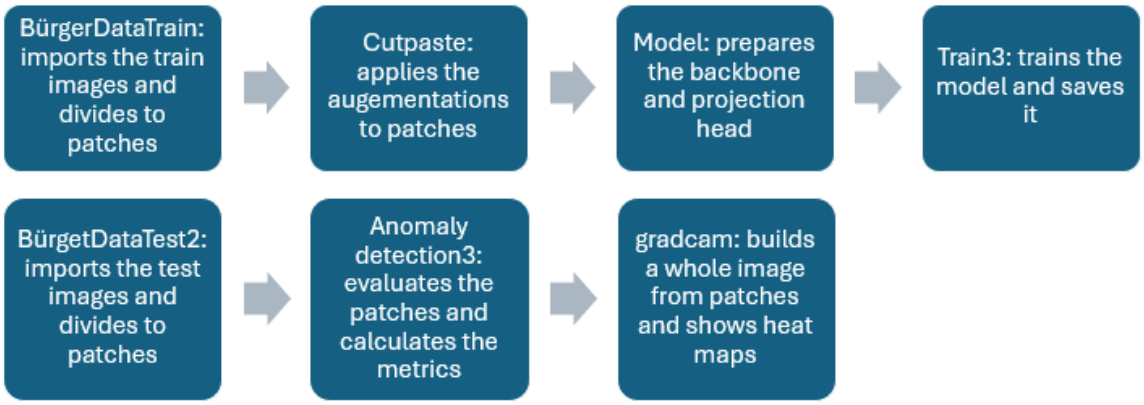


Figure 5.2: CutPaste workflow

5.4.1 Augmentation

We add an additional step to the data preparation section 3.2 which is the augmenation of data as per the methodology in 4.2.1. Some of the augmentations are according the original cutpaste paper (Li et al., 2021) and the others are to mimic the the anomalies in our data as shown in figure 6.5. The augmentations are as follows:

CutPaste Basic: it crops a patch from the original image and pastes it at a random location. We do not apply this in our model, as it does not mimic any anomaly in our data.

CutPaste-Scar: This variant introduces elongated scars:

- A patch of narrow width $(2 - 12$ pixels) and extended length $(20 - 50$ pixels) is cropped and pasted at a random location.
- Rotations $(-45°$ to $45°)$ are applied to simulate irregular tearing or scraping patterns.

Black Dot: A small circular anomaly is added to simulate specks or impurities:

- Dots with diameters of $6 - 10$ pixels are placed randomly.
- Transparency gradients mimic subtle blending into the background.

White Dot with Black Border: This anomaly mimics more pronounced, layered distortions:

- An inner white dot is surrounded by a black border, with transparency gradients ensuring seamless blending.
- Dot sizes range from $6 - 12$ pixels, and border thickness scales proportionally.

Arc Scar: This method introduces scars shaped as circular arcs:

- The radius of the arc ($35 - 90$ pixels) and thickness (5 pixels) are randomized.
- Transparency gradients simulate natural blending, with arc segments spanning $60°$ for realistic defect shapes.

To ensure diversity in anomalies, a random selection of augmentation scenarios is employed as can bee seen in figure 5.3. These augmentations are applied consistently during training to ensure the model encounters a wide spectrum of anomaly scenarios, thereby improving its robustness and generalization.

5.4.2 Model Implementation

The CutPasteNet model is designed to distinguish between normal and augmented data. It employs a ResNet-50 backbone for feature extraction, where pretrained weights on ImageNet are used to expedite training. The final fully connected layer of ResNet-50 or ResNet-18 is replaced with an identity layer, enabling the extraction of high-dimensional features for downstream tasks.
Following feature extraction, a projection head compresses these features into a lower-dimensional latent space using a sequence of fully connected layers, batch normalization, and ReLU activation. The projection head's dimensions for ResNet-18 [512, 512, 512, 512, 512, 512, 512, 512, 128] (for ResNet-50 replace the first value with 2048) ensure a progressive reduction of feature dimensionality, improving the model's ability to differentiate between normal and augmented samples.
A classification head maps the final projection output to logits, enabling binary classification. The model outputs both logits (for classification) and embeddings (for anomaly detection and localization).

5.4.3 Training Pipeline

The training process adheres to the methodology outlined in Section 4.2.1 and Section 4.2.3, focusing on the binary classification of two classes: **normal** (class 0) and **anomalous** (class 1). We do not apply CutPaste-3Way mentioned in 4.2.2. Synthetic anomalies are generated using the augmentation techniques described in the section before.
The cross-entropy loss function L_{CP} (Equation 4.11) is employed to optimize the model, guiding it to correctly classify normal and augmented samples. Optimization is performed using SGD, with hyperparameters tuned for stable and efficient training: a learning rate of 1×10^{-4}, momentum of 0.9, and weight decay of 3×10^{-5}.

A cosine annealing scheduler adjusts the learning rate dynamically over 15 epochs, balancing rapid initial learning with fine-tuning.
During training, the model processes batches of size 64, each containing a mix of normal and augmented samples. Training metrics, such as loss and accuracy, are logged in real-time using TensorBoard for performance monitoring. Additionally, automatic checkpointing ensures the best-performing model is saved for subsequent use. Early stopping was applied in all experiments to prevent the model from overfitting to vertical lines in the background, which are caused by camera noise.

5.4.4 Inference and Localization

Once trained, the model outputs feature embeddings that are processed using GDE to compute anomaly scores. This approach follows the methodology in Section 4.2.3, using the mean (μ) and covariance (Σ) of normal training data to identify deviations from the expected feature distribution. Anomalies are detected as samples with low-density estimates under the learned distribution of normal data.
During testing, the input image is divided into overlapping patches, and feature embeddings are extracted for each patch. Anomaly scores are computed for each patch using GDE, resulting in a detailed anomaly map. Gaussian smoothing is applied to these maps to propagate scores spatially, enhancing the continuity of the results.

5.4.5 Applying GRAD-CAM

We apply Grad-CAM to visualize the regions in an image that contribute most to the model's decision, thereby highlighting anomalies. To achieve this, we implement a Grad-CAM module that takes a trained model and a target layer. In our case, we select the early layer `encoder.conv1` to capture anomaly-related features before they are diluted through deeper layers. By computing gradients with respect to this layer, we extract activation maps that reveal the most influential features in the model's decision-making. These activations are visualized as heatmaps using `cv2.COLORMAP_HOT`, where darker colors indicate lower activation and lighter colors (ranging from red to white) signify stronger activations, corresponding to detected anomalies.
To aggregate the Grad-CAM heatmaps from smaller, overlapping patches, each patch's heatmap is mapped back to its corresponding location in a larger heatmap spanning the entire image. Due to overlap, multiple heatmaps contribute to the

same regions, introducing redundancy. To address this, we average the overlapping heatmaps to ensure no single patch disproportionately influences anomaly localization. Additionally, we apply a Gaussian smoothing filter to create seamless transitions between patches, eliminating sharp boundaries and enhancing the coherence of the final anomaly visualization.

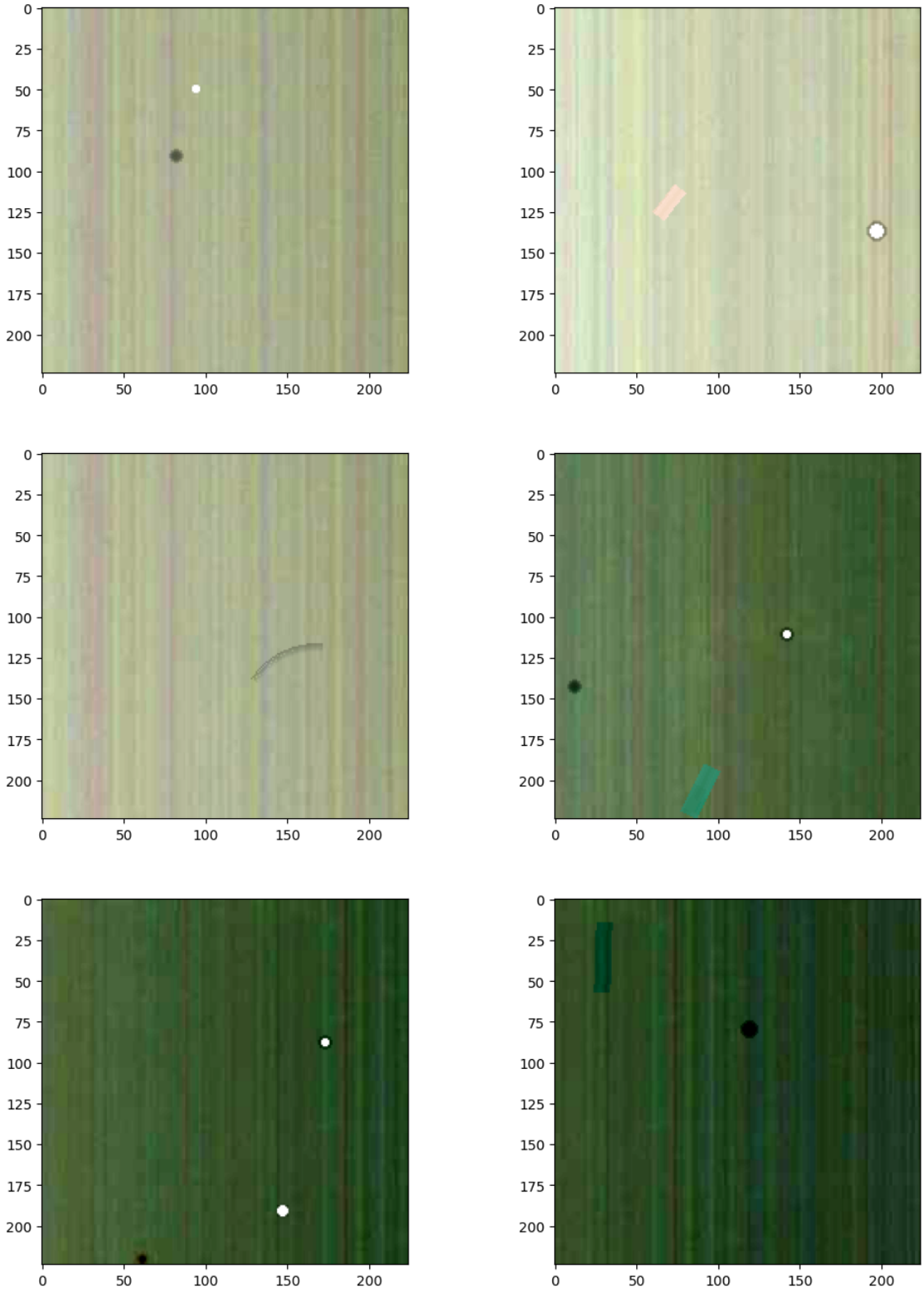


Figure 5.3: Different scenarios of the augmented images

CHAPTER 6

Results

This chapter begins by explaining the evaluation metrics used to assess the anomaly detection models. We then present the results for PatchCore and CutPaste, analyzing their performance across different configurations. Finally, we compare both methods, highlighting their strengths, weaknesses, and trade-offs in anomaly detection.

6.1 Evaluation Metrics

Evaluation metrics are crucial for assessing the performance of anomaly detection models, ensuring their reliability and robustness. This section outlines commonly used metrics: Area Under the Receiver Operating Characteristic Curve (AUC-ROC) (Fawcett, 2006), precision, accuracy, recall, and the confusion matrix. These metrics provide a comprehensive understanding of a model's predictive performance in identifying anomalies and normal cases.

6.1.1 Confusion Matrix

The confusion matrix is a tabular representation of a model's predictions, capturing the relationship between actual and predicted classes (Powers, 2011). It consists of four components:

- **True Positives (TP):** Instances correctly identified as anomalies.
- **True Negatives (TN):** Instances correctly identified as normal.
- **False Positives (FP):** Instances incorrectly classified as anomalies (Type I error).
- **False Negatives (FN):** Instances incorrectly classified as normal (Type II error).

The confusion matrix provides the foundation for calculating other performance metrics.

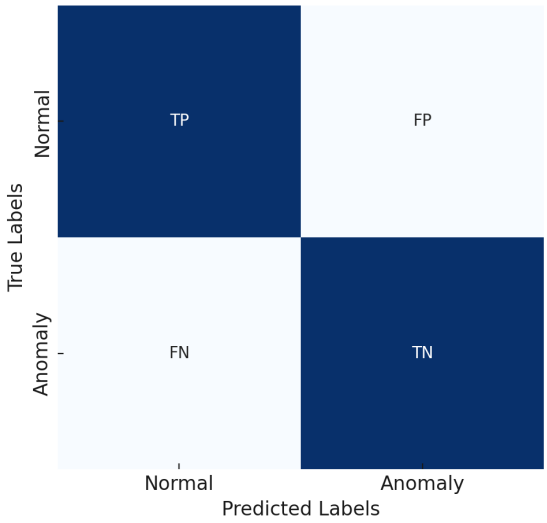


Figure 6.1: The Confusion Matrix

6.1.2 Accuracy

Accuracy measures the proportion of correctly classified instances, combining both normal and anomalous cases. It is expressed as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.1}$$

In image-based anomaly detection, accuracy can be misleading if the dataset is imbalanced, as is often the case. For instance, when anomalies represent only a small fraction of the data, a model predicting all instances as normal could achieve high accuracy while failing to detect anomalies effectively. Therefore, accuracy should be interpreted alongside other metrics.

6.1.3 Precision

Precision quantifies the proportion of correctly identified anomalies among all predicted anomalies. It is particularly relevant when false positives are costly. The formula is:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{6.2}$$

Higher precision indicates fewer false positives. For image anomaly detection, high precision ensures that most detected anomalies are genuinely defective images, reducing the time spent reviewing false alarms. This is critical in applications like manufacturing, where false positives may interrupt workflows or lead to unneces-

sary investigations.

6.1.4 Recall

Recall, also known as sensitivity or the true positive rate, measures the proportion of actual anomalies correctly identified. It emphasizes minimizing false negatives and is given by:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{6.3}$$

In image anomaly detection, high recall is essential to ensure that defective or anomalous images are not overlooked. High recall ensures that most anomalies are detected, although it may come at the cost of increased false positives.

6.1.5 Area Under the Receiver Operating Characteristic Curve (AUC-ROC)

The Receiver Operating Characteristic (ROC) curve plots the true positive rate against the false positive rate at various thresholds (Fawcett, 2006). The AUC-ROC summarizes the model's ability to distinguish between classes across all thresholds. It ranges from 0 to 1, with higher values indicating better discriminatory performance. An area under the curve close to 0.5 suggests random guessing, while a value near 1 indicates excellent performance.
For image anomaly detection, the AUC-ROC is valuable in evaluating the model's ability to differentiate anomalous images from normal ones across varying thresholds. A high AUC-ROC indicates that the model is effective in identifying subtle differences in features between normal and anomalous images, even when these anomalies are rare or visually challenging to detect.

6.1.6 t-Distributed Stochastic Neighbor Embedding (t-SNE)

t-Distributed Stochastic Neighbor Embedding (t-SNE) (Maaten and Hinton, 2008) is a dimensionality reduction technique for visualizing high-dimensional data. It maps complex feature spaces into two or three dimensions, preserving local structures while minimizing distortions.
Applied to anomaly detection, t-SNE reveals clustering patterns in learned feature representations. Well-separated clusters indicate effective differentiation between normal and anomalous instances, whereas overlapping regions suggest poor separation and potential areas for model improvement.

These metrics collectively provide insights into the model's performance from different perspectives, allowing researchers to make informed decisions about the trade-offs between false positives and false negatives.

6.2 Patchcore Results

In this section, we present the results of our PatchCore-based anomaly detection experiments. Our evaluation focuses on two key aspects: quantitative performance, where we analyze key metrics such as accuracy, precision, recall, and AUC-ROC, and qualitative assessment, where we visualize the model's performance in detecting anomalies for whole images as well as on a patch level.

6.2.1 Quantitative Results

We conducted six experiments using PatchCore, with three experiments for each category (review 3.1): **White** and **White with Edges**. Within each category, we evaluated three different backbone architectures: ResNet18, ResNet50, and ResNet50 with blur. For all the experiments, the input image size is 224, the stride is 112 and the features are extracted from the second and third layers. We can see the parameters and performance metrics of the experiments in the following table:

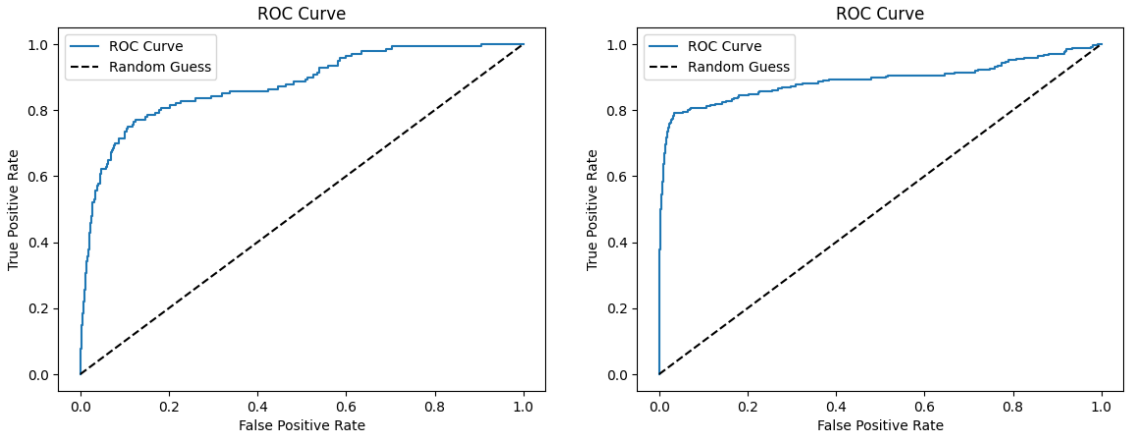
No.	Data Category	Backbone	Blur	ROC	Accuracy	Precision	Recall	Confusion Matrix
1	White with edges	Resnet50	With	74.51%	0.7143	0.0457	0.7571	5511 2212 34 106
2	**White with edges**	**Resnet50**	**Without**	**87.63%**	**0.8730**	**0.1005**	**0.7714**	**6757 966** **32 108**
3	**White**	**Resnet50**	**Without**	**89.06%**	**0.9608**	**0.3852**	**0.7909**	**8618 308** **51 193**
4	White	Resnet50	With	79.63%	0.8609	0.1277	0.7254	7718 1208 67 177
5	White with edges	Resnet18	Without	74.03%	0.8561	0.0985	0.5409	7719 1207 112 132
6	White	Resnet18	Without	76.62%	0.8323	0.0936	0.6106	7484 1442 95 149

Table 6.1: Resuts of the experiments for PatchCore using different parameters (bold highlights the best results for each category)

Blurring was implemented using OpenCV's `fastNlMeansDenoisingColored` function, which reduces noise in color images while preserving details by denoising

the luminance and color components separately. Key parameters include **h** (filter strength), **templateWindowSize** (patch size for weight calculation), and **searchWindowSize** (area for computing the weighted average). Adjusting these values helps balance noise reduction and detail retention.

The evaluation results highlight the influence of model depth, data category, and blur preprocessing on anomaly detection performance. The ResNet50 models (experiments 1-4) generally outperformed ResNet18 (experiments 5-6) in terms of ROC and accuracy, demonstrating the benefits of a deeper network. Among all configurations, For the White with Edges category, experiment 2 (ResNet50 without blur) achieved the highest ROC (0.8763) and accuracy (0.8730). Similarly, in the White category, experiment 3 (ResNet50 without blur) yielded the highest ROC (0.8906) and accuracy (0.9608). Their ROC graphs can be seen in figure 6.7. Both ROC curves show a steep initial rise, indicating that the models quickly achieve a high true positive rate with a low false positive rate, which is crucial for anomaly detection. The curves gradually flatten as the false positive rate increases, showing diminishing improvements in detecting anomalies. The first model, with an Area under the curve (AUC) of 0.83, rises strongly but slightly less aggressively than the second, meaning it performs well but with some limitations in separating anomalies from normal instances. The second model, with an AUC of 0.85, has a steeper initial rise, indicating faster anomaly detection with fewer false positives and slightly better overall performance.



(a) ROC graph of experiment 2

(b) ROC graph of experiment 3

Figure 6.2: ROC graph for experiment 2 and 3 of PatchCore

Blur was introduced to assess its impact on performance, but in both "White" (experiments 3 vs. 4) and "White with edges" (experiments 1 vs. 2) categories, applying blur resulted in lower ROC and accuracy scores. This indicates that sharp, detailed

features are more beneficial for anomaly detection in this scenario. Precision remained relatively low across all experiments, reflecting a trade-off with recall, where higher recall values suggest better anomaly detection but at the cost of increased false positives. The confusion matrices further show that while models classified the majority class well, misclassification rates varied depending on the configuration. Overall, these results suggest that using a deeper network and avoiding blur preprocessing leads to better anomaly detection performance.

6.2.2 Qualitative Results

For the evaluation of patches and full-image performance, we selected the best experiments based on AUC-ROC performance. Specifically, we used Experiment 2 (White with Edges) and Experiment 3 (White), as these provided the highest ROC scores (marked in bold). To assess the performance of these models, we selected one representative images from each category.

White Category

The following images belong to the White category, corresponding to Experiment 3 (see Figure 6.3). In Figure 6.3a, we see the original image. The ground truth image in Figure 6.3b highlights the anomaly, which is located in the mid-lower part of the image (white coloured). In Figure 6.3c, the anomaly map illustrates how the evaluation image was partitioned into smaller patches, where darker pixels represent the detected anomaly. The background and label areas are excluded from the evaluation, shown in dark blue. We can clearly see that the anomaly is very well detected in the anomaly map as black pixels. Additionally, four points are marked as anomalies that do not appear in the ground truth, indicating false positives. These false detections likely stem from small black dots that are not considered anomalies according to the manufacturer, which is why they are not highlighted in the ground truth. Finally, Figure 6.3d overlays the anomaly map onto the original image, allowing for precise localization of the detected anomaly in relation to the original image, demonstrating an effective detection.

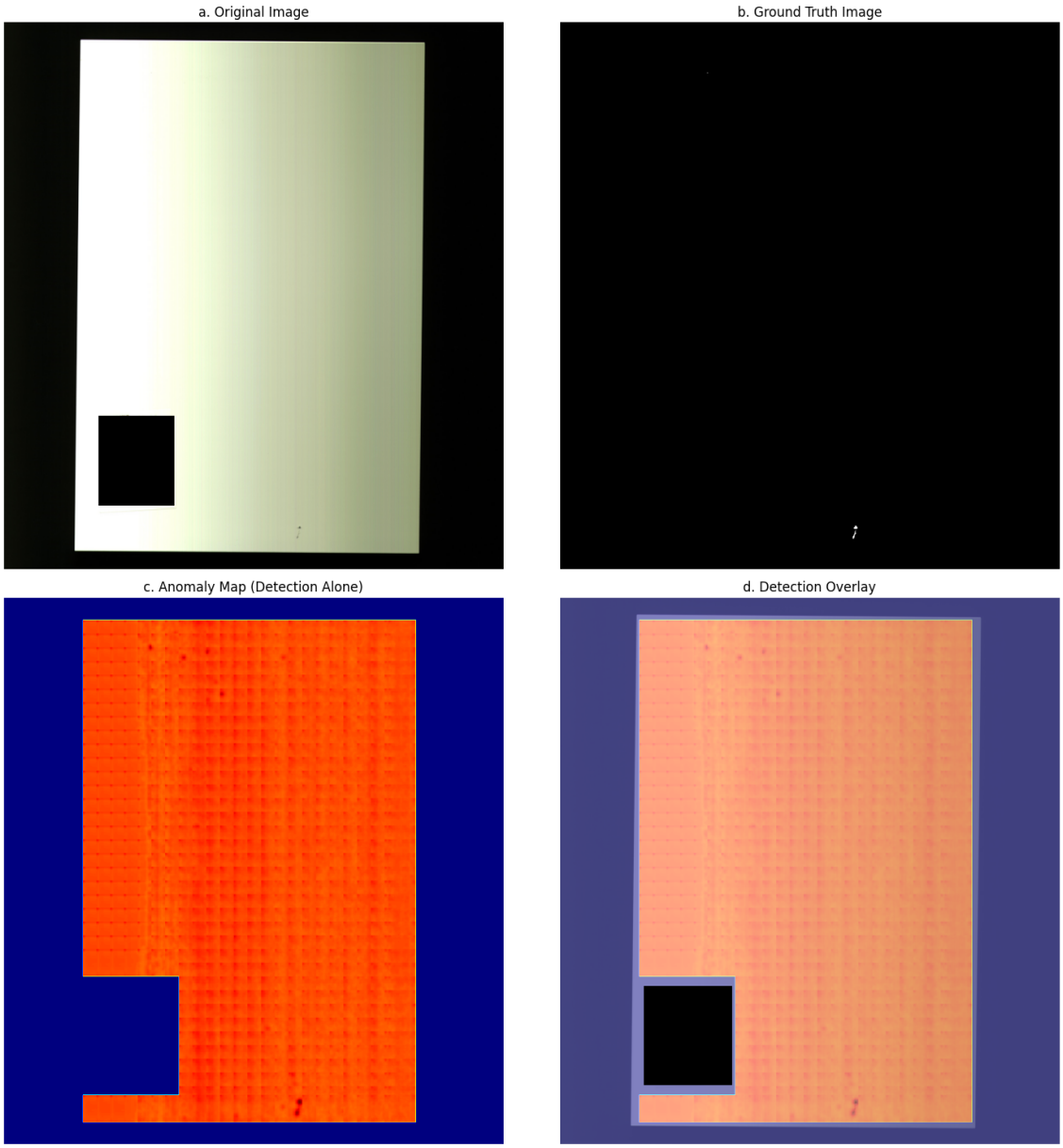


Figure 6.3: Sample image from the White category (ID: 231117-0569) - PatchCore

Further images of the same experiment can be seen in the Appendix A.

White with Edges Category

The following images belong to the White with Edges category, corresponding to Experiment 2. The original image can be seen in Figure 6.4a. The ground truth image in Figure 6.4b reveals two anomalies in the upper right part of the image.
In Figure 6.4c, similar to the previous case, the anomaly map illustrates how the evaluation image was partitioned into smaller patches, with darker pixels represent-

ing detected anomalies. The background and label areas are excluded from the evaluation, shown in dark blue. The anomalies are clearly detected with high intensity. There are also several smaller black dots that are detected as anomalies, but according to the manufacturer they do not represent anomalies.
Additionally, the inner corners of the raised rectangular structure are also flagged as anomalies, even though they are not actual defects. This likely occurs due to lighting variations, where the angle and direction of the light create darker regions. However, when the anomaly map is overlaid on the original image in Figure 6.4d, the true anomalies stand out distinctly, demonstrating a well-performing detection.

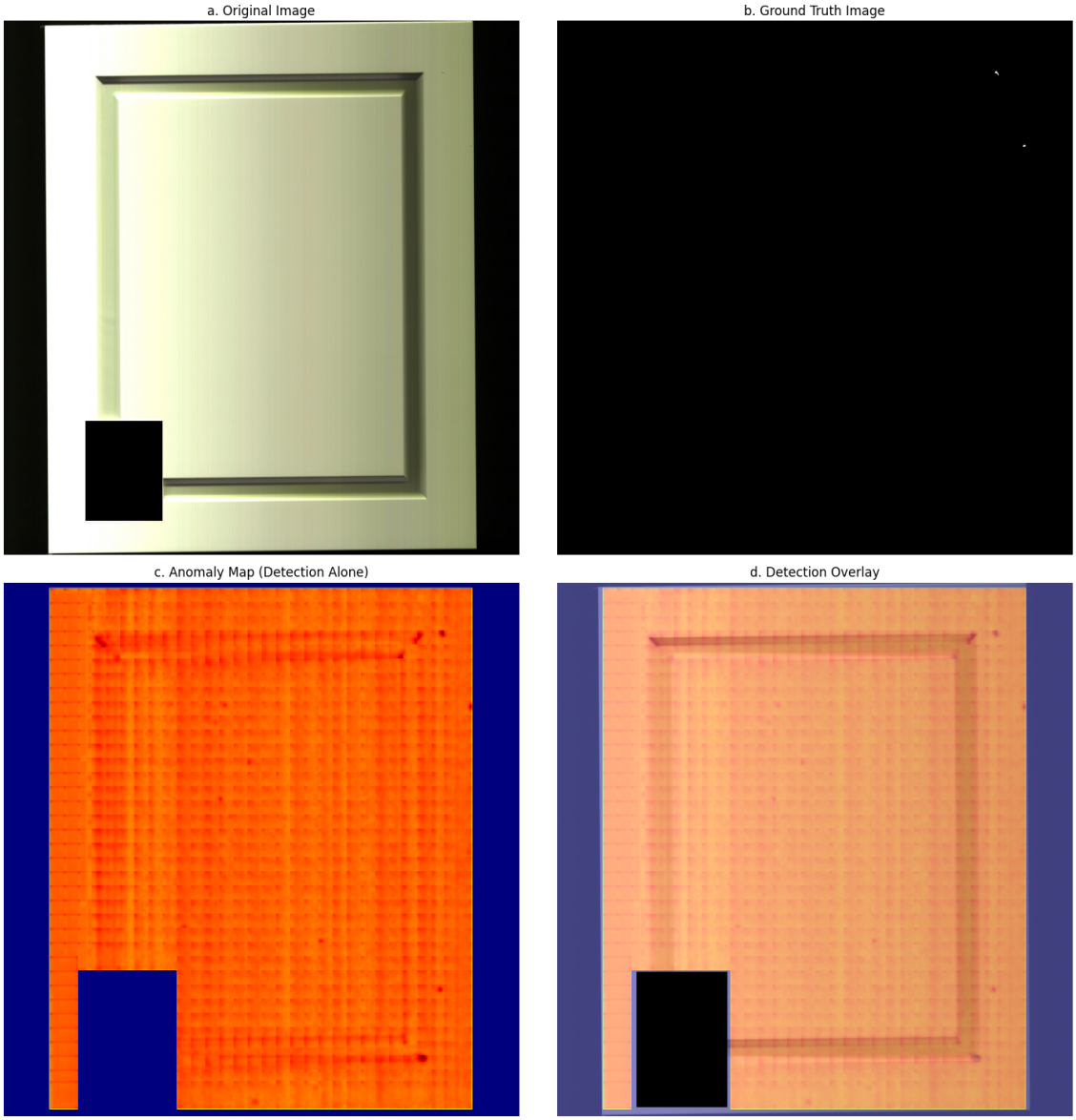


Figure 6.4: Sample image from the White with Edges category (ID: 231117-325) - PatchCore

Further images of the same experiment can be seen in the Appendix A.

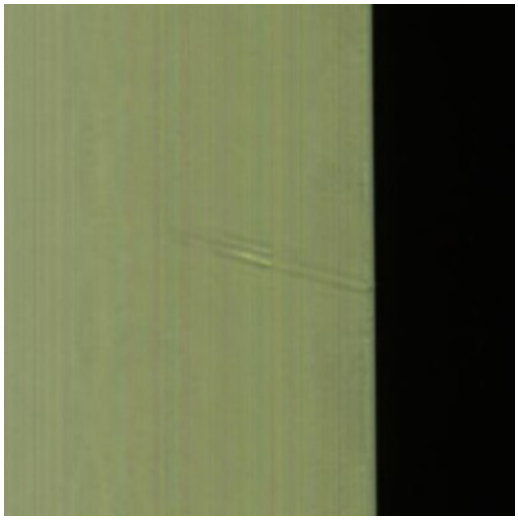
6.2.3 Performance review on patch level

We also apply the models on a patch level and assess their performance on the anomalies introduced in the figure 6.5. We represent the image here again for comparison, and can see the following:
The models demonstrated strong performance in detecting anomalies, with the highest detection accuracy observed for **Black Scar** and **Black Hole** anomalies. This can be attributed to the fact that these **localized anomalies** exhibit distinct, well-defined boundaries, in contrast to the more diffuse anomalies like **Splash** or **Dirt**, which are more spread out and less sharply delineated. Overall, the model showed exceptional sensitivity to **small anomalies**, highlighting its ability to capture subtle texture variations. These results indicate that the algorithm is highly suitable for industrial applications where fine-grained anomaly detection is crucial.

(a) Black hole anomaly

(b) Bump anomaly

(c) Splash anomaly



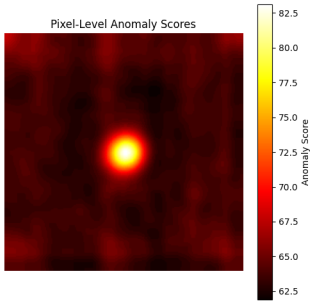
(d) White scar anomaly

(e) Irregular anomaly

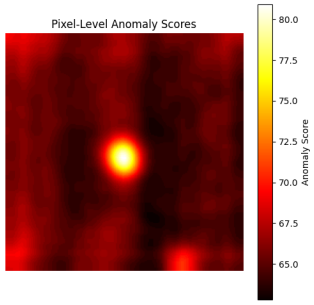
(f) Black scar anomaly

(g) dirt anomaly

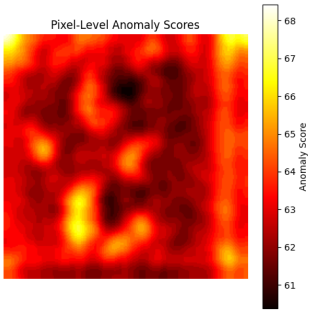
Figure 6.5: Overview of the different anomalies



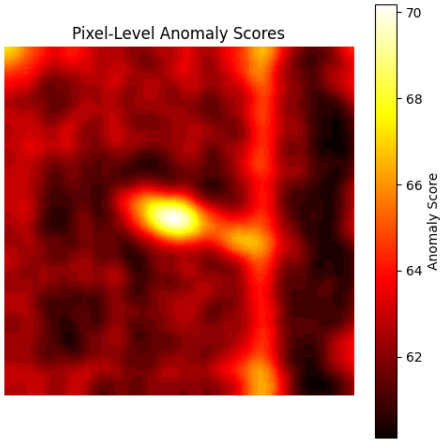
(a) Anomaly Score Heatmap - Black hole anomaly



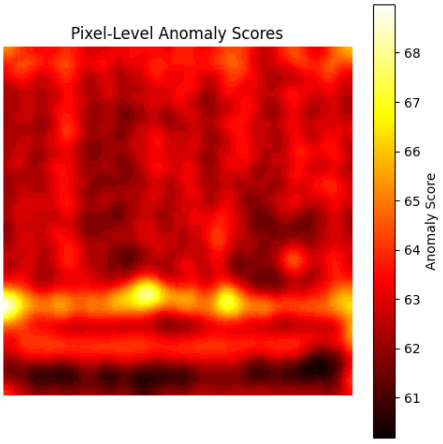
(b) Anomaly Score Heatmap - Bump anomaly



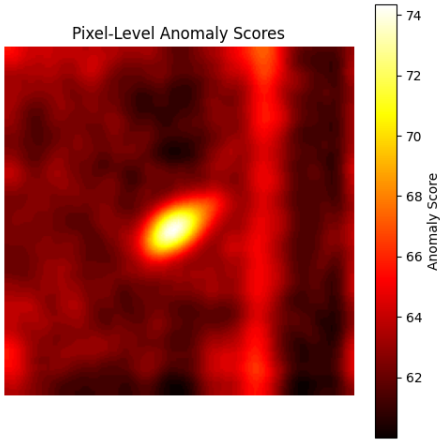
(c) Anomaly Score Heatmap - Splash anomaly



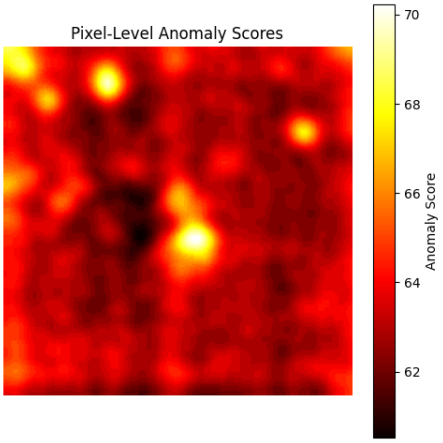
(d) Anomaly Score Heatmap - White scar anomaly



(e) Anomaly Score Heatmap - Irregular anomaly



(f) Anomaly Score Heatmap - Black scar anomaly



(g) Anomaly Score Heatmap - dirt anomaly

Figure 6.6: Anomaly heat maps of the different anomalies - PatchCore

6.3 CutPaste Results

Similary to the PatchCore Results, we present our CutPaste Results. Our evaluation focuses on two key aspects: quantitative performance, where we analyze key metrics such as accuracy, precision, recall, and AUC-ROC, and qualitative assessment, where we visualize the model's performance in detecting anomalies for whole images as well as on a patch level.

6.3.1 Quantitative Results

A total of eight experiments were conducted, with four assigned to each category: white and white with edges. Within each category, we evaluated ResNet-18 and ResNet-50, testing both models with and without blurring. All experiments were conducted with an input image size of 224 × 224, a stride of 112, a learning rate of 1e-4, a batch size of 64, and a CutPaste transparency of 15 to ensure seamless blending with the images. SGD was used as the optimizer. we can the results in table 6.2

No.	Data Category	Backbone	Blur	ROC	Accuracy	Precision	Recall	Confusion Matrix
1	White with edges	Resnet50	Without	79.50%	0.7116	0.0445	0.7428	5492 2231 36 104
2	**White with edges**	**Resnet18**	**Without**	**82.56%**	**0.893**	**0.1012**	**0.6357**	**6933 790** **51 89**
3	White	Resnet50	Without	78.67%	0.8004	0.0839	0.6557	7180 1746 84 160
4	White	Resnet18	Without	80.90%	0.8729	0.1271	0.64344	7848 1078 87 157
5	White with edges	Resnet50	With	64.81%	0.6563	0.028	0.5428	5085 2638 64 76
6	**White**	**Resnet50**	**With**	**84.82%**	**0.9256**	**0.2177**	**0.6926**	**8319 607** **75 169**
7	White with edges	Resnet18	With	79.90%	0.9261	0.1402	0.6142	7196 527 54 86
8	White	Resnet18	With	78.54%	0.8074	0.0886	0.6721	7240 1686 80 164

Table 6.2: Results of the experiments for CutPaste using different parameters (bold highlights the best results for each category)

Across all experiments, **ResNet18 consistently outperforms ResNet50 in the no-blur setting** for both "White" and "White with edges." For instance, in the no-blur "White with edges" scenario, ResNet18 achieves higher AUC-ROC and accuracy

than ResNet50, even though ResNet50 exhibits slightly better recall. A similar trend appears in the no-blur "White" category, where ResNet18 again surpasses ResNet50 in multiple metrics.

When **blur is introduced**, ResNet50 sees a substantial boost on the "White" dataset, achieving the highest overall AUC-ROC (84.82%). By contrast, in "White with edges" under blur, ResNet50 performs worse, whereas ResNet18 fares better and attains the top accuracy in that category.

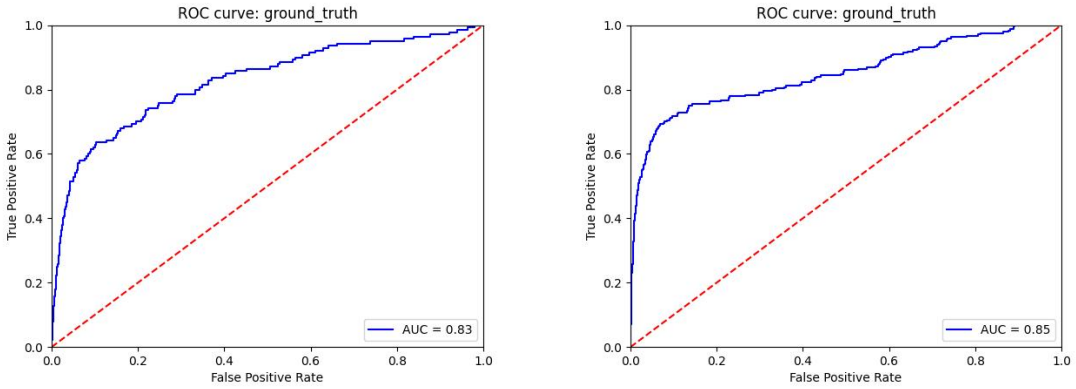
Blur using OpenCV's `fastNlMeansDenoisingColored` function was initially applied in CutPaste to reduce Grad-CAM's tendency to overfit to vertical background lines in images, enhancing its ability to detect anomalies. However, it also unexpectedly improved overall results.

Breaking it down by data category:

- **White with edges:** ResNet18 *without blur* achieves the highest AUC-ROC, while ResNet18 *with blur* attains the best accuracy. The model was trained for 6 epochs with 1800 steps, applying early stopping.
- **White:** ResNet50 *with blur* yields the highest AUC-ROC and accuracy, though ResNet18 *without blur* still outperforms ResNet50 *without blur*. Training was conducted for 1 epoch with 2500 steps, also applying early stopping.

The ROC graph for the best experiments (2 and 6) can be seen in figure 6.7 and the glstsne graphs can be seen in figure 6.8.

Overall, these findings demonstrate that **ResNet18 is generally more effective without blur**, whereas **ResNet50 benefits significantly from blur**—particularly for the "White" dataset.

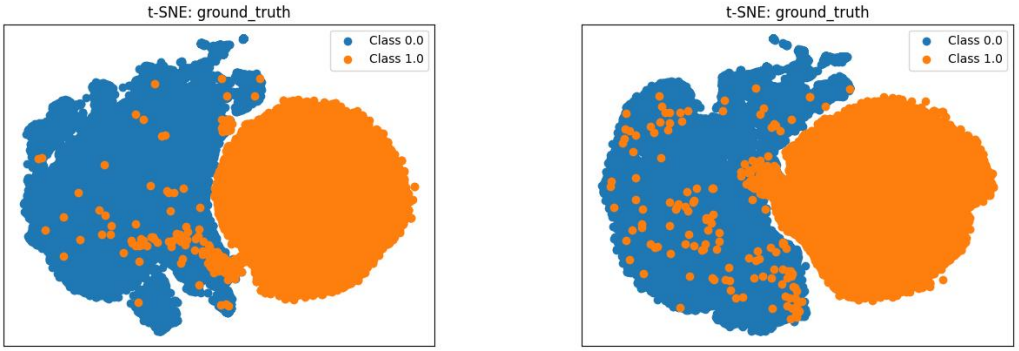


(a) ROC graph of experiment 2

(b) ROC graph of experiment 6

Figure 6.7: ROC graphs for experiments 2 and 6 of CutPaste

The ROC curve in Experiment 2 shows an AUC of 0.83, indicating a strong but not optimal model. The curve rises steeply at first but flattens out, suggesting some misclassification of anomalies. In Experiment 6, the AUC improves to 0.85, with a more convex curve that indicates a better balance between true positives and false positives. The model in this experiment achieves a lower false positive rate, making it more reliable for anomaly detection.
Regarding the t-SNE plots, both show two main clusters (blue vs. orange), but Experiment 2 has a tighter boundary between normal and anomalous points, with fewer orange points scattered in the blue cluster. In Experiment 6, there's a smoother boundary and more overlap, suggesting that anomalies are more interspersed among the normal data.



(a) tSNE graph of experiment 2 (b) tSNE graph of experiment 6

Figure 6.8: tSNE graphs for experiments 2 and 6 of CutPaste

6.3.2 Qualitative Results

White Category

The following images belong to the *White* category, corresponding to Experiment 2 (see Figure 6.9). In Figure 6.9a, we observe the original image. The ground truth image, shown in Figure 6.9b, highlights the anomalies as white-colored regions, revealing the presence of two distinct anomalies.
Figure 6.9c presents the anomaly map, which illustrates how the evaluation image was partitioned into smaller patches. Areas corresponding to the background and labels are excluded from the evaluation and are depicted in black. Notably, the patches on the left side exhibit higher pixel intensity according to Grad-CAM, indicating that the model is highly influenced by brightness. This effect persists despite

the data being normalized both during training and before being processed by Grad-CAM. In contrast, the middle and right portions of the image display relatively lower intensity, suggesting less influence from brightness.
The detected anomalies appear as small white dots, aligning well with their locations in the ground truth image. However, a false positive is observed in the middle-right region. Finally, Figure 6.9d overlays the anomaly map onto the original image, providing precise localization of the detected anomalies in relation to their original context.

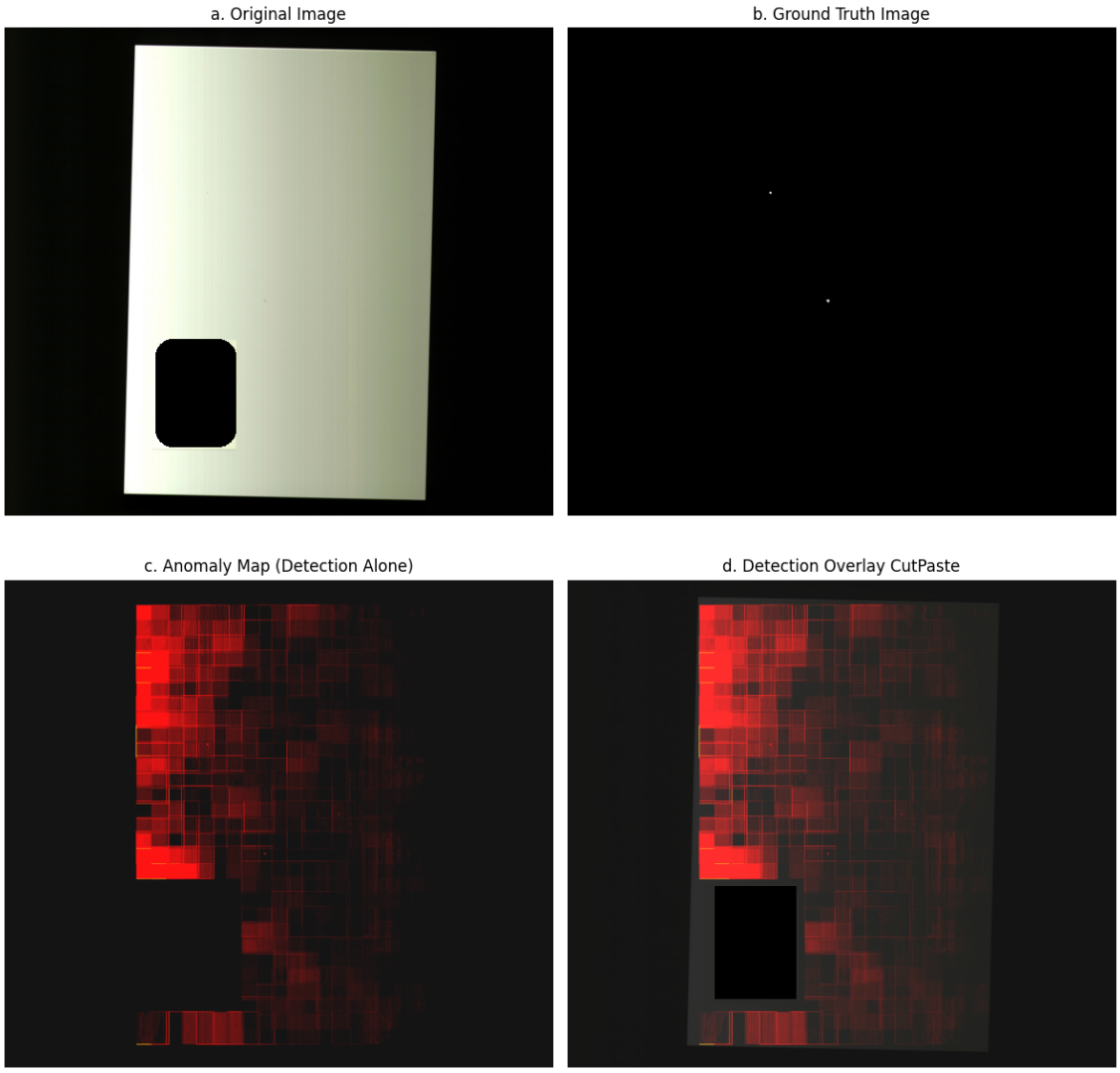


Figure 6.9: Sample image from the White category (ID: 231117-0443) - CutPaste

White with edges

The following images belong to the *White with edges* category, corresponding to Experiment 6 (see Figure 6.10). In Figure 6.10a, we observe the original image. The ground truth image, shown in Figure 6.9b, highlights the anomalies, with a main cluster located in the center and several smaller anomalies distributed around it, particularly towards the left.
Figure 6.10c presents the anomaly map, which illustrates how the evaluation image was partitioned into smaller patches. As before, areas corresponding to the background and labels are excluded from the evaluation and are depicted in black. Unlike previous cases, this model appears to be less affected by brightness variations. However, some light-colored patches are still present that do not correspond to actual anomalies.
The main anomaly is clearly visible in the center with a lighter intensity, while the smaller anomalies are also detected, though they appear less distinct. Additionally, we observe a small false positive on the right side.
Finally, Figure 6.10d overlays the anomaly map onto the original image, providing precise localization of the detected anomalies in relation to their original context.

6.3.3 Performance review on a patch level

The models demonstrated acceptable performance in detecting anomalies, with the highest detection accuracy observed for **black hole, bump, and dirt** anomalies. This can be attributed to these anomalies having better representation in the augmentations introduced in Section 5.4.1.
Detection performance for **irregular anomalies** and **black scar anomalies** was reasonable, while anomalies in the **white** category were barely visible. The **worst detection performance** was observed for **splash** anomalies.
Overall, the model performed better on **smaller anomalies** but struggled with **anomalies that blended into the background**, leading to lower detection accuracy in such cases.

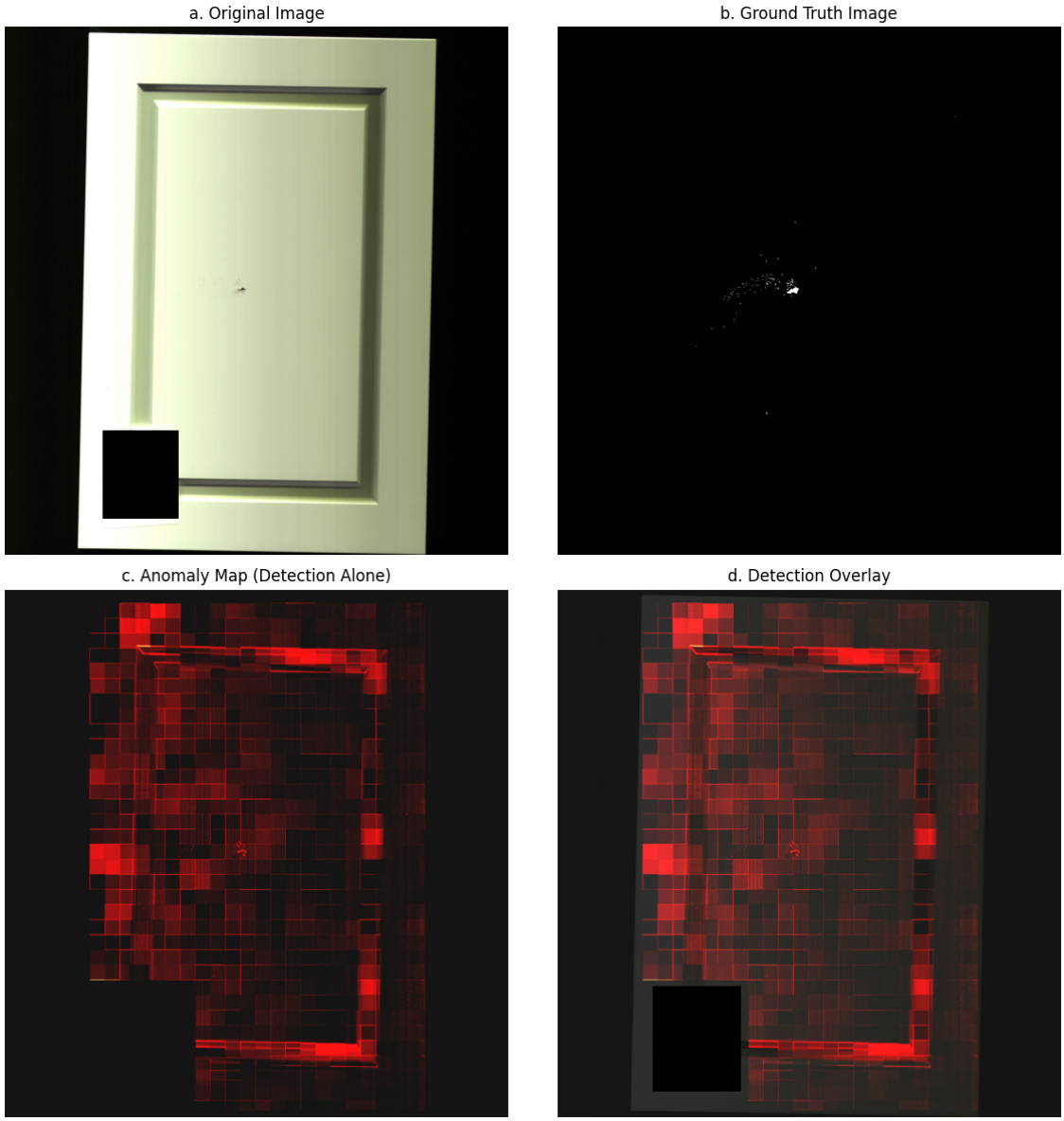
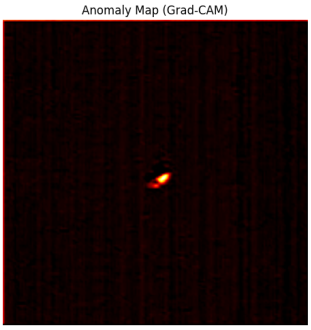
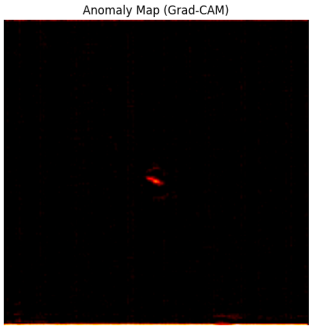


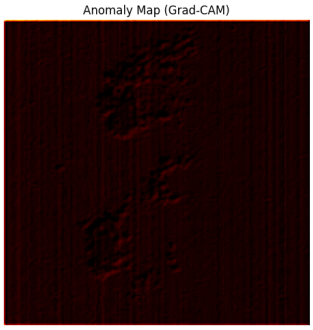
Figure 6.10: Sample image from the White with edges category (ID: 231117-639) - CutPaste



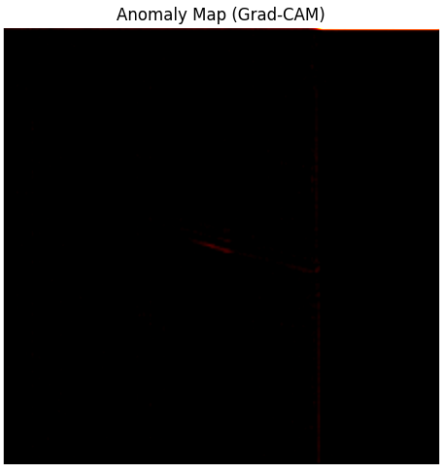
(a) Anomaly Score Heatmap - Black hole anomaly



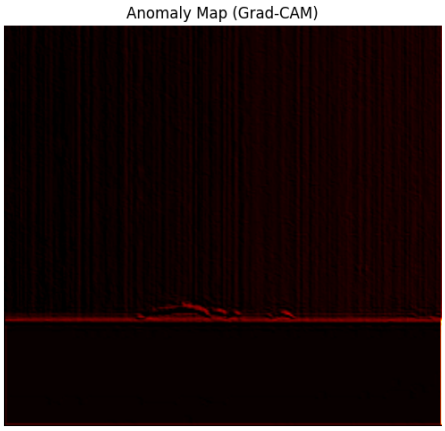
(b) Anomaly Score Heatmap - Bump anomaly



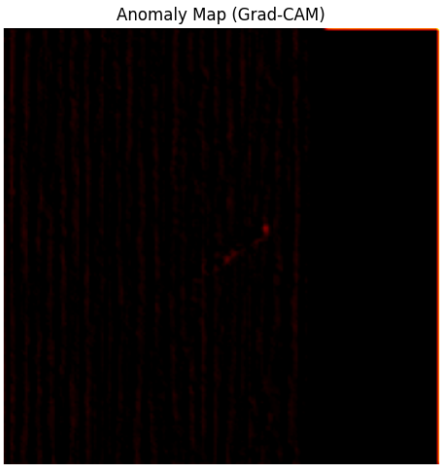
(c) Anomaly Score Heatmap - Splash anomaly



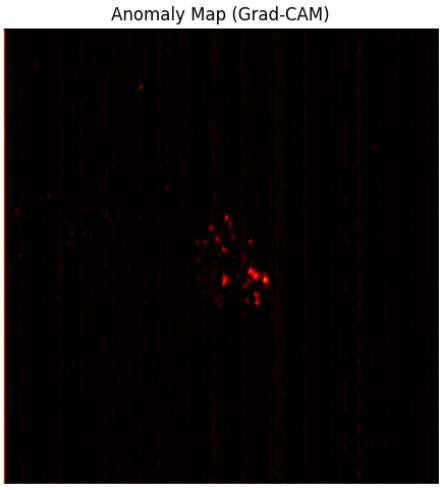
(d) Anomaly Score Heatmap - White scar anomaly



(e) Anomaly Score Heatmap - Irregular anomaly



(f) Anomaly Score Heatmap - Black scar anomaly



(g) Anomaly Score Heatmap - dirt anomaly

Figure 6.11: Anomaly heat maps of the different anomalies - CutPaste

6.4 Comparing the two Algorithms

6.4.1 Comparing the metrics

Table 6.3 compares PatchCore and CutPaste in the White category. Here, PatchCore attains higher ROC, precision, and recall, indicating that it not only detects more anomalies but also produces fewer false positives. This balance underscores PatchCore's robustness in anomaly detection. CutPaste, by contrast, shows lower precision and recall, implying it identifies fewer anomalies and mislabels more normal samples as anomalous. Additionally, PatchCore's accuracy surpasses CutPaste's, emphasizing PatchCore's more reliable overall performance in this category.

Method	ROC	Precision	Recall	Accuracy	Confusion Matrix
PatchCore	**89.06%**	**0.3852**	**0.7909**	**0.9608**	**[8618 308; 51 193]**
CutPaste	84.82%	0.2177	0.6926	0.9256	[8319 607; 75 169]

Table 6.3: Comparison of PatchCore and CutPaste performance metrics for White category

Table 6.4, PatchCore maintains a strong ROC and recall, indicating consistent anomaly detection capabilities. However, its precision drops significantly, suggesting an increase in false positives when edges are present. Meanwhile, CutPaste demonstrates lower recall but achieves a marginally higher precision and a relatively higher accuracy. These results imply that while PatchCore continues to detect anomalies more effectively in terms of recall, CutPaste manages fewer false positives overall in this context, leading to a slight edge in accuracy.

Method	ROC	Precision	Recall	Accuracy	Confusion Matrix
PatchCore	**87.63%**	**0.1005**	**0.7714**	0.8730	**[6757 966; 32 108]**
CutPaste	82.56%	**0.1012**	0.6357	**0.893**	[6933 790; 51 89]

Table 6.4: Comparison of PatchCore and CutPaste performance metrics for white with edges category

6.4.2 Comparing images

In Figure 6.12, which belongs to the White category, the original image is shown in Figure 6.12a, while Figure6.12b presents the ground-truth image containing two anomalies: one in the center and another in the lower-right area. Figure 6.12c shows the CutPaste anomaly map, where brighter regions indicate high model activation according to Grad-CAM. Although the central anomaly appears only faintly, the anomaly in the lower-right corner stands out more clearly.
By contrast, the PatchCore anomaly map in Figure 6.12d is less chaotic and makes the anomalies more noticeable as darker pixels. The anomaly in the lower-right corner appears slightly lighter than the one in the center, but both anomalies are readily identifiable. In conclusion, although both methods successfully detect the anomalies, PatchCore's map is clearer and easier to interpret overall.

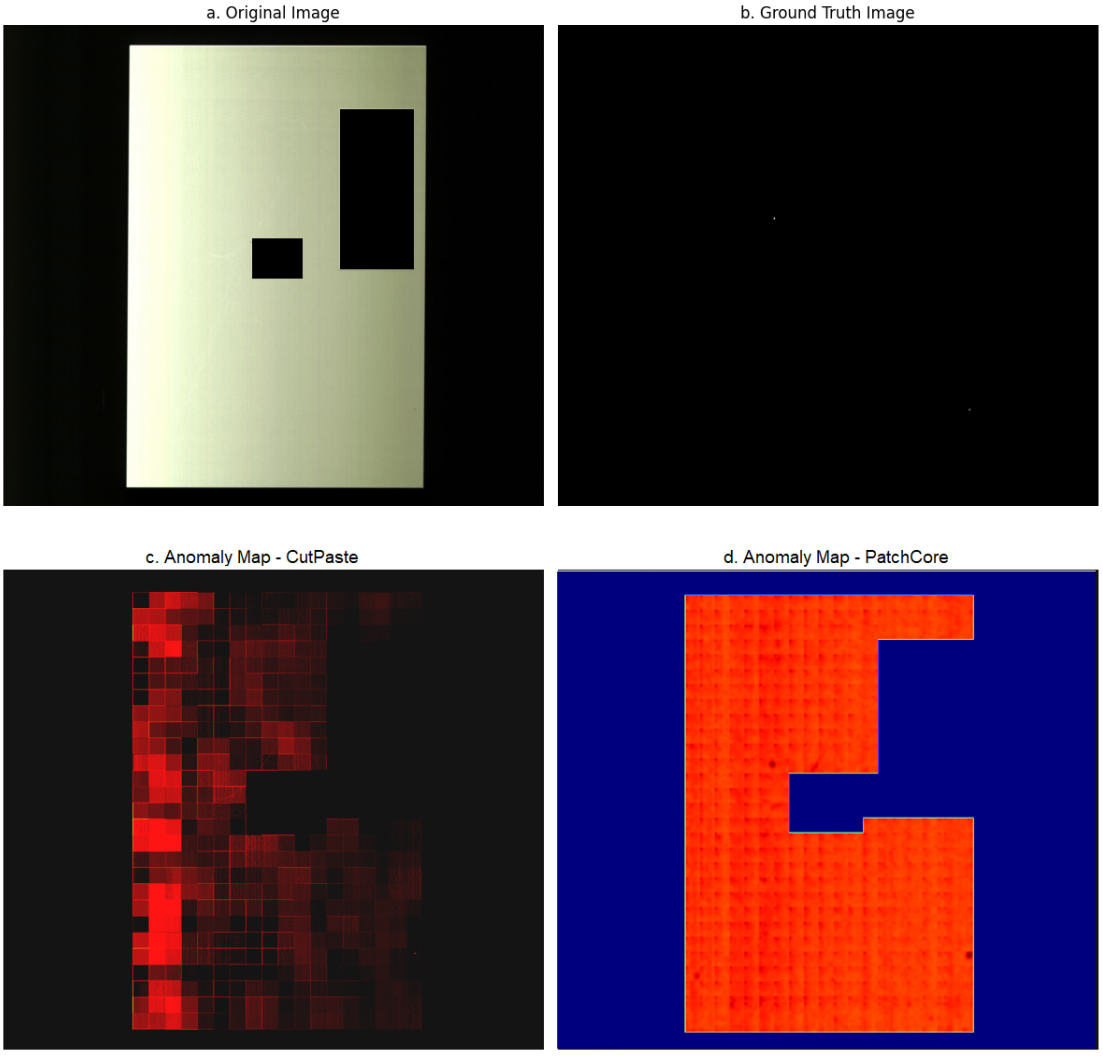


Figure 6.12: Sample image from the White category (ID: 231117-322) - Comparing CutPaste and PatchCore

In the white with edges scenario, the original image is shown in Figure 6.13a. According to the ground-truth image in Figure 6.13b, there are two anomalies: a larger one in the middle-lower part and a smaller one in the lower-right part.
In Figure 6.13c, the CutPaste anomaly map is less influenced by the bright background compared to other white category examples, but random patches with elevated brightness still appear. The larger anomaly in the middle-lower region is distinctly highlighted, whereas the smaller anomaly in the lower-right corner is not detected.
By contrast, the PatchCore anomaly map in Figure 6.13d clearly shows both anomalies, with the larger one appearing at a higher intensity. Although the smaller anomaly is also identified, its weaker response is indicated by fewer dark pixels. Overall, PatchCore produces more accurate and interpretable results than CutPaste.

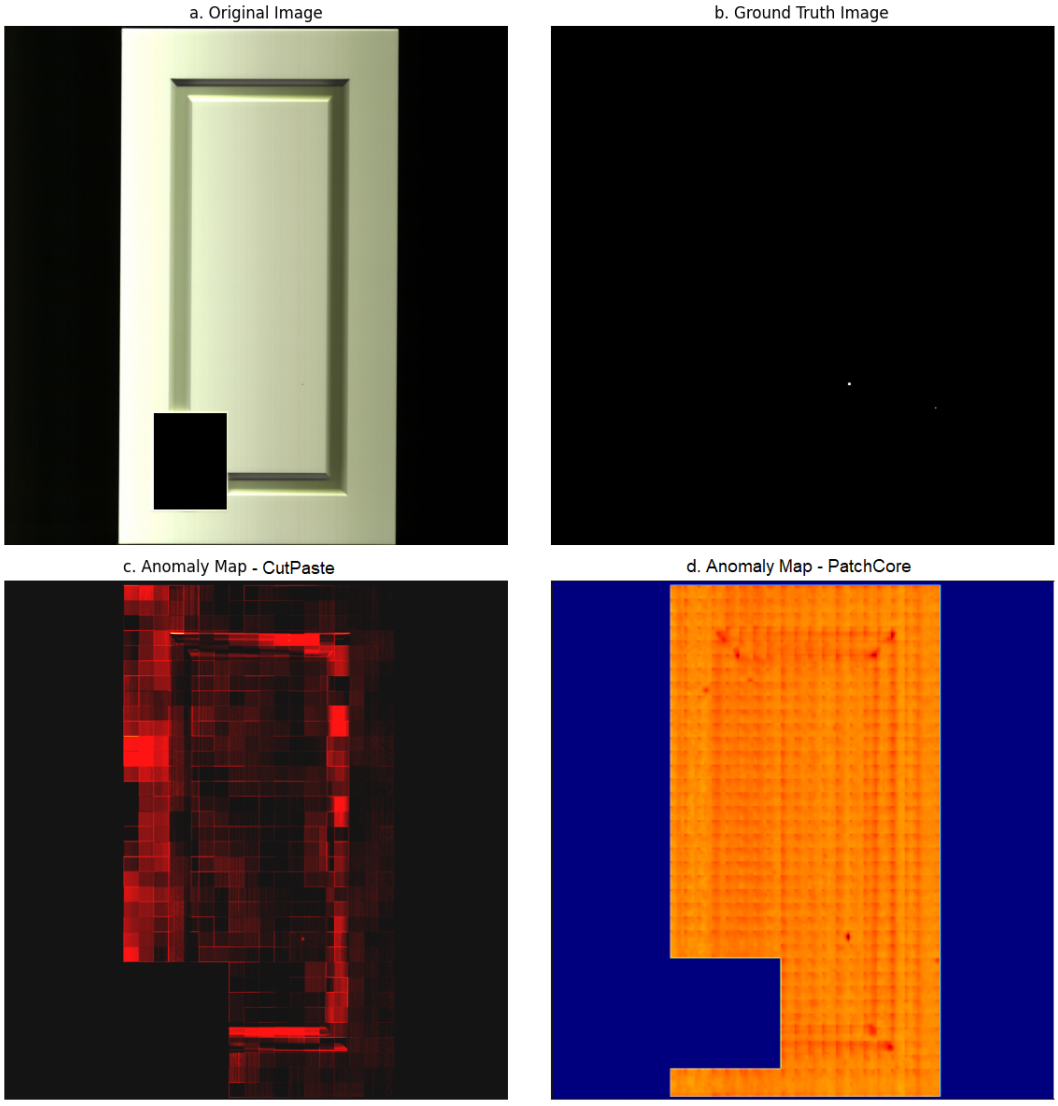


Figure 6.13: Sample image from the White with edges category (ID: 231117-372) - Comparing CutPaste and PatchCore

CHAPTER 7

Conclusion

Anomaly detection using deep learning plays a crucial role in quality control within manufacturing, particularly for detecting subtle defects in materials that are challenging to identify with the naked eye. This thesis aimed to evaluate the effectiveness of two state-of-the-art unsupervised anomaly detection methods, PatchCore and CutPaste, in identifying surface defects in industrial board manufacturing. Unlike their previous applications on benchmarking datasets such as MVTec, this research focused on real-world images captured in a kitchen carpentry production environment, where anomalies are often small and difficult to distinguish.

Our experimental results demonstrate that **PatchCore consistently outperforms CutPaste in both the White and White with edges categories**, demonstrating higher AUC-ROC scores, precision, and recall. Notably, the best-performing PatchCore model (ResNet50 without blur) attained AUC-ROC scores of **89.06%** in the White category and **87.63%** in White with edges, which is a clear evidence of its robust detection and classification capabilities. CutPaste, though less effective in recall, demonstrated a slight advantage in accuracy within the White with edges category, implying fewer false positives when edges are present.

From a qualitative perspective, **PatchCore produced clearer and more interpretable anomaly maps compared to CutPaste**, offering precise localization of defects with minimal noise. CutPaste, though effective, exhibited inconsistencies due to its reliance on Grad-CAM, which introduced sensitivity to brightness variations.

This thesis bridges the gap between research and practical industrial applications by demonstrating the feasibility of deploying advanced unsupervised anomaly detection methods in real-world manufacturing environments. The findings contribute valuable insights into selecting the most suitable deep learning-based anomaly detection approach for industrial surface inspection, supporting the ongoing pursuit of defect-free, high-quality production.

CHAPTER 8

Outlook

Future improvements to the system could address several limitations encountered during this study.

First, using a better camera could significantly reduce noise, such as the RGB vertical lines observed in the background, which caused the model to overfit to these unintended features. This overfitting was particularly problematic when early stopping was not employed, as the model focused on overly fine details that were not relevant to the task.

Second, upgrading to a better GPU would allow for more comprehensive training. Due to hardware constraints, it was not feasible to train the PatchCore model on the entire dataset using the required stride. Instead, a larger stride was used, which skipped some patches and potentially impacted model performance. Enhanced hardware could enable training with smaller strides and fuller dataset utilization.

Additionally, experimenting with different early stopping criteria, a wider range of epochs, and alternative optimization strategies could yield better results. These aspects were not explored in depth due to time constraints but represent promising avenues for future work. For instance, employing ADAM instead of SGD as the optimizer for the CutPaste method could improve convergence and potentially lead to better anomaly detection performance.

Furthermore, extending the CutPaste approach to more than two classes, such as implementing a three-way classification scheme as described in the original paper, could enhance the method's ability to distinguish different types of anomalies. Likewise, testing a broader range of augmentations or different augmentation combinations could improve generalization and robustness.

Lastly, testing alternative backbone models, such as EfficientNet, could provide further improvements over ResNet. These models might offer better feature extraction capabilities and could enhance the overall accuracy and robustness of the anomaly detection system.

Disclaimer

For the purpose of enhancing linguistic clarity, coherence, grammar and overall structure, This thesis have been reviewed and refined using ChatGPT, an artificial intelligence language model developed by OpenAI. While ChatGPT can generate and improve human-like text, it doesn't inherently validate the accuracy or factual content of the thesis.

Appendices

APPENDIX A

First Appendix Chapter

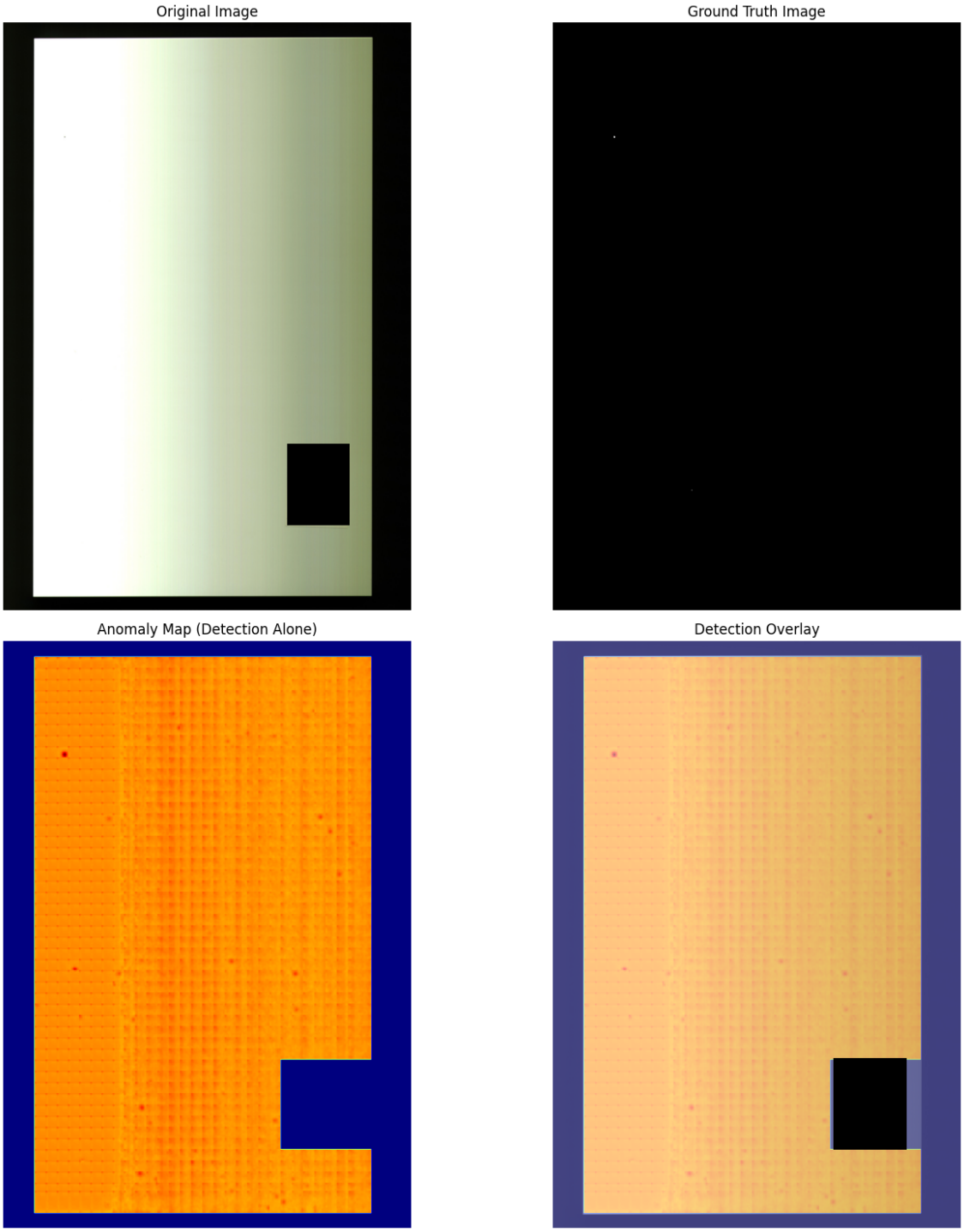


Figure A.1: Sample image from the White category (ID: 231117-332) for experiment 3 in Patchcore

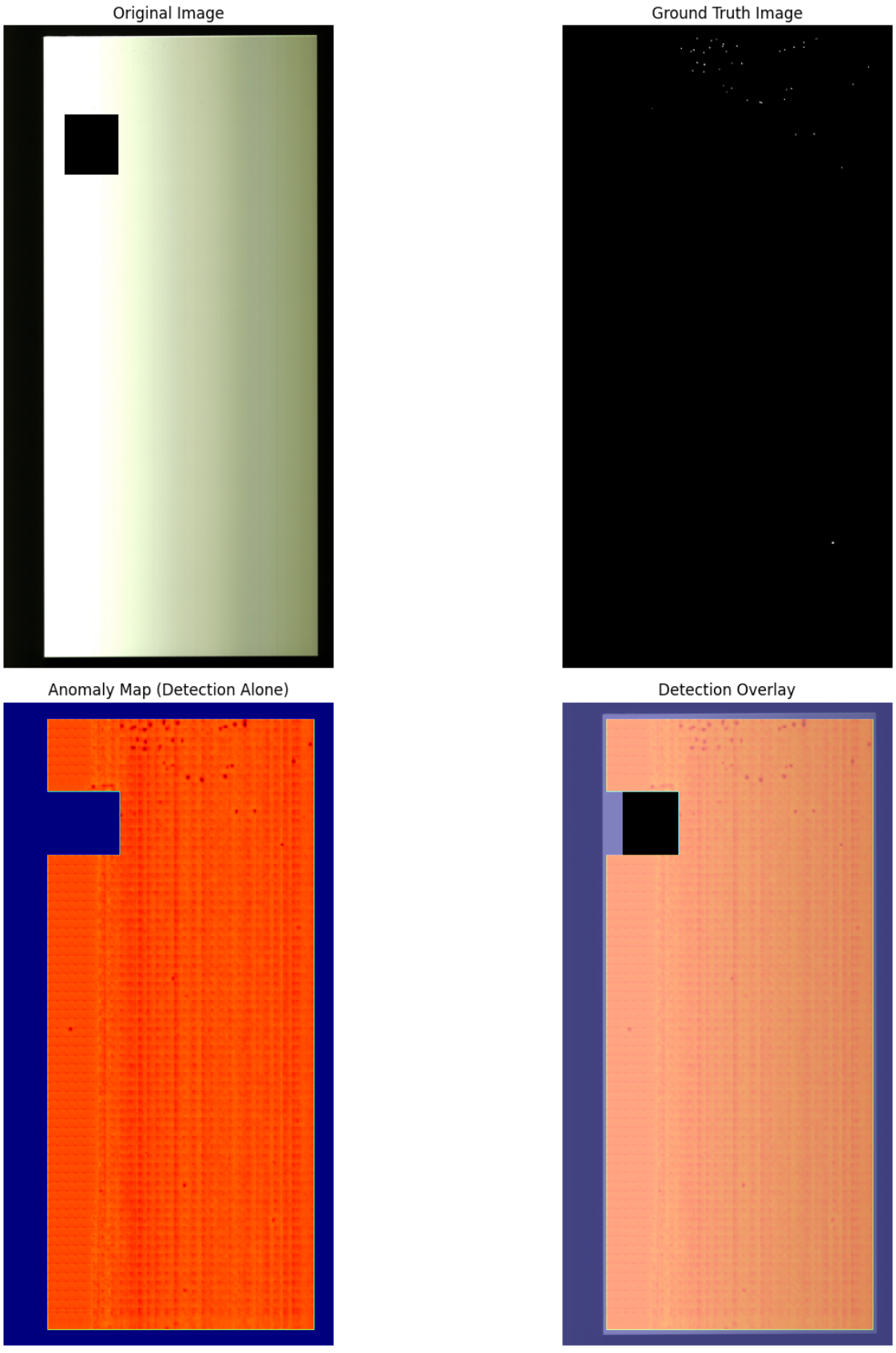


Figure A.2: Sample image from the White category (ID: 231117-316) for experiment 3 in Patchcore

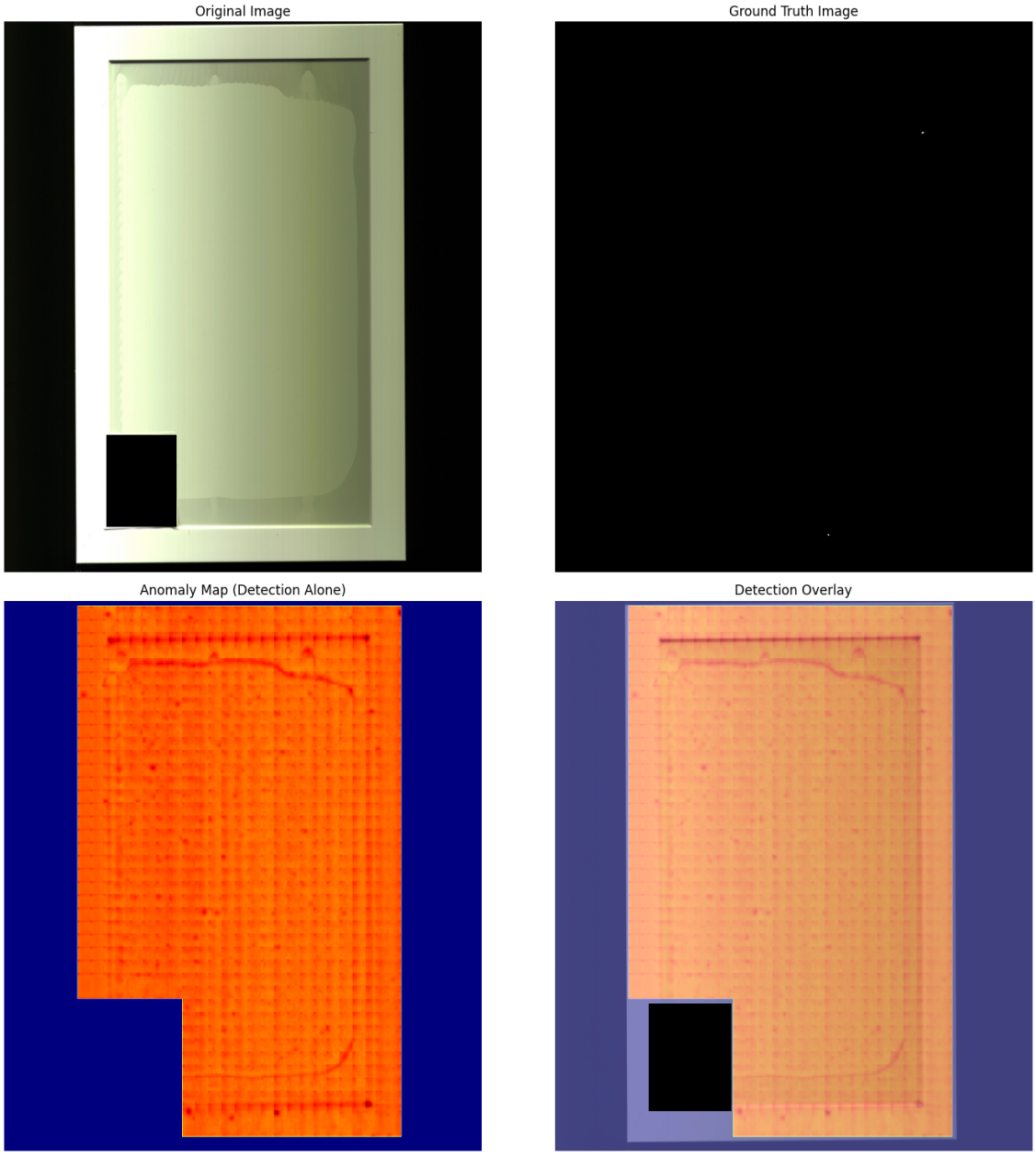


Figure A.3: Sample image from the White with Edges category (ID: 231117-664) for experiment 2 in Patchcore

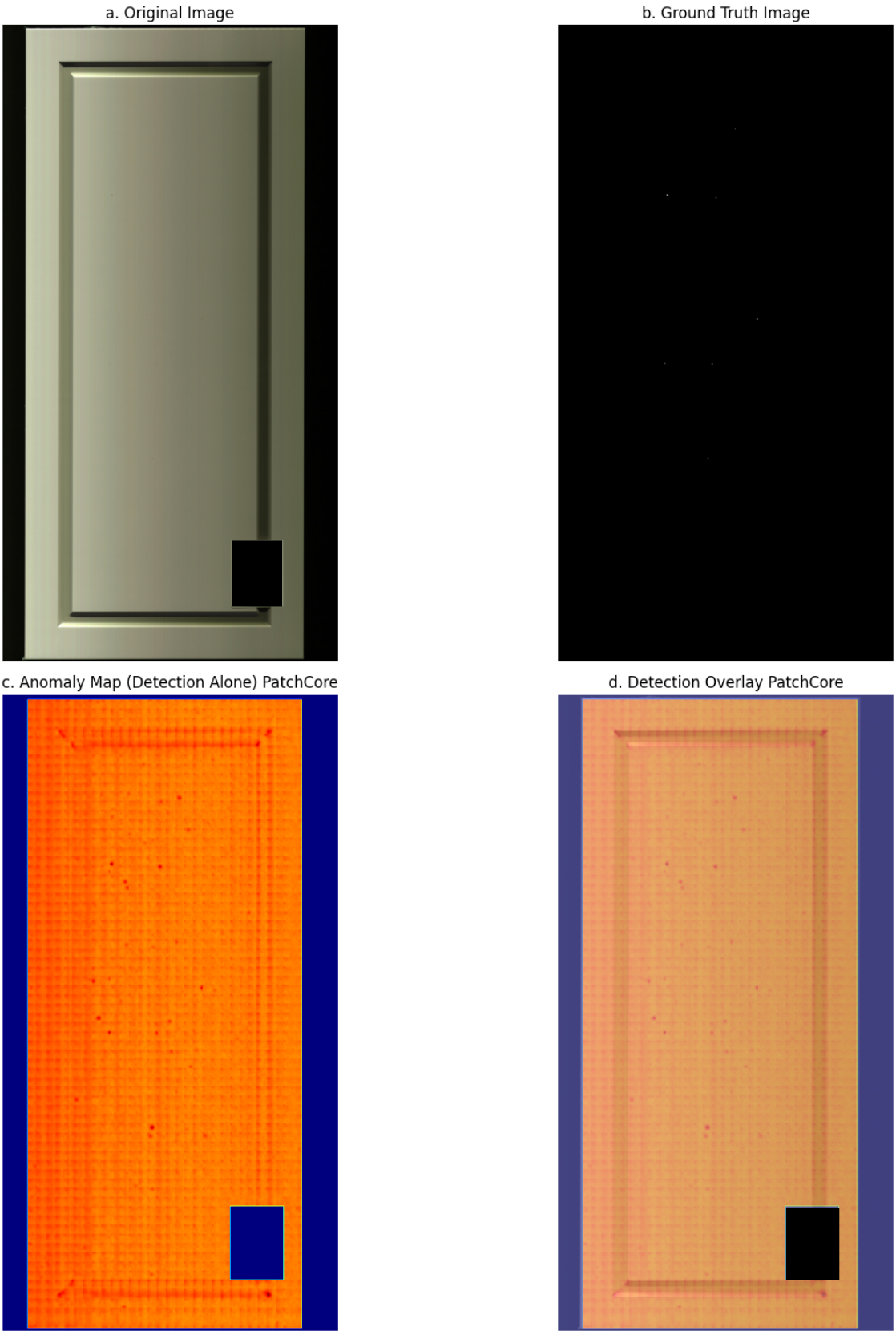


Figure A.4: Sample image from the White with Edges category (ID: 231117-230) for experiment 2 in Patchcore

Bibliography

- Chen Yen-Chi. *A Tutorial on Kernel Density Estimation and Recent Advances*. 2017. arXiv: 1704.03924 [stat.ME]. URL: https://arxiv.org/abs/1704.03924 (cit. on p. 4).
- Cui Yajie, Liu Zhaoxiang, and Lian Shiguo. "A Survey on Unsupervised Anomaly Detection Algorithms for Industrial Images." In: *IEEE Access* 11 (2023), pp. 55297–55315. DOI: 10.1109/ACCESS.2023.3282993 (cit. on p. 20).
- Dasgupta Sanjoy and Gupta Anupam. "An elementary proof of a theorem of Johnson and Lindenstrauss." In: *Random Structures & Algorithms* 22 (2003). URL: https://api.semanticscholar.org/CorpusID:10327785 (cit. on p. 33).
- Defard Thomas, Setkov Aleksandr, Loesch Angelique, and Audigier Romaric. *PaDiM: a Patch Distribution Modeling Framework for Anomaly Detection and Localization*. 2020. arXiv: 2011.08785 [cs.CV]. URL: https://arxiv.org/abs/2011.08785 (cit. on p. 21).
- Deng Jia, Dong Wei, Socher Richard, Li Li-Jia, Li Kai, and Fei-Fei Li. "ImageNet: A large-scale hierarchical image database." In: *2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 248–255. DOI: 10.1109/CVPR.2009.5206848 (cit. on p. 32).
- Fawcett Tom. "An introduction to ROC analysis." In: *Pattern Recognition Letters* 27.8 (2006), pp. 861–874 (cit. on pp. 49, 51).
- Fix Evelyn and Hodges Joseph L. "Discriminatory Analysis - Nonparametric Discrimination: Consistency Properties." In: (1951). URL: https://apps.dtic.mil/sti/citations/tr/ADA800276 (cit. on p. 3).
- GeeksforGeeks. *Artificial Neural Networks and its Applications* -. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/. 2024 (cit. on pp. 7, 8).
- Goodfellow I., Bengio Y., and Courville A. *Deep Learning*. MIT Press, 2016. URL: https://www.deeplearningbook.org/ (cit. on pp. 6, 13, 14).
- He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. "Deep Residual Learning for Image Recognition." In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90 (cit. on p. 17).

- Hornik K., Stinchcombe M., and White H. "Multilayer Feedforward Networks are Universal Approximators." In: *Neural Networks* 2.5 (May 1989), pp. 359–366. DOI: 10.1016/0893-6080(89)90020-8. URL: https://doi.org/10.1016/0893-6080(89)90020-8 (cit. on p. 7).
- Ioffe Sergey and Szegedy Christian. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." In: *arXiv preprint arXiv:1502.03167* (2015) (cit. on p. 12).
- Kingma Diederik P and Ba Jimmy. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on pp. 9, 10).
- LeCun Y., Bottou L., Bengio Y., and Haffner P. "Gradient-Based Learning Applied to Document Recognition." In: *Proceedings of the IEEE* 86.11 (Nov. 1998), pp. 2278–2324. DOI: 10.1109/5.726791. URL: https://doi.org/10.1109/5.726791 (cit. on p. 13).
- Li Chun-Liang, Sohn Kihyuk, Yoon Jinsung, and Pfister Tomas. "CutPaste: Self-Supervised Learning for Anomaly Detection and Localization." In: *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2021, pp. 9659–9669. DOI: 10.1109/CVPR46437.2021.00954 (cit. on pp. 1, 36, 43, 91).
- Loshchilov Ilya and Hutter Frank. "SGDR: Stochastic Gradient Descent with Restarts." In: *CoRR* abs/1608.03983 (2016). arXiv: 1608.03983. URL: http://arxiv.org/abs/1608.03983 (cit. on p. 11).
- Maaten Laurens van der and Hinton Geoffrey. "Visualizing Data using t-SNE." In: *Journal of Machine Learning Research* 9.86 (2008), pp. 2579–2605. URL: http://jmlr.org/papers/v9/vandermaaten08a.html (cit. on p. 51).
- MathWorks. *What Is a Convolutional Neural Network?* https://www.mathworks.com/discovery/convolutional-neural-network.html (cit. on p. 13).
- Paszke Adam, Gross Sam, Massa Francisco, Lerer Adam, Bradbury James, Chanan Gregory, Killeen Trevor, Lin Zeming, Gimelshein Natalia, Antiga Luca, et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library." In: *Advances in Neural Information Processing Systems* 32 (2019), pp. 8024–8035 (cit. on p. 19).
- Powers David Martin. "Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation." In: *Journal of Machine Learning Technologies* 2.1 (2011), pp. 37–63 (cit. on p. 49).
- Ramzan Farheen, Khan Muhammad Usman, Rehmat Asim, Iqbal Sajid, Saba Tanzila, Rehman Amjad, and Mehmood Zahid. "A Deep Learning Approach for Automated Diagnosis and Multi-Class Classification of Alzheimer's Disease Stages

Using Resting-State fMRI and Residual Neural Networks." In: *Journal of Medical Systems* 44 (Dec. 2019). DOI: 10.1007/s10916-019-1475-2 (cit. on p. 18).

- Rippel Oliver, Mertens Patrick, and Merhof Dorit. *Modeling the Distribution of Normal Data in Pre-Trained Deep Features for Anomaly Detection*. 2020. arXiv: 2005.14140 [cs.CV]. URL: https://arxiv.org/abs/2005.14140 (cit. on pp. 5, 37).
- Roth Karsten, Pemula Latha, Zepeda Joaquin, Schölkopf Bernhard, Brox Thomas, and Gehler Peter. "Towards Total Recall in Industrial Anomaly Detection." In: *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 14298–14308. DOI: 10.1109/CVPR52688.2022.01392 (cit. on pp. 1, 31, 91).
- Rumelhart D. E., Hinton G. E., and Williams R. J. "Learning Representations by Back-Propagating Errors." In: *Nature* 323.6088 (Oct. 1986), pp. 533–536. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0 (cit. on p. 8).
- Selvaraju Ramprasaath R., Cogswell Michael, Das Abhishek, Vedantam Ramakrishna, Parikh Devi, and Batra Dhruv. "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization." In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 618–626. DOI: 10.1109/ICCV.2017.74 (cit. on p. 37).
- Sener Ozan and Savarese Silvio. *Active Learning for Convolutional Neural Networks: A Core-Set Approach*. 2018. arXiv: 1708.00489 [stat.ML]. URL: https://arxiv.org/abs/1708.00489 (cit. on p. 33).
- Sinha Samarth, Zhang Han, Goyal Anirudh, Bengio Yoshua, Larochelle Hugo, and Odena Augustus. *Small-GAN: Speeding Up GAN Training Using Core-sets*. 2019. arXiv: 1910.13540 [stat.ML]. URL: https://arxiv.org/abs/1910.13540 (cit. on p. 33).
- Tibshirani Robert. "Regression shrinkage and selection via the lasso." In: *Journal of the Royal Statistical Society: Series B (Methodological)* 58.1 (1996), pp. 267–288. DOI: 10.1111/j.2517-6161.1996.tb02080.x (cit. on p. 11).
- Vorias R. *ind_knn_ad*. https://github.com/rvorias/ind_knn_ad/tree/master. Accessed: 2025-01-16. 2021 (cit. on p. 40).
- Yolyan Lilit. *CutPaste*. https://github.com/LilitYolyan/CutPaste. Accessed: 2025-01-16. 2021 (cit. on p. 42).
- Yoon Jinsung, Sohn Kihyuk, Li Chun-Liang, Arik Sercan O., and Pfister Tomas. *SPADE: Semi-supervised Anomaly Detection under Distribution Mismatch*. 2022. arXiv: 2212.00173 [cs.LG]. URL: https://arxiv.org/abs/2212.00173 (cit. on p. 20).

- Zeiler M. D. and Fergus R. “Visualizing and Understanding Convolutional Networks.” In: *European Conference on Computer Vision*. Springer, 2014, pp. 818–833. DOI: 10.1007/978-3-319-10590-1_53. URL: https://doi.org/10.1007/978-3-319-10590-1_53 (cit. on p. 14).

Acronyms

ADAM Adaptive Moment Estimation. 9, 10, 11, 73

ANN Artificial Neural Network. 6, 8, 13, 14

AUC Area under the curve. 53, 62

AUC-ROC Area Under the Receiver Operating Characteristic Curve. 49, 51, 52, 54, 60, 61, 71

CNN Convolutional Neural Network. 13, 15, 32

GDE Gaussian Density Estimation. 5, 37, 45

GPU Graphics processing Unit. 19, 29, 73

Grad-CAM Gradient-weighted Class Activation Mapping. 37, 45, 61, 62, 63, 68, 71

KDE Kernel Density Estimation. 4, 5

KNN K-Nearest Neighbor. 3

ML Machine Learning. 3, 6, 9, 16

OOM Out-Of-Memory. 41, 42

PaDiM Patch Distribution Modeling. 21

PDF probability density function. 4, 5

ReLU Rectified Linear Unit. 7, 14, 15

ResNet Residual Networks. 17, 40, 41, 73

ROC Receiver Operating Characteristic. 51, 53, 54, 61, 62, 67

SGD Stochastic Gradient Descent. 9, 10, 44, 60, 73

SPADE Semi-supervised Pseudolabeler Anomaly Detection with Ensembling. 20, 21

SSL Self-Supervised Learning. 6, 21

t-SNE t-Distributed Stochastic Neighbor Embedding. 51, 62

List of Figures

List of Tables

Eidesstattliche Versicherung

(Affidavit)

Neman, Shakib

Name, Vorname
(surname, first name)

236762

Matrikelnummer
(student ID number)

☐ Bachelorarbeit
(Bachelor's thesis)

■ Masterarbeit
(Master's thesis)

Titel
(Title)

Anomaly Detection on Diverse Kitchen Surfaces: A Case Study Using Industry Data

Ich versichere hiermit an Eides statt, dass ich die vorliegende Abschlussarbeit mit dem oben genannten Titel selbstständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

I declare in lieu of oath that I have completed the present thesis with the above-mentioned title independently and without any unauthorized assistance. I have not used any other sources or aids than the ones listed and have documented quotations and paraphrases as such. The thesis in its current or similar version has not been submitted to an auditing institution before.

Dortmund, 05/02/2025

Ort, Datum
(place, date)

Unterschrift
(signature)

Belehrung:
Wer vorsätzlich gegen eine die Täuschung über Prüfungsleistungen betreffende Regelung einer Hochschulprüfungsordnung verstößt, handelt ordnungswidrig. Die Ordnungswidrigkeit kann mit einer Geldbuße von bis zu 50.000,00 € geahndet werden. Zuständige Verwaltungsbehörde für die Verfolgung und Ahndung von Ordnungswidrigkeiten ist der Kanzler/die Kanzlerin der Technischen Universität Dortmund. Im Falle eines mehrfachen oder sonstigen schwerwiegenden Täuschungsversuches kann der Prüfling zudem exmatrikuliert werden. (§ 63 Abs. 5 Hochschulgesetz - HG -).

Die Abgabe einer falschen Versicherung an Eides statt wird mit Freiheitsstrafe bis zu 3 Jahren oder mit Geldstrafe bestraft.

Die Technische Universität Dortmund wird ggf. elektronische Vergleichswerkzeuge (wie z.B. die Software „turnitin") zur Überprüfung von Ordnungswidrigkeiten in Prüfungsverfahren nutzen.

Die oben stehende Belehrung habe ich zur Kenntnis genommen:

Official notification:
Any person who intentionally breaches any regulation of university examination regulations relating to deception in examination performance is acting improperly. This offense can be punished with a fine of up to EUR 50,000.00. The competent administrative authority for the pursuit and prosecution of offenses of this type is the Chancellor of TU Dortmund University. In the case of multiple or other serious attempts at deception, the examinee can also be unenrolled, Section 63 (5) North Rhine-Westphalia Higher Education Act (*Hochschulgesetz, HG*).

The submission of a false affidavit will be punished with a prison sentence of up to three years or a fine.

As may be necessary, TU Dortmund University will make use of electronic plagiarism-prevention tools (e.g. the "turnitin" service) in order to monitor violations during the examination procedures.

I have taken note of the above official notification:*

Dortmund, 05/02/2025

Ort, Datum
(place, date)

Unterschrift
(signature)

***Please be aware that solely the German version of the affidavit ("Eidesstattliche Versicherung") for the Bachelor's/ Master's thesis is the official and legally binding version.**