POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Leveraging Student-Teacher learning framework for semi-supervised visual anomaly detection and segmentation

Laurea Magistrale in Computer Science & Engineering - Ingegneria Informatica

Author: Alessandro Polidori

Advisor: Prof. Giacomo Boracchi

Academic year: 2021-2022

1. Introduction

Visual anomaly detection is a computer vision task which consists in detecting if an image is anomalous and, if possible, segmenting the anomalous regions. Traditional computer vision techniques have been widely used for industrial inspection to automatically detect defects since the birth of the research field. In recent years, similarly to what has happened over the past decade for the other visual recognition tasks like classification, object detection and segmentation, deep-learning based approaches started to achieve very competitive results in anomaly detection. In principle, AD could be framed as a multi-class classification problem, where the nominal (non-defective) is one of the classes.

In this work we will face the semi-supervised version of the problem: fitting a model using only nominal sample images. This is crucial in industrial settings, where it's unlikely to have at your disposal many occurrences of a defect. Many papers enriched the literature dedicated to this subject, especially in the last few years, providing a wide set of techniques.

We decided to focus on one of the most promising macro-category: techniques based on the Student-Teacher learning framework. These methods rely on the discrepancy between the feature maps generated by a powerful Teacher model and the corresponding ones produced by a weak Student (or an ensemble of them). This discrepancy is used as an anomaly score. These innovative methods already achieved remarkable results on MVTec AD datasets, which are a standard de facto for visual anomaly detection, but there is still much room for improvement.

In this thesis we investigate the two best known Student-Teacher based methods ([3],[4]) and we propose two novel solutions based on them. In the first one, inspired by [3], we add an anomaly scoring function based on the Students' ensemble variance to exploit the disagreement between the Students. In the second one we simplify Student's architecture in order to create a bottleneck and distill only essential knowledge. We show that both the solutions obtain promising results in terms of Image-level AUROC and Dice score on the MVTec AD datasets.

2. Problem Formulation

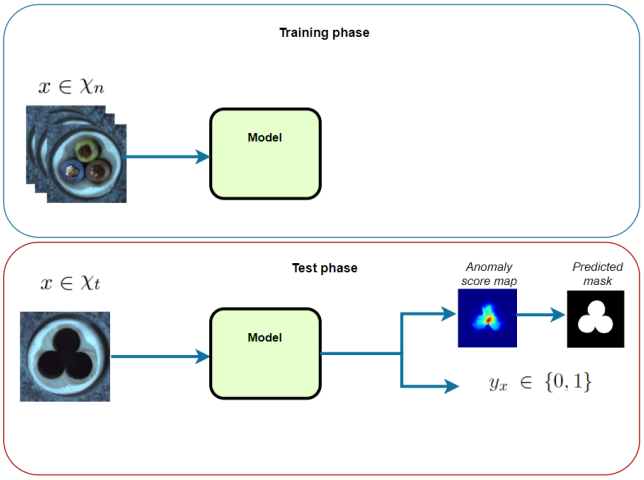

Figure 1: Inputs and outputs

Let us denote with $x \in \mathcal{X}_t$ a test image to be analyzed, defined over the pixel domain $\mathbb{Z}_p \in \mathbb{Z}^2$, with input size (h, w) and c channels. Given a channel, a pixel's intensity can range from 0 to $2^d - 1$, where d is the color depth. Visual anomaly detection can be defined as follows: given a test input image x, labeling it as anomalous (1) or normal (0) and, if anomalous, estimating the unknown anomaly mask $\Omega_x \in \mathbb{Z}_p \rightarrow \{0, 1\}$:

$$\Omega_x(i,j) = \begin{cases} 0 & \text{if pixel (i,j) is normal} \\ 1 & \text{if pixel (i,j) is anomalous} \end{cases}$$

Where Ω_x has size (h, w), equal to input image. So we will have two outputs: the image-level prediction $y_x \in \{0, 1\}$ and the predicted mask $\hat{\Omega}_x \in \mathbb{Z}_p \rightarrow \{0, 1\}$.

Given that we are considering a semi-supervised anomaly detection task, we assume that only images $x \in \mathcal{X}_n$ are used during training, where $\mathcal{X}_n$ is the set of non-defective (i.e. anomaly free) images. In our case, $\hat{\Omega}_x$ and y_x are obtained after choosing a threshold value. Our trained model $\mathcal{M}$ assigns an anomaly score to each pixel and an image-level score, which will then be converted to $\hat{\Omega}_x$ and y_x applying the threshold. Threshold can be chosen based, for example, on image-level F1-score maximization.

3. Background

3.1. Knowledge Distillation

Knowledge distillation was proposed as a compression technique for neural networks. It usually involves a knowledge transfer from a bigger pretrained network (Teacher) to an untrained lighter one (Student). Teacher and Student networks could have different architectures. The main objective is to ensure that the Student model imitates the Teacher. There are several possible approaches, in the next section we'll see the one exploited by our proposed solutions.

3.1.1 Feature-based distillation

An effective distillation strategy consist in transferring knowledge from the intermediate layers, to take into account multiple scales of feature representations. For every layer k we can establish a loss function:

$$\mathcal{L}_k = \|(f_t(\Phi_{t_k}(x)), f_s(\Phi_{s_k}(x)))\|_n \qquad (1)$$

where Φ_{t_k} and Φ_{s_k} are the k-th layer's feature maps of Teacher and Student networks respectively, f_t and f_s are the transformations needed in case they have different shapes and $\|\cdot\|_n$ is a distance. Note that other similarity functions can be used (e.g. cosine similarity).

3.2. Uninformed Students

Uninformed Students [3] is a semi-supervised technique that leverages the Student-Teacher framework to perform anomaly detection and segmentation. In this work the $(h \times w)$ images are fed in a patchwise manner. Meaning that for every pixel (m, n) of input image x, the sorrounding patch p is fed to the network. Firstly, a simple Teacher is trained on a large set of natural images (ImageNet) to match a pretrained network's output (last layer's feature maps):

$$\mathcal{L}_{teacher} = \|D(T(p)) - P(p)\|^2_{\ell 2} \qquad (2)$$

where D is a fully connected layer, referred as "decoder" in the paper, that is added to match the output dimension $(\mathbb{R}^d)$ of T with the pretrained network P.
Then, an ensemble of Students (same architecture as the Teacher) is trained to match the Teacher's output on nominal images of the target dataset:

$$\mathcal{L}_s = \frac{1}{wh} \sum_{(m,n)} \|\mu^s_{(m,n)} - (y^T_{(m,n)} - \mu) diag(\sigma^{-1})\|^2_{\ell 2} \tag{3}$$

where $\mu^s_{(m,n)}$ is the prediction made by the Student s for the pixel (m,n), $y^T_{(m,n)}$ is the Teacher's descriptor vector, while $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$ are the component-wise means and standard-deviations vectors computed on a set of non-defective validation images. The loss $\mathcal{L}_s$ is the scaled sum of all the pixels' prediction errors. The aim is to teach the Students to regress the output of the Teacher only on nominal patches. At test time, to evaluate the anomaly score in (m,n), the authors consider the regression error and, cleverly, the predictive uncertainty of the Students ensemble:

$$u_{(m,n)} = \frac{1}{M} \sum_{s=1}^{M} \|\mu^s_{(m,n)}\|^2_{\ell 2} - \|\mu_{(m,n)}\|^2_{\ell 2} \tag{4}$$

where M is the number of Students in the ensemble and $\mu_{(m,n)}$ is the ensemble's mean prediction. The assumption is that the group will give closer outputs on nominal patches. To manage multi-scale features, different networks with different patch-sizes are trained.

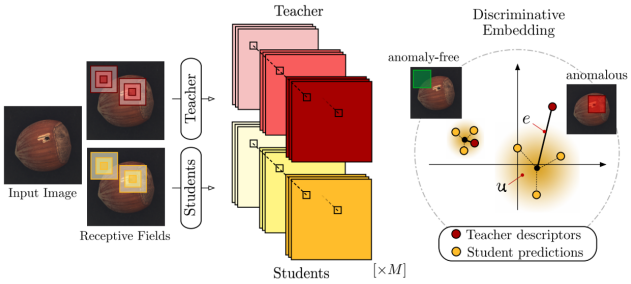

Figure 2: Uninformed Students schematic overview. During inference, ensemble's mean will yield a prediction error e and predictive uncertainty u ([3])

3.3. Student Teacher Feature Pyramid Matching

In [4], the authors solve the most critical issues of Uninformed Students and simplify the whole process at the same time. The first training phase gets bypassed using a pretrained ResNet as the Teacher network. A feature based distillation from multiple middle-layers (feature pyramid) removes the need for different patch-sizes, so a single network is capable to handle different scales of anomalies. Also, a single Student is used. Patch-wise processing is replaced by directly feeding the input image to the network. The Student is trained on nominal images to match Teacher's feature vectors setting as a distance metric the cosine similarity:

$$\mathcal{L}_k(i,j) = \frac{\|\hat{\Phi}_{t_k}(x)_{i,j} - \hat{\Phi}_{s_k}(x)_{i,j}\|^2_{\ell 2}}{2} \tag{5}$$

where $\Phi_{t_k}(x)_{i,j}$ and $\Phi_{s_k}(x)_{i,j}$ are the Teacher's and Student's k-th layer's feature vectors at position (i,j) in the feature maps. $\hat{\Phi}_{t_k}$ and $\hat{\Phi}_{s_k}$ are the ℓ_2 normalized versions. Note that the cosine similarity (5) can be expressed in terms of ℓ_2 distance when the vectors are normalized.

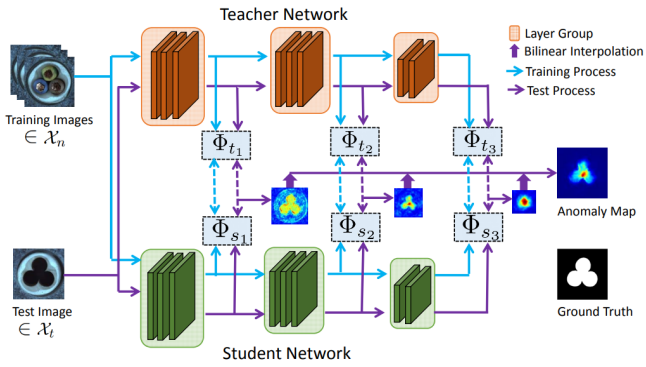

Figure 3: STFPM feature-based distillation ([4])

4. Proposed Solutions

4.1. Overview

Uninformed Students and STFPM both rely on the Student-Teacher framework and achieve remarkable results on the MVTec Anomaly Detection Datasets [2], a benchmark that became the de facto standard for semi-supervised visual anomaly detection in recent years. In this work we propose two new alternative methods based on these approaches. Given the simplified training scheme, we decided to use STFPM as a reference to build our methods. So the solutions introduced by STFPM and described in the previous chapter such as the multi-layer feature-based distillation and the whole-image processing will be employed in our proposed methodologies.

4.2. Solution 1: Feature Vectors Variance

In the first method, inspired by ([3]), we exploit an ensemble of Students. The aim is to equally train them on nominal images and then, at test time, use the variance of their feature vectors as an additional anomaly scoring function. The underlying assumption is that the ensemble's feature vectors will be less similar in correspondence of anomalous image regions. At first we separately train each Student to regress the Teacher's intermediate feature maps on anomaly-free images. As loss we employ the cosine similarity (5).

4.2.1 Students Uncertainty Measure

In Uninformed Students, a descriptor vector is obtained for each image patch (one patch for each pixel). The ensemble's predictive uncertainty with respect to these vectors' elements is measured for each pixel by (4). In our case, instead, we design a new uncertainty measure. To obtain it we first ℓ_2-normalize the Students' feature vectors:

$$\hat{\Phi}_{s_k}(x)_{i,j} = \frac{\Phi_{s_k}(x)_{i,j}}{\|\Phi_{s_k}(x)_{i,j}\|_{\ell 2}} \tag{6}$$

Where, given the student s, a vector $\Phi_{s_k}(x)_{i,j}$ is the concatenation of elements in position (i, j) along the k-th layer's feature maps. We need to measure the variance of these vectors' elements among the Students.
For each vector's element we compute standard-deviation amidst the ensemble, building a vector of the same length containing standard_deviation values and we denote it as $standard\_deviation_k(i, j)$.
The scalar uncertainty measure v_k is then:

$$v_k(i, j) = \|standard\_deviation_k(i, j)\|^2_{\ell 2} \tag{7}$$

4.2.2 Anomaly scoring function

To get the final anomaly score, we also need to compute the regression error between the ensemble's mean and the Teacher:

$$e_k(i, j) = \frac{\|\hat{\Phi}_{t_k}(x)_{i,j} - \overline{\hat{\Phi}}_{s_k}(x)_{i,j}\|^2_{\ell 2}}{2} \tag{8}$$

Where $\overline{\hat{\Phi}}_{s_k}(x)_{i,j}$ contains the average values of Students' (i, j) feature vectors.

Every k-th layer will be associated to an anomaly score map. This map will contain the sum between e_k and v_k in each (i, j) position.

4.3. Solution 2: Simple Student

The second method we propose is a variation of STFPM in which we simplify Student's architecture. That is not uncommon in a knowledge distillation setting, as the compression of the Student is usually the primary target. In STFPM, though, Student and Teacher models share the same architecture, since the authors are only interested in the anomaly detection task. What we argue is that the use of a simpler Student should increase the discrepancy between feature maps at test time when samples are anomalous. The rationale is that since Teacher's knowledge is distilled into a smaller architecture (less filters in convolutional layers), the distillation gets hindered. Therefore, Student network should learn to only represent essential nominal features needed to correctly reconstruct the feature maps.

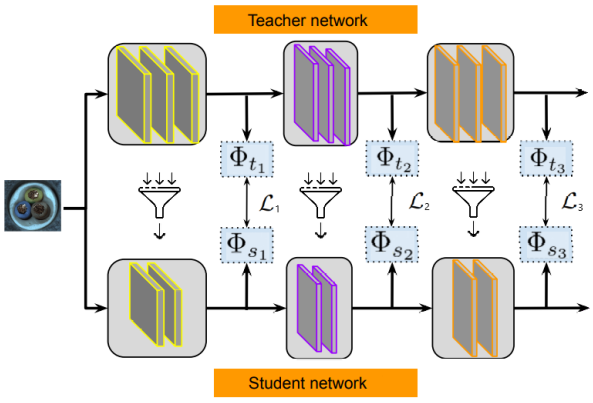

Figure 4: Simple Student feature-based distillation

In our implementation, we applied feature based distillation to layers 1,2,3 of a ResNet18. To get a more compact feature representation in each layer's latent space, we reduced the number of filters where possible. Note that, given that we are using ResNets, residual connections impose to have the same dimensions for input and output volumes in each basic block. Moreover, in layers 1,2,3 we also need to match Teacher's out-

puts dimensions (64, 128, 256) due to the feature based distillation.

5. Implementation Details

5.1. Anomaly maps upsampling

Note that, as in other approaches exploiting middle layers' activation maps, in our methods anomaly score maps need to be upsampled. That's a consequence of CNNs' pooling layers (e.g. maxpooling). Spatially reduced feature maps will produce anomaly score maps that are downsized with respect to input dimensions (h, w). Usually bilinear interpolation is applied to each k-th layer's map in order to reach input dimensions. After that, the resulting maps are combined through element-wise product to get scores that take into account different scales.

5.2. Setting Detection Threshold

In this work we are mostly interested in the anomaly detection aspect so we chose the threshold value that maximized the F1 score of the anomalous/nominal predictions $y_x \in \{0, 1\}$, disregarding the segmentation accuracy (for what concerns this choice). We always consider the image-level anomaly score as the maximum value of the combined anomaly score map described previously. Image-level F1 score optimization is not the only viable strategy. Another sensible choice would be to maximize the pixel-level F1 score, but that would imply that ground-truth masks are provided in the validation set, and that's not so common in industrial settings.

6. Experimental Results

In this section we show the results of our proposed methods and compare them with STFPM to appreciate the improvements brought by our contributions. All experiments were run on an Nvidia DGX system mounting Tesla V100 GPUs.

Figure 5: Example of segmentation output

6.1. Datasets

Tests have been executed on the MVTec AD datasets [2], a widespread benchmark composed of high resolution images divided into different subject categories (hazelnuts, bolts, carpets, pills...). Each of these categories contains a set of defect-free images to train the model. For testing, collections of defective images with hundreds of samples for every class of defect are included.

6.2. Metrics and Hyperparameters

In our tests we rely mainly on two metrics. AUROC (Area Under the Receiver Operating Characteristics) is very helpful for evaluating binary classification performance. In this case we are classifying anomalous (positive) and nominal (negative) samples. The ROC curve is plotted with True Positive Rate (TPR) on the y-axis against the False Positive Rate on the x-axis. The higher the area under the curve, the least amount of false positives are needed to obtain a certain TPR. Dice Score measures the overlap between segmentation and ground-truth mask. Even if we selected the threshold optimizing the image-level F1 score, it's still interesting to evaluate the pixel-level predictions of the model. Hyperparameters used in all the experiments are reported in the table below.

# Students	3
Batch Size	32
Input Size	224x224
Optimizer	SGD
Learning Rate	0.4
Momentum	0.9
Weight Decay	0.0001
feature extractor	ResNet18
Layer Blocks	1/2/3
Seed	42

6.3. Numerical Results

We report here the results obtained testing our methods on the MVtec AD Datasets. STFPM is used as a baseline. Best result in each category is bolded only for relevant improvements (> 0.005).

Image AUROC

	STFPM	Sol 1	Sol 2
Bottle	0.997	0.997	0.998
Cable	**0.962**	0.946	0.943
Capsule	0.450	**0.659**	0.544
Carpet	**0.977**	0.970	0.963
Hazelnut	0.974	0.971	**0.998**
Leather	1.0	1.0	0.997
Metalnut	0.954	**0.986**	0.558
Pill	0.516	**0.720**	0.448
Screw	0.748	**0.983**	0.956
Tile	0.983	**0.995**	0.988
Toothbrush	0.683	**0.763**	0.619
Transistor	**0.899**	0.874	0.814
Zipper	0.876	0.868	**0.912**
Wood	0.993	0.996	0.996

Dice Score

	STFPM	Sol 1	Sol 2
Bottle	0.661	0.663	0.555
Cable	0.513	**0.542**	0.488
Capsule	0.0133	**0.219**	0.0137
Carpet	0.641	0.642	0.594
Hazelnut	0.568	0.591	**0.610**
Leather	0.485	**0.507**	0.478
Metalnut	0.446	**0.605**	0.0104
Pill	0.0610	**0.132**	0.00157
Screw	0.120	**0.257**	0.238
Tile	0.520	**0.543**	0.519
Toothbrush	0.057	**0.126**	0.0643
Transistor	**0.611**	0.587	0.408
Zipper	0.446	0.448	0.417
Wood	0.573	0.559	0.570

7. Conclusions

For what concerns Feature Vectors Variance, the experimental results denote a clear improvement for many MVTec AD categories with respect to both Image AUROC and Dice Score. Notice how the Dice score improves in almost every category. A possible limitation concerns the computational resources needed by our Feature Vectors Variance solution. While STFPM only needs to keep two networks in memory, our technique requires an entire ensemble of Students, plus the Teacher. So the greater computational effort with respect to STFPM is undeniable. In some cases, though, a trade-off between resources and improved performances could be the most convenient choice. A trivial approach would be to use bigger networks like WideResNet-50, but that does not seem to improve the anomaly detection capabilities (see [1]). Therefore, in these situations our method offers a real advantage. Simple Student shows an improvement for less categories. That's likely a consequence of an excessive simplification of the Student's architecture. Still, results show that Simple Student can induce substantial improvements. A possible future development would be to better design the Student, maybe using 1×1 convolution in residual connections (to adapt basic block input/output tensors' sizes) and decoupling completely Student and Teacher architectures including fully connected layers to adapt the two networks' intermediate outputs dimensions (as it's done through the decoder in [3]). That would give more freedom to design a Student that is simpler but still able to represent nominal features well enough.

References

[1] Anomalib stfpm numerical results. `https://github.com/openvinotoolkit/anomalib/tree/main/anomalib/models/stfpm`. Accessed: 17-08-2022.

[2] Paul Bergmann, Kilian Batzner, Michael Fauser, David Sattlegger, and Carsten Steger. The mvtec anomaly detection dataset: A comprehensive real-world dataset for unsupervised anomaly detection. *International Journal of Computer Vision*, 129(4):1038–1059, Apr 2021.

[3] Paul Bergmann, Michael Fauser, David Sattlegger, and Carsten Steger. Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings. *CoRR*, abs/1911.02357, 2019.

[4] Guodong Wang, Shumin Han, Errui Ding, and Di Huang. Student-teacher feature pyramid matching for anomaly detection. 06 2022.

POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Leveraging Student-Teacher learning framework for semi-supervised visual anomaly detection and segmentation

Tesi di Laurea Magistrale in
Computer Science & Engineering - Ingegneria Informatica

Author: **Alessandro Polidori**

Student ID: 963429
Advisor: Prof. Giacomo Boracchi
Academic Year: 2021-2022

Abstract

Visual anomaly detection allows to detect and segment anomalous regions inside images. This task has attracted a lot of attention from the industry, as it provides valuable solutions in many domains (e.g. defect detection in manufacturing, medical images analysis, intelligent surveillance for security systems). A wide set of Machine Learning based techniques deal with the semi-supervised version of this task, in which the model is trained on nominal (anomaly-free) samples only. In the last few years, many Deep Learning techniques started to reach SOTA results thanks to CNNs' strong feature extraction capabilities. Student-Teacher learning based methods have already proven to be very promising. Given that these kinds of approaches for visual anomaly detection are relatively new, there still is a big margin for improvement. These methods rely on the feature maps' discrepancy between a powerful Teacher and a weak Student.

We propose two solutions that leverage Student-Teacher framework. In the first one, using an ensemble of Students, we define a direct uncertainty measure (Feature Vectors Variance) and include it to the anomaly score. In the second one, to increase feature maps discrepancy, we slim down Student's architecture in order to cause a bottleneck and distill from the Teacher only knowledge that is essential to the representation of nominal features.

At last, we show that both solutions obtain promising results in terms of Image-level AUROC and Dice score on the MVTec AD datasets ([4]).

Keywords: Anomaly Detection, Computer Vision, Deep Learning, Student Teacher, Knowledge Distillation

Sommario

La visual anomaly detection permette di rilevare e segmentare regioni anomale all'interno di immagini. Questo problema ha attratto molta attenzione poiché fornisce valide soluzioni in più settori (p.es. rilevamento di difetti nel manufatturiero, analisi di immagini nel settore medicale, sorveglianza intelligente per sistemi di sicurezza). Un ampio insieme di tecniche di apprendimento automatico trattano la versione semi-supervisionata di questo problema, nella quale il modello viene allenato usando solo immagini normali (prive di anomalie). Negli ultimi anni, molte tecniche basate sul Deep Learning hanno raggiunto risultati notevoli e rappresentano ora lo stato dell'arte grazie alla grande abilità delle CNN nell'estrarre features. I metodi basati sull'apprendimento Student-Teacher hanno già dimostrato di essere molto promettenti. Dato che queste tecniche per la visual anomaly detection sono relativamente recenti, c'è ancora un grande margine di miglioramento. Questi metodi fanno affidamento sulla discrepanza tra le feature maps della Teacher e quelle della Student.

Proponiamo due soluzioni che sfruttano il framework Student-Teacher. Nella prima, usando un ensemble di reti Student, definiamo una misura diretta dell'incertezza dell'ensemble (Feature Vectors Variance) e la sommiamo all'anomaly score. Nella seconda, per incrementare la discrepanza tra feature maps, snelliamo l'architettura della Student in modo da creare un "collo di bottiglia" e distillare dalla Teacher solo l'informazione essenziale alla rappresentazione di feature nominali.

Infine mostriamo che entrambe le soluzioni ottengono risultati promettenti in termini di image-level AUROC e Dice score sui dataset MVTec AD ([4]).

Parole chiave: Anomaly Detection, Computer Vision, Deep Learning, Student Teacher, Knowledge Distillation

Contents

1 | Introduction

1.1. Overview of visual anomaly detection

Anomaly detection, or novelty detection, is the task of deciding if a data sample belongs to the distribution of nominal samples or not. We humans are very good in determining if something deviates from the norm, even if we have seen a handful of normal examples. Our brain manages to fit models based on few data points brilliantly. Anomaly detection can be divided in three main types of tasks based on the labeling of the training data. In unsupervised AD, data points are not labeled, so the target is to fit a model of the nominal samples under the assumption that anomalies constitute a small portion of total training data. Supervised anomaly detection expects labeled data points. This is something which is at odds with typical anomaly detection use cases. In real scenarios we usually have at our disposal lots of nominal examples and few anomalous samples for each class of anomaly. In this thesis we will focus specifically on semi-supervised visual anomaly detection and segmentation. Semi-supervised means that our model can be trained on nominal samples only. This is the version of the problem that comes closest to real applications.

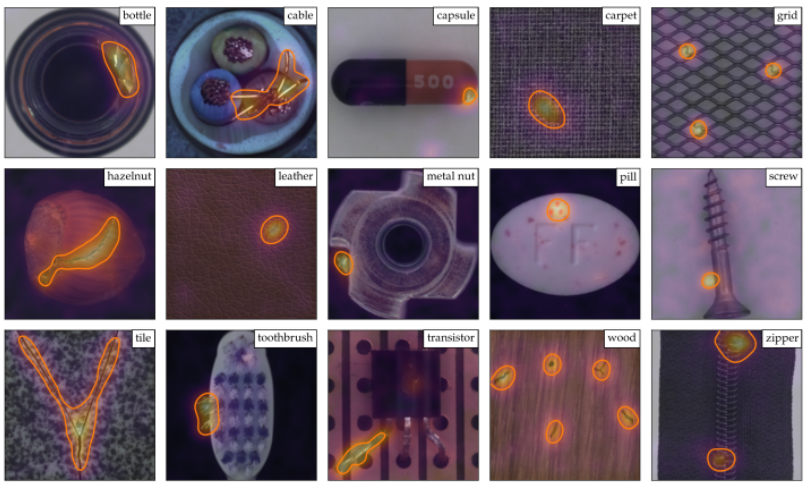

Figure 1.1: Examples from the MVTec benchmark datasets ([23])

Visual anomaly detection has attracted a lot of attention from different industries, as it provides valuable solutions to many problems like defect detection in manufacturing. In the field of medical image analysis it can be used to detect lesions, tumors and other pathological conditions. Intelligent surveillance systems obviously benefit from it also.

1.2. Overview of visual AD techniques

Traditional computer vision techniques have been widely used for industrial automated defect inspection since the birth of the research field thanks to their versatility and accuracy. In recent years, similarly to what has happened over the past decade for the other better-known vision tasks like classification, object detection and segmentation, deep-learning based anomaly detection approaches started to achieve very competitive results. Anomaly detection approaches, both classical and deep ones, have in common the fact that they are predominantly non-supervised (so they are unsupervised or semi-supervised). That comes from a physiological unsuitability of anomalous samples in pattern recognition domains. Anomalous data points bring several issues on the table. The major ones are: class imbalance (we already mentioned that in most cases we have few examples for some anomaly classes), concept drift (anomalies can evolve over time) and selection bias (for the training we label only anomalies we already managed to find). Best thing to do, so, is to find a way to model the distribution of normal features. We can then compute anomaly scores based on the distance from that distribution. We will see through the thesis that there exist other ways to asses the abnormality of a data sample. Moreover, what will be confirmed multiple times during this work, is the importance of feature representation capability. In visual anomaly detection we are dealing with complex high dimensional input data. As an example, a 32×32 rgb 8bit image contains $1024 \times 3(channels) \times 256(8\ bit\ depth)$ possible values. To filter information, classical Computer Vision approaches exploited handcrafted features based on color histograms, edges, HOGs and many other structures/extracted patterns. Deep Learning and CNNs (Convolutional Neural Networks) revolutionized almost every vision-related task (and not only), because they enabled to bypass one of the most critical aspect of Computer Vision pipelines: choice and extraction of features. This special kind of neural networks learns to extract the optimal "set of features" and, most importantly, it's able to exploit the hierarchical or latent structure that is often inherent to data through their multi-layered, distributed feature representations. That improves dramatically the model's ability to represent complex structures and patterns. As a matter of fact, virtually all recent visual anomaly detection approaches rely on CNNs in some way. Many papers enriched the literature about this subject, especially in the last few years, providing a wide set of techniques.

1.3. Overview of our work

We decided to focus on one of the most promising AD approach: techniques based on the Student-Teacher learning framework. These methods rely on the discrepancy between the feature maps generated by a powerful Teacher model and the ones outputted from a weak Student (or an ensemble of them). The discrepancy is used as an anomaly score. These innovative methods already achieved remarkable results on MVTec AD datasets, which are a de facto standard for visual anomaly detection, but there is still much room for improvement.

In this thesis we investigate the two best known Student-Teacher based methods and we propose two novel solutions based on them. In the first one, inspired by [3], we add an anomaly scoring function based on the Students' ensemble variance to exploit the disagreement between the Students. In the second one we simplify Student's architecture in order to create a bottleneck and distill only essential knowledge. We show that both the solutions obtain promising results in terms of Image-level AUROC and Dice score on the MVTec AD datasets.

1.4. Thesis structure

We organized the thesis as follows. Chapter 2 presents a formal definition of the task we are dealing with. In Chapter 3 we present what we consider the most relevant recent works in semi supervised visual anomaly detection. In Chapter 4 a theoretical background needed to fully grasp our proposed solutions. In Chapter 5 we present our two solutions. In Chapter 6 we discuss about the Datasets on which we tested our solutions, what metrics and hyperparameters have been used for the experiments and we report the experimental results. In Chapter 7 we report our conclusions and possible future developments directions.

2 | Problem Formulation

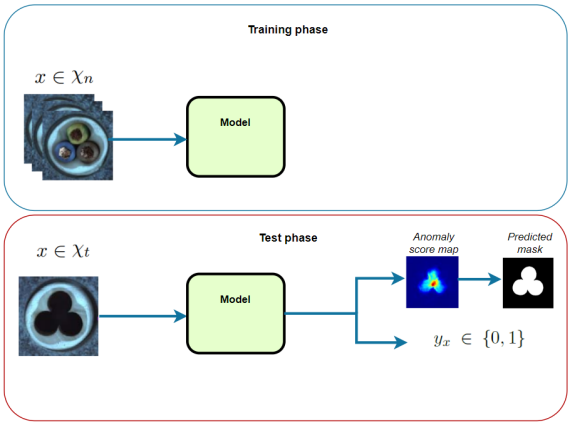

Figure 2.1: Inputs and outputs

Let us denote with $x \in \mathcal{X}_t$ a test image to be analyzed, defined over the pixel domain $\mathbb{Z}_p \in \mathbb{Z}^2$, with input size (h, w) and c channels. Given a channel, a pixel's intensity can range from 0 to $2^d - 1$, where d is the color depth. Visual anomaly detection can be defined as follows: given a test input image x, labeling it as anomalous (1) or normal (0) and, if anomalous, estimating the unknown anomaly mask $\Omega_x \in \mathbb{Z}_p \rightarrow \{0, 1\}$:

$$\Omega_x(i, j) = \begin{cases} 0 & \text{if pixel (i,j) is normal} \\ 1 & \text{if pixel (i,j) is anomalous} \end{cases}$$

Where Ω_x has size (h, w), equal to input image. So we will have two outputs: $y_x \in \{0, 1\}$ and the predicted mask $\hat{\Omega}_x \in \mathbb{Z}_p \rightarrow \{0, 1\}$. Given that we are considering a semi-supervised anomaly detection task, we assume that only images $x \in \mathcal{X}_n$ are used during training, where $\mathcal{X}_n$ is the set of non-defective (i.e. anomaly free) images. In our case, $\hat{\Omega}_x$ and y_x are obtained after choosing a threshold value. Our trained model $\mathcal{M}$ assigns an anomaly score to each pixel and an image-level score, which will then be converted to $\hat{\Omega}_x$ and y_x applying the threshold. Threshold can be chosen based, for example, on image-level F1-score maximization.

3 | Related Work

Semi-supervised anomaly detection literature is abundant and diverse. Several approaches have been proposed in the past, and they all rely on the ability to capture representations inherent to the nominal samples. In this work we will focus specifically on deep-learning based ones, as all the recent promising techniques rely on deep-convolutional neural networks in some way.

3.1. One Class Classification

A representative example of this transition from traditional computer vision to deep learning can be identified in Deep-SVDD [25], an approach built upon a one-class classification (OCC) technique: SVDD [29]. Kernel based methods like SVDD and OC-SVM [27] seek to isolate nominal handcrafted features in a higher dimensional space. This is done thanks to the well known "kernel trick", a convenient approach in which kernel functions are exploited. A kernel function k directly provides the inner product between two samples in a higher dimensional feature-space.

$$k(x_i, x_j) = \langle \phi(x_i), \phi(x_j) \rangle \tag{3.1}$$

The trick is that we can define objective functions that include only inner products and optimize them to fit the higher dimensional decision boundary without ever explicitly compute $\phi(x_i)$. Support Vector Data Description (SVDD) finds the smallest hypersphere enclosing most of the nominal samples (some outliers are left outside).

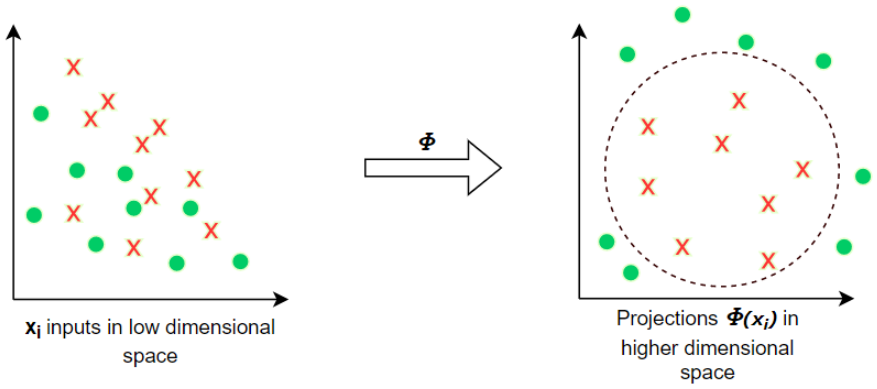

Figure 3.1: In higher dimensional space samples are linearly separable

OneClass-SVM (OC-SVM), instead, separates nominal inliers and outliers with an hyperplane of maximal distance from the origin in feature space. Note that the inner product $\langle \Phi(x_i), 0 \rangle = 0$, hence there is low similarity between the datapoints and the origin. The two methods are equivalent if the kernel is translation-invariant. Selecting the best kernels and handcrafted features for a specific application is not straightforward, especially for high dimensional input data like images. What Deep-SVDD does is to couple the one-class classification objective with the learning of useful feature representations through a deep CNN. So feature extraction becomes part of the learning process that aims to minimize 3.2.

$$\min_{w} \frac{1}{n} \sum_{i=1}^{n} \|\phi_w(x_i) - c\|^2 + R_w \tag{3.2}$$

Where x_i is a sample, $\phi_w(.)$ is the learned neural network transformation that centers hypersphere's center $\mathbf{c}$ and minimizes the distance of the representations $\phi_w(x_i)$ from it in feature space. The R_w term is a weight decay regularizer. At test time, for a given datapoint x_t (could be an image), an anomaly score s can be defined simply as

$$s(x_t) = \|\phi_w(x_t) - c\|^2 \tag{3.3}$$

Note that these classification-based methods do not provide a segmentation of the anomalous regions by default. Nevertheless, Image-level methods can always be applied in a patch-wise manner, providing a pixel-level segmentation but losing the image's semantic information.

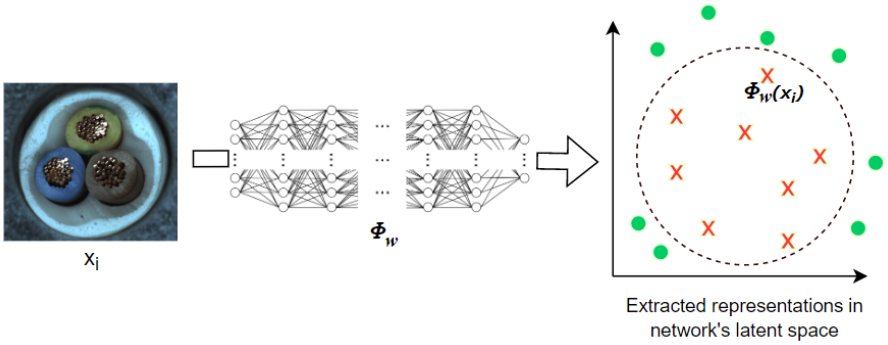

Figure 3.2: Deep-SVDD scheme

3.2. Generative models

A popular line of research is the one exploiting deep generative models like autoencoders, VAEs, GANs and Normalizing Flows. A model successfully trained on nominal samples will implicitly learn the density model describing the manifold of non-defective images. Some of the oldest deep approaches for visual anomaly detection rely on the reconstruction-error of these models. We denote with $\boldsymbol{\mathcal{X}}$ the input images' space, then $\phi_w : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{X}}$ is the model's transformation, usually composed by an encoding transformation $\phi_e : \boldsymbol{\mathcal{X}} \to \boldsymbol{\mathcal{Z}}$ and a decoding one $\phi_d : \boldsymbol{\mathcal{Z}} \to \boldsymbol{\mathcal{X}}$, where $\boldsymbol{\mathcal{Z}}$ is the latent lower dimensional space. Training is carried out on nominal input images $x \in \boldsymbol{\mathcal{X}}_n \subset \boldsymbol{\mathcal{X}}$ and the goal is to reconstruct them:

$$x \simeq \phi_d(\phi_e(x)) \tag{3.4}$$

The assumption is that if these models are trained on nominal samples, the anomalies will be reconstructed badly. The reconstruction error is the difference between input image x_t and the reconstructed one. This naive approach works in simple scenarios and many techniques nowadays also try to estimate the density model in latent space [34]. Recently, the need for stronger generative abilities led to the use of GANs. In case of autoencoders the encoding/decoding pipeline is already there. Unfortunately, GANs have no bijective mapping between image space and latent space by default. Many efforts have been made to "get access" to their feature representations. In this thesis we report two emblematic works ([26],[9]). In AnoGAN [26], the projection of a sample x from the image space $\boldsymbol{\mathcal{X}}$ to the latent space $\boldsymbol{\mathcal{Z}}$ is obtained solving an optimization problem:

$$\hat{z} = \min_z \|\mathcal{G}(z) - x\| + \lambda \log(1 - \mathcal{D}(\mathcal{G}(z))) \tag{3.5}$$

where $\mathcal{G}(\cdot)$ and $\mathcal{D}(\cdot)$ are the GAN's generator and discriminator, while z is the sample's projection in latent space. $\hat{z}$ is the projection that better approximates x when decoded by $\mathcal{G}(\cdot)$ and, at the same time, stays as close as possible to the manifold of normal data thanks to the second term of (3.5). At test time, for every sample x_t the optimization problem must be solved to obtain the anomaly score s:

$$s(x_t) = \|\mathcal{G}(\hat{z}_t) - x_t\| + \lambda \log(1 - \mathcal{D}(\mathcal{G}(\hat{z}_t))) \tag{3.6}$$

With BiGANs [9], instead, an explicit inverse mapping from data to latent space is learned in the form of an additional encoder $\mathcal{E}(\cdot)$.

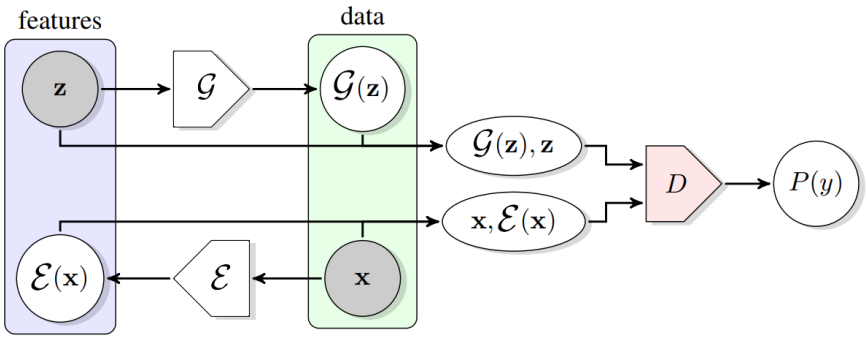

Figure 3.3: BiGAN scheme ([9])

The BiGAN loss function, as in vanilla GANs, is defined as a minimax objective:

$$\min_{\mathcal{G},\mathcal{E}} \max_{D} V(D, \mathcal{E}, \mathcal{G}) \tag{3.7}$$

$$\mathbb{E}_{x \sim p_{\mathbf{x}}(x)}[\log D(x, \mathcal{E}(x)] + \mathbb{E}_{z \sim p_{\mathbf{z}}(z)}[\log (1 - D(\mathcal{G}(z), z))] \tag{3.8}$$

Note that the discriminator takes as input the latent representations z and $\mathcal{E}(x)$ too. It is also optimized using the same alternating gradient based optimization of original GANs paper [11]. It can be shown (see [9]) that the global minimum of the training criterion is only achieved when $\mathcal{G} = \mathcal{E}^{-1}$. For what concerns the anomaly score, we have two main options: computing the likelihood of $p_{\mathbf{z}}(\mathcal{E}(x))$ or using directly the discriminator's posterior $D(x, \mathcal{E}(x))$, since the it should consider anomalous images as "fake" ones. Recently Normalizing Flows ([16],[24], [31]), another class of generative models, reached very competitive results on MvTecAD dataset [4]. Unlike GANs, they are designed to learn bijective transformations between data distributions so that the model can be used in both directions with ease. Flow based generative models provide exact latent-variable in-

ference and log-likelihood evaluation. In other probabilistic generative models like VAEs, one is able to infer only approximately the value of the latent variables that correspond to a datapoint because you can only optimize a lower bound of the exact log-likelihood of the data. Normalizing Flows, instead, consist in a series of differentiable and invertible functions that are composed in order to gradually map a complex distribution (like natural images manifolds) into a tractable one, like a Gaussian. Once we have trained a model with normalizing flows, we have a generative model that we can use to evaluate the likelihood for a new observation without having to explicitly access the distribution in image space thanks to change-of-variable formula. We won't delve deep into methematical details of this approach as it is out of this work's scope.

3.3. Self-supervision

Another popular class of approaches rely on self-supervised learning. As we have seen in the previous sections, most of the power of the anomaly detection relies on the feature representation quality, especially if we are considering complex datapoints like images. Self supervision is a way to obtain strong feature representations exploiting pre-text tasks. A pre-text task is a different problem than the one we aim to solve, and it's performed before the real task, or downstream task, in order to pre-train a model or, in general, to start solving the downstream task with better embeddings (feature representations) to begin with. What is peculiar about these pre-text tasks is that they can be solved without labels. That's because they manipulate the input datapoint in order to label it in an automatic way. Representative examples can be found in NLP, where self-supervision has been a godsend to obtain good embeddings of words. One possibile pre-text task in that case is the "center word prediction", in which you take a small chunk of words and, based on the center one, you make the model predict the others. Note that self-supervision usually needs extensive training on a large amount of data. Nowadays this is mostly a computational effort, because gathering unlabeled data became relatively easy thanks to the internet. What is interesting for us is that after the self supervised pre-training we have a model that is a very good feature extractor and it has been trained on our target dataset. This is a very good premise for us, because if we extract good feature vectors from the datapoints we can then treat them as random vectors and apply our favourite anomaly detection method (density based, distance based, etc...). In [10], dozens of geometric transformations τ_i are applied to the non-defective images during training.

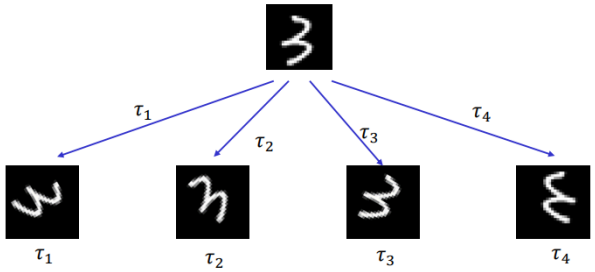

Figure 3.4: Set of rotations and horizontal/vertical flips of the input image

A Wide ResNet learns to discriminate the different transformations over the self-labeled dataset. At test time it is expected that the network will classify the anomalous transformed images with less confidence. At a practical level, the second last layer (the one before the softmax), is used as a feature vector describing the normal data. At this point you can model the nominal distribution with, for example, K-means clustering or density estimation methods like KDE and GDE. In CutPaste [20] instead, the pre-text task is a binary classification between nominal images and augmented ones. The augmentations proposed by the authors consist in a cut and paste of images' patches at random locations (optionally, they rotate or jitter pixel values in the patch). They then use GDE (Gaussian density estimation) to model the feature vectors and get log-likelihoods at test time. To segment the anomalies, beyond the always viable patch-wise approach (applying the method on image patches and upsampling the output to input dimensions), CutPaste authors also exploit visual explanation technique GradCam ([28]) on the image-level detector.

3.4. Feature modeling

Many other methods have at its core the same "two-phase" philosophy of self-supervised ones. They first model the high level features in some way, usually assisted by a deep convolutional network pre-trained on other huge datasets like ImageNet. Then they apply an anomaly detection method on the feature maps / embeddings. In [22], a pretrained ResNet is exploited to extract feature maps from nominal images' patches in order to embed them in a latent space. PCA is then used to reduce their dimensionality and their distribution is modeled with K-means. PaDiM [6], in a similar way, divides the input images in patches and extracts their embeddings with a pretrained network. After reducing the embeddings' dimensions it fits a MultiVariate Gaussian for every patch location using only nominal samples. At test time, the Mahalanobis distance between the

test patch and its respective estimated Gaussian is used as the anomaly score.

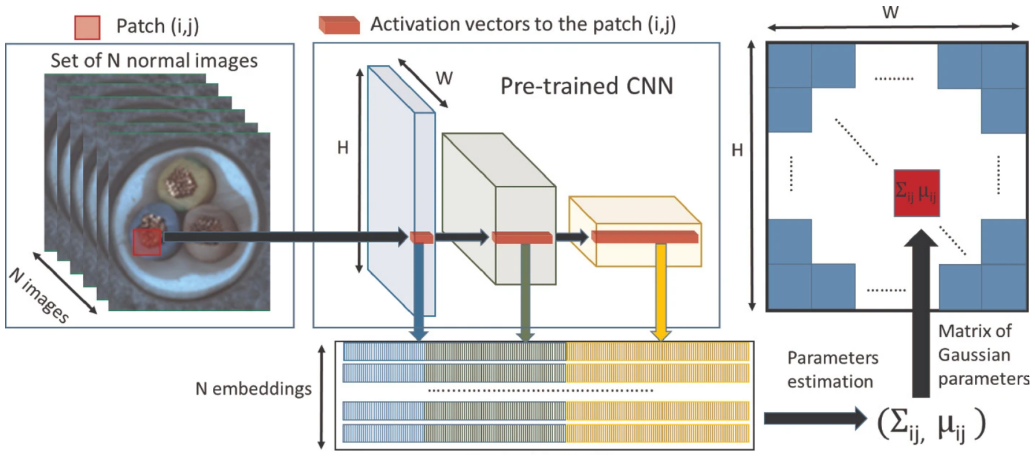

Figure 3.5: PaDim scheme ([6])

PatchCore [23], instead, stores nominal patch embeddings in a memory bank and computes the anomaly score as a weighted Euclidean distance from the closest embedding in latent space. Regarding the segmentation maps, these methods usually provide low resolution anomaly-score maps. They take the embeddings from mid-layers feature maps of CNNs, so the spatial dimensions reduce more and more along the network. Every embedding is associated to a specific input image's patch, and its dimensions are chosen in advance ([22]) or they are determined by the receptive field of the last layer considered ([6],[23]). Usually, bilinear interpolation is applied to the low resolution anomaly map to upsample it back to input dimensions.

3.5. Student-Teacher Framework

What we will investigate and extend in this thesis constitute a specific subclass of "Feature modeling" techniques based on the Student-Teacher learning framework. The idea is to distill the knowledge of a powerful (in terms of feature extraction capabilities) Teacher network into an untrained Student network using only nominal images. This is usually done minimizing a distance (Euclidean distance, Cosine similarity) between the Student's and the Teacher's feature maps. At test time, if the input is anomalous, we expect to observe more discrepancy between the two networks and we use that as an anomaly score. In [3] a double distillation scheme is put in place. At first, the Teacher network gets trained through self-supervised tasks and distillation from a pretrained ResNet. Then, an ensemble of Students learns to match their last layer's feature maps to the Teacher's one. Note that Student and Teacher architectures are the same. A mixture of Gaussians

can be obtained at each image pixel by equally weighting the ensemble's predictive distributions. At test time, predictive uncertainty of the Gaussian mixture is used as an additional anomaly score. This follows the idea that the Students generalize similarly in anomaly-free regions and differently in regions containing novel features, never seen during training. The way in which [3] deals with different scales of anomalies is simply using more Student-ensembles with different receptive fields. In addition, this method, to compute the pixel wise anomaly scores, works in a patch-wise manner. That means giving every possible patch as inputs to the networks, causing lots of overlaps (this problem is alleviated using Fast Dense Feature Extraction [2]). In STFPM [30], the authors solve these criticalities and simplify the training scheme using directly a pretrained ResNet as the Teacher network. The distillation from the Teacher into a single Student here is performed on multiple intermediate layers in order to take into account different scales with the same network.

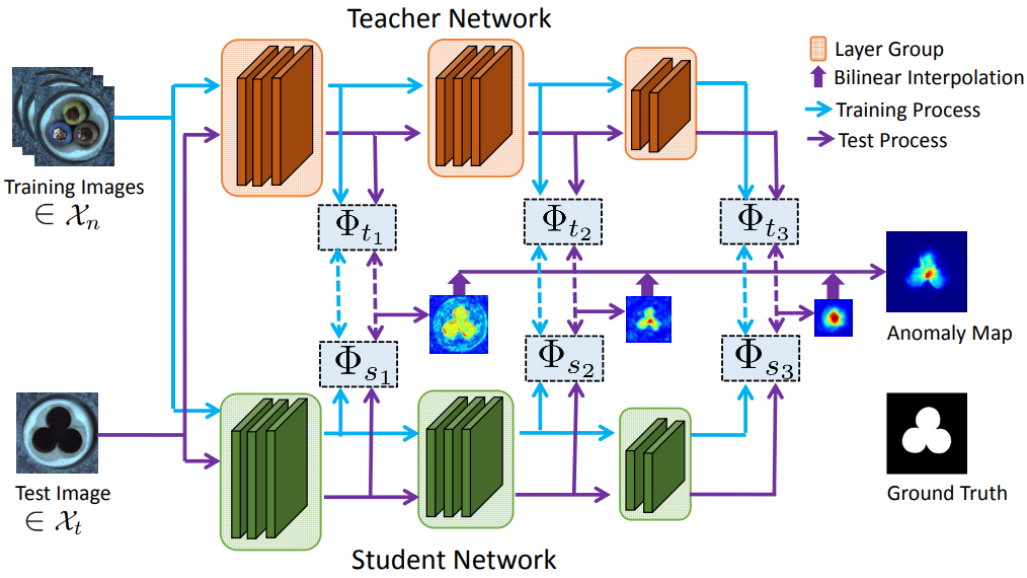

Figure 3.6: STFPM scheme ([30])

As these two works are the closest to our proposed solution, they will be discussed in more detail in the "Background" section. We want to also mention that recently, the authors of [7] reached very good results on the MVTec AD dataset reverting the conventional distillation scheme. In this new architecture, the Student network takes as input a latent representation (called one-class embedding) provided by the Teacher and an additional module (One Class Bottleneck Embedding), so it has to reproduce the Teacher's feature maps acting like a decoder. The authors state that the OCBE works as a bottleneck that retains only nominal features needed to reconstruct correctly the feature maps, amplifying

the discrepancy between Teacher and Student on anomalous inputs at test time.

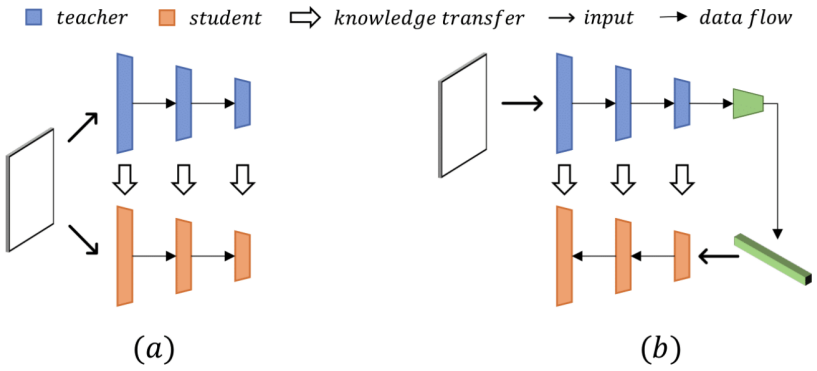

Figure 3.7: Conventional distillation scheme (a), Reverse distillation scheme (b) , OCBE in green ([7])

4 | Background

4.1. Feature extraction

Extracting distinctive features from images has always been a key step in computer vision pipelines. In fact, vision related tasks like object recognition, image alignment and stitching, classification, 3D reconstruction and many others need to compare images and find some kind of pattern associated to the semantics content. Finding these features is so important because little changes like orientation, lighting, subject position can dramatically change pixel values. So choosing patterns that are invariant with respect to these changes is a matter of great importance. In this section we describe the main concept behind classical and deep feature extraction methods. To grasp how important it is to filter out information, it's enough to consider that an image lives in a $h \times w \times 256 (\times 3$ also if rgb) space. If we consider a classification task, the most naive and inefficient approach would be to append every pixel value one after the other and directly feed this vector to a classifier (SVM, neural network, decision tree, etc...). Finding a good decision boundary in such a high dimensional space would be much harder and would require too many computing resources. The right approach, instead, is to extract patterns from the image and encode this information in a lower dimensional space. Feature extraction means to map the input data into a feature vector.

4.1.1. Classical feature extraction

One of the main problems of pre-CNNs feature extraction was to find the right set of features in order to extract relevant information from the input image. The best feature set would be the smallest one containing discriminating enough information for a specific task. A visual feature should be distinctive, invariant to irrelevant changes, efficiently comparable and, as we already said, relevant to the task. Listing all type of features appeared in computer vision research works in past decades would be unmanageable. In most cases, multiple kinds of features are extracted in order to collect all the needed relevant information for a specific task. Usually these feature descriptors are organized

as the elements of one single vector, commonly referred to as the feature vector. The dimensions of this vector define the feature space in which, following the classification example, the decision boundary will be searched by the classification algorithm. We report below the basic concepts behind the main classes of visual features to give an idea of how feature extraction works in the realm of classical computer vision.

Color/intensity

Pixels' intensity is obviously an important visual feature. In order to represent colors, models like RGB, CMYK, HSV combine intensities of multiple channels. One of the simplest method to retrieve a visual feature from an image (or from a region of interest) is to analyze how channels' intensities are distributed. Color histograms represent the global distribution of colors discretizing the intensity values in the image into a number of bins, and counting the number of image pixels in each bin.

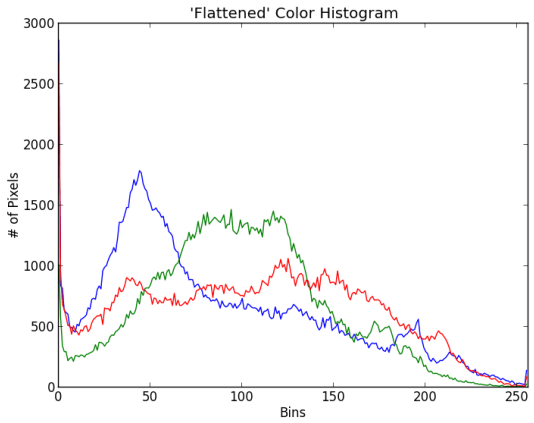

Figure 4.1: Color histogram

Color correlograms, instead, capture the spatial correlation of colors. Color moments (mean, standard deviation, skewness and higher ones) are scaling and rotation invariant. The mean is simply the average color in the image/patch and the standard deviation is, obviously, the square root of the variance of color distribution. Skewness and higher moments provide information about the shape of the color distribution. Since these features capture color and shape information, they are good candidates even in a context of small lighting changes.

Shape

Often object in images can be identified based on their shape. Numerous methods have been developed in order to describe shapes in the past. Object are found thanks to contours (abrupt changes in image intensities). Once you have the contour, various properties of it (perimeter, compactness, eccentricity, etc...) or of the region inside (area, geometric moments or structural properties such as convex hull, etc...) can be retrieved.

Texture

Another kind of global feature able to describe the semantic content would be a function of the spatial variation of intensity values within the image. These functions are the object of study of Texture Analysis. There are three general approaches:

- Statistical (describe texture via the statistical distribution of intensity values)
- Structural (describe texture as composed of texture primitives or textural elements named texels occurring repeatedly)
- Model-based (describe the texture estimating parameters of predefined models like autoregressive (AR), Markov, fractal models and many others)

Local features

For some tasks like object detection it's useful to localize discriminating patterns, in order to find correspondences in other images. These localized patterns are called "local features". The standard pipeline is divided in two main steps: keypoint detection and feature description.

Keypoint detection

Given an image, if we aim to find good location candidates we must rely on some kind of "relevance index" that measures how likely is that a distinctive feature is located in an area. We will report two famous keypoint detectors: Harris corner detector and DoG (Difference of Gaussians). Harris corner detector is based on the assumption that interesting locations are the ones in which the content is dissimilar to the neighboring ones'. Simply put, image locations containing corners will be much more recognizable with respect to homogeneous ones. Harris is an improved version of the simpler Moreavec detector, in which the average change of intensity resulting from small shifts of the respective patch

into various directions (x,y) is computed.

$$E_{(x,y)}(r,c) = \sum_{(u,v)\in U_{(r,c)}} w_{r,c}(u,v)[I(u,v) - I(u-x, v-y)]^2 \tag{4.1}$$

Where $U_{(r,c)}$ is the considered neighborhood centered in (r,c) and (x,y) are the displacement values. While Moreavec considers a fixed set of displacements, Harris detector computes a matrix M called "structural tensor", which is obtained from image's partial derivatives. If both eigenvalues of M have high values, then the image has a corner in that point.

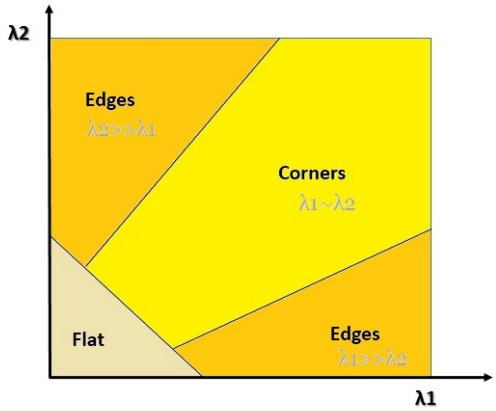

Figure 4.2: Structural tensor's λ-values

DoG is another popular keypoint detector. It's the one used in SIFT, a very successful local features extraction algorithm. It is based on an efficient approximation of the Laplacian function and carried out in two steps:

- Scale-space extrema detection: the image is convolved with multiple Gaussian kernels obtaining different outputs. Pairs of these ordered outputs are subtracted one from the other obtaining a Difference-of-Gaussians (DoG) pyramid. Stable keypoint locations are found searching for extrema in neighboring values and in neighbours from the above and below scales.
- Keypoint localization: at this point, to get a sub-pixel accuracy for the location, a quadric is fitted to the local sample points in order to determine the interpolated location of the extremum.

Feature description

After the detection, local features must be encoded in a descriptive way. The aim is to obtain a vector that describes the keypoint content, being less sensitive as possible to small changes in lighting, rotations, scale, etc... . An example is the Histogram of Oriented Gradients descriptor (HOG). In HOG, for each pixel in the neighborhood of the previously found keypoint, gradient orientations are computed and quantized into a fixed number of angular bins representing an orientation histogram. The descriptor is obtained concatenating the histogram entries in a vector and normalizing it. Note that HOG is not orientation-invariant.

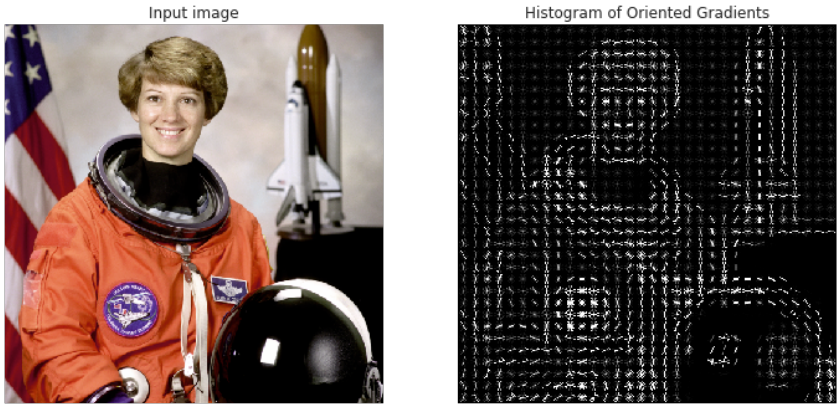

Figure 4.3: HOG visualization

Once the feature/descriptor vectors are computed, it's possible to get a measure of how similar they are using, for example, the $\ell 2$ norm in feature space or clustering algorithms.

4.1.2. Deep feature extraction

AlexNet (2012,[17]) popularized the use of deep convolutional neural networks for the classification of natural images and is considered by many a sort of turning point in Computer Vision. In this subsection we explain how CNNs work and why they revolutionized almost all computer vision subfields. We also describe in detail what are feature maps, as they constitute fundamental ingredients for all the methodologies investigated in this work.

Neural Networks

To better comprehend how a CNN works, we briefly discuss how a standard multi-layer feed-forward neural network process input data. Given a set of inputs x_i where $i \in$

$(0, 1, ..., D)$, every neuron in the next layer applies a non-linear activation function $h(\cdot)$ to a linear combination of all the previous layer output values (plus a bias).

$$a_j = \sum_{i=1}^{D} w_{ji} x_i + w_{j0} \tag{4.2}$$

$$z_j = h(a_j) \tag{4.3}$$

Where a_j is referred as activation, w_{ji} is the weight of the connection between input i and neuron j, while w_{j0} is the j-th neuron bias value. We won't dwell on the detailed explanation of how backpropagation algorithm, thanks to chain rule, computes efficiently the gradient of the loss function with respect to the network's weights in order to update them. For our purposes, the relevant information is that the standard NNs layers are densely connected, meaning that every neuron of a layer n receives a linear combination of all the $(n-1)$ layer's outputs.

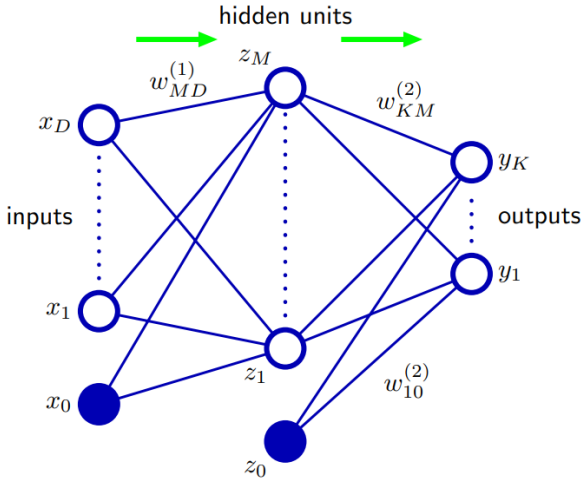

Figure 4.4: Two-layers neural network diagram

To process images, the standard approach before CNNs involved the computation of a feature vector with classical techniques (see 4.1.1). This vector was then fed to the network. In [19] LeCun et Al., working on handwritten characters recognition, coined the term "Convolutional Neural Networks" and demonstrated that using convolutions to process data outperformed all other techniques.

Convolution

In the Computer Vision context, convolution is a linear transformation in which a matrix of weights called "kernel" gets multiplied element-wise to a same-dimension patch of an input matrix. The output values of the multiplication are then summed. The single value returned by this sum encodes information from the processed input's area (and this encoding depends on kernel weights). This same operation is carried on in a sliding manner across the input matrix (which is bigger), in order to obtain another 2D matrix as an output.

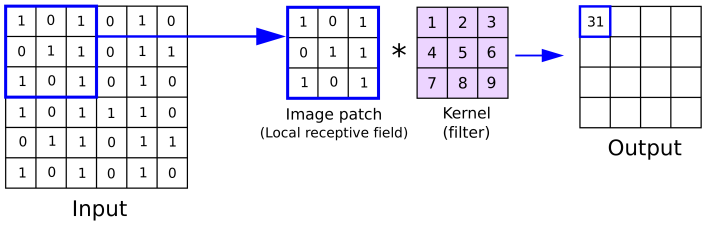

Figure 4.5: Convolution

Filters can extract different types of information from the input matrix. An illustrative example is the Sobel filter which, when convolved with an input image, returns the edges.

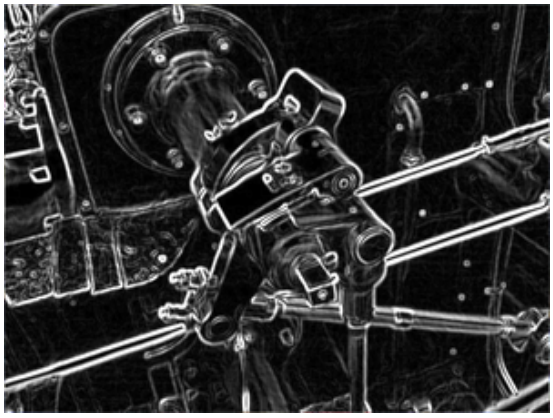

Figure 4.6: Sobel filter's output

Convolutional Layers

A convolutional layer is composed by a set of kernels that convolve with input volumes. When talking about CNNs the term volume is used to denote a multi-channel input/output. For example, an RGB image would be the first input volume of our network, and it would have 3 channels. The resulting output volume is called feature map

or activation map and is generated by stacking the activation maps of every filter along the depth dimension. Note that "feature maps" or "activation maps" usually refer to the non-linear activations of the convolutions' output volumes. Using this kind of layers solves two crucial problems we discussed earlier: curse of dimensionality for high dimensional datapoints and choice of features. We already explained that feeding the entire image directly to a multi-layer densely connected neural network wouldn't work as the search space for the classification/regression algorithm would be unmanageable. Convolutions (specifically, stacked convolutional layers) solve this problem reducing the connection and adding an "inductive bias" suited for image-data: closer values are correlated. How are the filters chosen? They aren't. The second critical aspect, the choice of features to extract, is solved integrating it to the learning pipeline. Filters' weights are learned, so the extraction of feature is optimized based on the loss function.

CNNs basic architecture

Replacing fully connected layers with convolutional ones is not the only ingredient for a functioning CNN. Another fundamental building block shared by all successful architectures is the subsampling layer. Subsampling layers usually apply a fixed (not learned) function on input maps in order to downscale them. For example, maxpooling layers take as input an $n \times m$ region from the input and return the maximum value. This operation, as the convolution, is applied in a sliding manner. Therefore, a 2×2 maxpooling layer will return maps with half the width and height. This is an essential feature of these kind of networks, as it promotes spatial invariance and a hierarchical composition of the semantics encoded by the feature maps at different stages. This means that lower layers will be responsible for low level image processing of the input image, while the top ones will manage high level structures, compositions and relationships between them. Other important ingredients in more modern CNNs (post-2012) are ReLU activation functions and batch normalization layers. Activation functions are the ones injecting non-linearity into the model. That allows the model to find complex decision boundaries and, in general, step up the representational power. In modern nets, functions like tanh and sigmoid got replaced by ReLU (Rectified Linear Unit) inside hidden layers. That's because ReLUs (and other variants of it) better control the gradient flow, making the model sparser and less prone to phenomena like vanishing gradient (ReLU's derivative is always equal to one for x>0).

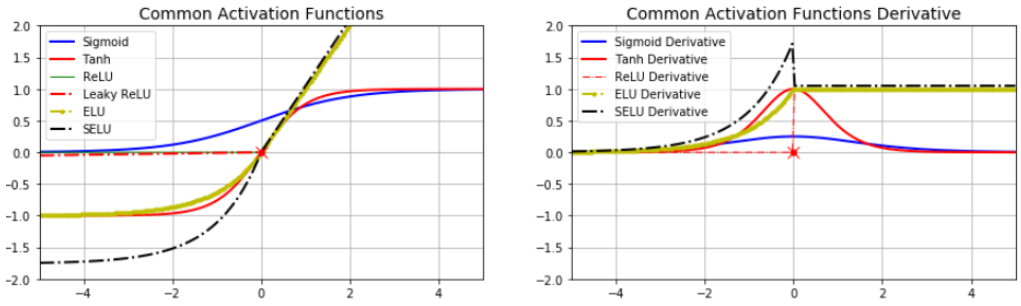

Figure 4.7: Activation functions and their derivative

Batch normalization layers, instead, favor learning standardizing the inputs to a layer for each mini-batch of datapoints. It's important to note that CNNs are usually considered the combination of two conceptually separated parts: the feature extractor and the fully connected classificator/regressor. That's because the first part of the network, the one composed by convolutional/ subsampling / normalizing layers stacked one after the other, has the role of extracting meaningful information and encoding it into a final feature vector, while the top of the net is basically a classical feed forward neural net that transforms the vector as an input.

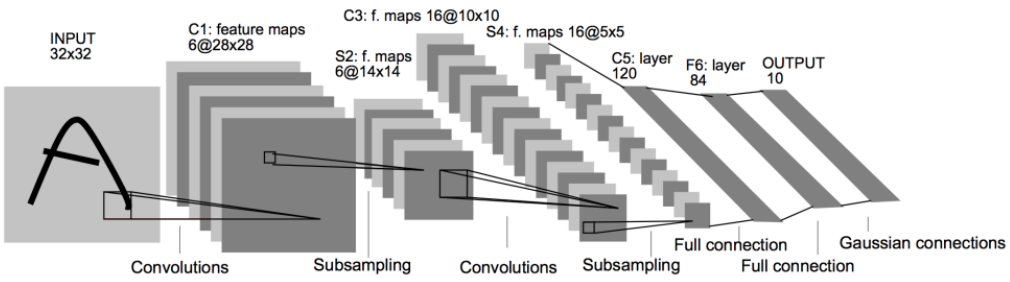

Figure 4.8: LeNet-5 architecture

ResNets

In this work we often refer to ResNets, a special subgroup of deep CNNs. They were first presented in [13]. Resnets are the answer to a strange problem. In standard CNNs accuracy improves stacking more layers one after the other but starts to degrade after a certain depth. This is counterintuitive, because if a shallower network is able to approximate a certain function, the deeper one should have no problem doing it. Moreover, the degradation happens for training accuracy too, so overfitting is not the problem here. The authors of [13] came to the conclusion that even if theoretically, deeper networks could learn identity mappings where it's necessary and "imitate" a shallower network, adding

skip connections between layers would simplify the identity mapping learning. That's because in resnets' residual blocks, the learned $\mathcal{F}(x)$ is something that gets added (a residual) to the untouched block's input x.

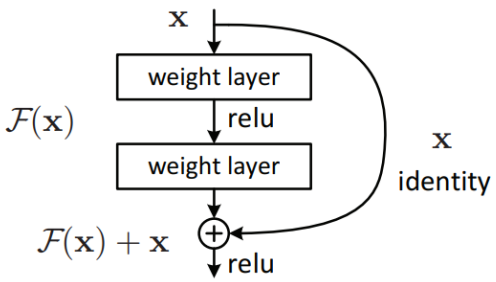

Figure 4.9: Residual block

This allowed to stack up a greater number of layers, improving significantly the network's performance. Nowadays residual connections are widely used in various state of the art architectures (Transformers, AlphaZero, AlphaFold and many others).

4.2. Knowledge Distillation

Knowledge distillation ([14],[5]) was born as a compression technique for neural networks. It usually involves a knowledge transfer from a bigger pretrained network to an untrained lighter one. If we want to classify a knowledge distillation scheme we have to consider three main aspects: knowledge type, distillation algorithm and Student/Teacher architectures ([12]).

4.2.1. Knowledge type

Response based

The simplest form of knowledge we can extract from a network resides in its outputs. In [14], Hinton et. Al popularized the term and proposed a response based distillation. Knowledge transfer was achieved forcing the Student to match Teacher's logits (outputs of the last fully connected layer of a deep model). Given that the pretrained network will probabily return values similar to the ground-truth (a very high value for the correct class and very low ones for the others), a temperature T parameter is included in order to soften the probability distribution over classes and improve distillation. The probability p_i of class i is calculated from the logits z as:

$$p_i = \frac{e^{\frac{z_i}{T}}}{\sum_j e^{\frac{z_j}{T}}} \tag{4.4}$$

The aim is to minimize the Kullback-Leibler Divergence $\mathcal{K}$ between softened probability distributions of the Teacher and Student models. If $T = 1$, (1) is equal to the softmax function. In [14] they also add the standard cross-entropy loss (so T would be set to 1) between Student's predictions and ground-truth labels. Therefore the total loss is:

$$\begin{aligned} \mathcal{L} &= \alpha * \mathcal{H}(y, \sigma(z_s, T = 1)) \\ &+\beta * \mathcal{K}(\sigma(z_t, T = \tau), \sigma(z_s, T = \tau)) \end{aligned} \tag{4.5}$$

where y is the ground truth label, $\mathcal{H}$ is the cross-entropy loss, $\mathcal{K}$ is the Kullback-Leibler Divergence loss, σ is the softmax function parameterized by the temperature T and α and β are coefficients. z_s and z_t are the logits of the Student and Teacher respectively.

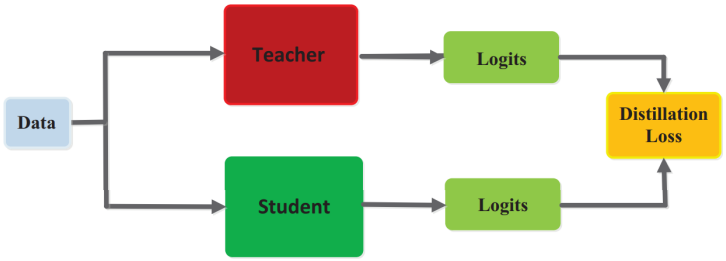

Figure 4.10: Response based distillation ([12])

Feature based

As we'll see in the next sections, a more effective distillation strategy is to transfer knowledge from the intermediate layers, in order to take into account multiple scales of feature representations. That's because semantic information is accumulated in a hierarchical way along the layers of deep neural networks, with low level features closer to the bottom layers and high level representations at the top.

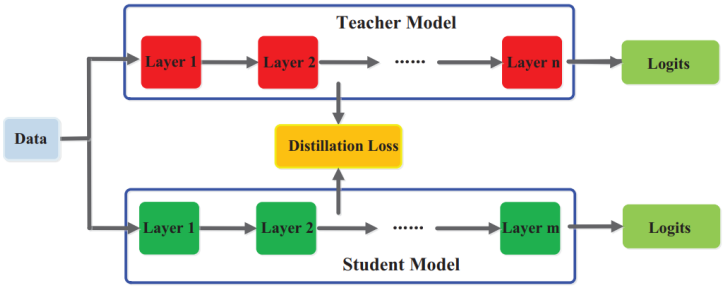

Figure 4.11: Feature based distillation ([12])

For every layer k we can establish a loss function:

$$\mathcal{L}_k = \|(f_t(\Phi_{t_k}(x)), f_s(\Phi_{s_k}(x)))\|_n \tag{4.6}$$

where Φ_{t_k} and Φ_{s_k} are the k-th layer's feature maps of Teacher and Student networks respectively, f_t and f_s are the transformations needed in case they have different shapes and $\|\cdot\|_n$ is a distance. Note that other similarity functions can be used (e.g. cosine similarity). This approach enforces closer feature representations along all the layers in the Student network.

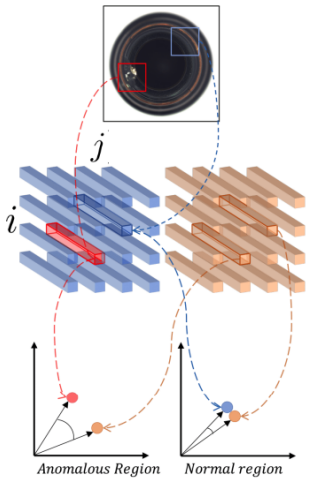

Figure 4.12: Feature Vectors similarity visualization ([7])

Relation based

In response and feature based distillation, knowledge is distilled from specific layers. In this last class of knowledge types, relationships between different feature maps are explored. The aim is to obtain a Student network in which the relationships between its layers are similar to the relationships existing between Teacher's ones. One method to

check the correlations between different layers is to calculate the Gram matrix (matrix of the inner products between features from two layers). So, for a pair of layers j and k, the loss function has the form:

$$\mathcal{L}_{j,k} = \|\Psi(\Phi_{t_k}(x), \Phi_{t_j}(x)), \\ \Psi(\Phi_{s_k}(x), \Phi_{s_j}(x)) \ \|_n \tag{4.7}$$

where Ψ is a similarity function between two layers' feature maps.

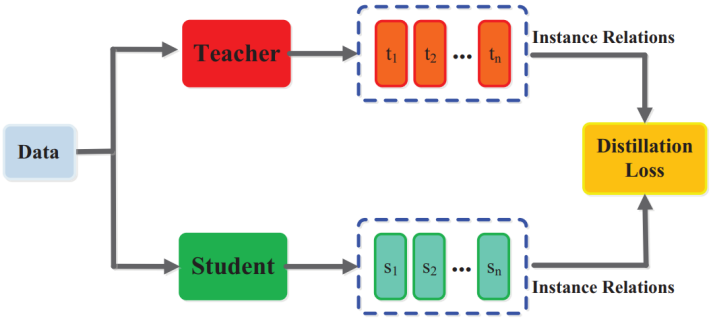

Figure 4.13: Relation based distillation ([12])

4.2.2. Distillation algorithm

Distillation can be carried on following different schemes. The three main ones are offline, online and self-distillation. The standard scheme is the offline one, and it's the one used by all the approaches presented in the next sections. It involves the use of a Teacher that is already trained. Given that we are investigating visual anomaly detection methods, the teacher network will probably be a Deep CNN already extensively trained on natural images on a big dataset such as ImageNet [8]. This ensures that the Teacher's feature maps are meaningful from a semantic standpoint. The big practical benefit is that nowadays very powerful pretrained networks are easy to get and use for other purposes, usually discarding the top (the classifier part) and keeping the feature extracting bottom layers. Despite the convenient decoupling between Student and Teacher of the offline schemes, some works showed that Student's performance can improve if student and teacher learn together. In online distillation ([33],[18]) Student and Teacher networks are interchangeable and get trained in a peer-teaching manner thanks to a loss function that combines the standard cross entropy loss from the ground truth labels with a Kullback Leibler Divergence based loss that measures the match of two networks' predictions. A special kind of online distillation is self-distillation, in which, given a single network, some layers act as a Teacher model for another set of layers. In [32], for example, knowledge from deeper layers is transferred to lower ones.

4.2.3. Student/Teacher architecture

The design of Student and Teacher model architectures affects heavily the quality of the knowledge transfer. Given that distillation has always been primarily a compression technique, there's a broad literature on methods addressing the simplification of the Student model. So the compressed model could simply have fewer layers and filters with respect to the Teacher as in Uninformed Students ([3]). Another possibility is to use a quantized version of the Teacher, meaning that the precision of the parameters is reduced, for example from 32 bit floats to 8 bit. Other more exotic approaches aim to optimize the student architecture. In [21], for example, the authors leverage a Neural Architecture Search (NAS), equipped with a KD-guided reward in order to find an optimal student architecture. We won't go into further detail about these methods because they are out of the thesis scope. Another option is to use the same architecture as the Teacher. That's what happens in [30], as the compression aspect of distillation is not necessary to the anomaly detection task.

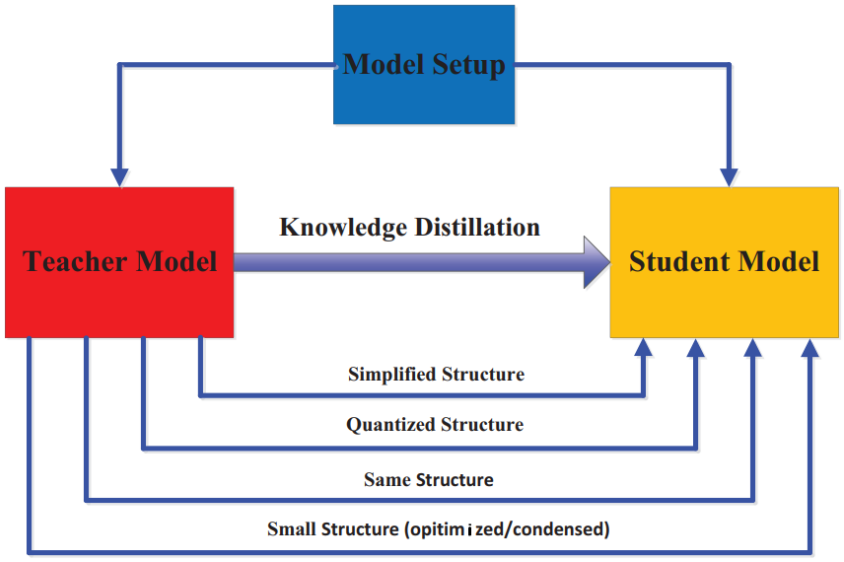

Figure 4.14: Student architectures scheme ([12])

4.3. Uninformed Students

We can now describe in detail the two Student-Teacher based semi-supervised visual anomaly detection methods which have inspired our work the most. Understanding how these two approaches tackle the anomaly detection task is essential to fully grasp how our proposed solutions leverage the student teacher learning framework.

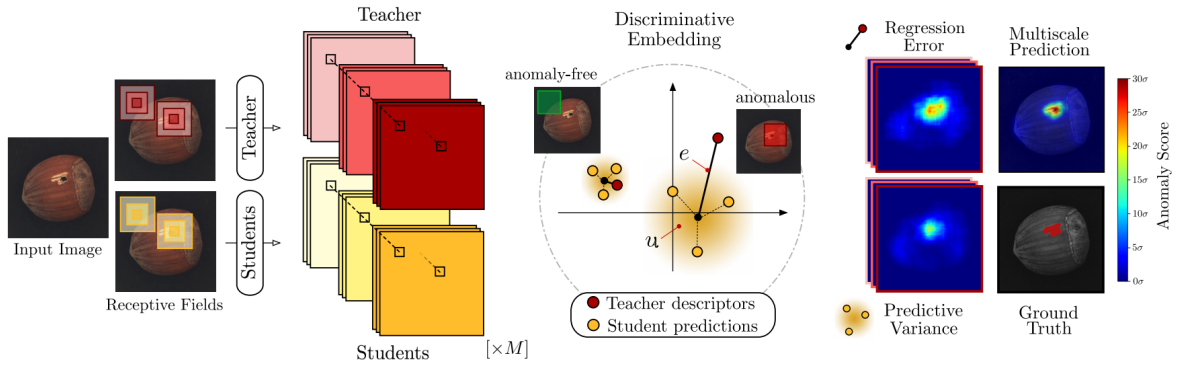

Figure 4.15: Uninformed Students schematic overview ([3])

In [3], the authors reduce the anomaly detection task to a regression one where an ensemble of Student networks are trained to mimic the output of a Teacher. Teacher and Students have all the same architecture (simple CNNs, as described in 4.1.2).

4.3.1. Patch-wise processing

An important premise regarding the modality in which the input images are processed must be maid. In Uninformed Students, images are not fed to the Student/Teacher models all at once. These CNN networks are trained to process single patches p and output a descriptor vector ($\mathbb{R}^d$), so there is no dense layer on top of the network. In order to have a dense estimation of the anomaly score in every image position (m, n), they make an inference for every possible patch. This obviously leads to a lot of overlapping between patches, jeopardizing performance. To alleviate this problem, the authors have benefited from FDFE ([2]), a clever method to optimize the computation of overlapping patches (as explained in [2] it becomes complicated in presence of a stride > 1 or pooling layers). Note that the patches have fixed size, so the authors suggest to use different instances (with different patch sizes) of the model to manage multi-scale detection.

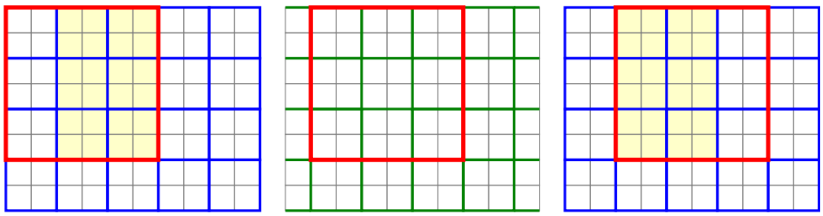

Figure 4.16: Patch at different image's positions. The first one (blue on the left) requires different 2×2 pooling than the second (green), but the same as the third (blue on the right) ([2])

4.3.2. Training

Training is divided in two main distillation (see 4.2) phases:

- Teacher training
- Students training

Teacher training

During first phase the goal is to obtain a powerful feature extractor as a Teacher. This is accomplished in two stages of extensive training on a large set of natural images (ImageNet, [8]). One of these stages consist in a knowledge distillation from a powerful pretrained deep CNN like a ResNet. Teacher is trained to match the pretrained network P's "output" (last layer's featuremap):

$$\mathcal{L}_{t_d} = \|D(T(p)) - P(p)\|_{\ell 2}^2 \tag{4.8}$$

where D is a fully connected layer, referred as "decoder" in the paper, that is added to match the output dimension of T with P ($\mathbb{R}^d$). This can be classified as a feature based distillation (see 4.2) in which only the last feature extracting layer is considered. The second stage is a self-supervised metric learning pre-text task that increases the feature extracting capabilities of the network. An example is triplet learning, in which a training image x is randomly cropped into p. p^+ and p^- are, respectively, a slightly translation of p and a patch taken from a completely different image. The metric learning loss would be:

$$\mathcal{L}_{t_m} = max(\|T(p) - T(p^+)\|^2 - \|T(p) - T(p^-)\|^2 + \alpha, 0) \tag{4.9}$$

Where α is a margin that keeps negative and positive embeddings far apart.

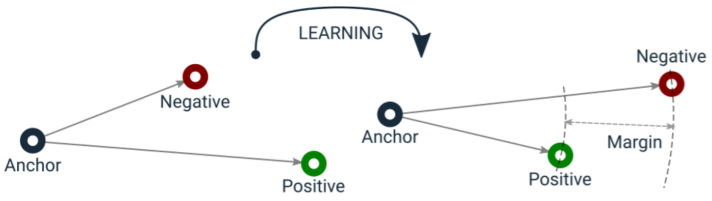

Figure 4.17: Triplet loss

The third and last loss function for the Teacher aims to minimize the correlation between

descriptors T(p) of each patch p within a minibatch.

$$\mathcal{L}_{t_c} = \sum_{m \neq n} c_{mn} \tag{4.10}$$

Where c_{mn} is the correlation between the output feature vectors given two different patches. The total loss is a combination of the three:

$$\mathcal{L}_t = \lambda_d \mathcal{L}_{t_d} + \lambda_m \mathcal{L}_{t_m} + \lambda_c \mathcal{L}_{t_c} \tag{4.11}$$

Students training

During the second training phase, Teacher's knowledge is distilled into a group of identical Students. This is done using only nominal patches (patches cropped from non defective images of the target dataset).

$$\mathcal{L}_s = \frac{1}{wh} \sum_{(m,n)} \| \mu^s_{(m,n)} - (y^T_{(m,n)} - \mu) diag(\sigma^{-1}) \|^2_{\ell 2} \tag{4.12}$$

where $\mu^s_{(m,n)}$ is the prediction made by the Student s for the pixel (m, n), $y^T_{(m,n)}$ is the Teacher's descriptor vector, while $\mu \in \mathbb{R}^d$ and $\sigma \in \mathbb{R}^d$ are the component-wise means and standard-deviations vectors computed on a set of non-defective validation images. Note that Students' output vectors are modeled as a Gaussian distribution $\mathcal{N}(y|\mu^s_{(m,n)}, cov)$ with constant covariance $cov \in \mathbb{R}$.

Anomaly scoring

In each pixel at position (m, n) the anomaly score is computed using the regression error $e_{(m,n)}$ of the Students ensemble's mean and the predictive uncertainty $u_{(m,n)}$ of Students' Gaussian Mixture (as defined in [15]).

$$u_{(m,n)} = \frac{1}{M} \sum_{s=1}^{M} \| \mu^s_{(m,n)} \|^2_{\ell 2} - \| \mu_{(m,n)} \|^2_{\ell 2} \tag{4.13}$$

They are then combined and normalized wrt a validation set of nominal patches:

$$score_{(m,n)} = \frac{u_{(m,n)} - u_\mu}{u_\sigma} + \frac{e_{(m,n)} - e_\mu}{e_\sigma} \tag{4.14}$$

4.4. Student Teacher Feature Pyramid Matching

In [30], many of the criticalities of Uninformed Students are bypassed thanks to a simplified scheme (see 3.6). What STFPM aims to improve:

- Feature extraction speed
- Teacher training reproducibility/simplicity
- Multi-scale support

4.4.1. Feature extraction speed

As we explained above, to obtain a pixel-wise anomaly score, Uninformed Students' networks process images one patch at a time, avoiding as many computations from overlapping regions as possible thanks to Fast Dense Feature Extraction ([2]), but slowing inevitably the whole process. In STFPM, entire images are directly fed to Students and Teacher networks, reducing to one the needed inferences. During the forward pass, input image x $(h \times w \times 3)$ is processed as we described in 4.1.2. The output volumes, even if down-sampled, will preserve spatial information. In next sections we will describe how STFPM manages to return a pixel-wise anomaly score.

4.4.2. Teacher training reproducibility/simplicity

In STFPM no Teacher pretraining is needed. This is a very critical aspect of Uninformed Students since the authors needed to extensively train the network on a very large dataset. It is well known that self supervised methods require a lot of computational power. This is completely eluded using as the Teacher a pretrained ResNet18. This obviously has a cost in terms of allocated memory, since you have to load two resnets on the vram. Note that in STFPM a single student is used.

4.4.3. Multi-scale support

In the previous section we mentioned that Uninformed Students managed multi-scale feature representation simply using different networks with smaller or bigger patch sizes. In STFPM, a feature based knowledge distillation from multiple intermediate layers is carried on. This ensures that one single Student learns to match Teacher's feature maps on more hierarchical levels. Therefore, only one instance of the model takes into account multiple scales of anomalies. Different combinations of ResNet18 layers are compared in [30].

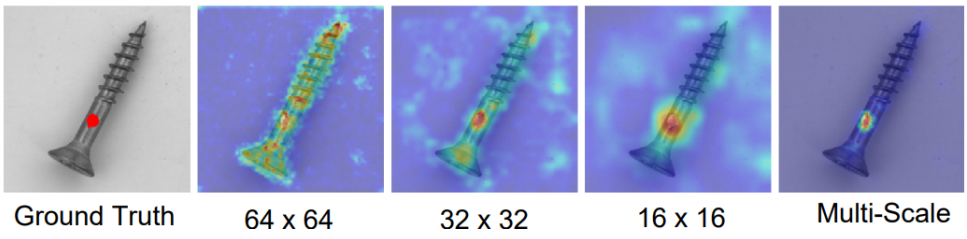

Figure 4.18: Anomaly maps from different intermediate layers ([30])

4.4.4. Training

Training is reduced to one single phase in which the Student learns to match Teacher's feature maps of different intermediate layers' outputs. Cosine similarity is chosen as a distance metric. Note that it can be easily expressed through the Euclidean distance if the vectors are $\ell 2$ normalized.

$$\mathcal{L}_k(i,j) = \frac{\|\hat{\Phi}_{t_k}(x)_{i,j} - \hat{\Phi}_{s_k}(x)_{i,j}\|^2_{\ell 2}}{2} \tag{4.15}$$

where $\Phi_{t_k}(x)_{i,j}$ and $\Phi_{s_k}(x)_{i,j}$ are the Teacher's and Student's k-th layer's featuremaps at position (i,j). $\hat{\Phi}_{t_k}$ and $\hat{\Phi}_{s_k}$ are the ℓ_2 normalized versions.

4.4.5. Anomaly scoring

Beside the binary prediction $y_x \in \{0,1\}$, even if we fed an entire image to the two networks, we still want to predict an anomaly mask $\hat{\Omega}$ of size equal to the input image (h,w). Given that, as we explained in section 4.1.2, CNNs reduce spatial dimensions along the way, anomaly maps obtained from each intermediate layer need be upsampled to input dimensions (h,w). Note that anomaly scores for each feature map's position (i,j) is computed through cosine similarity as in the loss (4.15). The final anomaly map will be the result of an element-wise product between these intermediate maps. As an image-level score, maximum value from this final map is taken.

4.5. ROC curve and AUC

After training, once we assigned an anomaly score to each image/pixel of a validation set, we still have to decide what threshold to use. Note that we could use two separate thresholds and treat the image-level anomaly detection problem separately from the seg-

mentation one. The choice of threshold can be based on a maximum percentage of false negatives we are willing to accept or on an optimization of some metric such as Precision, Recall or F1-score.

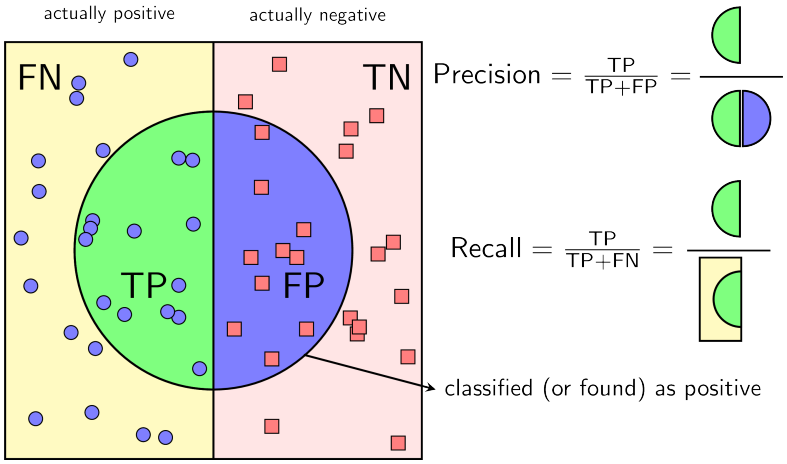

Figure 4.19: Visualization of Precision and Recall

To evaluate our model without choosing a specific threshold, but instead considering all possible ones, the ROC (Receiver Operating Characteristic) curve comes to our aid. The ROC curve is drawn plotting the false positives rate (on the x-axis) against the true positives rate (on the y-axis).

$$TPR = \frac{TP}{TP + FN}$$

$$FPR = \frac{FP}{FP + TN}$$

Needless to say, we want as few false positives as possible and a high TPR, so a steeper curve is what we aim to obtain from our model. Therefore, to compare different models, AUC (area under the curve) of the ROC until a specific FPR (usually 0.3) is often chosen as a metric.

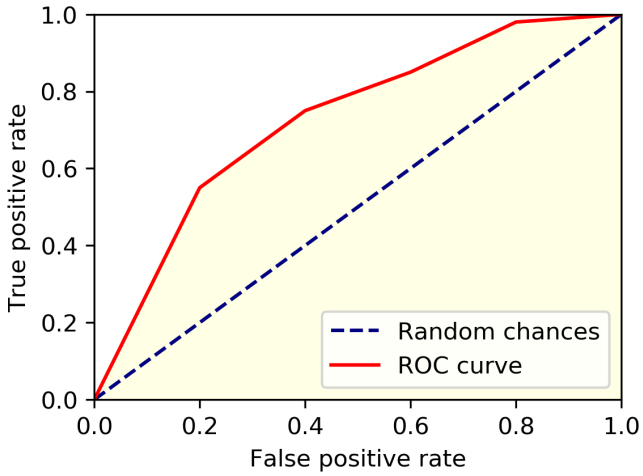

Figure 4.20: ROC curve graph

4.6. F1 score

In our proposed solutions, as we'll see, whenever we needed to choose a specific threshold to have a grasp of the performance in terms of false bads and false goods we relied on F1 score. In fact, we always took the threshold value that would maximize it. F1 score is a metric that takes into account both precision a recall. Specifically, it is the armonic mean between the two. An F1 score of 1 would reflect perfect precision and recall.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{4.16}$$

5 | Proposed Solutions

5.1. Overview

In this chapter we propose two methodologies for visual anomaly detection and segmentation. Once we analyzed the state of the art, we decided to investigate further how Student-Teacher framework could be leveraged in the context of visual anomaly detection. We identified two possible improvement directions and then we implemented and tested them. In both solutions we employ the simplified STFPM's training scheme: input image x is directly fed to the networks and feature-based knowledge (see 4.11) is distilled from a pretrained Teacher on multiple intermediate feature maps.

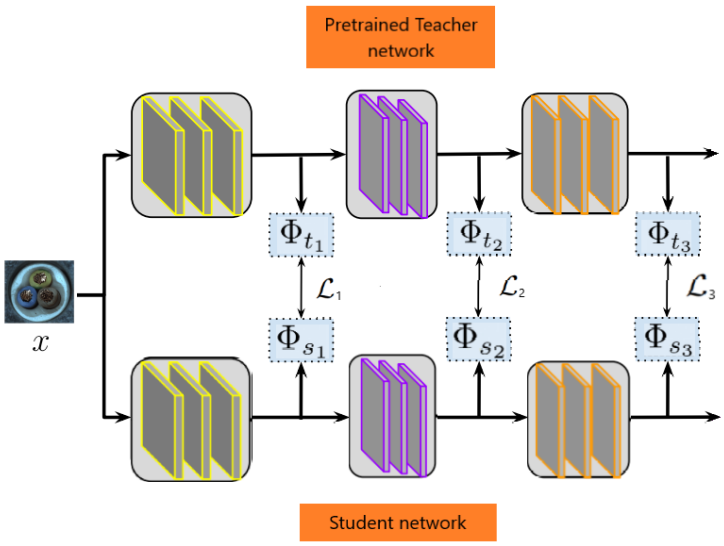

Figure 5.1: Basic training scheme of our methods

We present them in the next two sections. We then report common implementation details and, in the next chapter, we show experimental results.

5.2. Feature Vectors Variance

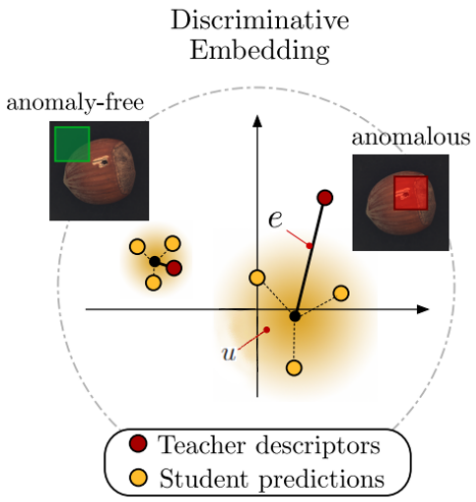

Figure 5.2: Visualization of Students' uncertainty on anomalous inputs

5.2.1. Rationale

In [3], the authors showed that adding the ensemble's predictive uncertainty to the total anomaly score improved the detection performance. The underlying assumption, that was then corroborated by the experimental results is that, given an ensemble trained on nominal patches only, the Students' predictions associated to defective patches have a higher predictive uncertainty. This is an interesting approach, as it leverages not only the regression error between Students' mean and Teacher but also the disagreement between Students. We adapted this idea to the simplified STFPM's scheme, designing a new expression for the ensemble's uncertainty.

5.2.2. Architecture

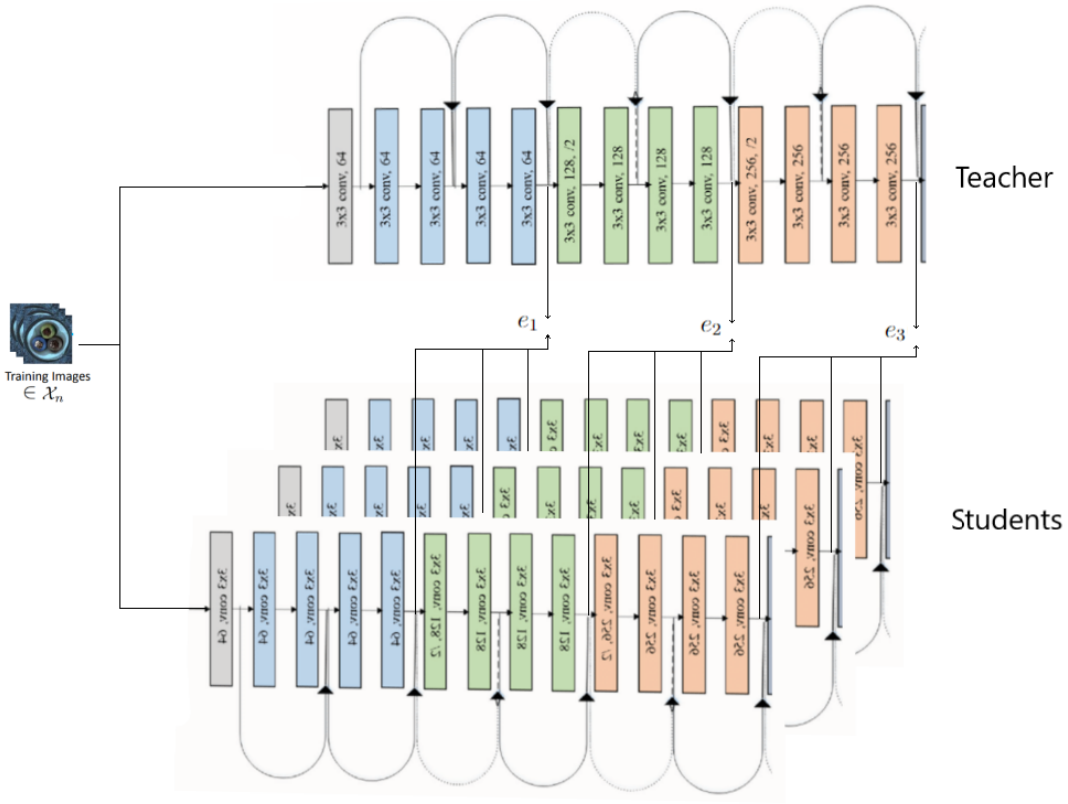

Figure 5.3: Feature Vectors Variance training scheme with layers 1 (blue), 2 (green) and 3 (red)

In Feature Vectors Variance, the model is composed by a pretrained ResNet18 and an ensemble of identical ResNet18 Students. Only the chosen set of intermediate layers is required, so we can avoid to load in memory the remaining part of the network. Notice that, in case we choose latter or non subsequent layers (2,3,4 or 1,2,4), Students will still need the missing ones during training.

Training time

Given that we use a pretrained Teacher, the training phase of our model coincides with the training of the Students. Note that only normal (anomaly-free) images are used. Each Student is initialized randomly following good practices (e.g. Xavier Glorot, He etc...) and the training is carried on separately. In our implementation they are trained with the same order of nominal images. The underlying assumption behind our choice

is that the Students should behave as much similarly as possible on normal images, and the Students' divergences caused by the different initialization is already enough to have disagreement (higher variance) on Feature Vectors associated to anomalous regions. The other possibility would be to train one Student at a time, shuffling the dataset in order to decouple as much as possible the training of the different Students.

$$L_{k(i,j)}(x) = \frac{\|\hat{\Phi}_{t_k}(x)_{i,j} - \hat{\Phi}_{s_k}(x)_{i,j}\|^2_{\ell 2}}{2} \tag{5.1}$$

As loss function, at each k-th layer's output we consider the cosine similarity between Teacher's normalized feature maps $\hat{\Phi}_{t_k}(x)$ and the Student's ones $\hat{\Phi}_{s_k}(x)$.

Based on how many ResNet's layers we want to take into account, we will have multiple cosine distances. To get the final loss, we stack them togheter, and we sum.

Test time

In Uninformed Students, a descriptor vector is obtained for each image patch (one patch for each pixel). The ensemble's predictive uncertainty with respect to these vectors' elements is measured for each pixel by 4.13. In our case, instead, for each k-th layer we design a new uncertainty measure. To obtain it we first $\ell 2$-normalize the Students' Feature Vectors, we compute their standard-deviations vectors and, as last thing, we take their squared $\ell 2$ norm. So for the layer k and the Student s we have:

$$\hat{\Phi}_{s_k}(x)_{i,j} = \frac{\Phi_{s_k}(x)_{i,j}}{\|\Phi_{s_k}(x)_{i,j}\|_{\ell 2}} \tag{5.2}$$

We recall that (i, j) is the feature maps' spatial position.

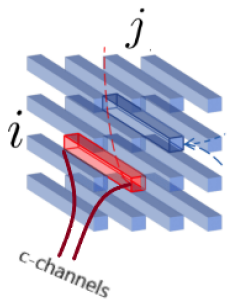

Figure 5.4: Visualization of a Student's Feature Vector in red

For each position (i, j) we take the corresponding vector containing each feature map's

(i, j) value. So if in a specific layer we have feature maps with size (l, m) and c channels, we will obtain $l \times m$ vectors of length c containing in each element the standard_deviation between the Students' feature maps' values for that channel and (i, j) position.

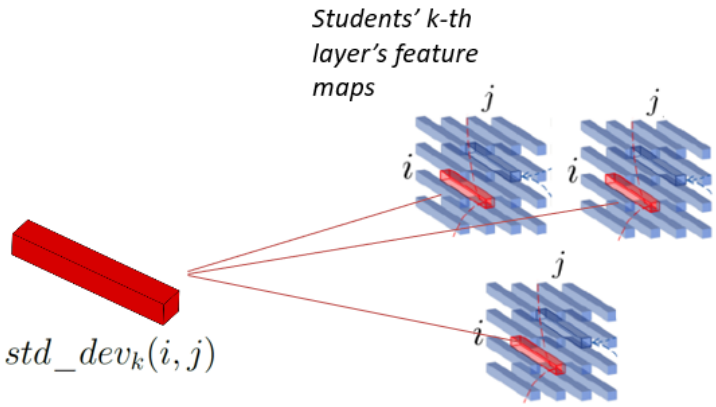

Figure 5.5: Visualization of (i, j) standard_deviation vector

The resulting vector containing standard_deviation values is $standard\_deviation_k(i, j)$. The scalar uncertainty measure v_k is then:

$$v_k(i, j) = \|standard\_deviation_k(i, j)\|^2_{\ell 2} \tag{5.3}$$

To compute the anomaly score we also need to measure the ensemble's regression error with respect to the Teacher.

$$e_{k(i,j)}(x) = \frac{\|\hat{\Phi}_{t_k}(x)_{i,j} - \overline{\hat{\Phi}}_{s_k}(x)_{i,j}\|^2_{\ell 2}}{2} \tag{5.4}$$

$\overline{\hat{\Phi}}_{s_k}(x)_{i,j}$ contains the ensemble's normalized (i, j) Feature Vector's average values for the k-th layer. Every (i, j) layer-map's position gets as anomaly score the sum between e_k and v_k.

5.3. Simple Student

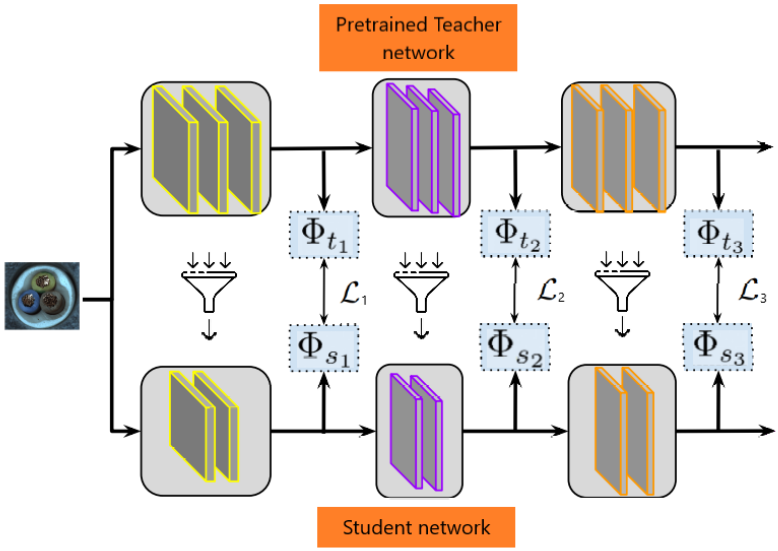

Figure 5.6: Visualization of feature based knowledge distillation from Teacher to Simple Student

5.3.1. Rationale

The second method is based on the assumption that simplifying the Student will force it to learn only the most essential features from the nominal training samples. In STFPM, Student and Teacher architectures are exactly the same. That's because, even if the Student-Teacher framework was initially designed as a network compression method, having a smaller Student is not required for the anomaly detection task. In Anomaly Detection via Reverse Distillation from One-Class Embedding ([7]), the authors propose a Teacher-Student model in which the Student acts as a "decoder", reconstructing the feature maps in reverse, starting from the latent space. A one-class bottleneck embedding (OCBE) is interposed between the two networks. The obtained compact embedding effectively preserves essential information on normal patterns, but abandons anomaly features.

Inspired by [7], we bottleneck distillation between Teacher and Student. We achieve it reducing convolutional filters in correspondence of the layers involved in the feature-based distillation. This should increase the feature maps discrepancy between Student and Teacher in correspondence of anomalous regions at test time.

5.3.2. Architecture

In order to bottleneck the distillation from the Teacher's intermediate layers we needed to slim down Student's architecture. The aim is to force the Student to have more compact latent representations in correspondence of what in ResNets is denoted as "layer". These ResNets' "layers" are actually a collection of basic blocks. Each of these blocks contains an arbitrary number of convolutional layers, always followed by BatchNormalization layers. In our implementation, we applied feature based distillation to layers 1,2,3 of a ResNet18. To get a more compact feature representation in each layer's latent space, we reduced the number of filters where possible.

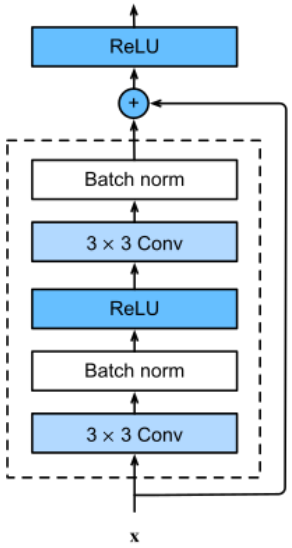

Figure 5.7: ResNet18 basic block scheme

Note that, given that we are using ResNets, residual connections impose to have the same dimensions for input and output volumes in each BasicBlock. Moreover, in layers 1,2,3 we also need to match Teacher's outputs dimensions (64, 128, 256) due to the feature based distillation. This restrictions could be bypassed decoupling the two networks' architectures interposing dense layers (fully connected) between Student and Teacher layers 1,2,3 in order to match the output dimensions. Another possible way to "relax" these constraints is to use a 1×1 convolution as the residual connection, in order to adapt the two tensors' dimensions. Still, we decided to keep the simplest possible configuration of layers as a first proof of concept.

In Appendix A we report the two networks' summaries.

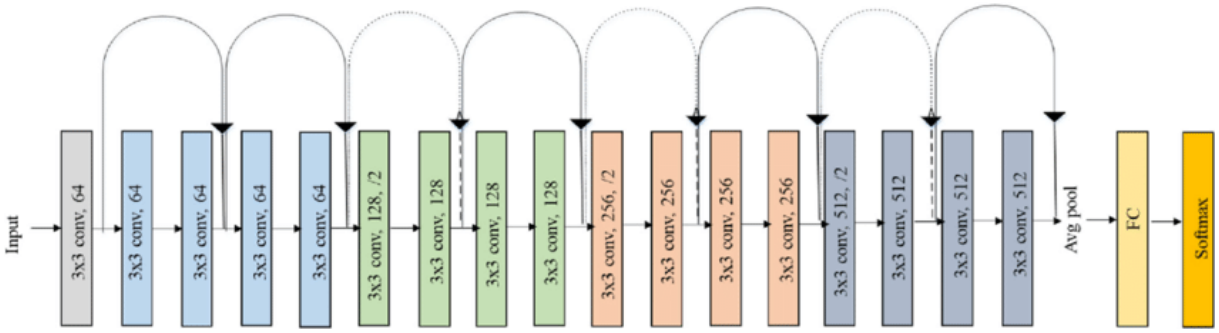

Figure 5.8: ResNet18 architecture: layer 1 in blue, layer 2 in green, layer 3 in red, layer 4 in grey

5.4. Implementation Details

In this section we include implementation details that are common to the two proposed methods.

5.4.1. Anomaly maps upsampling

For every ResNet's k-th layer we will obtain as output an anomaly score map. In these maps, each (i, j) position will be associated to an anomaly score computed as we described in the previous sections. What we want to specify here is that these intermediate score maps need to be upsampled. That's a consequence of maxpooling layers stacked between convolutional ones (or, equivalently, a stride > 1 in convolutional layers) as we explained in (4.1.2).

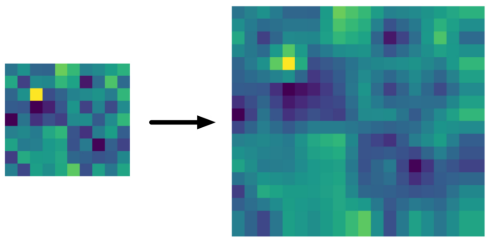

Figure 5.9: Upscaling of a generic activation map

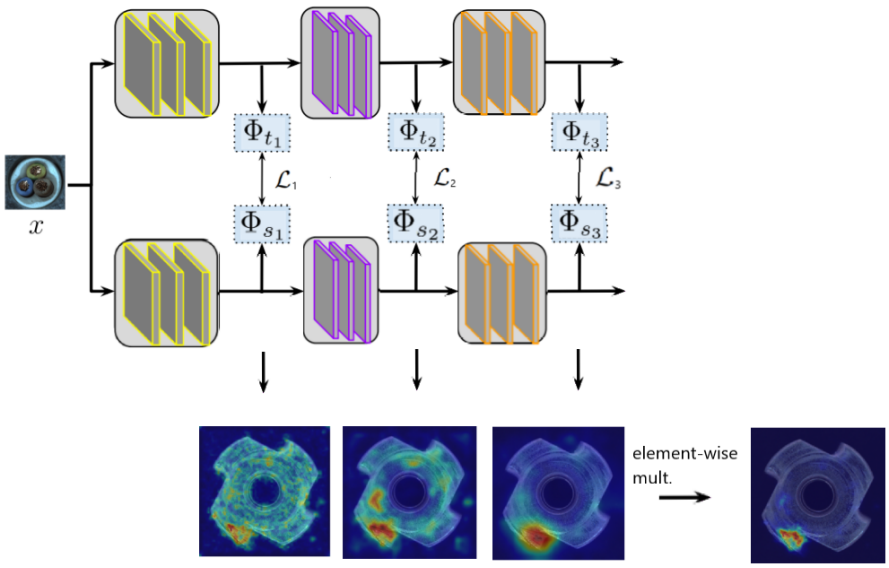

Figure 5.10: Intermediate anomaly score maps are upscaled to input dimensions and then combined by element-wise multiplication

Usually bilinear interpolation is applied to each intermediate score map in order to reach input dimensions (h, w). After that, the resulting maps are combined through element-wise product to get scores that take into account different scales of anomalies.

5.4.2. Setting Detection Threshold

The anomaly score map doesn't automatically imply a specific threshold value. In this work we are mostly interested in the anomaly detection aspect so we chose the threshold value based on the anomalous/nominal predictions $y_x \in \{0, 1\}$, disregarding the segmentation accuracy (for what concerns this choice).

$$anomaly\_score(x_t) = max(anomaly\_score(x_t(i,j)) \quad (i,j) \in \mathbb{R}^{(h,w)} \tag{5.5}$$

We always consider the image-level anomaly score as the maximum value of the combined heat_map described previously.

$$threshold_{optimal} = argmax\left(2 \cdot \frac{precision(t) \cdot recall(t)}{precision(t) + recall(t)}\right) \tag{5.6}$$

Given a threshold t, we find the value for which the image-level F1-score is maximized. This threshold will be used for the prediction of anomaly map $\hat{\Omega}_x$ and the false bads/goods computation.

Image-level F1 score optimization is not the only viable strategy. Another sensible choice would be to maximize the Pixel-Level F1 score, but that would imply that ground-truth masks are provided in the validation set, and that's not so common in industrial settings.

Min Max Normalization

Normalized test results are consistent throughout different datasets and give a clear sense of how distant a sample is from the decision boundary. So, after we find the optimal threshold, we always apply Min-Max normalization to our test-images anomaly scores and we shift them in order to center the threshold value in 0.5. For every test image x_t:

$$normalized\_score(x_t) = \frac{(anomaly\_score(x_t) - threshold_{optimal})}{(max\_val - min\_val)} + 0.5 \tag{5.7}$$

Where max_val and min_val are maximum and minimum image-level anomaly scores in the test set.

6 | Experimental results

In this chapter we report the results obtained experimenting with our two proposed methods, using STFPM as a baseline.

6.1. Datasets

Following along the lines of all the recent papers dealing with semi-supervised visual anomaly detection, we also use the MVTec AD Dataset as a benchmark. MVTec AD is a relatively recent dataset (2019) containing over 5000 high-resolution images divided into 15 different categories. Some of them represent a texture like Tiles, Wood, Carpet, and many others while the others are objects such as Hazelnuts, Screws, Pills and so on. Each of these categories comprises a set of defect free images and a collection of defective images for each type of defect. Usually there is a small number of defect types for each category. An anomaly segmentation mask is also provided for each defective image, in order to evaluate the pixel-wise segmentation accuracy of our models.

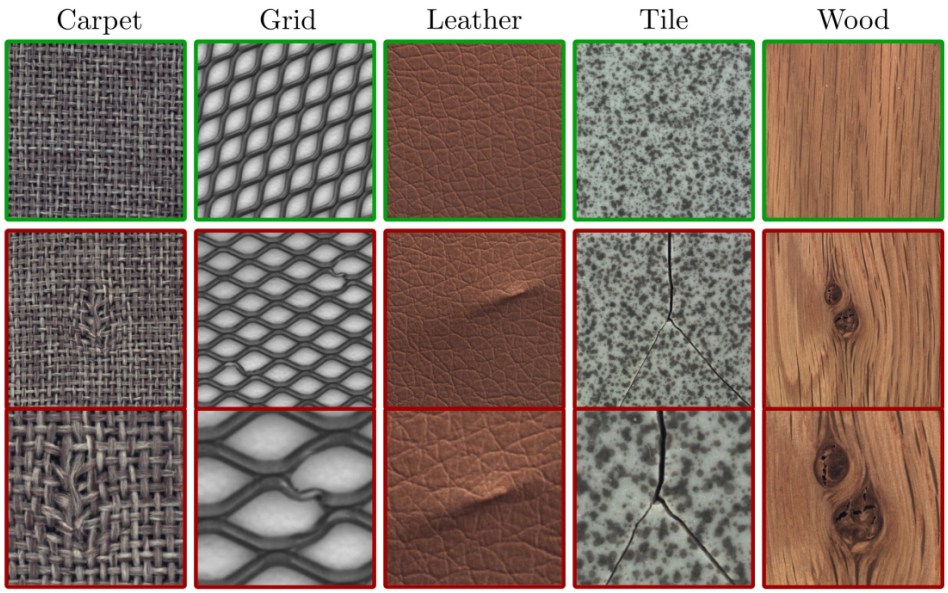

Figure 6.1: MVTec AD dataset textures samples

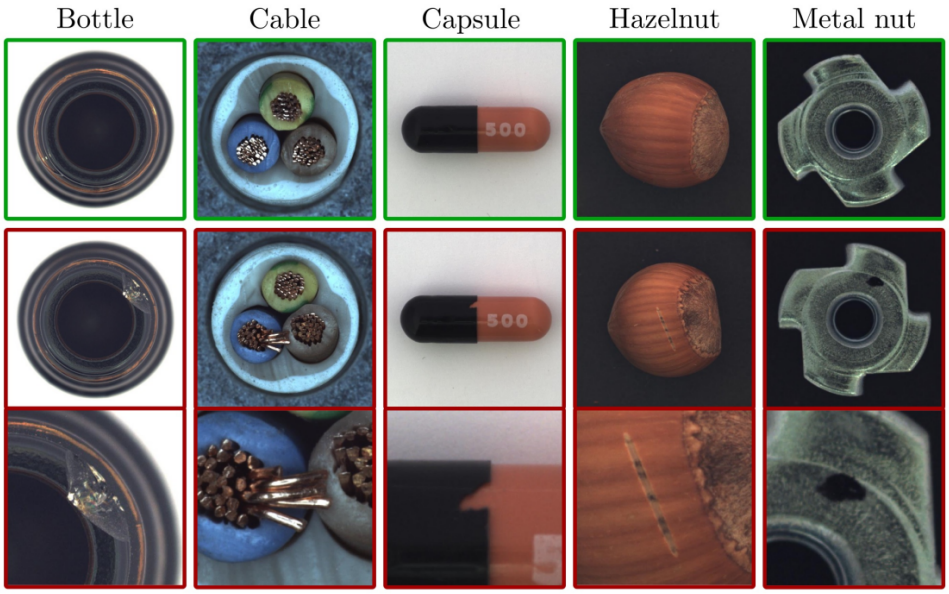

Figure 6.2: MVTec AD dataset objects samples

6.2. Metrics

To measure performance we rely on multiple metrics. In (4.5) we explained why AUROC (Area Under the Receiver Operating Characteristics) is a very helpful threshold-agnostic performance indicator in classification tasks. Even if we are not really training a classifier, to evaluate the anomaly detection performance is convenient to frame it as a nominal/defective binary classification task. As a segmentation metric we used the Dice Score. Dice score measures the overlap between segmentation and ground-truth mask. It is computed as 2 times the intersection area between the mask and the predicted mask, divided by the sum of the two masks' areas.

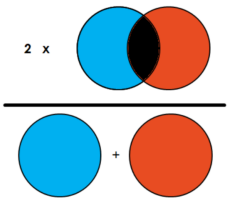

Figure 6.3: Dice score visualization

Note that even if we optimized the threshold on the image-level score, we are still interested in evaluating the pixel-level predictions of the model. Also false-bads and false-goods count is reported. Note that they were obtained setting the threshold as described in 5.4.2.

6.3. Hyperparameters

# Students	3	**Momentum**	0.9
Batch Size	32	**weight decay**	0.0001
Input Size	224x224	**feature extr.**	resnet18
Optimizer	SGD	**Layer Blocks**	1/2/3
LR	0.4	**Seed**	42

6.4. Numerical Results

We report here the results obtained testing our methods on the MVtec AD Datasets. STFPM is used as a baseline. Best result in each category is bolded only for relevant improvements (> 0.005).

6.4.1. Image AUROC

Image AUROC

	STFPM	Feature Vec.Var.	Simple Stu.
Bottle	0.997	0.997	0.998
Cable	**0.962**	0.946	0.943
Capsule	0.450	**0.659**	0.544
Carpet	**0.977**	0.970	0.963
Hazelnut	0.974	0.971	**0.998**
Leather	1.0	1.0	0.997
Metalnut	0.954	**0.986**	0.558
Pill	0.516	**0.720**	0.448
Screw	0.748	**0.983**	0.956
Tile	0.983	**0.995**	0.988
Toothbrush	0.683	**0.763**	0.619
Transistor	**0.899**	0.874	0.814
Zipper	0.876	0.868	**0.912**
Wood	0.993	0.996	0.996

6.4.2. Dice Score

Dice Score is obtained comparing predicted mask $\hat{\Omega}_x$ with the one provided by the dataset. To compute the mask, threshold based on image-level F1 score is used (see 5.4.2).

Dice Score

	STFPM	Feature Vec.Var.	Simple Stu.
Bottle	0.661	0.663	0.555
Cable	0.513	**0.542**	0.488
Capsule	0.0133	**0.219**	0.0137
Carpet	0.641	0.642	0.594
Hazelnut	0.568	0.591	**0.610**
Leather	0.485	**0.507**	0.478
Pill	0.0610	**0.132**	0.00157
Metalnut	0.446	**0.605**	0.0104
Screw	0.120	**0.257**	0.238
Tile	0.520	**0.543**	0.519
Toothbrush	0.057	**0.126**	0.0643
Transistor	**0.611**	0.587	0.408
Zipper	0.446	0.448	0.417
Wood	0.573	0.559	0.570

6.4.3. False bads/goods

False bads/goods count is obtained setting the threshold as described in 5.4.2.

	STFPM	Feature Vec.Var.	Simple Stu.
Bottle	1/0	1/0	2/1
Cable	6/6	10/5	11/8
Capsule	22/0	15/3	22/0
Carpet	1/5	1/4	0/7
Hazelnut	11/0	6/2	0/4
Leather	0/0	0/0	1/0
Metalnut	3/5	2/3	21/0
Pill	24/0	21/3	26/0
Screw	24/6	0/5	4/5
Tile	3/2	1/3	1/4
Toothbrush	10/0	7/1	10/0
Transistor	4/10	5/12	22/5
Zipper	10/3	10/6	9/2
Wood	0/1	0/1	0/1

Table 6.1: False bads/goods

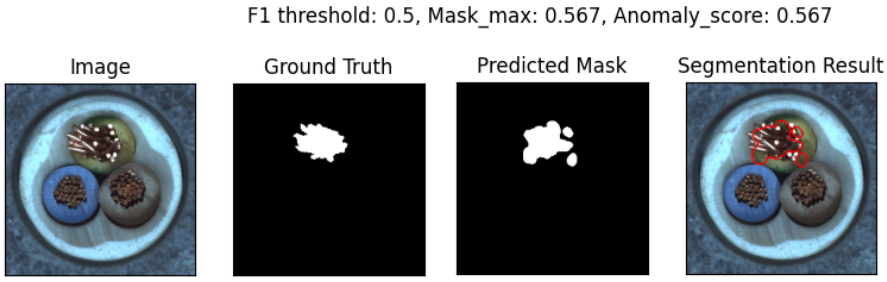

Figure 6.4: Feature Vectors Variance, output example 1

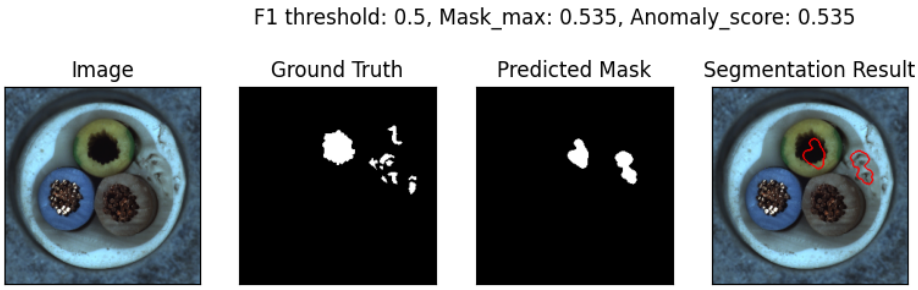

Figure 6.5: Feature Vectors Variance, output example 2

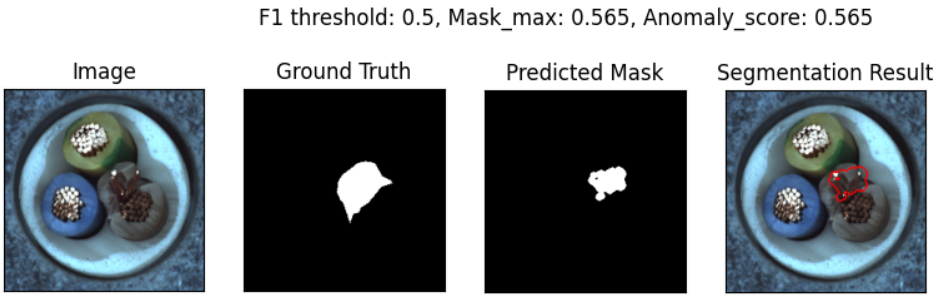

Figure 6.6: Simple Student, output example 1

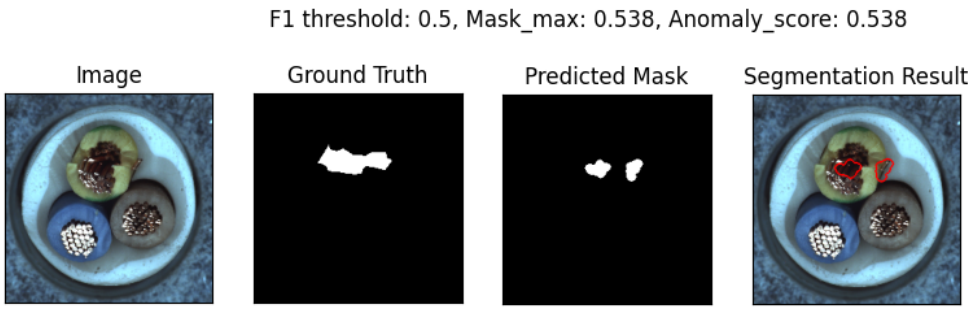

Figure 6.7: Simple Student, output example 2

6.5. Variance-only anomaly score

Another interesting experimental result is related to Feature Vectors Variance. In two object categories, using ensemble variance only as anomaly score leads to even better

results. This is an impressive outcome that further confirms the validity of this approach. Numerical results and segmentations reported below are obtained purely from Students' "disagreement". Unfortunately this happens only in these two datasets, while in the others the results are worse than the STFPM baseline. So we can only present it as a visual proof of how the Student networks produce more variable feature maps values in correspondence of anomalous regions.

	Variance-only	Dice score	False bads/goods
Capsule	0.786	0.113	13/0
Hazelnut	0.995	0.689	2/0

Table 6.2: Numerical results with variance-only anomaly scoring

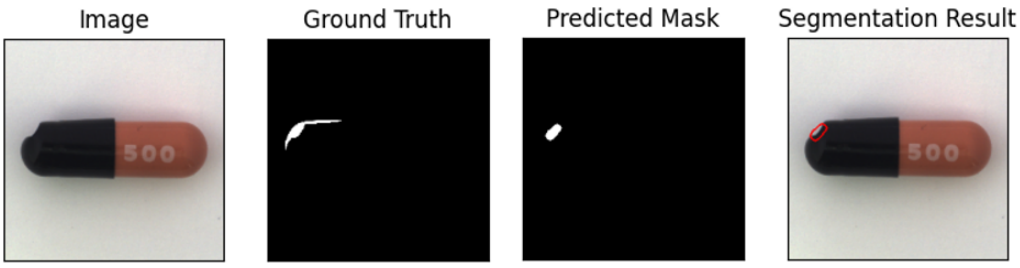

Figure 6.8: Capsule segmentation results with variance-only anomaly scoring

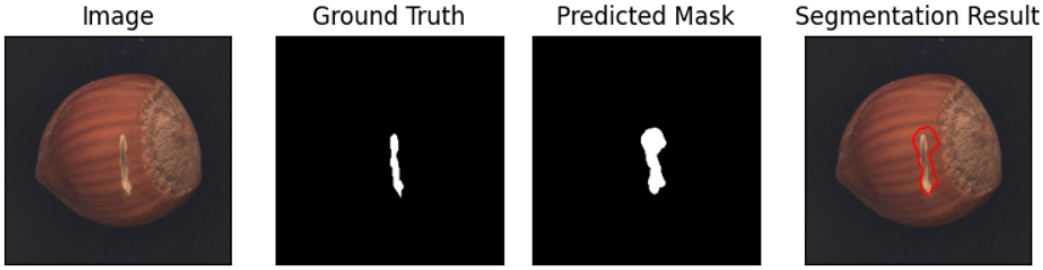

Figure 6.9: Hazelnut segmentation results with variance-only anomaly scoring

7 | Conclusions and future developments

For what concerns Feature Vectors Variance, the experimental results denote a clear improvement for some classes of objects, especially for what concerns the Dice Score, that is almost always improved by our method. Even the Image Auroc improves substantially for some categories (see Capsule, MetalNut, Pill, Screw and ToothBrush). A possible limitation concerns the computational resources needed by our method. While STFPM only needs to keep two networks in memory, our technique requires an entire ensemble of Students, plus the Teacher. Given that in Feature Vectors Variance Teacher and Students share the same architecture, evaluating the memory consumption increase is straightforward. While STFPM loads 2 models on the VRAM, our method loads 4 of them. So the greater computational effort with respect to STFPM is undeniable. In some cases, though, a trade-off between resources and improved performances could be the most convenient choice. A trivial approach would be to use bigger networks like WideResNet-50, but that does not seem to improve the anomaly detection capabilities (see [1]). Therefore, in these situations our method offers a real advantage. Simple Student shows an improvement for less classes of objects. That's likely a consequence of an excessive simplification of the Student's architecture. Still, results clearly show that Simple Student can induce substantial improvements. A possible future development would be to better design the Student, maybe decoupling completely Student and Teacher architectures and including fully connected layers to adapt the two networks' intermediate outputs dimensions (as it's done through the decoder in [3]). That would give more freedom to design a Student that is simpler but still able to represent nominal features well enough.

Bibliography

[1] Anomalib stfpm implementation. https://github.com/openvinotoolkit/anomalib/tree/main/anomalib/models/stfpm. Accessed: 17-08-2022.

[2] C. Bailer, T. Habtegebrial, K. Varanasi, and D. Stricker. Fast dense feature extraction with convolutional neural networks that have pooling or striding layers. 01 2017. doi: 10.5244/C.31.101.

[3] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger. Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings. *CoRR*, abs/1911.02357, 2019. URL http://arxiv.org/abs/1911.02357.

[4] 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*, 129(4):1038–1059, Apr 2021. ISSN 1573-1405. doi: 10.1007/s11263-020-01400-4. URL https://doi.org/10.1007/s11263-020-01400-4.

[5] C. Buciluundefined, R. Caruana, and A. Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '06, page 535–541, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595933395. doi: 10.1145/1150402.1150464. URL https://doi.org/10.1145/1150402.1150464.

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

[7] H. Deng and X. Li. Anomaly detection via reverse distillation from one-class embedding. *CoRR*, abs/2201.10703, 2022. URL https://arxiv.org/abs/2201.10703.

[8] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009. doi: 10.1109/CVPR.2009.5206848.

[9] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning, 2016. URL https://arxiv.org/abs/1605.09782.

[10] I. Golan and R. El-Yaniv. Deep anomaly detection using geometric transformations. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/5e62d03aec0d17facfc5355dd90d441c-Paper.pdf.

[11] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial networks, 2014. URL https://arxiv.org/abs/1406.2661.

[12] J. Gou, B. Yu, S. J. Maybank, and D. Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, mar 2021. doi: 10.1007/s11263-021-01453-z. URL https://doi.org/10.1007%2Fs11263-021-01453-z.

[13] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. URL http://arxiv.org/abs/1512.03385.

[14] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network, 2015. URL https://arxiv.org/abs/1503.02531.

[15] A. Kendall and Y. Gal. What uncertainties do we need in bayesian deep learning for computer vision? In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/file/2650d6089a6d640c5e85b2b88265dc2b-Paper.pdf.

[16] D. P. Kingma and P. Dhariwal. Glow: Generative flow with invertible 1x1 convolutions, 2018. URL https://arxiv.org/abs/1807.03039.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[18] x. lan, X. Zhu, and S. Gong. Knowledge distillation by on-the-fly native ensemble. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran As-

sociates, Inc., 2018. URL https://proceedings.neurips.cc/paper/2018/file/94ef7214c4a90790186e255304f8fd1f-Paper.pdf.

[19] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.42.7665.

[20] C.-L. Li, K. Sohn, J. Yoon, and T. Pfister. Cutpaste: Self-supervised learning for anomaly detection and localization, 2021. URL https://arxiv.org/abs/2104.04015.

[21] Y. Liu, X. Jia, M. Tan, R. Vemulapalli, Y. Zhu, B. Green, and X. Wang. Search to distill: Pearls are everywhere but not the eyes. *CoRR*, abs/1911.09074, 2019. URL http://arxiv.org/abs/1911.09074.

[22] P. Napoletano, F. Piccoli, and R. Schettini. Anomaly detection in nanofibrous materials by cnn-based self-similarity. *Sensors*, 18(1), 2018. ISSN 1424-8220. doi: 10.3390/s18010209. URL https://www.mdpi.com/1424-8220/18/1/209.

[23] K. Roth, L. Pemula, J. Zepeda, B. Schölkopf, T. Brox, and P. Gehler. Towards total recall in industrial anomaly detection, 2021.

[24] M. Rudolph, B. Wandt, and B. Rosenhahn. Same same but differnet: Semi-supervised defect detection with normalizing flows. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pages 1907–1916, January 2021.

[25] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft. Deep one-class classification. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4393–4402. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/ruff18a.html.

[26] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. *CoRR*, abs/1703.05921, 2017. URL http://arxiv.org/abs/1703.05921.

[27] B. Schölkopf, R. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. volume 12, pages 582–588, 01 1999.

[28] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra. Grad-cam: Why did you say that? visual explanations from deep networks via gradient-

based localization. *CoRR*, abs/1610.02391, 2016. URL http://arxiv.org/abs/1610.02391.

[29] D. M. Tax and R. P. Duin. Support vector data description. *Machine Learning*, 54 (1):45–66, Jan 2004. ISSN 1573-0565. doi: 10.1023/B:MACH.0000008084.60811.49. URL https://doi.org/10.1023/B:MACH.0000008084.60811.49.

[30] G. Wang, S. Han, E. Ding, and D. Huang. Student-teacher feature pyramid matching for anomaly detection. 06 2022.

[31] J. Yu, Y. Zheng, X. Wang, W. Li, Y. Wu, R. Zhao, and L. Wu. Fastflow: Unsupervised anomaly detection and localization via 2d normalizing flows. *CoRR*, abs/2111.07677, 2021. URL https://arxiv.org/abs/2111.07677.

[32] L. Zhang, J. Song, A. Gao, J. Chen, C. Bao, and K. Ma. Be your own teacher: Improve the performance of convolutional neural networks via self distillation. *CoRR*, abs/1905.08094, 2019. URL http://arxiv.org/abs/1905.08094.

[33] Y. Zhang, T. Xiang, T. M. Hospedales, and H. Lu. Deep mutual learning. *CoRR*, abs/1706.00384, 2017. URL http://arxiv.org/abs/1706.00384.

[34] B. Zong, Q. Song, M. R. Min, W. Cheng, C. Lumezanu, D. Cho, and H. Chen. Deep autoencoding gaussian mixture model for unsupervised anomaly detection. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJJLHbb0-.

A | Appendix A

We report here the Student and Teacher models architectures used for Simple Student experiments

A.1. Teacher's architecture

```
ResNet(
  (conv1d): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

A.2. Student's architecture

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 16, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(16, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(16, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
```

```
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

B | Appendix B

We report here anomalib's ([1]) STFPM implementation results. Anomalib is an Intel OpenVINO's project collecting state of the art visual anomaly detection models' implementations.

B.1. Image AUROC

	Avg	Carpet	Grid	Leather	Tile	Wood	Bottle	Cable	Capsule	Hazelnut	Metal Nut	Pill
ResNet-18	0.893	0.954	0.982	0.989	0.949	0.961	0.979	0.838	0.759	0.999	0.956	0.705
Wide ResNet-50	0.876	0.957	0.977	0.981	0.976	0.939	0.987	0.878	0.732	0.995	0.973	0.652

Screw	Toothbrush	Transistor	Zipper
0.835	0.997	0.853	0.645
0.825	0.5	0.875	0.899

B.2. Pixel AUROC

Note that in our work we estimated pixel-wise segmentation accuracy using a different metric (Dice score).

	Avg	Carpet	Grid	Leather	Tile	Wood	Bottle	Cable	Capsule	Hazelnut	Metal Nut	Pill
ResNet-18	0.951	0.986	0.988	0.991	0.946	0.949	0.971	0.898	0.962	0.981	0.942	0.878
Wide ResNet-50	0.903	0.987	0.989	0.980	0.966	0.956	0.966	0.913	0.956	0.974	0.961	0.946

Screw	Toothbrush	Transistor	Zipper
0.983	0.983	0.838	0.972
0.988	0.178	0.807	0.980

List of Figures

List of Tables

Acknowledgements

First of all, i want to thank Prof. Giacomo Boracchi for the opportunity to write this thesis with his supervision and for introducing me to the two companies that kindly hosted me during my thesis work: Antares Vision S.p.A. and Orobix S.r.l. . I want to also thank Daniele Lombardo, Silvia Bianchetti and Lisa Lozza for their advice and patience.

I'm also immensely thankful to my family and my girlfriend for their unconditional support. I want to express my gratitude to all those people that made this journey through Polimi easier and, more importantly, taught me how to be a better person.