University of Padova

Department of Information Engineering

Master Thesis in Computer Engineering

Optimized Visual Anomaly detection models for tiny machine learning

Supervisor
Prof. Gian Antonio Susto
University of Padova

Co-supervisor
Dr. Davide Dalle Pezze
PhD. Manuel Barusco
PhD. Francesco Borsatti
University of Padova

Master Candidate
Youssef Ben Khalifa

Academic Year
2024-2025

Learn from yesterday, live for today, hope for tomorrow. The important thing is not to stop questioning.
Albert Einstein

Abstract

This thesis explores the field of visual anomaly detection within the context of Tiny Machine Learning (TinyML), focusing on the optimization of existing models for deployment on resource-constrained edge devices. We begin by benchmarking several state-of-the-art unsupervised VAD models,such as PaDiM and PatchCore,on popular datasets like MVTec, ViSA, and Real-IAD. These benchmarks provide a performance baseline for comparison. We then explore how these models behave when trained on noisy or contaminated data. Finally, we apply specific optimizations to improve the models' performance on edge devices. These include simplifying distance calculations for PaDiM and applying quantization techniques to reduce memory usage in PatchCore. Our results show that it is possible to significantly improve the efficiency of these models without heavily impacting the performance of the original models, making them more suitable for real-world deployment on low-resource devices.

Contents

Abstract v

List of figures ix

List of tables xi

Listing of acronyms xiii

1 Introduction 1

2 Related Work 3
2.1 VAD Models 4
2.2 PaSTE 5
2.3 Product Quantization 5
2.4 Benchmark Datasets 6

3 Benchmark Baseline for IAD Datasets and Models 7
3.1 MVTec Dataset 7
3.2 Real-IAD Dataset 8
3.3 ViSA Dataset 11
3.4 Feature Extraction Backbones 11
3.5 Benchmark Setup 13
3.6 Performance Metrics 14
3.7 Benchmark Results 15
3.8 Observations 15

4 VAD Model in a Noisy Scenario 21
4.1 Training set contamination 21
4.2 Contaminated Benchmark 23
4.3 Contaminated Benchmark Results 23

5 PaDiM Optimizations 25
5.1 PaDiM Model 25
5.2 Mahalanobis Distance With Diagonal Covariance 27

6 PatchCore Optimizations 29

6.1 PatchCore Model . . . 29
6.2 Coreset Greedy Selection . . . 30
6.3 Memory-bank Quantization . . . 31
6.3.1 Product Quantization . . . 32
6.3.2 Partially Quantized Nearest Neighbor Search . . . 32
6.3.3 Hyperparameter Tuning: K and c in PQNN-Based Anomaly Detection . . . 34

7 Experiments 37
7.1 Experimental Setup . . . 38
7.2 MoViAD Library . . . 38

8 Experiment Results 39
8.1 Evaluation metrics . . . 39
8.2 PaDiM Experiments . . . 40
8.3 PatchCore Experiments . . . 41
8.3.1 Quantized PatchCore . . . 41
8.3.2 PQNN Search Performance . . . 42

9 Conclusions & Future Work 45
9.1 Experiment Results Observations . . . 45
9.2 Final Considerations and Conclusions . . . 45
9.3 Future Work . . . 46

10 Appendix 49

References 59

Acknowledgments 63

Listing of figures

3.1 MVTec Dataset Entries Examples. Source: [1] 9
3.2 RealIad Dataset Entries Examples. Source: [2] 10
3.3 VisA Dataset Entries Examples. Source: [3] 12
3.4 Performance Comparison of PaDiM across Datasets and Backbones 16
3.5 Performance Comparison of PatchCore across Datasets and Backbones . . . 17
3.6 Performance Comparison of CFA across Datasets and Backbones 18
3.7 Performance Comparison of STFPM across Datasets and Backbones 19

5.1 PaDiM model training pipeline overview 26

6.1 PatchCore Model overview.Source: [4] 30
6.2 Product Quantization sub vector. Source: [5] 32
6.3 Product Quantization. Source: [5] 33

8.2 Original PaDiM vs Diagonalized PaDiM Performance Comparison (MVTec, MobileNet-V2) 41
8.3 Memory Bank memory usage during inference in function of K (neighborhood size) (MVTec, MobileNet-V2) 42
8.4 PatchCore vs PatchCore Quantized Performance Comparison(MVTec, MobileNet-V2, K=300, c=1000) 43
8.5 image-Level F1 Score with multiple values of K and c 43
8.6 Pixel-Level F1 Score with multiple values of K and c 44
8.7 Image-Level AUROC score with multiple values of K and c 44
8.8 Pixel-Level Score AUROC with multiple values of K and c 44

Listing of tables

3.1 Backbone AD Layers for benchmark 13

8.1 Table showing the time training PaDiM time difference 40

10.1 PaDiM metrics for MVTec by category 49
10.2 PaDiM metrics for ViSA by category 50
10.3 PaDiM metrics for Real-IAD by category 51
10.4 PatchCore metrics for MVTec by category 52
10.5 PatchCore metrics for ViSA by category 52
10.6 PatchCore metrics for Real-IAD by category 53
10.7 CFA metrics for MVTec by category 54
10.8 CFA metrics for ViSA by category 54
10.9 CFA metrics for Real-IAD by category 55
10.10 STFPM metrics for MVTec by category 56
10.11 STFPM metrics for ViSA by category 56
10.12 STFPM metrics for Real-IAD by category 57

Listing of acronyms

MoViAD .... Mobile Visual Anomaly Detection

PQ .... Product Quantization

TinyML .... Tiny Machine Learning

PQNN .... Partially Quantized Nearest Neighbor

1 Introduction

Anomaly detection has become a central issue in the industry over the last years. In fact, the necessity of identifying the presence of an imperfection during production has become essential in order to guarantee an efficient quality control, along with classifying the type of anomaly detected. IAD, which supports image anomaly detection, has been identified as one of the most effective methods of doing so, since it provides a quick and reliable way for analyzing the final product anomalies directly on the production line, at least on paper. Multiple variations and models for IAD have been developed over time, each with its own issues and challenges, but almost all of these share one common issue: labeled data. As we all know, in the field of Machine Learning and, more importantly, Deep Learning, the need for bigger datasets has become a challenging issue more and more, since the performance of both machine learning and deep learning models is heavily influenced by the size and accuracy of the datasets used. On top of that, the resources needed to create accurate and large enough datasets are substantial. One way to tackle this is to adopt either self-supervised or unsupervised methods.

Unsupervised learning has emerged as a compelling approach in this context, particularly for visual anomaly detection (VAD). In contrast to supervised methods, unsupervised VAD can be trained solely on normal data, enabling the detection of deviations without requiring costly manual labeling of anomalies. This paradigm shift is crucial in industrial scenarios, where anomalous samples may be rare, diverse, and difficult to annotate. At the same time, such models must be efficient and fast enough to support real-time decision-making processes directly on the factory floor, which often involves using constrained hardware like micro controllers or

embedded processors.

In recent years, the growing field of Tiny Machine Learning (TinyML) has aimed to bring machine learning capabilities to such resource-limited edge devices. While much progress has been made in optimizing traditional classification or regression models for the edge, adapting complex computer vision tasks such as VAD still poses several technical challenges. Most state-of-the-art VAD models rely on deep convolutional networks that are too heavy for real-time inference on tiny devices. Hence, there is a pressing need to explore new optimization strategies, such as quantization, that can reduce memory usage and inference time while maintaining robust anomaly detection performance.

This thesis contributes to this area by benchmarking a set of well-known unsupervised VAD models across multiple datasets and lightweight backbones, and by proposing and evaluating model-specific optimizations tailored for the TinyML setting. In particular, we focus on improving the efficiency of PaDiM and PatchCore, two popular methods with promising detection performance but high computational demands. Additionally, we explore novel training-time and inference-time optimizations such as diagonal covariance estimation, product quantization, and batched coreset selection to significantly reduce the models' memory and compute footprint.

Our contributions can be summarized as follows:

- We compare state of the art VAD models, providing a benchmark baseline for VAD methods by evaluating using the MVTec, ViSA and Real-IAD datasets;
- We test the same VAD models in the contaminated dataset scenario, focusing ourselves on the MvTec dataset;
- We design and implement a set of targeted optimizations for the PaDiM and PatchCore models to adapt them more effectively for TinyML deployment;
- We compare the optimizations applied to the originally proposed models;

Through this work, we aim to bring VAD one step closer to real-world deployment in constrained environments, where efficiency, reliability, and autonomy are paramount.

2
Related Work

Over the last few years several approaches have been made to tackle the Anomaly detection issue, each method and approach with its own benefits and downsides. The visual approach to anomaly detection is one of the most commonly studied in literature, in which a good number of VAD models were proposed.
These methods can be categorized into two main types: feature based and reconstruction based methods . The first type of models exploit the usage of a deep learning network for extracting meaningful features from the given image, and based on the embeddings extracted these models are trained to infer whether or not the sample is anomalous or not. Reconstruction based methods are instead generative type networks, that are trained with a dataset of normal images, and then use the inferred image (for a given category) to compare with a given sample (normal or anomalous that can be) and based on the comparison determines whether or not input sample is normal.
Most, if not all, of the state of the art VAD models are trained through an unsupervised approach (as opposed to the classical supervised training strategy) so as to undercut the dataset labeling cost, which for the dataset dimensions we need can be very expensive. Thus, both feature-based and reconstruction based models are trained using a batches of normal images (which are easy to obtain), from those batches learn the features that characterize a normal sample (feature-based), or build a probability distribution that models the distribution of normal images for that dataset

2.1 VAD Models

Various unsupervised strategies have been devised that do not require the usage of labeled data, but instead a simple batch of normal images. In this thesis we care particular attention to the PaDiM [6], PatchCore [4], CFA [7] and STFPM [8] models.
All of these models are feature extraction based models, which means that they rely on the usage of a deep neural network (most commonly a CNN) to extract meaningful features from the input image, process them into a set of embeddings and then work with these in order to perform the classification and compute an anomaly score.
PaDiM (Patch Distribution Modeling) [6] is a deep feature-based method for unsupervised visual anomaly detection and localization that leverages the representational power of pretrained convolutional neural networks (CNNs), such as ResNet[9] (and also other image recognition models as we will see later on), to extract multi-scale feature embeddings from normal images. Unlike global feature approaches, PaDiM models the distribution of patch-wise features by fitting a multivariate Gaussian distribution to each spatial location in the feature map across the training set. During inference, anomaly scores are computed using the Mahalanobis distance between the test image's features and the learned Gaussian parameters. This patch-level probabilistic modeling enables accurate detection and precise localization of both structural and texture-based anomalies. PaDiM has demonstrated competitive performance on industrial benchmarks such as MVTec AD [1][10], establishing itself as a strong baseline in the VAD literature. Refer to the original paper [6] for more details about the experiments setup and results.
PatchCore, proposed [7], advances patch-based anomaly detection by combining a memory-efficient core-set sampling mechanism with deep feature embeddings from pretrained CNN backbones. The method extracts high-dimensional patch-wise features from normal images and retains a compact subset using greedy coreset selection to minimize redundancy. Anomaly detection is performed by computing the nearest-neighbor distance between test patches and the memory set in the feature space. This strategy balances performance and computational cost, enabling scalable anomaly detection with limited memory. Along with PaDiM, PatchCore achieves state-of-the-art results on MVTec AD and is particularly notable for its simplicity and high accuracy without requiring fine-tuning.
Another model CFA (Coupled-hypersphere-based Feature Adaptation), introduced in [11], is another feature-based method that uses applies a transfer-learning approach on a given training dataset with the goal of reducing the bias genereated from the pre-trained CNN networks used for feature extraction [11].

STFPM (Student-Teacher Feature Pyramid Matching) is a knowledge distillation-based approach to visual anomaly detection proposed in [8]. It employs a teacher network (typically a fixed pretrained CNN) to generate multi-scale feature representations from normal images, which a student network is trained to mimic. During inference, discrepancies between teacher and student features,especially in a pyramid-like, multi-scale structure,serve as indicators of anomalies. This framework avoids complex probabilistic modeling and leverages the assumption that the student can only approximate normal patterns. STFPM is efficient and lightweight, making it suitable for real-time and resource-constrained environments, while maintaining competitive performance on anomaly localization tasks.

2.2 PaSTE

This thesis also takes inspiration from the work done in the paper [12]. The paper address the pressing challenge of enabling visual anomaly detection (VAD) on edge devices with constrained computational and memory resources. Their work introduces a comprehensive benchmark of lightweight neural networks for anomaly detection using the MVTec dataset, and proposes a novel method named *Partially Shared Teacher-Student* (PaSTe), a resource-efficient variant of the well-known STFPM approach. By reusing initial layers between teacher and student models, PaSTe significantly reduces training memory (up to 76%) and inference complexity, while maintaining comparable or even improved detection performance. Their findings underscore the viability of deploying feature-based VAD models on edge platforms, reinforcing the importance of architectural optimizations in TinyML applications. This aligns with the goals of this thesis, which explores efficient deep learning strategies for reliable visual anomaly detection in constrained environments.

2.3 Product Quantization

Product Quantization (PQ), introduced by Jégou et al. [13], is a vector quantization technique widely used for efficient approximate nearest neighbor (ANN) search in high-dimensional spaces. PQ works by decomposing the original feature space into a Cartesian product of low-dimensional subspaces and quantizing each subspace separately. This reduces the memory footprint of feature vectors while enabling fast distance computations in compressed form.
Compared to scalar quantization or traditional vector quantization, PQ achieves a better tradeoff between compression rate and retrieval accuracy, making it particularly useful in large-scale

image retrieval and similarity search systems. The encoding is typically implemented via k-means clustering on each subspace, and distance computations are accelerated using precomputed lookup tables.
Subsequent enhancements to PQ have been proposed to improve accuracy and adaptability. Optimized Product Quantization (OPQ) [14] applies a rotation to the data before quantization to better align the subspaces with the data distribution. Additive Quantization (AQ) and Composite Quantization (CQ) [15], [16] further generalize the PQ framework by relaxing the independence assumption between subspaces to enhance approximation quality.
In TinyML and edge scenarios, where both memory and computation are constrained, PQ offers a viable strategy for compressing high-dimensional embeddings, such as those from neural networks, without significantly degrading performance.

2.4 Benchmark Datasets

To further evaluate the effectiveness of the above mentioned anomaly detection models, several benchmark datasets are widely used. **MVTec AD** is a standard dataset for industrial visual anomaly detection, containing high-resolution images across various object and texture categories, with both localized and global anomalies. **VisA**, a more recent benchmark, focuses on fine-grained anomalies in real-world grocery product images under diverse conditions, promoting research into scalable and robust VAD systems. **RealIAD** (Real-world Industrial Anomaly Detection), in contrast, presents a large-scale collection of real industrial defects with greater variability and noise, reflecting real deployment challenges. These datasets serve as critical tools for benchmarking performance and generalization in anomaly detection research. Most models we have explored during the research of this thesis were already tested and evaluated using the MVTec [1] [10] dataset for the performance benchmark.

3

Benchmark Baseline for IAD Datasets and Models

As mentioned in 3 various models and approaches that tackle the VAD issue already exist in literature, so before we can try any type of optimizations we first need a baseline benchmark of these models. The first contribution of this thesis is to take these models and extend their performance benchmarks to a multitude of datasets and backbones for the image level feature extraction.In the PaDiM [6], PatchCore [4], CFA [11] and STFPM[8] original papers the performance measures where given based on the MVTec dataset [1] [10]. Here we expand the datasets pool with the Real-IAD [17] and ViSA [3] datasets and also expand the range of backbones used for the respective Image feature extraction components. In particular we will make use of MobilenetV2 [18], ResNet [9], WideResnet [19], Phinet [20], McuNet [21]. This set of benchmarks will act as the baseline for the optimizations applied to the PaDiM and PatchCore models that we will explore later in this thesis.

3.1 MVTec Dataset

The MVTec dataset is probably among the most popular VAD dataset used in Anomaly detection research field. This is the same dataset used in the VAD original papers ([6], [12], [7], [8]) for the experiments. The dataset is composed of approximately 5,354 high-resolution im-

ages across 15 object categories, which is already split into training set and test set, with 3,629 defect-free images for the first and 1,725 images (both defect-free and defective) for the second. The datasets includes the following object categories:

- bottle
- cable
- capsule
- carpet
- grid
- hazelnut
- leather
- metal_nut
- pill
- screw
- tile
- toothbrush
- transistor
- wood
- zipper

Please refer to the original MVTec papers [1] [10] for more details on the dataset.

3.2 Real-IAD Dataset

The Real-IAD dataset is a collection of images representing a set of objects with anomalies, spanning through various object categories. The Real-IAD dataset aims at providing more real-world industrial-related multi-view images. The benchmarks on this dataset will evaluate the performance of the models in a more industrial scenario. It is composed of **151,050 images** of 30 diverse object categories, of which 99,721 normal and 51,329 anomalous. The Real-IAD dataset includes the following categories:

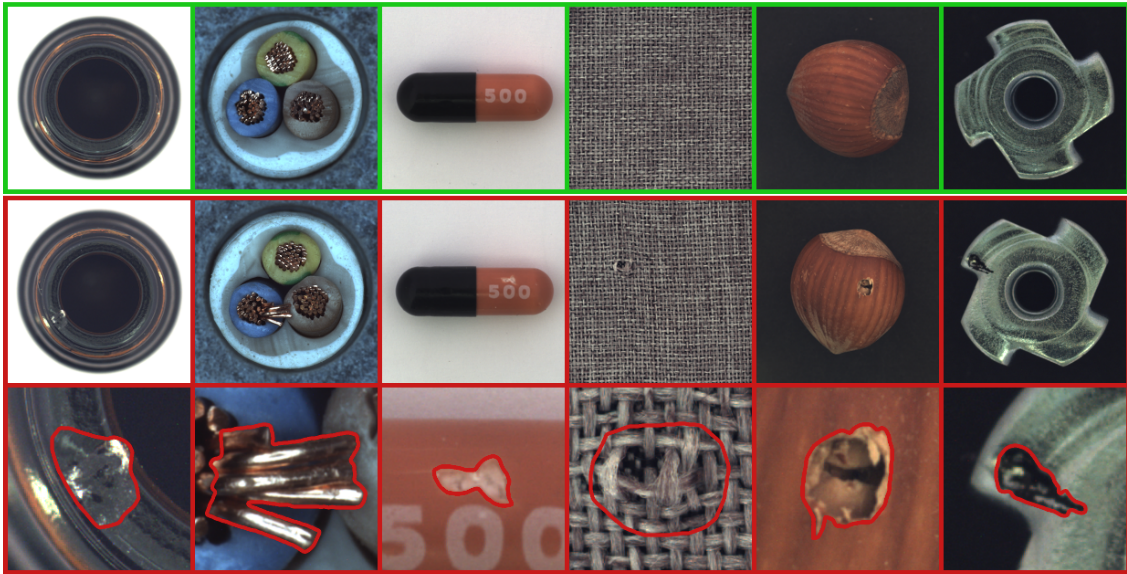


Figure 3.1: MVTec Dataset Entries Examples. Source: [1]

- audiojack
- bottle_cap
- button_battery
- end_cap
- eraser
- fire_hood
- mint
- mounts
- pcb
- phone_battery
- plastic_nut
- plastic_plug
- porcelain_doll
- regulator

- rolled_strip_base
- sim_card_set
- switch
- tape
- terminalblock
- toothbrush
- toy
- toy_brick
- transistor1
- u_block
- usb
- usb_adaptor
- vcpill
- wooden_beads
- woodstick
- zipper

Figure 3.2: Reallad Dataset Entries Examples. Source: [2]

This dataset is the largest we have in our benchmarks, and acts as a very good dataset for real world scenario performance or the VAD models and their optimizations.

3.3 ViSA Dataset

This is another IAD dataset comprehending a collection of images of objects with anomalies, composed of 10,821 high-res images across **12 object categories**, with 9,621 normal and 1,200 defective samples . For the full details about the contents of the dataset, refer to the original paper [3]. Again, refer to the dataset paper for more details about its content [3]. The images are split into the following categories:

- candle
- capsules
- cashew
- chewinggum
- fryum
- macaroni1
- macaroni2
- pcb1
- pcb2
- pcb3
- pcb4
- pipe_fryum

3.4 Feature Extraction Backbones

Each VAD model we will explore during this thesis are embedding similarity or feature-based [12] models, thus they rely on a feature extraction approach applied onto the images in order to determine whether or not these are anomalous or not. The feature extraction part of the model is entirely handled by a pre-trained deep learning network (most commonly a CNN type) that handles the feature and embedding extraction from the images. Along with testing different datasets, what we want to test is also how different backbones can function with the different

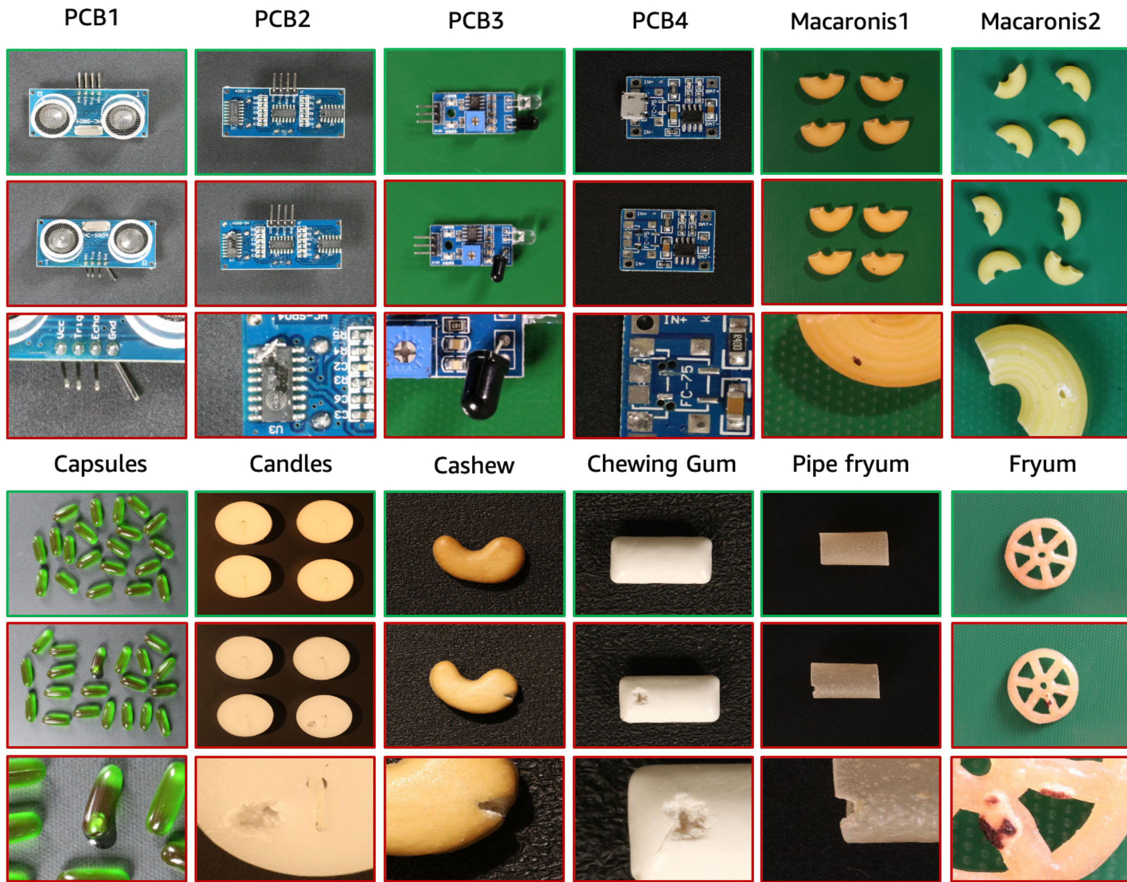


Figure 3.3: VisA Dataset Entries Examples. Source: [3]

models. Every model we presented use a feature extractor to, as the name says, extract patch features from every image, to then be converted to embeddings that can be later used by each model in its own way. The main idea is to try and exploit existing deep learning models for image recognition and perform Layer selection [12]). This process consists of basically separating the layers of the CNN used for image recognition and exploit only the output features from a given selection of those layers. This process mainly benefits not only performance, since we do not need to rely on the entire deep neural network to extract the features needed, and these are usually quite deep and complex CNN based architectures composed several million of parameters, but also allows us to select "lower" or "higher" level features, which can work worse or better, depending on the scenario for the task at hand.

The models used for the benchmarks are the following:

- MobileNet V2 [18];

- MicroNet [22];
- McuNet [21]
- PhiNet [20]
- ResNet [9]

The test were initially supposed to be also run using the WideResNet, unfortunately due to the dataset size the hardware was not able to run all the tests successfully. In addition to that, the few test test that were run, show a substantial increase or resource usage, hence concluding that the WideResnet was definitely not suited in a Tiny ML scenario.

3.5 Benchmark Setup

Here we go over hole list of benchmarks run. For every VAD model:

- PaDiM [6]
- PatchCore [7]
- CFA [11]
- STFPM [8]

we want to benchmark their performance for each category from the following datasets:

- MVTec Dataset [1][10]
- Real-IAD Dataset [2]
- ViSA Dataset [3]

All of these configurations will be tested on every mentioned backbone model with the following activation layers:

Backbone	MobileNet V2	McuNet	MicroNet	PhiNet	ResNet18
AD Layers	[1, 2, 3]	[3, 6, 9]	[2, 4, 5]	[2, 6, 7]	[1, 2, 3]

Table 3.1: Backbone AD Layers for benchmark

The samples from each dataset were resized from the their respective original resolution down to a (224, 224) resolution, no other pre-processing was applied onto the images.

3.6 Performance Metrics

For the benchmarks we will be evaluating the models based on the following metrics:

- **Image Level F1 Score**: The image-level F1 score measures the harmonic mean of precision and recall for image-wise anomaly detection. It is calculated by thresholding the predicted image anomaly scores to classify each image as normal or anomalous, then comparing these predictions to the ground truth image labels. A higher image-level F1 score reflects a better balance between correctly detected anomalies and false positives at the image level.

- **Image Level AUROC Score**: The image-level AUROC (Area Under the Receiver Operating Characteristic curve) score is a metric used to evaluate the performance of an anomaly detection model at the image level. It measures the model's ability to distinguish between normal and anomalous images by plotting the true positive rate against the false positive rate at various threshold settings. A higher AUROC score indicates better discrimination capability, with a value of 1.0 representing perfect separation and 0.5 indicating random guessing.

- **Pixel Level F1 Score**: The pixel-level F1 score quantifies the balance between precision and recall for pixel-wise anomaly segmentation. It is obtained by thresholding the predicted anomaly maps to generate binary segmentation masks and comparing them to the ground truth masks for each pixel. A higher pixel-level F1 score indicates more accurate and reliable segmentation of anomalous regions.

- **Pixel Level AUROC Score**: The pixel-level AUROC (Area Under the Receiver Operating Characteristic curve) score evaluates the model's ability to distinguish between normal and anomalous pixels across all test images. It is computed by comparing the predicted anomaly map values to the ground truth segmentation masks at the pixel level. A higher pixel-level AUROC indicates better pixel-wise discrimination, with 1.0 representing perfect separation and 0.5 indicating random performance.

the image level scores will help us determine whether or not the model correctly distinguishes between normal from abnormal samples, while the pixel level scores give us a measure of how accurately the model is able to identify the anomaly within the image.
AUROC (Area Under the Receiver Operating Characteristic curve) measures the model's ability to distinguish between classes across all possible thresholds. At the image level, it evaluates how well the model separates normal and anomalous images; at the pixel level, it assesses the separation of normal and anomalous pixels. AUROC is threshold-independent and summarizes performance over all possible decision boundaries.

F1 Score is the harmonic mean of precision and recall at a specific threshold. At the image level, it reflects the balance between correctly detected anomalous images and false positives after binarizing predictions. At the pixel level, it measures the accuracy of segmentation by comparing predicted and ground truth masks after thresholding. F1 is threshold-dependent and focuses on the chosen operating point.

3.7 Benchmark Results

Following up we will see the results of the benchmarks divided mainly by the various dataset. Due to the extremely large amount possible combinations between the dataset classes and the backbones available for the feature extractor, we will show the results only for the most representative category from each dataset, for more detailed results see the 10 section. The main goal is to firstly try and replicate the experimental result found in the respective models papers, using then the same hyperparameter, at least for the existing setups.

Note: *that the results may vary depending on the development setup as it is subject to different library version and in general different development and experimental environment given by different hardware used.*
The following plots show the performance metrics (F1 Image score, F1 Pixel score, Image AUROC score and Pixel AUROC score) for PaDiM [6], PatchCore [4], CFA [11] and STFPM [8] respectively.

3.8 Observations

The first thing to notice how the models perform on the three different datasets: on both the MVTec [1] [10] and VisA [3] we notice good performance across all models and backbones, while (almost) all models struggle in terms of both image level and pixel level classification, this may an indication of how VAD models are still far from been viable in real world applications. What is evident from plots 3.4, 3.5, 3.6 and 3.7 is that all 4 models show a low F1 Pixel-Level score performance on Real-IAD dataset when compared to the datasets and metrics . This difference in performance is most likely due to the type of objects that the Real-IAD dataset is composed of, thus highlighting a potential bias of the models towards certain categories of items to classify.
Another fact we can notice is how the different backbones perform: the MicroNet [22] back-

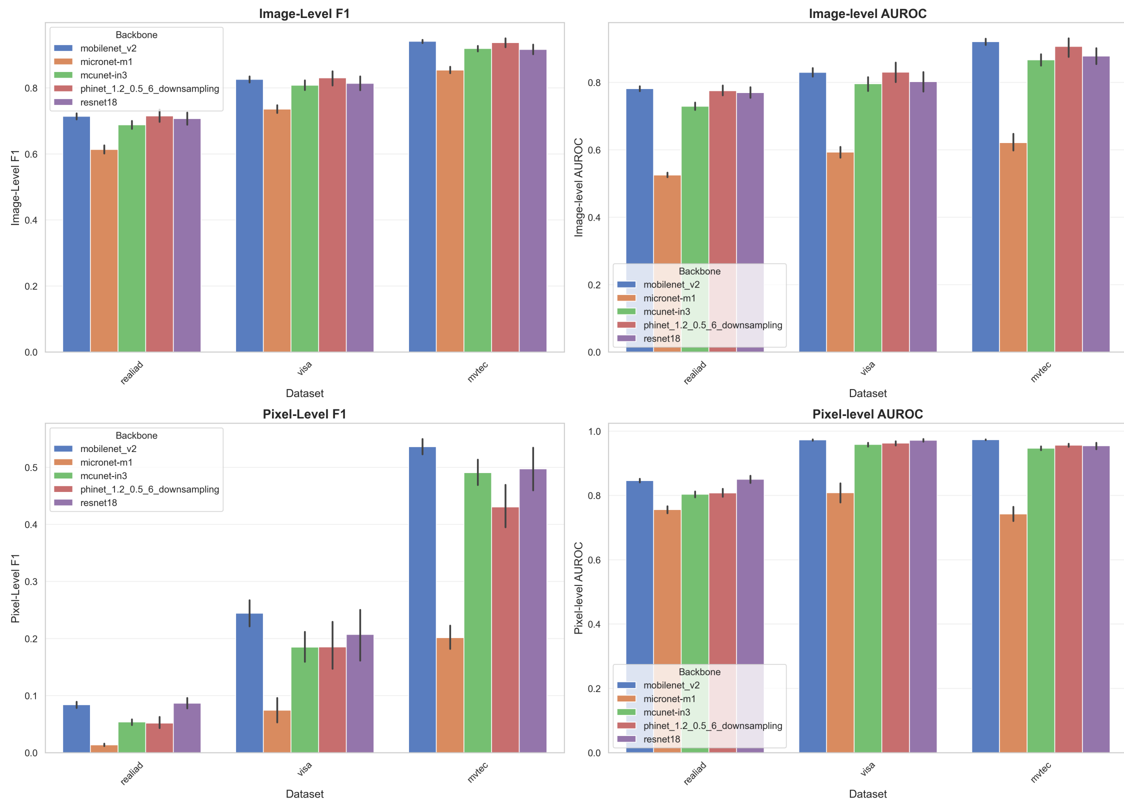


Figure 3.4: Performance Comparison of PaDiM across Datasets and Backbones

bone is the worst across the board, on both image and pixel level performance metrics. Different story is for the other backbones, where each backbone performs differently depending on the model its built upon. For example, the MobileNet [18] shows the best scores across all performance metrics in the PaDiM [6] model.

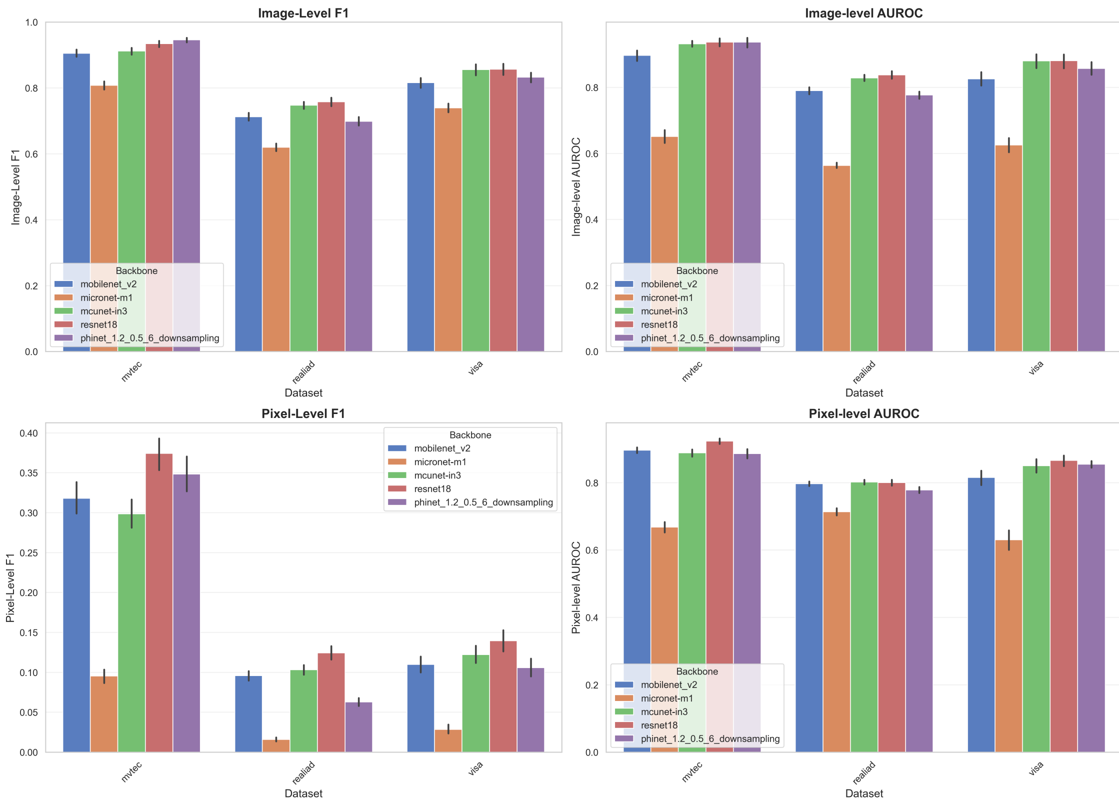


Figure 3.5: Performance Comparison of PatchCore across Datasets and Backbones

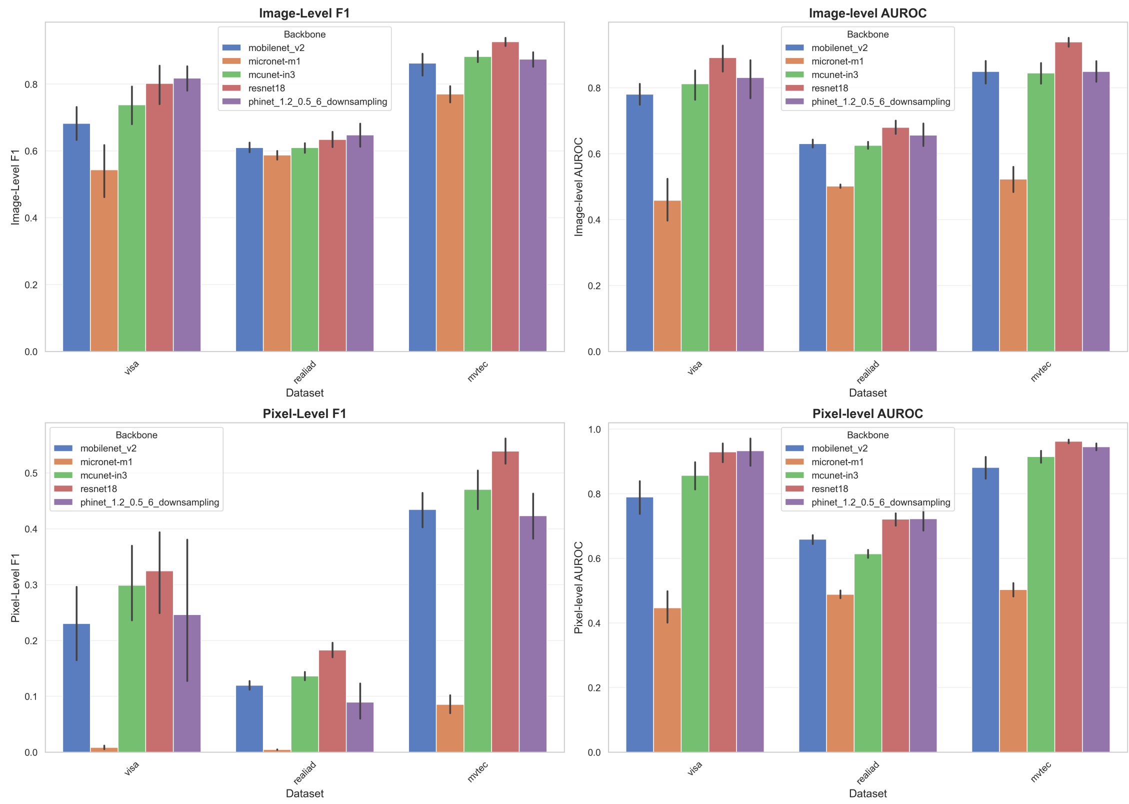


Figure 3.6: Performance Comparison of CFA across Datasets and Backbones

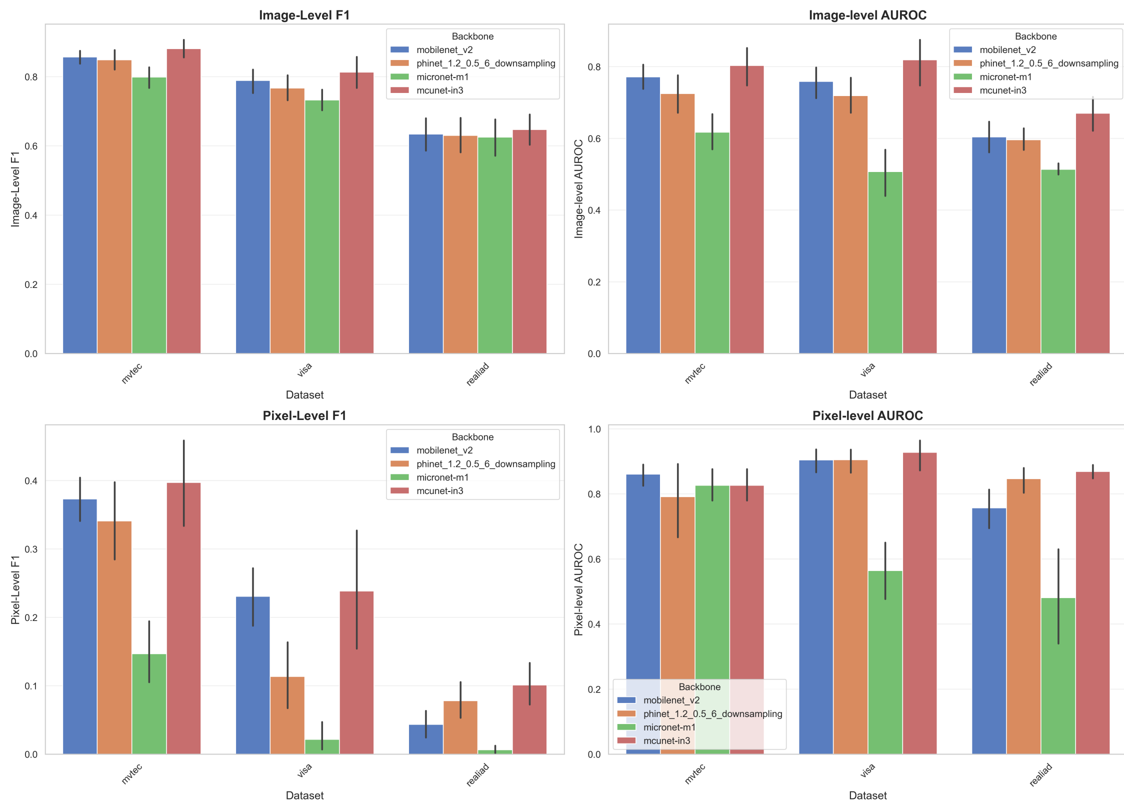


Figure 3.7: Performance Comparison of STFPM across Datasets and Backbones

4

VAD Model in a Noisy Scenario

Picture this scenario: during a normal day in the production line something unexpected happens, and some items destined to distribution either are damaged or defected in some way, thus not applicable for commercial distribution. These cases cannot be predicted: these can occur due to systematic errors that human intervention cannot prevent.

Now you are wondering: what does this have to do with what we are discussing? Well, what we are interested in is that we cannot know with absolute certainty that the samples coming out of a production line are considered as ”normal” at the classification level. Considering that most VAD models are trained in a unsupervised fashion, using datasets composed of normal image, and these datasets, in a real world scenario, are most probably created by simply placing a camera along the production line and make it take a picture every time an item passes, we may end up with **contaminated datasets**!

The goal of this section then is to briefly explain how we emulated the contaminated scenario in our datasets and explore state of the art VAD models behave in the contaminated scenario.

4.1 Training set contamination

The first step is to try and recreate the scenario we pictured in this chapter’s introduction. We then need a systematic and controlled way to introduce anomalies in the training datasets, with which VAD models will be trained.

Thankfully, there are datasets used in the VAD literature that include a test set with labeled entries, allowing us to properly distinguish between normal and anomalous images. The obvious course of action is to then take anomalous samples from the test set (of the same dataset)

To simulate contamination, anomalous samples are intentionally introduced into the training set until a specified contamination ratio, defined as the proportion of anomalies relative to the total number of training samples, is reached. This process is carried out while preserving the original training/test split ratio. To ensure consistency and realism, the anomalous samples used for contamination are drawn directly from the test set.

Here is how the dataset contamination process works. Starting from a given dataset split into training set $\mathcal{X}_{train}$ and test set $\mathcal{X}_{test}$ and a given contamination ration $p \in (0, 1] \subset \mathbb{R}$ we do the following:

1. using the contamination ratio p we compute the number of contaminated samples we need for our training set as:
$$\tilde{n} = p \cdot \mathcal{X}_{train} \tag{4.1}$$

2. we now sample $\tilde{n}$ anomalous entries from the test set $\mathcal{X}_{test}$ and build our set of anomalous samples, let us call this $\tilde{\mathcal{X}}$;

3. now we remove these samples from the initial test set $\mathcal{X}_{test}$ and "contaminate" the training set $\mathcal{X}_{train}$ with $\tilde{\mathcal{X}}$. Let us call the contaminated set $\tilde{\mathcal{X}}_{train}$ and the define $\hat{\mathcal{X}}_{test} := \mathcal{X}_{test} \setminus \tilde{\mathcal{X}}$. $\tilde{\mathcal{X}}_{train}$ will be our new "contaminated" training set and $\hat{\mathcal{X}}_{test}$ our new test set for our model.

4. Finally we compute the contamination $c(\tilde{\mathcal{X}}) \in \mathbb{R}$ as
$$c(\tilde{\mathcal{X}}) := \frac{\sum_{x \in \tilde{\mathcal{X}}} c'(x)}{|\tilde{\mathcal{X}}|} \tag{4.2}$$
where
$$c'(x) = \frac{|\text{contaminated pixels os } x|}{|\text{total pixels of } x|} \tag{4.3}$$

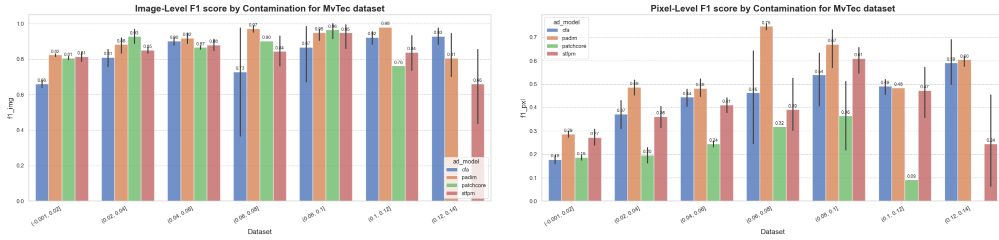
The need for the term $c(\mathcal{X})$ is to give us a measure of how much the contaminated samples are actually contaminated, by individually counting the amount of pixels that are part of the anomalous portion of the image over the total of pixels of which the image is composed of.

4.2 Contaminated Benchmark

Here we have the results for the same benchmarks we have seen in the previous chapter , only in this case we will be using the contaminated version of those same datasets. For each dataset we have set 3 contamination ratios: 0.1, 0.15 and 0.2. We will analyze the behavior of the 4 introduced VAD models using the MVTec [1][10] dataset as our test dataset.

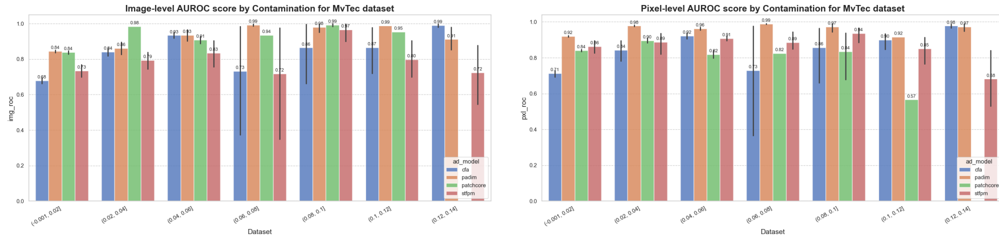
4.3 Contaminated Benchmark Results

In this section we will present the benchmark results for the contaminated datasets, divided by VAD model.



(a) Image-Level F1 Score for PaDiM, CFA, PatchCore and STFPM models trained and tested using contaminated sets

(b) Pixel-Level F1 Score for PaDiM, CFA, PatchCore and STFPM models trained and tested using contaminated sets



(c) Image-Level AUROC Score for PaDiM, CFA, PatchCore and STFPM models trained and tested using contaminated sets

(d) Pixel-Level AUROC Score for PaDiM, CFA, PatchCore and STFPM models trained and tested using contaminated sets

Plots 4.1a, 4.1b, 4.1c and 4.1d show the Image-Level F1, Pixel-Level F1, Image-Level AUROC and Pixel-Level AUROC scores for the PaDiM, PatchCore, CFA and STFPM models trained using the MVTec datasets across all of its categories, contaminated with 10%, 15% and 20% contamination ratios. On the horizontal axis of the plots we represent various ranges for the value of $c(\tilde{\mathcal{X}})$. The values are grouped up in order to make the results more readable, since the value of c varies depending on the category the model was tested on. For the contamination

ratios we tested, the results do see highlight any clear trend that relate the contamination of the datasets and any of the performance metrics

5 PaDiM Optimizations

In this section we will go over the optimizations done on the PaDiM model proposed in [6].

In the next chapter, we will briefly introduce the PaDiM model and describe its training and inference pipeline. We will then propose an improvement to the PaDiM model aimed at reducing memory usage and improving training and inference times in a TinyML scenario. Specifically, this involves modifying the computation of the Mahalanobis distance during inference by using only the diagonal of the covariance matrix, instead of the full $\Sigma_{i,j}$ matrix. This change not only improves the speed of the inference step but also significantly reduces the model's memory footprint, which otherwise scales linearly with the size of the image sample.

5.1 PaDiM Model

The PaDiM model is an embedding similarity-based method [6] that adopts pre-trained deep learning methods for embedding extraction and then uses a "probability distribution distance approach" to compute the anomaly score at pixel level, resulting in an anomaly map from which no only we are able to detect whether or not the given sample is anomalous, but also perform the localization of the anomaly at image level.

In [6] it is shown how PaDiM is able to out perform state of the art models like SPADE [7] on the MVTec[1][10] dataset

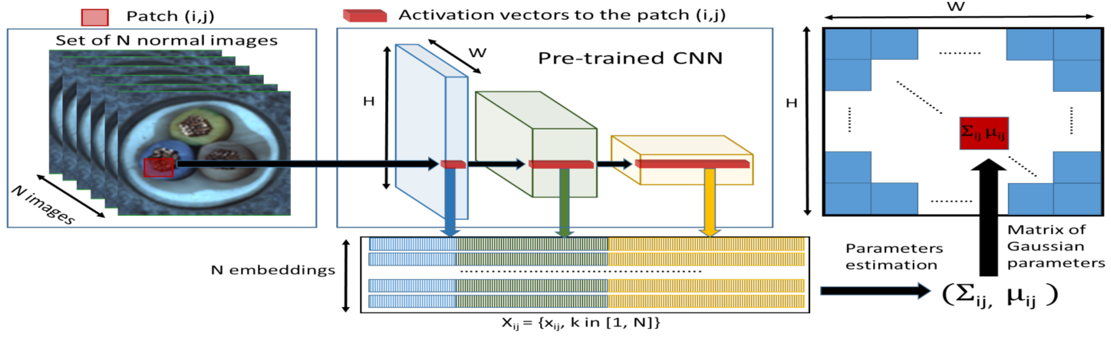


Figure 5.1: PaDiM model training pipeline overview

Training Pipeline The training of the PaDiM model is achieved in a unsupervised method, hence the model does not require a labeled dataset in order to be trained. Given then an unsupervised composed of solely "normal images" dataset $\mathcal{X}_{train}$ the PaDiM model aims at learning features that characterize the normal images. This is achieved by first computing the set of patch embedding vectors :

$$\mathcal{X}_{i,j} = \{x^k_{i,j}, k \in [1, N]\} \tag{5.1}$$

from the N normal images in the dataset. The embeddings for each image are computed from a pre-trained feature extractor $\mathcal{F}(x)$ that returns a vector in $\mathbb{R}^d$ representing the patch level embeddings.

Now, starting from the assumption that the patch embedding vector $\mathcal{X}_{i,j}$ is generated from a normal distribution $\mathcal{N}(\mu_{i,j}, \Sigma_{i,j})$, the model the mean computes $\mu_{i,j}$ from $\mathcal{X}_{i,j}$ and estimates $\Sigma_{i,j}$ as:

$$\Sigma_{i,j} = \frac{1}{N-1} \sum_{k=1}^{N} (\mathbf{x}^k_{i,j} - \mu_{i,j})(\mathbf{x}^k_{i,j} - \mu_{i,j})^T + \epsilon I \tag{5.2}$$

This results in a series of (assumed to be) normal gaussian distributions, one for each image patch, that during inference is used to compute an anomaly score for a given patch [6].

Inference pipeline At inference time, the anomaly score is computed by exploiting the estimated normal distribution $\mathcal{N}(\mu_{i,j}, \Sigma_{i,j})$ from each patch. Let x_{test} be the test image input to classify, the same feature extractor $\mathcal{F}$ used during training, is used to extract the patch level features of x_{test}, then the patch level embeddings are compared to the respective normal distribution $\mathcal{N}(\mu_{i,j}, \Sigma_{i,j})$ associated to patch (i, j) using the Mahalanobis distance[6]. The result of the Mahalanobis distance computation is then used to compute the anomaly scores, used to

determine whether or not the image is anomalous or not.

5.2 Mahalanobis Distance With Diagonal Covariance

At inference time, the anomaly score in the PaDiM model of sample x at position (i, j), is achieved through the computation of the Mahalanobis distance between the $x_{i,j}$ and the gaussian distribution $\mathcal{N}(\Sigma_{i,j}, \mu_{i,j})$:

$$M(x_{i,j}) = \sqrt{(x_{i,j} - \mu_{i,j})^T \Sigma_{i,j}^{-1} (x_{i,j} - \mu_{i,j})} \tag{5.3}$$

where $\Sigma_{i,j}$ is our covariance matrix for patch (i, j) computed at training time. The idea is to then substitute the covariance matrix in the computation of 5.3 with its diagonal. This should allow us to substantially improve the performance of the computation of the Mahalanobis distance, given that we go from a computation that involves an entire matrix of $d \times d$ to a single vector of length d, thus reducing the complexity of the computation from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. Using the diagonal of $\Sigma_{i,j}$ means that we need to change the formulation of 5.3: taking into account that every component of $\Sigma_{i,j}$ outside of the diagonal is equal to 0, that is $\Sigma_{i,j}(h, k) = 0, \forall(h, k) : h \neq k$, after some algebraic manipulation we obtain that:

$$M(x_{i,j}) = \sqrt{\sum_{h=0}^{n} (x_{ij,h} - \mu_{ij,h})^2 \Sigma_{ij,h,h}^{-1}} \tag{5.4}$$

where Σ_{ij}^{-1} is the inverse of the covariance matrix associated to patch (i, j) and $\Sigma_{ij,h,h}^{-1}$ is the h-th element on the of Σ_{ij}^{-1} diagonal. Then since for a diagonal matrix we have that

$$\Sigma_{i,j}^{-1} = \begin{bmatrix} \frac{1}{\sigma_{1,1}} & \cdots & 0 \\ \vdots & \frac{1}{\sigma_{k,k}} & \vdots \\ 0 & \cdots & \frac{1}{\sigma_{n,m}} \end{bmatrix} \tag{5.5}$$

the Mahalanobis distance in the case of a diagonal covariance matrix can be computed as

$$M(x_{i,j}) = \sqrt{\sum_{h=0}^{n} \frac{(x_{ij,h} - \mu_{ij,h})^2}{\sigma_{ij,h,h}}} \tag{5.6}$$

the computation of 5.6 is significantly less complex and resource hungry than 5.3, thus resulting in a faster computation of the patch anomaly score and the inference step.

6
PatchCore Optimizations

In the following section we will be exploring the optimizations made on the PatchCore model. We will first do a brief introduction to the model, along with the training and inference pipelines. After that we will introduce two optimizations: the first involves around a compression approach applied onto the PatchCore Memory bank with the goal of optimizing the performance at memory level of the inference step, and the second is the introduction of what we call PQNN Search that replaces the original Greedy Nearest-Neighbor search of the original model, with the goal of reducing memory usage at inference time.

6.1 PatchCore Model

Similarly to the PaDiM model, PatchCore is another embedding similarity based model. Instead of relying on a series of gaussian distributions, like PaDiM [6], the PatchCore model relies on the usage of memory bank, create and stored during training and applying a nearest-neighbor search approach. Its effectiveness comes from comparing feature representations of test patches against a memory bank of normal patch features.

At inference time, given a sample $x^{test} \in \mathcal{X}_{test}$, the PatchCore model computes the anomaly score $s \in \mathbb{R}$ from the maximum distance score s between the patch feature collection computed from x_{test} as [4]:

$$\mathcal{P}(x^{test}) = \mathcal{P}_{s,p}(\phi_j(x^{test})) \tag{6.1}$$

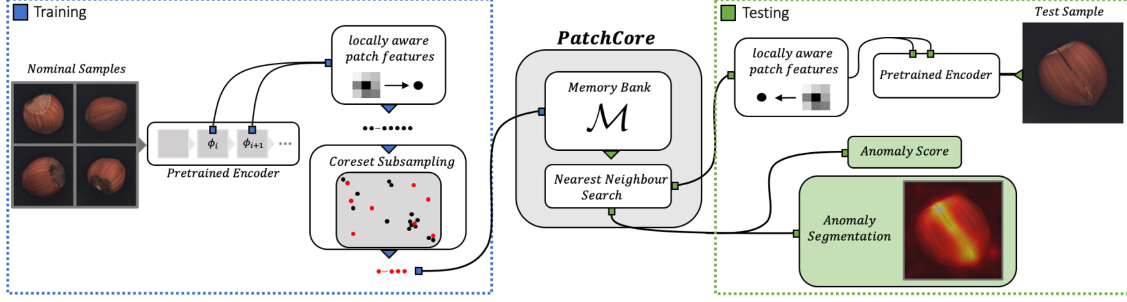


Figure 6.1: PatchCore Model overview.Source: [4]

and each nearest neighbor m in $\mathcal{M}$:

$$m^{test,}, m = \arg \max_{m^{test} \in \mathcal{P}(x^{test})} \arg \min_{m \in \mathcal{M}} ||m^{test} - m||_2 \tag{6.2}$$

$$s^* = ||m^{test} - m||_2 \tag{6.3}$$

The anomaly score s^* represents the maximum distance between any test patch feature and its closest normal patch feature in the memory bank. Higher values indicate greater deviation from normality, suggesting anomalous regions.

For image-level anomaly detection, the final score can be derived using the maximum patch-level score:

$$s_{image} = \max_{m^{test} \in \mathcal{P}(x^{test})} ||m^{test} - NN_{\mathcal{M}}(m^{test})||_2 \tag{6.4}$$

where $NN_{\mathcal{M}}(m^{test})$ computes the Nearest-Neighbour of m^{test} in the memory bank $\mathcal{M}$. For pixel-level anomaly detection (segmentation), PatchCore generates anomaly maps by up-sampling the patch-level scores to the original image dimensions. This is achieved by mapping each feature's anomaly score back to its corresponding spatial location in the input image, resulting in a heatmap that highlights anomalous regions.

6.2 Coreset Greedy Selection

In the original PatchCore model proposed in [4] the memory bank $\mathcal{M}$ is computed through a coreset sub-sampling mechanism, which aims at selecting a subset of meaningful feature maps given by the PatchDescriptor component. This coreset selection has the main purpose of selecting a subset $\mathcal{A}$ of the computed coreset $\mathcal{C}$ such that the problem solutions over $\mathcal{A}$ can be most closely and quickly approximated by those components over $\mathcal{S}$. In [4] a min-max facility

location is used :

$$\mathcal{M}_C^* = \arg \min_{\mathcal{M}_C \subset \mathcal{M}} \max_{m \in \mathcal{M}} \min_{n \in \mathcal{M}_C} \|m - n\|_2. \tag{6.5}$$

The computation of 6.5 is NP-Hard , thus an iterative greedy approximation approach is used [4]. Additionally, during the Coreset selection process linear random projections are used in order to further reduce the coreset selection time. The idea is to apply dimensionality reduction to the components $m \in \mathcal{M}$ in order to optimize the computation of 6.5 using random linear projections $\psi : \mathbb{R}^d \rightarrow \mathbb{R}^{d^*}$, where $d^* < d$ [4]. The complete algorithm can be found in Chapter 3.2 from [4].

This approach shows good improvement in inference time without showing significant loss in performance in both the classification and segmentation tasks. The PatchCore model, however, heavily relies on the usage of this memory bank during inference, this means that whenever we want to export this model and load it onto specific device, the size of the memory bank greatly affects the memory usage of the that specific device

In this thesis, we propose a direct compression of the Memory bank $\mathcal{M}$ in addition to the coreset selection used in [4]. Our approach involves using the power of Product Quantization in order to compress even more the memory bankj that is then used at inference time.

6.3 Memory-bank Quantization

As we have seen in at the start of the chapter, the PatchCore model computes the anomaly score at inference time using a Nearest-Neighbour search between the patch features collection $\mathcal{P}(x^{test})$ and the memory bank $\mathcal{M}$ computed at training time.

The computation of the NN can be quite cumbersome, especially in high dimensionality cases, like this one. In fact, the heaviest part of the nearest neighbor search stands on the euclidean distance computation. This causes a very high usage of computational resources, which is not ideal in a tiny ML scenario.

What we propose is an optimization of the nearest neighbor search by applying product quantization to the embeddings $m \in \mathcal{M}$, or **Partially Quantized Nearest Neighbor Search**. This approach aims at reducing the final size of the memory bank using **Product Quantization** in order to reduce the strain of the model on the system memory during the inference process.

6.3.1 Product Quantization

First let us introduce the concept of Product Quantization. Product Quantization is a very common method for vector dimensionality reduction that can be used whenever any type of vector similarity search process is applied. The main idea behind PQ is to partition a given vector $x \in \mathbb{R}^D$ into a list of subvectors $u = [u_1, u_2, ..., u_m]$ where each $u_i \in \mathbb{R}^{D^*}$ with

$$D^* := \frac{D}{m} \tag{6.6}$$

. Finally, each subvector is then assigned to a specific centroid from a computed set during a

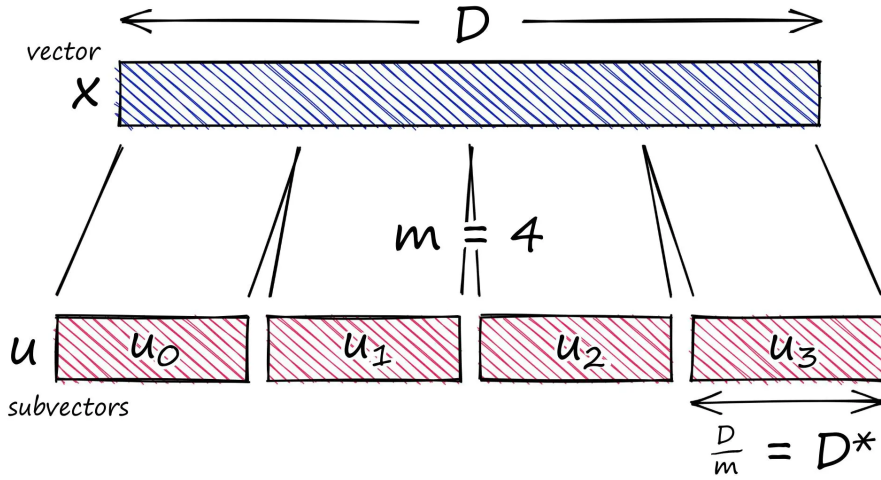


Figure 6.2: Product Quantization sub vector. Source: [5]

training phase from a given dataset, ending up with a compressed vector $\hat{x} = q(u) \in \mathbb{R}^h$, where $h = m \cdot k^*$ and k^* is the size need to identify a generic centroid from the centroid set.

6.3.2 Partially Quantized Nearest Neighbor Search

In this section we introduce what we call the Partially Quantized Nearest Neighbor Search, PQNN Search for short. This process aims at reduce the size of the memory bank during the inference step of the model by performing two subsequent searches:

1. **Partial Search phase**: First we perform a NN search on a neighborhood by extracting K neighbors for each quantized entry $\hat{x}$ from $\mathcal{M}$;
2. **Refined Search phase**: Then we perform final and more accurate NN search using the decompressed entries $\tilde{x}$ on the K points we found from the previous search, extracting the top c nearest neighbors we need.

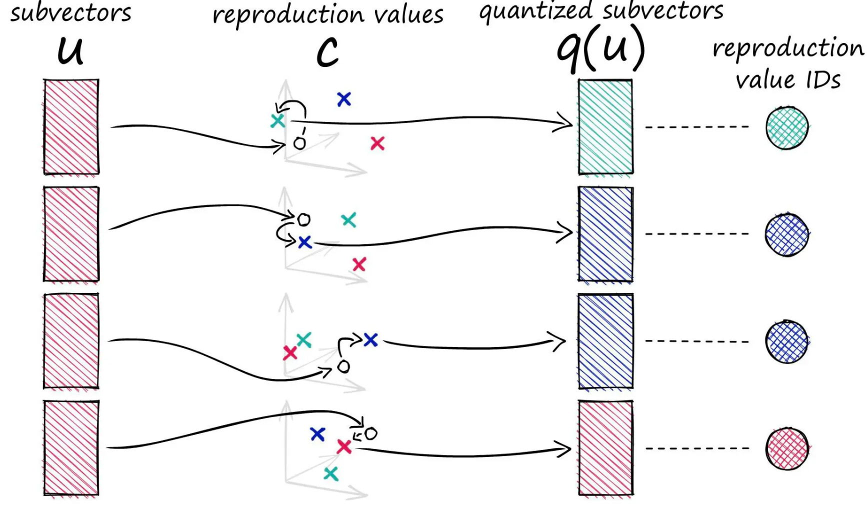


Figure 6.3: Product Quantization. Source: [5]

PARTIAL SEARCH PHASE For every embedding $v \in \mathcal{P}(x)$, where x is the input sample and $\mathcal{P}(x)$ is the patch feature collection, we compute the anomaly score using 6.3. These anomaly scores are initially computed against the quantized memory bank $\hat{\mathcal{M}}$:

$$\hat{s}^* = ||\hat{m}_x - \hat{m}|| \tag{6.7}$$

with $\hat{m}_x$ being the compressed/quantized embedding from $\mathcal{P}(x)$ and $\hat{m} \in \hat{\mathcal{M}}$. Note that $\hat{m}_x$ must be computed using the same product quantizer used for the memory bank compression achieved during training: this is essential so as to place the compressed embedding in the same vectorial space as in the quantized memory bank entries $\hat{m}$.

Once we have computed the anomaly scores, we take the top K as the nearest neighbors for each $v \in \mathcal{P}(x)$, along with their locations in the memory bank $\hat{\mathcal{M}}$, ending up with a matrix $A \in \mathbb{R}^{n \times K}$, where $n = |\mathcal{P}(x)|$ of vectors of size D^* (as defined in 6.6).

REFINED SEARCH PHASE Starting from the matrix A of neighbors computed in the Partial Search Phase, we now apply what we define as the Refined Search Phase. This process involves decompressing the K vectors from A for each embedding v and computing again the anomaly score s^* this time using the dequantized versions of both the embeddings and the memory bank. The goal of the Refined Search Phase is to compute the nearest-neighbor with more accuracy starting from the neighborhood computed from the Partial Search Phase.

6.3.3 Hyperparameter Tuning: K and c in PQNN-Based Anomaly Detection

The performance of the PatchCore training pipeline is significantly influenced by two key hyperparameters: K and c. These control, respectively, the breadth of the candidate search space in the quantized phase (this is also the same parameter that controls the search space in the greedy NN search from the original model [4]) and the granularity of the refined search phase. Proper selection of these parameters balances detection accuracy with memory and computation constraints, which is critical in the context of TinyML. Regarding this particular issue, we will include in chapter 8 the comparison between the original PatchCore approach [4] and our optimizations for various vales of K and c.

Parameter Definitions

- **K: Quantized Neighborhood Size**
  This parameter determines how many nearest neighbors are retrieved in the *quantized (coarse) search phase*. These candidates are selected using approximate distances computed via PQ codebooks and precomputed lookup tables. Increasing K generally improves recall but incurs greater memory access and latency during inference.

- **c: Refined Neighborhood Size**
  From the K quantized candidates, a smaller subset of size $c \leq K$ is chosen for *refined (exact) distance computation*. These distances may be computed using the original (non-quantized) feature vectors or a more accurate asymmetric distance measure (e.g., ADC). The final anomaly decision is based on the exact distances to this refined subset.

Role in Anomaly Detection

In a visual anomaly detection scenario, both K and c influence the *sensitivity and specificity* of the model:

- A small K may lead to missing relevant neighbors due to quantization noise, increasing false positives.

- A large K ensures better coverage but may exceed resource limits or slow down inference.

- A small c in the refinement phase may cause poor local ranking and noisy anomaly scores.

- A larger c improves scoring precision but adds to computation and latency, especially if full-resolution feature vectors are used.

Algorithm 6.1 PatchCore - PQNN Search

1: **procedure** PQNNSEARCH($embedding, n_neighbors$)
2: *Move memory bank to current device*
3: $quantized_embedding \leftarrow$ Encode $embedding$ using product quantizer
4: *Move $quantized_embedding$ to current device*
5: $distances \leftarrow$ Compute Euclidean distances between $quantized_embedding$ and memory bank
6: $(patch_scores, locations) \leftarrow$ Get top K nearest neighbors from $distances$
7: $K_neighbors \leftarrow$ Extract neighbors from memory bank using $top_K_locations$
8: *Initialize empty lists $patch_scores$ and $locations$*
9: **for** $embedding_index$ from 0 to $|K_neighbors| - 1$
10: $neighbours \leftarrow K_neighbors[embedding_index]$
11: $decoded_neighbors \leftarrow$ Decode $neighbours$ using product quantizer
12: $embedding_value \leftarrow embedding[embedding_index]$ (unsqueezed)
13: *Move $decoded_neighbors$ to current device*
14: $neighbour_distances \leftarrow$ Compute Euclidean distances between $embedding_value$ and $decoded_neighbors$
15: $(top_patch_score, top_location) \leftarrow$ Get top $n_neighbors$ nearest neighbors from $neighbour_distances$
16: $patch_scores$.append(top_patch_score)
17: $locations$.append($top_location$)
18: **end for**
19: $patch_scores \leftarrow$ Concatenate and squeeze all scores
20: $locations \leftarrow$ Concatenate and squeeze all locations
21: **return** $(patch_scores, locations)$
22: **end procedure**

36

7

Experiments

This chapter is dedicated to the evaluation of the optimization applied to the PaDiM and PatchCore models. The goal of these experiments is to compare the performance of the optimized models to the originally proposed ones in [6] and [7] and verify the gain in terms of training/inference time and memory utilization.

Here is the list of experiments we performed:

1. Training time and memory performance for the Original PaDiM model against PaDiM with Diagonalized Covariance matrix;
2. Memory performance, training and inference time for Original PatchCore model against PatchCore with PQNN;
3. Parameter K, c variation analysis for the PQNN Search in PatchCore;

What we expect from these experiments:

- Overhaul running time improvement of the PaDiM model, during both training and inference;
- Reduced memory allocation of the Memory Bank of the PatchCore model;

All while having no heavy impact of the performance of the original models.

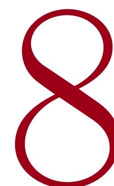
7.1 Experimental Setup

All experiments were conducted on a workstation running Ubuntu 22.04.5 LTS (64-bit) with kernel version 6.5.0-26-generic. The hardware configuration features an AMD Ryzen Threadripper PRO 5995WX processor with 128 logical cores running at 2.7 GHz. The machine is equipped with an NVIDIA RTX A6000 GPU, along with 512 GB of RAM, with approximately 24 GB utilized during the experiments.

All the experiments were run using the MVTec[10] [1] for both training and test dataset and the MobileNetV2[18] as the backbone with the same configuration specified in table 3.1. The pre-processing of the images is the same as explained in section 3.5.

7.2 MoViAD Library

The MoViAD library is an open-source Python framework designed for industrial and research in visual anomaly detection. It offers a unified environment for managing datasets, implementing and training models and conducting evaluations. MoViAD emphasizes modularity and reproducibility, enabling researchers to build, customize, and compare various anomaly detection approaches within the same environment. All the work done for this thesis, all experimental implementations, including model training, dataset preprocessing, and evaluation were carried out using the MoViAD library and its infrastructure. The complete implementations of our experiments, benchmarks, models, and datasets referenced in this thesis are available in the MoViAD source repository [23].



8 Experiment Results

In this section we will go over the the experiments performed over the implementations and the respective results.

8.1 Evaluation metrics

As evaluation metrics we will be using the same ones used in 3.6 for the benchmarks. In addition to those, we will be evaluating other two metrics that evaluate the model from a resource allocation perspective:

- **Inference/training time**: this measures the time (in seconds (s)) the time taken by the model in question to perform an single training/inference step. In the case of training time measurement, by training step we mean the process of taking a batch from the training set and completing one single *epoch* on that batch (we will be specifying the batch size), while in the case of inference we will take into account the time needed to perform a full classification of a single sample.
- **Object Memory Dump**: this measure refers particularly in the PatchCore case, where we want to measure the memory impact of the quantization on the memory bank. The Object Memory Dump tells us the size (in bytes) allocated by the hardware (more technically by the compiler, but for the sake of simplicity we assume this is purely depending on the hardware) in its computation memory (more commonly known as RAM).

8.2 PaDiM Experiments

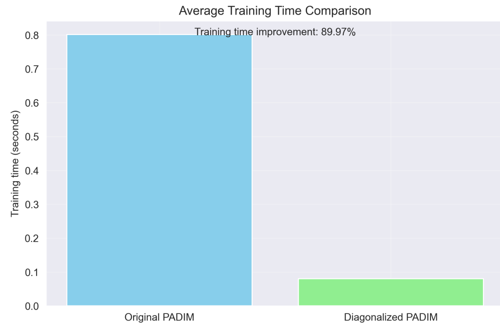
In the first experiment we will evaluate the performance we gained by applying the diagonalization of the covariance matrix during training, in particular we will be focusing on the training time gain and the memory performance between the originally proposed PaDiM training method from [6] and the optimization we applied. What we are looking for is an improvement of the training time and a difference in the memory usage, again during the training phase of the model:

Category	Original Time (s)	Diagonalized Time (s)	Improvement (%)
toothbrush	0.847	0.066	92.194
zipper	0.784	0.074	90.561
bottle	0.782	0.075	90.415
wood	0.791	0.076	90.410
metal nut	0.778	0.075	90.407
capsule	0.787	0.076	90.338
transistor	0.783	0.077	90.195
carpet	0.860	0.085	90.059
tile	0.813	0.081	90.015
leather	0.780	0.078	89.992
cable	0.788	0.081	89.750
grid	0.785	0.084	89.295
hazelnut	0.839	0.092	89.064
pill	0.786	0.086	89.038
screw	0.802	0.099	87.622

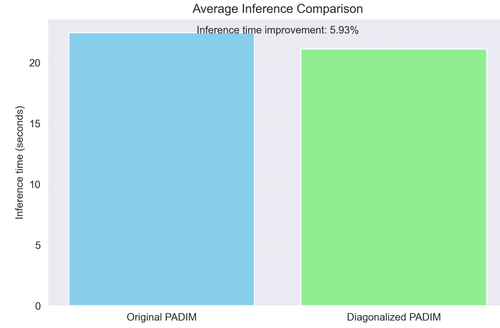
Table 8.1: Table showing the time training PaDiM time difference

From the table we can easily a major improvement of the diagonalized approach over the original method. This is also without any visible loss in terms of training performance:

Plots 8.2 show the performance comparison between the diagonalized version of the PaDiM model against the original PaDiM model:



(a) PaDiM Training time comparison



(b) PaDiM Inference time comparison

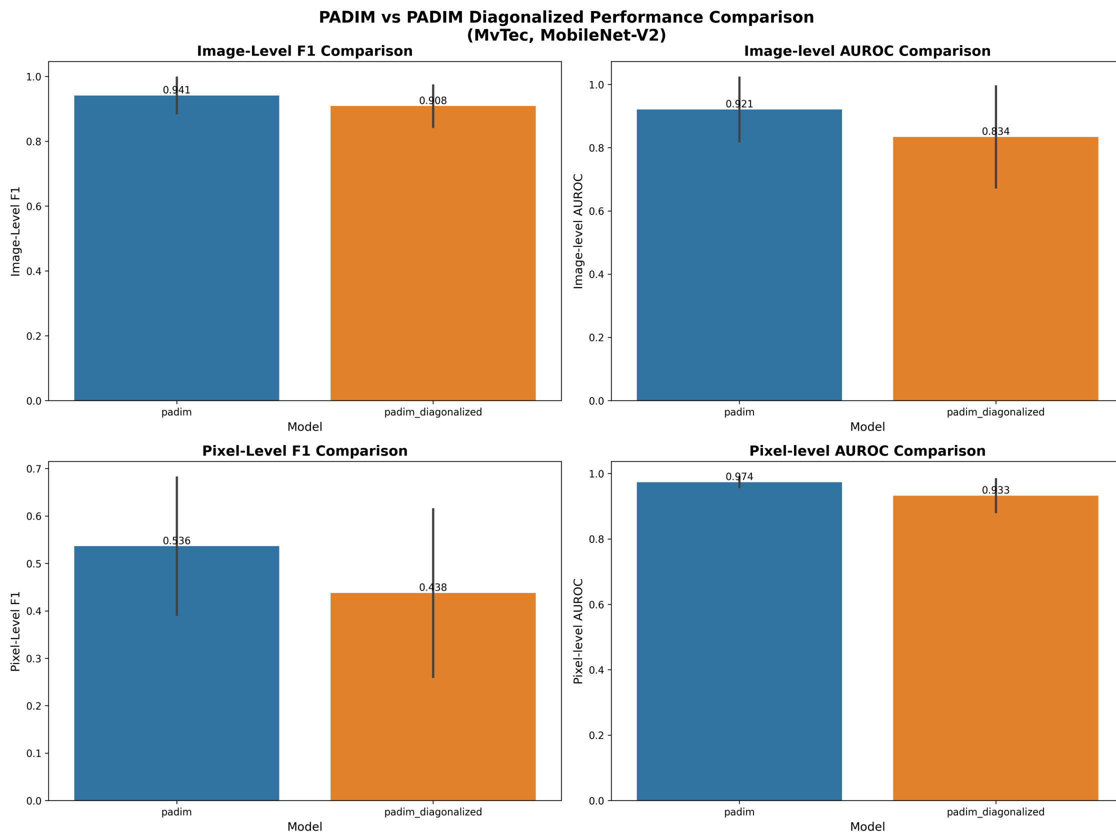


Figure 8.2: Original PaDiM vs Diagonalized PaDiM Performance Comparison (MVTec, MobileNet-V2)

8.3 PatchCore Experiments

8.3.1 Quantized PatchCore

Here we take a look at the improvement achieved using product quantization on the memory bank reduction: what we are looking for is an improvement over the memory usage for the

memory bank. We will then take a look at the Object memory dump for the Memory bank.

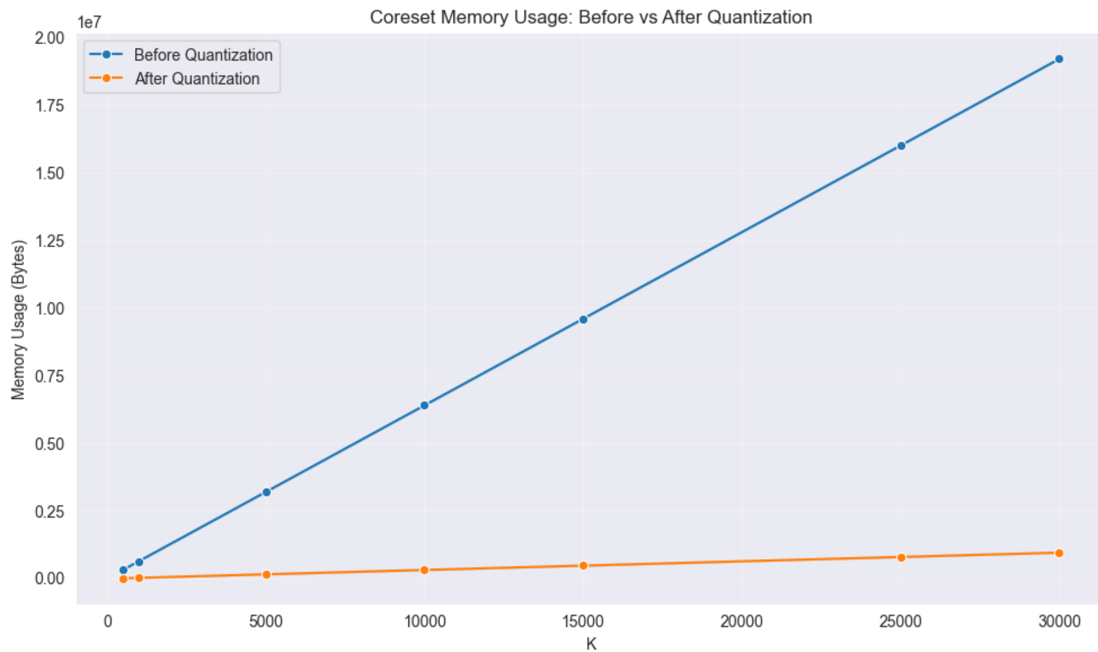


Figure 8.3: Memory Bank memory usage during inference in function of K (neighborhood size) (MVTec, MobileNet-V2)

Plot 8.3 compares the Object Memory dump (in bytes) for the memory bank of the PatchCore model against the same Memory bank compressed using Product Quantization across multiple values for c, while 8.4 shows the performance comparison between the original PatchCore model and out version with Product Quantization and PQNN Search for $K = 300$, $c = 1000$.

8.3.2 PQNN Search Performance

We have already seen the performance comparison between the quantized version of the PatchCore model, which includes the PQNN Search, what are interested here is to analyzed how the performance of the quantized PatchCore model with PQNN Search model varies depending on the two hyper-parameters K, c introduced in section 6.3.3.

Plots 8.5, 8.6, 8.7, 8.8 show the average Image-level F1 score, Pixel-Level F1 score, Image-Level and Pixel-Level AUROC scores respectively across all categories of the MVTec dataset.

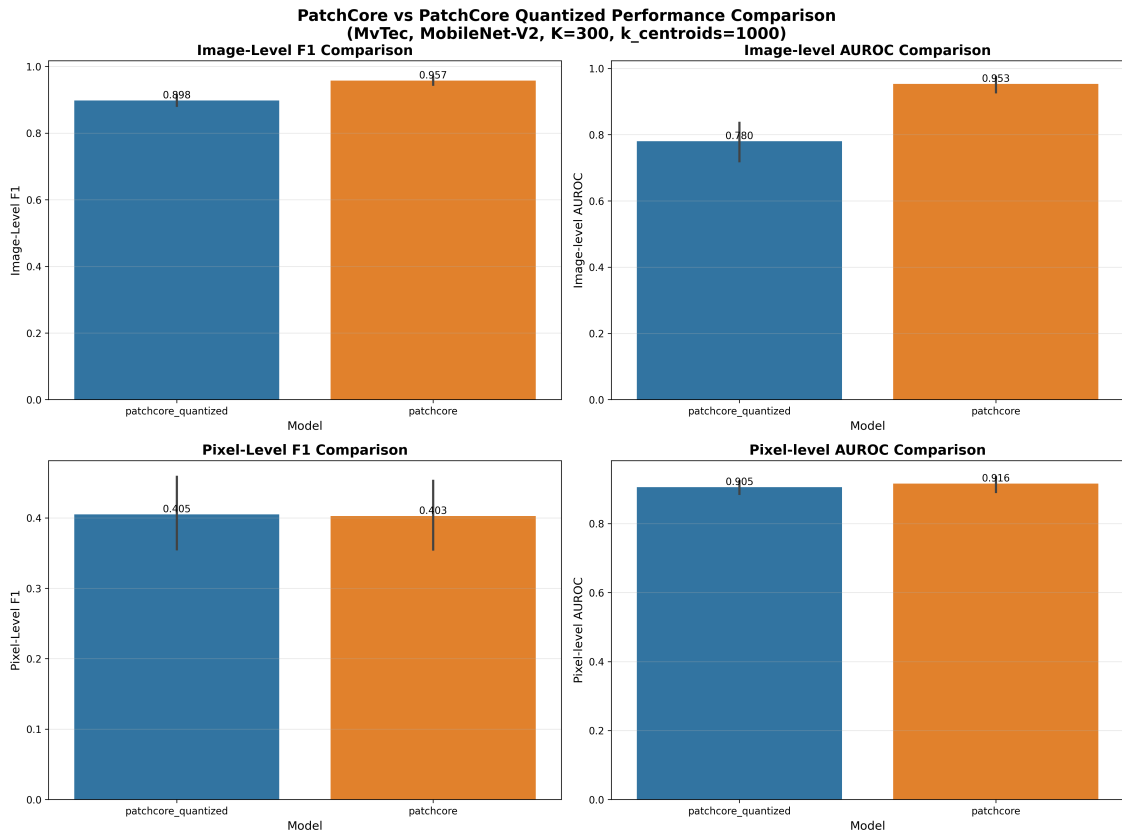


Figure 8.4: PatchCore vs PatchCore Quantized Performance Comparison(MVTec, MobileNet-V2, K=300, c=1000)

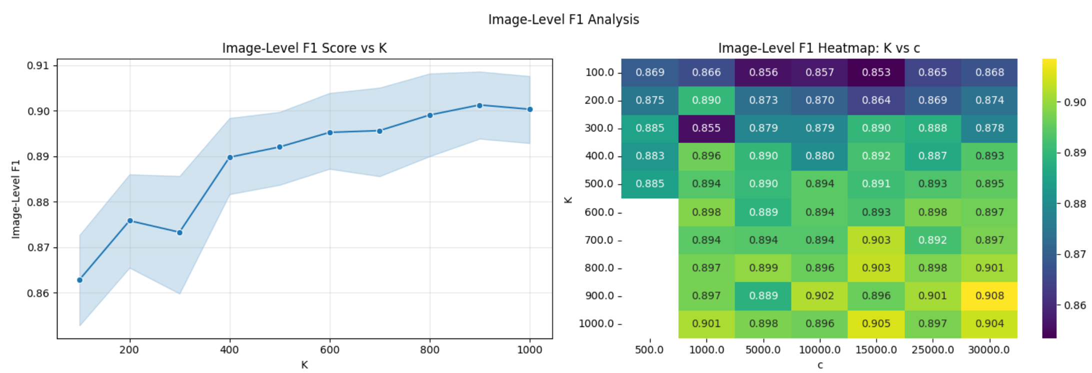


Figure 8.5: image-Level F1 Score with multiple values of K and c

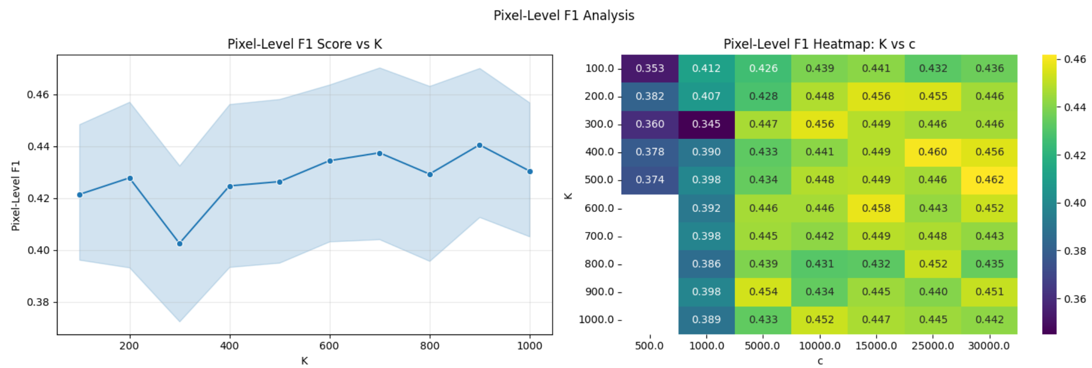


Figure 8.6: Pixel-Level F1 Score with multiple values of K and c

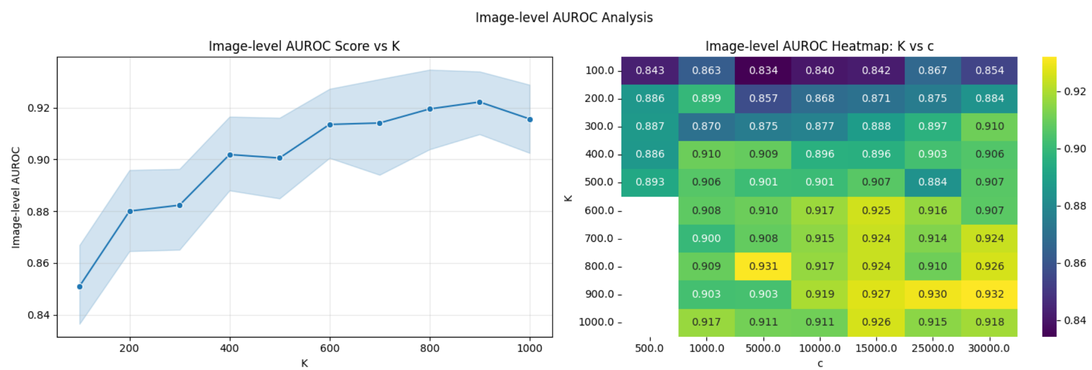


Figure 8.7: Image-Level AUROC score with multiple values of K and c

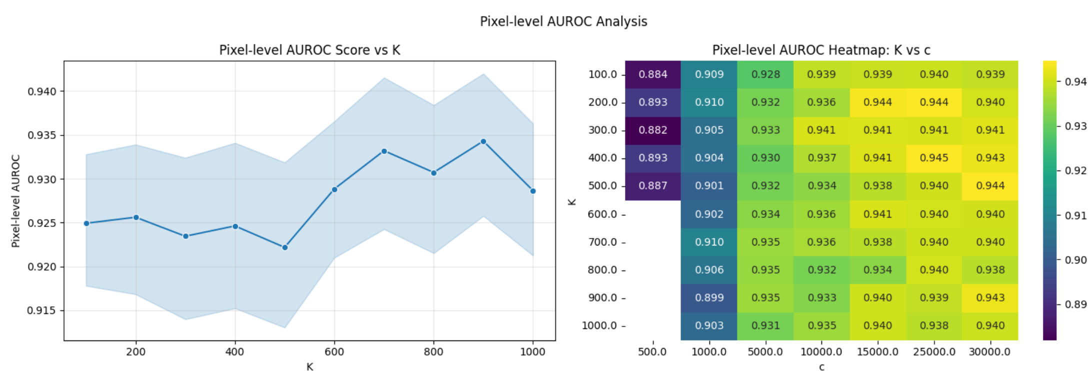


Figure 8.8: Pixel-Level Score AUROC with multiple values of K and c

9
Conclusions & Future Work

9.1 Experiment Results Observations

Looking at the plots 8.2 and 8.1a we can see over 90% of improvement on the training time, and approximately 5% on inference time. As expected, we have a slight performance loss across the metrics when applying the diagonalization on PaDiM, however we managed to keep the performance loss within a very reasonable margin, making the diagonalization of the covariance matrices a viable optimization onto the originally proposed PaDiM model [6] .

Like we managed with the PaDiM model, we manged to tremendously shrink the Memory bank size of the PatchCore model, while having no significant loss in performance. As far as the PQNN Search concerns, from plots 8.5, 8.6, 8.7, 8.8 we can see how, as we would expect, the performance of the PatchCore model increases with the size of the parameter c. However, it is not entirely the same story with the parameter K: in fact, across the tested values for K does not change (or at least not noticeably). This is an indication how the $PQNN$ Search performs.

9.2 Final Considerations and Conclusions

Through the benchmarks of the VAD models, we established a performance baseline while introducing results with two new datasets: Real-IAD and ViSA. The most interesting findings come from benchmarking on the Real-IAD dataset: this dataset is by far the largest among the

three we tested and takes data directly from a real-world scenario. Thus, testing the models with the Real-IAD dataset is the closest approximation we have to assessing performance in an actual production environment. As expected, all models show noticeable differences when tested on the three datasets, with the Real-IAD dataset resulting in the lowest performance across the board. These results highlight two key points: first, that all models performing with a confidence lower than 70% on the Real-IAD dataset suggests VAD applications are still far from being viable in real-world production environments; second, the models show a certain level of bias toward the different classes of objects they classify.

In the contaminated scenario benchmarks, we noticed an unexpected phenomenon: for certain ranges of contamination, performance actually improved compared to having no contamination. This contradicts our initial expectation, which assumed that at best the models would perform slightly worse as contamination increased. This result certainly warrants further investigation.

The backbones we tested show substantial differences as well: MobileNet V2 [18] consistently delivered the best average results across all models and datasets, while Micronet [22] performed the worst. Due to hardware and time limitations, we were unable to provide consistent benchmarks and experiments using WideResNet as the backbone for our models. However, even if WideResNet had delivered the best performance (which we would expect), it would not be suited for the scenario we targeted, given that WideResNet is among the deepest neural networks for image recognition known in literature.

Regarding our optimizations, we successfully implemented them for the PaDiM and PatchCore models, achieving promising results in both performance and resource utilization. In particular, we significantly reduced PaDiM's training time and improved inference time without notable impact on performance. Same goes for for the optimizations we made on the PatchCore model, however with a trade-off to consider: the quantization process, combined with the PQNN Search, not only impacts the memory used by the model, but also affects the inference time. In fact, during our experiments, we noticed a higher inference time of our version of the PatchCore model.

9.3 Future Work

One of the first future extensions of this thesis is to expand the experiments to all datasets and backbones. This would provide a more robust measure of the impact of the optimizations applied to the PaDiM and PatchCore models, in addition to verifying whether the results ob-

tained thus far are consistent.

Another promising direction for future work is the implementation and evaluation of a batched coreset construction strategy for the PatchCore model. In its original formulation, PatchCore performs coreset selection over the entire set of extracted features, which can be memory-intensive and computationally expensive, especially on large datasets or high-resolution images. By splitting the data into smaller batches and performing coreset selection incrementally, it may be possible to reduce peak memory usage and improve scalability during training. This approach could make the training pipeline more suitable for deployment on limited-resource hardware or integrated within continuous learning systems. Further work is needed to explore the trade-offs in accuracy and efficiency introduced by batching, and to determine optimal batching strategies (e.g., batch size, merging heuristics) for different use cases.

A potential future improvement for the PaDiM model involves adopting a batched strategy for estimating the covariance matrices during training. In the current implementation, covariance matrices are computed over the entire set of extracted embeddings at once, which can be memory-intensive and slow, particularly with high-resolution images or large datasets. Instead, the covariance matrices could be incrementally updated across batches of training samples using an online or streaming covariance estimation method. This would reduce peak memory usage and make the model training more scalable and compatible with constrained hardware.

10 Appendix

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
bottle	0.962	0.632	0.938	0.931
cable	0.847	0.409	0.814	0.902
capsule	0.951	0.375	0.841	0.945
carpet	0.943	0.477	0.855	0.872
grid	0.926	0.250	0.868	0.851
hazelnut	0.827	0.464	0.680	0.961
leather	0.965	0.433	0.920	0.950
metal nut	0.948	0.645	0.833	0.913
pill	0.937	0.495	0.834	0.925
screw	0.862	0.167	0.673	0.954
tile	0.941	0.509	0.936	0.861
toothbrush	0.923	0.447	0.846	0.975
transistor	0.818	0.583	0.886	0.928
wood	0.959	0.448	0.964	0.908
zipper	0.944	0.453	0.859	0.949

Table 10.1: PaDiM metrics for MVTec by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
candle	0.775	0.088	0.815	0.918
capsules	0.778	0.057	0.631	0.897
cashew	0.862	0.413	0.854	0.917
chewinggum	0.925	0.192	0.901	0.890
fryum	0.869	0.344	0.815	0.940
macaroni1	0.762	0.009	0.774	0.948
macaroni2	0.702	0.003	0.623	0.942
pcb1	0.806	0.314	0.817	0.961
pcb2	0.734	0.075	0.725	0.961
pcb3	0.712	0.157	0.693	0.968
pcb4	0.844	0.196	0.839	0.944
pipe fryum	0.880	0.456	0.814	0.978

Table 10.2: PaDiM metrics for ViSA by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
audiojack	0.596	0.033	0.674	0.884
bottle cap	0.736	0.072	0.789	0.768
button battery	0.798	0.066	0.747	0.828
end cap	0.763	0.040	0.700	0.836
eraser	0.687	0.071	0.759	0.835
fire hood	0.607	0.049	0.685	0.824
mint	0.643	0.015	0.566	0.638
mounts	0.658	0.138	0.750	0.855
pcb	0.773	0.016	0.701	0.892
phone battery	0.640	0.041	0.684	0.749
plastic nut	0.586	0.071	0.712	0.698
plastic plug	0.598	0.030	0.656	0.768
porcelain doll	0.567	0.014	0.670	0.806
regulator	0.489	0.039	0.617	0.827
rolled strip base	0.907	0.092	0.885	0.898
sim card set	0.784	0.052	0.785	0.834
switch	0.753	0.195	0.753	0.891
tape	0.775	0.118	0.844	0.832
terminalblock	0.777	0.115	0.772	0.888
toothbrush	0.726	0.047	0.678	0.801
toy	0.790	0.017	0.710	0.815
toy brick	0.614	0.025	0.623	0.754
transistor1	0.822	0.080	0.797	0.864
u block	0.610	0.029	0.730	0.760
usb	0.677	0.024	0.675	0.877
usb adaptor	0.631	0.035	0.699	0.784
vcpill	0.626	0.092	0.671	0.873
wooden beads	0.704	0.037	0.745	0.838
woodstick	0.525	0.112	0.683	0.681
zipper	0.840	0.083	0.809	0.856

Table 10.3: PaDiM metrics for Real-IAD by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
bottle	0.973	0.482	0.942	0.897
cable	0.896	0.325	0.881	0.836
capsule	0.949	0.314	0.890	0.891
carpet	0.923	0.277	0.818	0.851
cashew	0.877	0.121	0.875	0.758
grid	0.907	0.219	0.838	0.831
hazelnut	0.955	0.301	0.960	0.870
leather	0.949	0.392	0.922	0.938
metal nut	0.949	0.581	0.846	0.873
pill	0.935	0.353	0.865	0.809
screw	0.889	0.211	0.792	0.908
tile	0.960	0.418	0.946	0.826
toothbrush	0.948	0.301	0.940	0.904
transistor	0.842	0.211	0.853	0.657
wood	0.957	0.335	0.964	0.824
zipper	0.944	0.324	0.907	0.873

Table 10.4: PatchCore metrics for MVTec by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
candle	0.782	0.093	0.817	0.802
capsules	0.787	0.121	0.714	0.845
cashew	0.879	0.093	0.885	0.675
chewinggum	0.901	0.145	0.895	0.825
fryum	0.857	0.118	0.829	0.719
macaroni1	0.743	0.038	0.762	0.867
macaroni2	0.683	0.014	0.590	0.811
pcb1	0.811	0.124	0.845	0.788
pcb2	0.828	0.090	0.876	0.846
pcb3	0.795	0.102	0.814	0.851
pcb4	0.877	0.148	0.886	0.848
pipe fryum	0.894	0.132	0.862	0.774

Table 10.5: PatchCore metrics for ViSA by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
audiojack	0.645	0.073	0.757	0.795
bottle cap	0.710	0.038	0.763	0.729
button battery	0.749	0.086	0.704	0.763
end cap	0.748	0.048	0.701	0.751
eraser	0.707	0.069	0.784	0.799
fire hood	0.602	0.048	0.705	0.787
mint	0.646	0.025	0.647	0.672
mounts	0.685	0.110	0.782	0.807
pcb	0.779	0.101	0.724	0.849
phone battery	0.674	0.080	0.735	0.742
plastic nut	0.583	0.052	0.708	0.666
plastic plug	0.644	0.050	0.744	0.741
porcelain doll	0.611	0.042	0.735	0.764
regulator	0.479	0.031	0.611	0.761
rolled strip base	0.904	0.163	0.891	0.868
sim card set	0.858	0.127	0.908	0.799
switch	0.755	0.118	0.774	0.795
tape	0.785	0.108	0.859	0.781
terminalblock	0.767	0.097	0.780	0.834
toothbrush	0.767	0.093	0.780	0.806
toy	0.746	0.045	0.650	0.766
toy brick	0.612	0.057	0.655	0.745
transistor1	0.844	0.096	0.841	0.821
u block	0.658	0.074	0.773	0.729
usb	0.729	0.081	0.770	0.807
usb adaptor	0.608	0.027	0.687	0.719
vcpill	0.730	0.129	0.835	0.835
wooden beads	0.702	0.086	0.775	0.847
woodstick	0.508	0.047	0.663	0.685
zipper	0.895	0.214	0.869	0.879

Table 10.6: PatchCore metrics for Real-IAD by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
bottle	0.857	0.554	0.832	0.814
cable	0.806	0.422	0.810	0.860
capsule	0.900	0.406	0.784	0.860
carpet	0.861	0.392	0.790	0.863
grid	0.802	0.260	0.807	0.849
hazelnut	0.738	0.338	0.713	0.732
leather	0.917	0.342	0.890	0.889
metal nut	0.876	0.424	0.707	0.746
pill	0.915	0.455	0.815	0.855
screw	0.802	0.088	0.544	0.730
tile	0.920	0.471	0.892	0.859
toothbrush	0.897	0.493	0.866	0.891
transistor	0.784	0.487	0.888	0.899
wood	0.930	0.389	0.921	0.849
zipper	0.880	0.337	0.741	0.835

Table 10.7: CFA metrics for MVTec by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
candle	0.692	0.158	0.789	0.799
capsules	0.615	0.098	0.576	0.774
cashew	0.699	0.369	0.727	0.777
chewinggum	0.797	0.137	0.811	0.838
fryum	0.725	0.261	0.705	0.631
macaroni1	0.581	0.021	0.693	0.735
macaroni2	0.519	0.007	0.636	0.731
pcb1	0.685	0.434	0.815	0.821
pcb2	0.732	0.100	0.829	0.816
pcb3	0.654	0.290	0.734	0.800
pcb4	0.760	0.257	0.808	0.735
pipe fryum	0.803	0.339	0.804	0.814

Table 10.8: CFA metrics for ViSA by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
audiojack	0.502	0.063	0.585	0.615
bottle cap	0.604	0.076	0.606	0.415
button battery	0.718	0.158	0.581	0.629
end cap	0.719	0.075	0.606	0.644
eraser	0.572	0.105	0.608	0.627
fire hood	0.516	0.098	0.589	0.526
mint	0.630	0.046	0.536	0.560
mounts	0.540	0.124	0.631	0.612
pcb	0.746	0.077	0.601	0.637
phone battery	0.585	0.136	0.616	0.624
plastic nut	0.482	0.080	0.575	0.618
plastic plug	0.537	0.060	0.579	0.622
porcelain doll	0.491	0.048	0.580	0.448
regulator	0.409	0.062	0.534	0.514
rolled strip base	0.816	0.124	0.701	0.705
sim card set	0.729	0.165	0.652	0.726
switch	0.681	0.151	0.615	0.636
tape	0.581	0.151	0.648	0.682
terminalblock	0.684	0.113	0.615	0.585
toothbrush	0.692	0.084	0.593	0.608
toy	0.724	0.053	0.572	0.533
toy brick	0.561	0.061	0.555	0.624
transistor1	0.728	0.134	0.632	0.570
u block	0.488	0.081	0.578	0.662
usb	0.622	0.084	0.587	0.619
usb adaptor	0.555	0.054	0.564	0.631
vcpill	0.562	0.173	0.609	0.667
wooden beads	0.607	0.108	0.617	0.665
woodstick	0.424	0.093	0.556	0.621
zipper	0.787	0.166	0.703	0.660

Table 10.9: CFA metrics for Real-IAD by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
bottle	0.884	0.440	0.830	0.940
cable	0.765	0.263	0.705	0.766
capsule	0.897	0.189	0.573	0.843
carpet	0.895	0.447	0.772	0.857
grid	0.700	0.100	0.611	0.549
hazelnut	0.826	0.427	0.887	0.968
leather	0.949	0.414	0.895	0.993
metal nut	0.891	0.489	0.780	0.881
pill	0.891	0.434	0.700	0.901
screw	0.773	0.026	0.417	0.901
tile	0.987	0.626	0.997	0.960
toothbrush	0.787	0.078	0.397	0.453
transistor	0.559	0.229	0.681	0.611
wood	0.900	0.387	0.948	0.913
zipper	0.870	0.328	0.751	0.917

Table 10.10: STFPM metrics for MVTec by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
candle	0.717	0.044	0.650	0.767
capsules	0.778	0.053	0.608	0.778
cashew	0.855	0.298	0.804	0.791
chewinggum	0.892	0.192	0.843	0.854
fryum	0.825	0.277	0.741	0.906
macaroni1	0.693	0.011	0.637	0.926
macaroni2	0.672	0.005	0.482	0.890
pcb1	0.717	0.179	0.670	0.886
pcb2	0.709	0.039	0.654	0.890
pcb3	0.690	0.172	0.664	0.908
pcb4	0.816	0.176	0.855	0.811
pipe fryum	0.872	0.355	0.823	0.891

Table 10.11: STFPM metrics for ViSA by category

category	Image-Level F1	Pixel-Level F1	Image-level AUROC	Pixel-level AUROC
audiojack	0.527	0.046	0.619	0.862
bottle cap	0.610	0.036	0.586	0.579
button battery	0.728	0.087	0.584	0.837
end cap	0.729	0.028	0.535	0.737
eraser	0.607	0.109	0.652	0.676
fire hood	0.558	0.071	0.649	0.730
mint	0.639	0.019	0.539	0.581
mounts	0.600	0.050	0.590	0.700
pcb	0.615	0.055	0.605	0.715
phone battery	0.620	0.060	0.610	0.725
plastic nut	0.610	0.040	0.595	0.705
plastic plug	0.605	0.045	0.600	0.710
porcelain doll	0.630	0.055	0.615	0.730
regulator	0.590	0.050	0.590	0.700
rolled strip base	0.605	0.040	0.600	0.710
sim card set	0.620	0.050	0.605	0.720
switch	0.615	0.045	0.600	0.710
tape	0.600	0.035	0.590	0.700
terminalblock	0.610	0.050	0.605	0.715
toothbrush	0.620	0.055	0.610	0.725
toy	0.630	0.060	0.615	0.730
toy brick	0.610	0.045	0.600	0.710
transistor1	0.620	0.050	0.605	0.720
u block	0.605	0.040	0.595	0.705
usb	0.615	0.050	0.605	0.715
usb adaptor	0.620	0.055	0.610	0.725
vcpill	0.600	0.040	0.595	0.705
wooden beads	0.610	0.045	0.600	0.710
woodstick	0.605	0.040	0.595	0.705
zipper	0.615	0.050	0.605	0.715

Table 10.12: STFPM metrics for Real-IAD by category

References

[1] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger, "MVTec AD — a comprehensive real-world dataset for unsupervised anomaly detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2019, pp. 9584–9592.

[2] C. Wang, W. Zhu, B.-B. Gao, Z. Gan, J. Zhang, Z. Gu, S. Qian, M. Chen, and L. Ma, "Real-iad: A real-world multi-view dataset for benchmarking versatile industrial anomaly detection," 2024. [Online]. Available: https://arxiv.org/abs/2403.12580

[3] Y. Zou, J. Jeong, L. Pemula, D. Zhang, and O. Dabeer, "Spot-the-difference self-supervised pre-training for anomaly detection and segmentation," 2022. [Online]. Available: https://arxiv.org/abs/2207.14315

[4] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler, "Towards total recall in industrial anomaly detection," 2022. [Online]. Available: https://arxiv.org/abs/2106.08265

[5] J. Johnson, "Faiss: The missing manual," https://github.com/facebookresearch/faiss/wiki/Faiss:-The-Missing-Manual, 2023, accessed: 2025-04-16.

[6] T. Defard, A. Setkov, A. Loesch, and R. Audigier, "Padim: a patch distribution modeling framework for anomaly detection and localization," 2020. [Online]. Available: https://arxiv.org/abs/2011.08785

[7] N. Cohen and Y. Hoshen, "Sub-image anomaly detection with deep pyramid correspondences," 2021. [Online]. Available: https://arxiv.org/abs/2005.02357

[8] G. Wang, S. Han, E. Ding, and D. Huang, "Student-teacher feature pyramid matching for anomaly detection," 2021. [Online]. Available: https://arxiv.org/abs/2103.04257

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1512.03385

[10] P. Bergmann, K. Batzner, M. Fauser, D. Sattlegger, and C. Steger, "The mvtec anomaly detection dataset: A comprehensive real-world dataset for unsupervised anomaly detection," *International Journal of Computer Vision*, vol. 129, no. 4, pp. 1038–1059, 2021.

[11] S. Lee, S. Lee, and B. C. Song, "Cfa: Coupled-hypersphere-based feature adaptation for target-oriented anomaly localization," *IEEE Access*, vol. 10, pp. 78 446–78 454, 2022.

[12] M. Barusco, F. Borsatti, D. D. Pezze, F. Paissan, E. Farella, and G. A. Susto, "Paste: Improving the efficiency of visual anomaly detection at the edge," 2024. [Online]. Available: https://arxiv.org/abs/2410.11591

[13] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 1, pp. 117–128, 2011.

[14] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 36, no. 4, pp. 744–755, 2013.

[15] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 931–938.

[16] T. Zhang, C. Du, and J. Wang, "Composite quantization for approximate nearest neighbor search," in *Proceedings of the 31st International Conference on Machine Learning (ICML)*. PMLR, 2014, pp. 838–846.

[17] C. Wang, W. Zhu, B.-B. Gao, Z. Gan, J. Zhang, Z. Gu, S. Qian, M. Chen, and L. Ma, "Real-iad: A real-world multi-view dataset for benchmarking versatile industrial anomaly detection," 2024. [Online]. Available: https://arxiv.org/abs/2403.12580

[18] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "Mobilenetv2: Inverted residuals and linear bottlenecks," 2019. [Online]. Available: https://arxiv.org/abs/1801.04381

[19] S. Zagoruyko and N. Komodakis, "Wide residual networks," 2017. [Online]. Available: https://arxiv.org/abs/1605.07146

[20] F. Paissan, A. Ancilotto, and E. Farella, "Phinets: a scalable backbone for low-power ai at the edge," 2021. [Online]. Available: https://arxiv.org/abs/2110.00337

[21] J. Lin, W.-M. Chen, Y. Lin, J. Cohn, C. Gan, and S. Han, "Mcunet: Tiny deep learning on iot devices," 2020. [Online]. Available: https://arxiv.org/abs/2007.10319

[22] Y. Li, Y. Chen, X. Dai, D. Chen, M. Liu, L. Yuan, Z. Liu, L. Zhang, and N. Vasconcelos, "Micronet: Improving image recognition with extremely low flops," 2021. [Online]. Available: https://arxiv.org/abs/2108.05894

[23] AMCO-UniPD, "MoViAD: Modular Visual Anomaly Detection," https://github.com/AMCO-UniPD/moviad, 2025, accessed: 2025-06-26.

Acknowledgments

This work marks the end of a long and intense chapter of my life, one that has been by far the most challenging, teaching and especially eye opening. Along the way, I've had the privilege of being surrounded by people who guided, supported, and inspired me, and to them, I owe my gratitude and admiration.

First and foremost, I want to thank the AMCO Research team at DEI. Working with you has been an incredible opportunity. Also thank you for letting me borrow an entire GPU for 2 months.

A heartfelt thank you to my supervisors, whose guidance and patience were instrumental throughout this journey. Your insight and feedback shaped both the work and the way I approached it.

To my family and friends thank you for being there in the moments when things felt overwhelming, for the late-night conversations, the words of encouragement, and for reminding me of who I am outside of all this.

Finally, I want to thank the University of Padua for being the place where this journey happened. It has been far from easy, but it has taught me more than I ever expected about myself, about research, and about perseverance.

This has been the most difficult and the most valuable experience of my life.

Once again, thank you.