On Deep Machine Learning Methods for Anomaly Detection within Computer Vision

PHILIP ANTHONY ADEY

On Deep Machine Learning Methods for Anomaly Detection within Computer Vision

Philip Adey

A Thesis presented for the degree of
Doctor of Philosophy

Department of Computer Science
Durham University
United Kingdom
December 2021

Abstract

This thesis concerns deep learning approaches for anomaly detection in images. Anomaly detection addresses how to find any kind of pattern that differs from the regularities found in *normal* data and is receiving increasingly more attention in deep learning research. This is due in part to its wide set of potential applications ranging from automated CCTV surveillance to quality control across a range of industries. We introduce three original methods for anomaly detection applicable to two specific deployment scenarios. In the first, we detect anomalous activity in potentially crowded scenes through imagery captured via CCTV or other video recording devices. In the second, we segment defects in textures and demonstrate use cases representative of automated quality inspection on industrial production lines. In the context of detecting anomalous activity in scenes, we take an existing state-of-the-art method and introduce several enhancements including the use of a region proposal network for region extraction and a more information-preserving feature preprocessing strategy. This results in a simpler method that is significantly faster and suitable for real-time application. In addition, the increased efficiency facilitates building higher-dimensional models capable of improved anomaly detection performance, which we demonstrate on the pedestrian-based UCSD Ped2 dataset. In the context of texture defect detection, we introduce a method based on the idea of texture restoration that surpasses all state-of-the-art methods on the texture classes of the challenging MVTecAD dataset. In the same context, we additionally introduce a method that utilises transformer networks for future pixel and feature prediction. This novel method is able to perform competitive anomaly detection on most of the challenging MVTecAD dataset texture classes and illustrates both the promise and limitations of state-of-the-art deep learning transformers for the task of texture anomaly detection.

Declaration

The work in this thesis is based on research carried out at the Department of Computer Science, Durham University, United Kingdom. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I am grateful to my main supervisor, Professor Toby Breckon, and the founder and CEO of COSMONiO, Dr Ioannis Katramados, for giving me the opportunity to undertake this PhD at Durham University with support from The EPSRC. Directly, it has given me new skills and experience; through consequence, it has given me friends, journeys to places I had never been, and, as of three weeks ago, a wife.

I am also grateful to have been joined at the start by Dr Magnus Bordewich as secondary supervisor. Having the opportunity to work under Toby, Magnus and Ioannis was great fortune both personally and professionally. They have been supportive, encouraging and most accommodating of my needs and circumstances. They gave me the freedom to explore my own ideas and valuable feedback. Deepest thanks to you all.

Working with COSMONiO (who later became Intel) as an industry partner, I was fortunate to have peers around me both in the lab at Durham and in The Netherlands. My collaborations and weekly meet-ups with them were an integral part of these past four years. I will remember them fondly.

I would also like to thank the people who have been the closest from the very beginning and who have always been there with support, encouragement and an unwavering belief in me. These special people are my mother, my father and my brother.

Finally, to my new wife Rebecca, who stayed up late into the night with me to roll potatoes along a treadmill, thank you for your constant love, support, encouragement, and the most memorable moments of the past four years.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This thesis considers how to use modern machine learning techniques to perform anomaly detection in images. Speaking generally, an anomaly is any deviation from the kinds of patterns observed in *normal* data. This interest in detecting any deviation from normality is what defines anomaly detection and separates it from other machine learning tasks. Other tasks, such as image classification, require predefining a set of relevant classes and supplying large quantities of samples for each at training time [54]. This is not viable in anomaly detection since the set of all anomalies could be considered a set of limitless classes, almost all of which are unknown and unrepresented by any collected samples. In anomaly detection, we therefore develop methods that can be trained using only samples of the plentiful normal data and operate by detecting deviations from that normality at inference time.

We address two specific contexts of anomaly detection. In the first context, we aim to detect abnormal activity in scenes that may be captured for example via CCTV, webcam or smartphone. Examples of such scenes include pedestrianised zones and views of conveyor belts on industrial production lines. Here, our research is motivated by the interest in the field and the availability of CCTV style datasets

[65, 62, 64] on which existing anomaly detection methods have already been tested [36, 81, 18, 40]. Computerised anomaly detection methods have potential application in the automated monitoring of scenes to flag criminal activity or defected or foreign objects that may have fallen onto a production line. In the second context, we aim to segment defects in textural patterns such as textiles. This context is motivated by interest from our industrial partner and the release of a new dataset [11] that provides a high-quality and thorough benchmark for such defect segmentation. Automated methods in this context are a key component within quality control for a range of high-speed, high-yield manufacturing industries that rely on camera-based visual inspection techniques. Computerised anomaly detection methods deployed in these two contexts carry several advantages: they never become tired, can be deployed 24 hours a day, and can be replicated across a network of locations for a low cost.

Although the state-of-the art in computer vision tasks in general has made tremendous progress over the past decade, there has only been recent focus on anomaly detection. The effectiveness and suitability of automated anomaly detection methods in our stated contexts is bounded by their rates of false-positive and false-negative errors. On the former type of error, automated methods serve no purpose if they erroneously flag anomalies too frequently, tying up the resources they were intended to save and causing alertness fatigue. On the latter type of error, automated methods fail at the very purpose they were deployed for if they frequently neglect to detect anything novel. This thesis aims to contribute to the state-of-the-art in this field to improve the quality of our automated systems.

1.1 Contributions

We introduce three new methods of anomaly detection, each of which has its own set of contributions.

The first method is the result of a series of enhancements to a method of anomaly detection that is based on Kernel Density Estimation (KDE) on features extracted from a deep, multi-task, neural network [40]. We improve processing speed by

changing the region proposal system from a combination of classical hand-crafted algorithms [51, 28] to our modified Region Proposal Network [80]. This pushes the method into the realm of real-time anomaly detection, being capable of operation on live video streams during both the training and inference phases. Leveraging this enhancement in speed, we improve anomaly detection performance by constructing higher-dimensional models. We further improve performance via a modified feature preprocessing strategy, removing the ReLU activation on the features, and by taking features from a lower layer.

The second method is a new method for textural defect segmentation that surpasses all current state-of-the-art methods. We use a relatively simple architecture, in contrast to other methods that often employ more complex architectures such as the Generative Adversarial Network [5, 9, 85], or the Variational Autoencoder [60, 21]. This method is able to perform better despite its simplicity due to a key reformulation of the task: we train an architecture to output the pixel shifts required to restore a texture to normality, rather than to output the restored image. This task is much simpler and enables the model to indicate areas of normality via outputting all zeros, which overcomes the difficulty associated with producing high-frequency outputs.

The final method introduces the concept of future pixel and feature prediction for textural defect segmentation and is the first example of such a method. We utilise a Transformer-based model for predicting the sequence of pixels or features that follows some given sequence of context pixels. At the time of our experimentation there were no Transformer-based methods of anomaly detection and at the time of writing there are still no pixel prediction methods that we are aware of.

The contributions presented in Chapter 3 and Chapter 4 of this thesis have been previously published in the following peer reviewed publications:

- P. Adey et al. Region based anomaly detection with real-time training and analysis. In *International Conference On Machine Learning And Applications*, pages 495–499. IEEE, 2019

- P. Adey et al. Autoencoders without reconstruction for textural anomaly detection. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2021

The details of these contributions are presented in Chapters 3 - 5.

In addition, the method presented in Chapter 3 has been integrated into the NOUS software product developed by our industrial partner COSMONiO (acquired by Intel as of late 2020). See appendix A for an overview of some of the work carried out during this integration period.

1.2 Methodology

The output of all our anomaly detection methods is an *anomaly heat map*. That is, for each input frame we output a grey-scale image of the same size whose pixel values represent anomaly scores from 0.0 (not at all anomalous) to 1.0 (most anomalous). When used as figures throughout this thesis, these heat maps are inverted for greater clarity so that the darker areas represent the more anomalous regions. Heat maps are a very effective tool in the evaluation of anomaly detection algorithms since they enable us to see why a particular numerical evaluation score was achieved. In anomaly detection, it is always the case that some errors are better or worse than others but this is not reflected in the binary ground truth anomaly segmentation masks provided by datasets and consequently, not reflected in the final numerical evaluation scores. For example, given a surveillance style dataset in which people are considered normal and vehicles are considered anomalous, an anomaly detection method that detects n pixels of a strangely coloured shirt as anomalous is penalised as harshly as one that detects n pixels of sky as anomalous. Heat maps allow us to

see precisely where high anomaly scores are assigned and make better judgements about the performance of an anomaly detection method. For this reason, It is common to see researchers making use of this kind of evaluation [60, 15, 85].

For the purpose of quantitative evaluation, a binary decision needs to be made for each pixel concerning whether it is anomalous (positive) or not. This is achieved by setting an anomaly score threshold. When a method assigns an anomaly score to a pixel that is above this threshold, then the method has made a positive prediction for that pixel. We construct a Receiver Operating Characteristic (ROC) curve by thresholding the anomaly scores many times using a range of thresholds and plotting the true-positive rate against the false-positive rate for each. The final evaluation metric is then the Area Under the ROC Curve (AUC), which ranges from 0.0 to 1.0 with 1.0 representing perfect performance. For the surveillance style datasets, this evaluation is carried out at both the frame-level and pixel-level, while for the texture style datasets, there is only pixel-level evaluation. The definition of pixel-level evaluation varies between the surveillance and texture style datasets. For the texture style datasets, each individual pixel across the test set generates either a true-positive or false-positive result for the ROC curve generation. For the surveillance style datasets, each frame generates either a true-positive or false-positive result based on anomalous pixel coverage. We follow other methods [104, 62, 40, 78] in assigning a true-positive result for a frame if the positive pixel predictions cover at least 40% of the ground-truth positive pixels and a false-negative result if, for a negative frame, any positive pixel predictions are made. This allows for consistency of evaluation for better comparison of methods.

Throughout this thesis we graphically represent the performance of our methods using a box plot of AUC scores. Different AUC scores are produced by repeating the training and testing phases eight times with different random seeds. This is important to consider when comparing our anomaly detection performance measures with others' who most frequently report only a single AUC score. We found that using a single fixed seed invariably results in a selection of hyperparameters tuned to that seed. In this scenario, evaluation metrics such as the AUC score are inflated because they do not reflect the performance on a typical run, but rather,

a lucky run. From this point, any change in circumstances negatively impacts the reported performance, even changes as immaterial as code refactoring that in no way changes the anomaly detection method. An example of how this may occur is if two sub-networks are initialised in a different order after refactoring, then the identical sequence of numbers drawn from the random number generator are placed into different parameters and thus, the networks start in a different initial condition that is inevitably not as favourable.

There is frequent use of *leave-one-out* anomaly detection evaluation [107, 106, 31] where a classification dataset is used to test an anomaly detection method by defining one class to be the anomaly and the rest to be normal. We strongly question this practice for evaluating anomaly detection methods and do not include any such evaluation in this thesis. Firstly, anomalies could be *any* deviation from normality, so there should be some variety in anomalous classes in the test set. Secondly, anomalies may range from subtle to blatant and it is unlikely that a test set with a single anomalous class will test a range of distances from the normal data. Thirdly, in a leave-one-out scenario, it is likely that the anomaly class will be significantly different from the others in some consistent way, which undermines the task. Finally, having the normal class span multiple classification classes may not be faithful to a real anomaly detection context.

1.3 Scope

Research into image-based anomaly detection takes place across a range of spectral bands including thermal [73], visible [36, 60], and X-ray [35] imaging. In addition, some research focuses on anomaly detection in medical imagery [86, 63], which spans a range of spectral bands and also includes ultra-sound. We limit our scope to anomaly detection in visible-band images.

The kinds of visible-band images used in anomaly detection research include pictures of scenes as could be obtained from CCTV or webcam (usually of pedestrianised zones) [59, 64, 62], samples of textures [13], medical scans [63], and imagery at the micro or nano-scales [71]. We deal exclusively with human-scale scene and

texture datasets.

Griffin et al. [35] provide a taxonomy of anomalies in X-ray security. We generalise this taxonomy to our scope as follows:

1. *Appearance*: an unusual shape, texture or colour.
2. *Semantic*: an unusual category of object.
3. *Appearance-given-semantics*: A normal category of object appears, in some way, unusual.
4. *Relative appearance*: A subset of objects appearing different from others e.g. among a set of uniformed workers in an office, one is wearing casual clothes.
5. *Arrangement*: Unusual distribution of objects.
6. *Artefact*: An image looks abnormal due to artefacts left by the capturing process e.g. camera malfunction, noise or compression.
7. *Co-occurrence*: An unusual collection of objects to see together.

We consider item one a *low-level* anomaly in that it may be detectable through inspection of local pixel data directly or through a small number of filters, such as hand-crafted features or features learned at a layer in a deep neural network close to the pixels. In contrast we consider item two a *high-level* anomaly. These anomalies may be very difficult to detect through direct inspection of low-level pixel data, rather, leveraging high-level concepts learned by units near the output layer of a deep neural network may produce better results.

The scope of this thesis covers items one, two and three. In principle, item six is also covered by the method presented in our second contribution chapter (Chapter 4), but this is not tested. Although we deal with CCTV-style input in Chapter 3, we only consider each object individually. We do not consider the location, number or interaction of objects, and neither do we compare them.

Some anomaly detection methods in the literature do not require training [22, 44]. They simply receive the input imagery and find objects that are most easily sepa-

rated from the others in some feature space. These kinds of method are the most apt for satisfying item four, but we do not experiment with such methods.

We do not include evaluation on the Subway [1] and UMN [70] datasets reviewed in Section 2.2. These datasets are old with poor quality imagery and do not contain the pixel-level ground truth we consider to be best for evaluation. They also target areas of the taxonomy that are outside of the scope of our methods such as relative appearance and arrangement. We also do not use UCSD Ped1 [65] because the pixel-level annotations are incomplete and again, the quality is quite low such that when examining individual frames by eye it is often hard to identify the anomalies. The ShanghaiTech Campus dataset [64] is a good quality dataset but so few other methods have used it and none we have found with pixel-level evaluation. This coupled with the computational time required to process this dataset means that we have omitted this dataset too.

1.4 Structure

The literature review presented in Chapter 2 is split into three sections. The background literature is discussed in Section 2.1 before anomaly detection datasets and methods are discussed in Sections 2.2 and 2.3 respectively.

Following the literature review, there are three contribution chapters that each present an original method of anomaly detection. The method presented in Chapter 3 foregoes any training of deep-learning models, relying exclusively on pretrained models that have already been exposed to a large variety of imagery in combination with a classic KDE based algorithm. Chapter 4 presents a method that rethinks the application of autoencoders in anomaly detection, resulting in a simple, fast and effective method that surpasses all state-of-the-art methods in textural defect segmentation on the MVTecAD dataset [11]. Inspired by the recent rise of transformer networks in the image domain, our final method explores the possibility of performing anomaly detection through future pixel prediction, resulting in a novel method that can out perform some of the previous generation's state-of-the art methods. This method is presented in Chapter 5.

Each of these chapters evaluates their respective method in the context for which they were designed. For the first method, that is the detection of abnormal activity in scenes. For the remaining two methods, that is the segmentation of textural defects. Following the three contribution chapters, Chapter 6 evaluates all methods across contexts.

The thesis concludes in Chapter 7 with an overall discussion of the methods presented, contributions made, and possible directions for future work.

CHAPTER 2

Literature Review

We review the literature in three parts: background literature (Section 2.1), anomaly detection datasets (Section 2.2) and anomaly detection methods (Section 2.3).

2.1 Background

This section reviews the relevant literature that forms the basis for the current state-of-the-art anomaly detection methods reviewed in Section 2.3 and those presented in the three method chapters of this thesis (Chapters 3 - 5).

2.1.1 Kernel Density Estimation

Kernel Density Estimation (KDE) [87] is a nonparametric means of estimating the Probability Density Function (PDF) of a random variable $\boldsymbol{x} \in \mathbb{R}^d$ given a set of n observations $\{\boldsymbol{x}_i\}$. It forms the basis for anomaly detection in our first proposed method (Chapter 3), where it is used to estimate the probability density of features extracted from test material.

KDE requires a kernel function $K : \mathbb{R}^d \mapsto \mathbb{R}$ that is centred at $\mathbf{0}$ and that integrates to 1.0, common examples of which include the step and Gaussian func-

tions. In a simplified manner, the algorithm may be understood as building up the probability density function by adding the kernel function at every point in the observed set. Thus, every point in the observed set contributes a lump in the surface of the PDF. Many different kernel shapes are possible, but the shape has less impact on the performance of the algorithm than the bandwidth parameter, h. The bandwidth determines the spread of the lump over the d-dimensional space, and one can imagine how too much or too little spread will negatively impact the performance. Taking the Gaussian kernel as an example, one could imagine a very tall but narrow kernel whose width is far smaller than the scale of distances between the observed points. This results in an estimated PDF whose value is close to zero at nearly all points in the d-dimensional space, but suddenly very large at the observed points. Conversely, if the kernel is so wide as to envelop all of the observed points, then the estimated PDF will be too smooth and will obscure underlying structures in the observed points. There is a lot of attention given to bandwidth selection in KDE [87, 88]. One rule-of-thumb often used is Scot's rule [87] in which the bandwidth is set as follows:

$$h = n^{\frac{-1}{d+4}} \tag{2.1}$$

Note that this rule-of-thumb only depends on the number of observed points and the dimensionality of the space. This implies that there is some expected scale over which all of the observed points occur. In our first contribution chapter (Chapter 3), we propose a preprocessing stage applied the observed points that improves our anomaly detection results.

The resulting PDF is a function of the observed points. Consequently, they need to be remembered by the algorithm and are used in calculations to find the estimated PDF at an arbitrary point. Therefore, as the number of observed points increases, the time taken to calculate the PDF at an arbitrary point increases, and does so quadratically. The main contribution of Chapter 3 is to significantly increase the speed of the KDE-based anomaly detector by reducing the number of observed points required.

We have discussed a simplified view of KDE. In practice, the algorithm does not simply accumulate the PDF via a series of summations with the kernel at the observed points. there is also whitening of the input data and scaling to ensure a valid PDF.

2.1.2 Image Classification Networks

The goal of image classification is to determine the label of the most salient object within an image. Intermediate layers within a deep image classification network are utilised in our first proposed method (Chapter 3) as feature representations for test material.

Starting in 2010, the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [83] has been among the most widely used benchmarks for evaluating image classification algorithms. From 2012, the deep-learning class of image classification algorithms began to dominate, with AlexNet [53] winning the competition that year. One of the key innovations was their GPU implementation of the Convolutional Neural Network (CNN) [55] to facilitate the training of the deepest architecture of the time. AlexNet also introduces the use of Rectified Linear Units (ReLU) in the architecture's hidden layers and Local Response Normalisation. Since then, deep-learning models for image classification have steadily advanced the state-of-the-art. Simonyan et al. [89] investigated the effect of further increasing the depth of these models, winning ILSVRC 2014. Concurrently, Szegedy et al. [96] developed GoogLeNet (Inception-v1), achieving similar results and winning another branch of the competition. A milestone is reached in 2015 with He et al. [39] surpassing human performance on the 2012 challenge by introducing Parametric ReLU (PReLU) units and an enhanced initialisation strategy. The same research team won ILSVRC 2015 with their ResNet [38] model that utilises residual connections in its architecture. The Inception team incorporate this idea into their architecture, resulting in Inception-v4 [95]. The methods introduced in this thesis make use of models and concepts from this generation of classifiers. Although research into image classification has continued, this is out of the scope of this thesis.

Later, it emerged that the features learned at the intermediate layers of these

models may be generalisable to other tasks [105]. Features learned at the lower layers are the most general and tend to become more specific as layers approach the output layer. This concept is very important for transfer learning [105] and feature extraction [40].

2.1.3 Object Detection Networks

Object detection extends classification. In this task, the aim is to determine the label of *all* salient objects within an image and furthermore, we aim to predict bounding boxes to locate them. In our first proposed method (Chapter 3), Object detection networks are used to form region proposals within the test material. These region proposals determine the locations where anomaly detection will be performed.

Among the most significant contributions towards solving this task are the Region-CNN (RCNN) [30] series of models developed by Girshick et al. The original RCNN model utilises an external region proposal system to generate candidate bounding boxes that may contain an object. Girshick et al. employ Selective Search [99], although many other techniques are also suitable. They then extract features from each region using AlexNet [53] and classify them using a per-class set of support vector machines. RCNN is very slow due to requiring a forward propagation through AlexNet for each region proposal. Fast-RCNN [29] addresses this by forward propagating the entire image through the convolutional layers once and then pooling a fixed size feature map for each region proposal. Only from this point are the regions processed using separate computations through the fully-connected layers. This iteration of the RCNN series introduces the bounding box regressor and replaces the support vector machines with a softmax layer for outputting probabilities for each class plus a catch-all *background* class. At this point, the external region proposal system becomes the bottle-neck. Ren et al. [80] dispense with this external system and produce region proposals within the RCNN framework by adding a Region Proposal Network (RPN). The RPN shares the convolutional features with the bounding box regressor and softmax branches of Fast-RCNN and is trained alongside them. Other systems for object detection have been developed during and since the RCNN series as well as networks for object segmentation including

Mask-RCNN [37], the latest in the RCNN series. However, the anomaly detection methods introduced in this thesis only make use of concepts from the RCNN series before segmentation was introduced.

2.1.4 Autoencoders

A conventional autoencoder is a Multi-Layer Perceptron (MLP) that encodes an input into a latent-space representation and then attempts to decode this representation into a reconstruction of the input [32]. During training, distance between inputs and outputs are used to tune the autoencoder towards making more faithful reconstructions. In our second proposed method (Chapter 4), We modify an autoencoder so as to predict the pixel-shifts required to push a test image towards normality. These pixel shifts are then used to measure abnormality.

Masci et al. [66] adapt the conventional architecture by using convolutional layers. These Convolutional Autoencoders (CAE) are better suited to operating on image data and are often applied to anomaly detection in images and videos [36, 12, 60, 21].

Bengio et al. [10] introduce Denoising Autoencoders (DAE). DAE function similarly to the conventional autoencoder except that the training inputs are intentionally corrupted before encoding. DAE attempt to output a reconstruction of the original clean input. Xu et al. [104] apply DAE in their anomaly detection method.

Whatever variant of the autoencoder is employed, anomaly detection is usually accomplished via one of two methods. The first method utilises the inability of the model to reconstruct novel inputs so that the reconstruction error becomes a proxy for abnormality [36]. The second method uses the learned encoding function as a dimensionality reduction algorithm. Samples are compressed before being fed into a subsequent classification algorithm for anomaly detection such as K-Nearest Neighbours [91] or a one-class support vector machine [104, 98].

Autoencoders often have difficulty reconstructing the higher frequency information in the input image, which manifests in blurriness in the reconstructions [68]. When applied to anomaly detection, this results in errors in the values of normal pixels and consequently, anomaly scores that are often too high [12]. In the second

method chapter of this thesis we present an autoencoder-based method for anomaly detection in textures that overcomes this drawback and produces state-of-the-art results (Chapter 4).

2.1.5 Generative Adversarial Networks

Goodfellow et al. [33] introduce Generative Adversarial Networks (GAN) as a means of training a generative model to capture some training data distribution $p_x(x)$ over the data domain $\mathcal{X}$. The GAN framework has become a popular choice in anomaly detection methods [6, 85] and the first iteration of our modified autoencoder used in Chapter 4 was a subset of the GAN used by Akçay et al. [5].

In the GAN framework, there are two networks: a generator G and a discriminator D. The generator network is a function $G\colon \mathcal{Z} \mapsto \mathcal{X}$ that maps a latent vector z sampled from some prior distribution $p_z(z)$ over a domain $\mathcal{Z}$ to a point $x \in \mathcal{X}$. The distribution p_z and the domain $\mathcal{Z}$ are freely chosen by the implementation. The probability distribution over $\mathcal{X}$ observed when G produces outputs mapped from inputs sampled from p_z is $p_g(x)$. The discriminator network is a function $D\colon \mathcal{X} \mapsto \mathbb{R}$ that maps an input $x \in \mathcal{X}$ to the model estimation of the probability that x was sampled from p_x. D is trained to improve its estimation of the probability that its inputs are sampled from $p_x(x)$ rather than from $p_g(x)$, while G is trained to minimize the performance of D. The optimal solution for G is found when $p_g(x) = p_x(x)$, in which case samples from the output of G look identical to samples from the training distribution, and D can do no better than to predict a probability of 0.5 everywhere in $\mathcal{X}$. Unlike the standard autoencoder (Section 2.1.4), the training objective of GAN has an effective way of penalising the generation of blurry images since the discriminator can easily learn to identify these as not belonging to the data distribution. This procedure is equivalent to minimizing the Jensen-Shannon distance between the distributions p_g and p_x.

Although GAN has proved to be a powerful framework, a number of shortcomings have emerged. Firstly, mode collapse can occur [76, 84]. If p_x is a multi-modal distribution, there is no incentive for g_x to distribute density among the modes. A common strategy for G is to allow all outputs to converge onto a point that fools

the discriminator well. The discriminator may learn that this point is likely from the generator, but then the generator simply shifts its output point around the data space [84]. Secondly, instability in the training of training GAN often produces generators that yield noisy or incomprehensible outputs [76]. Finally, in the image domain, GAN are only capably of generating low-resolution images [76]. Salimans et al. [84] provide a suite of techniques for improving the training of GAN, most relevantly, feature matching. When applying this technique, features are extracted from discriminator inputs at an intermediate layer. The generator is trained to minimize the distance between features extracted from its own outputs and those extracted from the training data.

Radford et al. [76] propose a new GAN architecture for scaling up to higher-resolution image generation with more stability. They call their architecture Deep Convolutional GAN (DCGAN). Furthermore, they use trained DCGAN models as feature extractors for other image-based tasks and visualise their convolutional filters to show that specific filters learn to draw specific objects. They also show that certain directions in the latent space correspond to semantically meaningful variations in the training distribution such as gender or presence of glasses, and that smoothly varying the input latent space vector produces generated images that likewise vary smoothly in terms of their semantic content. The architectural changes include removing all pooling and fully-connected layers in favour of convolutional layers, using batch-norm, and changing the activation functions at each layer. They favour using the leaky ReLU function for all layers in the discriminator, and the ReLU function in all but the final layer of the generator, where they instead use the $tanh()$ function to squeeze the values back into the $[-1..1]$ range. The addition of batch-norm seems to alleviate mode collapse at the cost of allowing instances within a batch to affect each other [84].

The semantic nature of the latent space described above hints at the possibility that vectors from this space could serve as powerful features for use in down-stream tasks such as image classification [25]. Unfortunately, while the generator is able to map from the latent space to the data space, it is unable to perform the reverse mapping and so cannot be used as a feature extractor. Donahue et al. [25] therefore

introduce Bidirectional GAN (BiGAN) that simultaneously learns an encoder network to map from the data domain back into the latent space alongside the generator and discriminator of the original GAN framework. They are able to demonstrate that the BiGAN encoder is an adversarially learned feature extractor that can be trained without explicit labels, in contrast with typical supervised approaches [53].

Arjovsky et al. [8] improve the mode collapse and training instability by proposing the Wasserstein GAN (WGAN). WGAN performs gradient decent on the Wasserstein distance between the distributions p_g and p_x, rather than the JS distance. They rigorously show that the Wasserstein distance has much better properties.

2.1.6 Transformers

Transformer Networks have revolutionised the way we apply deep learning to sequence-to-sequence tasks. They were first introduced by Vaswani et al. [100] for use in language translation, superseding Recurrent Neural Networks (RNN) that were previously ubiquitous in the language processing domain. Transformers soon spread into other applications of deep learning, eventually finding applications in image processing [14, 74, 26]. In Chapter 5, we use a Transformer network to target anomaly detection through future pixel prediction.

A Transformer network takes a sequence of tokens as input and transforms them into an output sequence of tokens. Internally, the network represents each token of the sequence as an embedding in $\mathbb{R}^n$, which may be learned alongside the network parameters. Each layer of the network consists of an MLP that is independently applied at each position in the sequence, transforming each position into a new point in $\mathbb{R}^n$. Before the MLP however, an attention operation [100] is applied, which is the key feature that most differentiates the Transformer architecture from those that have come before.

The input to the attention operation consists of the query $Q \in \mathbb{R}^k$ and a set of key value pairs $\{K_i \in \mathbb{R}^k : V_i \in \mathbb{R}^v\}$ (often, $v = k$). The output is a weighted sum of the values where each weight is determined via a scaled dot product of the query with the associated key. A softmax function is used on the dot products to determine the scaling factor of each value. In this way, the attention mechanism causes each

position in its output to be a mixture of values, where each value is expressed according to the similarity between its key and the query. When the query, keys and values all come from the preceding layer in the network, this is referred to as *self-attention*. Vaswani et al. [100] also introduce multi-headed attention in which the query, keys, and values undergo multiple projections into a lower-dimensional space before the operation, and are combined and projected back into the original space after the operation.

The advantage that transformer networks have over the more conventional CNN architectures in the image domain is that while the kernels of the convolutional layers imply a hard-coded inductive bias that relevant information is only contained in neighbouring pixels, the attention mechanism of the transformer network facilitates learning where attention is best concentrated. However, this comes at the cost of a larger parameter space that is particularly costly in the image domain where inputs are often high-dimensional. Various schemes are employed to reduce computation requirements of the attention mechanism in the image domain including downscaling and clustering [14], memory blocks [74] and patch-based embedding [26].

The original Transformer architecture [100] consists of an encoder and a decoder, each of which is made up of a stack of six identical blocks. A block in the encoder performs multi-headed self-attention [100] over its input embeddings followed by a transformation through an MLP. Layer normalisation is applied after each step in the block. A block in the decoder is similar, except that the multi-headed self-attention is masked so that attention cannot be given to tokens following the current position, and multi-headed encoder-decoder attention follows the self-attention. The MLP and use of layer normalisation after each step remains the same.

A series of modifications to the original transformer architecture results in the GPT-2 model [77] and its application in the image domain [14]. This forms the basis of the architecture used in the third method chapter of this thesis, in which we propose a method of anomaly detection via pixel prediction (Chapter 5).

Liu et al. [58] provide the first modification to the original Transformer architecture by removing the encoder stack, forming the Transformer Decoder (TD). This architecture performs better under increasing sequence lengths; furthermore, they

suspect that for their monolingual task, training separate encoder and decoder stacks results in redundant learning and harder optimisation. This insight is interesting because our proposed pixel prediction method could also be considered monolingual, since the input and output tokens are from the same colour to token mapping.

Radford et al. [77] use the TD architecture with generative pretraining on a large corpus of text and then fine-tune on a range of tasks. Due to the Generative PreTraining, they call their method GPT. They also demonstrate an improved method, called GPT-2, whose architecture is modified by changing the parameter initialisations, moving the layer normalization to the start of each sub-block, adding an additional layer normalization after the final self-attention block, and expanding the vocabulary and context.

Chen et al. [14] apply the GPT-2 architecture [77] to the task of learning high-quality latent image representations for use in down-stream tasks such as image classification. In one of their approaches for learning these representations, they use next pixel prediction. Here, the sequence-to-sequence task is to map an input sequence of pixels, called the context, to the sequence of pixels that follows.

2.2 Anomaly Detection Datasets

A range of datasets for anomaly detection are commonly used in research. At one end of this spectrum is the abstract, algorithmically generated dataset contributed by the DAGM Symposium 2007 [102] that is intended to mimic the kinds of textures and defects observed at the micro-scale in industrial optical inspection. At the other, there are the surveillance style datasets such as that contributed by Shanghai Tech [64]. These datasets contain videos of pedestrianised zones in which there are periods of abnormal activity such as the appearance of a bike or other vehicles. In between these extremes the MVTecAD dataset for industrial optical inspection [11] supplies real image samples of textures and objects, a subset of which contain various defects such as dents, folds or contaminants. We review the surveillance style datasets first, followed by the texture style datasets in chronological order.

	Subway		UMN	UCSD		Avenue	Shanghai
	Ent.	Exit		Ped1	Ped2		
Year	2008		2009	2010		2013	2017
Type	Video		Video	Video		Video	Video
Channels	Gr		RGB/Gr	Gr		RGB	RGB
Size	512x384		320x240	238x158	240x360	640x360	856x480
Scenes	1	1	3	1	1	1	13
Training Set							
Videos	1	1	-	34	16	16	330
Frames	20,000	7,500	-	6,800	2,550	15,328	274,516
Testing Set							
Videos	6	4	1	36	12	21	107
Frames	124,249	57,401	7,739	7,200	2,010	15,324	40,791
Anomalies							
Count	66	19	11	50	20	58	136
Unique	5	3	1	6	3	11	25
Annotation							
Frame	✓	×	×	✓	✓	✓	✓
Pixel	✓	×	×	10 / 36	✓	✓	✓

Table 2.1: Summary of the surveillance-style datasets. The anomaly count is subjective, since there are overlapping anomalies, and anomalies that repeat after a short time period. In these cases there is a judgement to make about whether or not the anomalies are separate. Likewise, the count of unique anomalies is also subjective, since there is a judgement to make about how different one anomaly needs to be from another to belong to a separate category.

	DAGM07	Leaf	MVTecAD
Year	2007	2015	2019
Type	Image	Image	Image
Channels	Gr	RGB	RGB/Gr
Size	256x256	256x256	1024x768
Classes	6	14	5
Training Set			
Images	6000	15,084	1266
Testing Set			
Images	900	39,196	515
Anomalies			
Count	900	39,196	382
Unique	6	26	25
Annotation			
Coarse-Pixel	✓	×	×
Pixel	×	×	✓

Table 2.2: Summary of the texture-style datasets. The grid class of the MVTecAD dataset is grey-scale but all others are RGB.

Figure 2.1: Examples of frames from the Subway dataset. The left image is a frame from the subway entrance, and the right image is a frame from the exit.

2.2.1 Subway Entrance & Exit

Adam et al. [1] introduce the 2008 subway dataset, which has been used in the evaluation of a range of anomaly detection methods [36, 18, 64, 45, 22, 44, 109].

The dataset consists of CCTV videos of two similar scenes: a subway entrance and exit. Each scene has one long training video, and several test videos. Despite the reasonably high resolution, the quality of the video is not as good as most of the other datasets, and it is quite noisy. Features within the imagery that would be clear in the other datasets are blurry in this dataset.

Anomalies include people moving in the wrong direction, failure to pay, loitering and irregular interactions. Some of these are quite subtle, and rely on a high-level understanding of the situation. Anomalies of this kind of character are rare among the anomaly datasets.

While this dataset is included here due to the number of methods that report evaluation on it, this dataset will not be further utilised in this thesis because of the concerns about quality mentioned above and because the nature of the anomalies mentioned above do not fit within the scope of the thesis as outlined in Section 1.3.

2.2.2 UMN Unusual Crowd Activity

The University of Minnesota provides a number of datasets, one of which is frequently used to evaluate the detection of unusual crowd activity [45, 22, 44, 42, 109]. This dataset is commonly referred to as *the* UMN dataset in anomaly detection research. The UMN dataset consists of three different scenes as shown in Figure

Figure 2.2: Examples of frames from the UMN dataset. All three scenes are depicted, starting with scene one in the top row, and ending with scene three in the bottom row. The first column depicts a frame from the normal period, while the second column depicts a frame from the abnormal period.

2.2. In each scene, the same scenario plays out some number of times. First, there is period of normality lasting approximately 20 seconds or 600 frames in which a group of about 18 people walk around the scene in a random but calm manner. The scenario ends with a period of abnormality lasting approximately five seconds or 150 frames in which the people suddenly flee in all directions.

The video quality is fairly poor; the resolution is low and compression artefacts can be seen throughout the frames. Anomaly detection methods tested on this dataset need to be robust against this noisy input.

Similar to the Subway dataset discussed above, this dataset is discussed here only due to the number of methods that report evaluation on it. We do not use this dataset in our research because better quality datasets now exist and because we are interested in detecting anomalies of a different nature as described in Section 1.3. The only kind of anomaly addressed by this dataset is the panic scenario. In contrast, we are interested in detecting a wide variety of anomalies that we do not know in advance. In addition, this kind of anomaly requires consideration of the motion of actors relative to each other, whereas we consider the abnormality of individual actors.

2.2.3 UCSD Ped1 & Ped2

The UCSD dataset was provided by Mahadevan et al. in 2010 [65]. It is split into two parts: Ped1 and Ped2, each of which consists of grey-scale videos of pedestrians walking along a path. The training videos only show people walking on the paths and they move regularly with a fairly narrow distribution of speeds. The testing videos show people as well as other entities moving along the path such as bikes and skateboards. See Figure 2.3 for an example frame from each of the two parts.

In Ped1, the camera is pointed partially along the path, so that as the pixel row increases, the distance to the path decreases. Consequently, objects at the top of the video are smaller than objects at the bottom. Approximately 50% of the anomalies are bikes, 25% are skaters and 12% are motorised vehicles. The remaining few are wheelchairs, grass-walkers and one instance of someone pushing a cart along the path.

Figure 2.3: Example frames from the UCSD dataset. Left: the Ped1 dataset. Right: the Ped2 dataset.

In Ped2, the camera is pointed across the path, which reduces the difference in size between objects at the top and bottom of the video. There is slightly more detail than Ped1, due to the camera positioning, and so the appearance of bikes and skateboards is more obvious.

This dataset is popularly used for the evaluation of methods that target the kinds of anomalies that we are interested in [104, 36, 81, 110, 18, 64, 44, 40]. Therefore, this dataset is important for our research and will be used further in this thesis. While pixel and frame-level annotations are included for both Ped1 and Ped2, the pixel-level annotations for Ped1 are incomplete, and its quality is such that it is often difficult to identify anomalies by eye given a single frame. Anomalies are most apparent when watching the video, where the motions of the anomalous objects capture attention. Consequently, we focus on Ped2 which does not suffer these shortcomings.

2.2.4 CUHK Avenue

In 2013, Lu et al. [62] created a new dataset by capturing video from CUHK campus avenue. The frames are in colour and their resolution is higher than that of the UCSD datasets. Consequently, the level of detail is far superior, which significantly increases the range of identifiable patterns. Anomaly detection methods applied to this dataset will need to have a more sophisticated model of what is normal as they will encounter much greater variety of inputs including all combinations of clothes, accessories and rucksacks people could be wearing and items they could be carrying. There is also a greater variation in what people are doing; in the UCSD datasets,

Figure 2.4: An example frame from the CUHK Avenue dataset.

walking is the only normal activity, whereas in the Avenue dataset, people are standing, walking, meandering, patrolling, reading, using phones, wheeling suitcases etc. In addition, the light levels change from video to video, as does the colour-tone. Similar to the USCD datasets, the background is very static, crowd density varies and there is much occlusion. Figure 2.4 shows an example frame from this dataset.

The test set contains 58 anomalies, but there is a lack of variety. Of the 58 anomalies, 15 are a man throwing a bag, three are the same man throwing papers, and fifteen are the same man bending over to pick up either the bag or the papers. The objects are mainly thrown and picked up in exactly the same way, and so there is little variety within these three examples of anomaly. Nine of the anomalies are people running. Six are people entering or exiting from the path on which the camera is situated. This makes the people appear exceptionally large on screen as they walk past the camera. Unfortunately, there is an instance of this in the training set. Four of the anomalies are a boy skipping and the remainder are a boy loitering at the front of the scene, a bike being wheeled through the scene, some people dancing and a man walking on the grass. At one point, the man who threw the earlier items enters, hesitates and quickly leaves, which counts as the final anomaly.

There are also occurrences of events that are anomalous with respect to the training set, but that are not labelled as anomalous. This subjectivity can lead to algorithms reasonably generating false-positives, but being penalised for them as harshly as when other algorithms raise false-positives for no justifiable reason. For example, there is a woman sat on the grass with bright red bags, another woman loiters on the grass for a long time without moving very much at all. At another time, a man hurries through with colourful bags, and there are people on patrol who

gesture enthusiastically.

Hinami et al. [40] had observed similar issues with this dataset, and so removed five offending test videos. The remaining subset of videos is referred to as the Avenue17 dataset. On this subset, algorithms perform better [45] due to the lower apparent false-positive rate. Algorithms tested on this dataset are marked with an asterisk in Table 2.4 and Table 2.5.

Within this thesis, We include evaluation on the Avenue dataset in Chapter 3 to match that provided by a closely related method [40]. While the quality of this dataset is greater than all others discussed so far, that brings with it a range of complexities and issues as discussed above. From this point onwards in the development of anomaly datasets, it would be most useful to have a more refined ground truth that marks each pixel as either normal, anomalous or both; because with this level of detail, there are so many axes along which events can vary and whether or not a particular axis should be considered a sensitive anomaly trigger is a judgment call. For example, if someone enters the scene wearing a bright orange jumper, is that abnormal? It is certainly rarely seen. A method that triggers over such an event should be penalised less severely than a method that segments sections of pavement as abnormal. Currently, this is not the case and all deviations from the ground truth are penalised equally. This limits the usefulness of datasets of this level of complexity onwards.

2.2.5 ShanghaiTech Campus

Luo et al. [64] introduced the ShanghaiTech Campus dataset in 2017. This dataset is larger than all of the other datasets combined, with the training set containing 330 videos that span 13 scenes. Similar to the Avenue dataset, the level of detail and colour is sufficient to significantly expand the range of inputs to anomaly detection methods as shown in Figure 2.5. However, This dataset is better at capitalising on this, and explores the normal input space more thoroughly.

Considering the range of normal inputs, pedestrians walk either along a path, or on a pedestrianised zone. Similar to other datasets, there is large variety in crowd density, and people frequently occlude each other. Different to the UCSD

Figure 2.5: One example frame from each of the 13 scenes in the ShanghaiTech Campus dataset. This figure begins with scene one depicted in the top-left corner, and the scenes first increase to the right and then down.

and Avenue datasets, there are many normal paths that pedestrians can take. A number of scenes have multiple entrances and exits, allowing paths to cross, and the pedestrianised zones allow people to explore an area rather than a line. Anomaly detection methods must be robust against large variations in light level. Sometimes it is overcast with no shadows, sometimes it is brighter with well-defined shadows. The appearance of objects can change abruptly as they move from light areas to shaded areas. The dataset features lakes and large buildings that are fronted with dark glass, which provide a significant source of distorted, wavy reflections that could potentially trigger false-positive detections. Unique to this dataset, cameras sometimes rotate slightly between two videos of the same scene. Some scenes are the same as others, but shot from a different position and angle. No other dataset provides differing perspectives in this way.

Considering the range of anomalous inputs, the test set contains 107 videos that span 12 of the 13 scenes. This is far more testing videos than in other datasets, but each video is short, and most often cover a single anomaly. By cropping the anomalies out in this way, the range of normal inputs present in the test set is reduced, thus decreasing the opportunity for algorithms to produce false-positives. There are many more anomalies in this dataset, and they have greater variety. This can be seen in Table 2.3, which shows the anomaly counts. For example, there are several occurrences of pushing, falling and fighting that do not present in other datasets. However, there are still overwhelmingly more instances of bikes than other

Table 2.3: Summary of anomalies in the ShanghaiTech Campus dataset.

Actions		Vehicles	
Run / chase	19	Bike	46
Jump / repeatedly jumping	7	Skateboard	10
Fight / small conflict	6	Moped	6
Fall over / Push over	5	Van	2
Throw bag	5	Car	3
Abnormal path	4	Small 3-wheeled truck	2
Jogging	3	Segway	1
Steal bag	3	Other	
Hop over railing	2	Pram	3
Swing tripod like batton	2	Wheeling trolley	1
Drop object	1	Kids stare at camera	1
Bend to pickup object	1	Toddler runs	1
Ambush	1	Step onto small wall	1

anomalies, leaving these new and novel anomaly classes only partially explored.

This dataset is not included further in this thesis because so few other methods have provided evaluation on it, and none with pixel-level evaluation. There is also a large computational time cost to using this dataset and of course, the issues discussed with the Avenue dataset relating to increased complexity are relevant for this dataset too.

2.2.6 DAGM07

The Symposium of the German Association for Pattern Recognition 2007 (DAGM07) held a competition for weakly supervised learning in which challengers were tasked with detecting defects in ten different algorithmically generated textures [102]. Figure 2.6 provides an example from each of these different textures. The images, being algorithmically generated, bear no resemblance to anything in the real-world; however, the challenge is intended to simulate the task of industrial optical inspection of products. One drawback of this dataset is that each texture class only exhibits one kind of defect, although there is a wide variety in how the defect manifests itself

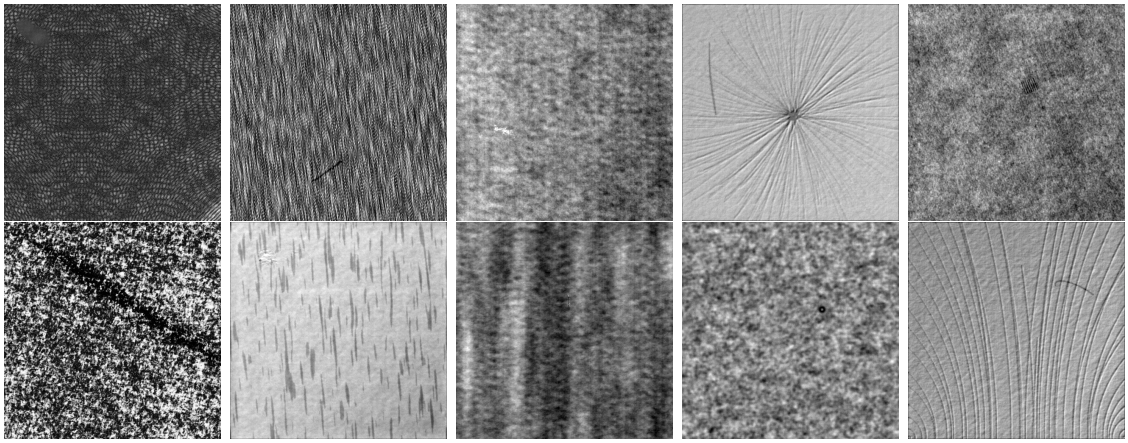

Figure 2.6: One example defected image from each of the ten DAGM07 classes.

across the images in terms of location, size, and orientation in most cases.

While there is ground truth supplied with this dataset, it is unfortunately very coarse in that ellipsoids are provided to mark the general location, size and shape of the anomalies rather than a pixel mask as in other datasets. Therefore, methods that segment anomalies are severely penalised for not labelling the normal pixels captured in the ellipse. We therefore use this dataset only for qualitative evaluation of our method presented in Chapter 4. This dataset is not relevant for our method presented in Chapter 3, since that operates on surveillance style datasets. We omit this dataset from the evaluation of our final method presented Chapter 5 since we were already able to demonstrate both the promise and the limitations of the method using a dataset with fine-grained ground truth. In addition, this method is costly in terms of computational time. It is also more important to include this dataset for extra evaluation in Chapter 4 in particular, since there are hard-coded aspects to the method that require demonstration that they have not been over-fit to one particular dataset.

2.2.7 Leaf

Hughes et al. [43] provide the Leaf dataset, which consists of a very large number of images of leaves spanning 15 species, some of which are healthy and some of which show signs of various diseases. Figure 2.7 shows one example of each kind of leaf with disease.

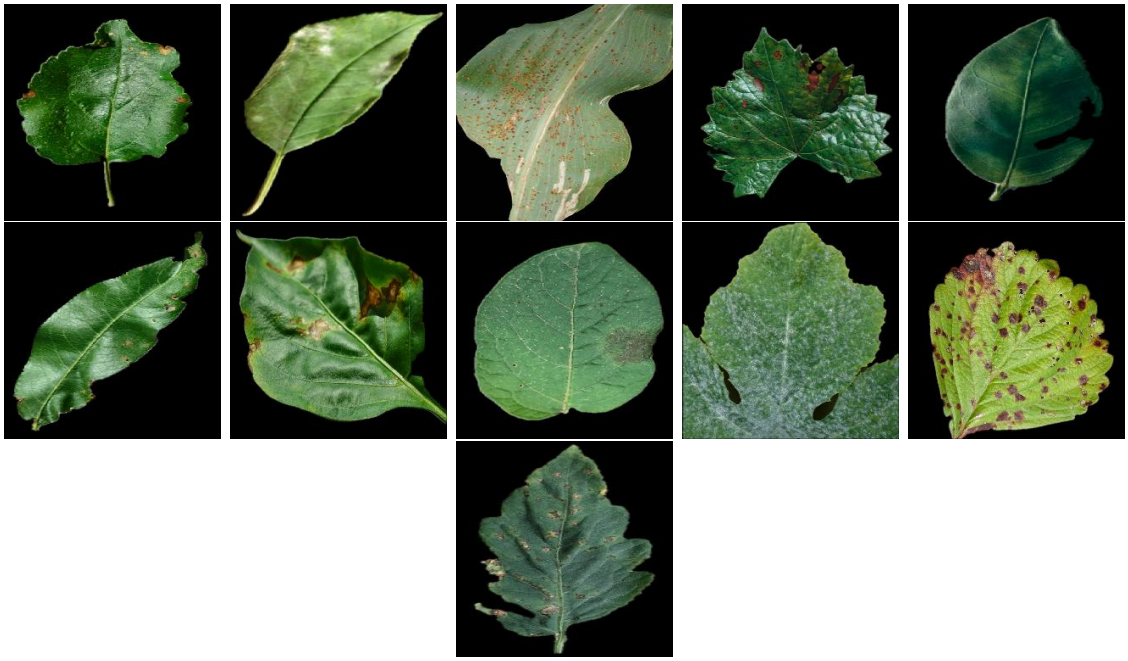

Figure 2.7: One example image from each species of leaf in the Leaf dataset that has a diseased category: apple, cherry, corn, grape orange, peach, pepper, potato, squash, strawberry, and tomato.

Although none of the methods reviewed in Section 2.3 are tested on this dataset, we use it as an additional test for the method that we present in Chapter 4, which achieves state-of-the-art textural anomaly detection results. Justification for inclusion and exclusion in each method chapter of this thesis is identical to that explained for the DAGM07 dataset in the previous section.

2.2.8 MVTecAD

The anomaly detection dataset released by MVTec (MVTecAD) [11] focuses on applications in industrial optical inspection, similar to DAGM07, but this dataset provides real visible-band images of macro-scale textures and objects. This dataset consists of five texture classes and ten object classes, with each class split into defect-free samples for training and defective samples for testing.

Figures 2.8 and 2.9 depict one defective sample from each texture class and object class respectively. There are a range of defects present in each class' test set. For example, the grid test set contains images where foreign material of either metal or thread is present on the grid and images where the grid is either bent (as shown),

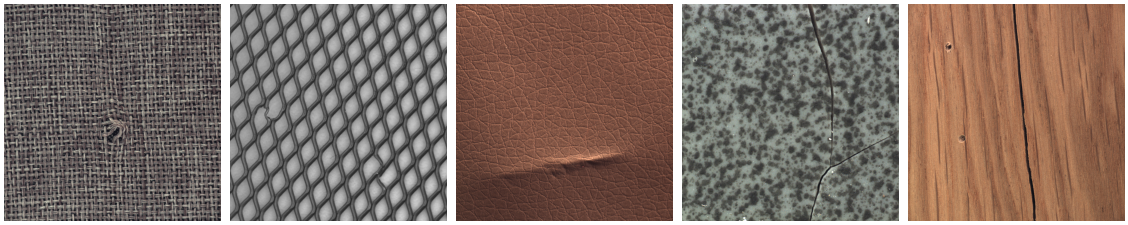

Figure 2.8: One example defective image from each of the five texture classes in the MVTecAD dataset: carpet, grid, leather, tile and wood.

Figure 2.9: One example defective image from each of the ten object classes in the MVTecAD dataset: bottle, cable, capsule, hazelnut, metal nut, pill, screw, toothbrush, transistor, and zip.

broken, or contaminated with glue. The test set for each class typically contains four or five different types of defect along with some defect-free examples. The images represent a good variation of each type of defect ranging from obvious to subtle.

This dataset is the main dataset we use to evaluate both of our textural defect segmentation methods (Chapters 4 and 5) since the dataset aligns strongly with our particular objectives within anomaly detection, it is high-quality, it provides accurate pixel-level ground truth, and it is widely used for evaluation on other methods [11, 12, 63].

2.2.9 Conclusion

We have examined the datasets most frequently used to evaluate the performance of anomaly detection methods. Broadly, we covered two main classes of dataset: the surveillance and texture style datasets and highlighted the similarities and differences among the datasets within each class. We conclude with a reflection on

the anomalies occurring in these datasets with a view to stimulating consideration over the datasets we use and their potential merits and drawbacks.

Overexplored Anomalies

While there is nothing wrong with using pedestrians to represent normality and bikes to represent abnormality, it is not good that this is always the case. There needs to be a variety of contexts because bikes are not inherently anomalous any more than pedestrians are inherently normal. If the scene was of a cycle path, then we would require the pedestrians to be labelled as anomalous and cyclists as normal. Since most anomalies are bikes, an algorithm could achieve a reasonable AUC score even if that is the only anomaly it recognises.

Composite Anomalies

Many of the anomalies exhibited in test sets are anomalous for multiple reasons. Take the most common anomaly as an example; that of a bike being ridden through a pedestrianised zone. The bike is an irregular object, since it was never observed in the training set, while the cyclist is a regular object moving at irregular speeds and potentially with irregular paths. Since these anomalies may be split into several anomaly classes in this way, it is possible that only a subset of these actually trigger anomaly detection. For example, it is not clear if sitting on a stationary bike would trigger anomaly detection, since this does not occur in the datasets. This is a problem with anomalies that always hit multiple areas of an anomaly taxonomy simultaneously. It would be better if there was a mix of pure and composite anomalies, to test algorithms on a wider variety of events.

Labelling anomalies

Considering the content of these datasets, it seems reasonable to assume that not all false-positives are equal. Some false-positives may be reasonable if the training set did not adequately cover the space of normality, or if the falsely detected anomalies truly are anomalous in some sense, even if the human annotators considered them to be uninteresting. It's possible for a competent anomaly detector that only gives

reasonable false-positive detections to achieve an equal AUC score to an incompetent anomaly detector.

Scale of normal and anomalous events

Every surveillance style dataset defines normality as pedestrians walking, while anomalous events are typically vehicles moving, items being thrown or people moving quickly. A consequence of this is that anomalous events are often larger in scale or faster than normal events. Our datasets could expand the concept of normality to include larger and faster objects like cars, joggers and cyclists. Likewise, test sets could exhibit anomalies that are subtle. If both of these changes are made, then the scales and speeds of normal and anomalous events will have a much greater overlap.

To see why this is important, assume a poor anomaly detection algorithm that assigns anomaly scores purely based on the size and speeds of certain patterns of pixels. Such a detector could achieve a high AUC score since there is a correlation between anomalous events and the events that the detector assigns high anomaly scores to. In contrast, to achieve a high AUC score on a dataset where scales and speeds are better merged, an algorithm must learn to suppress anomaly scores for normal events that have a very high impact on the pixel values, while flagging anomalies that only make subtle changes to the pixel values.

2.3 Anomaly Detection Methods

This section reviews a range of modern methods for anomaly detection arranged by category based loosely on the kinds of techniques they employ. Within each category, the methods are reviewed in chronological order. Anomaly detection performance is reported in Tables 2.4 and 2.5 for the surveillance style datasets at the frame-level and pixel-level respectively and Table 2.6 for the texture style datasets. Where methods present several variants, we report the best performing overall variant.

For the texture style datasets, these are literal pixel-level evaluations in that each individual pixel across the test set generates either a true-positive or false-positive result for the ROC curve. For the UCSD and Avenue datasets these are

Table 2.4: Frame-level AUC scores achieved by anomaly detection methods on surveillance style datasets. Results marked with an asterisk were obtained on the Avenue17 subset (Section 2.2.4).

Method	Subway		UMN	UCSD		Avenue	Shanghai
	Ent.	Exit		Ped1	Ped2		
Prediction							
U-Net [59]	-	-	-	0.83	0.95	0.85	0.73
Reconstruction							
AMDN [104]	-	-	-	0.92	0.91	-	-
ConvAE-Has [36]	0.94	0.81	-	0.81	0.90	0.70	-
ConvAE-Rib [81]	-	-	-	0.57	0.85	0.77	-
STAE-Zhao [110]	-	-	-	0.92	0.91	0.77	-
STAE-Chong [18]	0.85	0.94	-	0.90	0.87	0.80	-
Deep CNN Features							
Recount [40]	-	-	-	-	0.92	0.90*	-
Dictionary							
150-FPS [62]	-	-	-	0.92	-	-	-
BSD [78]	-	-	-	0.71	-	-	-
TSC [64]	-	-	-	-	0.92	0.82	0.68
NMC [45]	0.92	0.95	0.99	-	-	0.88*	-
No Training							
Del [22]	-	-	0.91	-	-	-	-
Unmask [44]	0.71	0.86	0.95	0.68	0.82	-	-
Other							
GPR [17]	0.93		-	0.84	-	-	-
SFA [42]	-	-	0.97	-	-	-	-
LSH [109]	-	-	0.99	-	-	-	-

pseudo pixel-level evaluations in that each *frame* generates either a true-positive or false-positive result based on anomalous pixel coverage (Section 1.2). Methods may be omitted if they did not provide any pixel-level results nor frame-level results for either the Subway, UMN or Shanghai datasets.

Table 2.5: Pixel-level AUC scores achieved by anomaly detection methods on surveillance style datasets.

Method	UCSD		Avenue	Shanghai
	Ped1	Ped2		
Reconstruction				
AMDN [104]	0.67	-	-	-
Deep CNN Features				
Recount [40]	-	0.89	-	-
Dictionary				
150-FPS [62]	0.64	-	-	-
BSD [78]	0.56	-	-	-
NMC [45]	-	-	0.94	-
No Training				
Unmask [44]	0.52	-	-	-
Other				
GPR [17]	0.63	-	-	-
LSH [109]	0.77	0.90	-	-
Gas [93]	0.65	-	-	-

Table 2.6: Pixel-level AUC scores achieved by anomaly detection methods on texture style datasets.

Method	Carpet	Grid	Leather	Tile	Wood	Mean
Reconstruction						
BergAE (L_2) [12]	0.59	0.90	0.75	0.51	0.73	0.70
BergAE (SSIM) [12]	0.87	0.94	0.78	0.59	0.73	0.78
VEVAE [60]	0.78	0.73	0.95	0.80	0.77	0.81
SMAI [56]	0.88	0.97	0.86	0.62	0.80	0.83
P-Net [63]	0.57	**0.98**	0.89	**0.97**	**0.98**	0.88
VAEgrad [21]	0.74	0.96	0.93	0.65	0.84	0.82
Generative Adversarial Networks						
AnoGAN [85]	0.54	0.58	0.64	0.50	0.62	0.58
GANomaly [5]	0.70	0.71	0.84	0.79	0.83	0.78
Deep CNN Features						
CNN Feats. [71]	0.72	0.59	0.87	0.93	0.91	0.80
Other						
GMM [13]	0.88	0.72	0.97	0.41	0.41	0.68
FCDD [61]	**0.96**	0.91	**0.98**	0.91	0.88	**0.93**

2.3.1 Future Frame Prediction

Videos introduce a temporal dimension to the problem: even if each individual frame appears to be normal, there may be irregular motions that are only apparent over a sequence of frames. Analysis of sequential inputs often relies on Recurrent Neural Networks (RNN); however, training an RNN can be challenging due to the vanishing gradient and exploding gradient problems [75]. Long Short Term Memory (LSTM) [41] is a class of RNN that has mostly eliminated the vanishing gradient problem and has proven to be very successful at tasks involving sequential inputs.

In future frame prediction, input frame sequences are mapped to output frame sequences. The general problem of mapping sequences to sequences had a long evolution before it was applied to frame prediction. Graves et al. [34] introduced depth into the LSTM network to generate very long and realistic sequences of text and handwriting. Sutskever et al. [94] extended this work so that the input and output sequences could be of arbitrary length. Srivastava et al. [92] applied this to predicting future frames, and showed that the accuracy of the output sequence could be improved by adding a branch that simultaneously reconstructs the input sequence.

Up to this point, all of the models used for mapping sequences to sequences were fully-connected; Xingjian et al. [103] argued that this fully-connected structure discarded valuable spatial information, and so they proposed a convolutional LSTM network based on Srivastava's architecture. Other researchers have applied this convolutional LSTM architecture to anomaly detection in videos. In particular, Medel et al. [69] use convolutional LSTM to predict future frames, and use the accuracy of prediction to measure the normality of the input frames.

When predicting future frames, it is common to obtain blurry images [92]. Mathieu et al. [68] explain that using the ℓ_2 loss function during training can be responsible for this, since this loss function is minimised by predicting images that are averages over all probable outcomes. Using the ℓ_1 loss function has similar consequences, but is not as severe. To alleviate this problem, they suggest two new and complementary loss functions that may be used alongside the ℓ_p loss:

- **A gradient difference loss function.** The *gradient* considered is the rate of change in pixel intensity within a single frame. This gradient is calculated at a pixel by considering its change with respect to the pixels to the left and below it. The *difference* considered is between the target and predicted images. Equation 2.2 gives the loss for a single frame:

$$L_{gdl}(\boldsymbol{Y}, \hat{\boldsymbol{Y}}) = \sum_{i,j} \left(\left| |\boldsymbol{Y}_{i,j} - \boldsymbol{Y}_{i-1,j}| - |\hat{\boldsymbol{Y}}_{i,j} - \hat{\boldsymbol{Y}}_{i-1,j}| \right|^{\alpha} + \left| |\boldsymbol{Y}_{i,j} - \boldsymbol{Y}_{i,j-1}| - |\hat{\boldsymbol{Y}}_{i,j} - \hat{\boldsymbol{Y}}_{i,j-1}| \right|^{\alpha} \right) \tag{2.2}$$

  Where $\boldsymbol{Y}$ is the ground-truth frame, $\hat{\boldsymbol{Y}}$ is the predicted frame, and α is a parameter. The variables i and j index pixels within the images. This loss function works with the ℓ_p loss, since the ℓ_p loss encourages the model to predict images that are close to the ground-truth, while the gradient difference loss encourages the model to predict images that are sharp where the ground-truth images are sharp.

- **An adversarial loss function.** The model needs to invent a plausible sequence of future frames $\hat{\boldsymbol{Y}}$ given a sequence of previous frames $\boldsymbol{X}$. In this respect, the model can be considered a generator in a conditional GAN; one only needs to add a discriminator $D(\boldsymbol{X}, \boldsymbol{I})$ to predict a confidence level in the range $[0, 1]$ that the sequence of images $\boldsymbol{I}$ truly does follow the sequence $\boldsymbol{X}$. The discriminator network should be able to easily identify blurry images as generated, thus the generator network is penalised for producing such images. Only when the generator produces plausible, sharp images will the discriminator network begin to have difficulty. During training, the generator predicts a sequence of future frames $\hat{\boldsymbol{Y}}$ conditioned on a sequence of previous frames $\boldsymbol{X}$. The discriminator is shown the same set of previous frames, plus a sequence of future frames that is either the ground-truth future frames $\boldsymbol{Y}$, or the generated future frames $\hat{\boldsymbol{Y}}$. The adversarial loss function encourages the generator to predict images that the discriminator classifies as originating from the ground-truth distribution:

$$L_{adv}(\hat{\boldsymbol{Y}}) = L_{MSE}(D(\boldsymbol{X}, \hat{\boldsymbol{Y}}), 1) \tag{2.3}$$

The adversarial loss function also works with the ℓ_p loss, since the generator may learn to produce images that fool the discriminator despite being far away from the ground-truth. If the generator attempts this, it will be penalised by the ℓ_p loss. Of course, the discriminator is also trained, but the details of this are not included since it is the generator that is of interest here.

Mathieu et al. [68] use their new loss functions to significantly improve future frame prediction beyond what can be achieved using the ℓ_p loss function alone.

Liu et al. [59] build on the work of Mathieu et al. [68] and apply it to anomaly detection. To generate future frames, they use a variation of the U-Net architecture [82]. These frames are evaluated as being real or fake by a discriminator network based on the patch discriminator of Isola et al. [46], thus forming the basis of the adversarial loss. Isola et al. [46] use a patch discriminator because they note that the ℓ_p loss is good for training models to produce images with the correct low frequency structure, and fails only at allowing models to produce images with the correct high frequency structure. Therefore, the GAN only needs to address the high frequency structure, which is facilitated by only discriminating small patches of the generated images.

GAN requires an input noise vector, conditional GAN requires an additional input on which to condition the output. Mathieu et al. [68] and Isola et al. [46] both found that conditional GAN tends to ignore the noise input, and may be omitted. The consequence of this is that the stochastisity of the output of a conditional GAN is limited, or is entirely deterministic when the noise is omitted. Liu et al. [59] do indeed omit the noise; however, stochastisity is not required. In addition, they introduce an optical flow loss to include a temporal consideration in the evaluation of predicted frames. They use Flownet [27] to compute the optical flow, which is applied twice: once to calculate the optical flow between the last seen frame and the *predicted* next frame, and once to calculate the optical flow between the last seen frame and the *actual* next frame. The difference between these forms the flow loss.

2.3.2 Reconstruction

Typically, the aim of reconstruction methods is to achieve the following:

- Learn an encoding function that compresses an input down to a smaller feature vector.
- Learn a decoding function that reverses the compression, such that normal inputs are reconstructed very accurately while anomalous inputs are reconstructed poorly.
- Use the accuracy of reconstruction as a measure of normality.

The encoding and decoding functions are usually learned by training an autoencoder on a dataset of normal examples.

Xu et al. [104] introduce a method they called Appearance and Motion Deep Net (AMDN). AMDN uses three separate stacked denoising autoencoders to learn three different representations of patches taken from normal video frames. The first learns to encode the original patches, the second learns to encode the corresponding optical flow patches, and the last learns to encode both patches fused together along the channel dimension. Each of the three encodings is used to train a separate one-class SVM. Testing a sample frame entails passing image, optical flow and fused patches to the respective encoders, and feeding the resulting encodings into the three one-class SVMs, each of which makes a contribution towards determining whether or not the input sample is anomalous.

Hasan et al. [36] use a convolutional autoencoder to learn the regularities in a set of training videos. Their convolutional autoencoder attempts to reconstruct a stack of single-channelled input frames, where the frames are selected via a sliding window over the current video. The input is compressed through a series of convolutional and pooling layers, and then uncompressed through a series of transposed convolutional and unpooling layers to form the reconstructed version of the input. The ℓ_2 difference between the original stack and its reconstruction forms the loss function during training; while during testing, it forms the basis of a regularity score. Their justification for using a convolutional autoencoder was that a regular

fully-connected autoencoder loses spatial information, whereas convolutional layers produce feature maps that preserve spatial information. We refer to this method as ConvAE-Has.

A similar approach is taken by Ribeiro et al. [81]; however, they only use *single* frames as input rather than a stack of sequential frames. Consequently, their model does not have access to the same temporal information. They experiment with fusing the input frame with features from either the Canny Edge Detector, Optical Flow or both. Where Optical flow is used, this reintroduces the temporal information missing from the single-frame approach. Curiously, introducing temporal information in this way most often weakens performance, while the combinations with only spatial information outperform ConvAE-Has [36] on the Avenue dataset. Ribeiro et al. suggest that for this dataset, temporal information is not as relevant as the spatial information for identifying the anomalies. Performance on the UCSD Ped1 dataset is weak compared to all other methods mentioned. We refer to this method as ConvAE-Rib.

The methods previously mentioned use two-dimensional convolutions, even when operating on spatio-temporal inputs that are three-dimensional. This causes temporal information to be lost, since a neuron in a feature map connects to all channels. Zhao et al. [110] address this weakness by moving to three-dimensional convolutions, resulting in what they called a Spatio-Temporal AutoEncoder (STAE). This was inspired by the three-dimensional convolutions used to increase performance on action recognition problems [97, 47], which also involve recognising patterns of motion. Following Srivastava et al. [92], their method also entails training the model to predict future frames concurrently with reconstructing the current frames. By forcing the model to learn to predict future frames, it needs to attend to the regular motions of objects in addition to the appearance of the frames. On the Avenue dataset, this approach is very similar to that of ConvAE-Rib; however, high performance is maintained across *both* the UCSD Ped1 and Ped2 datasets, where ConvAE-Rib only achieves good results on the UCSD Ped2 dataset. We refer to their method as STAE-Zhao.

Chong et al. [18] address the loss of temporal information by applying two-

dimensional convolutions over *individual* frames, rather than a stack of frames as done by Hasan et al. [36]. This results in a sequence of feature maps that summarises the spatial information, while avoiding collapse of the temporal information. Following this extraction of spatial features, the sequence is fed into a convolutional LSTM [103] to summarise the temporal information. A mirror image of this network attempts to reconstruct the input video volume in a similar way to the above methods. We refer to this method as STAE-Chong.

Bergmann et al. [12] apply a simple CAE architecture on their proposed MVTecAD dataset [11]. Since this dataset consists of images rather than videos, they do not need to consider any special treatment of the temporal information as discussed in the above methods. They perform experiments using two variants: one trained using the ℓ_2 distance between the input and reconstruction and one trained using a distance based on the SSIM measure [101]. Similar to the findings of Mathieu et al. [68] described above, they find that using the ℓ_2 distance can form blurry reconstructions that hinder performance. They obtain significantly better results using the SSIM distance. We refer to their autoencoder architecture as the Bergmann autoencoder or BergAE. Bergmann et al. [11] also test a range of contemporaneous methods [85, 71, 13] on their dataset, which we review in their respective categories.

Since Bergmann et al. [12], other researchers have tested new methods on the MVTecAD dataset. Luo et al. [63] develop the P-Net architecture consisting of structure extraction and image reconstruction modules. The image reconstruction module takes input not only from the original image, as in previous methods, but also from the extracted structure. In addition to comparing input images with their reconstruction, they also extract structure from the the reconstructed image and compare with that from the original image to give an additional measure of abnormality. Li et al. [56] propose a method they refer to as Superpixel Masking and In-painting (SMAI). This method is a variation on the theme of reconstruction methods presented so far. They mask out a randomly selected superpixel from each training instance and train the PEN-Net image in-painting network [108] to restore the instance. When processing a test instance, superpixels are masked out in turn and the trained PEN-Net model restores them. Inaccuracies in the restoration

measure the abnormality of the superpixel region. Dehaene et al. [21] propose improving autoencoder reconstruction by projecting input test samples onto the learned normal data manifold via iterative gradient descent. This overcomes the difficulty of the autoencoder to reconstruct high-frequency information and improves performance on a range of autoencoder variants. They obtain their best results using a VAE-based variant (VAEgrad); therefore, we use the performance of this variant for comparison in results tables in this thesis.

2.3.3 Generative Adversarial Networks

When applied to anomaly detection, GAN are trained on the normal distribution of data such that the outputs of the generator function are on the manifold of normal data. In the AnoGAN method for anomaly detection, Schlegl et al. [86] use the DCGAN [76] architecture to generate a sample on the normal manifold that has a minimal distance to some test sample. Unfortunately, since the generator of the DCGAN framework is unable to map from the data space to the latent space, the generated sample must be found through a costly iteration over latent space vectors, which makes the inference phase very computationally expensive. In a subsequent work they introduce a fast version of AnoGAN called f-AnoGAN [85] that side-steps the requirement to iterate over latent-space vectors by training an encoder network alongside the generator and discriminator. They test the BiGAN [25] methodology for training the encoder as well as introducing two new methodologies that train the encoder in a separate step, making use of the pretrained generator network. In addition, they substitute WGAN [8] in place of the DCGAN [76] architecture. They demonstrate improved anomaly detection and segmentation performance when using their encoder training methodologies.

Concurrently with f-AnoGAN, Akçay et al. [5] improve upon AnoGAN using a novel encoder-decoder-encoder pipeline they call GANomaly, which yields significantly better performance in terms of both anomaly detection and computational cost. The first encoder maps an input image to a latent-space vector and the decoder generates an image by mapping the vector back to the input space. The second encoder maps the generated image back again to the latent space. Meanwhile, a

discriminator network is employed as in the vanilla GAN framework to discern real input images from generated ones. Three losses are used to train the architecture: A context loss measures the distance between real and generated images so that the encoder-decoder architecture generates images with the correct content; an encoder loss measures the distance between the two latent vectors so that both encoders learn a similar mapping to the latent space; and an adversarial loss is used as in the vanilla GAN framework to train the encoder-decoder architecture to generate realistic looking images, except that the training signal is formed from the distance between features extracted from the discriminator rather than the discriminator output. At inference time, the anomaly score is calculated via the distance between the two latent space vectors formed from the test input. Since the generated image should be missing the anomalies present within the test input, so too should the final encoding vector be missing the anomalous information. Akçay et al. [6] later improve GANomaly via the introduction of skip connections in the architecture, resulting in Skip-GANomaly.

2.3.4 Deep Learned Features

The anomaly detection methods reviewed in this section use CNN architectures that have been pretrained on large scale image datasets such as ImageNet [23], leveraging the transferability of the features extracted from such networks [105] and the generality of the imagery that these networks have been exposed to.

Recounting of anomalies requires explaining why a particular input is anomalous. Hinami et al. [40] investigate joint anomaly detection and recounting by adapting The Girshick Fast R-CNN architecture [29] (Section 2.1.3). They extend this network by adding two branches: one for attribute labels and one for action labels. Attributes are adjectives that describe the object, such as blue, tall, old and broken; while actions are verbs such as bending, climbing and sitting. The class, attribute and action labels are simultaneously trained under the multi-task paradigm such that the network layer common to these three outputs must encode high-level semantic information about all three tasks. They learned regularities based on these high-level semantic features using KDE (Section 2.1.1). When their anomaly detector flags an

input as anomalous, they are able to recount that anomaly by examining the network output branches. From these, they can determine which object in the scene caused the anomaly, and describe its attributes and action. Interestingly, they achieve their best result (as displayed in Tables 2.4 and 2.5) when using a variant trained only on the action labels.

Napoletano et al. [71] extract features from a ResNet-18 model [38] pretrained on images from the ILSVRC 2015 challenge [83]. In the training phase, they extract features from normal image tiles and form a dictionary of normal features by clustering with the KMeans algorithm. In the inference phase, the Euclidean distances between a given test tile feature representation and those of the m most similar members of the dictionary are averaged to form an abnormality score.

2.3.5 Dictionary Learning and Sparse Coding

Sparse coding is the process of finding a vector $\boldsymbol{x} \in \mathbb{R}^n$ that constructs a point $\boldsymbol{y} \in \mathbb{R}^n$ given a dictionary $\boldsymbol{D} \in \mathbb{R}^{n \times m}$ and a sparsity constraint as accurately as possible [32]. The vector $\boldsymbol{x}$ is said to be a *sparse representation* of $\boldsymbol{y}$. To arrive at this sparse representation, one would like to minimise the construction error under an ℓ_0 constraint; however, Cheng et al. [16] explain that this minimisation is NP-hard. Instead, the minimisation may be performed under a relaxed ℓ_1 constraint, which is equivalent under certain conditions [16]. Ren et al. [79] explain that the ℓ_1 minimisation benefits from the existence of efficient algorithms, and well-developed theory. The most common ℓ_1 minimisation used is that of Equation 2.4.

$$\hat{\boldsymbol{x}} = \arg\min_{\boldsymbol{x}} \frac{1}{2}||\boldsymbol{D}\boldsymbol{x} - \boldsymbol{y}||_2^2 + \lambda||\boldsymbol{x}||_1 \tag{2.4}$$

This approach tries to balance minimising the construction error $||\boldsymbol{D}\boldsymbol{x} - \boldsymbol{y}||_2^2$ with minimising the ℓ_1 norm of the representation. Even if a point $\boldsymbol{y}$ may be constructed perfectly from atoms in the dictionary, doing so might require many atoms, thus pushing up the penalty imposed by the ℓ_1 norm regularisation.

In sparse coding based anomaly detection methods, the space $\mathbb{R}^n$ is the input space. Each point in this space may be a frame, an image patch, a spatio-temporal

cuboid, or some feature vector extracted from any of the above. Dictionary methods are typically agnostic to the type of representation used; however, the choice of representation may affect the performance. To apply sparse coding to anomaly detection, one can measure the normality of a test sample $\boldsymbol{y}_i$ by the sparsity of its sparse representation $\boldsymbol{x}_i$ [20, 62]. The dictionary $\boldsymbol{D}$ needs to be learned to facilitate this. It is common to find the dictionary that minimises Equation 2.4 over all training samples $\boldsymbol{y}_i$; thus, one would like to take the arg min over both $\boldsymbol{x}$ *and* $\boldsymbol{D}$. While this problem is non-convex, each minimisation on its own *is* convex, and so one normally proceeds via alternately minimising over $\boldsymbol{x}$ and $\boldsymbol{D}$ while keeping the other fixed [62]. Once the dictionary is found, it can be used as a constant in future instances of Equation 2.4 as previously described.

Cong et al. [20, 19] proposed the Multiscale Histogram of Optical Flow (MHOF) algorithm as a feature extraction mechanism. These features extracted from the training videos form a very large feature pool. The feature pool is pruned to reduce the number of noisy and redundant features by searching for a minimal subset that can best recreate the entire pool via their sparse linear combinations. The surviving features form the atoms of a dictionary. The learned dictionary is used to derive sparse representations of test features using a method very similar to Equation 2.4; however, Cong et al. add a weight to each atom that affects the relative cost of using the atom in representations. The weight is determined by the frequency of the atom in the training pool, so that when a representation uses an atom that is very common, it is penalised less than when it uses an atom that is rare.

A problem with sparse coding methods is that it takes a lot of computation to find the optimal subset of atoms from which to form the sparse representation [62, 64]. There are many possible combinations of s atoms drawn from m atoms to search through, considering the high dimensionality n of the input space, and the requirement of $m >> n$ atoms in the dictionary. To improve the performance of sparse coding methods, Lu et al. [62] proposed learning a set of optimal combinations of atoms in advance, and only searching through those at test time. In 2013, they achieved very fast performance of 150 FPS on a regular desktop computer running MATLAB, without significantly compromising the detection accuracy. This work

introduced the Avenue dataset (Section 2.2.4).

Ren et al. [78] learn a dictionary whose atoms are grouped into *behaviours*. Rather than reconstructing samples from any combination of atoms, only combinations from a single behaviour may be used. They call this a Behaviour Specific Dictionary (BSD). They use the K-SVD algorithm [4] to yield an initial dictionary that satisfies $||\boldsymbol{y} - \boldsymbol{D}\boldsymbol{x}||_2 \leq \epsilon$ subject to $||\boldsymbol{x}||_0 \leq s$ followed by spectral clustering [72] to separate the atoms into behaviour groups. This technique respects the fact that many normal behaviours are possible, and that a new example of any of these behaviours should be constructible using only a representative sample of the same behaviours. In contrast, previous techniques allow the mixing of atoms from diverse behaviours.

Luo et al. [64] added temporal coherency to this class of anomaly detection method by enforcing that neighbouring frames have similar sparse coefficients if the frames themselves are similar; they called this Temporally-coherent Sparse Coding (TSC). They showed that TSC can be interpreted as a stacked RNN, which increases the speed of the algorithm during the testing phase, and facilitates optimisation of parameters simultaneously. Using TSC directly would require choosing hyper-parameters to optimise the reconstruction coefficients. This work introduces the ShanghaiTech Campus dataset (Section 2.2.5).

Ionescu et al. [45] use Narrow Motion Clusters (NMC) to detect anomalies. They extract features from spatio-temporal cubes, cluster them using k-means, discard outlying clusters, and then shrink each cluster using an SVM. These narrowed clusters form the dictionary.

2.3.6 Anomaly Detection Without Training

The methods reviewed in this section do not use the training set. They are automatically ready for deployment on the test set without any previous exposure to the dataset distribution. This requires a shift in our definition of an anomaly. Up to this point, anomalies have been defined by the training set; but for these methods that do not see the training set, anomalies are defined by their *discriminability* i.e. how easily a testing sample can be separated from the complete corpus of testing

samples. Similar to the sparse coding methods, the test samples are usually some features extracted from the source material, rather than the source material itself.

Del et al. [22] propose a method in which a sliding window passes over the features extracted from every frame of a test video. All features inside the window are assigned the anomalous label, while all those before the window are assigned the normal label. Features after the window are not used. Note that these labels are assigned purely on the basis of whether or not the samples happen to be inside the window at its current position; there is no regard for the truth of these labels. A simple logistic regression classifier is trained on these labelled features. Following this training, the classifier is applied to each feature inside the window and its confidence that the feature is anomalous is used as a measure of the discriminability of that feature. If a feature lies close to the decision boundary and is classified with a low confidence of near 0.5, then it is assumed that feature is not easily separated.

As described, this method performs change detection. To extend to anomaly detection, the features are shuffled prior to running the sliding window over them. The features are repeatedly shuffled, and the process is repeated for each shuffle. An average discriminability is used as a measure of abnormality. On the Avenue dataset, the ROC curves produced by this method were very similar to those produced by the sparse coding method of Lu et al. [62].

Ionescu et al. [44] have a similar approach, but they only consider observations inside the window. Within the window, they label the first half of the frame features as normal and the remaining ones anomalous. They then perform unmasking [50] on these features, i.e. they repeatedly train a classifier on the features inside the window and then remove the most easily discriminated features. They track the classification accuracy across the iterations. Normality for the second half of the window is measured by the extent to which accuracy drops. Their reported performance on the Avenue dataset is omitted from our results tables since they only used five of the dataset videos.

Andrews et al. [7] use a Forest of Random-Split Trees (FRST) as an anomaly detector. Each tree in the forest is produced from the features extracted from every test image. The test images are split into two groups based on a randomly selected

feature and a randomly selected value of the feature. The same process is applied to each of the two groups to produce four groups, and so on until every feature ends up in a group on its own, becoming a leaf on the tree. This produces a single tree. The forest is produced by repeating this many times. Those features which are easily separated will tend to form leaves early on in this process, whereas those features that are similar to many others will need to wait longer to be separated, and will form leaves only at a much deeper part of the tree. The depth of leaf is then used as a measure of normality.

2.3.7 Other Methods

The remaining methods reviewed in this chapter cannot easily be grouped with others or each other and so we review each of them here in isolation. The justification for their inclusion is that they have performed exceptionally well on at least one dataset and together they represent the wider range of anomaly detection methods.

Cheng et al. [17] employ Gaussian Process Regression (GPR) to make inferences about the likelihood of anomalies in videos. They first locate Spatio-Temporal Interest Points (STIP) in an input video, and then extract features from each STIP using an empirically chosen method that depends on the nature of the input. During the training phase, these extracted features are clustered using the K-means algorithm to form a visual vocabulary of normal features. A feature has a *pattern similarity measure*, which is the k-nearest neighbours distance of the feature to the visual vocabulary. Applying a threshold on the pattern similarity measure allows the detection of abnormal events. However, they were also interested in the interactions between events that would allow them to detect global anomalies. To achieve this they form a code book of normal interaction templates between local STIP points as follows. During training, they use a three-dimensional sliding window over the input video. For each window, they locate the STIP points, extract the features, and derive their pattern similarities. Each window is used to form an ensemble, consisting of the relative location, pattern similarity and matched code words of each STIP in the window. After filtering to remove bad ensembles, they cluster the ensembles to form the code book. Gaussian Process Regression is used to learn a model of normal

interaction templates that maps relative location to pattern similarity.

Hu et al. [42] use five layers of Slow Feature Analysis (SFA) nodes to extract slowly varying features from input videos. Each SFA node in a layer receives input from a spatial patch within the layer below, where the spatial patches of neighbouring nodes half overlap in both directions. A training set is used to determine how the features should be extracted. To detect anomalies in a test video, they calculate the sum of square derivatives of each SFA node output. This is a measure of how quickly the output features change, which of course should be a very small measure if the input data is similar to that seen during training. This is similar to the reconstruction methods seen above, where if input $\boldsymbol{x}$ is similar to those seen during training, then the auto encoder should be able to compress the input and reconstruct it accurately; likewise in this method, if input $\boldsymbol{x}$ is similar to those seen during training, then the model should be able to succeed in extracting slowly varying features. Since each node is only affected by a certain patch within the input video, anomalies can be simply localised.

Zhang al. [109] use Locality-Sensitive Hashing Filters (LSHF). HOF features are extracted from spatio-temporal cuboids in the training videos to produce the training examples $\boldsymbol{x}_i \in \mathbb{R}^d$. A hashing function maps an example to an integer and a sequence of such functions maps an example to a sequence of integers. This sequence of integers is the hashing bucket for a given example. Training points that are close to each other are likely to be mapped to the same bucket. In this way, the hashing process serves a similar function to the clustering that has been observed in previous methods. The centre and radius of each bucket is calculated by considering the points that fell into it. The centre of a bucket is the average position of the all points that fell into it, while its radius is determined by the furthest point from that centre. A test sample is hashed into a test bucket using the same hashing functions. The abnormality degree of the test sample is calculated by considering the distance of the test sample to the centre of the training bucket nearest the test bucket. This distance is compared with the radius of that training bucket to obtain the abnormality degree.

Bottger et al. [13] build a Gaussian Mixture Model for texture anomaly detection

from normal patches using maximum likelihood. The model estimates the probability of test-time patches to give a normality score. Using larger patches is beneficial for allowing the model to capture long-range structure, but this results in a quadratic increase in computational cost and also more redundancy. They therefore propose a texture compression scheme to convert patches into features to reduce redundancy and computational cost.

Liu et al. [60] attempt to Visually Explain VAE models (VEVAE) by producing attention maps via back-propagation of the latent vector to the last convolutional layer. Subsequently, they show that these attention maps are effective at localising anomalies in images.

Liznerski et al. [61] apply one-class classification on features extracted from a Fully Convolutional Network (FCN). This requires train-time access to samples not belonging to the set of normal inputs, but rather than using anomalous samples (which would not be in the spirit of anomaly detection), they use images from publicly available large-scale datasets or synthetically generated images. The FCN is trained such that for normal inputs, the loss is high when the network produces features that are far away from some centre $\boldsymbol{c}$ in the output space and vice-versa for inputs from the external dataset. Since the network is entirely convolutional, spatial information is preserved. The output feature map can therefore be upscaled to generate an anomaly heat map in the input space. They call their method Fully Convolutional Data Description (FCDD).

2.3.8 Conclusion

Some methods make use of features extracted from networks that have been pre-trained on large scale image datasets [40, 71]. The appeal of these methods in an anomaly detection setting is that having been trained on a wide variety of image data, the networks can be expected to have learned rich features for discerning classes with wide-ranging appearances [71]. Since in anomaly detection we only have access to normal data and aim to detect an unbounded range of variations from that normal data, leveraging these classification networks as feature extractors is desirable. Methods from this class of anomaly detection technique have the

advantage that they can be deployed very quickly after having seen the normal data, since there is no need to perform forward and backward propagations to train the model; however, they can be very slow at inference time if they have been exposed to a large quantity of normal data [40]. In Chapter 3, we propose a real-time method of anomaly detection based on the gathering of normal features extracted from a pretrained network. This is achieved by gathering features more sparingly, improving the quality of the collection while, most crucially, reducing its size.

At the core of many methods is a model for generating imagery based on some test input. Usually, this generation is performed by an autoencoder [36, 81, 18, 12] or by a GAN [85, 5, 6, 63]. These methods show promise because they are very effective at producing imagery with anomalies removed, and so those anomalies are readily exposed via a distance measure between the test input and the model-generated imagery. However, as some point out [68, 12], differences also occur over the normal pixels due to the poor generation of high-frequency visual details. A range of solutions are suggested to tackle this problem that either help to produce sharper images [68], or make use of a less strict distance measure that can forgive a certain amount of infidelity [12]. Nevertheless, the efficacy of this class of methods will always be bounded by the model's ability to accurately generate normal regions. In Chapter 4, we propose a method for textural defect segmentation whose ability to accurately generate normal regions is ensured by passing normal information from the input to the output directly, resulting in unsurpassed anomaly detection performance on the MVTecAD [11] texture classes.

None of the methods reviewed make use of the more recent Transformer networks [100], despite their growing popularity in the image domain [14, 74, 26]. In Chapter 5, we propose a method towards filling this gap by targeting anomaly detection through Transformer-based future pixel and future feature prediction.

CHAPTER 3

Anomaly Detection by Density Estimation

We introduce a method of anomaly detection that operates on a live-stream of video data such as obtained from a CCTV camera or a webcam. Potential applications include the automatic monitoring of CCTV for abnormal activity or the detection of foreign objects on a production line. Given this context, the primary objective is to perform anomaly detection in real-time during the inference phase. In addition to meeting this criterion, our method is able to capture and process a training live-stream of normal events in real-time and is ready to begin inference immediately after its observation. Depending on the complexity and variability of the normal live-stream, the required observation period may be very short.

Our method is based on the multi-task Kernel Density estimation (KDE) variant of the method proposed by Hinami et al. [40] whose pipeline consists of three stages: region proposal, feature extraction and KDE. We replace their traditional region proposal stage with one that utilises a more modern deep learning approach and adapt the feature preprocessing before KDE resulting in the following contributions:

- A method for classless region proposal that produces far fewer regions than traditional approaches [51, 28], resulting in faster processing in down-stream pipeline stages. When applied as the region proposal stage in the multi-task

KDE model, this facilitates:

  - Training a single density estimator to cover the entire frame space, rather than twelve density estimators that each cover a zone within the frame space.
  - The opportunity to carry more information through the feature extraction stage for enhanced density estimation, since less compression needs to be applied to the features when far fewer regions are considered.
  - Real-time performance in both the training and inference phases.

- An alternative feature preprocessing step that results in higher accuracy density estimation.

The method entails extracting features from a deep neural network pretrained on very generalised object datasets. Therefore, this method is mostly applicable to videos of scenes, as in the example applications described above, where high-level knowledge of objects is relevant. We therefore focus this chapter on the CCTV dataset UCSD Ped2 [65] where the anomalies are scene objects, rather than datasets like MVTecAD [11], where anomalies are low-level defects at the pixel level. See Chapter 6 for evaluation across all considered datasets.

This method has been integrated into NOUS, an industrial software product developed by the company COSMONiO that has since been acquired by Intel. See Appendix A for anomaly detection examples obtained while developing this method for practical applications rather than academic.

3.1 Method

The proposed anomaly detection pipeline consists of three stages: region extraction, feature extraction and density estimation.

The purpose of the region extraction stage (Section 3.1.1) is to select a number of rectangular areas within the imagery to pass to feature extraction. The region extraction stage should balance two competing criteria: to select a minimal number

of regions and to not miss regions that may have relevant content. The former reduces the computational time, facilitating real-time performance, while the latter is a prerequisite for good anomaly detection performance.

The feature extraction stage maps the visual data from each extracted region to a point in a feature space (Section 3.1.2). Features are obtained from a pretrained AlexNet network [53], which has been modified following the procedure of Hinami et al. [40] to perform multi-task classification. The seventh layer of this network forms the feature representation of a region. This layer is the final fully-connected layer before the output class logits.

Once a frame has been observed in the training phase, we store the feature representation of each region and repeat the process of region extraction followed by feature extraction for the next frame. After all frames have been observed, a KDE model is formed from all of the stored features (Section 3.1.3). This model provides an estimate of the probability density function over the feature space, which is used during the inference phase to measure the probability density of features extracted from the live video stream. A region whose feature representation has a probability density less than a certain threshold is thus determined to be anomalous.

3.1.1 Region Extraction

Hinami et al. [40] use Geodesic Object Proposals [51] (GOP) and Moving Object Proposals [28] (MOP) to extract approximately $2,500$ regions from each frame. Section 3.1.3 explains how the time performance of the density estimation stage is sensitive to the number of regions seen during the training phase. Both training *and* analysis phases are negatively affected when the number of regions extracted during training is high. To cope with this the input frames are divided into a grid with twelve cells, and a separate KDE model is trained for each.

In contrast, our proposed method extracts regions using a Faster-RCNN [80] model with a ResNet50 [38] backbone. This model was pretrained on Microsoft's Common Objects in Context (COCO) dataset [57] and obtained from the MaskRCNN-Benchmark project [67]. Typically, this method produces only tens of regions per frame, and so it is sufficient to train only a single normality model. However, in

anomaly detection we do not know in advance what objects may appear; therefore, the post-processing of the proposed regions needs to be altered to produce regions that are more scattered over the input image, yet cluster preferentially around probable targets. The following paragraph explains the specific alterations that are made to the Faster-RCNN post-processing, and an example of the regions extracted via this method is shown in Figure 3.1.

The bounding box regression head of the Faster-RCNN model produces a unique region per class from each region it receives from the earlier RPN stage, including the *background* pseudo class. The standard procedure in Faster-RCNN is to post-process these boxes on a per class basis: for each class, all boxes whose class score is below a threshold of 0.05 are discarded, and then Non-Maximum Supression (NMS) is performed on the remaining class boxes using an IoU threshold of 0.5. In contrast, we perform post-processing over all boxes together, since we are uninterested in classes. We also use a lower class threshold to bring out boxes that are not so easily recognisable as belonging to one of the COCO classes. Firstly, all boxes whose class

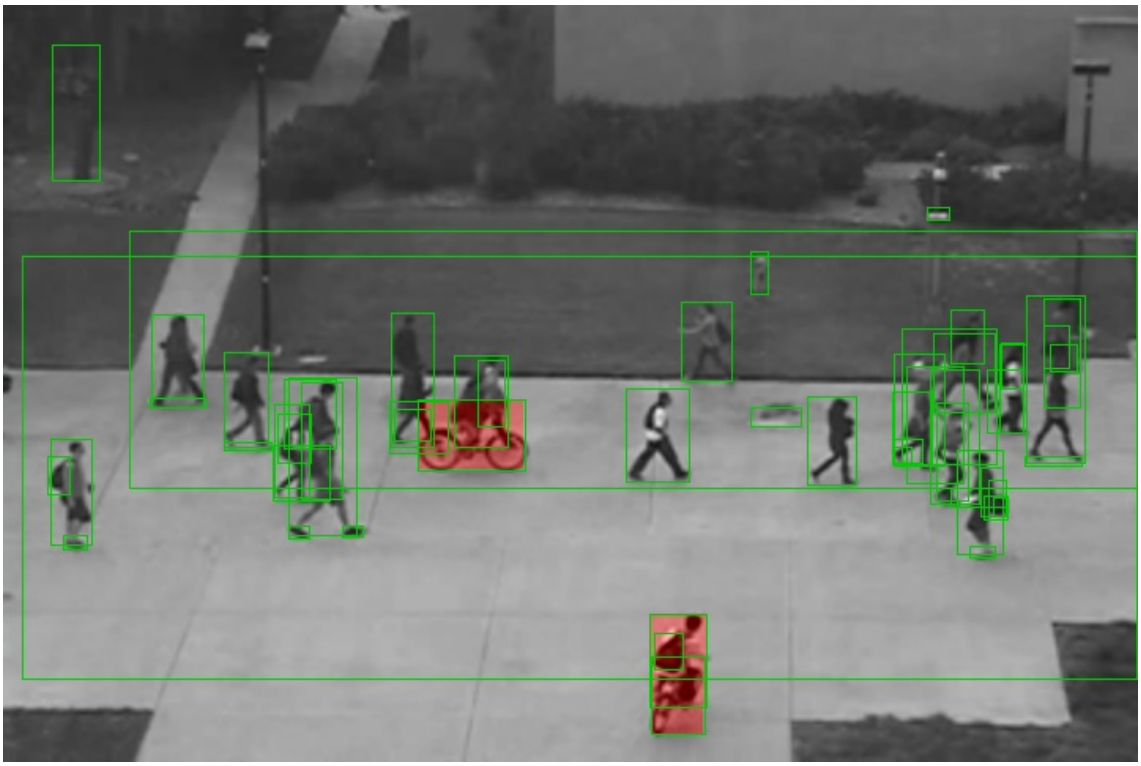

Figure 3.1: Examples of regions extracted using the Faster-RCNN model with modified postprocessing. Red regions indicate a detected anomaly.

score is lower than 0.001 are removed and then all background boxes are removed. Finally, the boxes are ordered by class score before being passed through NMS all together.

These values were chosen empirically. They were observed to produce sensible candidate regions that consistently covered the vast majority of meaningful areas in the scene while excluding background areas. In addition, for each captured entity such as a person or a bike, further regions were captured at smaller scales within the entity such as backpacks and wheels. We suspect that this is beneficial because a wide variety of high-level entities, such as a person, may collapse to within a small region in the feature space. If there is something abnormal about a backpack, perhaps this will not be apparent in the feature representation for the whole person. Changing the value of the NMS parameter will alter the degree to which these subregions are included, resulting in either a proliferation of decreasingly meaningful subregions or a reduction of subregions. This will also impact the computational performance accordingly. Changing the other values will result in more or fewer regions covering high-level entities. There is a balance to be found between covering all interesting regions and excluding non-interesting regions.

3.1.2 Feature Extraction

Hinami et al. [40] pass the regions extracted in the first stage along with the input frame to a Fast-RCNN [29] model that has an AlexNet [53] backbone. They adapt the architecture by removing the bounding box regression head and adding two new heads: one for attribute labelling and one for action labelling. Attributes are descriptive labels describing things like colour and shape, while actions include verbs such as *playing* or *flying*. These new heads are trained using the Visual Genome dataset [52], while the original object classification head is trained using the COCO dataset [57]. These heads receive activation from the seventh layer in the network, which is common to all three tasks and must therefore encode information relevant for solving all three tasks. It is expected that the activations of this layer will be sensitive to various interesting shifts in the input, making them an ideal basis for anomaly detection. Therefore, this layer is taken as the feature representation for a

region.

Our method employs exactly the same feature extraction stage; however, the feature extractor is trained using region proposals from the Region Proposal Network (RPN) of Faster-RCNN [80] rather than GOP and MOP. It is not clear whether or not Hinami et al. [40] include the Rectified Linear Unit (ReLU) activation sublayer in their representation. We perform experiments excluding the ReLU activation in our main variant and investigate the effect of including it in the analysis (Section 3.3.1). We also investigate extracting features at layer six rather than layer seven, since it is expected that these features may be more generalisable (Section 3.3.2).

3.1.3 Density Estimation

The seventh layer of AlexNet [53] taken as the feature representation consists of $4,096$ units. This gives the feature space too high a dimensionality for the density estimation stage. To solve this, Hinami et al. [40] use Principal Component Analysis (PCA) to compress the features down to sixteen dimensions. The feature vectors are then normalised before beginning density estimation.[1] We note that this procedure casts every point in the 16-dimensional feature space onto the unit sphere, which discards valuable information. Instead, we record the magnitude of the largest 16-dimensional feature vector seen during training, and scale every vector down by that magnitude. This puts every normal feature vector *inside* the unit sphere rather than on its surface, thus retaining the radial information. The magnitude used for scaling is saved for use in the inference stage. We find that this produces significantly better results than normalising (Section 3.3.3).

In one variant of their method, Hinami et al. [40] use KDE to estimate the probability density function over the 16-dimensional feature space. They use a Gaussian kernel with the band-width determined by Scott's Rule [87]. The method introduced here uses the same density estimation stage except that we produce only one KDE model rather than twelve and we scale features rather than normalising them. In addition, our less zealous region proposal system allows us to investigate

[1] This is not specified in their paper, but was observed in the associated reference implementation they supplied.

increasing the number of PCA components to discard less information (Section 3.2).

We refer to the process of collecting the features from the normal samples, compressing them with PCA and constructing a KDE instance from them as *forming a normality model*. To achieve real-time performance in both training and testing, one needs to be careful about the quantity of regions extracted. PCA scales poorly with large numbers of points when forming the normality model; however, it is very fast during inference. In contrast, KDE is very fast when forming the normality model, but if it is formed with too many normal points then its speed during inference suffers. If the number of features collected from the training stream approaches 10^5, then the performance requirements set out in the introduction to this chapter are no longer satisfied. Our method of region extraction reduces the number of extracted regions to a manageable level.

3.2 Results

We test our method on the same pedestrian based datasets as Hinami et al. [40]: the UCSD Ped2 dataset [65] at both the frame and pixel levels, and the Avenue17 dataset [62, 40] at the frame-level. Within UCSD Ped2, it is normal for pedestrians to be walking along the path, but the appearance of any kind of vehicle is abnormal (Section 2.2.3). Within Avenue17, normality is the same as in Ped2, but people running or throwing items is abnormal (Section 2.2.4). We first consider the pixel-level Ped2 evaluation.

Figure 3.2 shows examples of success and failure cases on the Ped2 dataset. Our method is effective at detecting anomalies based on their appearance, such as the bikes and the van, but skateboarders are seldom detected since they appear similar to the pedestrians. Such anomalies would benefit from a consideration of motion, requiring the method to analyse multiple frames at a time. It is interesting to note that the skateboard fails to be detected despite having its own region that is separate from the skateboarder as a whole. We can suggest two possibilities to account for this: firstly, the region is so small that any errors in the ROI pooling operation may miss the information required to detect the anomaly; and secondly, considering only

the pixels captured by the region, it is not at all clear what, if anything, the region contains. Perhaps a higher-resolution image would allow the detector to work better in this particular case.

We construct a Receiver Operating Characteristic (ROC) curve and use the Area Under Curve (AUC) evaluation metric to measure the performance quantitatively. The ROC curve is constructed following the pixel-level procedure commonly used on this dataset [65]. In this procedure, each frame is considered *positive* or *negative* depending on whether there are any positive pixels present in the frame. When analysing a positive frame, the result is true-positive when an algorithm labels at least 40% of the positive pixels as such. When analysing a negative frame, the result is false-positive when an algorithm labels any of the pixels as positive.

Figure 3.3 shows the performance of our method based on this metric. Where Hinami et al. [40] test their method using only sixteen PCA components, we test our method over a range of used PCA components, as facilitated by our more discerning region extraction stage. The dotted line represents the performance achieved by Hinami et al. [40] when applying KDE on the features extracted from their multi-task Fast-RCNN model. To evaluate the stability of our KDE model, we repeat experiments eight times using a different random sample of at least 80% of the features and use the repeated results to form a box plot.

Considering that the original feature space has $4,096$ dimensions, a reduction down to sixteen dimensions is a severe compression, which results in the lowest AUC score. Performance steadily increases as more PCA components are maintained, until we are eventually able to match the performance reported by Hinami et al. [40] at 128 components. The reason why our results are lower until that point is most likely because we train a single KDE model to cover the entire frame space, rather than having twelve KDE models that are each able to specialise to the patterns of data observed in one-twelfth of the frame.

Next we consider the Frame-level evaluations on the Ped2 and Avenue17 datasets. We report our results in Table 3.1 using a single figure rather than a box plot, since the AUC scores are stable to within the level of precision reported.

It is interesting to note that we observe far less variation in the results in this case.

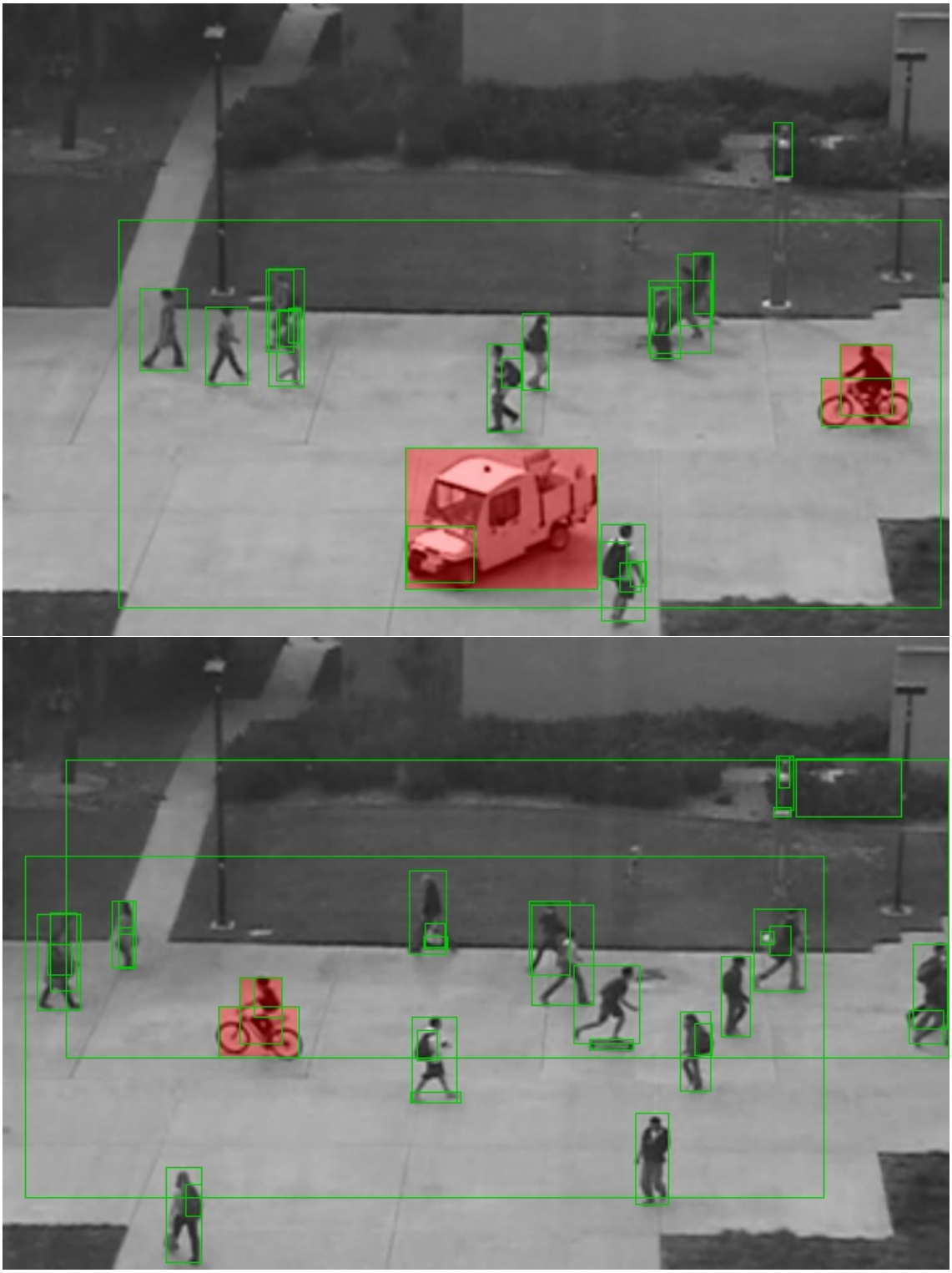

Figure 3.2: Examples of anomaly detection outputs obtained from our method. Top: the vehicle is successfully detected. Bottom: the bike is successfully detected, but the skateboarder is missed.

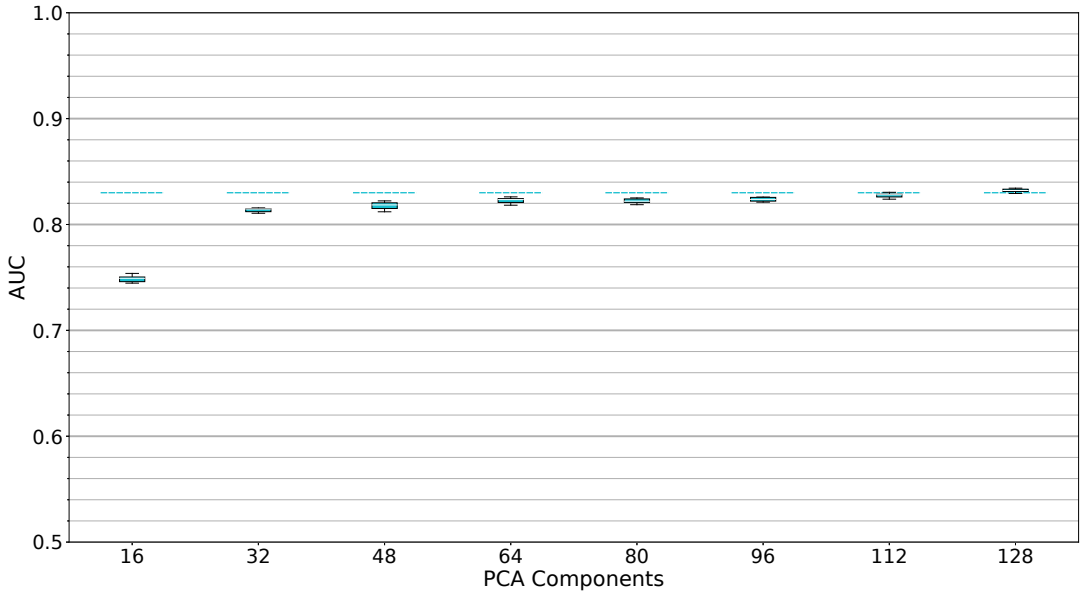

Figure 3.3: Pixel-level AUC evaluation of our KDE-based method on the UCSD Ped2 dataset over a range of used PCA components when features are extracted from the seventh layer without ReLU activation. The dotted line represents the performance reported by Hinami et al. [40].

We suggest that this is due to a shift in how the per pixel abnormality scores map to the true-positive or false-positive outcome for a frame. In this frame-level analysis, once there is a single pixel above the threshold then the frame is automatically a positive one. In contrast, for the pixel-level results we have seen thus far, the number and distribution of above-threshold values is important. This results in a far richer and fine-grained differentiation between frames that allows greater opportunity for deviation in outcomes.

Overall, there is not a significant difference in the performance between our method

Table 3.1: Frame-level AUC evaluation of our KDE-based method on the UCSD Ped2 [65] and Avenue17 [62, 40] datasets when features are extracted from layer seven without ReLU activation.

PCA	16	32	48	64	80	96	112	128
Ped2: Hinami et al. report 0.88								
	0.88	0.92	0.92	0.92	0.93	0.93	0.94	0.94
Avenue17: Hinami et al. report 0.88								
	0.87	0.88	0.88	0.88	0.88	0.88	0.88	0.88

and that of Hinami et al. [40], although our method performs slightly better on the Ped2 dataset. In the following analysis section we test variants of our proposed method to isolate the effect of various components and we examine the computational time benefits of our new region extraction stage.

3.3 Analysis

We perform a range of additional experiments to answer some questions about our proposed method. Firstly, we investigate the effect of choosing to include the ReLU activation sublayer in our feature representations (Section 3.3.1). Secondly, we investigate whether extracting features from layer six rather than layer seven improves results (Section 3.3.2). Since features tend to become more specialised nearer the output layer [105], perhaps it is beneficial to extract features from this lower layer. The justification for this is that we are applying a fixed, pretrained feature extraction network on an anomaly dataset that is different to the datasets on which it was trained. Thirdly, we investigate the effect of our proposed feature preprocessing method. We scale features by a constant factor prior to KDE, rather than normalising them. Finally, we determine the computation time benefits of using our replacement region proposal system (Section 3.3.4).

3.3.1 Including the ReLU Activation

In our main variant described in Section 3.1, we extract features as the raw activations of the seventh layer before ReLU activation. Figure 3.4 shows the results obtained when the features are extracted with ReLU activation. Here we observe that performance is negatively impacted by the ReLU function. A likely explanation for this is that the raw activations provide more information about the extracted regions since two features that differ mainly in their negative values will look much more similar after the ReLU function has transformed all negative values to zero. We further speculate that using activation functions that maintain negative values, such as the $tanh()$ or Leaky ReLU functions, will also perform better than the ReLU function but perhaps not as well as the identity function, since the Gaussian kernels

used in the KDE will not be adapted to the changes in the scale of the feature space introduced by the non-linearities.

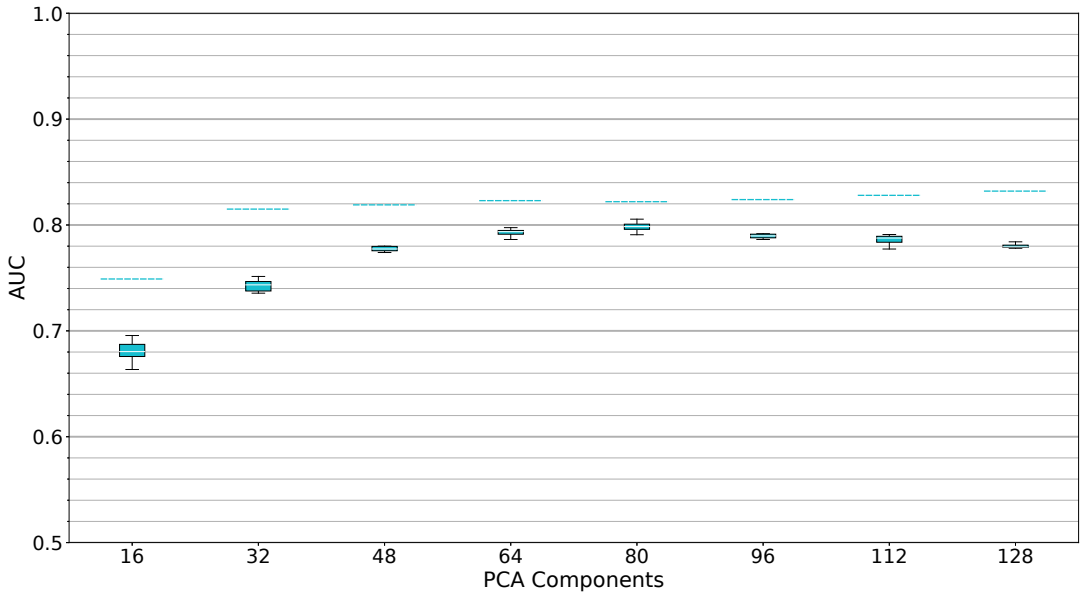

Figure 3.4: AUC performance of our method on the UCSD Ped2 dataset over a range of used PCA components when features are extracted from the seventh layer including ReLU activation. The dotted lines represent the median performance achieved when excluding ReLU activation as in our main results.

3.3.2 Using Lower Level Features

Figure 3.5 shows the AUC performance of our method when using features from layer six rather than layer seven. The comparison of these results with those obtained by our main layer seven variant (shown in dotted lines) demonstrates that the sixth layer produces much better features for anomaly detection on the UCSD Ped2 dataset. It may be the case that extracting features from even further down the network are better still, but this would cross over the ROI pooling layer into the convolutional layers and would require different handling. This is identified as an area for future work.

Table 3.2 makes a similar comparison for the frame-level evaluation on the Ped2 [65] and Avenue17 [40] datasets. In this case there is not a significant difference between the layer six and layer seven features; although, the layer seven features perform a little better on the Avenue17 [40] dataset at sixteen PCA components.

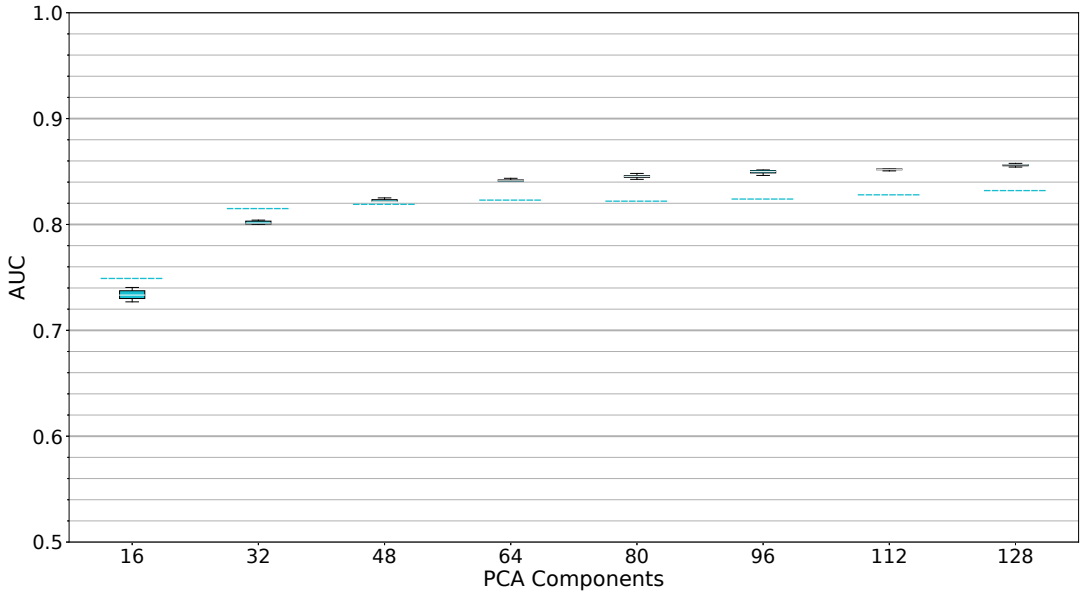

Figure 3.5: AUC performance of our method on the UCSD Ped2 dataset over a range of used PCA components when features are extracted from the sixth layer without ReLU activation. The dotted lines represent the median performance achieved when using the seventh layer as in our main results.

Table 3.2: Frame-level AUC evaluation of our KDE-based method on the UCSD Ped2 [65] and Avenue17 [62, 40] datasets when features are taken without ReLU activation.

PCA	16	32	48	64	80	96	112	128
Ped2								
Layer 6	0.88	0.90	0.91	0.93	0.93	0.93	0.94	0.94
Layer 7	0.88	0.92	0.92	0.92	0.93	0.93	0.94	0.94
Avenue17								
Layer 6	0.83	0.86	0.86	0.86	0.86	0.87	0.87	0.87
Layer 7	0.87	0.88	0.88	0.88	0.88	0.88	0.88	0.88

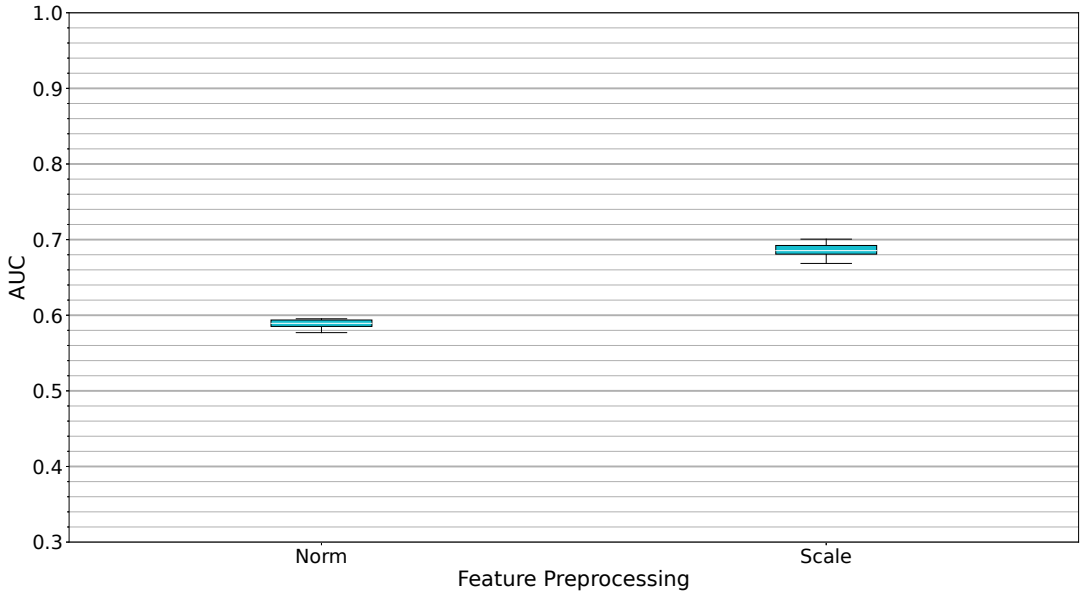

Figure 3.6: AUC performance of our method on the UCSD Ped2 dataset using sixteen PCA components when the features are normalised (as done by Hinami et al. [40]) and scaled (as in our main results).

The performance on the Ped2 [65] dataset is better than that on the Avenue17 [40] dataset and this is almost certainly due to a combination of at least two factors: firstly, the issue of determining anomalies in more complex scenarios as discussed in Section 2.2; and secondly, the fact that in Ped2 [65], normality and abnormality are largely determinable via identification of classes that were very prominent in the training set for the class branch of the feature extraction network.

3.3.3 Normalising the Features

As described in section 3.1.3, we depart from Hinami et al. [40] in the way that we preprocess the features before density estimation. Figure 3.6 illustrates the difference resulting from scaling the features rather than normalising them. We observe a significant increase in performance when preprocessing features using our scaling approach rather than normalisation. This is most likely due to the information lost during normalisation, which is preserved in our approach as described in Section 3.1.3.

3.3.4 Real-Time Performance

The key advantage of our method is in its suitability for real-time applications. We demonstrate the effect of allowing the number of regions extracted per frame to increase to the quantity used in the method presented by Hinami et al. [40] and establish a lower-bound on the increase in computational efficiency achieved when using our proposed modifications. We implement a simulation of Hinami et al. [40] whose region extractor selects 208 randomly chosen areas per frame. This is the number of regions per frame per KDE model used by Hinami et al. [40] (recall that they use twelve KDE models). In this simulation, we also disable the random sampling of features so that the same number of points is fed into the PCA and KDE models across all repeats of the experiment and across both variants. Otherwise, the simulation is identical to our method described in Section 3.1. In contrast to the original method of Hinami et al. [40], our simulation does not need to compute GOP and MOP, and only computes a single PCA and KDE model rather than twelve. Table 3.3 compares the computational efficiency of our method to the simulation in terms of the time taken to form the PCA model and the inference-time Frames Per Second (FPS). Despite the significant advantages afforded to our simulation, it still cannot compete with our proposed method on these metrics and is still not suitable for real-time application. These experiments are performed on a desktop computer with a GTX 1080 Ti, a 3.60GHz i7-7700 CPU and 16GB of RAM.

Table 3.3: Computational Time Comparison

PCA	PCA Time (s)		Inference FPS	
	Hinami-sim	Ours	Hinami-sim	Ours
16	7.0 ± 0.8	0.6 ± 0.2	1.4 ± 0.1	8.3 ± 0.3
32	8.0 ± 0.9	0.6 ± 0.1	1.0 ± 0.1	8.4 ± 0.2
64	10.7 ± 1.0	0.8 ± 0.2	0.6 ± 0.1	7.9 ± 0.2
96	11.5 ± 1.2	1.1 ± 0.1	0.4 ± 0.1	7.8 ± 0.1
128	14.9 ± 1.5	1.4 ± 0.1	0.3 ± 0.1	7.6 ± 0.1

3.4 Optical Flow

Our method operates on a single frame at a time and can only make use of information contained within that frame to discern normal from anomalous events. If it were able to consider information in neighbouring frames then the time-varying aspects of the scene could also inform the anomaly scores assigned to pixels. This could be very useful in a dataset like UCSD Ped2 and in other situations where anomalous entities tend to move at different speeds and directions to normal entities. We experiment with augmenting our method with optical flow so that it also has access to information about the motion of objects.

We use the 64-component layer six variant without ReLU activation to process the frames as normal. Alongside this, we compute the optical flow for the frame and take a set of statistics from each area corresponding to an extracted region: the maximum, minimum and mean optical flow value for the horizontal and vertical dimensions are taken for this purpose. From these, we form an additional KDE model with a six dimensional feature space. At inference time, we use the two KDE models to give two anomaly scores for each extracted region and sum them to produce a final anomaly score for the region. Each KDE model has its output shifted and scaled based on a calibration subset of normal points to bring their anomaly scores into a consistent range.

Incorporating optical flow in this manner gave an AUC score of 0.70 in our preliminary experiments. The result using the optical flow anomaly score alone was 0.38.

Among our other preliminary experiments were alternative attempts to incorporate optical flow. We attempted to form a single KDE model by fusing semantic and optical flow features into a single feature vector, resulting in no performance change. This is likely because the optical flow features have only six dimensions, while the semantic features have $4,096$ dimensions. Therefore, the optical flow features make up only a small portion of the fused feature vector and their information is not likely to be transmitted through the PCA step. Fusing the features after PCA avoids this problem but results in an undefined AUC score. We also substitute the minimum, maximum and mean statistics with a histogram of optical flow, with similar results.

Finally, we replaced the green and blue channels of the input frames with horizontal and vertical optical flow frames while keeping the red channel unmodified. Since UCSD Ped2 is a grey-scale dataset, this causes no loos of information. This resulted in an AUC score of 0.75.

Our inability to improve results using optical flow in our preliminary experiments prevented us from taking this further. We leave attempting to incorporate temporal information via optical flow or other means for future work. The most likely reasons for this outcome include the setting of the hyper-parameters for the optical flow algorithm and the fact that the feature extraction model was not trained on optical flow information.

3.5 Conclusion

We were able to perform real-time training and analysis in anomaly detection tasks due to a couple of key factors. Firstly, we extract features from fixed networks that have been pretrained on general recognition tasks. Using a network that is fixed during training avoids the computational cost of adapting network weights, and since there is no need to iterate over training examples multiple times, the method can observe a live video stream and be ready for deployment immediately afterwards. Secondly, we use a region proposal system that produces far fewer candidate regions that are better targeted, which reduces the computational load at the KDE stage. We showed that this reduction in region proposals allows us to significantly increase the dimensionality of the KDE model for enhanced anomaly detection performance.

CHAPTER 4

Anomaly Detection by Restoration

We consider the problem of detecting abnormal or anomalous regions within complex textural patterns such as leather, wood or carpet. A solution to this problem would be applicable in many industrial contexts including the manufacture of electrical components and the production of textiles, where anomalies may present in the form of dents, folds, scratches, contaminants and other defects in the product. The process of filtering defected samples could be made more efficient by the introduction of automatic detection systems based on modern machine learning.

We propose a new method for anomaly detection in this context and evaluate it on the MVTecAD [11], DAGM07 [102] and Leaf [43] datasets (Sections 2.2.8, 2.2.6, and 2.2.7). Responding to the challenge represented by the variation and complexity of these datasets, we make the following contributions:

- A new fast and simple method for training an encoder-decoder architecture that allows the final layer to *directly* represent the per-pixel anomaly scores without first having to generate an input reconstruction.
- Introduction of the Reflected Rectified Linear Unit (Reflected ReLU) output activation function that eases the training of the architecture under this new methodology.

- Improved anomaly detection results on the texture classes of the MVTecAD [11] dataset with respect to current state-of-the-art methods [71, 13, 12, 85, 5, 60, 56, 21, 63, 61], achieving an Area Under the ROC Curve (AUC) statistical performance measure of 96% and consistent performance across all texture classes.

We use a simple autoencoder-based technique for performing anomaly detection on textures. Although previous autoencoder methods for anomaly detection vary, they usually share two common features that impede their performance: firstly, they use a loss function based on the ℓ_1 or ℓ_2 distance between the input example and its reconstruction [12], and secondly, the inability of the autoencoder to effectively reconstruct high-frequency information is not addressed [68]. Bergmann et al. [12] point out the flaws in using ℓ_1 or ℓ_2 loss and substitute them for a perceptual loss based on Structural SIMilarity (SSIM) [101]. The main contribution of this work is to address the second issue by training the autoencoder to directly output the per-pixel anomaly map rather than an input reconstruction.

In contrast with our previous method (Chapter 3), this method was developed for low-level defect detection. Therefore, there is no consideration of how to prepare the model with knowledge of high-level concepts such as objects. For this reason, the focus of this chapter is on the MVTecAD [11] dataset. See Chapter 6 for evaluation across all considered datasets.

4.1 Method

Ultimately, our task is to find an anomaly map: a tensor the same shape as the input, including the channel dimension, whose elements measure the abnormality of each pixel component. This anomaly map can then be summed along the channel dimension to provide the two-dimensional per-pixel anomaly scores. When using a conventional or denoising autoencoder (Section 2.1.4), this task is approached indirectly by arranging for the model to first output a reconstruction of the input [36]. The difference between the input and its reconstruction is then used to create the anomaly map. In our proposed method, the anomaly map is the direct output

of the model. Assuming that most pixels are normal, on the basis that anomalies are rare, our model aims to output near-zero values in the majority of cases. By contrast, conventional and denoising autoencoders need to output a detailed image accurately with all image features exactly aligned.

An overview of our approach is shown in Figure 4.1. During training, an input tile I is intentionally corrupted with random noise, resulting in the noised input I_n. The encoder-decoder architecture transforms I_n directly into the anomaly map A, which may be thought of as the per-pixel adjustment required to repair the noise. To tune the architecture towards producing better anomaly maps A, we compare the repaired image $I' = I_n + A$ with the original, clean input image I. This comparison is performed by extracting features from both I and I' using the VGG-11 network [89] and taking ℓ_2 distance between these. We choose the VGG network because Johnson et al. [48] successfully used this network for a similar purpose in their research. We choose the VGG-11 network specifically because it is computationally cheaper while still performing very well. The images I and I' are normalised according to the requirements of the pretrained VGG-11 network before undertaking this comparison. During testing, we do not add noise to the input tiles and the values in A are treated as the pixel-shifts required to repair any naturally occurring anomalies. Different to other methods, there is no need to form the reconstructed image I' during testing, since the anomaly map A is produced

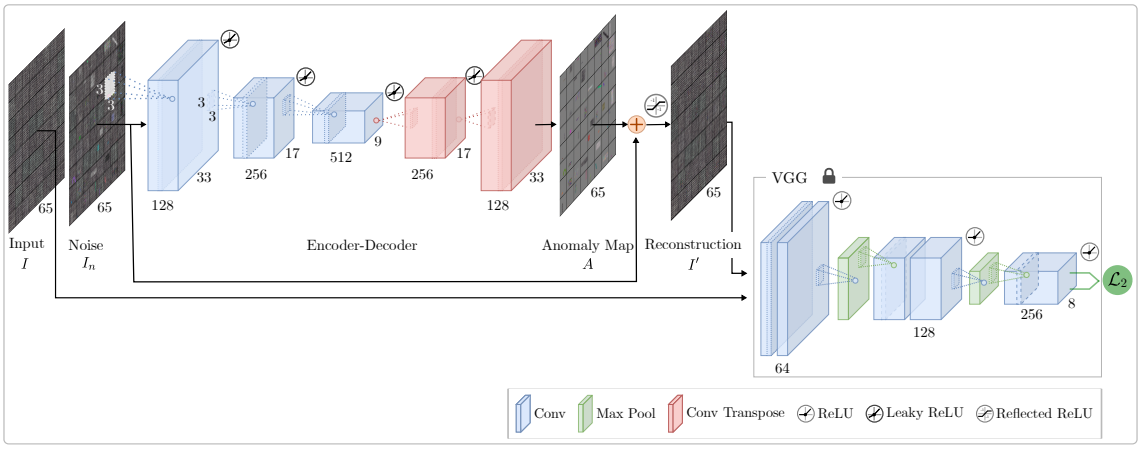

Figure 4.1: The architecture of our proposed texture restoration method.

Credit: Samet Akçay.

earlier in the pipeline.

The following subsections each describe a component our approach. These include the preprocessing (Section 4.1.1), Noising Filter Bank (Section 4.1.2), encoder-decoder architecture (Section (4.1.3), output activation function (Section 4.1.4), loss calculation (Section 4.1.5), training procedure (Section 4.1.6) and testing procedure (Section 4.1.7).

4.1.1 Preprocessing and Normalisation

Our architecture expects input image tiles whose values are normalised to the range $[-1..1]$ and whose dimensions are 65×65 pixels. These tiles are obtained from the dataset images via different methods in the training phase (Section 4.1.6) and testing phase (Section 4.1.7). All dataset images are downscaled to 256×256 before training and testing. The original size of the dataset images are 1024 × 1024 for the MVTecAD dataset [11], 512 × 512 for the DAGM07 dataset [102], and 256 × 256 for the leaf dataset [43].

4.1.2 Noising Filter Bank

Each training tile I is intentionally corrupted using a noising algorithm randomly selected from a collection of algorithms that we refer to as the Noising Filter Bank. This results in the noised training tile I_n in Figure 4.1. The bank includes noise filter operators that corrupt the image with the following types of image noise: salt, pepper, salt and pepper, Gaussian blur, Gaussian noise, rectangle, line, ellipse arc, shading, erosion, dilation and the identity filter [90]. Figure 4.2 provides examples of each of these filters.

If the identity filter is chosen for a particular image, then that image is left unaltered. Each of the remaining filters begins by creating a noise mask that defines which pixels may be affected by the filter, thus protecting the rest of the pixels from corruption.

In the case of the line and ellipse arc filters, the noise mask is formed by using planar drawing commands. All position parameters are drawn from a uniform

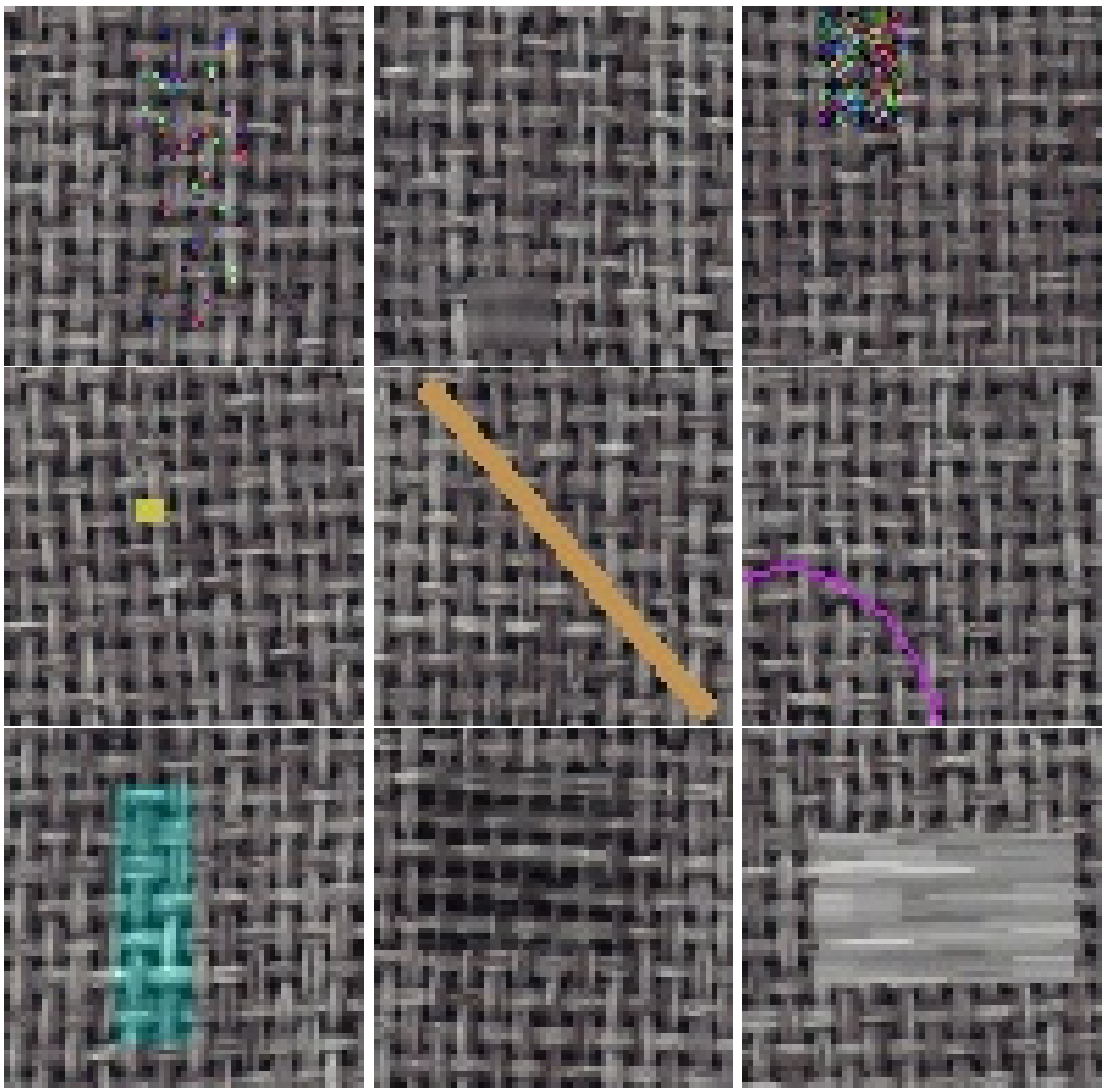

Figure 4.2: Examples of noise patterns generated by the noising filter bank. From top-left to bottom-right these are: salt and pepper, Gaussian blur, Gaussian noise, rectangle, line, ellipse arc, shading, erosion and dilation.

distribution within the area of the image. Line width parameters, measured in pixels, are random integers in the range [1..6]. Major and minor axes are random integers in the range [1..30]. Angles and angle ranges are random floats in the range [0..360].

In the case of the remaining filters, the noise mask is a rectangular region of randomised width and height inside the area of the image.

Noise is then applied to the pixels that are activated in the mask as follows. For the salt and pepper filters, pixels are changed to 1.0 or -1.0 with a probability drawn from the range [0.0..0.5]. For the Gaussian blur filter, a new image is formed by blurring the original image with a random kernel and sigma. Pixels from the blurred image are then copied to the original image according to the noise mask. Each axis of the kernel is a random odd integer in the range [3..15], while each axis of the sigma is a random float in the range [1.0..100.0]. For the Gaussian noise filter, Gaussian noise is added to the masked pixels, where the mean of the noise is zero, and the standard deviation is a random float in the range [0.3..0.6]. For the rectangle, line and ellipse arc filters, the masked pixels are changed to a randomly chosen RGB colour. As described above, the mask determines whether the affected region is a rectangle, line or arc. For the shading filter, a random float is drawn from the range $[-0.5..0.5]$ and added to the pixels activated in the rectangular mask. Erosion and dilation both use a circular kernel whose size is an odd integer in the range [1..7] with an iteration count in the range [1..2].

For all algorithms, the resulting image is clamped to the range $[-1.0..1.0]$. If the original input image is grey-scale, as is the case for the grid class, then any randomly selected colours are constrained to be gray-scale despite the use of a multi-channel input.

4.1.3 Encoder-Decoder Architecture

The encoder-decoder architecture is comprised of three convolutional layers followed by three transpose convolutional layers. Each layer has a kernel size of three, a stride of two and padding of one unit thickness on every side. The first convolutional layer has 128 feature maps and this number doubles at each additional convolutional layer

and halves at each convolutional transpose layer as in DCGAN [76]. The number of output feature maps is equal to the number of channels in the input. A Leaky ReLU activation function with a slope of 0.01 is applied after each layer except for the final layer, which is discussed in Section 4.1.4.

4.1.4 Reflected ReLU Activation

Often, a $tanh()$ output activation function is used to squeeze the output back into the $[-1..1]$ range [5, 76]; however, this activation function is unsuitable for our architecture because it prevents the learning of the simplest encoder-decoder function. Ideally, the network should be able to simply output zeros wherever the input is normal, thus minimising the amount of detail expected from the network in the most common case: that of a pixel being normal. Unfortunately, applying the $tanh()$ activation would prevent this from happening. Consider a normal pixel in the noised input I_n whose value is 0.5. If the network outputs a zero for this pixel in A, then when A is added to I_n it will produce a value of 0.5. This represents the perfect reconstruction that we would like to find in I', but applying the $tanh()$ at this point would result in a change in the value. Consequently, the network will need to learn to output a value in A that will undo the effect of the $tanh()$ function. Furthermore, the appropriate value to output would depend on the value of the input, which is the kind of coupling between input and output that we would like to avoid. Therefore, we propose the Reflected ReLU function, which is shown in the plot in Figure 4.3 and defined as:

$$RefReLU(x) = \begin{cases} 0.01x - 0.99, & x \leq -1. \\ x, & -1 \leq x \leq +1. \\ 0.01x + 0.99, & x \geq +1. \end{cases} \tag{4.1}$$

This activation function leaves the reconstruction pixel values unchanged when they are in the range $[-1..1]$, while reducing how far the they deviate from the dynamic range. This ensures that the range of the input to the VGG-11 network [89] is closer to the range on which it was trained. Section 4.1.4 analyses the effect

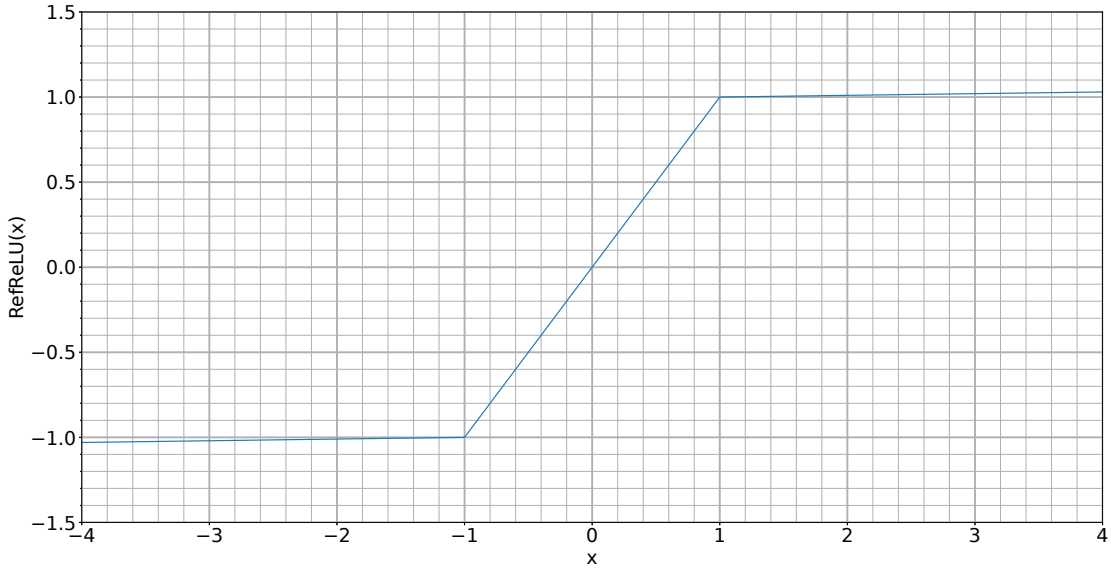

Figure 4.3: Plot of the reflected ReLU function.

of using this activation function.

A further disadvantage of the $tanh()$ function is that a pixel may need to be shifted by a magnitude greater than 1.0, and such values do not lie in the range of the $tanh()$ function.

4.1.5 Loss Function

We use a perceptual loss similar to that of Johnson et al. [48]. In our case, features are extracted from both the input image I and its reconstruction I' using a VGG-11 network [89] pretrained on ImageNet [23]. The ℓ_2 distance between these extracted features forms the perceptual loss as shown in Figure 4.1. The features are taken from the fourth convolutional layer after max-pooling and ReLU activation function are applied.

4.1.6 Training Procedure

Before training commences, 80,000 tiles of size 65×65 are randomly cropped from the downscaled training dataset images. The training batch for each iteration is formed by randomly selecting 64 of these tiles without replacement. This batch of 64 tiles is propagated through the pipeline shown in Figure 4.1 to produce the loss used to train the encoder-decoder network via backpropagation. The method of

optimisation is the same as that used in DCGAN [76] i.e. we use the Adam optimiser [49]. We follow those researchers and use an initial learning rate of 0.0002, $\beta 1 = 0.5$, and $\beta 2 = 0.999$. Training is stopped after 3,500 iterations, since this number proved to be a compromise among the optimal number of iterations of each dataset class.

4.1.7 Testing Procedure

Each testing iteration processes a single image from the downscaled testing dataset. We extract tiles that measure 65×65 pixels with a stride such that there is approximately 50% overlap horizontally and vertically with every pixel represented by at least one tile. These tiles are batched and propagated through the pipeline depicted in Figure 4.1 until the anomaly map A for each tile is produced. No reconstruction is formed, instead, for each tile we take the absolute value of the corresponding anomaly map and sum it along its channel dimension. This yields the anomaly score for each pixel. Anomaly scores from each tile are composed to form the per-pixel anomaly scores for the entire test image. Where multiple tiles share a set of pixels, anomaly score contributions from each tile are averaged.

4.2 Results

We train and test our proposed method on the texture classes of the MVTecAD dataset [11]: carpet, grid, leather, tile and wood using eight different random seeds. The performance is very strong across each of these classes as indicated by the AUC results displayed in the box plot of Figure 4.4.

Table 4.4 compares the mean of the results from each class with those achieved by the current state-of-the-art methods. In addition to outperforming these methods, some of the closest performing methods employ the use of more complex Generative Adversarial Network (GAN) [63] or Variational AutoEncoder (VAE) [21, 60] architectures that incur significant training requirements. By contrast, our method is very simple and quick to train, requiring only $3,500$ iterations and less than ten minutes of training time.

The key advantage of our method is that it is able to squeeze anomaly scores for

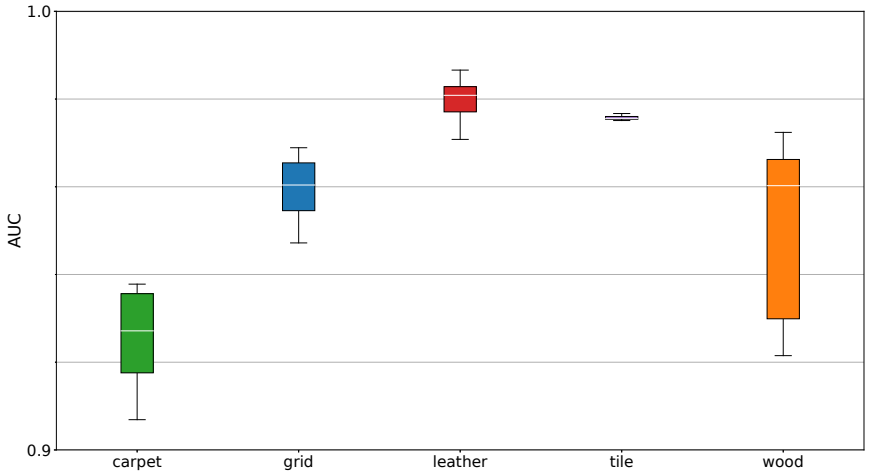

Figure 4.4: Box-plot of the AUC results produced by our proposed method on each of the MVTecAD [11] texture classes.

Table 4.1: Anomaly detection performance using textural restoration (AUC).

Method	Carpet	Grid	Leather	Tile	Wood	Mean
AE (L_2) [12]	0.59	0.90	0.75	0.51	0.73	0.70
AE (SSIM) [12]	0.87	0.94	0.78	0.59	0.73	0.78
AnoGAN [85]	0.54	0.58	0.64	0.50	0.62	0.58
CNN Feats. [71]	0.72	0.59	0.87	0.93	0.91	0.80
GMM [13]	0.88	0.72	0.97	0.41	0.41	0.68
GANomaly [5]	0.70	0.71	0.84	0.79	0.83	0.78
VEVAE [60]	0.78	0.73	0.95	0.80	0.77	0.81
SMAI [56]	0.88	0.97	0.86	0.62	0.80	0.83
VAEgrad [21]	0.74	0.96	0.93	0.65	0.84	0.82
P-Net [63]	0.57	**0.98**	0.89	0.97	**0.98**	0.88
FCDD [61]	**0.96**	0.91	**0.98**	0.91	0.88	0.93
Ours	0.93	0.95	**0.98**	**0.98**	0.95	**0.96**

normal pixels close to zero, resulting in anomaly segmentations that are incredibly clean. This is apparent in the examples shown in Figure 4.5, where normal regions are displayed in a near-constant white indicative of a low anomaly score. This is achieved without any post-processing of the anomaly maps. Our method is also sensitive to very fine anomalies that are not labelled in the ground truth of the dataset. Two examples of subtle anomaly detection can be seen in the bottom example of Figure 4.5, which depicts an example from the tile class that contains two small blemishes above the main defect to the left and right.

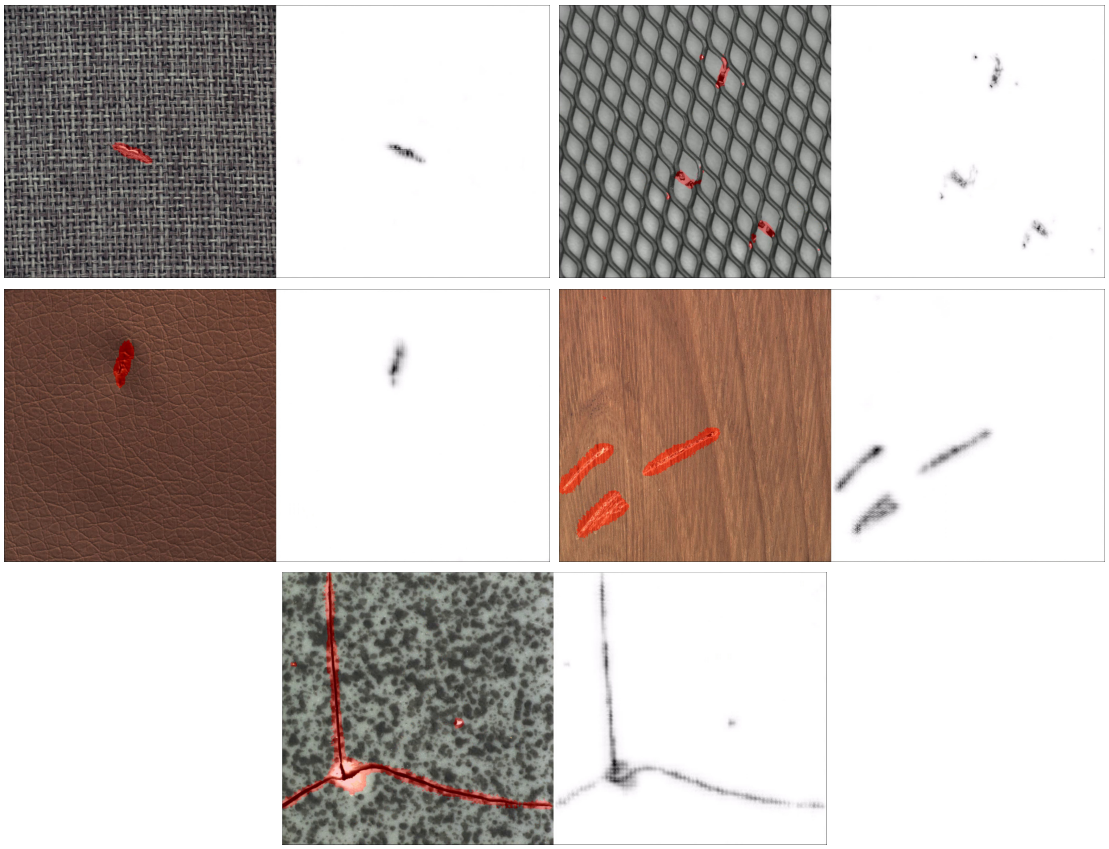

Figure 4.5: An example anomaly detection output for each texture class. Left images show a texture containing a defect with an anomaly segmentation in red. Right images show a heat-map of the per-pixel anomaly scores output from the autoencoder. Segmentations are produced by setting a threshold on the heat-map.

Since the Noising Filter Bank consists of a constant set of hand-crafted algorithms, it might be the case that they happen to fit the distribution of anomalies seen in the MVTecAD dataset particularly well. Therefore, we test our method on two further datasets: the DAGM-2007 dataset [102] and the Leaf dataset [43].

The DAGM-2007 dataset [102] is an algorithmically generated dataset of artificial

textures with defects. Example anomaly detections on this dataset are shown in Figure 4.6. Unfortunately, the ground-truth is too coarse to allow for adequate quantitative performance measure at the pixel-level; however, qualitatively we observe the same strong anomaly signatures against near-constant backdrops of low scoring normal pixels. The detections are consistent across the dataset, even for very subtle examples such as the middle-left example of Figure 4.6.

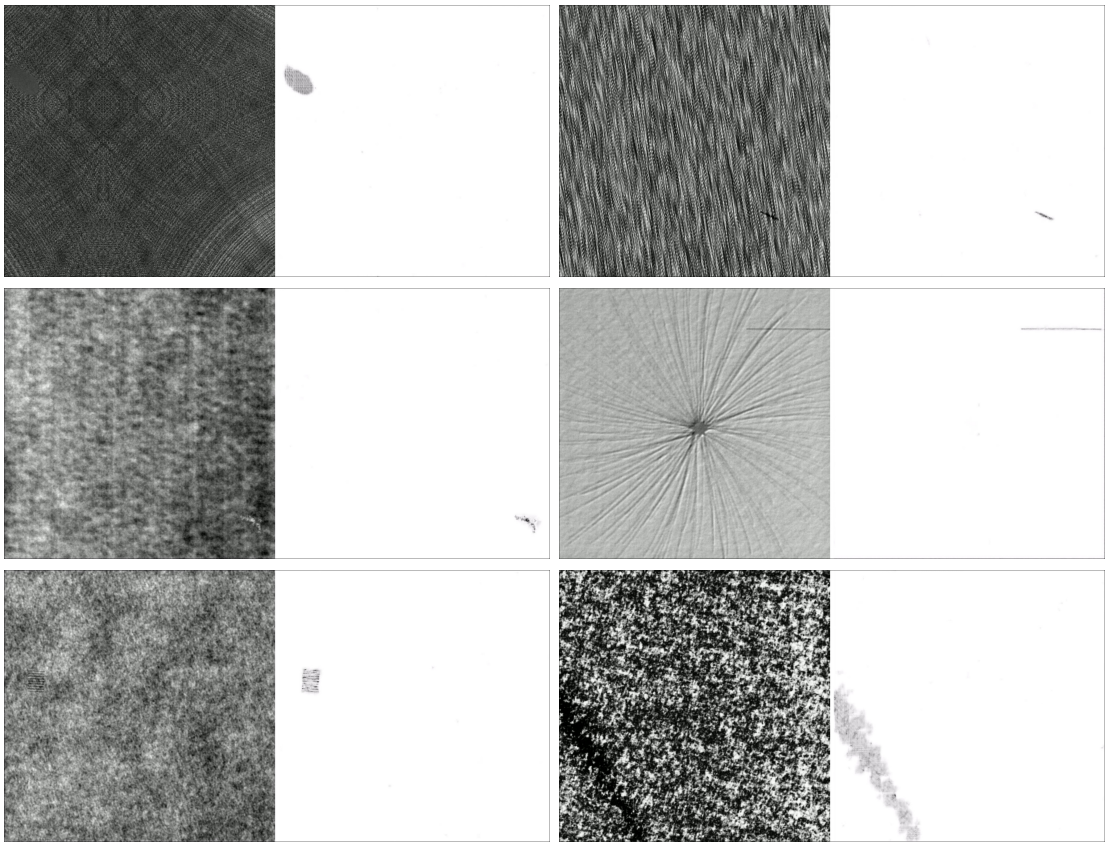

Figure 4.6: Examples of anomaly detection results on the DAGM-2007 dataset [102]. Anomaly detection overlay has been omitted for clarity.

The Leaf dataset [43] consists of approximately 50,000 images of fourteen different species of leaf. A subset of these images are of leaves showing signs of disease, while the remaining images are of healthy leaves. We train our method on the healthy subset of images and then subsequently attempt to segment the diseased pixels in the diseased subset. No ground-truth is provided with this dataset but qualitatively, overall anomaly detection performance is strong as shown in the examples provided in Figure 4.7 that are representative of the majority of cases. In a minority of cases, our method fails to produce good anomaly segmentations on

this dataset as shown in Figure 4.8.

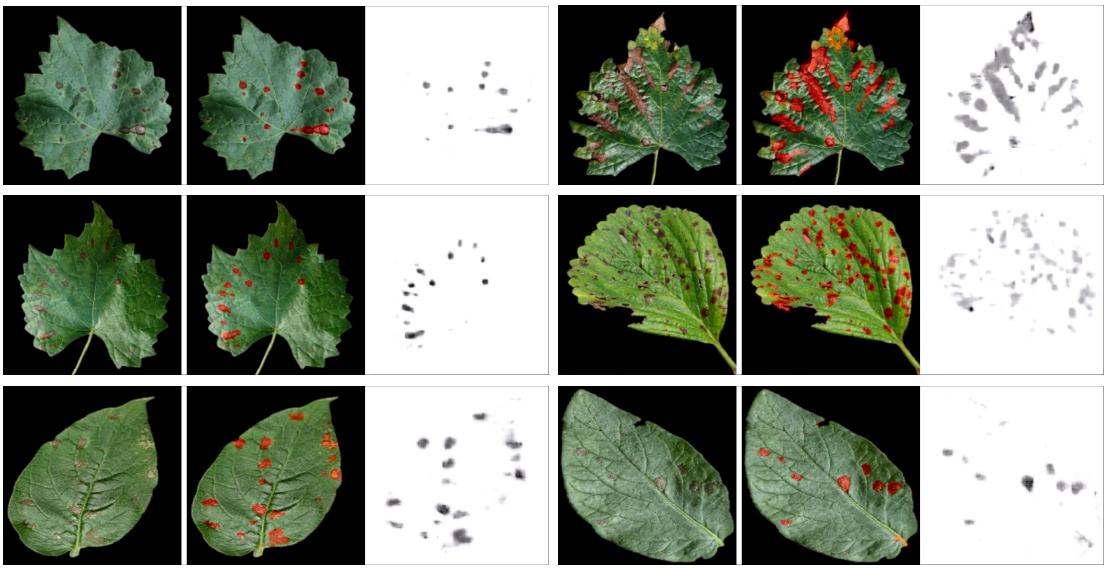

Figure 4.7: Examples of anomaly detection results on the Leaf dataset. Left images show an example leaf that contains a disease, middle images show the same images with anomaly segmentation overlaid in red, and the right images show the anomaly heat map.

Figure 4.8: Examples of poor anomaly segmentations in the Leaf dataset.

The DAGM07 and Leaf datasets are good examples of use cases for our method and we include them here as additional support for our method. Unfortunately, to our knowledge, other researchers have not applied their methods to these datasets and so we do not provide comparative results.

4.3 Analysis

Our method differs in several ways to the conventional autoencoder. The following subsections consist of ablation studies and investigations to measure the effect of

using the perceptual loss instead of the ℓ_2 loss (Section 4.3.2), downscaling the dataset images (Section 4.3.3), using the noising bank (Section 4.3.4), outputting the error map rather than a reconstruction (Section 4.3.5), and using the Reflected ReLU activation function (Section 4.3.6).

Downscaling the dataset images is a preprocessing step that could also be applied to the conventional autoencoder. For this reason, Section 4.3.3 also includes an analysis of the effect of downscaling images when using the conventional autoencoder of Bergmann et al. [12]. This analysis finds that downscaling dataset images *hinders* the performance of the conventional autoencoder due to compressing image patterns into a higher-frequency space.

For ease of comparison with our main results, the box-plots located throughout this section include dashed lines representing the medians found in the box plot of Figure 4.4. Also for comparison, we include a section that analyses the progress of the training phase of our method (Section 4.3.1), which may be referred to in subsequent sections.

4.3.1 Progression of Training

In this section, we describe typical features of the training, using only the Carpet class as an example for brevity. Our method requires less than ten minutes of training or $3,500$ iterations to achieve the performance outlined in Section 4.2. Figures 4.9 and 4.10 show how the training loss progresses with iteration count when the perceptual and ℓ_2 losses are used respectively, with each showing a graph with and without downscaling applied.

Training always progresses with loss decreasing in a stable manner, and the behaviour of these graphs is consistent across multiple runs of the method. Perceptual losses are larger than ℓ_2 losses due to the higher dimensionality of the feature space compared to the image-tile space. Interestingly, the full-scale loss tracks above the downscale loss when using the perceptual loss, but this characteristic is reversed when using the ℓ_2 loss. In addition to examining the progress of training in terms of the training loss, we also provide example training output batches at 10, 100, 1000 and 3500 iterations in Figure 4.11.

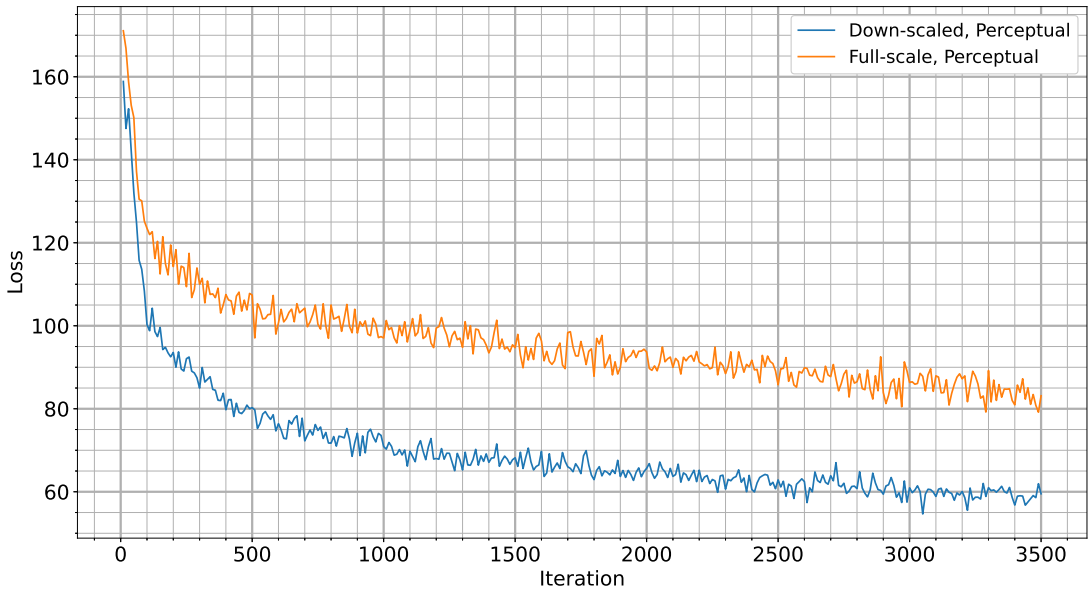

Figure 4.9: The perceptual training loss of our method on the carpet class of the MVTecAD dataset [11] both with and without the downscaling described in Section 4.1.

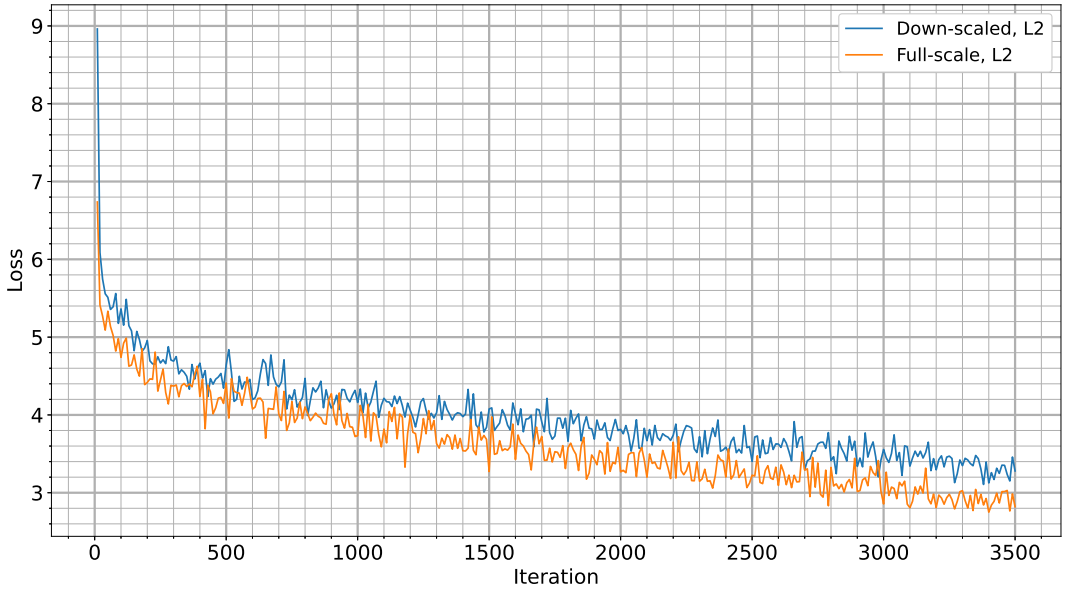

Figure 4.10: The ℓ_2 training loss of our method on the carpet class of the MVTecAD dataset [11] both with and without the downscaling described in Section 4.1.

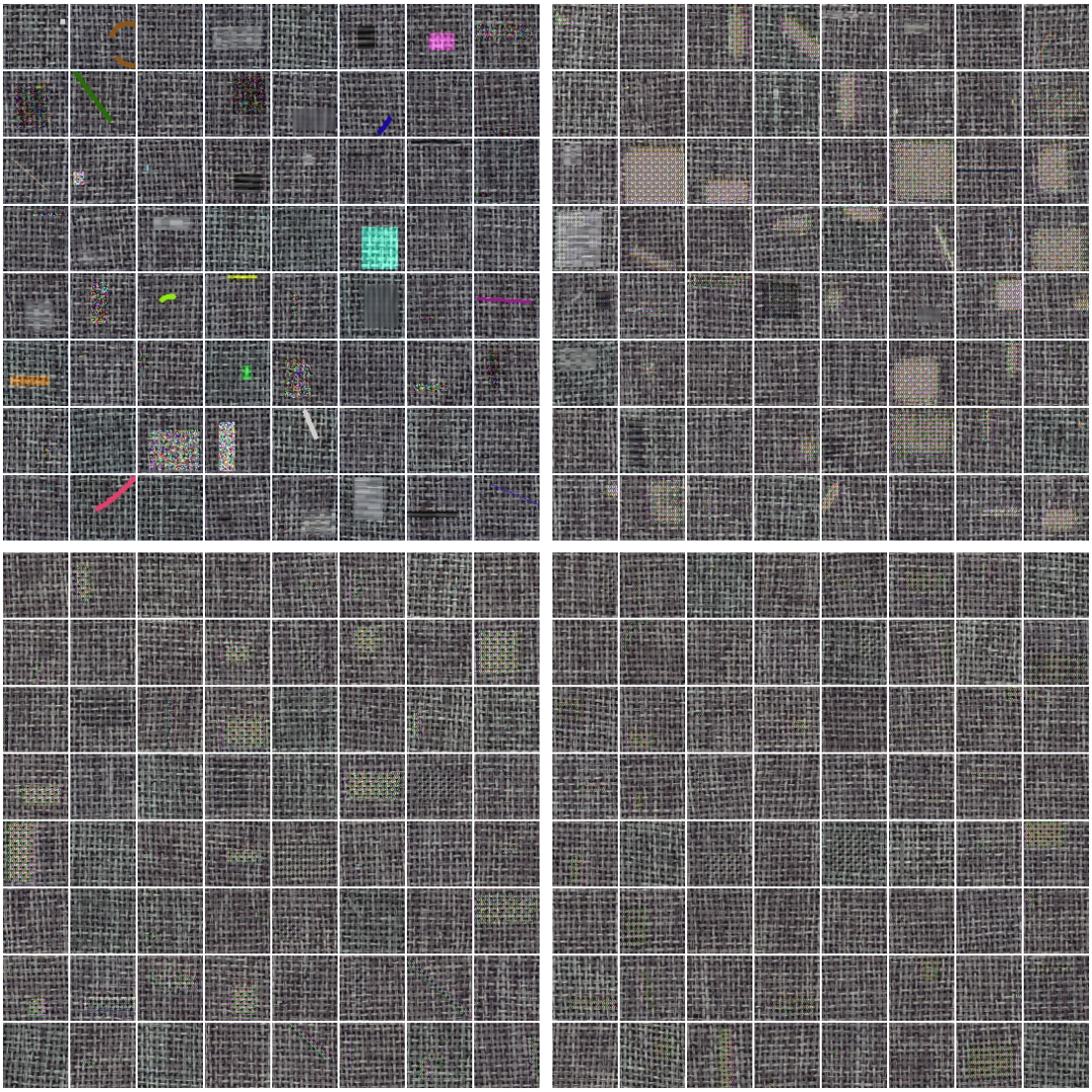

Figure 4.11: Example training output batches at various iterations. From top-left to bottom-right the iteration counts are: 10, 100, 1000 and 3500.

Within 10 iterations the model has learned to pass through all information i.e. the encoder-decoder architecture has learned to output near-zero values for every pixel in its output regardless of whether these pixels correspond to noised pixels or not. As training progresses, the model is learning to output larger magnitude values where the output pixels correspond to noised pixels. At first, this results in an obvious defacing of the input tiles, but gradually, the values are tuned to phase out the noise so that the output tiles look cleaner.

This training objective is slightly at odds with the task. Wherever pixels are affected by noise, we would like the model to output values as large as possible rather than values that are just large enough to counter the noise. However, all attempts to target this outcome resulted in worse performance. We infer from this that there may be some utility in forcing the model to attend to the underlying normal pattern during training.

4.3.2 Effect of Perceptual Loss

We use perceptual loss [48] to measure the distance between a clean input tile and its training-phase reconstruction (Section 4.1.5). Figure 4.12 shows the performance of our method when we instead use ℓ_2 loss.

Similar to Bergmann et al. [12], we find that using ℓ_2 loss is inferior to using a perceptual loss. This result still stands despite investigating a different perceptual loss to Bergmann et al. Nevertheless, the difference in performance is not enough to account entirely for the success of our method. The performance of the ℓ_2 variant of our method is still far in excess of most of the state-of-the-art methods referenced in Section 4.2. We suggest that the other components of our method mitigate to a large extent the drawbacks of ℓ_2 loss. Firstly, in our method, all image features in the input and reconstructed tiles are automatically exactly aligned, since we predict pixel deviations rather than pixel values. This is important for ℓ_2 loss since, unlike perceptual losses, there is no position invariance. Secondly, our method produces reconstructions that have no limit on their sharpness. While perceptual losses can forgive some lack of fidelity, the ℓ_2 loss is critical of every pixel error. Therefore, when applying our method, the advantages offered by perceptual losses are somewhat

diminished.

4.3.3 Effect of Downscaling

One of the preprocessing steps in our method is to downscale all dataset images to 256×256 (Section 4.1.1). The box plot in Figure 4.13 shows the results obtained when this downscaling is not applied. From this, we can see how downscaling significantly improves performance, especially for the carpet and grid classes. Examples of anomaly segmentations in the high-resolution setting are depicted in Figure 4.14 for the leather, tile and wood classes including examples from all failure cases, if any.

The anomaly segmentations are very precise; consequently, they may only be sparsely filled due to the presence of normal pixels within the anomalous region. In such cases, the anomaly map resembles a cast of the anomaly rather than a solid segmentation as required by the ground truth. This is particularly evident in the top-left example of the wood class and the bottom two examples of the tile class. At full size we also observe that anomaly segmentations sometimes do not cover the

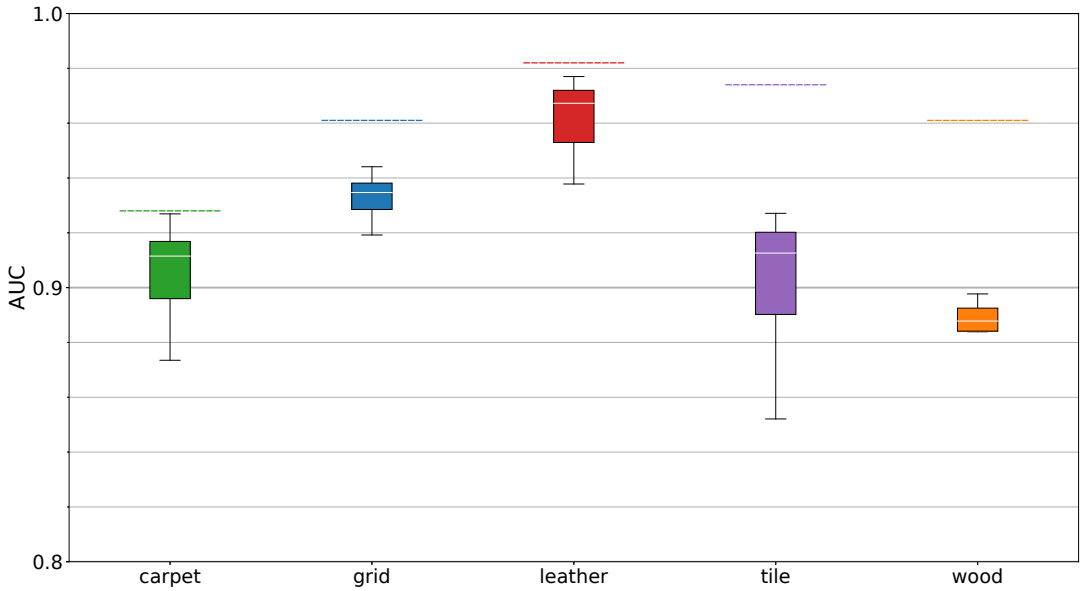

Figure 4.12: AUC results achieved by our proposed method when ℓ_2 loss is used instead of perceptual loss [48]. The dashed lines represent the medians of the box plot in Figure 4.4 in which perceptual loss was used.

entire anomalous region as can be seen in the bottom-left example of the leather class. In addition, there are a total of two failure cases across the three datasets, namely the presence of glue on tile (top-right example) and the presence of liquid on wood (bottom-right example). Sparsely filled and low coverage detections may be relieved in part by post-processing operations such as opening and closing.

In summary, although some of the performance gain acquired by downscaling can be accounted for by considering the degree to which anomalies are fully covered, the more significant factor is that downscaling allows the model to detect more of the anomaly types (e.g. contamination with glue and liquid). When evaluating the efficacy of the model as an anomaly detector, one could argue that the former factor is irrelevant, since in these cases, the model is still able to tag anomalies with a high concentration of positive detections while ignoring the vast majority of the normal pixels. It is the latter factor that is relevant because here, the model fails to trigger for certain anomaly types. Only when evaluating the efficacy of the model as an anomaly segmentation mechanism is the former factor relevant.

Downscaling improves performance on the Carpet and Grid classes the most. We suspect that this is because for these classes, the repeating characteristic of the

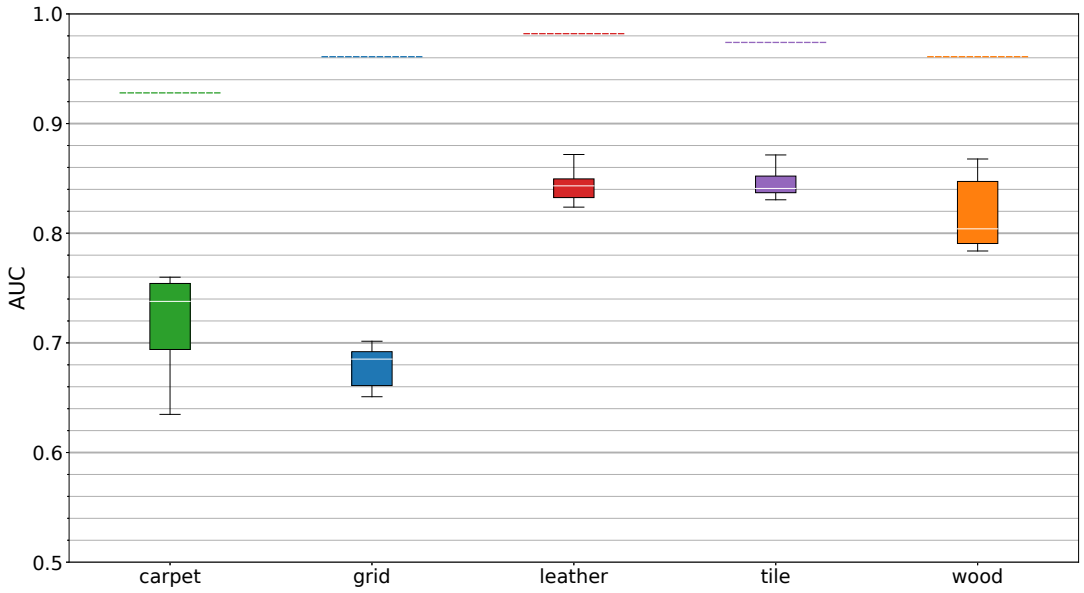

Figure 4.13: AUC results achieved by our proposed method when the images are not downscaled before training and testing. The dashed lines represent the median performance when downscaling to 256×256 is applied, as shown in Figure 4.4.

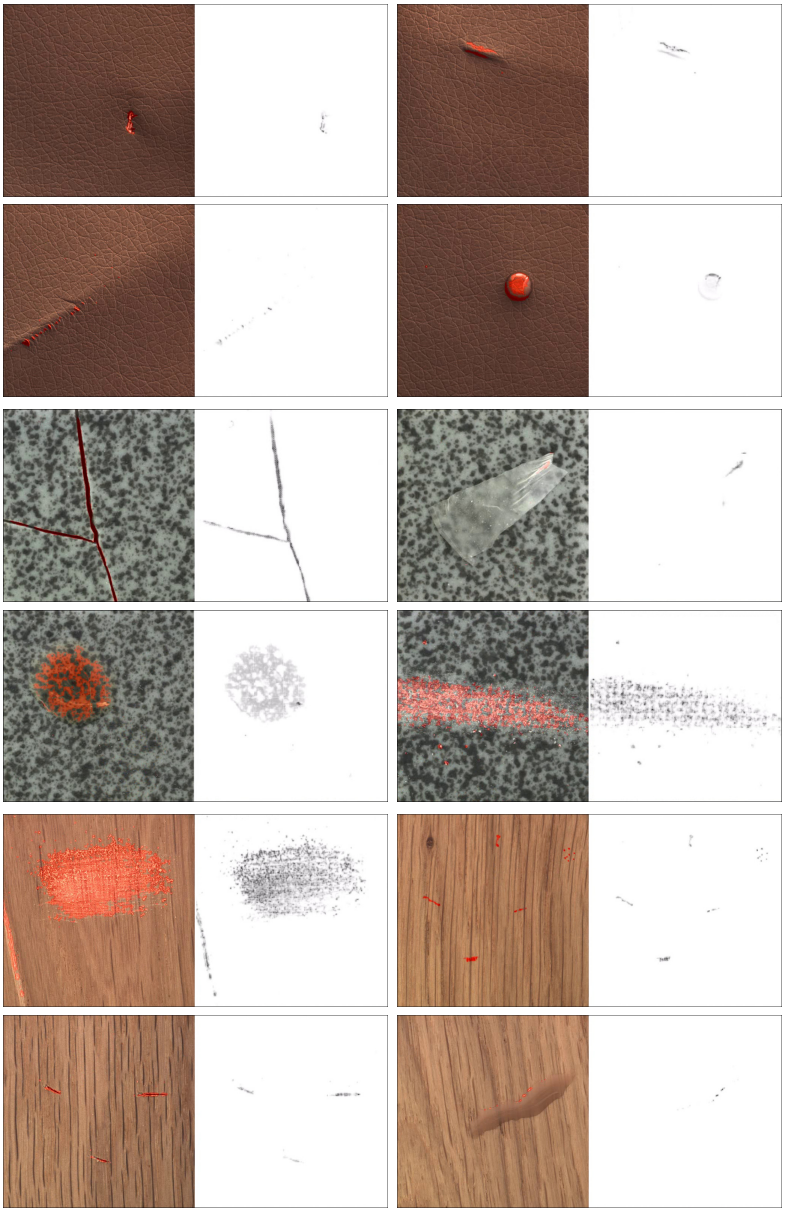

Figure 4.14: Examples of anomaly detection results on the leather, tile and wood classes of the MVTecAD dataset at full resolution.

textural pattern is only apparent over a larger area of pixels. This is particularly true for the Grid class, in which there is a lot of space between neighbouring segments of wire mesh, but a similar principle holds for the Carpet class in which the weave of the fabric requires a large area of pixels to be clear. In contrast, the other classes are more homogeneous. This might present a challenge because the inner-most units of the encoder-decoder architecture have a very small receptive field and thus are not provided with enough contextual information when the relevant patterns are spread in space. Downscaling squashes the pattern into a tighter space, which may mitigate this. Without downscaling, very fine blemishes were detected by our method. Whether or not these minor defects are still visible after downscaling is not relevant to the performance of the model since these were not labelled anomalies.

Since downscaling is a preprocessing step that can as easily be applied to the conventional autoencoder, we need to consider whether or not this method can similarly benefit from downscaling.

Applying Downscaling to the Conventional Autoencoder

We consider the conventional autoencoder of Bergmann et al. [12] on the carpet class, since this class was one of the two classes that most benefited from downscaling. The discussion focuses on the training procedure over tens of thousands of iterations. For comparison, recall that we used $3,500$ iterations to produce the results presented throughout this chapter. Figure 4.15 shows how the training loss decreases with the number of iterations when using full-scale dataset images. Training improves the quality of reconstructions by introducing successively higher frequencies as shown in the examples overlaid in the figure. First, the model learns to reconstruct the lowest frequency, the DC-component of the Fourier space. At this point, the model is able to reconstruct the average colour of the input tile but nothing more. Next we observe some slightly higher frequency details, but nothing significantly meaningful. A sudden decrease in training loss occurs when the model is able to reconstruct the frequency of the weave of the carpet in one dimension and then in both dimensions. From here the training loss falls slowly as the model introduces higher frequency details into the weave.

Reconstructions begin to look competent after about $3,500$ iterations as shown in the final example of Figure 4.15, but even after $80,000$ iterations the autoencoder is failing to introduce the sharpness of the highest frequencies as shown in Figure 4.16. Reconstruction quality stagnates as is indicative of the flattening of the training loss curve.

Contrast this behaviour with that observed when the dataset images are downscaled to 256×256. Figure 4.17 shows the training loss with example reconstructions taken from the same key points. The first thing to notice is that the model progresses through the same frequency milestones as before, only this time it takes much longer. The model gets temporarily stuck at each of these milestones, giving the training loss curve a more pronounced step-like appearance. Reconstruction ability becomes competent only after about $15,000$ iterations, where once again we observe stagnation and flattening of the training curve as before, only this time at a higher training loss. By downscaling the images, we push the weave of the carpet into a higher frequency space, which is a weakness of the conventional autoencoder.

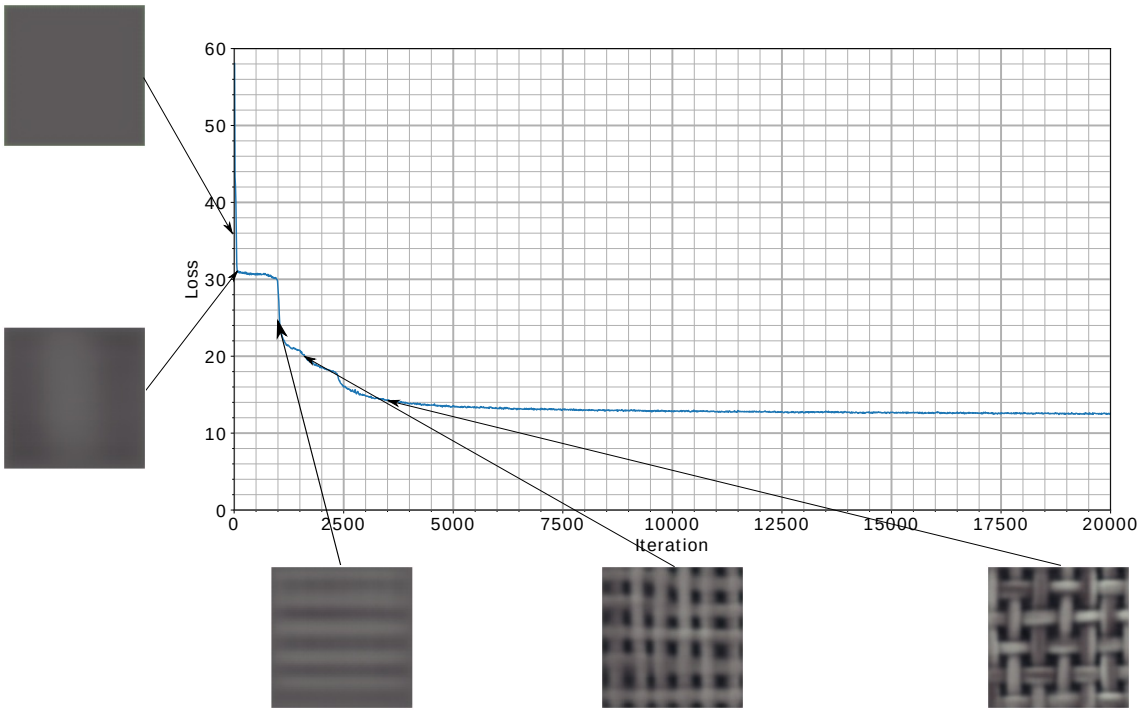

Figure 4.15: The training loss of the ℓ_2 Bergmann autoencoder [12] on the carpet class of the full-scale MVTecAD dataset, together with example training reconstructions at key points.

Comparing this behaviour with that observed in Figure 4.9, we observe that the training loss never gets stuck when using our method. The figure shows that this is true whether we downscale or not. For better comparison, Figure 4.10 demonstrates the same for the ℓ_2 variant of our method. Figure 4.11 demonstrates how much more quickly the reconstruction quality progresses. The encoder-decoder output of both methods progresses in frequencies, reaching the DC-component very quickly; while for the conventional autoencoder this means reconstructing the average colour of the input tile, for our method this means reconstructing the noised tile with all frequency components expressed. Likewise, the encoder-decoder output of both methods fail to reach the highest frequencies; while for the conventional autoencoder this means reconstructions will lack sharpness, for our method this means reconstructions will not eliminate quickly varying noise patterns. This is an unimportant consequence since we are not aiming to eliminate the noise, but for the conventional autoencoder, this is a serious flaw.

Figures 4.18 and 4.19 show example input instances and their reconstructions for the full-scale and downscaled variant respectively for a middling iteration count of $10,000$. We see how downscaling *hinders* the reconstruction quality of the conventional autoencoder for a given number of iterations.

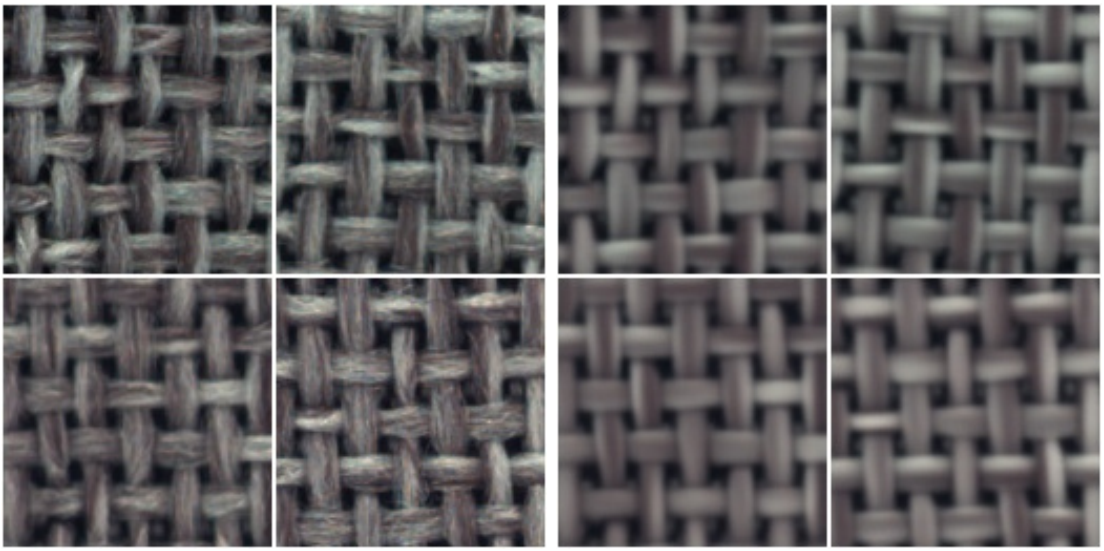

Figure 4.16: Example instances from the $80,000$th training iteration of the ℓ_2 Bergmann autoencoder [12] on the carpet class of the full-scale MVTecAD dataset. Left: input examples. Right: reconstructions.

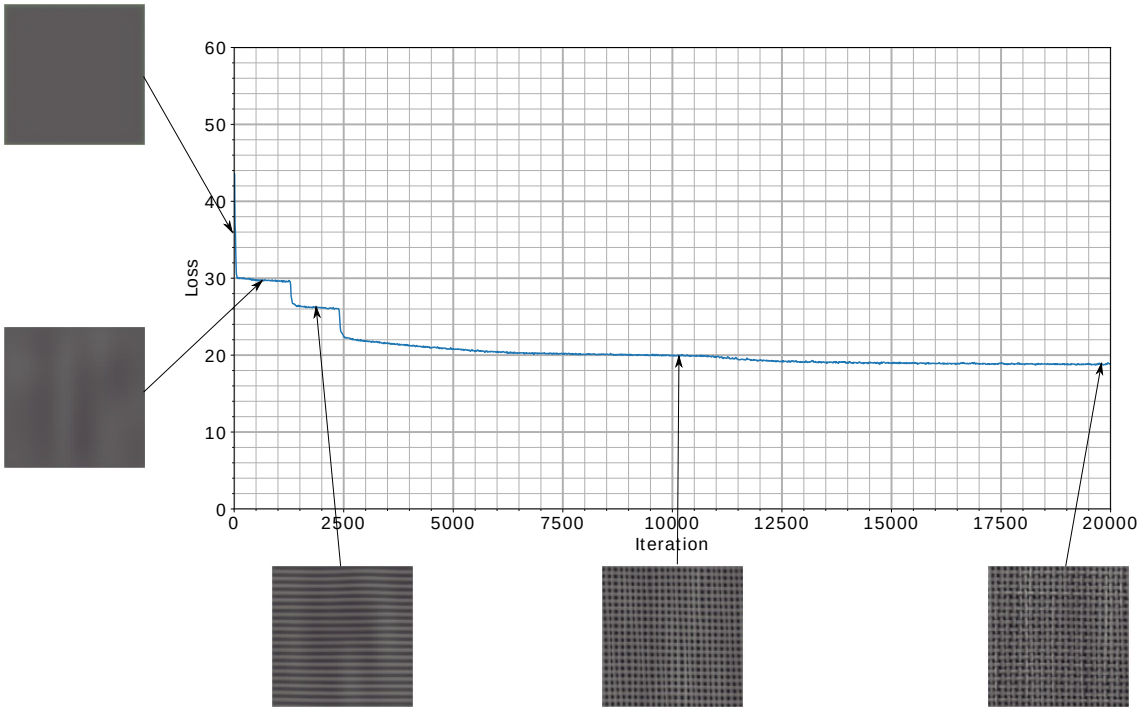

Figure 4.17: The training loss of the ℓ_2 Bergmann autoencoder [12] on the carpet class of the downscaled MVTecAD dataset, together with example training reconstructions at key points.

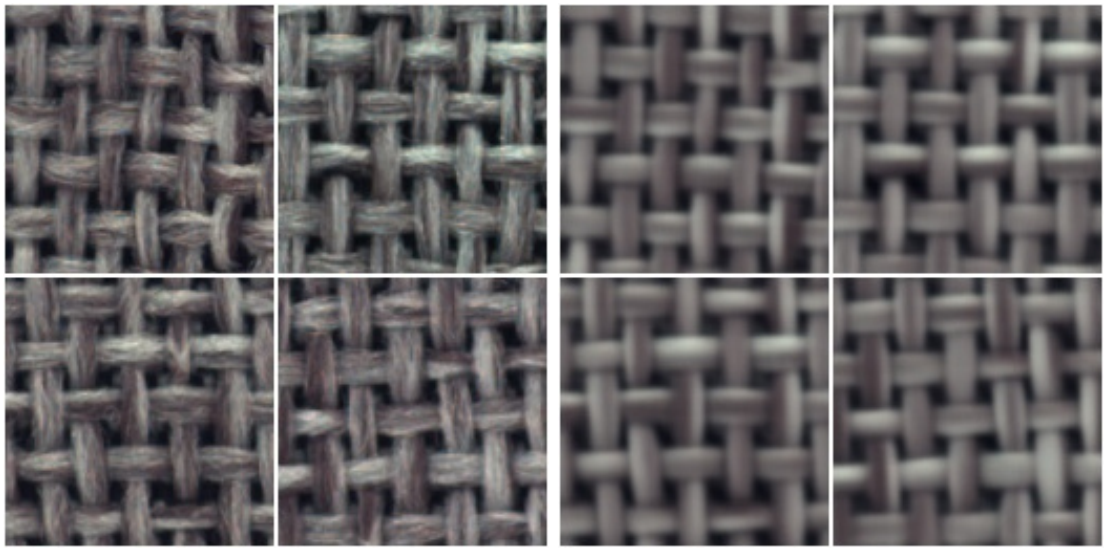

Figure 4.18: Example instances from the $10,000$th training iteration of the ℓ_2 Bergmann autoencoder [12] on the carpet class of the full-scale MVTecAD dataset. Left: input examples. Right: reconstructions.

Altogether, this results in lower performance from the conventional autoencoder and a decrease in that performance when images are downscaled. The performance ranges of the conventional autoencoder on full-scale and downscaled images are shown in Figure 4.20. We trained up to $80,000$ iterations, at which point it appears as though performance on the downscaled dataset is saturating while performance on the full-scale dataset may still be able to increase. The performance of our reimplementaion of the conventional autoencoder [12] has exceeded that reported in the original publication on the MVTecAD dataset [11] (Table 4.1), but is nevertheless still substantially below the performance achieved by our method after only $3,500$ iterations. See Figure 4.4 and Table 4.1 for results on the downscaled dataset and Figure 4.13 for results on the full-scale dataset.

4.3.4 Ablation of Noising Filters

Our proposed method entails corrupting the input training tiles using noising filters randomly selected from the Noising Filter Bank (Section 4.1.2). This raises the question of how the method would perform when using different subsets of the Noising Filter Bank. Different subsets are produced by splitting the algorithms into five families: opaque, consisting of rectangle, line and ellipse arc; transparent,

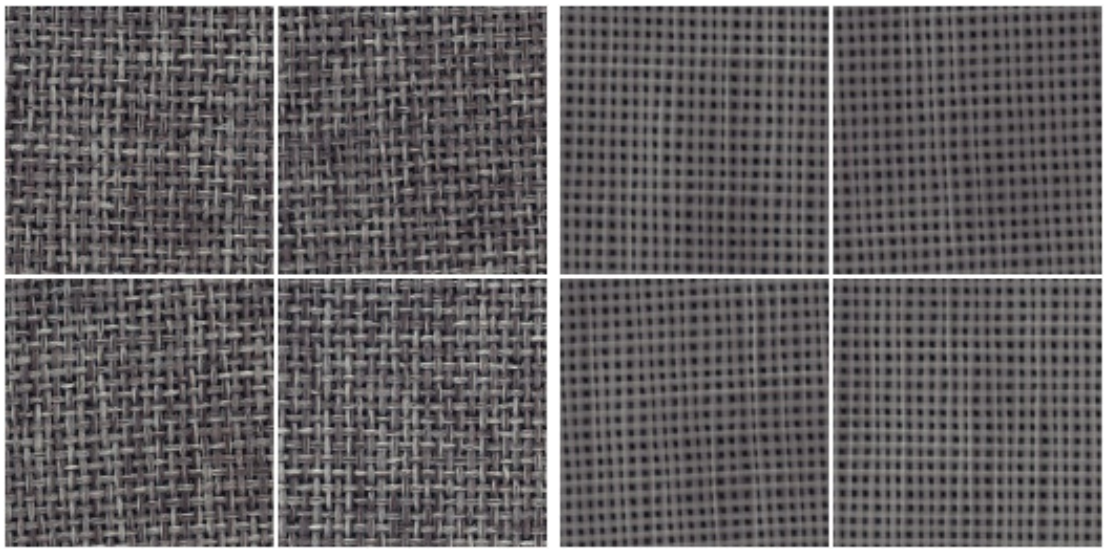

Figure 4.19: Example instances from the $10,000$th training iteration of the ℓ_2 Bergmann autoencoder [12] on the carpet class of the downscaled MVTecAD dataset. Left: input examples. Right: reconstructions.

consisting of shading and Gaussian blur; speckle, consisting of salt and pepper noises; distribution, consisting of Gaussian noise; and morphological, consisting of erosion and dilation. Positive and negative noise ablation tests are performed using these families and their results displayed in Figures 4.21 and 4.22 respectively. During a positive noise ablation study, a single family is tested in isolation, while during a negative noise ablation study, a single family is excluded.

These results show the benefit of increasing the variety of noises; no single kind of noise is responsible for good performance.

4.3.5 Resistance to False-Positives

We investigate the key advantage of our proposed method: that anomaly scores for normal pixels are kept very low, providing a quiet backdrop with minimal noise against which anomalies may be identified clearly. Figure 4.23 shows a histogram of anomaly scores assigned to all normal pixels in the leather dataset. Our proposed method (shown in blue) keeps almost all anomaly scores below 0.03. By contrast, when our method is modified to behave like a regular denoising autoencoder (shown

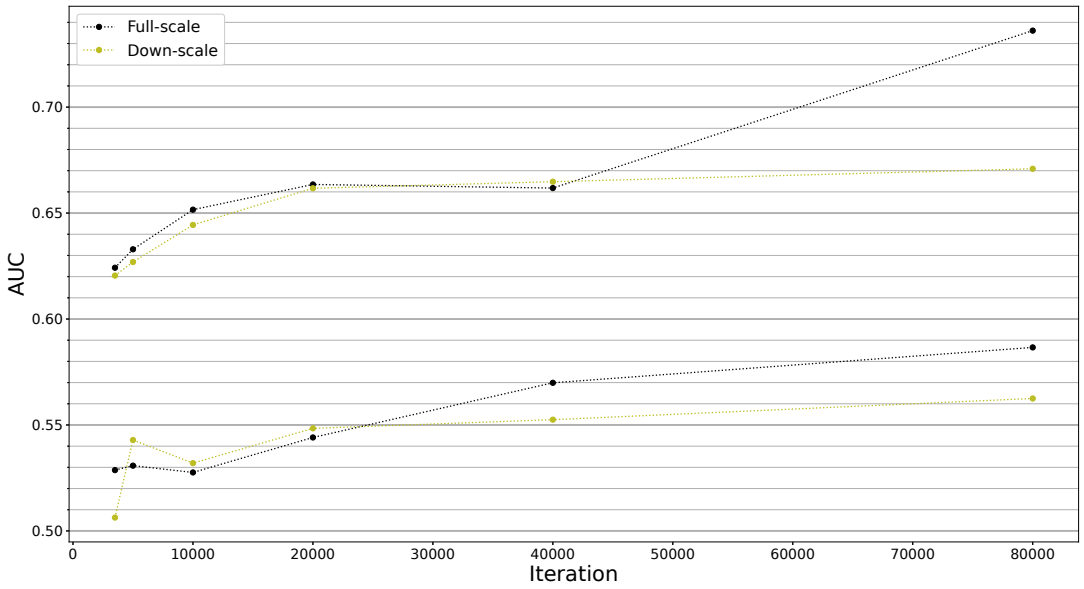

Figure 4.20: Performance ranges of the ℓ_2 Bergmann autoencoder [12] on downscaled and full-scale images of the carpet class of the MVTecAD dataset over a range of training iterations.

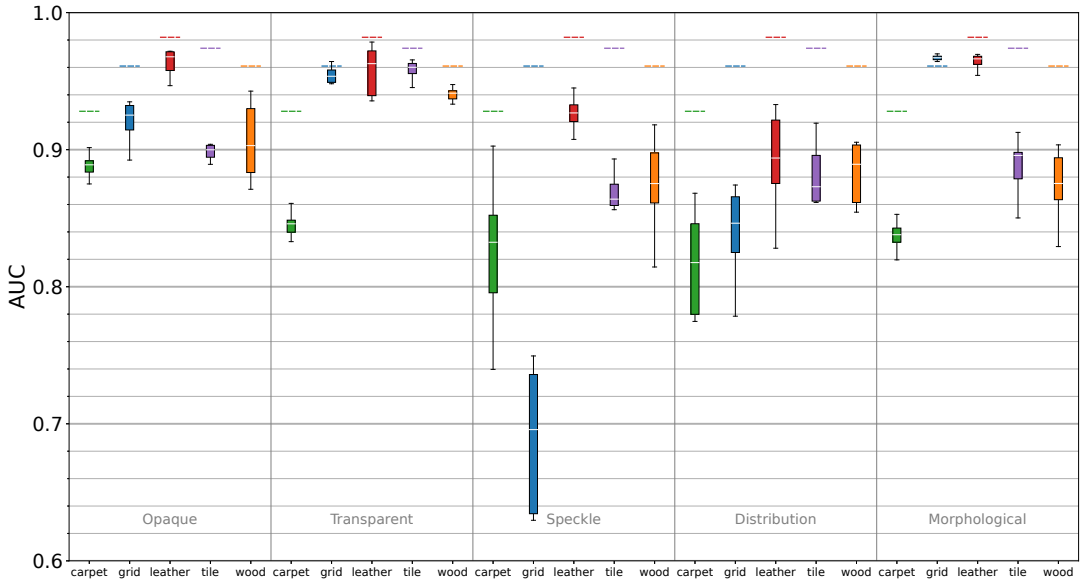

Figure 4.21: AUC results achieved by our proposed method when the Noising Filter Bank consists of only a single family of noising filters. The dashed lines represent the median performance when all noising algorithms are used, as shown in Figure 4.4.

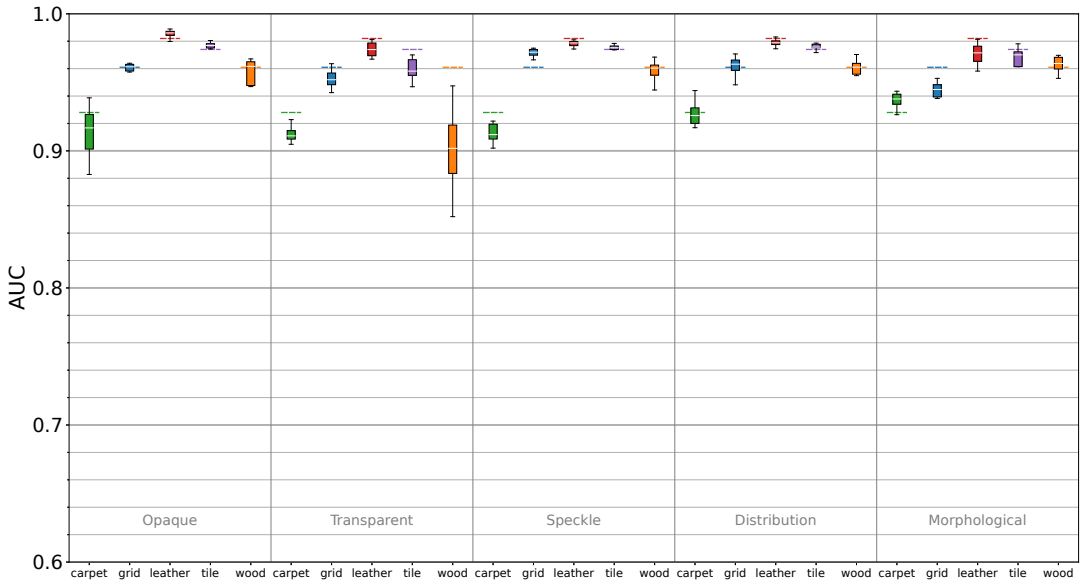

Figure 4.22: AUC results achieved by our proposed method when the Noising Filter Bank consists of all families of noising filters except one. The dashed lines represent the median performance when all noising algorithms are used, as shown in Figure 4.4.

in orange), anomaly scores fill a much wider spread of values centred approximately on 0.13. The difference is that the regular denoising autoencoder needs to output a reconstruction of each input tile, which is a more error-prone task than simply measuring deviations from normality. Consequently, our approach creates anomaly maps that are very quiet in normal areas of the image, as shown in Figure 4.24.

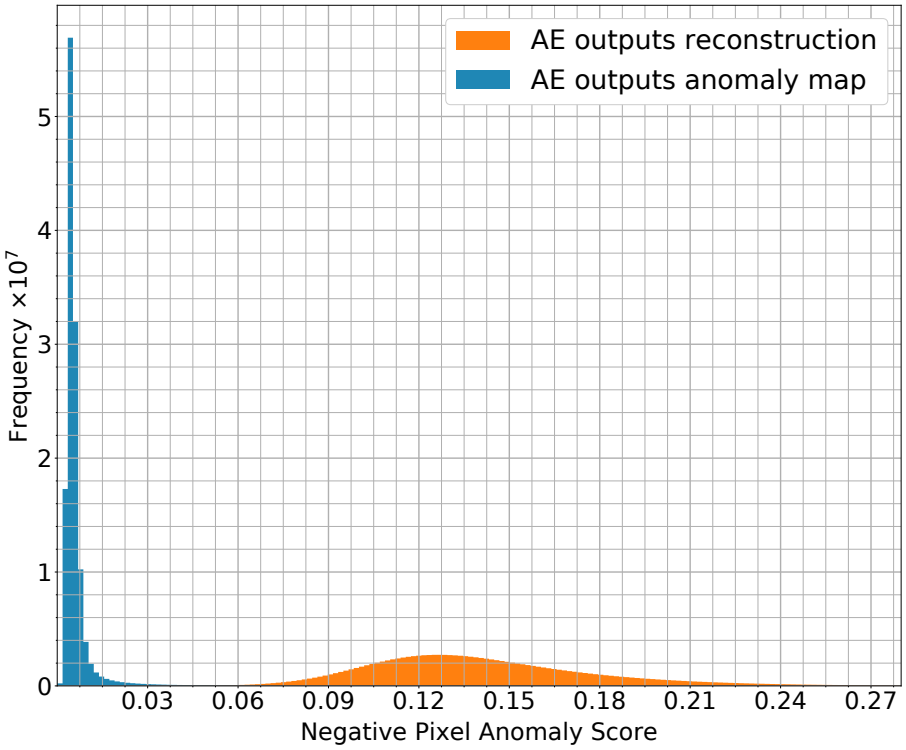

Figure 4.23: Histogram of anomaly scores assigned to negative (normal) pixels across the entire leather dataset. The blue area shows the scores assigned by the unmodified encoder-decoder architecture that outputs anomaly maps directly. The orange area shows the scores assigned when the encoder-decoder training is modified to mimic the behaviour of a regular denoising autoencoder. In this case, the architecture outputs reconstructions.

4.3.6 Effect of the Reflected ReLU

Often, a *tanh*() output activation is used in encoder-decoder architectures similar to ours [76, 5]; however, we replace this with the proposed Reflected ReLU function (Section 4.1.4). Figure 4.25 shows the effect of using the original *tanh*() activation function and no activation function. The results support the hypothesis that the new Reflected ReLU function is superior for use in our proposed method, likely because

it simplifies the task by allowing the autoencoder to output zeros for normal pixels. Compared to using no output activation function, the benefit of using the Reflected ReLU function is less significant. The most significant factor is the fact that the output of $tanh()$ function differs from its input through the useful output range of the encoder-decoder architecture. Using either the Reflected ReLU function or no function will avoid this flaw and so the difference between the two is small.

Figure 4.24: Examples of anomaly maps generated for a sample frame from the leather dataset. Left: the encoder-decoder architecture outputs the anomaly map directly. Right: the encoder-decoder architecture outputs a reconstruction of the input.

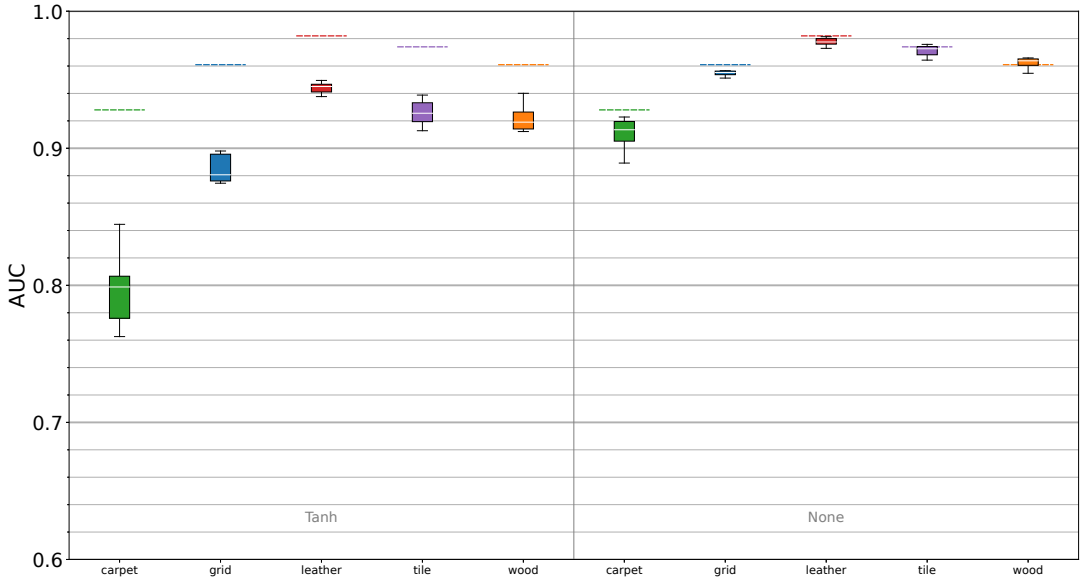

Figure 4.25: AUC metrics produced by our proposed method when our Reflected ReLU activation function is either replaced with the $tanh()$ function or removed. The dashed lines represent the median performance when using the Reflected ReLU function, as shown in Figure 4.4.

4.4 Improving Performance at Higher Resolution

We attempt to improve the performance of our method at a higher resolution of 512×512. We begin by considering the fact that two of the five dataset classes greatly benefited from downscaling while the remaining three benefited by a smaller degree. The classes that benefited the least: leather, tile and wood, are textures where information about the nature of the pattern is contained in a very local area of pixels. One could describe these textures as more homogeneous. In contrast, the carpet is made up of threads that weave among themselves and the grid is made up of metal wire that is twisted into a particular pattern. Therefore, a local area of pixels does not contain the basic repeating unit of the texture. Our encoder-decoder architecture is shallow and quick to train, but a drawback of having only three convolutional layers is that pixel information is not aggregated over a very wide area. We perform experiments at varying input kernel sizes to investigate whether there is a relationship between the major frequencies contained in the image data and the preferred kernel size.

Figure 4.26 is a box plot of results achieved under a range of varying input kernel sizes. The strides of the kernel are maintained at $\lceil k/2 \rceil$ and the tile size expands with the kernel size so as to maintain the same number of units at every layer within the architecture. At higher kernel sizes we notice some training instability due to the perceptual loss network. Figure 4.27 shows a box plot of the experiment repeated using the ℓ_2 loss. It is possible that adapting the learning rate may also have alleviated this problem.

We find that increasing the input kernel size improves performance at the higher resolution of 512×512. This provides some support to the hypothesis that the benefit of downscaling was to increase the percentage of the pattern that could be consumed by individual units in the architecture.

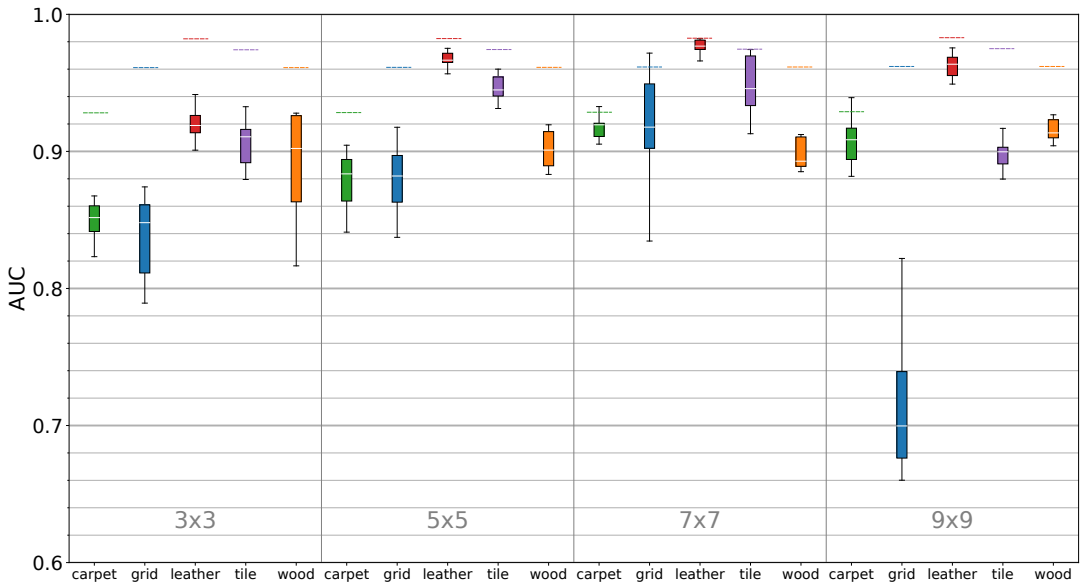

Figure 4.26: The performance of our method when images are downscaled only to 512×512. Each section represents the performance when the input-layer kernel size, k, is adapted to the size shown in the section label, and the stride is maintained at $\lceil k/2 \rceil$. Tile size is adapted to maintain the number of units throughout the network. Dashed lines represent the performance of our method in its original configuration, as presented in Figure 4.4.

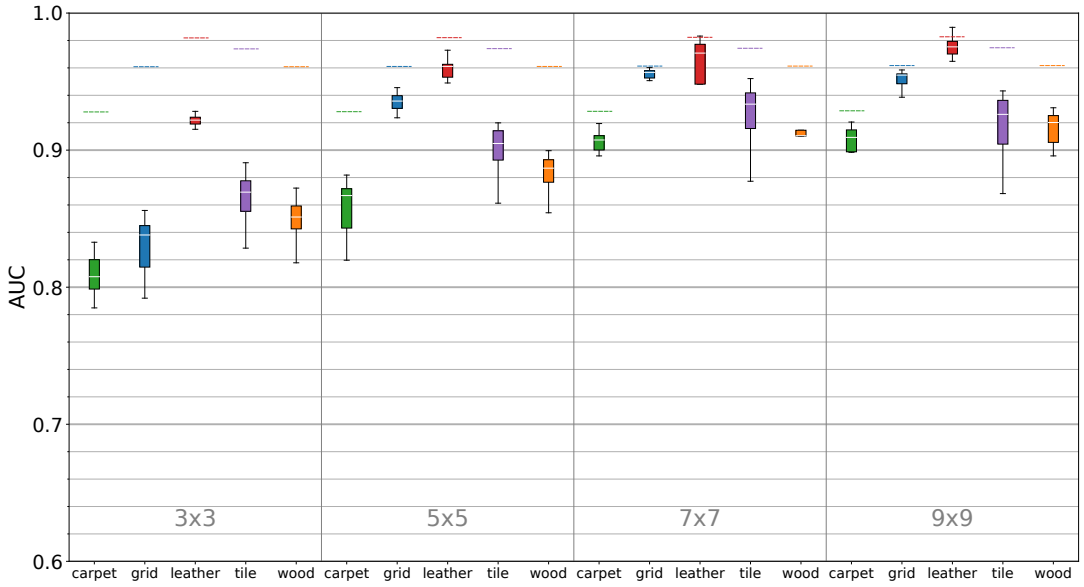

Figure 4.27: The performance of our method under the same conditions as in Figure 4.26 except that the ℓ_2 loss is used rather than the perceptual loss. Dashed lines represent the performance of our method in its original configuration, as presented in Figure 4.4.

4.5 Conclusion

We introduce a new autoencoder based architecture for textural anomaly detection. Rather than presenting training examples directly to the autoencoder, we first corrupt them with random noise in a way comparable to Denoising AutoEncoders (DAE). In contrast with DAE, we use a bank of noising filters to increase the variability of the noise. Furthermore, our autoencoder does not directly output a noise-free reconstruction of the input, rather, it predicts the negative of the noise, which may be used directly as an anomaly map during the testing phase when no artificial noise is added. In combination with the new Reflected ReLU output activation function, this allows the autoencoder to output a near-zero value for each normal pixel, which facilitates the pass-through of high-frequency information during the training-phase. Our proposed method is capable of achieving an average AUC score of 96% across the MVTecAD texture classes [11], where the best state-of-the-art method achieves 93%. Furthermore, we achieve this with a method that is simple, fast and stable during training and that requires less than ten minutes of training.

CHAPTER 5

Anomaly Detection by Pixel Prediction

Previous methods have attempted anomaly detection in video data by predicting future frames [59, 69] (Section 2.3.1). With the recent rise in Transformer networks for generalised sequence-to-sequence tasks, and their even more recent application in the image domain, we investigate the possibility of performing anomaly detection in image data by predicting future pixels. See Section 2.1.6 for the relevant background on Transformer networks.

We take an input sequence of pixels as context and use a Transformer network to predict the sequence of pixels that follows (Section 5.1). Having been trained on normal pixel sequences, the Transformer network is unable to predict novel pixel sequences and therefore the difference between the predicted and actual pixel sequences can be used as a measure of abnormality at inference time. We later extend this idea to predicting future features rather than pixels, leading to some improvement (Section 5.5).

At the time of experimentation there were no Transformer-based anomaly detection methods and at the time of writing there are still no Transformer-based pixel-prediction methods. The contribution of this chapter is then a first and novel attempt at targeting anomaly detection through this means. Using this method, we

demonstrate competitive anomaly detection on most of the challenging MVTecAD dataset texture classes [11] and outperform some of the previous generation state-of-the art methods [85, 5, 12, 13] (Section 5.2).

Similar to our previous method (Chapter 4), this method was developed for low-level defect detection and we once again focus on the MVTecAD [11] dataset. See Chapter 6 for evaluation across all considered datasets.

5.1 Method

Given a sequence of pixels to serve as context, we use the Transformer-based GPT-2 architecture [77] and our variations thereof to predict the sequence of pixels that follows. Chen et. al [14] also use GPT-2 for predicting pixel sequences (Section 2.1.6) but, in their case, this is a generative pretraining step for tasks other than anomaly detection. We modify the work of Chen et. al [14] to reduce computational time, facilitate anomaly detection, and to explore possibilities for tailoring the architecture for our own purpose. In particular, we generalise the method of predicting individual pixels to small arbitrarily sized regions of pixels we call brush-strokes (Section 5.1.1). We still refer to this image generation strategy as pixel prediction, and speak of predicting a sequence of pixels.

Unfortunately, applying transformer networks on image data can be computationally intensive [14, 74, 26] (Section 2.1.6). Therefore, we downscale dataset images to a size of $D_{img} \times D_{img}$ and then extract tiles measuring $D_{tile} \times D_{tile}$. Unless otherwise stated, we use $D_{img} = 256$ and $D_{tile} = 64$ so that 16 tiles are required to cover the image. The method described in this chapter operates on tiles.

The input and output of the GPT-2 architecture is a sequence of integers or *tokens*, S. The set of all possible tokens $\mathcal{V}$ is referred to as the *vocabulary* of the model and determines all that is interpretable by the model at the input and expressible by the model at the output. Input tiles must therefore be tokenised before propagation through the network (Section 5.1.1). This is facilitated by having used K-means clustering on normal brush-strokes to yield a set of brush-stroke centroids or *visual words*, each identifiable by an index that may be used as a token. After tokenisation,

an input tile has been transformed into a linear sequence of $|S|$ tokens, where the first token carries information about the top-left of the tile and subsequent tokens represent areas of the tile first spreading to the right and then down. This particular ordering of the sequence is not important for the method, but it is the one we adopt.

During the training phase, the model takes some number L_{ctx}^{train} of the tokens as context and attempts to predict the next (Section 5.1.3). The output is a probability distribution over the tokens in the vocabulary, which is analogous to the output of a regular classification network that outputs a probability distribution over classes. We therefore train in a similar manner using the cross-entropy loss function with the ground-truth taken from the known tile sequence.

During the inference phase, the model takes the first half of the tile tokens as context i.e. $L_{ctx}^{test} = \frac{|S|}{2}$. A forward propagation yields the first predicted token, which is then appended to the context and the process is repeated to generate the second half of the tile (Section 5.1.4). The completed tile is then assigned anomaly scores at the brush-stroke level (Section 5.1.5). By setting the vertical stride of the tiles to be $\frac{D_{tile}}{2}$, the model can provide anomaly scores for every pixel in the dataset image except for those in the first $\frac{D_{tile}}{2}$ rows.

A second model is trained simultaneously to read a tile from the bottom to the top. This second model provides alternative anomaly scores that may be averaged with those from the first and fills in the top of the dataset image for which the first model had no context (Section 5.1.6).

An overview of the pixel prediction method is depicted in Figure 5.1 with $D_{tile} = 8$ and $b = 2$. The top-left of this figure represents an input tile with each coloured square representing a pixel. From here, the pipeline progresses via the red arrows during the training phase and the green arrows during the inference phase. The figure shows how, during both phases, the input tile is transformed into a sequence of brush-strokes. In the stack of brush-strokes depicted, the top brush-stroke is the first in the sequence and was taken from the top-left of the input tile. Subsequent strokes progress first across the image and then down as described earlier. In this example, it takes four brush-strokes to cover the top row and eight to cover the top half. Following the training and inference paths one step further, the figure depicts

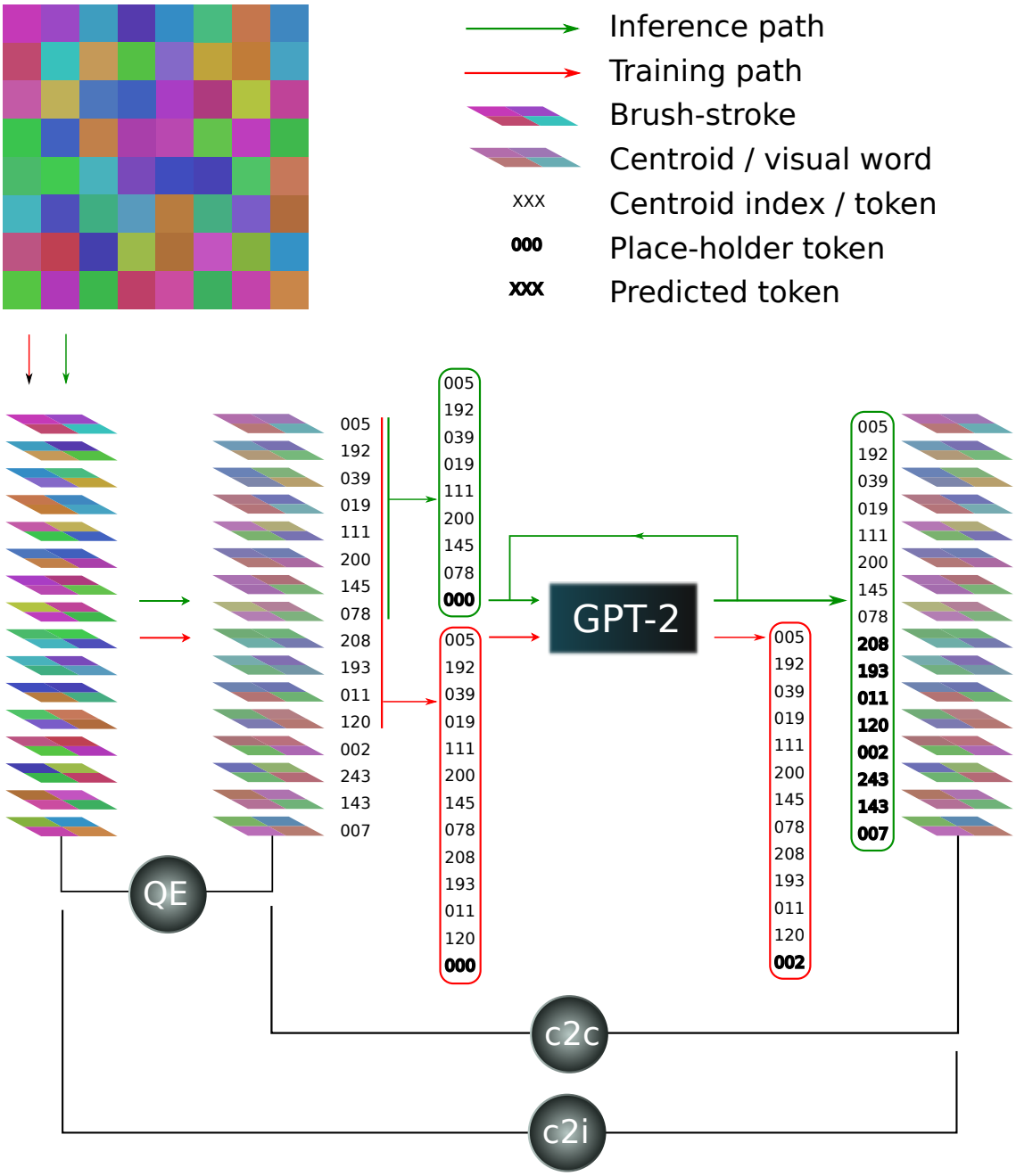

Figure 5.1: A schematic of the pixel prediction method, starting from the input tile (top-left).

each brush-stroke being mapped to its nearest visual word (Section 5.1.1). This forms a sequence of words whose indices are the sequence of tokens that are the input to the GPT-2 architecture (Section 5.1.2). At this stage, there is a divergence in the training and inference paths. During training, we select a random number of tokens greater than half the total number of tokens (Section 5.1.3) while during inference, we select the first half of the tokens that correspond to the top half of the input tile (Section 5.1.4). In either case, we append the place-holder token to the sequence, which the GPT-2 model will replace with its prediction of the next token. This marks the end of the training path. At this point, the error in the predicted token is back-propagated through the GPT-2 model and the pipeline starts again with a new input tile. During the inference phase however, a prediction loop is entered by appending the predicted token to the input sequence and forward propagating through the GPT-2 model again to make the next prediction. This process is repeated until the whole bottom-half of the image is predicted. We now have a choice of how to measure the prediction accuracy of the model. In the figure, *QE* represents the quantisation error, *c2c* represents the centroid-to-centroid distance, and *c2i* represents the centroid-to-image distance. We can also measure the *nearest centroid probability*, but this is not shown in the figure since it takes place inside the box representing the GPT-2 model in the figure. All of these measures of prediction accuracy are discussed in Section 5.1.5.

5.1.1 Forming the Vocabulary

Chen et al. [14] use K-means clustering on the RGB training pixel values to produce a smaller palette of colour values. Each colour value in the palette is represented in the GPT-2 input and output as the value's centroid index, as described above. Under this formulation, one token represents a single pixel. Instead of converting each pixel to a token, we take brush-strokes of size $b \times b$ and convert those into tokens in a similar way. Unless otherwise stated, we use a brush-size $b = 4$. The model then expresses its prediction in terms of a sequence of brush-strokes rather than a sequence of pixels, allowing the model to paint a larger area of image much more quickly. Since the prediction requires a forward propagation per output token in the

output sequence, this drastically increases the speed at the cost of sacrificing some level of model expressiveness. Clearly, a much smaller proportion of the possible brush-strokes are representable with a given vocabulary size than possible pixel values; however, this can be mitigated to some extent by increasing the vocabulary size, which, up to a point, has little impact on speed.

The vocabulary is formed by applying K-means directly on the normal brush-strokes of the dataset class, forming $|\mathcal{V}|$ visual words. We use a vocabulary size $|\mathcal{V}| = 256$. Figure 5.2 shows the set of visual words obtained on the grid class of the MVTecAD dataset.

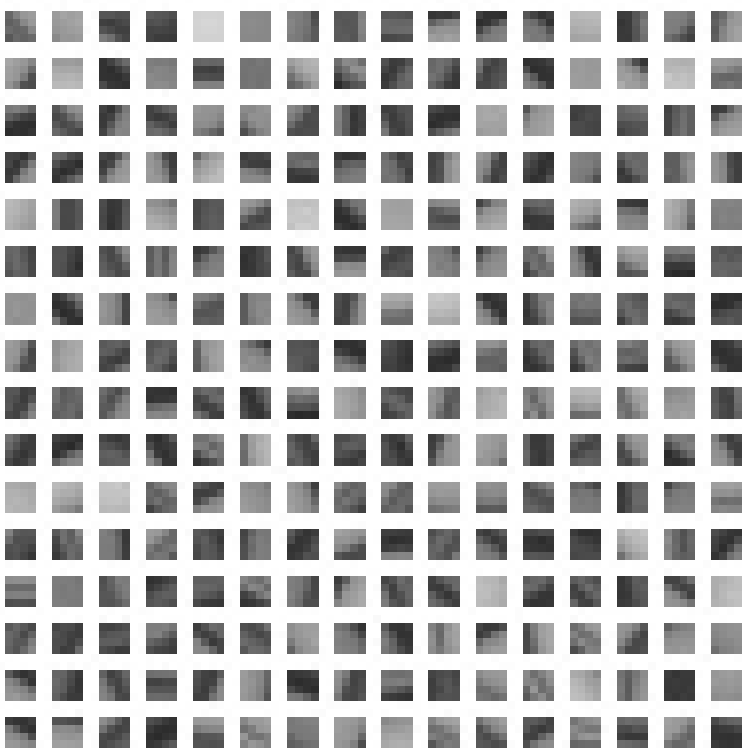

Figure 5.2: Visual words obtained on the grid class of the MVTecAD dataset.

Due to the large number of brush strokes in the training set, we apply stochastic K-means with a batch size of $1,024$ on a random sample of $10,000$ brush-strokes from the training set. In stochastic K-means, rather than applying each iteration on the whole dataset, a random subsample of points is chosen for each iteration.

5.1.2 Architecture

We configure the GPT-2 architecture [77] (Section 2.1.6) to use an eight-block decoder with each block using 16-dimensional token embeddings, dual-headed attention and an MLP-scale of four, as shown in Figure 5.3.

In this figure, the sequence dimension is directed from left to right and the embedding dimension is directed into the page as shown by the legend in the bottom left of the figure. Therefore, this figure depicts a situation where eight context tokens are provided and the model is required to predict the ninth token. The context positions in the sequence correspond to the first eight columns in the figure, while the predicted position in the sequence corresponds to the final column in the figure. Along the right edge of the figure, the numbers represent the length of the embedding dimension. The GPT-2 architecture begins by embedding the input integer tokens in $\mathbb{R}^d$, where $d = 16$ in our case. In addition, the position indices $[0..L_{ctx}]$ are also embedded and added to the token embeddings to encode positional information. These embeddings are learned separately. The embedding dimension remains fixed throughout the architecture except in the hidden layer of the MLP, where our MLP-scale of four extends the embedding dimension temporarily to 64. At the output layer, the output weights project the embedding dimension into $\mathbb{R}^{|\mathcal{V}|}$ so that for each position, there is one output unit per possible token. A softmax activation converts the raw output into a probability distribution over tokens. The decoder block, repeated for i from 0 to 7 in the figure, consists of an attention operation followed by an MLP. The subscripts in the MLP weight matrix symbols indicate that each block and each MLP layer have unique weights, while weights are shared among the sequence positions. This facilitates the use of arbitrary context lengths, which is important during the testing phase. The Gaussian Error Linear Unit (GELU) activation function is used in the MLP. The attention operation requires three inputs: the query Q, the keys K and the values V. The figure shows that all three inputs come from the same set of embeddings, indicating that this is a self-attention operation.

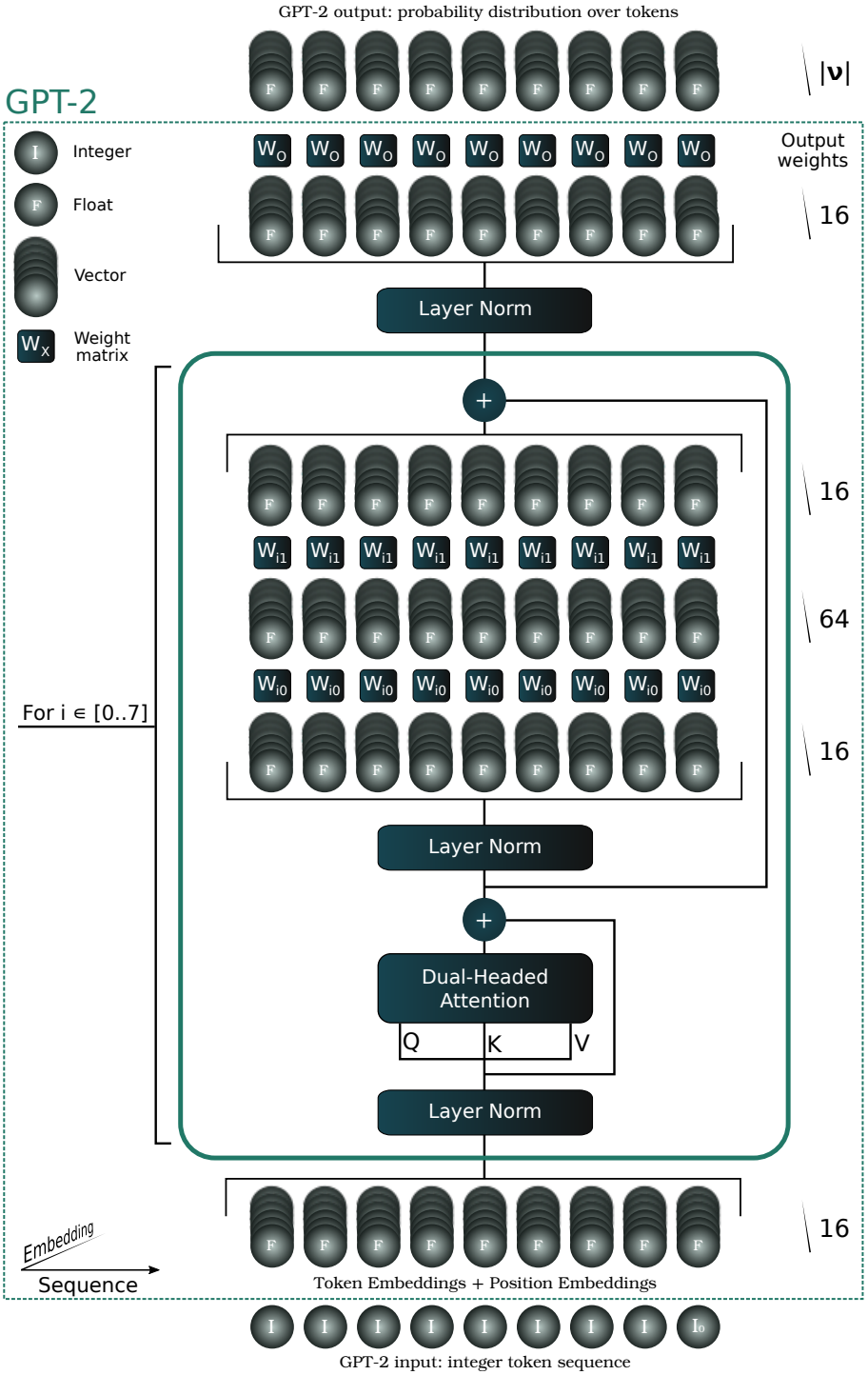

Figure 5.3: Our configuration of the GPT-2 architecture.

5.1.3 Training Procedure

Each training batch consists of $B = 32$ tiles and is assigned a training context length L_{ctx}^{train} uniformly sampled from the range $[L_{ctx}^{test}..|S|-1]$. Sampling a training context length in this way is not part of the procedure employed by Chen et al. [14]. Our justification for doing so is that since the model will be used to generate tokens one at a time during the inference phase, there is a prediction generation loop over context lengths in that range. Our scheme simulates a single randomly selected step in the prediction generation loop, thus training the model over the entire range of context lengths it will encounter while keeping the training inputs randomised and balanced fairly.

Each training tile is mapped to its corresponding sequence S of $|S|$ tokens, of which the first L_{ctx}^{train} tokens serve as context for the model to predict the $(L_{ctx}^{train}+1)th$ token. The training sequence length is $L_{seq}^{train} = L_{ctx}^{train} + 1$, where the final token is the place-holder into which the model will populate the predicted next token. The first L_{ctx}^{train} tokens of S and place-holder are forward propagated through the GPT-2 model to yield the output sequence. The cross-entropy loss is applied to the output tokens with the ground-truth class labels taken to be the first L_{seq}^{train} tokens of S. This loss is back-propagated through the model to train the model to copy the context from the input to the output and to replace the place-holder with the prediction. Figure 5.4 shows an example training input batch and the corresponding output. The image depicting the input batch shows how the input tiles are quantised into a sequence of visual words. Since each word is associated with a token number (the K-means centroid index), the quantised image represents a sequence of tokens starting in the top-left corner and progressing to the right then down. For this particular batch, $L_{seq}^{train} = 159$, which means that the first 158 of these tokens plus a place-holder token are propagated through the model. In the output image we see the 159 output tokens mapped to their corresponding visual word. The first 158 of these are the given context and the $159th$ is a prediction. All positions thereafter are filled with token index zero and are not used.

We use the Adam optimiser [49] and follow Chen et al. [14] in using linear warm-up and cosine annealing with a learning rate of 3×10^{-3}, $\beta_1 = 0.9$ and $\beta_2 = 0.999$.

We train for $150,000$ iterations since we found that was sufficient for producing competent predictions.

5.1.4 Inference Time: Completing the Tiles

We generate output tokens one at a time for a given test tile starting with a context length of $\frac{|S|}{2}$. This means that the model first receives tokens from the upper-half of an input tile as context, and on the first forward propagation generates the first token of the next row of the tile. The model output is a probability distribution over the tokens in $\mathcal{V}$ allowing the possibility to sample this output token from the distribution. The chosen token is then appended to the context sequence and a second forward propagation generates the next token. This process is repeated until the entire lower-half of the tile is generated.

In this way, we are able to generate the unseen lower-half of the test tile. However, it is important to consider that in some cases, numerous pixel sequences could plausibly follow a given context. If the model only predicted one possible tile completion then it may perform poorly due to predicting sequences that are only slightly more probable than others and that are consequently incorrect much of the time. For this reason, we make numerous predictions and choose the version that most closely matches the test sample. This is achieved by repeating the input n_{sample} times along the batch dimension. Since the predicted tokens are always sampled from the output probability distribution, each entry in the extended output batch will have a slightly different completion. Unless otherwise stated, we choose the number

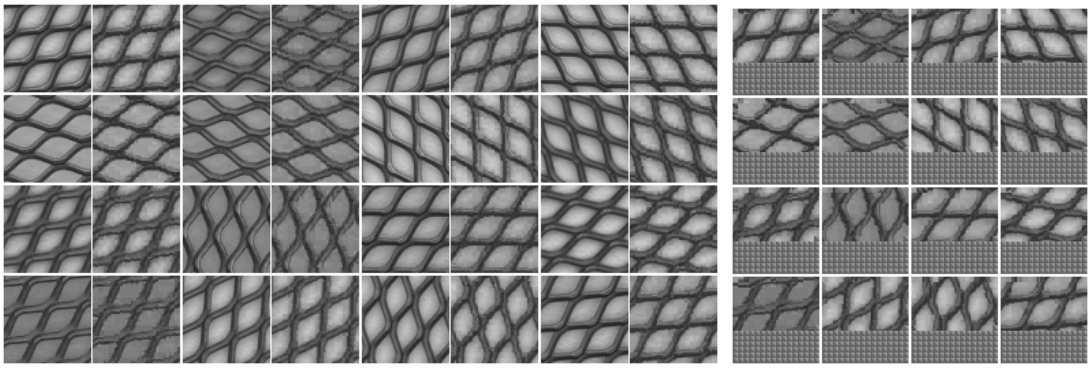

Figure 5.4: An example training batch. Left: the input batch and its quantisation. Right: the output batch.

of samples to be $n_{sample} = 48$. Figure 5.5 shows an example test tile from the grid class and the predictions generated when the top half of this example tile is used as context.

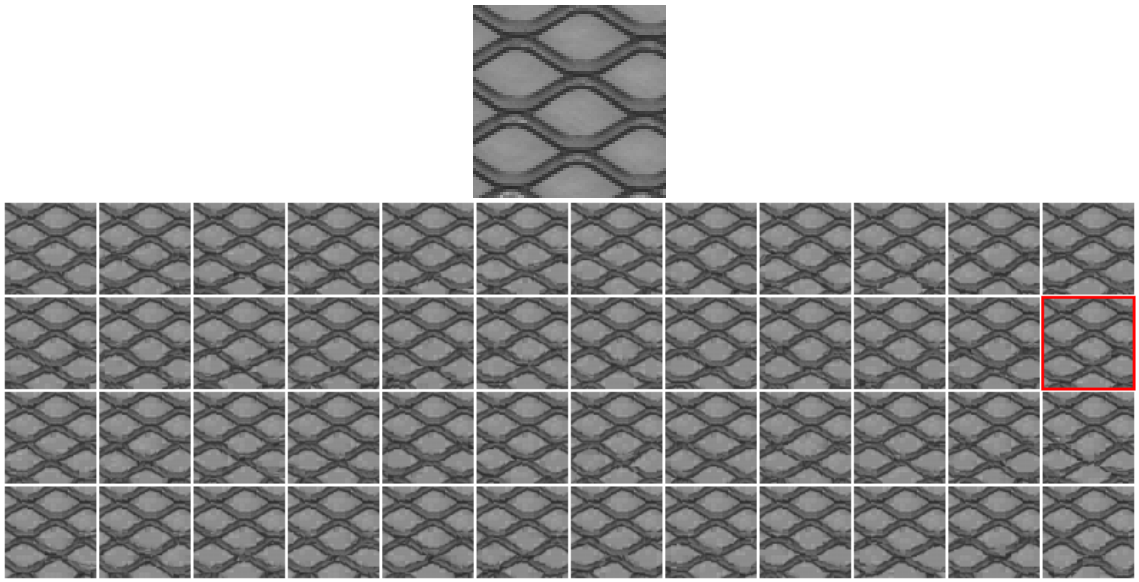

Figure 5.5: Top: an input tile from the grid class of the MVTecAD dataset. Bottom: the n_{sample} predictions generated by the model when given the top half of the above tile as context. The closest prediction is highlighted in red.

The samples look very similar at first glance. However, on closer inspection one can see alternative visual words that have been chosen to texture the grid and mild distortions in the structure that vary from one sample to the next. For the most part, these alternative texture and structure choices vary within normal parameters.

5.1.5 Inference Time: Scoring the Tiles

This method allows a variety of anomaly scoring measures. We apply these scoring measures at the brush-stroke level and assign a uniform anomaly score over the pixels of the brush-stroke. Brush-strokes are typically much smaller than anomalies and so we are still able to obtain reasonably fine-grained anomaly segmentation masks. They are also sufficiently small that in the discussion of the scoring measures below, we assume they carry colour information but negligible structural information.

Quantisation Error (QE)

This scoring measure is particularly fast to compute since we do not require even a single forward propagation through the model. We simply simulate a perfect tile completion by mapping each input brush-stroke to its nearest visual word and then computing the Euclidean distance between each brush-stroke of the original image and its counterpart in the simulated completion. This is essentially using the K-means clustering algorithm alone to predict anomalies. Intuitively, this score will perform well when anomalies present as discolourations but will perform poorly when anomalies present as structural deformations. The extent to which the anomaly detection performance of this method is due to the K-means clustering alone is explored in Section 5.3.1.

Centroid-to-Centroid Distance (c2c)

The anomaly score for each brush-stroke region is the Euclidean distance between the predicted visual word and the visual word closest to the brush-stroke from the original tile.

Centroid-to-Input Distance (c2i)

The anomaly score for each brush-stroke region is the Euclidean distance between the predicted visual word and the corresponding brush-stroke from the original tile. This method for scoring the tiles combines elements from c2c and QE. Errors in the model prediction are associated with unpredictability in structure and result in incorrect visual words; meanwhile, even for a perfect prediction, there remains some distance between the predicted visual word and the original brush-stroke due to the K-means quantisation. This distance captures deviation in the colour of the sample brush-stroke from those seen in the training set.

Nearest Centroid Probability (NCP)

This scoring measure is fast to compute since we do not need to take samples. We start with the input context and forward propagate to produce the output

probability distribution over $\mathcal{V}$ for the first predicted token as usual. At this point we query the probability of the correct token, where the correct token is taken as the index of the visual word nearest the input brush-stroke. The anomaly score for this brush-stroke region is then one minus the found probability. We append the correct token to the input sequence and proceed with the subsequent predicted tokens. Since we automatically fill in the predictions with the correct brush-strokes, there is no need to sample.

5.1.6 Prediction Direction Averaging

As described to this point, this method would produce anomaly scores for all pixels except those that served as context for the top row of tiles. To solve this problem and to incorporate an ensemble into the method, we train two models simultaneously. The first is trained as described, and the second is trained as described except that the tiles are vertically flipped before entering the training and testing pipelines. Consequently, one iteration over one tile will produce anomaly scores for every pixel, where those scores in the top half were assigned by the second model that saw the bottom half as context, and those scores in the bottom half were assigned by the first model that saw the top half as context. A further consequence is that the second model learns to make predictions in row-reversed order, which may influence its decisions. Therefore, we maintain the vertical tile stride at half the tile length so that most pixels will obtain a score from both models. These scores are averaged.

5.2 Results

We test our pixel prediction method on the texture classes of the MVTecAD dataset. Figure 5.6 shows the results obtained using the three different pixel-prediction based measures of abnormality: c2i, c2c and NCP when the method is configured as described in Section 5.1 i.e. $D_{img} = 256$, $D_{tile} = 64$, $b = 4$ and $|\mathcal{V}| = 256$. For the c2i and c2c measures that support sampling, $n_{sample} = 48$. Table 5.1 compares our results with those obtained by state-of-the-art methods.

We find that the best and most consistent results are obtained on the grid class,

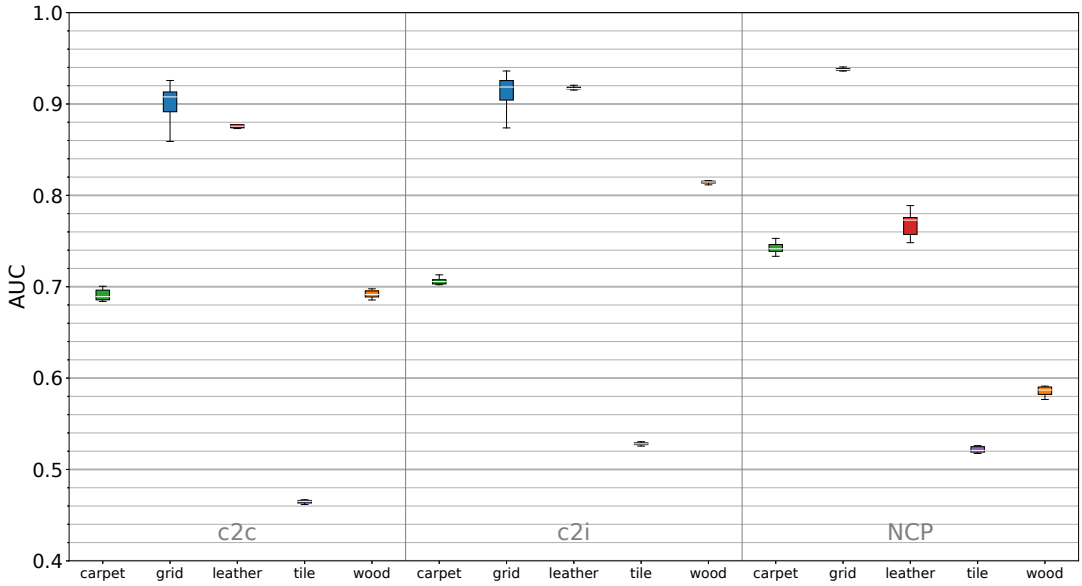

Figure 5.6: Box plot of AUC scores obtained by pixel prediction on the MVTecAD texture classes when configured as described in Section 5.1.

Table 5.1: Anomaly detection performance using pixel prediction (AUC).

Method	Carpet	Grid	Leather	Tile	Wood	Mean
FCDD [61]	**0.96**	0.91	**0.98**	0.91	0.88	**0.93**
P-Net [63]	0.57	**0.98**	0.89	0.97	**0.98**	0.88
SMAI [56]	0.88	0.97	0.86	0.62	0.80	0.83
VAEgrad [21]	0.74	0.96	0.93	0.65	0.84	0.82
VEVAE [60]	0.78	0.73	0.95	0.80	0.77	0.81
CNN Feats. [71]	0.72	0.59	0.87	0.93	0.91	0.80
Ours: c2i	0.71	0.92	0.92	0.53	0.81	0.78
GANomaly [5]	0.70	0.71	0.84	0.79	0.83	0.78
AE (SSIM) [12]	0.87	0.94	0.78	0.59	0.73	0.78
Ours: c2c	0.69	0.91	0.88	0.46	0.69	0.73
Ours: NCP	0.74	0.94	0.77	0.52	0.59	0.71
AE (L_2) [12]	0.59	0.90	0.75	0.51	0.73	0.70
GMM [13]	0.88	0.72	0.97	0.41	0.41	0.68
AnoGAN [85]	0.54	0.58	0.64	0.50	0.62	0.58

while by far, the worst performance is on the tile class. The discrepancy between these classes is explored in the analysis in Section 5.4. As shown in Table 5.1, our method does not reach the state-of-the-art standard set by some contemporary anomaly detection approaches [63, 61, 56]; however, it does comfortably outperform some of the previous generation state-of-the-art methods such as AnoGAN [85], especially when taking the c2i anomaly score.

Overall, the c2i anomaly score is the strongest performer across these dataset classes. We hypothesise that this is because the c2i score is the only one to be influenced by both medium-scale structural predictability and small-scale colour novelty. Combining both of these aspects appears to be stronger than using just one or the other. The c2i anomaly score dominates the c2c anomaly score, surpassing its performance on every class; however, is is beaten by NCP on the carpet and grid classes by a small margin. NCP is often either the best anomaly score or matches c2i but curiously, its performance drops significantly on the leather and wood classes. There may be some benefit in combining these procedures rather than using them in isolation. In particular, combining the c2i and NCP procedures could be interesting because they each consider opposite aspects of the problem: c2i considers how close the most likely pixels are to the ground truth, whereas NCP considers the likelihood of the ground truth. Unfortunately, these anomaly scores are very different in character and are not trivial to combine in a meaningful way. The c2i score produces a distance in image space that behaves linearly: twice the score implies twice the distance. In contrast, the NCP score is calculated via a probability. In theory, twice the probability implies twice the likelihood but in practice, the true interpretation of the probability figure is unknown and is unlikely to be linear. We base this on the observed behaviour of classification networks, where measurement of confidence through analysis of the output probability distribution is often unreliable. Therefore, we anticipate that the solution to this problem will be much more involved than a weighted combination of scores. The NCP score may be improved by considering the top-N nearest visual word probabilities. Figures 5.7 - 5.11 show example detections using the c2i anomaly score.

Using pixel prediction, it is easy to detect anomalies in the structure of grid class

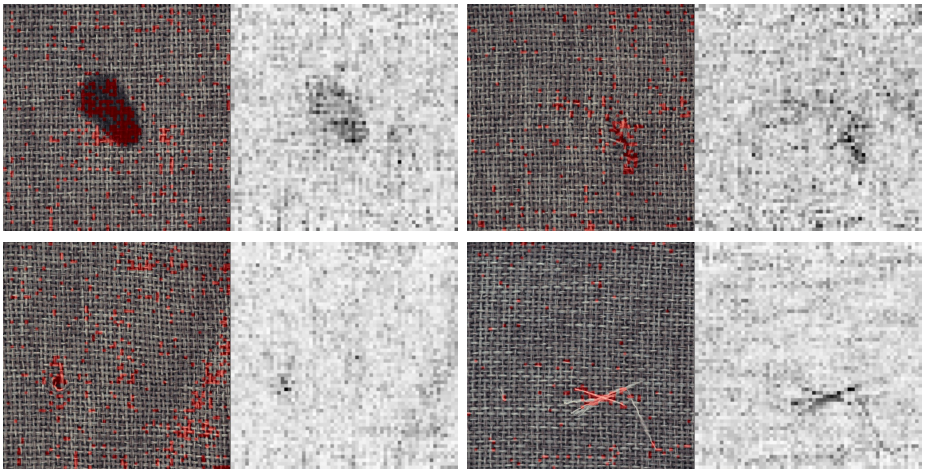

Figure 5.7: Example detections on the carpet class of the MVTecAD dataset using the c2i anomaly score. Right images show the per-pixel anomaly scores with darker pixels indicating larger scores. Left images show the dataset image with detections overlaid in red.

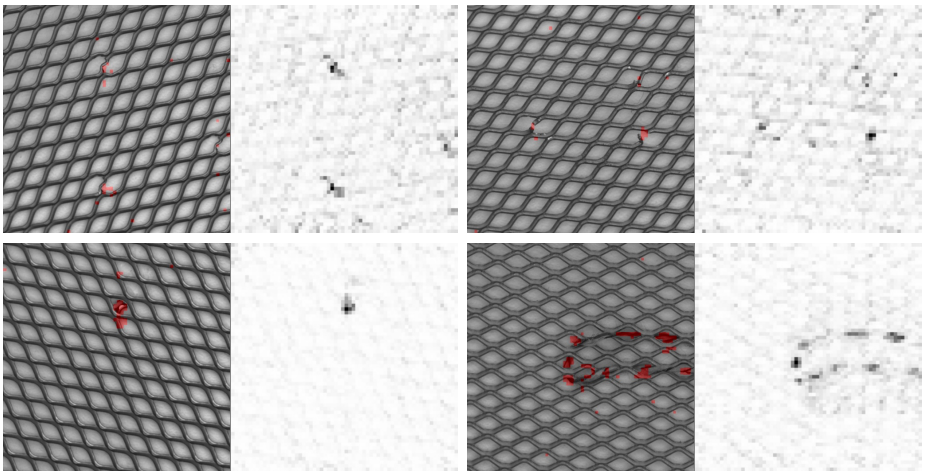

Figure 5.8: Example detections on the grid class of the MVTecAD dataset using the c2i anomaly score. Right images show the per-pixel anomaly scores with darker pixels indicating larger scores. Left images show the dataset image with detections overlaid in red.

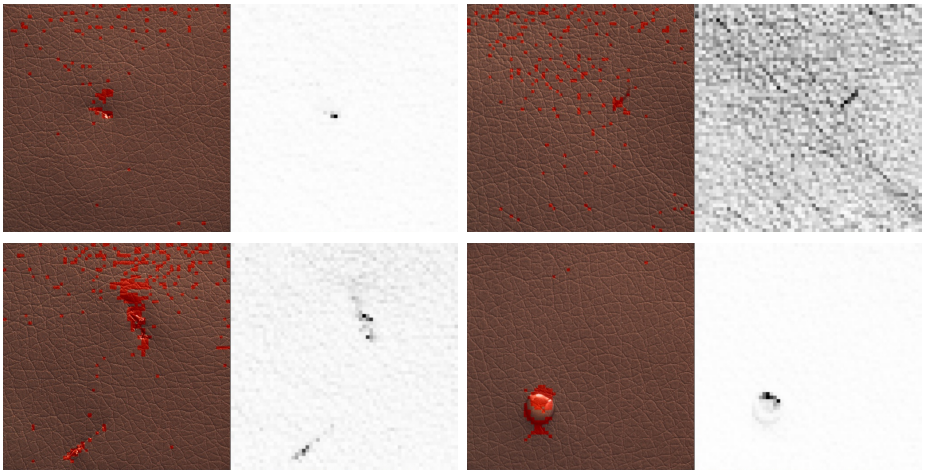

Figure 5.9: Example detections on the leather class of the MVTecAD dataset using the c2i anomaly score. Right images show the per-pixel anomaly scores with darker pixels indicating larger scores. Left images show the dataset image with detections overlaid in red.

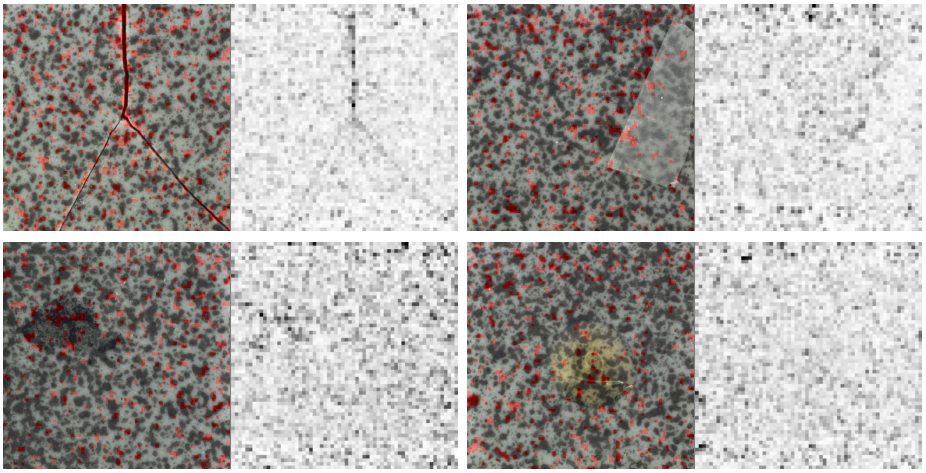

Figure 5.10: Example detections on the tile class of the MVTecAD dataset using the c2i anomaly score. Right images show the per-pixel anomaly scores with darker pixels indicating larger scores. Left images show the dataset image with detections overlaid in red.

images. In the case of bends or breaks, a line of grid material exhibits a discontinuity where the model predicts that it should continue in a predictable manner as shown in Figure 5.12. Likewise, the presence of foreign material is detected without much difficulty as shown in Figure 5.13.

Often, this method has difficulty where normal variation is quite diverse. In the

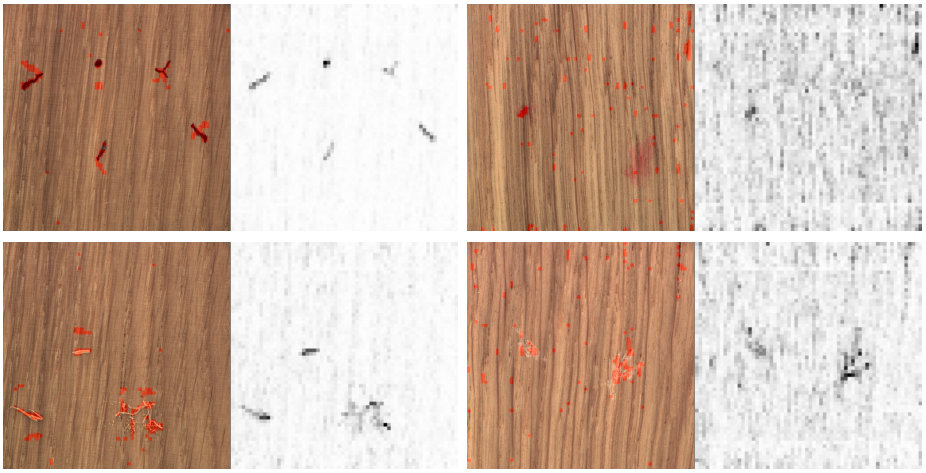

Figure 5.11: Example detections on the wood class of the MVTecAD dataset using the c2i anomaly score. Right images show the per-pixel anomaly scores with darker pixels indicating larger scores. Left images show the dataset image with detections overlaid in red.

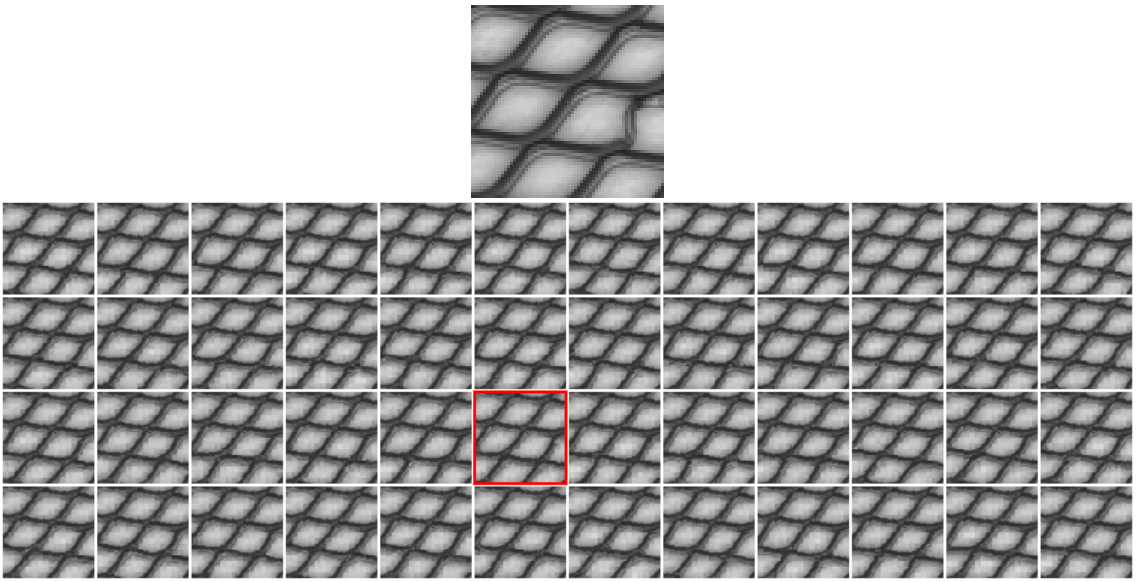

Figure 5.12: Top: an input tile from the grid class of the MVTecAD dataset in which there is a bend in the grid structure. Bottom: the n_{sample} predictions generated by the model when given the top half of the above tile as context. The closest prediction is highlighted in red.

carpet class, we observe an increased density of false-positive detections where the colour tone of the carpet suddenly but subtly changes and where the weave is subtly distorted. Similarly, in the leather class, false-positive detections often cluster along the cracks where it is difficult to determine the precise route they will take and all kinds of routes would be normal. In Figure 5.14 we observe the model predicting all manner of crack-routes across the 48 samples. Due to the difficulty of predicting these routes, more samples may benefit the performance of the method on this class. In the same figure, we also observe the behaviour of the K-means clustering alongside the behaviour of the pixel prediction model. In the context region, the clustering is responsible for concealing the rip to some extent; while in the predicted region, the model joins forces with the clustering to repair the rip altogether. This causes a large distance between the anomalous regions of the input and predicted tiles as intended.

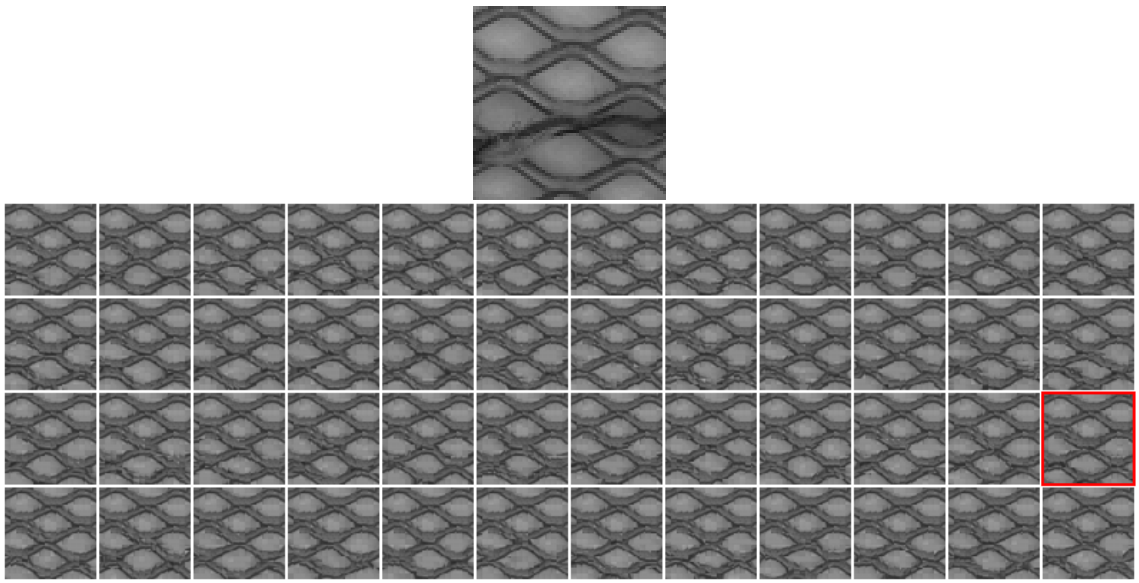

Figure 5.13: Top: an input tile from the grid class of the MVTecAD dataset that contains some foreign material. Bottom: the n_{sample} predictions generated by the model when given the top half of the above tile as context. The closest prediction is highlighted in red.

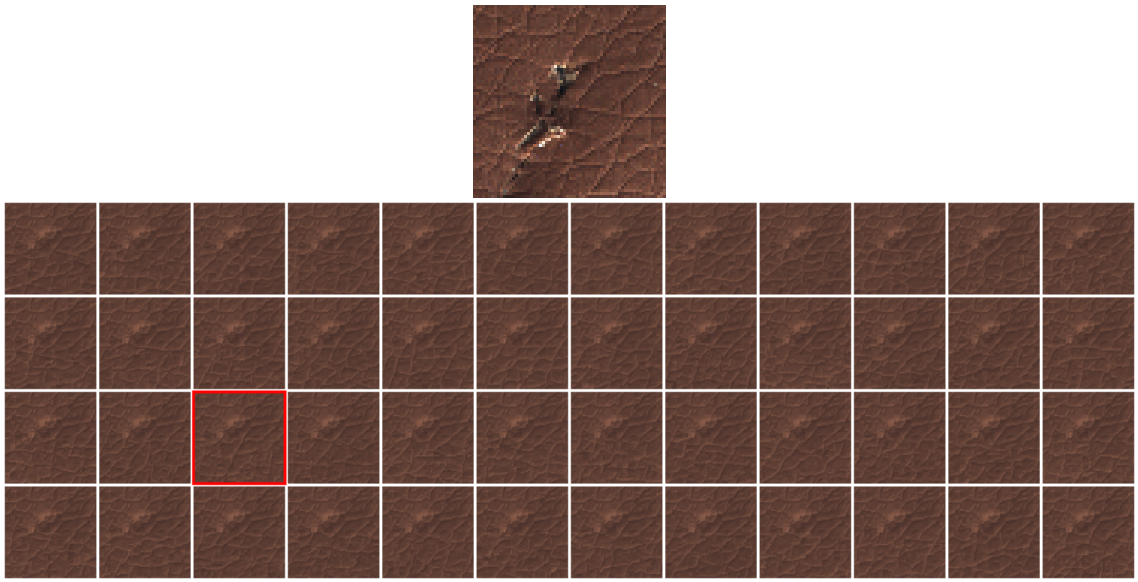

Figure 5.14: Top: an input tile from the leather class of the MVTecAD dataset in which the leather is ripped. Bottom: the n_{sample} predictions generated by the model when given the top half of the above tile as context. The closest prediction is highlighted in red.

5.3 Analysis

Each of the following subsections examine an isolated aspect of the method described in Section 5.1. The aspect is modified in some way or removed and the new results are compared with those displayed in Section 5.2 to see what difference it made. This section examines the extent to which the anomaly detection performance was due to K-means clustering rather than prediction (Section 5.3.1), the benefit of taking multiple samples for the c2c and c2i anomaly scoring schemes (Section 5.3.2), and the effect of using a secondary model for simultaneous backwards prediction (Section 5.3.3).

5.3.1 Anomaly Detection by Clustering

The accuracy of the future pixel prediction is affected not only by what the model has learned about the patterns in normal sequences, but also by the clustering algorithm that affects what is interpretable and expressible by the model. If, for example, all normal samples of carpet have a particular colour, then the K-means algorithm will not find visual words that are of a vastly different colour. At test time, we may encounter a sample of carpet that has paint spilled on it, drastically changing the colour of the carpet. In this case, tokenisation will map this sample to a sequence of tokens representing regularly coloured brush-strokes. One could imagine that in such cases, successful anomaly detection could be achieved by dispensing with the model entirely and simply taking a distance measure between the input brush-strokes and their nearest visual word. We perform this experiment and present the box-plot of AUC results in Figure 5.15. Dashed lines are used to compare these box plots with those from Figure 5.6, in which future-prediction is employed.

In most cases, not using the future pixel prediction models severely harms performance. This is not the case for the tile dataset that performed very poorly under future pixel prediction but was greatly improved under pure quantisation error. Section 5.4 explores the tile dataset further and offers an explanation for why this may be the case. Overall, these results suggest that there is some utility in using the Transformer-based models for predicting upcoming pixel sequences. This

allows for detection of anomalies in the structure of patterns such as warps, bends, breaks, rips and folds that do not necessarily entail deviations in colour.

5.3.2 Sampling

As described in Section 5.1, we produce a number of alternative plausible tile completions based on the tile's top half by sampling from the model output distribution. The results discussed in Section 5.2 were produced using $n_{sample} = 48$, allowing the method to choose the most accurate prediction from among 48 alternatives. This applies to the c2c and c2i anomaly scores for which sampling is relevant. Using the same models, but setting $n_{sample} = 1$, we produce the results shown in Figure 5.16.

With the exception of the tile class on both anomaly scores and the leather class on the c2c anomaly score, the results are worse in every case, which supports the claim that taking multiple samples is good for performance.

There are a few different scenarios to consider when reasoning about the effect of sampling from a purely theoretical perspective. Given a tile with an anomaly contained entirely outside of the context, it is inconceivable that any number of

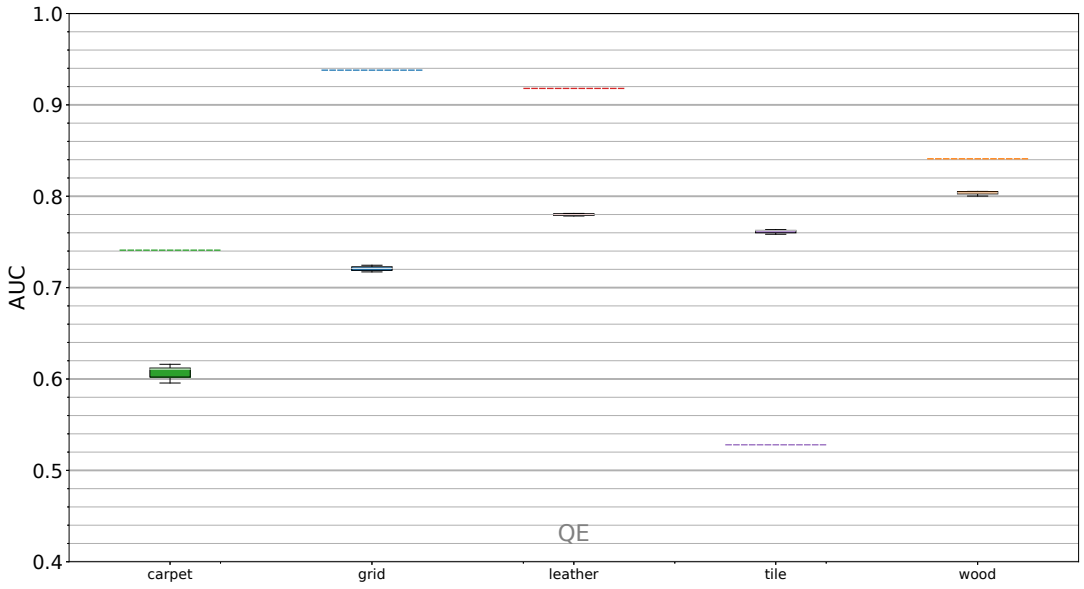

Figure 5.15: The AUC performance when using the QE anomaly score. Dashed lines show the greatest median performance achieved when using future pixel prediction as displayed in Figure 5.6.

samples will generate the anomaly, in which case, more samples can only increase performance. Given a tile with an anomaly contained entirely inside of the context, the model should be able to generate accurate tile completions, but the novel sequence of tokens in the model input may decrease the probability of this happening. In this way, the anomaly may cause high anomaly scores in the neighbouring predicted region that is normal. In this case, sampling mitigates this effect by offering the model more chances to predict the correct sequence. Given a tile with an anomaly spanning the context and prediction regions, sampling may have afforded the model more chances to generate the anomaly and in this case, may have negatively affected performance. We observe that this does not typically happen. For example, Figure 5.17 shows a tile from the tile class that has a crack running vertically down the middle. In the samples, we observe that the crack usually terminates on entry to the predicted regions. It is interesting to note that in some samples, the model does attempt to continue the crack down into the predicted region for a short while using a centroid that is close in colour, but the model always favours eventually returning to a more typical distribution of pattern.

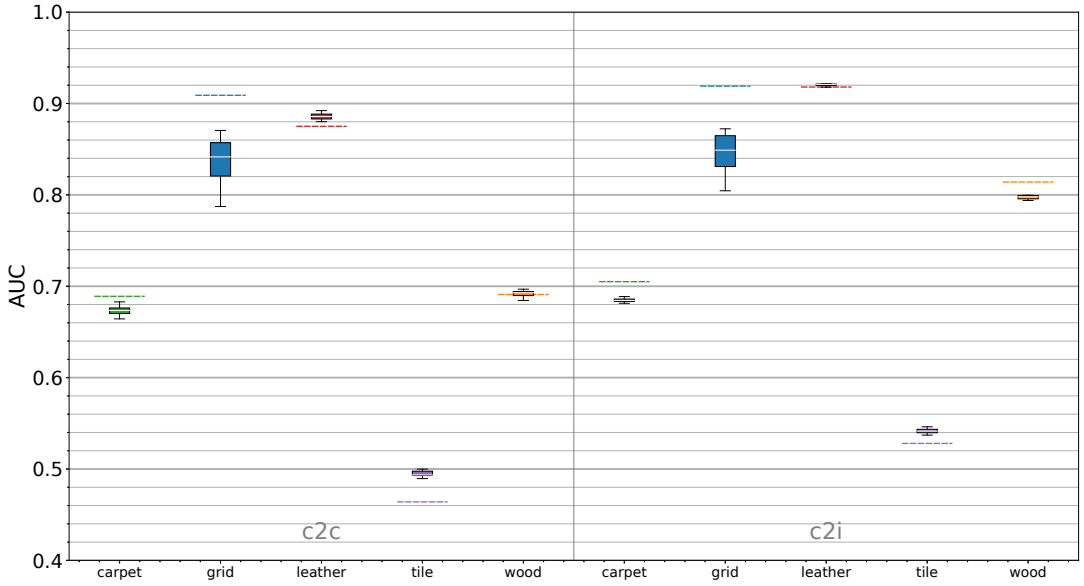

Figure 5.16: AUC results obtained from the same model used to obtain the results in Section 5.2 when only one sample prediction is taken. Dashed lines show the results from Section 5.2.

5.3.3 Prediction Direction Averaging

The results shown in Section 5.2 were produced by averaging the anomaly scores assigned by two different models: one that is passed sequence tokens from top-left to bottom-right and one that is passed sequence tokens from bottom-left to top-right (Section 5.1.6). To investigate the utility of performing this averaging, we measure the performance of each of these models in isolation. A consequence of only using a single model is that there will be an unreachable block of pixels located either at the very top or very bottom of each dataset image, depending on which model is being investigated. These pixels are discarded before calculating the AUC scores.

Figure 5.18 displays the AUC results achieved by each of the two models in isolation. In the left half of the figure, the results are from the model that receives the tiles without flipping and therefore takes the top-half as context. In the right half of the figure, the results are from the model that receives the flipped tiles and therefore takes the bottom half as context. The dashed lines are for comparison with the base-line results from figure 5.6 in which the two models are combined.

When using the c2i and c2c anomaly scoring methods, performance is always improved by using both models in combination. Even the tile class, which has

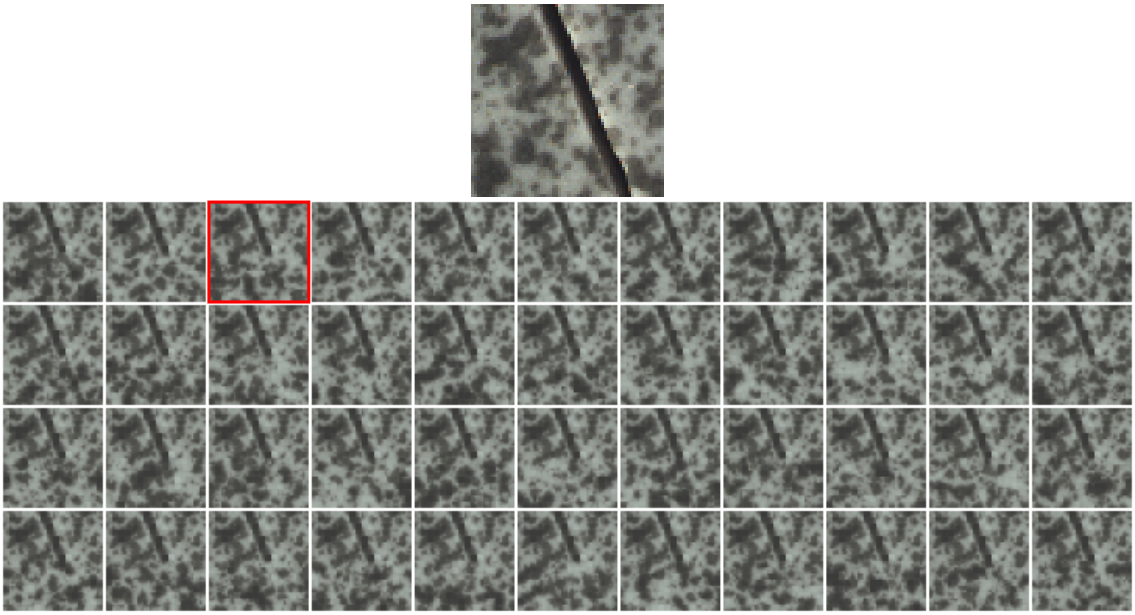

Figure 5.17: Top: an anomalous input tile from the tile class of the MVTecAD dataset. Bottom: the n_{sample} predictions generated by the model when given the top half of the above tile as context. The closest prediction is highlighted in red.

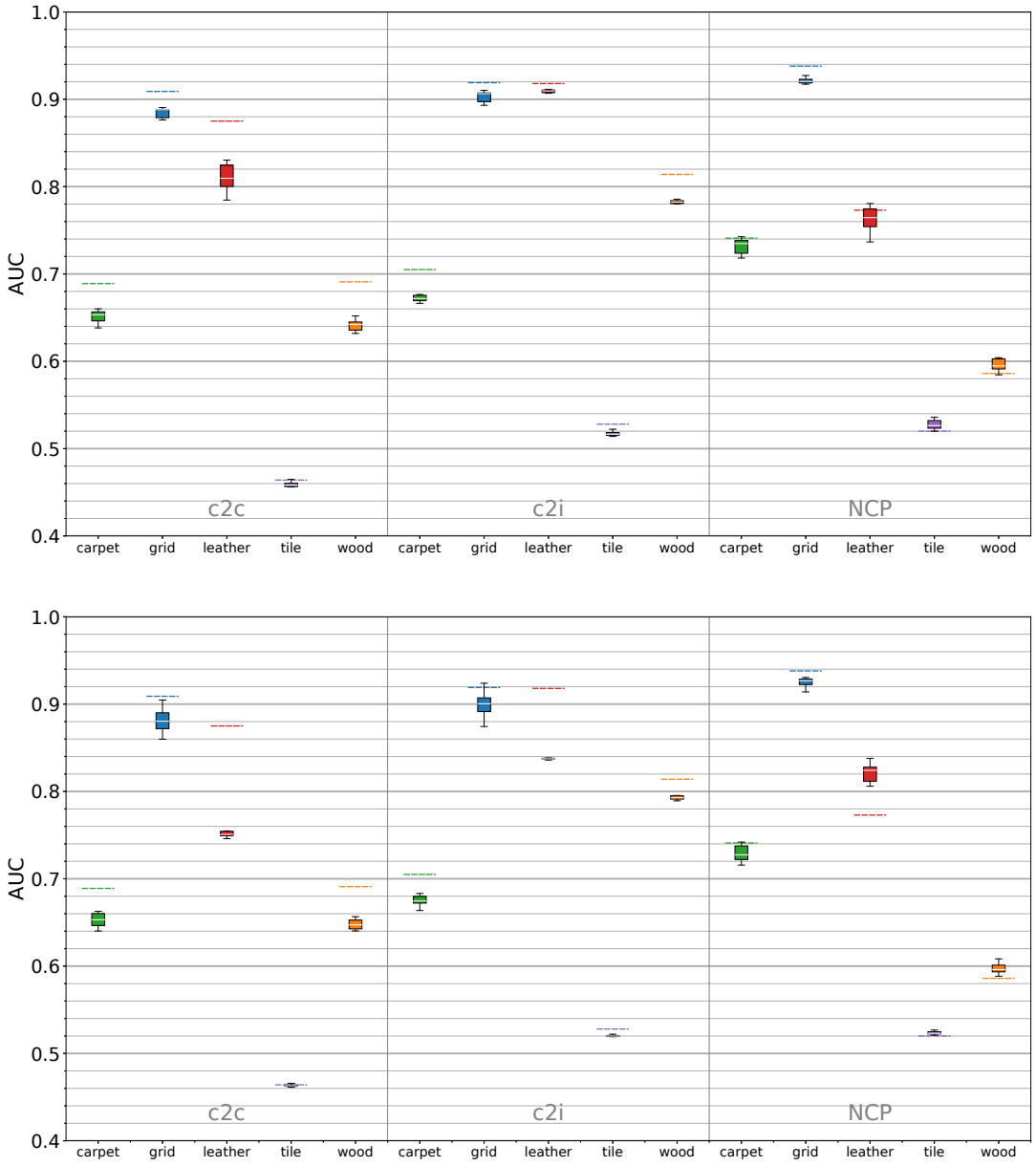

Figure 5.18: AUC results obtained from each of the models in isolation using each of the three model-based methods of anomaly score. Top: the model that uses the top-half of the tile as context to predict the bottom-half. Bottom: the model that uses the bottom-half of the tile as context to predict the top-half. Dashed lines show the median performance of both models combined as displayed in Figure 5.6.

acquired a reputation for breaking trends, is marginally improved. In contrast, when using the NCP scoring method, performance on the tile and wood classes is slightly hindered. The most interesting observation from these results is that for the leather class, there appears to be a preference for reading the tiles in row-reversed order i.e. the model that receives the bottom-half of the tile as context has the advantage. This behaviour is very curious, and has also been observed in preliminary experiments under different conditions. There is a systematic asymmetry in the images of the leather class: cracks present as dark lines with light lines running underneath, which does cause the character of the images to change when they are flipped; however, it is unclear if this is the cause and why this would be important.

5.4 The Tile Class

The results obtained on the tile class were very poor. This section explores why this may have been the case, drawing comparisons with the grid class that performed very well.

Figure 5.19 shows the n_{sample} predictions generated by the model for the shown test tile. Unlike the grid example shown in Figure 5.5, the diversity in the samples is very large. Although all variations predicted by the model are plausible, they do not begin to cover the full range of possibilities because the pattern exhibits very high entropy. It is very hard to predict exactly how the tile pattern will continue to develop from the context because exactly where the mottling in the pattern will occur is only weakly related to where it has been distributed before. A dataset like this may need very many more samples, but this will make the distance-based anomaly scores very slow to compute. Alternatively, this class may benefit from less downscaling and a smaller tile size, both of which will restrict the degree of possible variations within an input tile.

This inherent lack of predictability is evident in the model output distribution over centroids. Figure 5.20 depicts a typical output distribution for both the grid and the tile classes. For the grid class, the model is often able to predict with high confidence the centroid that is supposed to come next. On the other hand, for

the tile class, the model output is much closer to a uniform distribution and so we observe that the high entropy in the texture is reflected in the high entropy in the output distribution.

However, as shown in Figure 5.17, the model does fail to accurately predict anomalous patterns as intended. The failure of the tile class is therefore due to the model's inability to predict normal sequences due to their inherently random nature.

The findings in Section 5.3.1 also support the hypothesis that it's the entropy of the tile class that is to blame. There, we saw that the tile class is the only one

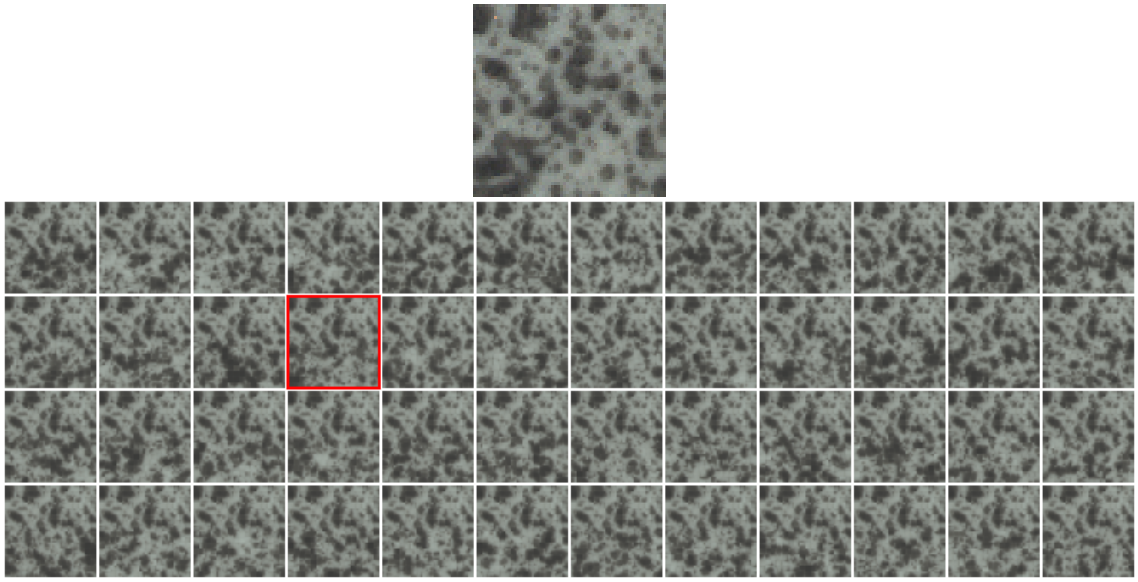

Figure 5.19: Top: a normal input tile from the tile class of the MVTecAD dataset. Bottom: the n_{sample} predictions generated by the model when given the top half of the above tile as context. The closest prediction is highlighted in red.

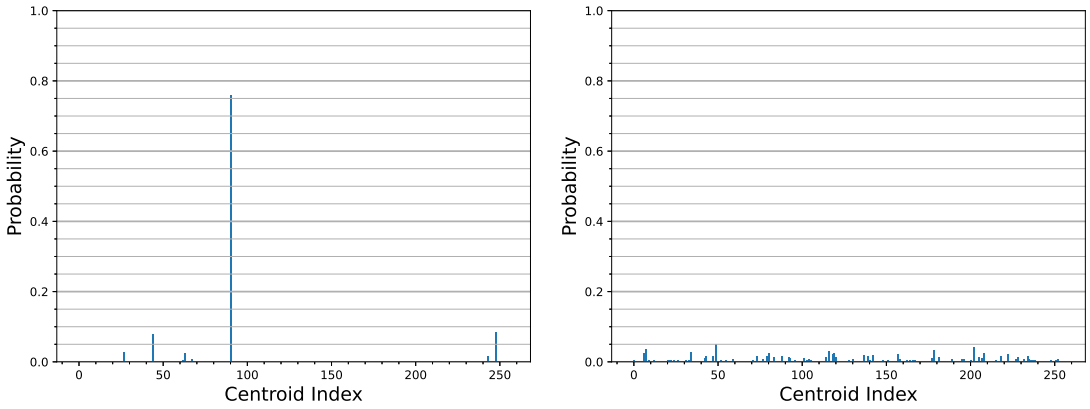

Figure 5.20: Bar charts showing the typical probability distributions over centroids obtained on two of the classes. Left: grid. Right: tile

to benefit from disabling the future pixel prediction and relying exclusively on the quantisation error.

5.5 Feature Prediction

We propose a variant that predicts features, rather than pixels. To form the new vocabulary, we first propagate each brush-stroke through a pretrained VGG-11 network [89] to yield its feature representation. In this case, features are extracted from the third convolutional layer, and the brush-size is chosen to ensure that at that layer, the network activations have collapsed to a size of 1×1 in the spatial dimensions. The features are then used in the K-means algorithm as before. Based on a range of preliminary experiments, we choose a tile size of $D_{tile} = 96$ and extract features from the third convolutional layer after ReLU activation is applied, since this configuration gave the most consistent performance. This requires a brush-size $b = 12$ to collapse the features in the spatial dimensions as described above. Figure 5.21 compares the results obtained using future feature prediction with those obtained using future pixel prediction.

In almost every case, results are significantly improved when predicting features rather than pixels. The only notable exception is the leather class for which the performance is adversely affected. It is not clear why leather in particular should be affected in this way.

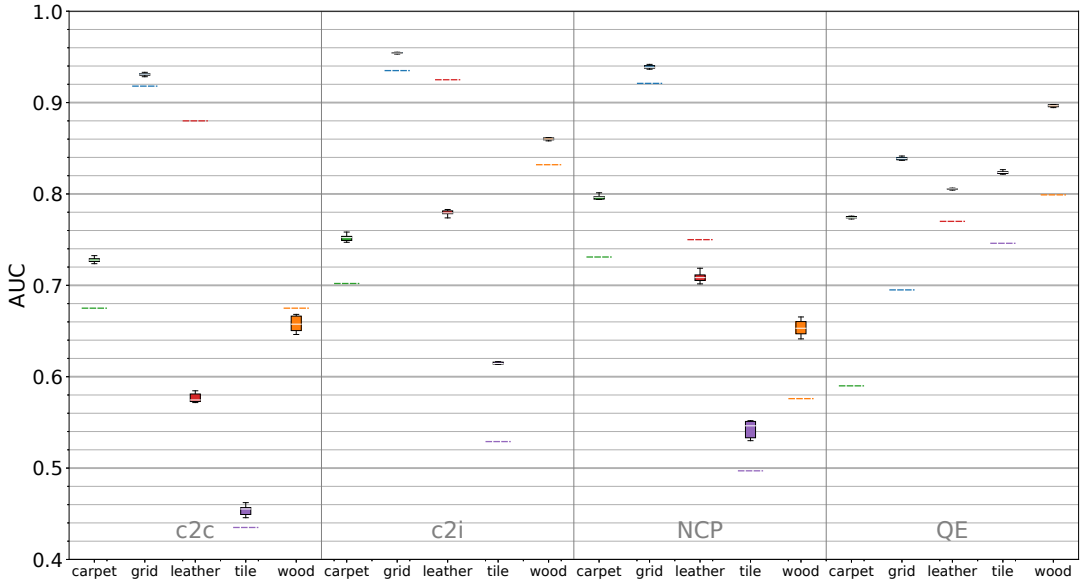

Figure 5.21: AUC results obtained when using future feature prediction rather than future pixel prediction. Features are extracted from the VGG-11 network [89] at the third convolutional layer after ReLU activation is applied. Dashed lines show the performance of future pixel prediction.

5.6 Conclusion

We contribute a first attempt at targeting anomaly detection through Transformer-based pixel prediction. At the time of experimentation there were no Transformer-based anomaly detection methods and at the time of writing there are still no Transformer-based pixel-prediction methods to our knowledge. We are able to very consistently generate a range of plausible future pixel predictions based on a given input context, which leads to competent anomaly detection in most cases; however, when textural patterns exhibit high entropy, the variability in plausible patterns limits the anomaly detection performance. Nevertheless, we outperform some of the previous generation state-of-the art methods such as AnoGAN [85].

We have shown that the method is made more robust by sampling multiple tile completions and by ensembling models that read the data in different directions. In principle, both of these techniques may be extended and it is likely that this would improve results. Extra samples could be taken, and extra models could be trained to predict horizontally rather than vertically, but this would come at a significant computational cost. In addition, we showed that making predictions in the feature-space rather than the pixel-space most often leads to improved performance.

CHAPTER 6

Cross-Context Evaluation of Methods

The previous contribution chapters evaluated methods applied within their own anomaly detection subdomains. Chapter 3 evaluated our KDE-based method on a CCTV dataset, a domain where anomalies are at the object-level, while Chapters 4 and 5 evaluated the methods they present on the MVTecAD dataset, a domain where anomalies are more pixel-level defects. Here, we investigate the performance of our methods when applied across these two genres of anomaly detection.

6.1 KDE-based detection on MVTecAD

Figure 6.1 shows the AUC results obtained by the KDE-based method presented in Chapter 3 on the texture classes of the MVTecAD [11] dataset. Since the application of the region extractor makes little sense in this context, we use manually extracted regions of side-length 64. During the training phase, the regions are extracted from 50 random locations per frame. During the inference phase, the regions are tiles spanning the entire frame. We downscale images to 512×512, since this is the middle of the range of sizes experimented with in Chapter 4. We use the layer six variant of the feature extractor without ReLU activation and 48 PCA components.

The features from layer six without ReLU activation proved to be the best on the Ped2 dataset, and 64 PCA components gave a good balance between performance and computational complexity.

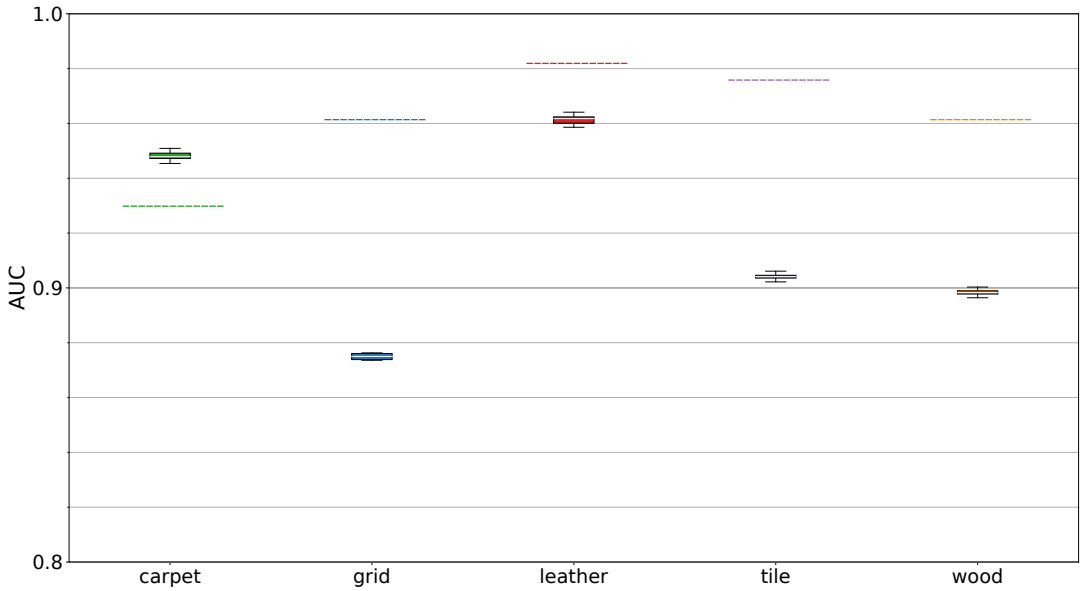

Figure 6.1: AUC results obtained by our KDE-based method on the MVTecAD texture classes. Dashed lines represent the performance of our best performing method shown in Figure 4.4

The performance on the texture classes is very good on the whole, especially considering that the texture classes are very different to those that the feature extractor was originally trained on. This demonstrates the generality and transferability of the layer six features. When comparing these performance metrics with our other two methods, it should be understood that this method is a region-level anomaly detection method, while the other two are pixel-level anomaly segmentation methods. Counter-intuitively, advantage is gained by the region-level detector when being evaluated against a pixel-level ground-truth. Since normal pixels usually outnumber anomalous pixels by a large margin, it is far better to apply high anomaly scores liberally in anomalous areas, striking all anomalous pixels at the expense of also striking some peripheral normal ones, than it is to accurately segment anomalies, missing almost all normal pixels but also missing a few anomalous ones. Indeed, one easy way to significantly improve results for our two anomaly segmentation methods is to simply blur the anomaly heat maps produced so as to bleed detections into

the normal regions, but we consider this not really within the original spirit of the evaluation criteria intended by the dataset designers [11].

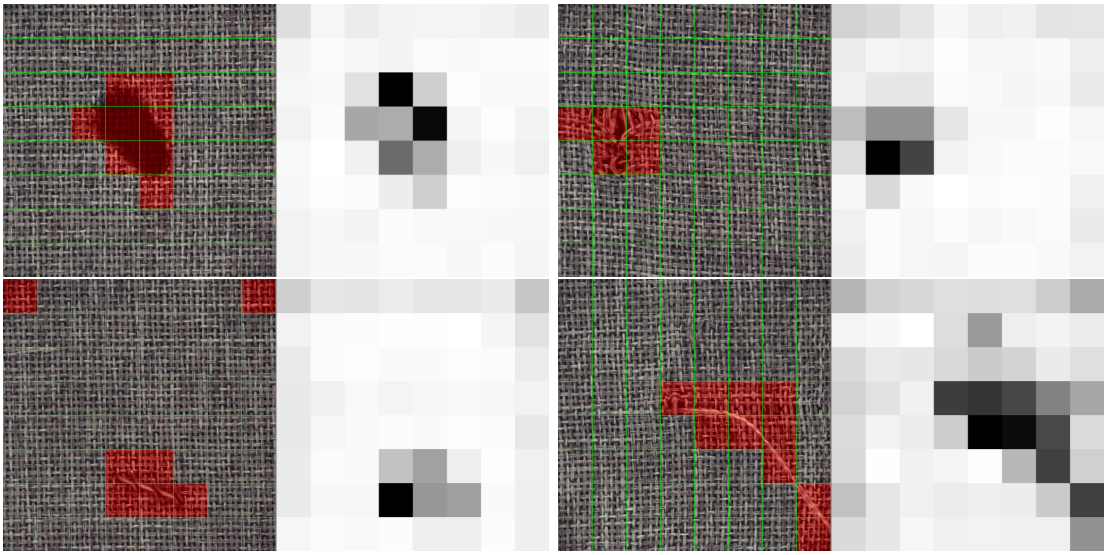

Figure 6.2: Example anomaly detections obtained by our KDE-based method on the carpet class of the MVTecAD dataset.

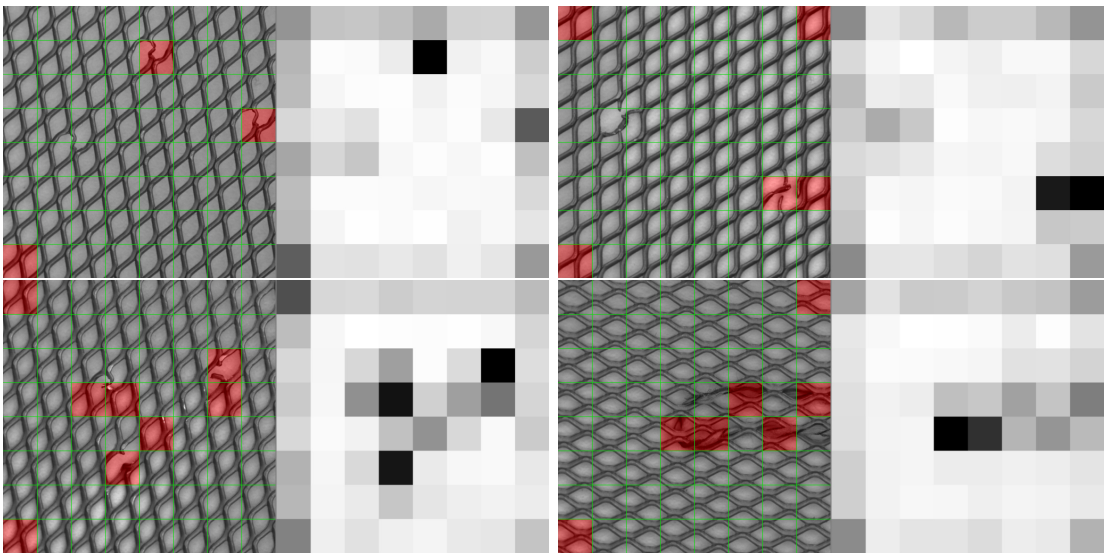

Figure 6.3: Example anomaly detections obtained by our KDE-based method on the grid class of the MVTecAD dataset.

Figures 6.2 - 6.6 show example anomaly detections on the same dataset classes. Some kinds of anomaly are sometimes only partially detected. For anomalies where this is the case, the example shown is from a poorly detected anomaly, but it would have been possible to cherry-pick a perfect example. In the examples for the three poorest performing classes: grid, tile and wood, we can see the reason for the lower

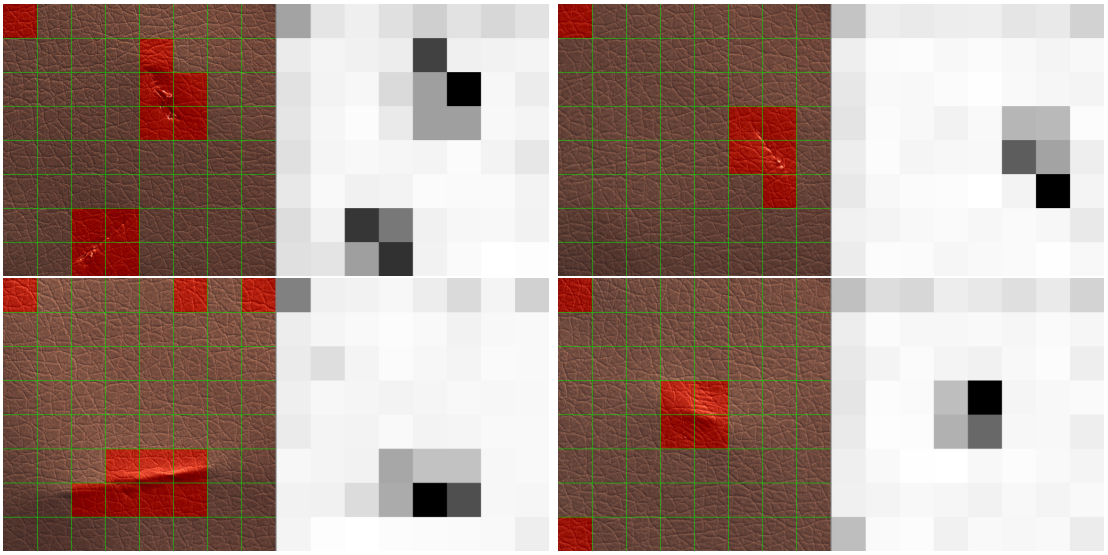

Figure 6.4: Example anomaly detections obtained by our KDE-based method on the leather class of the MVTecAD dataset.

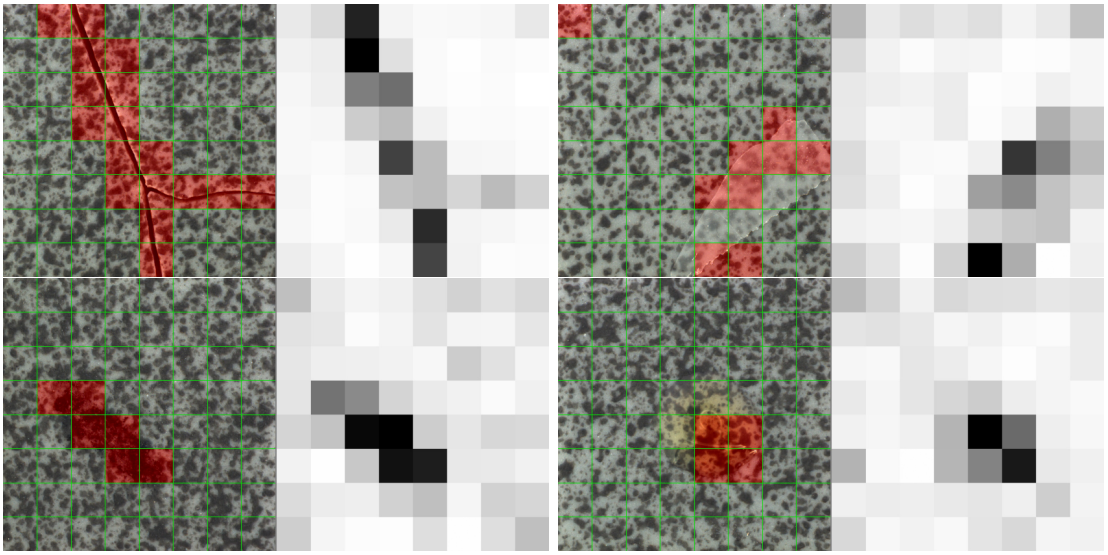

Figure 6.5: Example anomaly detections obtained by our KDE-based method on the tile class of the MVTecAD dataset.

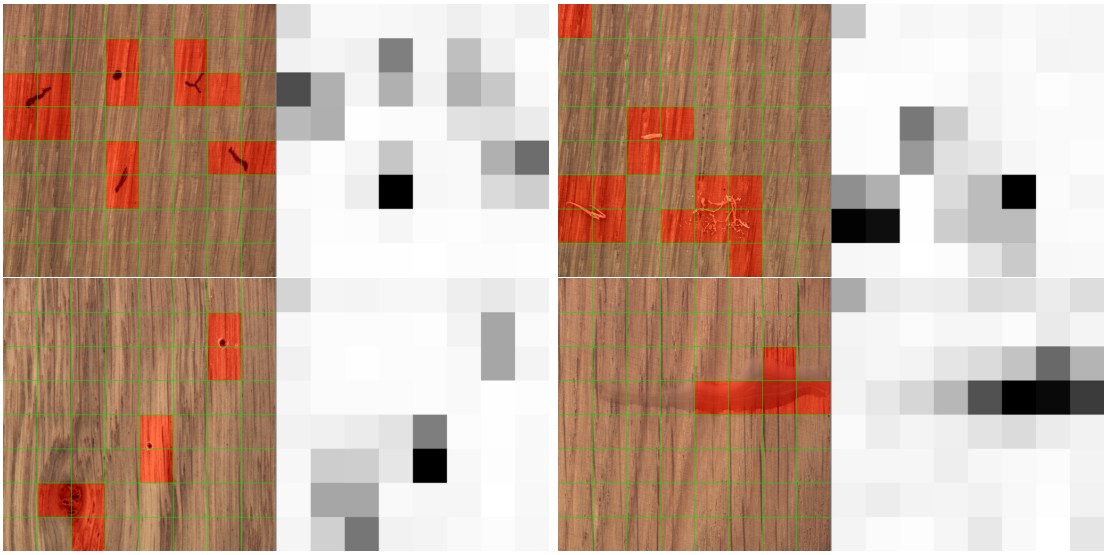

Figure 6.6: Example anomaly detections obtained by our KDE-based method on the wood class of the MVTecAD dataset.

performance. These classes contain examples where at least one kind of anomaly has a significant proportion of anomalous pixels left undetected.

6.2 Restoration Model on Ped2

To apply our restoration method (Chapter 4) to the UCSD Ped2 dataset [65], it makes sense to apply the region extractor used in our KDE-based method to first filter out the background scene. The restoration model can then focus training on restoring the figures of people rather than large areas of sky and ground. The extracted regions are forced to be square and of a size acceptable to the architecture. Following our procedure described in Section 4.1, we force extracted regions to be 65×65 but we perform no resizing of the dataset images. Figure 6.7 shows example anomaly segmentations given by our restoration method on this dataset. We can see that the method consistently segments the wheels of the bikes and other fine anomalous details like the skateboards. This method detects skateboards far more competently than the KDE-based anomaly detector that makes decisions based on entire regions as a whole, rather than on the fine details of the pixels they contain.

We would like to produce AUC metrics to compare performance with the KDE-based method. Unfortunately, where region-based anomaly detectors gain a moder-

ate advantage over anomaly segmentation models in a literal pixel-level ground truth context, they gain an enormous advantage in a Ped2 pixel-level context, to the point where comparison is meaningless. Consider that our restoration method produces a unique anomaly score per pixel. Given a normal frame, if any one of those pixels is scored too highly then the entire frame counts against the model. Conversely, given an anomalous frame, the method needs to produce a high anomaly score for at least 40% of the anomalous pixels. Most anomalies consist of cyclists, and the ground-truth contains a solid convex region encompassing the bike, the person, and a fair number of background pixels that can be seen through the wheels, bike frame and much of the area around the cyclist. The restoration method, considering only pixels, never assigns a high score to the cyclist, only pixels belonging to some of the bike. Also, in the case of the skateboarders, only the skateboard itself is considered anomalous by the restoration model. Therefore, the method never gains credit for detecting any of the skateboard, despite doing so better than the KDE-based method; seldom gets credit for detecting any of a bike; and almost always scores a false-positive on negative frames, despite leaving the vast majority of pixels undetected. Consequently, AUC scores average around 0.003, which does not provide any useful information about the performance of the method.

The former of those problems, the under-segmentation of normal parts of anoma-

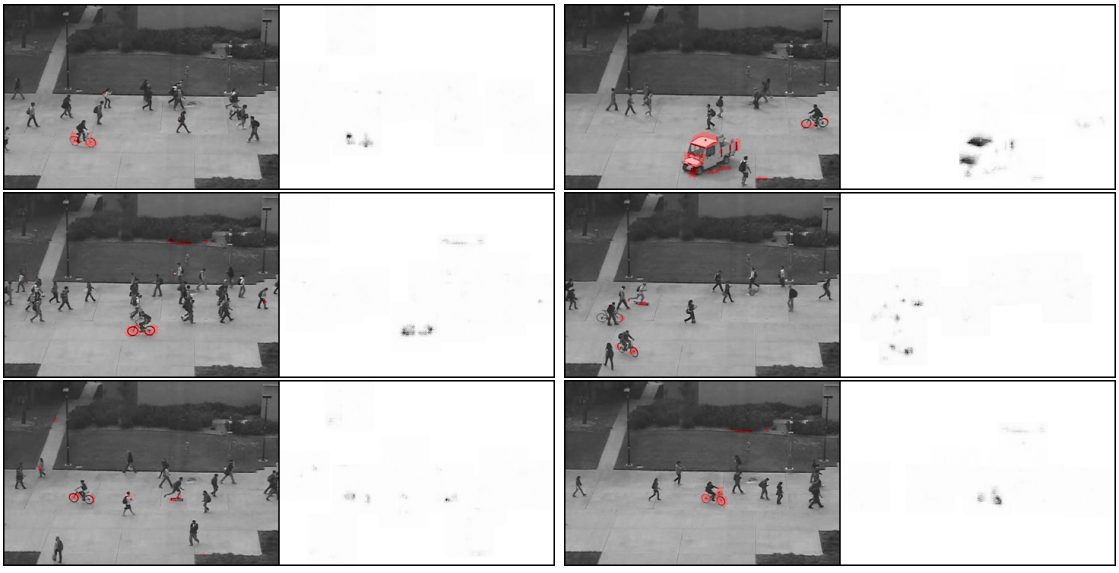

Figure 6.7: Example anomaly segmentations obtained by our restoration method on the UCSD Ped2 dataset [65].

lous entities, could be addressed in a post-processing step. For example, connected components of the segmentation mask could be grown to fill areas of the image that are similar. If the background is fixed, as in this dataset, then this could be made more robust by growing to fill areas of the image that do not belong to the background instead. The major shortcoming of this is that the segmented region will occasionally leak out of the anomalous object into the wider image. It may be better to classify anomaly detectors as fine-grained or coarse-grained and allow users to select what kind of anomaly detection is appropriate, rather than to try to fit an output to one particular definition of abnormal set by some ground truth.

6.3 Pixel Prediction on Ped2

We use the same region extraction procedure as in the previous section to apply our pixel prediction method on the UCSD Ped2 dataset. The method is applied as described in Section 5.1 using the c2i anomaly scoring method. Figure 6.8 demonstrates how this method fails to transfer to this domain. Segmented regions appear to be random and are not stable frame-to-frame. AUC metrics on this dataset are not included for the same reason described in the previous section for our restoration method.

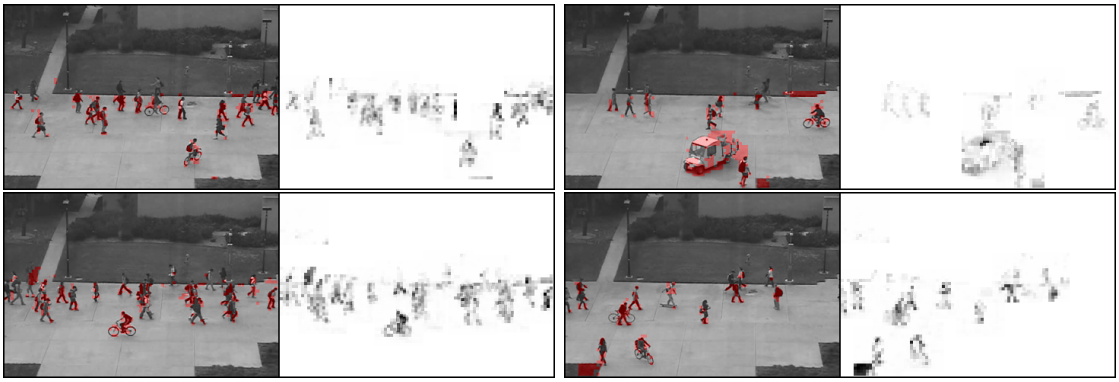

Figure 6.8: Example anomaly segmentations obtained by our pixel prediction method on the UCSD Ped2 dataset [65].

6.4 Conclusion

Our KDE-based method for abnormal activity detection in scenes is the most transferable, making a more than satisfactory attempt at textural defect detection when its region extractor is disabled in favour of simple tiling. In the context of abnormal activity detection in scenes, our restoration method is hindered by being too fine-grained and low-level since anomalies in this context lend themselves to region-level detection based on high-level concepts. Nevertheless, it competently segments fine anomalous details in the scene such as bike wheels and skateboards. This is consistent with our previous experiments that consistently show that the restoration method is a very strong pixel-level defect segmentation method. In contrast, our pixel prediction method is not strong enough to transfer to this anomaly detection context.

CHAPTER 7

Conclusion

We conclude this thesis with a discussion of our three proposed methods, detailing the contributions made and highlighting areas for future development.

7.1 Contributions

Within this thesis, we introduce three original methods for anomaly detection: one specialised towards detecting abnormal activity in scenes and two specialised towards defect segmentation in textures.

The first method is based upon Kernel Density Estimation (KDE) on features extracted from a neural network pretrained on a combination of large-scale image datasets. Some existing state-of-the-art methods similarly employ features extracted from pretrained networks to build a model of normality [40]. Relative to these, we achieve significant increasaes in computational speed and real-time performance by replacing classical region proposal systems with our modified Region Proposal Network (RPN), which significantly reduces the number of regions to be processed at the KDE stage. Furthermore, we achieve greater anomaly detection performance via three design choices. The first is to take advantage of the reduced computational

load to dramatically increase the dimensionality of the KDE model. The timings reported in Section 3.3.4 produced from our simulation of an existing technique using classical region proposal systems, suggest that increasing the dimensionality in this way is not a viable option without our RPN modification. The remaining two design choices are to take features from a lower network layer before the ReLU activation and to scale the features rather than normalising them. These alterations could be applied to existing methods without difficulty. Removing the ReLU activation allows much greater variation in the extracted features because the negative values remain. This helps to separate normal and anomalous features since they are able to differ in the negative values in addition to the positive. Taking features from a lower network layer may be beneficial because they are more general [105] i.e. they are less specialised towards indicating the output classes of the classification network, whose training examples are unlikely to be representative of the imagery observed in the anomaly detection dataset. Using the pixel-level evaluation on the Ped2 dataset [65], our modifications increase the inference-time frame-rate by at least six times when using sixteen PCA components and 25 times when using 128 PCA components, as shown by experiment in Section 3.3.4. While maintaining real-time performance, we are able to increase the number of PCA components from sixteen until anomaly detection performance is equal to that reported by Hinami et al. [40]. The remaining design choices improve performance by a further 3%.

The second method is a result of rethinking the application of autoencoders to anomaly detection. Many existing state-of-the-art methods make use of autoencoders by training them to reconstruct normal input frames [36, 104], resulting in a model that is able to form reconstructions with anomalies omitted, but is unable to form reconstructions with high-frequency detail and sufficient accuracy across the normal pixels. Consequently, the anomaly map formed via the difference between the input and reconstructed frames is often noisy, reducing performance. In contrast, our method aims to output the anomaly map directly, which is a much simpler image than the texture itself. The normal regions in the anomaly map, having a constant target anomaly score of zero, have no high-frequency content to produce. It is trained by performing texture restoration on normal samples that have been

artificially defaced using one of a wide variety of noising algorithms. We arrange for the model to output the pixel shifts required to restore the texture, rather than the restored texture itself. These shifts become the direct output anomaly map at inference time and facilitate the generation of anomaly maps with stark contrast between the normal and anomalous pixels, which significantly enhances performance. Direct generation of the simple anomaly map rather than the complex texture allows our method to achieve a new state-of-the art in textural defect segmentation on the MVTecAD dataset [11]. Our method achieves an average Area Under the ROC Curve (AUC) score of 96% across all of the texture classes compared with 93% for the next best performing model. In addition, we demonstrate the generalisability of our method to other challenging datasets with vastly differing characters [43, 102]. We show conclusively in Section 4.3.4 that the wide variety of noising algorithms is key to achieving good performance from our method, which is something that is not considered in other denoising methods [104] nor in the use of denoising autoencoders in general [10].

The final method is an experimental exploration towards performing textural defect segmentation via Transformer-based future pixel and feature prediction. While Transformer-based architectures are increasingly used in the image domain [14, 26], at the time of experimentation there were no Transformer-based architectures for anomaly detection specifically. Our final method therefore contributes towards filling that gap in the research. Through developing this method, we introduce a means of processing image data in units of brush-strokes rather than pixels to cover image generation much more quickly, the method of sampling from the transformer output distribution to select the best possible completion, the method of ensembling Transformer models that predict sequences in different directions, and four different anomaly scoring metrics. We prove the utility of applying the future prediction model over and above simple clustering alone in Section 5.3.1, and we show concretely the benefit of sampling and ensembling in Sections 5.3.2 and 5.3.3 respectively. We then extend this idea to predicting sequences of features rather than pixels, with some improvement in results as shown in Section 5.5. We demonstrate both the potential of this approach and its limitations. The potential of this method

is in its consistency in generating a range of plausible texture completions with anomalies omitted, however, it is unable to compete with the former two methods in terms of speed and performance. The barrier to speed is the large computational cost of requiring one forward propagation per predicted token, when one predicted token only covers a brush-stroke area. The barrier to performance is the wide variety of plausible texture completions given the context, especially for textural patterns with high entropy.

Overall, the strongest of the methods we introduced is the second method: the texture restoration method described in Chapter 4. This method provides very stable and consistent outputs compared to the other two methods. All figures in Chapter 4 that depict example detections were created using a fixed threshold that was not especially tuned. In contrast, the other two methods required setting a different threshold for each dataset class, and the appearance of the output was sensitive to the choice of value. This follows from the difference in character of the per-pixel heat maps of the anomaly scores, where for the second method, heat maps are characterised by a deep contrast between normal and anomalous pixels, while for the other two methods, heat maps can appear more noisy.

7.2 Learning From Cross-Context Experiments

In the methods we present, there are two main components involved in specialising a method towards either detecting abnormal activity in scenes or segmenting textural defects. The first is whether or not anomaly scores are assigned per region or per pixel. The second is whether or not high-level concepts are leveraged from pretrained deep neural networks that have been exposed to a wide variety of object categories from a large-scale dataset. On the surveillance style datasets, methods for texture defect detection are too focused on small details and do not consider objects as a whole, resulting in noisy false-positive detections and under segmented positive regions. This is a significant disadvantage. However, methods for abnormal activity detection do not show a similar disadvantage when applied to textural defect segmentation. Naturally, these systems severely over-segment anomalous

regions, but this is not critical for either real-world deployment nor quantitative evaluation. For deployment, we are usually more interested in whether there is a defect rather than the shape of the defect. For quantitative evaluation, detecting all of the anomalous pixels at the expense of falsely detecting normal peripheral pixels is a good strategy, since the number of normal pixels that are far away from defects far exceeds the number that are close to defects.

7.3 Limitations

This section highlights the limitations of the three methods presented in this thesis, while the following section suggests future work that may address some of these limitations.

The KDE-based method presented in Chapter 3 only ever uses a single source of feature representations: one whole layer of activations from the feature extraction network. There are many other features throughout the network that may contain relevant information and conversely, there may be features within the selected layer that are redundant. This method may also be hindered in general application when both the region and feature extraction networks are pretrained on a specific application; however, we show in Chapter 6 that this need not be a significant problem.

The restoration method presented in Chapter 4 uses a hard-coded noising filter bank. This is not flexible and requires many design choices that are hard to justify theoretically rather than empirically.

The pixel prediction method presented in Chapter 5 may benefit from alternative uses of the Transformer architecture. Currently, a sequence of tokens plus the placeholder token gets mapped to an identical sequence of tokens plus predicted token. It is a lot of work during training and inference to produce mainly the same tokens as we supplied at the input, when all that is required is a predicted token. The anomaly detection performance of this method is limited by the need to find the correct completion among a large quantity of plausible completions, while the computational performance is limited by the need for one forward propagation per predicted token.

7.4 Further Work

Considering the KDE-based method presented in Chapter 3, it would be interesting to explore whether the performance may be improved by preselecting a set of random subsets of features from the feature extraction network, and then employing the method using each of these subsets in an ensemble. Features may be randomly selected from within a particular layer or across layers. An ensemble could also be formed from a set of entirely separate feature extraction networks, each trained on different datasets. On top of the feature extractor, a range of different classical techniques may be applied to measure the anomaly scores such as the k-Nearest Neighbour (kNN) distance or the distance to centroids found via clustering algorithms. These may also be investigated.

For the restoration method presented in Chapter 4, it would be desirable to move away from the hard-coded noising filter bank and instead use a noising network that learns how to add noise to the training samples. This would facilitate an adversarial element to the training of the restoration network since the noising network can be continuously adapted to find noising patterns that the restoration network cannot undo. By taking snapshots of these noising patterns, a learned noising filter bank could be automatically grown over time that could potentially cover a more substantial range of noising patterns than any hard-coded filter bank. We suspect that without taking snapshots, the restoration network may simply adapt to the current noising pattern and forget how to solve previously seen patterns. Meanwhile, the noising network may aimlessly explore the space of noising patterns, maybe repeating those previously found. The difficulty we found with this approach is that the noising network learns a kind of noise that is unrealistic in terms of its high-frequency content, deposits noise liberally across the input images, and rather than exploring many different modes of noise, it simply increases the severity of the currently found noise. All attempts to address these issues did not result in better performance. These include learning the noise in Fourier space and penalising the use of higher frequencies, penalising the use of high-magnitude noise and those pixels taken out of dynamic range by the noise, aiming for a chosen target error in the restoration network rather than aiming to maximise its error,

and producing an output batch of noising patterns and penalising their structural similarity. Nevertheless, a solution to this problem may be obtainable through other means and significantly enhance the method.

Using the BERT objective [24] to train the Transformer architecture of the pixel prediction method may be more suitable since more than one token could be predicted at a time and both forward and backward context could be considered. The former of those two points will help the computational performance, while the latter of those two points may help the problem associated with finding the correct completion.

Bibliography

[1] A. Adam, E. Rivlin, I. Shimshoni, and D. Reinitz. Robust real-time unusual event detection using multiple fixed-location monitors. *Transactions on Pattern Analysis and Machine Intelligence*, 30(3):555–560, 2008.

[2] P. Adey, S. Akçay, M. Bordewich, and T. Breckon. Autoencoders without reconstruction for textural anomaly detection. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2021.

[3] P. Adey, O. Hamilton, M. Bordewich, and T. Breckon. Region based anomaly detection with real-time training and analysis. In *International Conference On Machine Learning And Applications*, pages 495–499. IEEE, 2019.

[4] M. Aharon, M. Elad, and A. Bruckstein. K-svd: an algorithm for designing overcomplete dictionaries for sparse representation. *Transactions on Signal Processing*, 54(11):4311–4322, 2006.

[5] S. Akcay, A. Atapour-Abarghouei, and T. P. Breckon. Ganomaly: semi-supervised anomaly detection via adversarial training. In *Asian Conference on Computer Vision*, pages 622–637. Springer, 2018.

[6] S. Akçay, A. Atapour-Abarghouei, and T. P. Breckon. Skip-ganomaly: skip connected and adversarially trained encoder-decoder anomaly detection. In *International Joint Conference on Neural Networks*, pages 1–8. IEEE, 2019.

[7] J. T. Andrews, N. Jaccard, T. W. Rogers, and L. D. Griffin. Representation-learning for anomaly detection in complex x-ray cargo imagery. In *Anomaly Detection and Imaging with X-Rays (ADIX) II*, volume 10187, 101870E. International Society for Optics and Photonics, 2017.

[8] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein generative adversarial networks. In *International Conference on Machine Learning*, pages 214–223. Omnipress, 2017.

[9] C. Baur, B. Wiestler, S. Albarqouni, and N. Navab. Deep autoencoding models for unsupervised anomaly segmentation in brain mr images. In *International MICCAI Brainlesion Workshop*, pages 161–169. Springer, 2018.

[10] Y. Bengio, L. Yao, G. Alain, and P. Vincent. Generalized denoising auto-encoders as generative models. In *Advances in Neural Information Processing Systems*, pages 899–907, 2013.

[11] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger. Mvtec ad–a comprehensive real-world dataset for unsupervised anomaly detection. In *Conference on Computer Vision and Pattern Recognition*, pages 9592–9600, 2019.

[12] P. Bergmann, S. Löwe, M. Fauser, D. Sattlegger, and C. Steger. Improving unsupervised defect segmentation by applying structural similarity to autoencoders. *International Conference on Computer Vision Theory and Applications*:372–380, 2019.

[13] T. Böttger and M. Ulrich. Real-time texture error detection on textured surfaces with compressed sensing. *Pattern Recognition and Image Analysis*, 26(1):88–94, 2016.

[14] M. Chen, A. Radford, R. Child, J. Wu, H. Jun, D. Luan, and I. Sutskever. Generative pretraining from pixels. In *International Conference on Machine Learning*, pages 1691–1703. PMLR, 2020.

[15] Y. Chen, Y. Tian, G. Pang, and G. Carneiro. Unsupervised anomaly detection and localisation with multi-scale interpolated gaussian descriptors. *arXiv preprint arXiv:2101.10043*, 2021.

[16] H. Cheng, Z. Liu, L. Yang, and X. Chen. Sparse representation and learning in visual recognition: theory and applications. *Signal Processing*, 93(6):1408–1425, 2013.

[17] K.-W. Cheng, Y.-T. Chen, and W.-H. Fang. Video anomaly detection and localization using hierarchical feature representation and gaussian process regression. In *Computer Vision and Pattern Recognition*, pages 2909–2917. IEEE, 2015.

[18] Y. S. Chong and Y. H. Tay. Abnormal event detection in videos using spatiotemporal autoencoder. In *International Symposium on Neural Networks*, pages 189–196. Springer, 2017.

[19] Y. Cong, J. Yuan, and J. Liu. Abnormal event detection in crowded scenes using sparse representation. *Pattern Recognition*, 46(7):1851–1864, 2013.

[20] Y. Cong, J. Yuan, and J. Liu. Sparse reconstruction cost for abnormal event detection. In *Computer Vision and Pattern Recognition*, pages 3449–3456. IEEE, 2011.

[21] D. Dehaene, O. Frigo, S. Combrexelle, and P. Eline. Iterative energy-based projection on a normal data manifold for anomaly localization. *International Conference on Learning Representations*, 2020.

[22] A. Del Giorno, J. A. Bagnell, and M. Hebert. A discriminative framework for anomaly detection in large videos. In *European Conference on Computer Vision*, pages 334–349. Springer, 2016.

[23] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: a large-scale hierarchical image database. In *Computer Vision and Pattern Recognition*, pages 248–255. IEEE, 2009.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics, June 2019.

[25] J. Donahue, P. Krähenbühl, and T. Darrell. Adversarial feature learning. *International Conference on Learning Representations*, 2017.

[26] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2021.

[27] A. Dosovitskiy, P. Fischer, E. Ilg, P. Hausser, C. Hazirbas, V. Golkov, P. van der Smagt, D. Cremers, and T. Brox. Flownet: learning optical flow with convolutional networks. In *International Conference on Computer Vision*, pages 2758–2766. IEEE, 2015.

[28] K. Fragkiadaki, P. Arbelaez, P. Felsen, and J. Malik. Learning to segment moving objects in videos. In *Computer Vision and Pattern Recognition*, pages 4083–4090. IEEE, 2015.

[29] R. Girshick. Fast r-cnn. In *International Conference on Computer Vision*, pages 1440–1448. IEEE, 2015.

[30] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, pages 580–587. IEEE, 2014.

[31] I. Golan and R. El-Yaniv. Deep anomaly detection using geometric transformations. *Advances in neural information processing systems*, 31, 2018.

[32] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[33] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems*, pages 2672–2680, 2014.

[34] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[35] L. D. Griffin, M. Caldwell, J. T. Andrews, and H. Bohler. Unexpected item in the bagging area: anomaly detection in x-ray security images. *Transactions on Information Forensics and Security*, 14(6):1539–1553, 2018.

[36] M. Hasan, J. Choi, J. Neumann, A. K. Roy-Chowdhury, and L. S. Davis. Learning temporal regularity in video sequences. In *Computer Vision and Pattern Recognition*, pages 733–742. IEEE, 2016.

[37] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. In *International Conference on Computer Vision*, pages 2980–2988. IEEE, 2017.

[38] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Computer Vision and Pattern Recognition*, pages 770–778. IEEE, 2016.

[39] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: surpassing human-level performance on imagenet classification. In *International Conference on Computer Vision*, pages 1026–1034. IEEE, 2015.

[40] R. Hinami, T. Mei, and S. Satoh. Joint detection and recounting of abnormal events by learning deep generic knowledge. In *International Conference on Computer Vision*. IEEE, Oct. 2017.

[41] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[42] X. Hu, S. Hu, Y. Huang, H. Zhang, and H. Wu. Video anomaly detection using deep incremental slow feature analysis network. *IET Computer Vision*, 10(4):258–265, 2016.

[43] D. Hughes, M. Salathé, et al. An open access repository of images on plant health to enable the development of mobile disease diagnostics. *arXiv preprint arXiv:1511.08060*, 2015.

[44] R. T. Ionescu, S. Smeureanu, B. Alexe, and M. Popescu. Unmasking the abnormal events in video. *International Conference on Computer Vision*, 2017.

[45] R. T. Ionescu, S. Smeureanu, M. Popescu, and B. Alexe. Detecting abnormal events in video using narrowed normality clusters. In *Winter Conference on Applications of Computer Vision*, pages 1951–1960. IEEE, 2019.

[46] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Computer Vision and Pattern Recognition*, pages 1125–1134, 2017.

[47] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[48] J. Johnson, A. Alahi, and L. Fei-Fei. Perceptual losses for real-time style transfer and super-resolution. In *European Conference on Computer Vision*, pages 694–711. Springer, 2016.

[49] D. P. Kingma and J. Ba. Adam: a method for stochastic optimization. *International Conference on Learning Representations*, 2015.

[50] M. Koppel, J. Schler, and E. Bonchek-Dokow. Measuring differentiability: unmasking pseudonymous authors. *Journal of Machine Learning Research*, 8(Jun):1261–1276, 2007.

[51] P. Krähenbühl and V. Koltun. Geodesic object proposals. In *European Conference on Computer Vision*, pages 725–739. Springer, 2014.

[52] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, et al. Visual genome: connecting language and vision using crowdsourced dense image annotations. *International Journal of Computer Vision*, 123(1):32–73, 2017.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[54] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[55] Y. LeCun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. Handwritten digit recognition with a back-propagation network. *Advances in Neural Information Processing Systems*, 2, 1989.

[56] Z. Li, N. Li, K. Jiang, Z. Ma, X. Wei, X. Hong, and Y. Gong. Superpixel masking and inpainting for self-supervised anomaly detection. In *British Machine Vision Conference*, pages 7–10. IEEE, 2020.

[57] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft coco: common objects in context. In *European Conference on Computer Vision*, pages 740–755. Springer, 2014.

[58] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer. Generating wikipedia by summarizing long sequences. *International Conference on Learning Representations*, 2018.

[59] W. Liu, W. Luo, D. Lian, and S. Gao. Future frame prediction for anomaly detection–a new baseline. In *Computer Vision and Pattern Recognition*, pages 6536–6545. IEEE, 2018.

[60] W. Liu, R. Li, M. Zheng, S. Karanam, Z. Wu, B. Bhanu, R. J. Radke, and O. Camps. Towards visually explaining variational autoencoders. In *Computer Vision and Pattern Recognition*, pages 8642–8651. IEEE, 2020.

[61] P. Liznerski, L. Ruff, R. A. Vandermeulen, B. J. Franks, M. Kloft, and K.-R. Müller. Explainable deep one-class classification. *International Conference on Learning Representations*, 2021.

[62] C. Lu, J. Shi, and J. Jia. Abnormal event detection at 150 fps in matlab. In *International Conference on Computer Vision*, pages 2720–2727. IEEE, 2013.

[63] W. Luo, Z. Gu, J. Liu, and S. Gao. Encoding structure-texture relation with p-net for anomaly detection in retinal images, 2020.

[64] W. Luo, W. Liu, and S. Gao. A revisit of sparse coding based anomaly detection in stacked rnn framework. *International Conference on Computer Vision*, 2017.

[65] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos. Anomaly detection in crowded scenes. In *Computer Vision and Pattern Recognition*, pages 1975–1981. IEEE, 2010.

[66] J. Masci, U. Meier, D. Cireşan, and J. Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *International Conference on Artificial Neural Networks*, pages 52–59. Springer, 2011.

[67] F. Massa and R. Girshick. Mask R-CNN-Benchmark: fast, modular reference implementation of instance segmentation and object detection algorithms in PyTorch. `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018. Accessed: 01/08/19.

[68] M. Mathieu, C. Couprie, and Y. LeCun. Deep multi-scale video prediction beyond mean square error. In *International Conference on Learning Representations*, 2016.

[69] J. R. Medel and A. Savakis. Anomaly detection in video using predictive convolutional long short-term memory networks. *arXiv preprint arXiv:1612.00390*, 2016.

[70] R. Mehran, A. Oyama, and M. Shah. Abnormal crowd behavior detection using social force model. In *Computer Vision and Pattern Recognition*, pages 935–942. IEEE, 2009.

[71] P. Napoletano, F. Piccoli, and R. Schettini. Anomaly detection in nanofibrous materials by CNN-based self-similarity. *Sensors*, 18(1):209, 2018.

[72] A. Y. Ng, M. I. Jordan, and Y. Weiss. On spectral clustering: analysis and an algorithm. In *Advances in Neural Information Processing Systems*, pages 849–856, 2002.

[73] G. Park, M. Lee, H. Jang, and C. Kim. Thermal anomaly detection in walls via cnn-based segmentation. *Automation in Construction*, 125:103627, 2021.

[74] N. Parmar, A. Vaswani, J. Uszkoreit, L. Kaiser, N. Shazeer, A. Ku, and D. Tran. Image transformer. In *International Conference on Machine Learning*, pages 4055–4064. PMLR, 2018.

[75] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318. Omnipress, 2013.

[76] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *International Conference on Learning Representations*, 2016.

[77] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[78] H. Ren, W. Liu, S. I. Olsen, S. Escalera, and T. B. Moeslund. Unsupervised behavior-specific dictionary learning for abnormal event detection. In *British Machine Vision Conference*, pages 28–1, 2015.

[79] H. Ren, H. Pan, S. I. Olsen, and T. B. Moeslund. A comprehensive study of sparse codes on abnormality detection. *arXiv preprint arXiv:1603.04026*, 2016.

[80] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems*, pages 91–99, 2015.

[81] M. Ribeiro, A. E. Lazzaretti, and H. S. Lopes. A study of deep convolutional auto-encoders for anomaly detection in videos. *Pattern Recognition Letters*, 2017.

[82] O. Ronneberger, P. Fischer, and T. Brox. U-net: convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[83] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

[84] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. In *International Conference on Neural Information Processing Systems*, pages 2234–2242, 2016.

[85] T. Schlegl, P. Seeböck, S. M. Waldstein, G. Langs, and U. Schmidt-Erfurth. F-anogan: fast unsupervised anomaly detection with generative adversarial networks. *Medical Image Analysis*, 54:30–44, 2019.

[86] T. Schlegl, P. Seeböck, S. M. Waldstein, U. Schmidt-Erfurth, and G. Langs. Unsupervised anomaly detection with generative adversarial networks to guide marker discovery. In *International Conference on Information Processing in Medical Imaging*, pages 146–157. Springer, 2017.

[87] D. W. Scott. *Multivariate density estimation: theory, practice, and visualization*. John Wiley & Sons, 2015.

[88] B. W. Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[89] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *International Conference on Learning Representations*, 2015.

[90] C. Solomon and T. Breckon. *Fundamentals of Digital Image Processing: A practical approach with examples in Matlab*. John Wiley & Sons, 2011.

[91] H. Song, Z. Jiang, A. Men, and B. Yang. A hybrid semi-supervised anomaly detection model for high-dimensional data. *Computational Intelligence and Neuroscience*, 2017, 2017.

[92] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International Conference on Machine Learning*, pages 843–852. Omnipress, 2015.

[93] Q. Sun, H. Liu, and T. Harada. Online growing neural gas for anomaly detection in changing surveillance scenes. *Pattern Recognition*, 64:187–201, 2017.

[94] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems*, pages 3104–3112, 2014.

[95] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Conference on Artificial Intelligence*, 2017.

[96] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition*, pages 1–9. IEEE, 2015.

[97] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *International Conference on Computer Vision*, pages 4489–4497. IEEE, 2015.

[98] H. T. Tran and D. Hogg. Anomaly detection using a convolutional winner-take-all autoencoder. In *Proceedings of the British Machine Vision Conference*. British Machine Vision Association, 2017.

[99] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 104(2):154–171, 2013.

[100] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in Neural Information Processing Systems*:6000–6010, 2017.

[101] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Transactions on Image Processing*, 13(4):600–612, 2004.

[102] M. Wieler and T. Hahn. Weakly supervised learning for industrial optical inspection. In *DAGM Symposium*, 2007.

[103] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: a machine learning approach for precipitation nowcasting. In *Advances in Neural Information Processing Systems*, pages 802–810, 2015.

[104] D. Xu, Y. Yan, E. Ricci, and N. Sebe. Detecting anomalous events in videos by learning deep representations of appearance and motion. *Computer Vision and Image Understanding*, 156:117–127, 2017.

[105] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *International Conference on Neural Information Processing Systems-Volume 2*, pages 3320–3328, 2014.

[106] H. Zenati, C. S. Foo, B. Lecouat, G. Manek, and V. R. Chandrasekhar. Efficient gan-based anomaly detection. *arXiv preprint arXiv:1802.06222*, 2018.

[107] H. Zenati, M. Romain, C.-S. Foo, B. Lecouat, and V. Chandrasekhar. Adversarially learned anomaly detection. In *International conference on data mining*, pages 727–736. IEEE, 2018.

[108] Y. Zeng, J. Fu, H. Chao, and B. Guo. Learning pyramid-context encoder network for high-quality image inpainting. In *Conference on Computer Vision and Pattern Recognition*, pages 1486–1494. IEEE, 2019.

[109] Y. Zhang, H. Lu, L. Zhang, X. Ruan, and S. Sakai. Video anomaly detection based on locality sensitive hashing filters. *Pattern Recognition*, 59:302–311, 2016.

[110] Y. Zhao, B. Deng, C. Shen, Y. Liu, H. Lu, and X.-S. Hua. Spatio-temporal autoencoder for video anomaly detection. In *Multimedia Conference*, pages 1933–1941. ACM, 2017.

Appendices

APPENDIX A

NOUS Integration For Industrial Applications

Our KDE-based method from Chapter 3 was integrated into the NOUS system developed by COSMONiO, which was was later acquired by Intel. We share some test material that was produced during the method development in the lead-up to that integration. This material was not included in the main body of the thesis due to its limited academic significance. It does however, demonstrate the method working in a wider range contexts.

Figure A.1 shows example detections on a dataset we recorded to simulate a real-world anomaly detection scenario of interest to a potential client of COSMONiO. We aimed to show the method to be capable of detecting foreign objects on a conveyor belt. In the simulated case, the purpose of the conveyor belt was to transport potatoes. Anything other than a potato was considered to be foreign. Following the success of this demonstration, COSMONiO produced a promotional video using the model output we supplied.

After COSMONiO obtained the code to the method, they purchased a model conveyor belt and ran a wide range of their own tests. They performed live demonstrations of the method on their new conveyor belt during the Hannover Messe trade fair, one of the leading global platforms for industrial innovation. Figure A.2 shows

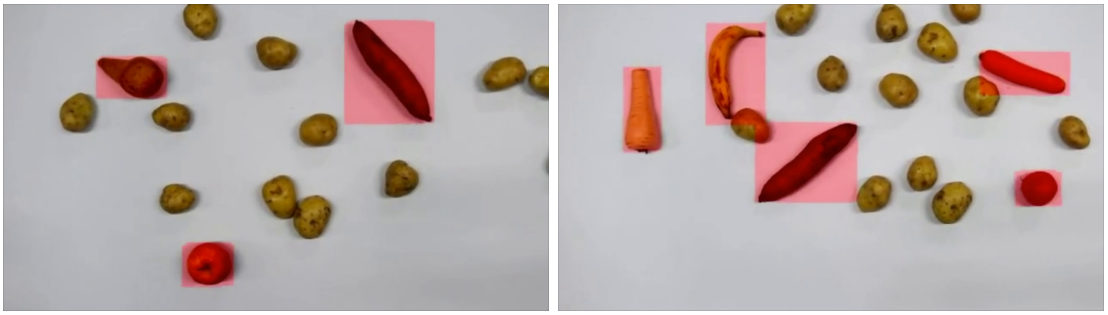

Figure A.1: Examples of anomaly detections on our Potato dataset.

some example frames taken from one of the videos they produced on their conveyor belt.

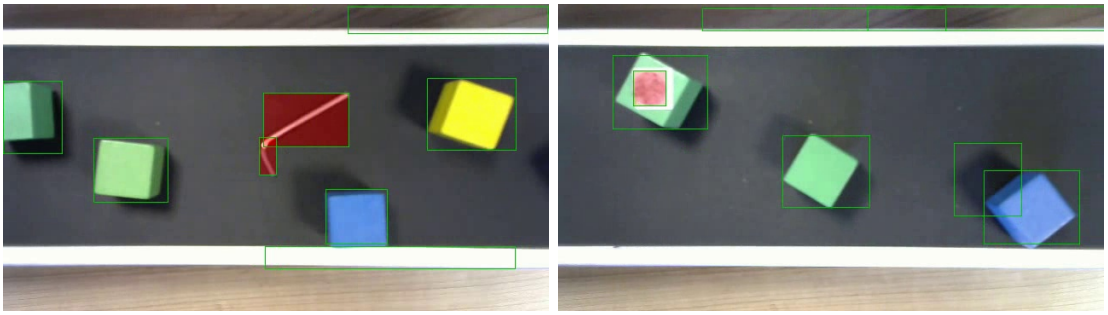

Figure A.2: Examples of anomaly detections performed by COSMONiO on their own experimental conveyor belt setup.

The intended domain of the KDE-based method was live footage of scenes containing objects, where anomalies of interest are on the scale of objects. COSMONiO became interested in learning whether the method could be adapted for slightly different domains, such as detecting small defects within objects. We were able to demonstrate that by altering some parameters within the region extractor, once could significantly transform the nature of the regions produced to suit other domains. Figure A.3 demonstrates this capability on a video taken of a business card with small defects drawn using a pen.

The method has now been integrated into NOUS and in use for approximately two years.

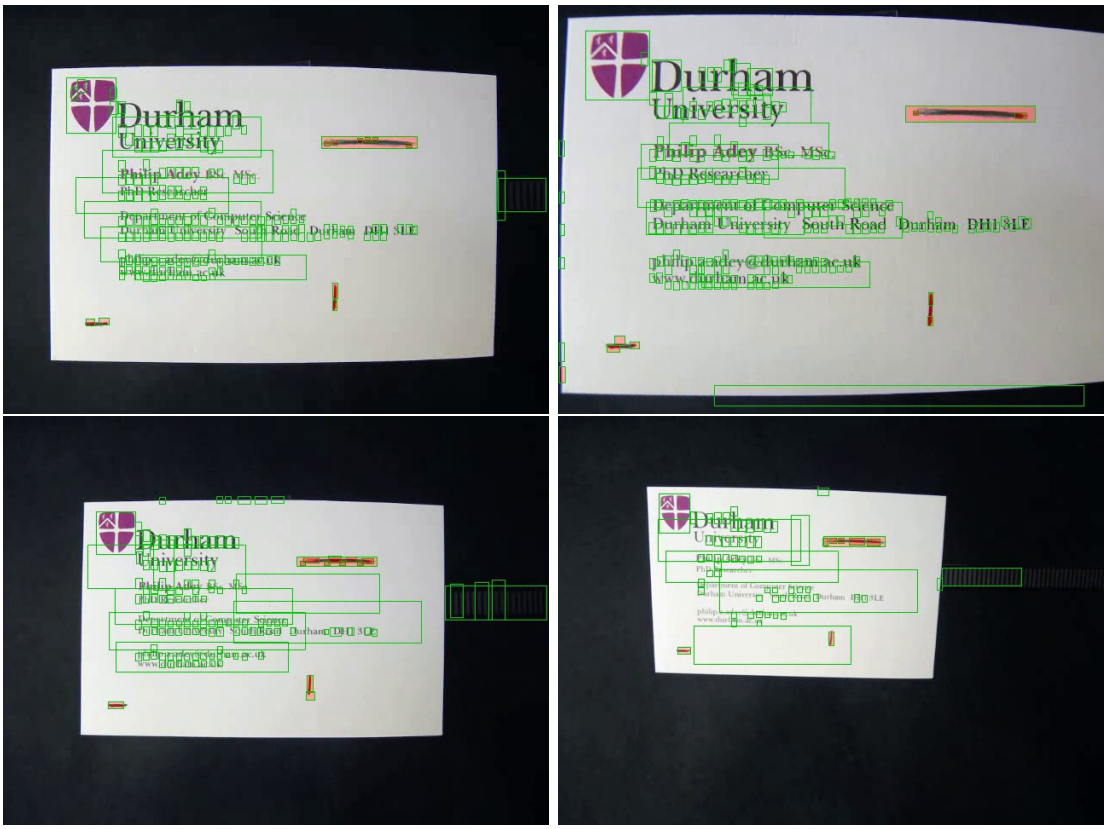

Figure A.3: A demonstration of the effect of altering parameters within the region extractor. Regions are modified to suit the different domain