Industrial Video Anomaly Detection Using a Weakly Supervised Predictive Autoencoder

Master's thesis in Electical Engineering

Petter Dittmer

Department of Electrical Engineering

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2025
www.chalmers.se

Master's thesis 2025

Industrial Video Anomaly Detection Using a Weakly Supervised Predictive Autoencoder

PETTER DITTMER

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
Gothenburg, Sweden 2025

Industrial Video Anomaly Detection Using a Weakly Supervised Predictive Autoencoder
PETTER DITTMER

Supervisors: Victor Pettersson, Department of Electrical Engineering, and Björn Berntsson, EyeAtProduction
Examiner: Giuseppe Durisi, Department of Electrical Engineering

Master's Thesis 2025
Department of Electrical Engineering
Division of Communications, Antennas, and Optical Networks
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Heat map of the prediction error of a fallen can moving along a conveyor.

Typeset in LaTeX
Gothenburg, Sweden 2025

Abstract

In this thesis, a predictive autoencoder model and video data pipeline are formulated for detecting anomalies in production flow in industrial environments. Common modifications to deep neural networks, such as spatial attention blocks, dropouts and skip connections, are investigated to assess whether they affect the overall anomaly-detection performance for the intended industrial scenario. The project was done in cooperation with the company EyeAtProduction AB in Borås, Sweden.
The model is designed for flexibility, robustness and short training times in new environments, rather than state of the art performance. It uses a pretrained version of the image recognition network ResNet-18 for encoding sequences of four video frames. The encoded frames are merged with a 1×1 convolution operation, and then decoded via transposed convolutions, resulting in a prediction of the next frame following the sequence. By only training the network on footage of normal production, it will become proficient at predicting normal movement and spatial features, but struggle to reconstruct anomalous sequences and objects. Anomalies can therefore be detected based on the degree of error between the prediction and the true next frame.
The models show promise in both controlled environments and real-world cases, but even with heavy data augmentation they are still sensitive to lighting changes and vibrations in the camera, making them prone to false positives. More research would need to be done to minimize this problem further, but possible solutions could be collecting larger and more diverse training sets, and making the threshold adapt to the long term shifts in the prediction scores during inference.

Keywords:*Anomaly detection, autoencoder, convolutional network, ResNet, production processes.*

Acknowledgements

I would like to thank Victor Petterson for his supervision and guidance, and Giuseppe Durisi for taking the role of examiner. Thanks also to Björn Berntsson, Anders Tingström and Olivia Jonson at EyeAtProduction for their support, and to Philip Vinensjö for providing the factory footage. Finally, I appreciate the UCSD Visual Surveillance Lab for constructing and publishing the Ped2 dataset.

Petter Dittmer, Gothenburg, June 2025

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

CNN	Convolutional Neural Network
CPU	Central Processing Unit
FPS	Frames Per Second
FPR	False Positive Rate
GB	Gigabyte
GPU	Graphics Processing Unit
MSE	Mean Squared Error
ML	Machine Learning
PSNR	Peak Signal-to-Noise Ratio
RGB	Red–Green–Blue (color channels)
ReLU	Rectified Linear Unit
ROI	Region of Interest
SSIM	Structural Similarity Index
TPR	True Positive Rate
VAD	Video Anomaly Detection

Nomenclature

Below is the nomenclature of indices, sets, parameters, and variables that have been used throughout this thesis.

Indices

i	Pixel (or element) index within an image or feature map.
t	Frame (time) index within a sequence.
m, n	Kernel indices (height and width coordinates of convolutional filters).

Parameters & Hyperparameters

B	Batch size (number of sequences per gradient-update).
T	Sequence length (number of input frames).
C	Number of channels per frame (e.g, 3 for RGB).
H, W	Height and width of each frame.
M, N	Spatial size of convolutional kernels.
p	Zero–padding size in convolutional layers.
s	Stride length in convolutional layers.
α, β, γ	Weights for combining MSE, gradient and SSIM losses in total loss.
N	Number of training sequences sampled per epoch.

Variables & Intermediate Quantities

x	Input frame or feature map (possibly flattened).
$\hat{x}$	Reconstructed or predicted frame/feature map.
I_i, $\hat{I}_i$	Pixel intensity of the ground truth and predicted frames, respectively.
z	Pre-activation (weighted sum) in a layer, e.g. $z = Wx + b$.

a	Post-activation output, $a = f(z)$.
W, b	Learnable weights and biases of a layer.
$\mu_{\mathcal{B}}$, $\sigma^2_{\mathcal{B}}$	Batch mean and variance (for batch-norm).
y	True label or target value (e.g. next-frame in prediction).

Contents

List of Figures

List of Tables

1
Introduction

Since the 1990s, anomaly detection (AD) machine learning (ML) models have been a rapidly developing field, and significant progress has been made year after year in making the algorithms more accurate, more efficient, and more generally applicable. In simple terms, anomaly detection is the process of detecting unusual patterns in streams of data. Some examples of common applications are for cybersecurity, fraud protection, trading algorithms, predicting illnesses in hospital patients, and more. In recent years, with the constant development and increased availability of parallel processing power from hardware such as GPUs, real-time anomaly detection in video streams has emerged as one of the larger application areas of AD. Common applications of this technology include autonomous vehicles, crowd monitoring, and industrial monitoring [3] [4]. These latter two applications significantly reduce the need for manual video review by automatically flagging unusual events, allowing for much quicker response times and more efficient analysis of large-scale video data.
In many of today's manufacturing and production facilities, some error-prone processes are still manually monitored despite the recent trend towards industrial automation. This is what the partner company EyeAtProduction AB is currently striving to solve. Their main product and strategy is a system that detects motion in production with a laser detector connected to a phone camera that is continuously recording. Whenever a specific trigger occurs, such as "no movement detected within the last 10 seconds", the software marks the video recorded for a predetermined time before and after as anomalous, making it easy to find the anomalies detected and determining the underlying problem [5]. In this report, the possibility of solving this task using an ML-based video anomaly detection (VAD) model will be explored.
The definition of normal and anomalous is crucial in anomaly detection, but can also often be very challenging using traditional statistical methods. In order to define normal, one needs to be able to mathematically describe what normal data looks and acts like, which is made even more difficult due to real-world data being affected by noise, possible errors in measurement instruments, other artifacts and shifts etc. This is the reason why ML anomaly detection models have a possible advantage over statistical approaches. In these models, normal and anomalous behavior can be learned with weak or even no supervision or explicit definitions beforehand.
The intention in this report is for the constructed model to be flexible enough to be applicable in any setting involving clearly visible anomalous events and a clearly defined normal process, and for it to be quick to train in new scenarios. This high degree of generality means that the model would have a myriad of possible applications. However, this is often a double-edged sword, since this increased generality often results in lower performance in the specific case. The fact that "normal" and

"anomalous" can take many different forms depending on the dataset means that the anomaly detection strategy used is usually constructed or tweaked for that specific type of data or dataset. With a model designed to be applicable in most situations and settings out-of-the-box, it is unavoidable that some precision and optimization will be lost. To circumvent this, the model should ideally have some clearly defined input parameters to be entered by the user, such as the desired sensitivity, the time scale of anomalous events, or the expected degree of lighting shift and background noise. This could give the model some of the benefits of customization, while keeping it interpretable and approachable for the general user.

1.1 Aim

The purpose of this project is to explore how VAD with ML can be applied to simpler industrial production processes and sub-processes, such as conveyors, feeders and pickers. The project will be carried out in collaboration with EyeAtProduction AB in Borås.
The goal is to develop a method for collecting data, a data pipeline, and a ML model that is straightforward to set up and use, without requiring expensive hardware or extensive expertise. The model should also be quick to train and deploy in new situations. The model is intended to be deployed on a Google pixel 9 smartphone, but this is not included in the report due to time constraints. The main focus will be to develop a flexible and efficient model and data pipeline, which will be trained and evaluated on a personal computer or using existing cloud services. A few different commonly used modifications to the model (skip connections, dropout, attention) will also be evaluated to see their individual effects on the behaviour.

1.2 Scope

This work focuses on developing a weakly supervised predictive autoencoder for video anomaly detection in industrial settings, prioritizing flexibility and fast training over state-of-the-art absolute performance. The model is explicitly limited to use single-camera feeds, return sequence-level prediction errors (PSNR) as the anomaly signal, and assumes the existence of a clearly defined and clearly visible “normal” process in the application cases. It does not explore event-level grouping or classification beyond the binary "anomalous or not". Sophisticated thresholding functions and algorithms were also not explored due to time constraints. The intention is to create a model that can be retrained in minutes on modest hardware with minimal parameter tuning, at the cost of some precision compared to more specialized or compute-intensive approaches.

1.3 Outline

The remainder of this report is organized as follows: Chapter 2 contains the main theory behind the machine learning strategies used in the construction of the model

and the surrounding functions and algorithms, such as encoder-decoder models, loss functions, and convolutional neural networks. It also gives a general introduction to video data handling and anomaly detection as a whole. Chapter 3 details the method used in generating the results, including data collection and preprocessing, the design of the predictive autoencoder (including spatial attention, dropout, and skip-connection variants), and the training and evaluation approaches. Chapter 4 presents the experimental results from deploying the models on three different datasets and the UCSD Ped2 benchmark dataset. This includes ROC graph comparisons, both between models and between data augmentation approaches, and more specific case-by-case anomaly score graphs to show how the models behave. Finally, Chapter 5 summarizes and discusses the findings and the possible causes, evaluates the limitations of each model, and provides suggestions for future work. The report ends with a short conclusion in Chapter 6.

2
Theory

This chapter will highlight and explain the theory behind the main machine learning components and statistical strategies used in the suggested model and data pipeline, as well as give an introduction to the field of ML anomaly detection as a whole.

2.1 Anomaly detection

At its core, anomaly detection refers to the identification of events or observations that do not conform to the expected and normal behavior of a dataset. Hence, the main work of constructing an anomaly detection method is to define what normality and abnormality are for the data in question. Some often used statistical methods are Z-score classification, chi-square distribution and studentized residuals [6].

2.1.1 Machine learning approaches

In machine learning approaches, normality and abnormality are learned directly from a set of initial data, known as the training set. The training set is fed into the model, which adjusts its parameters continuously to minimize a predefined loss function that quantifies how well it fits the training data. After training, the model is deployed on unseen data, and any events that deviate from the tendencies of the training data are flagged. There are three main approaches when performing this type of learning. They are:

Supervised learning

In supervised learning, the training data points are fully labeled as either normal or anomalous, which the model uses as ground truth. Since this is the only approach of the three where the model will have access to labeled anomalies during training, it's the only one that allows classification of anomalies instead of only outputting "normal" or "abnormal. The main disadvantage of supervised learning is that it requires knowledge of, and data from, every anomaly that might occur in order for the model to be effective. Also, the process of labeling can be arduous as machine learning models often require quite large training sets [7].

Semi-supervised learning

In semi-supervised learning, the model is instead fed only normal data for training. This circumvents the need of explicitly labeling the data points, and of including all possible anomalies in the training set. This makes the model more general in regards to which anomalies can be detected, but often results in lower

precision. It can also be challenging to collect data that covers every possible state of normality [8].

Unsupervised learning
In unsupervised learning, there are no prerequisites on the training data at all. The model learns normality simply based on the assumption that normal events are more common than anomalous events, so by training on general footage the model will become skewed towards recognizing normal states. This method is widely applicable but less precise and stable than the others, and also relies heavily on the assumption that normal events are more frequent than anomalous events [8].

2.1.2 Video data

The data that will be used in this report is video footage. Deploying machine learning models on images and video data is an area of great potential, but it also brings a few important challenges. Since video and image data are inherently human-readable, it is one of the less abstract applications of machine learning, and one can often explicitly examine the steps of how the model interprets and processes the image. Another upside is the vast amount of video and image data already available, which means that there is often no need to collect new data for a specific ML video model [9]. As for challenges, two of the main ones are data size and interpretability of the data.

Video frames contain a very large amount of information. As an example, an uncompressed RGB-encoded video taken in 720p at 25 frames per second contains $720 \times 1280 \times 3 \times 25 = 69.12 \times 10^6$ bytes of information per second, or approximately 250 GB per hour. The data can be very effectively compressed for the purpose of simple viewing, which is why an hour of video is usually no more than a few GB, but in video ML one often processes the video frame by frame. Because of this, the performance of video models are often constrained simply by the amount of computational time and power available for training and inference. Significant consideration of data-reducing methods such as compression, pruning of data and downsampling is required to decrease the amount calculations required [10]. The challenge of large datasets is closely related to interpretability in video handling. In this case, interpretability refers to the model's ability to understand what matters in the video. In the general case, a video will contain some degree of both important process information and unimportant background information, and it can be challenging to make the model focus on what's actually important for the task at hand. There are many methods for circumventing this problem, each with different pros and cons. For example, a common strategy is to define a region of interest (ROI) in the video, and only deploy the model on the pixel values within that region. Another strategy is to deploy the model on the difference between neighboring frames instead of on the frames themselves, thus lowering the importance of still areas. This can be very effective when studying moving events with static backgrounds [11].

2.1.3 Types of anomalies

When constructing an anomaly detection model and pipeline, it is important to consider the nature of the anomalies that are to be detected. In the general case, the main metric that requires consideration is the time scale of the anomalies. A model specifically designed for flagging sudden changes or spikes (small temporal context) might not be able to detect slow anomalous drifts (larger temporal context), and vice versa. If the model takes sequential data, then the time window needs to be large enough to include the context required to decide if an event is anomalous or not, since an isolated event might only be anomalous under specific circumstances. In these sequential models, the "time window" simply means how much past information you feed the network at once, such as the span of video frames or time steps that make up each input sequence. To clarify, not every architecture is bound by this explicit temporal window. Recurrent networks, among others, can carry a hidden state or external memory that lets them learn from far back in time without requiring the input to contain all the necessary context at every step [12].
In the case of video AD, one often wants to detect both instant (spatial) and temporal anomalies. Here, spatial anomalies refer to abnormal events that can be recognized in a single frame, with no other context. Examples of spatial anomalies could be a car on the sidewalk, a mislabeled can on a conveyor belt, or a fire in a frame of surveillance footage. Temporal anomalies, on the other hand, require temporal context. Examples of these could be a car driving too fast, a robot performing tasks in the wrong order, or a stopped conveyor belt. If we consider the event of a robot arm dropping an object, there will be both anomalous movement (temporal) from the object accelerating towards the ground, and spatial deviation from the object being somewhere it isn't supposed to.
As mentioned before, detecting longer temporal anomalies generally requires longer time windows. Because of the large size of frame data, inputting more frames will require a higher grade of frame compression in order to maintain the same complexity. This is where the compromise between spatial and temporal context comes from. On the one hand, finer spatial details might be lost if the grade of compression is too high, but on the other, longer temporal details may be lost if the time window is too short.

2.2 Encoder-decoder models

An encoder-decoder architecture is a class of models used for various sequence-to-sequence tasks, where the goal is to map an input sequence to an output sequence. The architecture generally consists of two main parts: the encoder and the decoder. The encoder processes the input and compresses it into a lower-dimensional latent representation, and the decoder then takes this representation and generates the output. Models that attempt to reconstruct the representation into the original input are a subset of encoder-decoder methods called autoencoders [9].
In the case of time-series or video data, the encoder learns to extract the most important temporal and spatial features from the input sequence, while the decoder attempts to either reconstruct the input or predict future frames or steps in the

sequence based on this compressed representation [13].

This architecture is applicable in many different cases, such as machine translation [13], speech recognition [14], and video frame prediction [15]. In video anomaly detection, encoder-decoder models can be used to predict or reconstruct certain frames in a sequence based on the input frames. When the constructed frame deviates significantly from the actual frame, it signals a potential anomaly [16].

2.2.1 Encoder-decoder architecture

Like most neural networks, both the encoder and decoder blocks are built from successive layers that perform a linear transformation (weights × input + bias) followed by a nonlinear activation function. What distinguishes an encoder–decoder architecture is simply how those layers are arranged.

The encoder consists of layers that encode the input data. It does this by progressively reducing the dimensionality, learning to preserve only the most fundamental and important latent features of the input data. The compressed output from the encoder can be processed, pruned or mixed, and then fed into the decoder block. The decoder block learns to interpret these encoded features, and to decode them into the desired output format [9]. A simple high-level illustration of the base structure can be seen in Figure 2.1.

The encoder and the decoder both have similar structures. They are built up of layers, which in turn typically contain one set of weights and one activation function. The input to each layer is multiplied with the weights in some pre-specified way, the product is added to a learnable bias for shifting, and the activation function is applied to the result. The bias term is important to make sure that the network can model data that isn't centered around zero. The activation function is usually a quite simple function, and its main purpose is to introduce nonlinearity to the model. This is important since many real-world data sources include complex, nonlinear relationships. Without activation functions, the model would simply behave like a linear regression model, regardless of how many layers it has. Common examples of activation functions are ReLU, which is simply $f(x) = \max(0, x)$, and $\tanh(x)$, where x would be the resulting feature maps after multiplying the input of the layer with its weights [17].

This output from the activation function is then fed into the following layer, and the process repeats until the end of the block. There is a vast amount of modifications, recursions, normalizations, etc. . . that can be added to this, both within and between the layers, but the core structure rarely deviates from what was described above.

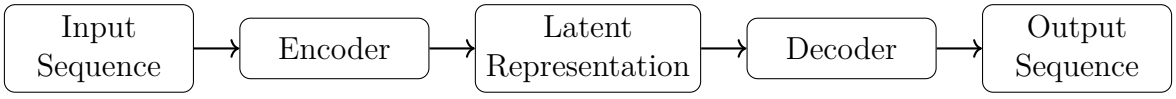


Figure 2.1: High-level encoder–decoder architecture.

2.2.2 Convolutional layers

A convolutional neural network (CNN) is a type of machine model commonly used for decoding and encoding image and time series data. In an encoder-decoder model, it can fill the role of either or both blocks. It uses convolutional layers to automatically extract spatial relationships and abstract features, such as edges and patterns, from the input data [9].

The layers of a CNN are defined by sets of kernels, as well as pooling and activation functions. A kernel is, in the two-dimensional case of image data, simply a square matrix of weights. In most cases it's of the size 3×3, 5×5, or 7×7. Larger kernels can often be equivalently downsized to multiple smaller ones, so it is less common to see kernels larger than 11×11 [18]. Kernels can also be one- or three-dimensional if the data calls for it, but in this report the focus will be on the two-dimensional case.

The function of the kernel is to "slide" along the dimensions of the data for each channel, taking the dot product between itself and the data it is covering at each stop along the way. In mathematical terms, the process can be expressed as:

$$S(i,j) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} I(i+m, j+n) \cdot K(m,n) \tag{2.1}$$

where $S(i,j)$ is the output value at index (i,j), $I(i,j)$ is the input value at index (i,j), $K(m,n)$ is the value of the weight at index (m,n) in the kernel, and M and N are the height and width of the kernel, respectively, so for square kernels we have $M = N$ [9]. See Figure 2.2 for an illustration of the process. In literature and code these kernels are referred to as two-dimensional, but it is important to note that they also scale in the channel dimension. Since image data has three channels, being the red, green and blue pixel intensities, a kernel that processes image data will itself consist of three two-dimensional "subkernels" with individual weights, one for each channel in the input. The summation at each stopping point of the kernel will then occur both along the height and width dimensions as in equation (2.1), but also along the channel dimension.

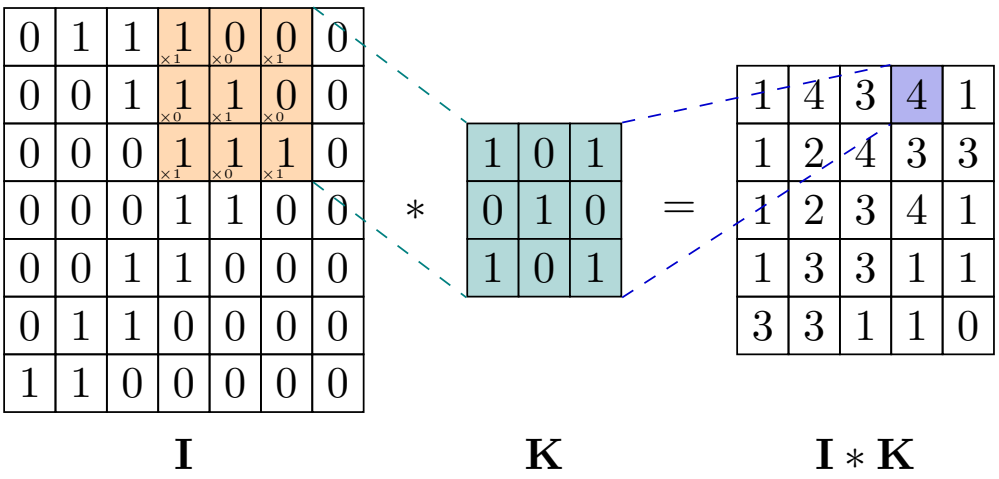


Figure 2.2: Illustration of applying a 3×3 kernel with stride one to a single feature map. © 2021 Janosh Riebesell. Licensed under the MIT License [1].

The kernel is also defined by the stride, the bias, and to a lesser extent, the padding. The stride is an integer constant that governs the step size of the kernel along the dimensions of the data. A larger stride would therefore make the output dimensions smaller, which is often used in the encoder block of encoder-decoder models to decrease the number of features in the representation [19]. The padding is also an integer constant that is used to add zeroed columns and rows along the edges of the data. This is usually used to match dimensions between layers, but also to preserve the information at the borders as much as possible [9].

The next step in an average convolutional layer is to, just like in most other encoder-decoder layers, apply an activation function to each output product, for reasons explained in Section 2.2.1. After the activation, we next optionally pool the results. Pooling is a process where you slide another, non-learnable kernel along the resulting feature map from the activation, with the explicit intent of compressing the results or extracting the most meaningful feature points. The kernel can have different ways of doing this; some common strategies are max-pooling, where only the maximum value in the current kernel position is saved, or average-pooling, where the feature points in the kernel position are averaged together [20]. By summarizing neighboring points in this way, the network can become more robust to small shifts or distortions in the original data and can focus more effectively on the most important information.

The other commonly applied optional step is batch normalization. This is simply the process of normalizing the incoming feature maps of the current batch after applying the convolutional kernels, but before the activation function. The feature maps are rescaled to have a mean of zero, and unit standard deviation. They are then multiplied with a learnable scaling factor γ and added to a learnable bias term

β [19]. The process can be defined by these equations:

$$\mu_{\mathcal{B}} = \frac{1}{m}\sum_{i=1}^{m} x_i,$$

$$\sigma_{\mathcal{B}}^2 = \frac{1}{m}\sum_{i=1}^{m}\Big(x_i - \mu_{\mathcal{B}}\Big)^2,$$

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}},$$

$$y_i = \gamma\,\hat{x}_i + \beta.$$

Since batch normalization is applied directly after the convolution operation, it makes the bias term redundant. The functionality of the original bias term is therefore taken over by the parameters (γ, β). The original bias terms are therefore usually removed in layers that utilize batch norm.
Convolutional layers contain multiple kernels of identical size and structure, but with independent weights. Each kernel processes the input in parallel and produces its own feature map, which means the channel dimension of the output will be the same as the number of kernels in the layer. By compressing spatial dimensions and expanding channel depth, the network can be made to shift its attention away from fine-grained, low-level details and toward more abstract patterns and higher-level concepts.
This process is easiest to understand when considering image data, but can be applied to any data with the appropriate dimensions. To give an image data example, let's consider a RGB image frame of size 224×224 as the input to the first layer of a convolutional encoder. The frame has a channel dimension of three, being the red, green and blue pixel values at each position. The first layer might contain 64 kernels of sizes 7×7, each with a stride of two. Given that the number of incoming channels is three, each kernel will have an effective size of $7 \times 7 \times 3$, meaning that it will have three subkernels. The first kernel starts in the corner of the input, covering the pixels from $(1, 1)$ to $(7, 7)$. The dot-product is taken between each subkernel and its corresponding channel patch, as in (2.1), and the result is summed across the three channels to produce a single feature map value, which is placed at position $(1, 1)$ of the kernel's output feature map. The kernel then moves two steps (since the stride is set to two) along the width dimension, and will now cover and process the pixels from $(3, 1)$ to $(9, 7)$. Once the kernel reaches the opposite boundary of the image, it will simply move two steps (stride) in the height dimension and repeat the process until it has passed through the whole image. This is then repeated for each of the 64 kernels in the layer, resulting in 64 different feature maps. Their spatial dimensions can be calculated as [9]:

$$D_{\text{out}} = \left\lfloor \frac{D_{\text{in}} + 2p - k}{s} \right\rfloor + 1, \tag{2.2}$$

where D_{in} and D_{out} are the input and output dimensions, respectively, p is the padding, k is the size of the side of the kernel (assumed square), and s is the stride.

In the above example we had a square image of size 224×224 with no padding, a 7×7 kernel and a stride of two, so we get

$$H_{\text{out}} = W_{\text{out}} = \left\lfloor \frac{224 + 2 \cdot 0 - 7}{2} \right\rfloor + 1 = \lfloor 217/2 \rfloor + 1 = 109. \tag{2.3}$$

where H_{out} and W_{out} are the resulting height and width of the feature map. Hence, the output will be a tensor of dimension $(64, 109, 109)$, which can be visualized as 64 different two-dimensional feature maps of size 109×109. This tensor is then processed as described earlier in this section, i.e, optionally batch-normed, activated, and then possibly pooled to reduce the spatial dimensions even further and concentrate the features.
It is a common strategy to add some zero padding to the input to make the output dimensions a divisor of the input dimensions. In our example, we might have added a padding of $p = 3$, which would make the output dimensions $H_{\text{out}} = W_{\text{out}} = 112$, making it easier to work with in the following layers [9].
Next, the resulting activated feature maps from the first layer are optionally processed more between layers, and then sent to the next convolutional layer. By halving the spatial dimensions at each layer, we cut down the number of positions each kernel must scan, which heavily reduces computational and memory requirements while expanding the channel depth to preserve (and even increase) the model's capacity for diverse feature representation [19]. When the spatial dimensions decrease, it means that the kernels in the next layer will cover a larger portion of the image, giving it greater context to recognize relationships and patterns from different areas of the image. The wider channel "bandwidth" ensures that there is still space for a diverse set of these more abstract features.
Once all the convolutional layers have been applied, the resulting compressed feature maps can then be used directly for purposes such as classification or clustering. In encoder-decoder models, however, the feature maps are sent to the decoder to generate an output. This could, among other things, be a reconstruction, a translation or a prediction. This report will focus on reconstruction and prediction of video frames, which can be used to detect discrepancies in the incoming video. This is explained further in the next subsection.

2.2.3 Training process

An encoder-decoder model can be trained in both a supervised and unsupervised manner. The choice of training strategy depends primarily on the type of output that is desired from the model. Examples of suitable supervised cases include machine translation or speech-to-text models. In these approaches, the model is fed the untranslated text or audio, and the decoded output is compared to a predefined correct output (the correct translation or transcription). In the unsupervised case, the ground truth instead comes directly from the input data itself, or sometimes from a partly encoded representation of it [8].
In both approaches, the model learns to improve the decoding based on how much the output deviates from the ground truth. The amount of deviation between the ground truth and the corresponding constructed output from the model is often

referred to as the *loss*, and can be quantified in multiple ways. Loss functions will be explained more thoroughly in the next section.

2.2.3.1 Loss functions

The choice of loss function controls what types of reconstruction errors the model will focus on and try to minimize. Therefore, choosing an appropriate loss function is of significant importance, and the best choice often depends on both the type of data that is being processed, as well as what type of anomalies one wants to focus on.
There are many ways to construct a loss function. As mentioned before, the most appropriate choice for a particular model is heavily dependent on the type of data and the aim of the model, which makes it difficult to give an overview of commonly used loss functions [9]. However, for image and video data there are a few very commonly used options that each focus on slightly different types of errors in the image or frame comparison. A few examples of them are:

Mean squared error (MSE):

MSE is one of the more basic and widely used functions for image comparison. It works simply by calculating the squared difference between each pixel in the constructed image and the ground truth image, and then averages this value over all pixels. The equation is the following:

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(I_i - \hat{I}_i)^2 \tag{2.4}$$

where I_i is the pixel value of the output from the decoder, and $\hat{I}_i$ is the ground truth pixel value. Because of the squared difference, this equation will cause the model to focus on single large pixel differences rather than small differences over a larger area, which can cause some excessive smoothing in certain cases. It can also make the training unstable if the data contains outliers or spikes, which will be heavily focused on in that case. Another inherent problem is that it uses no spatial context, so important changes in structure or texture might not be recognized if the individual pixel values remain similar [9].

Structure Similarity Index (SSIM):

SSIM is a more advanced alternative. It mainly evaluates luminance (brightness), contrast (difference in pixel intensities) and structure (structural pattern of pixel intensities).
The formula for SSIM is as follows:

$$\text{SSIM}(I, \hat{I}) = \frac{(2\mu_I\mu_{\hat{I}} + C_1)(2\sigma_{I\hat{I}} + C_2)}{(\mu_I^2 + \mu_{\hat{I}}^2 + C_1)(\sigma_I^2 + \sigma_{\hat{I}}^2 + C_2)} \tag{2.5}$$

Where μ_I and $\mu_{\hat{I}}$ are the average pixel values of the original image I and the predicted image $\hat{I}$, respectively, σ_I and $\sigma_{\hat{I}}$ are the standard deviations (measuring contrast or variability in pixel intensities) of I and $\hat{I}$, $\sigma_{I\hat{I}}$ is the covariance between the original and predicted images, and C_1 and C_2 are constants added to avoid division by zero and to stabilize the formula when the values of the means

and standard deviations are small. SSIM is constructed to simulate human perception, so it often yields more similar-looking reconstructions. However, it is more computationally complex compared to MSE. Also, since it calculates pixel loss based on structural changes, it may struggle to extract meaningful information if the reconstruction is excessively blurry or misaligned [21].

Perceptual loss:
Perceptual loss focuses on feature similarity instead of actual pixel similarity. In this approach, both the true frame and the constructed frame are passed through a pre-trained network, and the loss is calculated from the resulting encoded feature maps. These feature maps contain more abstract information about the image, such as structure and texture. The key idea is that human perception of image similarity is not strictly reliant on pixel-by-pixel matching, but more on these higher-level features. So, the point of calculating the loss from the feature maps is to simulate the way a person processes and evaluates images, focusing on important visual patterns and textures rather than minor pixel differences [22].

Gradient loss:
Gradient loss is based on applying a gradient operator to the reconstruction and the ground truth and calculating the difference from there. The difference can be estimated in a few different ways, the most common ones being simply a L_1 or L_2 norm. If using the L_1 norm, the equation for the gradient loss is the following:

$$\mathcal{L}_{\text{grad}} = \frac{1}{n}\sum_{i=1}^{n}\left\|\nabla I_i - \nabla \hat{I}_i\right\|_1 \tag{2.6}$$

where $\nabla = (\partial_x, \partial_y)$ denotes the spatial gradient operator, I_i is the ground truth frame, and $\hat{I}_i$ the constructed frame. Since the gradients contain information on the differences between neighboring pixels, it will cause the model to focus more on sharp edges and transitions. Hence, this equation is often used to balance the smoothing effect of pure MSE loss [23].

Since these functions each have their own specializations, they are in most cases used in combination with each other. By assigning a weight to each function and simply summing the results, one can control which types of errors are penalized most while still maintaining some focus on the less important ones.

2.2.3.2 Backpropagation and gradient descent

Once the loss has been calculated, it is used to estimate how the weights of the model should be changed in order to minimize the loss on the next pass. The most common approach to this is with gradient descent.
In machine learning, gradient descent is performed by taking the partial derivative of the loss function with respect to every weight in the network individually. This gives an estimation of how much the loss would change if the weight was changed by a small amount. Each gradient contains the magnitude and direction of the changes needed for the weight to reduce the loss. Gradients are calculated using the chain rule as the loss depends on all the weights in the network. Since each weight affects

the output indirectly through the weights in subsequent layers, the gradients are propagated backward through the network using the chain rule, effectively removing the impact of the following layers. The process can be expressed like this:

$$\frac{\partial L}{\partial W_k} = \frac{\partial L}{\partial a_n} \cdot \frac{\partial a_n}{\partial z_n} \cdot \frac{\partial z_n}{\partial W_n} \cdot \cdots \cdot \frac{\partial a_k}{\partial z_k} \cdot \frac{\partial z_k}{\partial W_k} \tag{2.7}$$

where L is the loss function, W_k is the weight at layer k, a_k is the activation function, and z_n is the input to the activation function. An illustration of this process in a one-layer network can be seen in Figure 2.3 below. Once the gradients have been calculated they can be used with an optimization algorithm to calculate the optimal step sizes and update the weights accordingly. The most basic way to do this is simply called gradient descent, and consists of multiplying the gradient from each sample by a static step size, and subtracting the result from the associated weight value. There are, however, a multitude of modified versions of gradient descent, and also algorithms that combine it with other optimization strategies. The most common modifications concern how many samples should be used to build the gradients before taking a step, as well as adding some stochasticity to the process. The number of samples per step affects the volatility of the training. The method of calculating all gradients of the dataset before taking a step is called batch gradient descent. This approach is the most stable, but it might take longer to train and is more memory intensive. Another common approach is mini-batch gradient descent, where only a subset of the gradients are collected before taking a step. This approach maintains some stability while decreasing the memory and computational resources required. These methods are often combined with some form of noise or randomness, which can prevent the optimization algorithm from getting stuck in local minima and help it explore more of the loss landscape [9, 24].

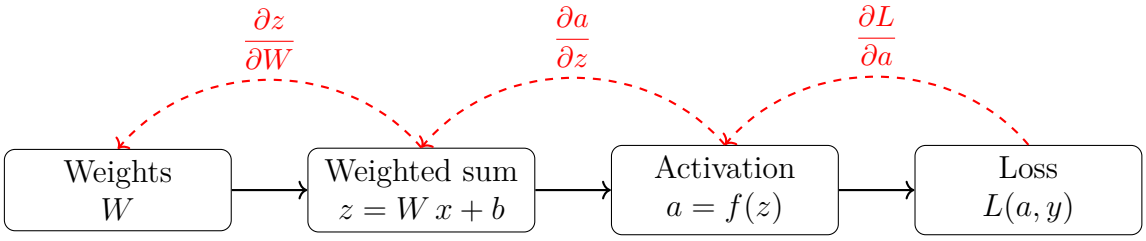


Figure 2.3: Forward (black) and backward (red dashed) flows through one layer. Backward arrows show the explicit partial derivatives that make up $\frac{\partial L}{\partial W}$.

2.2.4 Anomaly classification

In an unsupervised encoder-decoder model designed for anomaly detection, there is no incoming information about what is normal or not, so there is no explicit ground truth or target. Therefore, the ground truth has to be extracted from the data itself. In autoencoders, this is done by simply defining the incoming data as the target, which will cause the network to learn how to encode and decode the input in order to give as accurate a reconstruction as possible [9]. In other encoder-decoders, one might instead use the feature maps from somewhere between the encoder layers as

the target, and have the model learn to reconstruct that. Another strategy that is commonly used for time-series data is predictive decoding instead of reconstruction, where the model is fed a short sequence of data points, with the target defined as the data point immediately after the last point in the sequence [25]. The accuracy between the predicted data point and the actual next point can then be calculated. During training, the level of accuracy between the reconstruction or prediction and the target is measured via loss functions (see Section 2.2.3.1).
Whichever approach is used, the intention is that the network will become more proficient in predicting or reconstructing normal cases compared to anomalous cases. This is why the assumption of unsupervised learning—that the process is normal most of the time—is important. The only reason why the model will be better at reconstructing normal cases is because it has seen them more, and therefore focused more on learning how to process them [8]. If that isn't true, then the process in question is likely not a suitable case for unsupervised surveillance of this nature.
The weakly supervised case is very similar, with the only difference being that the assumption of a normal majority is taken to its extreme, where you assume that the training set only contains normal events.

2.2.4.1 Reconstruction/prediction error

Once a model has been trained, it can be used to evaluate unseen data. The incoming data goes through the model in the same way as the training data, but this time all the weights are frozen, and no learning is taking place. Instead, the reconstruction error is used directly to gauge the normality in the corresponding input data. The reconstruction error calculation is very similar in process and reasoning to the loss calculation during training. The main difference between the two is that the reconstruction error to be used for inference does not need to be differentiable, so the reconstruction error function can be chosen solely on how well it correlates with anomalies [6].
A few common choices for reconstruction error include the pixel-wise L_1 norm

$$E_{L_1}(x, \hat{x}) = \sum_{i=1}^{N} |x_i - \hat{x}_i|, \tag{2.8}$$

and the mean-squared L_2 norm (MSE) given in (2.4) [9]. The SSIM, introduced in Section 2.2.3.1 as a loss function, is also commonly used for the same qualities here [21]. Finally, peak signal-to-noise ratio (PSNR) is defined as

$$\text{PSNR}(x, \hat{x}) = 10 \log_{10}\Big(\frac{MAX^2}{\frac{1}{N}\sum_{i=1}^{N}(x_i - \hat{x}_i)^2} \Big), \tag{2.9}$$

where MAX is the maximum pixel value. The denominator is simply the MSE loss, see (2.4), meaning the PSNR score is closely related to MSE, and has the same functionality but with a decibel scale [26].

2.2.4.2 Thresholding

Once a reconstruction error function that gives satisfying error scores has been selected, a thresholding system is commonly added to decide whether to actually

classify a returned score as an anomaly [6]. Once again, this is very dependent on both the characteristics of the data itself and the types of anomalies that are of the highest importance. A common basic approach is to simply define a static threshold score, and flag any cases with scores lower than it. The threshold can be either hard coded directly, but in many cases it's better to calculate it from a validation set. For example, it could be defined as a number of standard deviations lower than the mean scores when the model is deployed on unseen normal data [27]. This way, the threshold is adapted to the fluctuation and noise levels in the normal data. This strategy is appropriate in cases where normal scores are generally stable and anomalies occur in short bursts. The threshold can also be adapted in an online manner to account for slow but normal drifts such as lighting changes [28]. However, if actual anomalies happen slowly over a longer time window, they might cause a general lowering of error scores, but one that doesn't exceed the frame-by-frame threshold. In these cases, a more appropriate approach could be using a sliding window of averages along the error scores [29]. This way, short fluctuations are averaged out, and drifts can be more accurately recognized.
More sophisticated methods of classification can also be applied in some cases, such as fitting gaussian mixture models to the normal case [30], clustering the features of the validation set [31], and training isolation forest algorithms to name a few [32]. However, given the unpredictable and often times noisy nature of unsupervised and weakly supervised encoder-decoder models, these approaches require thorough supervision and tuning to be worthwhile, and might make the model excessively sensitive to small shifts in the incoming data.

2.3 Transfer learning

Transfer learning leverages knowledge acquired by a model on a large, general–purpose task and dataset, and adapts it for a different but related task [33]. In computer vision, models pretrained on large and varied image datasets, such as ImageNet [34], have already learned to detect low-level edges, mid-level textures and high-level shapes in their early and middle layers. Rather than training from scratch, which can require large quantities of (possibly labeled) data and long training times, one can reuse these pretrained general weights and layers to speed up convergence and reduce overfitting on smaller target datasets [33].
There are two main approaches to transfer learning, being *feature extraction* and *fine tuning*.
In feature extraction, the weights of the pretrained network are frozen entirely, and only the final global-pool and fully connected layers are modified or exchanged to be tailored for the new task, and then trained on the specific dataset in question. Hence, the pretrained layers work as a static feature encoder. A simple use-case for this is creating an image classifier with custom classes. For example, you might want to make a model that takes in video frames of a conveyor in a factory, and returns which items are currently on the belt. You could then utilize a general convolutional model trained on ImageNet as the starting point, adding your own fully connected classification layer and training on your own training frames containing the specific items you want to differentiate between. This will significantly reduce the number of

trainable weights in your model, which in turn decreases the training time drastically compared to training a model from scratch.
In fine-tuning, the initial approach of removing the final classification head is the same as in feature extraction. However, this time one unfreezes part or all of the pretrained weights after training only the fully connected layer for some time, and continues training both the newly unfrozen layers and the fully connected layer with a lower learning rate. This allows the pretrained backbone layers to adapt slightly to the distribution and characteristics of the new data [33]. Going back to the conveyor example, this could allow the model to become more adept at understanding specifically factory environments. The ImageNet dataset contains a lot of natural imagery and outside lighting conditions, so by fine-tuning the model with factory images, it can be made to shift its focus to the possibly harsher lighting conditions, sharp edges and metal reflections that are common in factory settings.

2.3.1 ResNet as a backbone

One commonly used pre-trained group of networks for computer vision is ResNet [35] models. They were introduced in 2015, and stand out for their large number of convolutional layers and their use of residual connections, also known as skip connections. A skip connection is simply a connection between the input and the output of a layer or a group of layers, which allows the incoming feature maps to bypass the layers unchanged. These unchanged features are then added to the resulting feature maps that were calculated as usual by the layers [35]. A simple illustration of one of the ResNet residual layers can be seen in Figure 2.4.

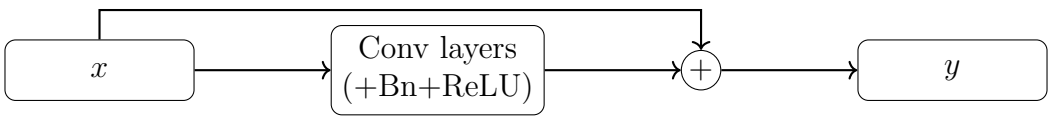


Figure 2.4: A single ResNet residual block: the identity (skip) connection bypasses the convolutional layers and is added back in.

The main benefit of using skip connections in the ResNet models is for greatly reducing the vanishing gradient problem. The vanishing gradient problem arises in deep networks when the back-propagated gradients shrink exponentially as they make their way backwards through the network. This is because they are constantly being multiplied by the small derivatives from the activation functions at every layer, meaning the earlier layers will often receive vanishingly small gradients [36]. The skip connection aids with this by effectively providing shortcuts for the earlier gradients. This is a vital part of ResNet models, as they can contain upwards of 150 convolutional layers, and would therefore likely be very affected by vanishing gradients otherwise.
This depth allows them to learn very diverse feature hierarchies. Shallow layers pick up simple edges and color gradients, middle layers assemble those into textures and object parts, and the deepest layers detect high-level shapes and entire objects. When trained on a large and diverse dataset such as ImageNet, it can be used very effectively both for classification and as an encoder for other purposes [35].

3
Method

This chapter describes the end-to-end methodology for the anomaly detection experiments. It starts by presenting the datasets and the procedure for extracting and preprocessing raw video frames into training, validation and test splits. The architecture of the modified autoencoder implemented in PyTorch is introduced, together with the training approach, loss functions and hyperparameter definitions. Reconstruction errors are next defined, and the thresholding and classification strategies and evaluation metrics are detailed. The chapter ends with a brief layout of alternative models and modifications that were implemented and compared.

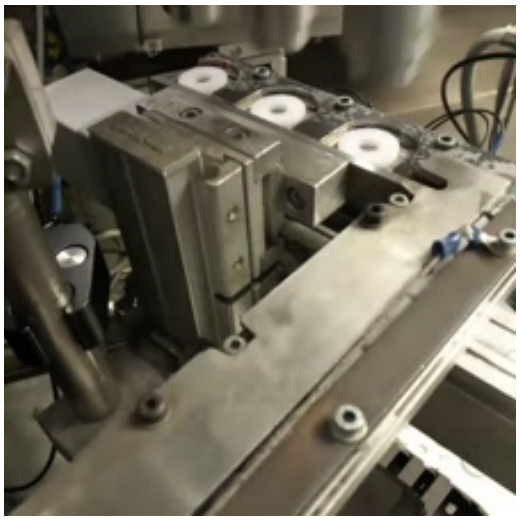
3.1 Datasets

Two out of three of the datasets used for training and evaluation were extracted from actual cases from an industrial setting collected during applications of the partner company's current surveillance system. The third set was collected using a circular can conveyor in the EyeAtProduction offices, which was used for more controlled and isolated testing. Video of the can conveyor was collected at 30 FPS with a Google Pixel 9 telephone (the target inference device), and for the factory cases, with a CAT S42 smartphone.
The three datasets used consisted of:

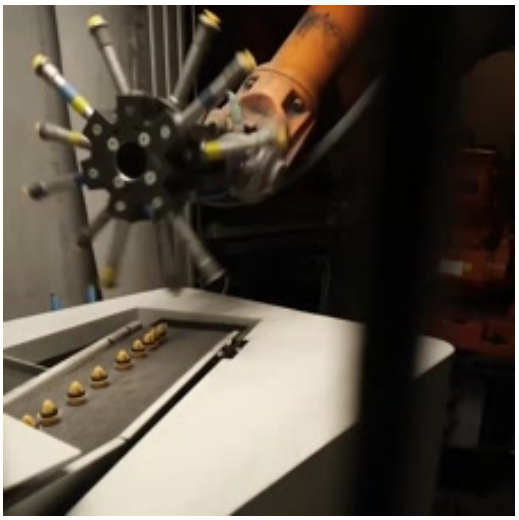
- A feeder and a suction picker transferring foam rings between each other (factory footage).
- A robot arm collecting plastic tops from a conveyor (factory footage).
- Soda cans moving along part of a circular conveyor (controlled office setting).

The cases were chosen for their different types of difficulties as seen by the model, as well as on the basis of having a few different optically obvious anomalies and sufficient normal data for training. A lot of cases were unusable simply because video was saved only when anomalies occurred.
To give a comparable metric of the model's performance, it was also evaluated on the UCSD Ped2 dataset [37] [2]. This dataset contains 16 "normal" training videos and 12 test videos (each 200–300 frames long) captured at 10 FPS from a fixed overhead camera overlooking a pedestrian walkway. Anomalies are defined as non–pedestrian objects, such as bicycles, skateboards or vehicles entering or traversing the scene. It is commonly used for benchmarking video anomaly detection models. The frames in the set have an original resolution of 240×360 but were resized to 224×288, since ResNet models require that the input dimensions be divisible by 32 to work correctly. See Figure 3.1 below for example frames of each of the four datasets.

More examples are also included in Appendix A.1.



(a) Full-size frame from the foam set. The actual training frames were cropped to remove irrelevant areas.



(b) Full-size frame from the feeder set. The actual training frames were cropped to cut out the robot arm.

(c) Still frame from the can dataset.

(d) Still frame from the UCSD Ped2 dataset.

Figure 3.1: Sample frames from each of the four datasets used in the evaluation. Note that the two factory sets were cropped before training, see Appendix A.1 for all versions.

3.2 Preprocessing

The raw incoming .mp4 video data of the evaluation cases was divided into image frames using FFmpeg (v7.1) [38], and saved as individual .jpg files. To keep the input size to the model consistent, all frames were reshaped to a size of 224×224. The can frames were simply compressed using FFmpeg. In all factory cases however,

this was done by selecting an appropriately sized square region of interest (ROI) in the video, and only saving the information within that square. Since the video was taken without the intention of being used for this type of model, the use of ROI selection was critical to avoid background noise and irrelevant areas of movement. It will likely be useful in actual deployment cases as well, as the camera can be set up to capture a large part of the process while still having the model focus only on the most important and least noisy areas.
The size of 224×224 was chosen due to the use of ResNet-18 as a feature extractor, which will be detailed more in the next section. Most pretrained ResNet models available have been trained on images of those dimensions, making it a natural choice. It also means that the images maintain most subtle details, while keeping the computation time comparatively low. The frames were also normalized to have the same mean and standard deviation as the images in ImageNet, which is what the pretrained ResNet network expects.

3.2.1 Dataset structure

Once the frames had been extracted, appropriately reshaped and normalized, they were split into training, validation and testing sets. The training sets contained between five to eight minutes of normal footage, or 9000 to 14400 frames, and the validation sets between two to four minutes, or 3600 to 7200 frames. This was mainly because of the availability of data, but also to keep the training time down. Since the intent is for the model to be relatively quick to set up and train in new scenarios, investigating training on larger datasets was deemed outside the scope of this report. The validation sets were originally built up of randomly selected sequences from within the training sets, but this led to them being too similar to the training sets. Therefore, the validation sets were instead changed to contain frames of normal process taken from a different time than the training sets, to more accurately give an assessment of the performance on unseen normal data.
The validation sets were used for both evaluation of the performance, and for calibration of the anomaly classification thresholds.
The testing sets were made up of both normal and anomalous events. Each event's frames were placed in a separate folder, which also contained a .json file detailing which frame sequences were anomalous. This could then be used for evaluating both the model performance itself, and for testing classification thresholds and algorithms.
The models constructed in this work all had the same input and output format. They accept four sequential frames of sizes 224×224, and return a prediction of the fifth.

3.2.2 Data augmentation

To improve robustness and generalization, different degrees and variations of data augmentation were applied to the training sequences. Data augmentation is a common strategy in many ML applications, where the incoming training data is modified or perturbed slightly in controlled ways, in order to make the model more robust towards those types of changes. A simple example is randomly increasing or de-

creasing the brightness in the frames slightly to attempt to decrease the model's dependence on specific lighting conditions, and make it more focused on the actual contents of the images. The specific augmentation applied to a frame or sequence is usually recalculated between epochs. Other common augmentations are random crops, horizontal flips, slight rotations, and small color jitters. By training on these modified versions of the same sequences, the model is also prevented from overtraining on specific pixel-exact patterns in the training data. In our models it is very important to apply the same augmentation to all frames in the same single training sequence, as well as to the target. Otherwise, the predictive logic of the model will be lost.
Two different augmentation pipelines were tested, namely:

- General_aug, which applies a general perturbation strategy to the data by slightly rotating, translating, and adding lighting and saturation jitter to the sequences. The degree of transformation was picked uniformly and individually for each type and for each frame sequence. The ranges were set to $[-5°, 5°]$ for rotation, $[-2, 2]$ pixels vertically or horizontally for translation, and $\pm 10\%$ shift for the saturation and lighting jitter.
- Lighting_aug, which applies a global lighting shift of $\pm 15\%$ to the entire frame. The intention was to specifically target slow lighting drifts such as sunshine at different times of day, since that is a likely phenomenon to occur in actual applications.

3.3 Complete model description

An illustration of the architecture of the encoder-decoder AD models can be seen in Figure 3.2. The models consist of a baseline implementation, from which all alternative models are built. The layers of the base model are colored black, and the additions of the alternate versions are represented by differently colored blocks. The models are implemented with the help of the PyTorch library [39] and its companion package Torchvision [40]. PyTorch includes modules for defining data sets, data loaders, optimizers, as well as most components in the actual model, while Torchvision supplies video input and output utilities and image transforms. The training data was split into sequences of four input frames and a target frame by a PyTorch Dataset class (leveraging Torchvision for transforms and augmentation), converted into PyTorch tensors, and fed into the model in batches with a DataLoader. For optimization, we use the Adam optimizer [41], implemented with PyTorch.

3.3.1 Model description

When the input tensor with the four sequential frames enters the network, it is first flattened along the time dimension. This is because the ResNet encoder is applied independently frame-by-frame, so the time and batch dimensions do not matter for this step. The encoder used is ResNet-18 [35] pretrained on ImageNet, which is available from Torchvision. ResNet-18 is the smallest of the available pretrained ResNets, and was chosen specifically for speed of inference. The encoder is used purely as a feature extractor, meaning only the actual classification head

was removed, and the weights of the convolutional layers are kept frozen throughout training. The use of a pretrained encoder is one of the main things that makes the model light-weight and fast to train compared to other VAD approaches.
Once encoded, the batch and time dimensions are recovered, giving the data tensor as a whole the dimensions $(B, T, 512, 7, 7)$, meaning each frame has been encoded into 512 feature maps of size 7×7. Next, the feature maps of each sequence is flattened along the time dimension, making the tensor dimensions $(B, 512T, 7, 7)$. A 1×1 convolution is then applied along these $512T$ channels, reducing the channel dimension to 512. The 1×1 convolution has the same functionality as larger convolutions, and will return weighted superpositions of the features at each position in the different channels, which allows the network to assign degrees of importance to each incoming channel. This process effectively merges the feature maps from the four sequential frames into one representation, from which to build the predicted frame.
The merged feature maps from the four frames are now passed to the decoding block for prediction.
The upsampling blocks in the decoder consist of conv2d operations just like the encoder but, in a sense, inverted. Each block applies a transposed convolution (`ConvTranspose2d(kernelsize=3, stride=2, padding=1, outputpadding=1)`) to double the spatial resolution. Effectively, it inserts zeros between the input feature points, then slides a 3×3 learnable kernel over this expanded grid, distributing each point into a small patch of the output, and summing overlaps. This lets the network learn exactly how to invert the compressed feature representation of the encoder. Each layer also applies batch normalization and the ReLU activation function to normalize the feature maps and inject nonlinearity before passing them to the next layer. The number of channels in the representation is halved as the spatial dimensions are doubled, mirroring the encoding process. This is done at each layer leading up to the final one.
At the final layer, the input dimensions will be $(B, 32, 112, 112)$. A final transposed convolution with three kernels is applied, one for each dimension in pixel color space (R,G,B). The output from this convolution is the final output of the model, i.e, the predicted frames. No activation function or normalization is applied here, since we want the model to have access to and be able to explore the entire output space and optimization landscape.

3.3.1.1 Alternative versions

Three modifications to the original model were implemented and tested for comparison, to give an estimate of how each added strategy affects the network as a whole. The modified versions are described below.

Dropout model:

The first modification involves adding dropout between select layers during training. Dropout is a simple strategy that is commonly used to combat overfitting where you discard a set percentage of the incoming features, setting them to zero. The features are randomly and uniformly selected from all channels. The point of this is to force the network to be less reliant on specific single features, making it more general and robust toward small changes. In this ver-

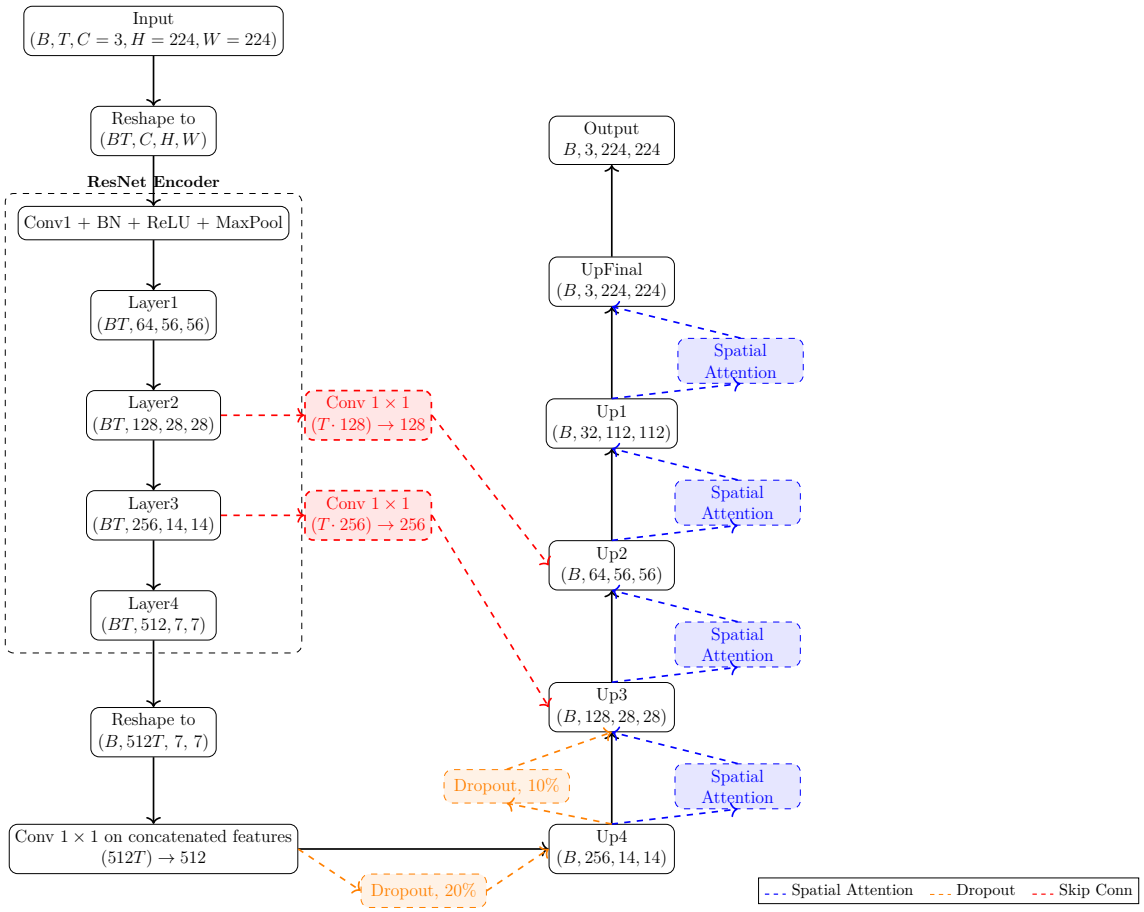


Figure 3.2: Base CNN architecture with alternate versions marked with different colors and dashed lines. The alternate versions employ either spatial attention (blue), dropout (orange), or skip connections (red). Note that the incoming spatial dimensions were assumed to be 224×224, but the network can accept any spatial dimensions provided they are divisible by 32. The dimensions in the convolutional layers concern the output, not the input.

sion, a 20% dropout was applied right before the first decoder layer, and a 10% dropout was applied between the first and the second.

Skip Connections (Skiptwice model):

In the second modification we add two skip connections, one between the second encoder and decoder layers, and one between the third. The time dimensions of the feature maps from the encoder layers are first flattened into the channels, and then reduced by a factor of four using a 1×1 convolution. This follows the same logic as the 1×1 convolution that is applied in the base network after the last layer of the encoder, effectively merging the most important parts of each of the four feature maps of each sequence. Skip connections are, as mentioned in the introduction to ResNets, commonly associated with the vanishing gradient problem of deep networks [36]. However, in this model we are not training any parts of the encoder, meaning these connections won't actually be used for gradient propagation as in the ResNets. Instead, the reason for the

skip connections here is simply to increase the reconstruction quality. Instead of the network having to learn to predict frames from only the mixed and compressed information from the output of the decoder, it is also provided with lower-level information from the earlier in the decoding process, which makes reconstructing small details easier. This can be both a good and a bad thing, since it doesn't necessarily mean that the difference between non-anomalous and anomalous predictions will be greater. It could simply make both of them sharper, which might not help to differentiate between the two cases.

Spatial Attention model:
In the third modification, we introduce spatial attention blocks between the decoder layers. The function of these layers is to amplify the spatial regions that the network is focused on. The layers work by first, at each spatial position (h, w), calculating the maximum and the average feature value across all channels. This results in two maps with the same spatial dimensions as the input, one containing the maximums and one containing the averages. Then, a learnable 7×7 convolutional kernel is slid across these two maps, and the result is multiplied with a sigmoid activation function which rescales the values to within $[0, 1]$. The output is effectively a mask signifying which 7×7 pixel regions contain the most information, and are most important for reconstruction. The mask is multiplied with the original incoming feature maps, thereby amplifying these more important spatial regions while dampening the impact of the less important ones.

3.3.2 Implementation details and training setup

The training of the model is governed by the loss function, which is defined as a weighted sum of MSE, gradient loss, and SSIM, defined in (2.4), (2.6), and (2.5). With a weighting variable for each term, this gives a total loss function of

$$L_{total}(I, \hat{I}) = \alpha L_{MSE}(I, \hat{I}) + \beta L_{gradient}(I, \hat{I}) + \gamma L_{SSIM}(I, \hat{I}) \tag{3.1}$$

where I and $\hat{I}$ are the ground truth and the predicted image, respectively, and L_{MSE}, $L_{gradient}$ and L_{SSIM} represent each of the three loss terms. The parameters α, β and γ are simply scaling factors to control how much each loss function contributes to the total (see Table 3.1). These values were chosen simply by trial and error early on in development. A range of value sets were tried, and the final set was chosen based on the sharpness of the reconstructions and the difference in PSNR between anomalous and normal cases. With a higher MSE parameter the reconstructions became excessively blurry, and with increased gradient and SSIM loss the images looked better upon inspection, but had lower PSNR scores overall, which partly drowned out the erroneously predicted anomalous regions, meaning the contrast between anomalous and normal PSNRs was lessened.
The learning rate, number of sequences per epoch, batch size, and total epochs are also summarized in Table 3.1. For each epoch, N sequences are randomly picked from a pool of all allowed sequences in the training set. This was done to keep the training times consistent between cases with differently sized training sets. The batch size was chosen as a balance between GPU memory requirements and training

stability. Models were trained for 10 epochs for all cases except UCSD Ped2, where versions trained for 40 epochs were also evaluated. Longer trainings did, in general, result in a very slight increase in sharpness in the predictions, but also made the models more sensitive to slight angle shifts and lighting differences, so it was deemed unnecessary for our purposes. The training was performed on a desktop workstation equipped with an NVIDIA GeForce RTX 3060 GPU and an AMD Ryzen 3 2200G CPU.

Table 3.1: Hyperparameter settings used in the final experiments.

Hyperparameter	Value
MSE weight α	0.6
Gradient weight β	0.25
SSIM weight γ	0.15
Learning rate	3×10^{-4}
Sequences per epoch N	4000
Batch size	32
Epochs (factory datasets)	10
Epochs (UCSD Ped2)	10, 40

3.4 Anomaly scoring and classification

After training, the models are tested on unseen footage containing both anomalous events, normal flow, and "bait" events such as global lighting changes, shadows moving in the frame, or light vibrations. The predicted frames are compared with the actual frames using the PSNR given in (2.9).
A static classification threshold is used for the actual anomaly classification. The most appropriate threshold depends both on the noisiness and complexity of the normal process, as well as the desired true positive rate (TPR) vs false positive rate (FPR). Naturally, if the allowed PSNR region is made smaller, it will increase both rates. To filter out frame by frame noise, an averaging window of size 10 is first applied to the PSNR scores of both the validation set and the incoming test data. This frame-by-frame noise is often caused by unpredictable events, such as an object or machine part jerking into motion or suddenly stopping. This averaging window can be adapted to each case, allowing the user to select the time scale of the expected anomalies.
In this report, the threshold was calculated from the standard deviation of the PSNR scores of the validation set. However, because of the more sporadic and jerky movement in the two factory sets, and the fact that both processes also involved a large portion of static waiting, they both had PSNR scores that fluctuated by a few dB even during a normal cycle. Therefore, it might have been more effective to define the threshold as a set number of dB below the worst PSNR score in the validation set results.
To perform a comparison between the models and the augmentation techniques, receiver operating characteristic (ROC) curves were constructed for each of the

different datasets, and the area under the curve was calculated. The ROC curve is an evaluation metric commonly used for binary classification models, for example in [37] and [8]. It functions by continuously sliding the threshold from 0% sensitivity (flagging no sequences as anomalous) to 100% (flagging all sequences as anomalous). This gives an estimate of the overall performance of each model, independent of the thresholds chosen [42]. It can also be used to find an appropriate threshold for each deployment case. The slope of the curve at any point will be equivalent to

$$\frac{dTPR}{dFPR} = \frac{\text{increase in true positives}}{\text{increase in false positives}}. \tag{3.2}$$

In other words, the slope gives an estimation of the cost of increasing the threshold at any given point. A steep slope means that you gain many true alarms for few false alarms, whereas a shallow slope means that each additional true alarm costs you many more false positives. The results of the training on each data set, including the ROC curves and some case-by-case PSNR graphs, will be presented in the next chapter.

4

Results

In this chapter we quantify how well our predictive autoencoders detect anomalies under different conditions. We begin in Section 4.1 by comparing the three model variants (baseline, with dropout, with skip–connections, and with spatial attention) on each of our factory test sets, as well as the can set. We will first look at specific anomalous events to give an idea of the differences in performance between scenarios, and then compare the ROC curves and AUC scores for each dataset as a whole. In Section 4.2 we assess how the two augmentation pipelines ("General_aug" vs. "Lighting_aug") affect detection performance, especially false positive rates and resilience to lighting changes. Finally, Section 4.3 then benchmarks our best models on the UCSD Ped2 dataset to provide a comparison metric for existing work. The models used to generate all graphs were trained for 10 epochs with 4000 sequences per epoch (unless noted otherwise), which took an average of 20 minutes in the unagmented case on the desktop described in Section 3.3.2. The augmented versions increased the training time to approximately 24 minutes, which can be attributed in part to the rather low-end CPU used. The different model versions didn't differ much in training times since the modifications didn't significantly impact the number of trainable weights.

As mentioned earlier, we compute anomaly scores by measuring how much the predicted frame differs from the true frame. Figures 4.1 and 4.2 illustrate this for an anomalous and a normal sequence, respectively.

In Figure 4.1, a trained model was fed the four-frame sequence right before the ground truth frame in the left panel. The sequence shows a fallen can moving along the conveyor. The middle panel is the prediction produced by the model, where the fallen can is almost entirely omitted, leaving a slight blur. The right panel is a heat map of the mean absolute error. Notice the bright region where the can should be and the more diffuse error in the larger surrounding area. That surrounding blur comes from the hierarchical nature of the decoder layers. Each up-sampling layer in the decoder spreads reconstruction errors over an increasingly large receptive field, leading to a slightly lowered quality compared to in the normal case in 4.2, even outside the anomalous region. The PSNR score for this prediction is printed above the images.

Figure 4.2 shows the same three panels resulting from inputting a normal four-frame sequence taken immediately before the fallen can enters the frame, so lighting and context match closely. Here, you can still see some blurring at the edges of the moving cans (visible in the heat map), but the overall PSNR remains much higher than in the anomalous case.

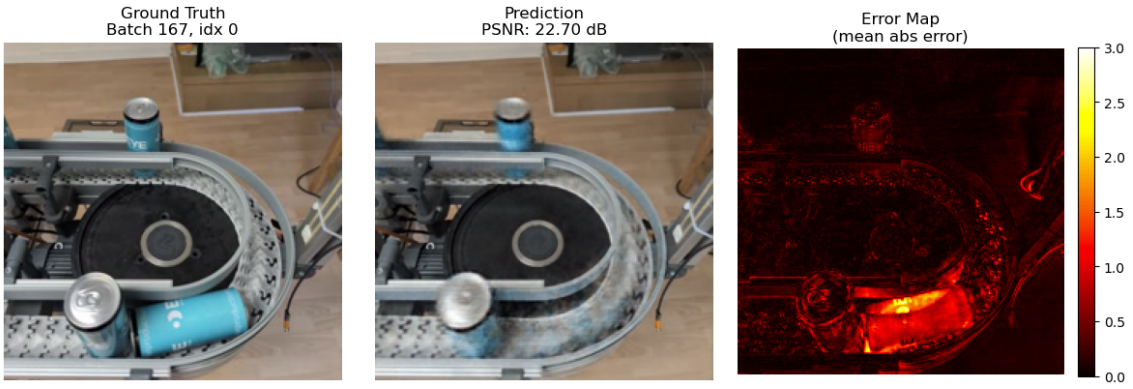


Figure 4.1: Reconstruction of an anomalous sequence in the can dataset, where a can has fallen over. The heat map to the right shows the degree of error at each pixel position.

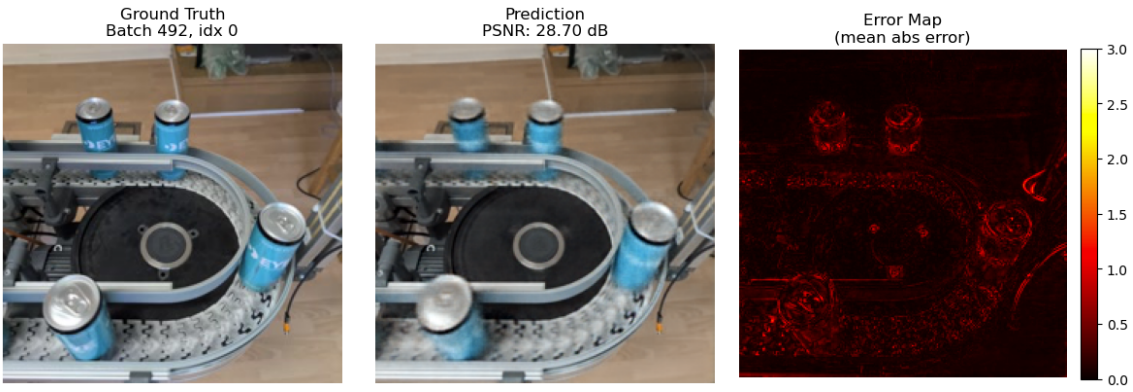


Figure 4.2: Reconstruction of a normal sequence in the can dataset.

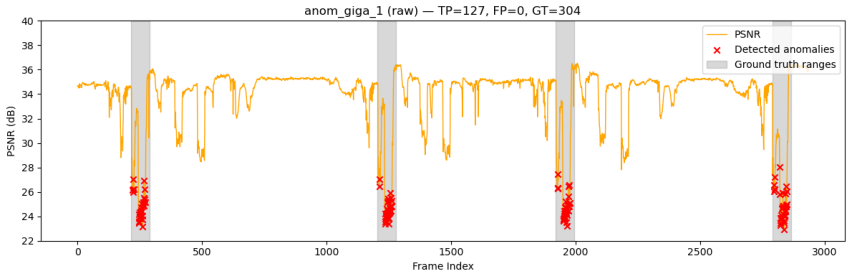
4.1 Model comparisons

Below are the results from training and deploying the models on each of the evaluation datasets. No data augmentation was applied here. Each section outlines the results from training and deploying the different versions of the model on the respective dataset. The first set of figures in the foam section will show the PSNR scores from feeding a typical test set into each model after training, as well as the scores from the validation set and the smoothed versions of both curves, where an averaging window has been applied. The true anomalous regions are shaded, and the detected anomalous frames are marked by the red "x"es. The blue dashed lines show the basic static threshold calculated using the mean and standard deviation from the validation PSNR scores. These threshold values are also printed in the legends of the smoothed graphs. The number of true positive and false positive detections, counted by frame, are printed at the top of the graphs. These graphs are not shown for the other two datasets, as the differences in model behavior are very similar for all three sets. The other type of figure shown in this section will be the frame-by-frame and case-by-case ROC curves. The frame-by-frame ROC curves simply treat each frame within the anomalous regions as a positive, and all frames outside that as a negative. The case-by-case curves however, instead treat each anomalous or non-anomalous interval as one single case. If the model detects any of the frames within one of these intervals, then the whole interval is interpreted as having been classified anomalous. This might give a better understanding of how the model might perform in actual applications, where the number of frames classified as anomalous during a single anomalous event might not be as important as whether or not the event was noticed at all.

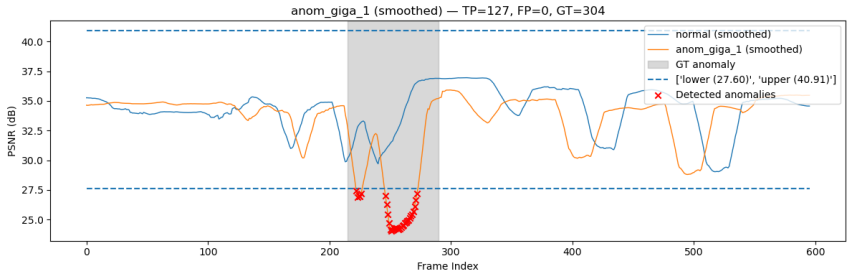
4.1.1 Foam set

The test set used for construction of the graphs contains four cycles of partly anomalous production. The foam ring tray first enters the frame in three separate motions from the right, which is normal. It then stays idle for around 15-20 seconds until the suction picker enters the frame from the top. The picker moves down in one motion and presses on the rings. It then moves up again in the same manner, taking the rings with it. The anomaly studied here is when one of the rings do not stick to the picker and gets left behind on the tray. The anomalous regions mark the period from when the picker first starts to move away from the tray without one of the rings, until the tray with the missed ring leaves the frame completely. Some still frames from the foam dataset can be seen in Figure A.3(see Appendix A.1).
Looking at Figure 4.3a, we can clearly see the differences in the quality of the prediction for the different stages of the process. The graph shows the PSNR scores from deploying the base model on one of the test sets, where the left-most foam ring is missed four cycles in a row. Right after each anomalous event, i.e after the foam tray has left the frame for refilling, we see three distinct increasingly large drops in PSNR score caused by the three steps that the foam tray takes when entering the frame. The subsequent drops right after the movement and right before the anomaly takes place are caused by either shadows from the picker moving above the

frame, or the downward motion of the picker when it first enters the frame. Figure 4.3b shows a zoomed in view of one of the anomalous periods, where the curve has been averaged with a window of size 10. This figure also includes the scores from the same sequence in the validation set (blue curve), but where the ring is correctly picked up.
The performance and tendencies of the base model, Figure 4.3, the dropout model, Figure 4.4, and the attention model, Figure 4.5, were all very similar in this case. All three results show very similar scores for both the normal and the anomalous events. As shown in the legends in the smoothed graphs, there is a slight narrowing of the range of the normal scores in the dropout model, and a slight increase in the normal scores in the attention model compared to the baseline.
The skip connection model, Figure 4.6a, showed the largest deviation compared to the others. It shows a general elevation of the general PSNR scores, but also a higher degree of very short-term decreases in the non-anomalous intervals. The spikes are most severe when the foam tray starts one of its movement cycles or when the picker first enters the frame. The reason behind this is thought to be that the model relies more on the spatial features and less on motion for prediction, since it has access to less obscured feature maps compared to the other models. Because of this it might not have been forced to learn as much temporal context as the other models, leading to worse performance in these more motion-dependent sequences. However, the impact of these spikes is lessened by the averaging window, which can be seen in the smoothed version of the curve, Figure 4.6b.



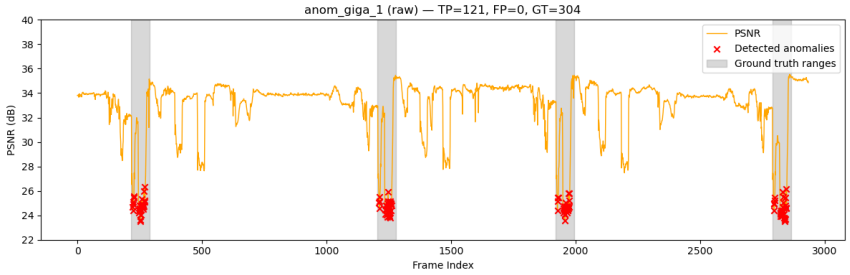
(a) Original, base model.



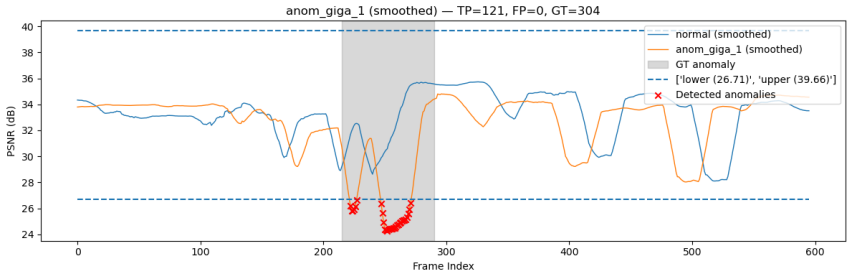
(b) Smoothed, window-size 10, base model.

Figure 4.3: PSNR scores produced by the base model when deployed on one of the foam test cases. True anomalous regions are shaded.

The ROC curves for the foam dataset can be seen in Figure 4.7. In addition to the test case used for the graphs above, another video of the same anomaly occuring at a later time was used, as well as one in which a metal rod drifts in and out of frame.

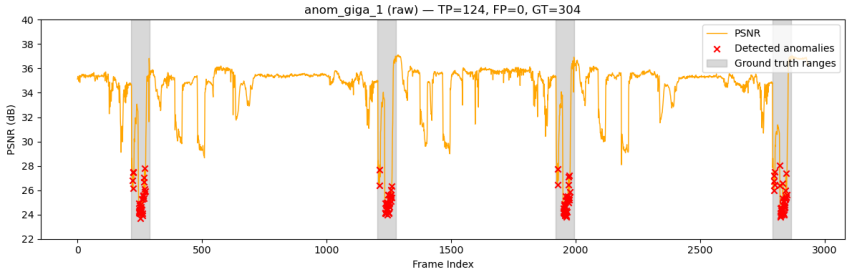


(a) Original, dropout model.

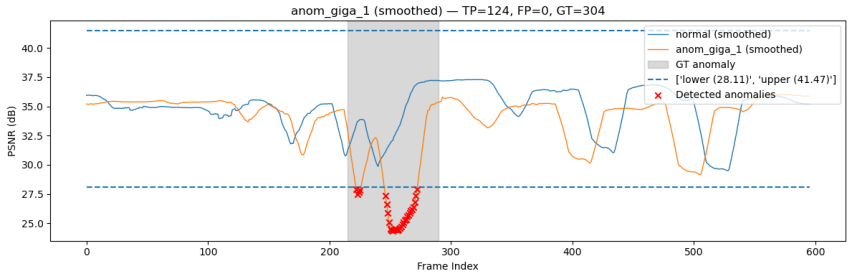


(b) Smoothed, window-size 10, dropout model.

Figure 4.4: PSNR scores produced by the dropout model when deployed on one of the foam test cases.

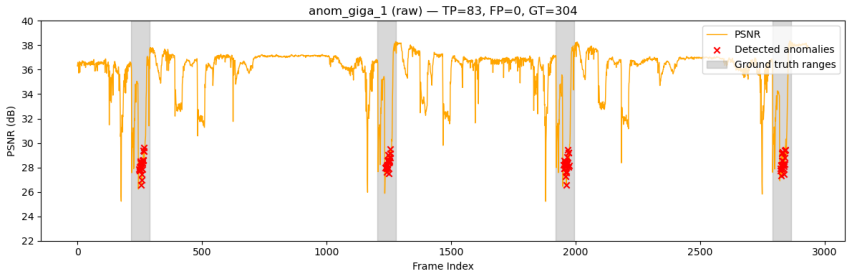


(a) Original, spatial attention model.

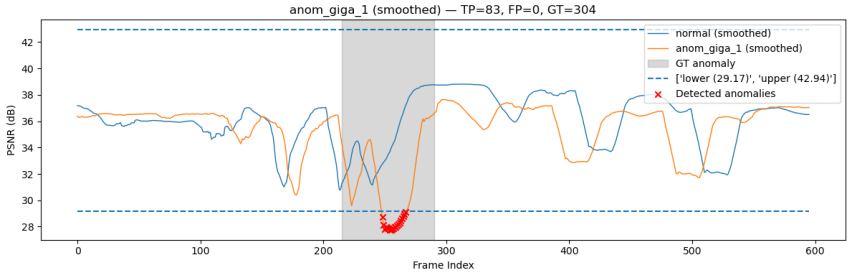


(b) Smoothed, window-size 10, spatial attention model.

Figure 4.5: PSNR scores produced by the spatial attention model when deployed on one of the foam test cases.



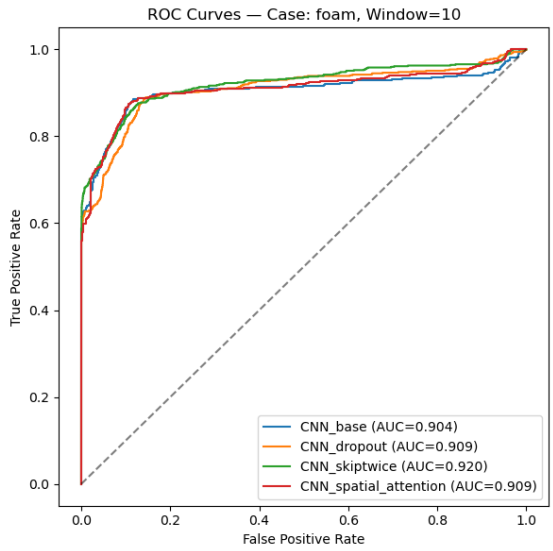
(a) Original, skiptwice model.



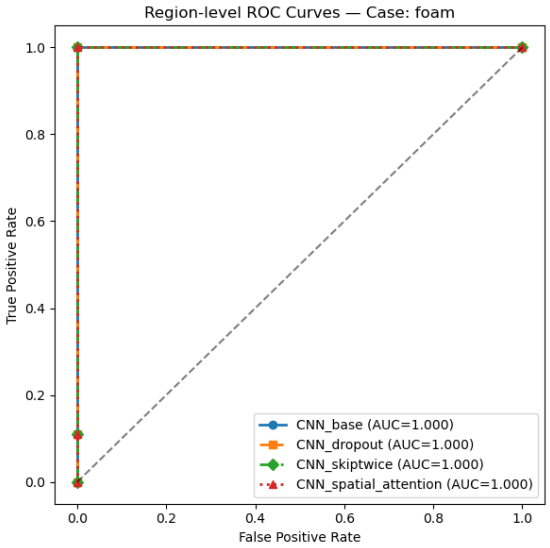
(b) Smoothed, window-size 10, skiptwice model.

Figure 4.6: PSNR scores produced by the skiptwice model when deployed on one of the foam test cases.

This dataset does not contain any heavy light shifts or other large potential causes for false positives, which is why the casewise ROC curve, Figure 4.7b shows perfect performance for all models. What this curve means is that the worst prediction in each of the anomalous regions had a lower score than the worst reconstruction in any of the normal regions. Note that both of these curves were calculated using the smoothed PSNR scores, which is why the large spikes in Figure 4.6a don't show up as false positives.
In the frame-by-frame ROC curve, Figure 4.7a, we see a slightly better performance displayed by the skiptwice model compared to the others, with an AUC value of 0.920. It also displayed a slightly higher true positive rate (TPR) at zero false positive rate (FPR) than the other models. This can be seen in the graph, where the skiptwice curve stays parallel to the y-axis slightly longer than the other models before starting to flatten out.



(a) ROC for the foam set (frame-by-frame)



(b) ROC for the foam set (case-by-case)

Figure 4.7: Two versions of ROC curves for the foam set, constructed by deploying each trained model on all testing sequences in the set.

4.1.2 Feeder set

Some still frames from the feeder dataset can be seen in Figure A.2 (see Appendix A.1), along with a frame showing a typical anomalous occurrence, Figure A.2g. The anomalies in the test sets for this case are mainly occurrences where the tops on the conveyor arrive into frame in clusters rather than neatly in a line. This causes the robot picker to struggle to accurately assess their individual positions, and as a result, many of them are not picked up off the conveyor belt. The averaging window was set to a size of five. This was to be able to detect short-term anomalies where the erroneous clusters do not stop moving within the frame, while still filtering out the shortest spikes.

There are two main attributes that distinguish this dataset from the others. First, the conveyor moves in quick bursts rather than continuously, as it needs to be still while the robot arm picks up the tops. Second, the tops on the conveyor do not come in a standardized configuration in the normal case, nor do they have a standardized stopping point. The behavior is anomalous only if a top deviates from the middle line of the conveyor or if it has fallen over. This, coupled with the long waiting times between motions, made it difficult for the models to get an accurate understanding of the normal flow. Since the training set only contains approximately eight minutes of normal footage, where the conveyor might only have moved approximately 20-30 times, the models will only have seen that many still configurations during training. The prediction quality of unseen incoming normal configurations was therefore partly tied to how familiar that specific configuration is, i.e, how close it was to one of the configurations in the training set. This behavior can be seen in the averaged PSNR graph in Figure 4.8. The graph shows the resulting PSNR scores from deploying the base model on one of the test sets (orange) where a cluster of tops enters and

leaves the frame, as well as on the validation set (blue). While the scores from the anomalous event are still significantly lower than any of the validation scores, we can also clearly see the differences in scores due to normal configurations from the blue validation curve. The scores increase at frame indexes 120 and 250 which is when the conveyor moves in the validation set, bringing a new configuration of tops into view.

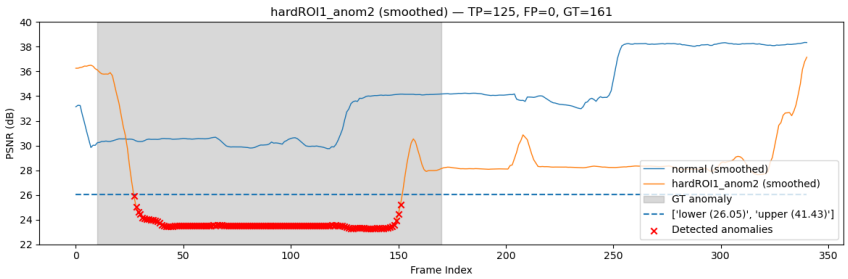


Figure 4.8: Smoothed PSNR scores produced by the base model when deployed on one of the feeder test cases. Window-size 5.

This behavior appears in the same manner in both the spatial attention model and the dropout model, so these graphs will be left out. However, the skiptwice model showed some slight promise when it comes to tackling this problem. The scores from deploying the skiptwice model on the same test and validation sets can be seen in Figure 4.9. Here we once again see the result of the more spatially oriented nature of this model. The configuration problem is not seen as clearly while the conveyor is still here, but instead we get increased drops whenever the conveyor is moving. The scores are still lower during anomalous movement compared to normal movement, but these drops still constitute a possibly hightened risk for false positives during inference.

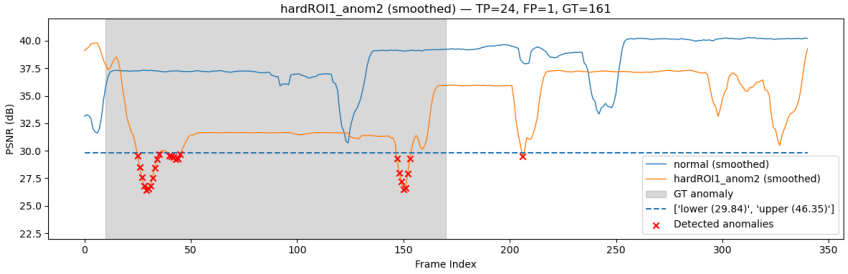
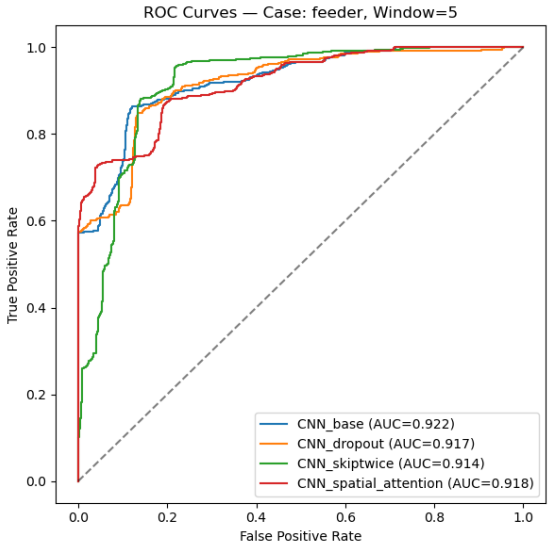


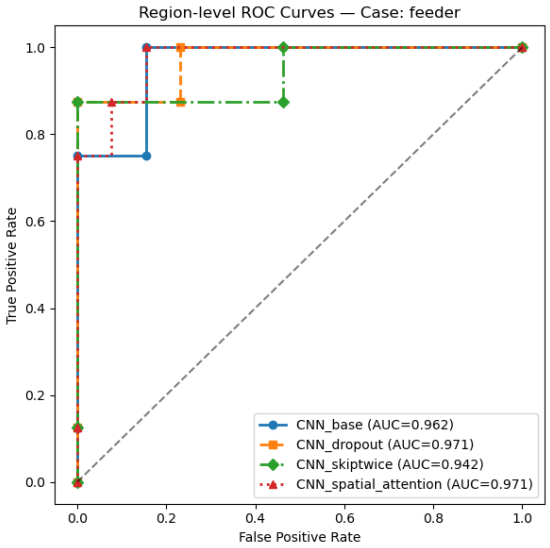
Figure 4.9: Smoothed PSNR scores produced by the skiptwice model when deployed on one of the feeder test cases. Window-size 5.

The ROC curves from deploying the models on all the feeder test cases can be seen in Figure 4.10. In the frame-by-frame graph, Figure 4.10a, we can clearly see the effect from the spiky nature of the skiptwice model, as it starts to produce false positives very early on. However, it still achieve a higher TPR than the others before the curve flattens completely. As for the other models, they perform quite similar to each other. The spatial attention model reaches a slightly higher peak than the

others before starting to produce false positives, but beyond a TPR of about 0.75, its performance levels off and is on par with or slightly behind the other models.
In the case-by-base graph, Figure 4.10b, we can see that the dropout and the attention models achieved the highest AUC scores. The skiptwice model did reach the same TPR at zero FPR as the dropout model, and higher than the attention model, but it then made a few more erroneous classifications than the others before reaching the final anomalous case.



(a) ROC for the feeder set (frame-by-frame)



(b) ROC for the feeder set (case-by-case)

Figure 4.10: Two versions of ROC curves for the feeder set, constructed by deploying each trained model on all testing sequences in the set

4.1.3 Can set

Some still frames from the can dataset can be seen in Figure A.1(see Appendix A.1), along with frames showing a few of the anomalous cases tested. This dataset included a few test scenarios showing anomalous events after turning off the ceiling lamp to assess the sensitivity to heavy lighting changes. The lighting differences can be seen in the anomalous frames, where Figures A.1d and A.1e were taken with the lights turned on, and A.1f was taken with the lights off. The averaging window was not applied here since the cans move slowly and continuously along the conveyor, meaning the spiky behavior of the skiptwice model was not observed here.
To get an estimate of the performances of the models in this case, see Figure 4.11. The graph shows the PSNR scores from deploying the base model on both the validation set and one of the test sets, just like in the graphs in the previous sections. The test set in question first shows a can with a post-it note on the side traversing the conveyor, and then a fallen-over can traversing it. As seen in the anomalous region on the left part of the graph, the post-it note caused a very minimal decrease in prediction scores. This is likely due to the small size of the note in combination

with the angle reducing how much of the frame it is taking up even more, which means that even if the error is large in the small region where the note is supposed to be, it will not be enough to lower the total error of the prediction significantly. However, in the right side of the graph we can see that the fallen-over can causes a quite noticeable drop. The performance of the other models on this test set was largely the same, so those graphs will not be included.

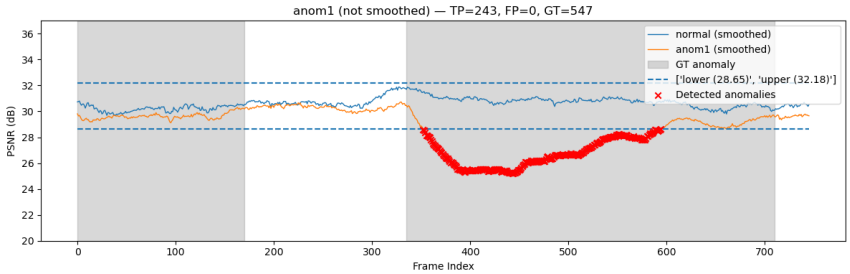


Figure 4.11: Smoothed PSNR scores produced by the base model when deployed on one of the can test cases. Window-size 1.

Another result worthy of discussion is from the deployment of the models on the test set in which the ceiling lamp was turned off. The result for the base model can be seen in Figure 4.12. The lamp was off for the entire duration. The three anomalous events in order of occurrence were a can with a post-it note, then a fallen can, and then an upside-down can moving along the conveyor. It is obvious from the graph that the anomaly detection qualities are almost completely lost in this case, since even relatively large anomalies like the fallen can give almost no change in performance compared to the normal regions. This behavior is mirrored in the other models. The skiptwice model raises the PSNR scores of both the validation and the test set by a few dB, but it does not detect the actual anomalous regions to any larger degree than the others.

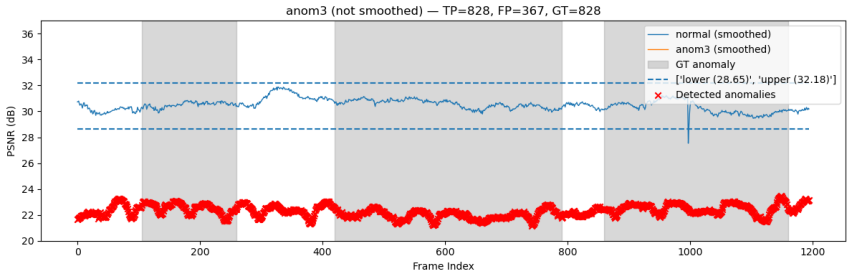
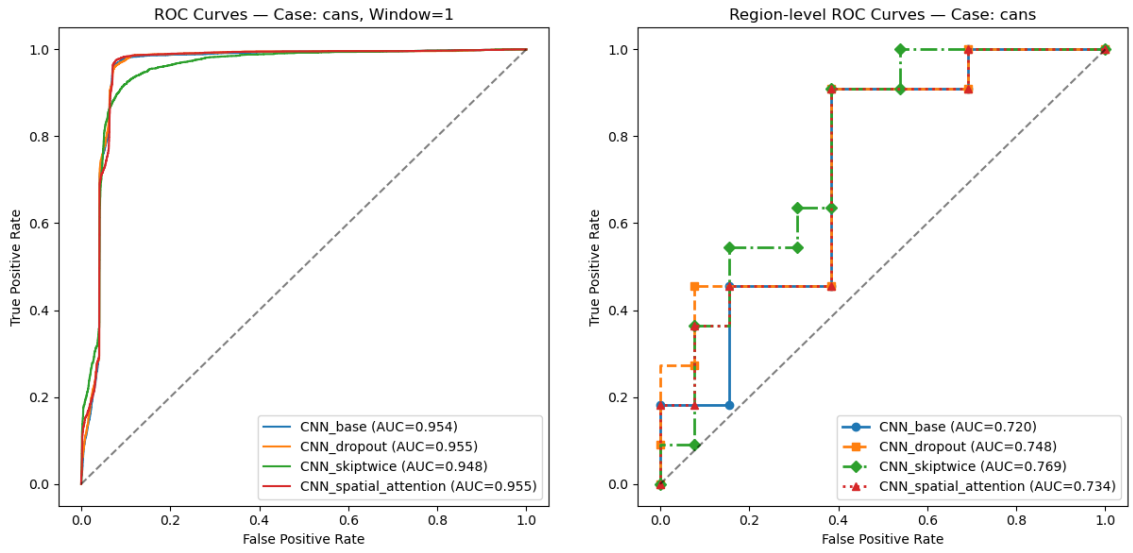


Figure 4.12: Smoothed PSNR scores produced by the skiptwice model when deployed on one of the can test cases. Window-size 1.

The ROC curves from deploying the models on all the can test cases can be seen in Figure 4.13. In the frame-by-frame graph, Figure 4.21a, we can clearly see the effects of the lighting changes in the bottom left of the graph, as some false positives are produced very early on. The PSNR scores from those poorly lit test sets were lower than most of the actual anomalous scores from the sets with normal lighting, meaning that all frames in those sets, both inside and outside of the actually

anomalous regions, will almost immediately be classified as anomalous. After that, the TPR rises quickly for all models with little FPR cost. The skiptwice model doesn't reach quite as high of a peak in TPR as the other models before deviating. The performances of the other models are almost identical for this set.

In the case-by-case ROC, Figure 4.21b, we see a slightly increased initial performance by the dropout model which is later taken over by the skiptwice model. The severity of the poor lighting condition robustness is made clear here as well, as every one of the models classify those cases as anomalous very early. This problem of changing lighting conditions and its possible solutions will be discussed in the next section.



(a) ROC for the can set (frame-by-frame) **(b)** ROC for the can set (case-by-case)

Figure 4.13: Two versions of ROC curves for the can set, constructed by deploying each trained model on all testing sequences in the set.

4.2 Data augmentation

In this section, the results of adding slight data augmentation to the incoming sequences will before training will be presented. Two different methods of augmentation were tested, one that applies four different slight perturbations, and one that only applies a global light shift. They will be referred to as General_aug and Lighting_aug respectively, and their exact definitions can be found in Section 3.2.2. This was done as an initial exploration of the lighting sensitivity problem shown in the results from the can set, as well as the problem of overfitting on specific object configurations from the feeder set. Since the test data was not collected for this specific purpose, not all of the cases or datasets were affected by these problems. Therefore, only results where the augmentations had a significant impact on the performance will be shown and discussed.

4.2.1 Augmented feeder set

As mentioned in Section 4.1.2, a potential cause for concern in this case is the risk of overtraining on specific configurations of the tops in the training set. This was shown in the large differences in PSNR scores when applying the models to the validation set (the blue curve in Figure 4.8), where the normal scores ranged from 30 dB to 38 dB.
The graphs from deploying the base model, trained on the feeder set using each of the augmentation strategies, on the same test and validation set as in Figure 4.8 can be seen below. In the graph where General_aug has been applied, Figure 4.14, we see a clear decrease in the fluctuations of the validation set (the blue curve), as the scores when the conveyor is still now range between 31 dB to 34 dB. However, it seems to have come at the cost of some motion knowledge, as the PSNR scores now drop slightly more while the conveyor is moving compared to the unaugmented version. In the Lighting_aug graph, Figure 4.15, we see the same tendencies but to a lesser degree. The PSNR scores in the normal case now range between 31 dB to 36 dB when the conveyor is still, but the drops during motion are not as present.
The changes in PSNR scoring were consistent across all models, meaning they all returned more stable normal scores, but were slightly worse at prediction motion. For the skiptwice model which already had this problem to some degree, this made the motion drops even more severe, further increasing the risk for false positives. Since it seems like the effects of skip connections and data augmentation are related in cases like this, it might be wise to focus on utilizing one of them, and then possibly using the other to a lesser degree as a compliment.
In general, these results show potential for the augmentation strategy as a whole when it comes to this problem, but also that care needs to be taken not to apply too much of it, so as to not lose too much temporal understanding in the models.

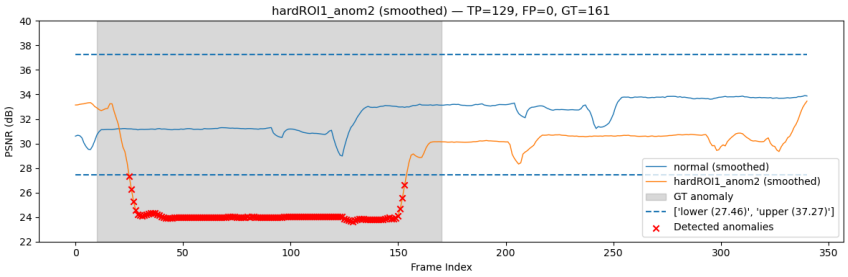


Figure 4.14: Smoothed PSNR scores produced by the base model trained with General_aug, deployed on the same feeder test set as in the unaugmented case. Window-size 5.

The ROC curves from the deployment of the augmented versions of the models on all the feeder test sets can be seen below. All parameters and settings were the same as in the unaugmented case during training and inference, in order to give an accurate assessment of the impact of the augmentations. Figure 4.16 shows the frame-by-frame curves. When comparing these graphs to the unaugmented results in Figure 4.10a, we see a clear improvement in AUC scores for the base, dropout and spatial attention models in the General_aug case in Figure 4.16a. The graphs stay closer

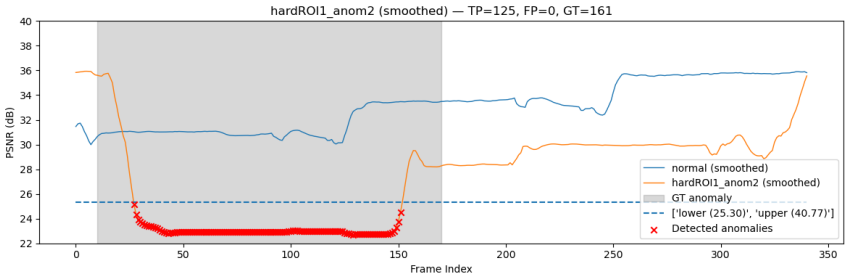
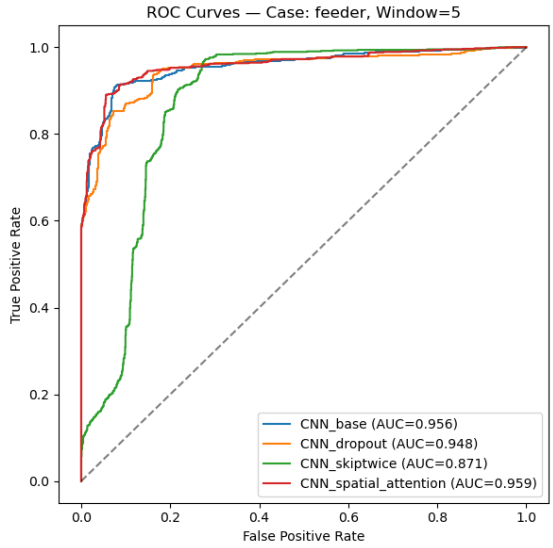
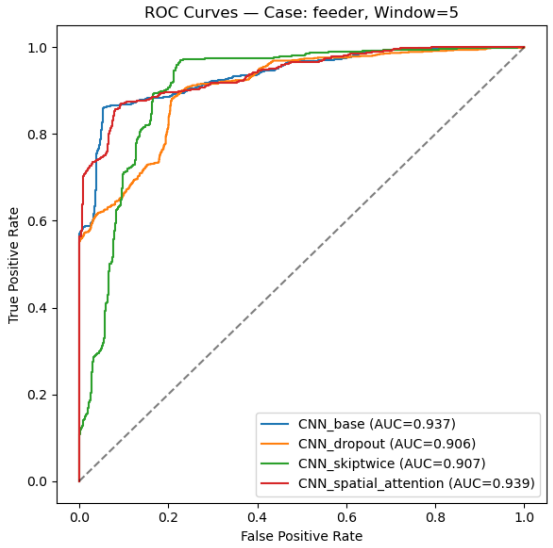


Figure 4.15: Smoothed PSNR scores produced by the base model trained with Lighting_aug, deployed on the same feeder test set as in the unaugmented case. Window-size 5.

to the y-axis much longer before flattening out, meaning fewer false positives are being produced while the TPR increases. However, the skiptwice model performs slightly worse than in the unaugmented case, producing a significant amount of false positives very early on. This is likely due to the excessive motion drops discussed earlier, as almost all movement, anomalous or not, will be classified as anomalous before any of the still PSNR scores are seen.
The Lighting_aug curves in Figure 4.16b exhibit some of the same changes, but to a lesser degree. The AUC scores of the base and the spatial attention models are slightly higher than in the unaugmented case, but not as high as in the General_aug case. As for the skiptwice and dropout models, their AUC scores were slightly worsened by this augmentation.
Moving on to the case-by-case curves in Figure 4.17, we see that both types of augmentation served to improve the TPR at zero FPR for all models except for the spatial attention model in the Lighting_aug case. However, it seems to have made the final anomalous sequence slightly harder to detect, requiring a few more false positives to reach 100% TPR than in the unaugmented case.

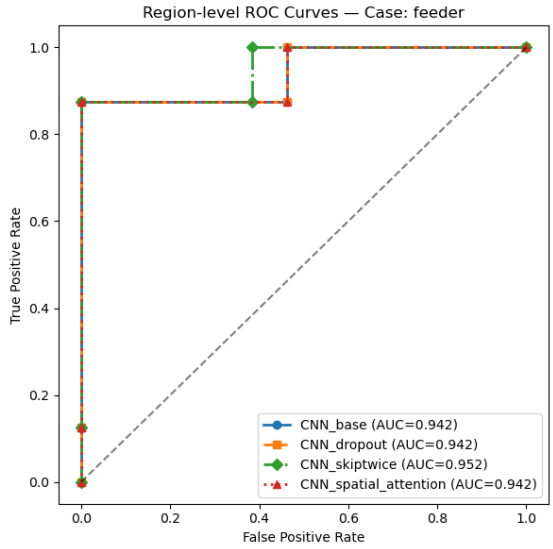


(a) Frame-by-frame ROC for the feeder set. General_aug.

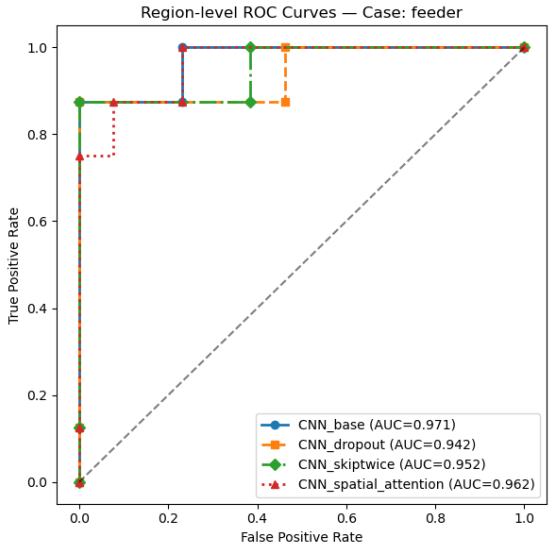


(b) Frame-by-frame ROC for the feeder set, Lighting_aug.

Figure 4.16: Frame-by-frame ROC curves from training the models with the two augmentation strategies. Feeder set.



(a) Case-by-case ROC for the feeder set. General_aug.



(b) Case-by-case ROC for the feeder set. Lighting_aug.

Figure 4.17: Case-by-case ROC curves from training the models with the two augmentation strategies. Feeder set.

4.2.2 Augmented can set

In the results from the can set we saw the severe effects of large lighting on the performance of the models. Therefore, this section outlines how the two augmentation techniques affect this problem.
We will first look at the test set where the ceiling lamp was turned off. The PSNR scores from the unaugmented base model can be found in Figure 4.12, and the same scores using the two augmentations with the base model can be found in Figures 4.18 and 4.19 below. In both augmentation cases we can see that the PSNR scores from the test set are still significantly lower than the validation scores, meaning the problem is not solved in either case. However, the augmentations did serve to move the test scores up by a tiny bit, while also stabilizing the normal scores slightly, more so in the General_aug case than in the Lighting_aug. This does come at the cost of lowering the average scores of the validation set by approximately 2 dB in the Lighting_aug case and 3 dB for General_aug. Whether this change represents any actual improvement when it comes to anomaly detection is unclear.
It is very possible that the augmentation applied was not strong enough to mimic the change caused by the lamp turning off, so further investigation into this problem and the augmentation strategies would need to be done to draw any definite conclusions.

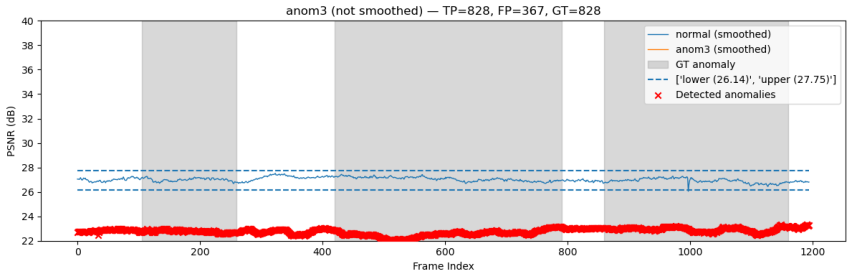


Figure 4.18: Base model trained with General_aug, deployed on the differently lit can test set.

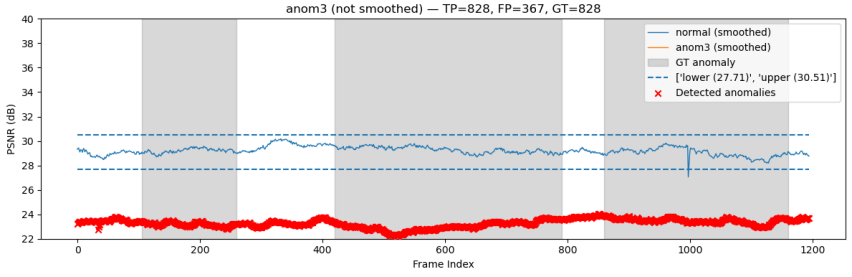
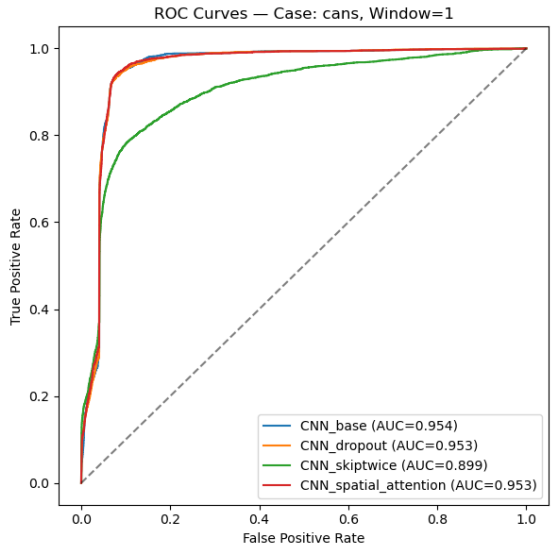


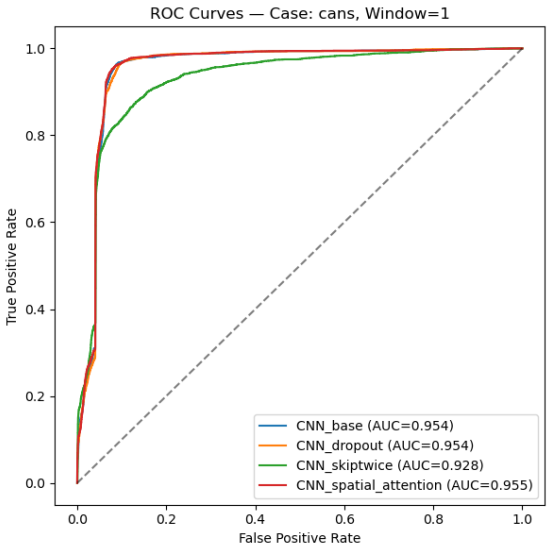
Figure 4.19: Base model trained with Lighting_aug, deployed on the differently lit can test set.

The ROC curves from the deployment of the two augmented versions of the models on all test sets in the can dataset can be found in Figures 4.20 and 4.21. It is clear from both figures that there was no significant general increase in performance

from using the augmentations in this case. The performance of the skiptwice model was slightly worsened in the frame-by-frame graphs, and the early performances of the other models were slightly worsened or equal to the unaugmented versions in Figure 4.13. Since the problem of the lighting conditions was not solved by the augmentation, it makes sense that there is no improvement shown. The results from the other test sets that did not include lighting changes will not be improved by the augmentation, meaning it will only serve to slightly worsen the reconstructions without any discernible increase in robustness. It might be worthwhile to record and include scenes with more varied degrees of lighting changes in the future, in order to more accurately map how the augmentations affect the robustness.

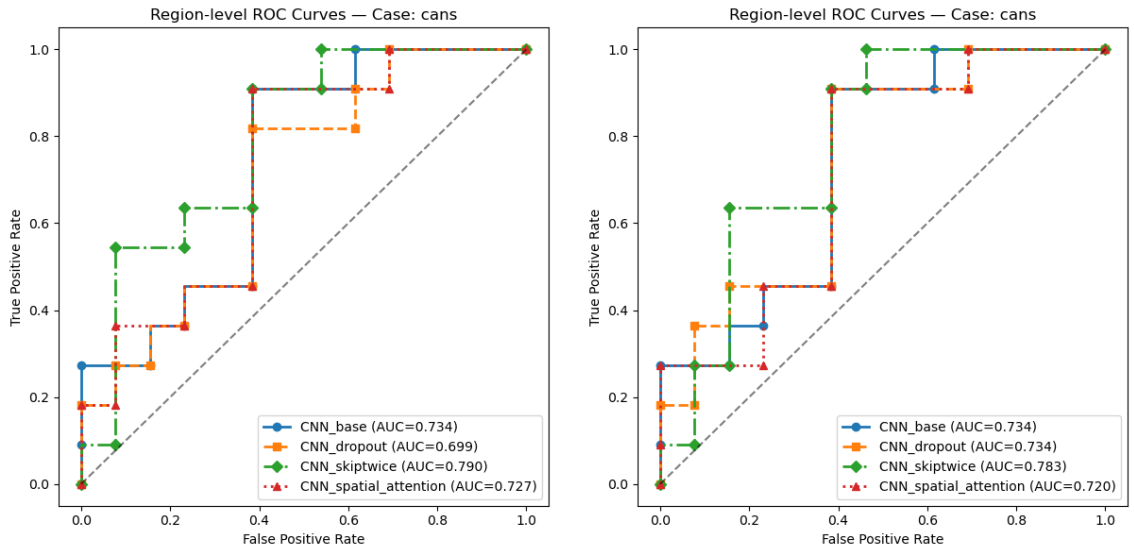


(a) Frame-by-frame ROC for the can set, General_aug



(b) Frame-by-frame ROC for the can set, Lighting_aug

Figure 4.20: Frame-by-frame ROC curves from training the models with the two augmentation strategies. Can set.



(a) Case-by-case ROC for the can set, General_aug

(b) Case-by-case ROC for the can set, Lighting_aug

Figure 4.21: Case-by-case ROC curves from training the models with the two augmentation strategies. Can set.

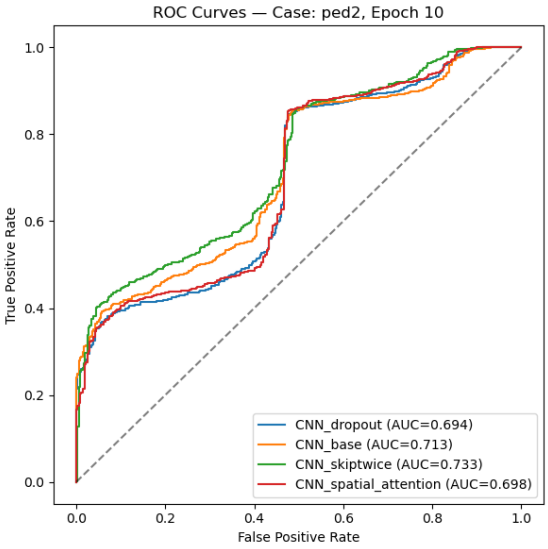
4.3 UCSD Ped2 Dataset

In this section, the ROC curves from deploying the four models on the UCSD Ped2 dataset [2] will be presented. Some still frames from the dataset can be found in A.4(see Appendix A.1), where the top three images are normal training frames, and the bottom three are examples of typical anomalies. The anomalies in the test sets consist of 12 different scenarios in which a skater, a biker, or a golf cart enters or traverses the frame. The ROC curves for each model can be found in Figure 4.22 below. In the construction of the left graph the models were all trained for the usual 10 epochs, and in the right graph this number was increased to 40. This result is included because, unlike for the factory setting datasets, the increased training time did improve the performance significantly for one of the models, being the skiptwice model.

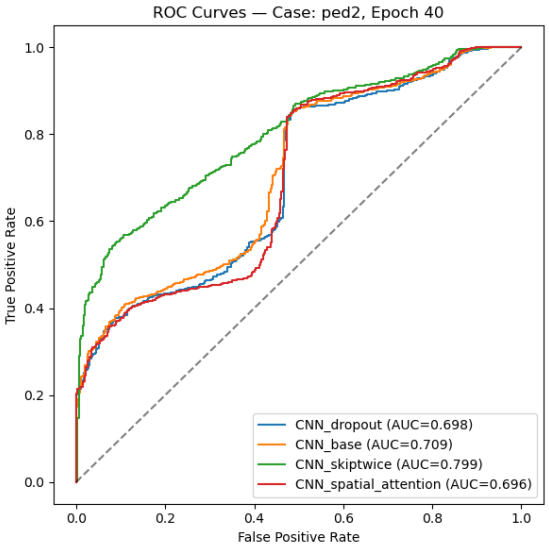
Recent state-of-the-art approaches reach AUC scores above 0.99 on Ped2 [43, 44]. Our models' lower scores are not surprising, since they were specifically designed to be small and flexible rather than optimized for maximum accuracy. Because the pedestrians in the scenes occupy rather small sections of the frame, sharp edge reconstruction is required. However, our predictions exhibit a slight blur around each person. Since each blurred pedestrian contributes to the overall reconstruction error, normal frames containing many people will accumulate enough error to be flagged as anomalous before the truly anomalous frames with fewer people.

Addressing this would likely require adding more spatial focus or sharper decoding layers to the model. This idea is exemplified by the increased performance of the skiptwice model, especially in the 40-epoch case. The skiptwice model has consistently shown an increased spatial focus and a reduced temporal sensitivity when

deployed on the other datasets, as evidenced by its higher PSNR on slow or static sequences and large drops during sudden movement. This could make it more suited for this specific dataset, and is likely what is giving it the edge here.



(a) Frame-by-frame ROC for the Ped2 dataset, 10 epochs.



(b) Frame-by-frame ROC for the Ped2 dataset, 40 epochs.

Figure 4.22: Frame-by-frame ROC curves from deployment on UCSD Ped2. No data augmentation was used during training.

5
Discussion

Some of the most interesting results from testing the models on the different datasets were the impact of and compromise between temporal understanding and spatial focus in the models, as well as the impact of unwanted data artifacts such as lighting changes or unbalanced training sets, and how data augmentation interacts with this problem. To summarize the findings, it seemed that the attention model or the base model performed best for the foam set and the feeder set, due to the more sporadic motion in those cases. For the can and Ped2 sets, the skiptwice model performed best, likely due to the slower and more consistent movements in those datasets. Both augmentation strategies improved the performances in most cases, but often removed too much of the motion focus when used in combination with the skiptwice model.
The models were quite quick to train, taking around 20 minutes for each new case. This can mostly be attributed to the use of the pretrained encoder, which cuts down the number of trainable weights to around 2.6 million, which is orders of magnitude lower than many other approaches, such as transformers and 3D CNNs.

5.1 Spatio-temporal tradeoff

In this section, the temporal/spatial relationship will be outlined and discussed. It was clear in all four datasets that the skiptwice model generally returned higher-quality predictions, both for the anomalous and non-anomalous frames, which can be both a good and a bad thing. However, it seems to have come at the cost of a lower focus on motion modeling. This was most evident in the feeder and foam datasets, which have more sporadic movement than the can and Ped2 scenes. In the foam set, we saw larger drops during starts and stops of motion cycles compared to the other models. These motion drops were even clearer in the feeder set, where any movement in the conveyor would cause the PSNR scores to drop by up to 7 dB (see Figure 4.9). Comparing this to the results from the base model (Figure 4.8), where motion causes only a wide and shallow drop, the difference is clear. In more slow-moving consistent processes, this increased spatial focus from the skipping model may be a significant improvement, while cases with jerky and inconsistent movement seem to require the heavier temporal context to keep the prediction scores consistent during normal evaluation. Therefore, this trade-off between motion modeling and a sharper spatial focus appears to be one of the customization options that would benefit most from per-application tuning by the user. Based on their input, one could increase or decrease the number and depth of the skip connections

accordingly. This would require more testing to confirm, as there is no guarantee that increasing the number of connections or making them more shallow would only affect this spatial/temporal relationship. These changes might bring with it other unwanted artifacts and behavior, or reduce robustness. Another strategy that could improve the stability in these rarely moving jerky cases such as the feeder set is balancing the training data. As it stands, the training data shows the process in its raw form, which means that the training frames in the feeder dataset are heavily skewed toward showing the conveyor while it's idle, and motion sequences become very scarce. This means that the models will be significantly more rewarded for focusing on static prediction, as that will decrease the loss for many more sequences compared to predicting the scarce moving frames. Balancing the dataset by filtering out long sequences of no movement might therefore force the skipping models to divide their focus more evenly across moving and static scenes. This balancing could, for example, be done beforehand using the raw frames by simply subtracting adjacent frames and removing a proportion of frames whose difference is below some threshold. Another strategy could be to cluster the encoded combined feature maps into different sets based on similarity, and then balancing the training by picking the same number of sequences from each set. However, this could introduce less predictability into the training and may require careful monitoring of the clustering algorithm used.
The other notable effect of the spatial/temporal focus could also be seen in the feeder set, where the PSNR scores for different normal static configurations fluctuated heavily. This was most evident in the models without the skip connections and therefore with lower spatial focus. This problem may be alleviated slightly by implementing one skip connection between one of the deeper layers, which could decrease the fluctuations while keeping the motion drops at an acceptable level. The dataset balancing strategy might also be effective here. In its current state, configurations that appear only for a short time will receive less attention during training, which might be a contributing factor to the heavy fluctuations. The balancing won't explicitly increase the number of seen patterns during training, but it might still help divide the focus better across the ones that do show up.

5.2 Data augmentation effects

Moving on to the effects of the data augmentation, two versions of augmentation were tested on all datasets, and they both showed some promise.

5.2.1 Feeder dataset

The effects were most visible in the feeder set, where both augmentations significantly improved the performances of the base and spatial attention models. The dropout model responded well to the general augmentation, but was slightly degraded by the lighting augmentation. As for the skiptwice model, its performance significantly deteriorated from both versions.
The general augmentation did achieve the desired effect of reducing the specific pattern focus mentioned earlier, but it also caused the average prediction quality

to go down slightly by increasing blur. This isn't necessarily a bad thing, since what is most important in our case is the contrast between anomalous and normal sequences. The lower PSNR scores could mean that small spatial anomalies may be harder to detect. However, it is difficult to draw definitive conclusions whether this is actually the case or not, as the augmentation also served to significantly lower the fluctuations of normal predictions. This means that a tighter anomaly threshold can be used, which could cancel out the effect of the increased general blurriness of the reconstructions.
The lighting augmentation had similar effects to the general augmentation in the feeder set, but to a lesser degree. This appears logical: It may have contributed to lowering the pixel-by-pixel overtraining, but because it did not apply any spatial perturbation, it was not as effective as the general augmentation in reducing the pattern-recognizing tendencies.

5.2.2 Can dataset

In the can dataset, neither augmentation strategy was effective in handling the large lighting change caused by the ceiling lamp. This does not rule out the potential benefit of more targeted augmentations. The augmentations tested may simply have been too mild or too simplistic to cover the effects of the lamp, so stronger or more realistic lighting transforms are worth exploring. However, in these heavily perturbed cases, augmentation alone may not suffice and risks over-smoothing the reconstructions. Other strategies that could be used include applying some sort of contrast normalization to the frames beforehand, such as in [45], or adding a small CNN block that learns to correct lighting changes in the frames before feeding them to the main network. Another method that was considered early on in the report is training partly or entirely on optical flow maps instead of on the raw frames, such as in [46]. An optical flow map consists of the motion vectors of each pixel position between two images, meaning it is completely independent of lighting conditions in the ideal case. However, it could increase the training and inference times significantly depending on the algorithm used. One would also lose the benefits of ResNet, since at the time of writing there is no pre-trained feature extractor for optical flow data.

5.3 Adaptive thresholding and error metrics

Other topics that would warrant investigation in future work are the thresholding function and error calculations. The thresholding function was not explored thoroughly in this report, but it will be impactful in actual application scenarios. While the augmentation techniques may alleviate some of the impacts of drift during inference, they will probably not remove them completely. The long-term average of the PSNR scores are likely to shift to some degree during longer applications, which means the threshold would need to adapt to this shift to avoid false positives. There are many ways to do this, but as it would effectively be a form of unsupervised learning, it would require the process to be normal a majority of the time. Some strategies involving training lighter ML models on the normal PSNR

scores were also considered. However, due to the desired generality of the models and detection strategies, these were deemed too unstable and unpredictable if not monitored properly.
As for the error metric, it may also be improved upon. Given that the PSNR approach is pixel-by-pixel based, it could over-sensitize the detector to lighting differences. This could possibly be alleviated by simply apply a low-pass filter beforehand, so only differences exceeding a set threshold contribute to the final score. The detection process may also benefit from switching out the PSNR score altogether, possibly for a more edge-focused or object-focused alternative.

6
Conclusion

This study has shown that lightweight predictive autoencoders can detect a variety of industrial video anomalies, but that their success depends heavily on the balance between spatial detail and temporal focus in the models, the composition of the training data, and the choice of pre- and post-processing. The skip-twice model delivered the sharpest reconstructions on static or slowly changing scenes, yet suffered pronounced error spikes during fast and jerky movement. The other model variations tested performed very similarly in all cases. Slight lighting and position augmentations during training helped counter over-fitting to static patterns, which made the most difference in the feeder dataset. However, they could not solve the heavy illumination shifts in the can dataset, where more sophisticated normalization or learned correction appears necessary. When benchmarked on UCSD Ped2, the architectures performed moderately well with AUC scores of ≈ 0.7. The skiptwice model showed significant improvement when the training time was increased to 40 epochs, reaching a AUC of 0.799 . This was likely due to its comparatively increased spatial focus compared to the other model versions.
Put together, these findings suggest several areas that could be worthy of deeper investigation. It would be interesting to explore adaptive thresholding methods that adjust to the likely gradual PSNR drift over long deployments, and to test stronger and more advanced data augmentation techniques, such as simulated shadows, directional light or more contrast changes, to better mimic real-world lighting variations. Combining prediction-error scores with complementary data metrics such as optical flow methods, or employing a more specialized error function may help reduce false positives in more complex or noisy settings. By investigating these directions further, one can hope to develop an anomaly detection system that is both accurate and robust across many possible applications in the production industry.

Bibliography

[1] J. Riebesell, "Mit license for latex-generated image," Licensed under the MIT License, 2021, full text available at https://opensource.org/licenses/MIT.

[2] University of California, San Diego, "Ucsd anomaly detection dataset (ped2)," https://paperswithcode.com/dataset/ucsd, accessed: 2025-05-12.

[3] E. Duman and O. Erdem, "Anomaly detection in videos using optical flow and convolutional autoencoder," *IEEE Access*, vol. 7, pp. 183 914 – 183 923, 12 2019.

[4] Y. S. Chong and Y. H. Tay, "Abnormal event detection in videos using spatiotemporal autoencoder," 2017. [Online]. Available: https://arxiv.org/abs/1701.01546

[5] Eye at Production, "Eye at Production System," https://eyeatproduction.com/en/, 2025, accessed: 2025-05-19.

[6] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection: A survey," *ACM Comput. Surv.*, vol. 41, no. 3, Jul. 2009. [Online]. Available: https://doi.org/10.1145/1541880.1541882

[7] D. M. Tax and R. P. Duin, "Support vector data description," *Machine learning*, vol. 54, no. 1, pp. 45–66, 2004.

[8] N. Arunraj, R. Hable, M. Fernandes, K. Leidl, and M. Heigl, "Comparison of supervised, semi-supervised and unsupervised learning methods in network intrusion detection system (nids) application," *Anwendungen und Konzepte der Wirtschaftsinformatik*, vol. 6, pp. 10–19, 2017.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] X. Li, J. Xue, and Y. Tian, "Efficient video compression for deep neural network analysis: An overview," in *IEEE Transactions on Multimedia*, 2018.

[11] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Laurent, "Review and evaluation of commonly-implemented background subtraction algorithms," *International Journal of Computer Vision*, vol. 90, pp. 59–71, 2010.

[12] A. Graves, A. rahman Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013. [Online]. Available: https://arxiv.org/abs/1303.5778

[13] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," *NIPS*, pp. 3104–3112, 2014.

[14] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," 2015. [Online]. Available: https://arxiv.org/abs/1508.01211

[15] N. Azizi, N. Wandel, and S. Behnke, "Complex valued gated auto-encoder for video frame prediction," 2019. [Online]. Available: https://arxiv.org/abs/1903.03336

[16] Y. S. Chong and Y. H. Tay, "Abnormal event detection in videos using spatiotemporal autoencoder," in *Advances in Neural Networks - ISNN 2017*, F. Cong, A. Leung, and Q. Wei, Eds. Cham: Springer International Publishing, 2017, pp. 189–196.

[17] V. Nair and G. Hinton, "Rectified linear units improve restricted boltzmann machines vinod nair," vol. 27, 06 2010, pp. 807–814.

[18] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

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

[20] D. Scherer, A. Müller, and S. Behnke, "Evaluation of pooling operations in convolutional architectures for object recognition," in *Artificial Neural Networks – ICANN 2010*, K. Diamantaras, W. Duch, and L. S. Iliadis, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101.

[21] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE Transactions on Image Processing*, vol. 13, no. 4, pp. 600–612, 2004.

[22] J. Johnson, A. Alahi, and L. Fei-Fei, "Perceptual losses for real-time style transfer and super-resolution," 2016. [Online]. Available: https://arxiv.org/abs/1603.08155

[23] M. Mathieu, C. Couprie, and Y. LeCun, "Deep multi-scale video prediction beyond mean square error," 2016. [Online]. Available: https://arxiv.org/abs/1511.05440

[24] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct 1986. [Online]. Available: https://doi.org/10.1038/323533a0

[25] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff, "Lstm-based encoder-decoder for multi-sensor anomaly detection," 2016. [Online]. Available: https://arxiv.org/abs/1607.00148

[26] A. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," in *2010 20th International Conference on Pattern Recognition*, 2010, pp. 2366–2369.

[27] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016516841300515X

[28] E. R. H. P. Isaac and A. Sharma, "Adaptive Thresholding Heuristic for KPI Anomaly Detection," 2023. [Online]. Available: https://arxiv.org/abs/2308.10504

[29] S. Ahmad and S. Purdy, "Real-time anomaly detection for streaming analytics," pp. 4–6, 07 2016.

[30] D. Reynolds, *Gaussian Mixture Models*. Boston, MA: Springer US, 2015, pp. 827–832. [Online]. Available: https://doi.org/10.1007/978-1-4899-7488-4_196

[31] T. Ale, N.-J. Schlegel, and V. P. Janeja, "Harnessing feature clustering for enhanced anomaly detection with variational autoencoder and dynamic threshold," 2024. [Online]. Available: https://arxiv.org/abs/2407.10042

[32] F. T. Liu, K. M. Ting, and Z.-H. Zhou, "Isolation forest," in *Proceedings of the 2008 IEEE International Conference on Data Mining (ICDM)*, 2008, pp. 413–422.

[33] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[34] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Miami, FL, USA, 2009, pp. 248–255.

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

[36] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2013. [Online]. Available: https://arxiv.org/abs/1211.5063

[37] V. Mahadevan, W. Li, V. Bhalodia, and N. Vasconcelos, "Anomaly detection in crowded scenes," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2010, pp. 1975–1981.

[38] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, no. 146, p. 10, 2006.

[39] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[40] P. Contributors, "Torchvision: Datasets, transforms, and models for computer vision," https://github.com/pytorch/vision, 2025.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[42] T. Fawcett, "An introduction to roc analysis," *Pattern Recognition Letters*, vol. 27, no. 8, pp. 861–874, 2006, rOC Analysis in Pattern Recognition. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016786550500303X

[43] W. Liu, H. Chang, B. Ma, S. Shan, and X. Chen, "Diversity-measurable anomaly detection," 2023. [Online]. Available: https://arxiv.org/abs/2303.05047

[44] J. Micorek, H. Possegger, D. Narnhofer, H. Bischof, and M. Kozinski, "Mulde: Multiscale log-density estimation via denoising score matching for video anomaly detection," 2024. [Online]. Available: https://arxiv.org/abs/2403.14497

[45] P. Musa, F. Rafi, and M. Lamsani, "A review: Contrast-limited adaptive histogram equalization (clahe) methods to help the application of face recognition," 10 2018, pp. 1–6.

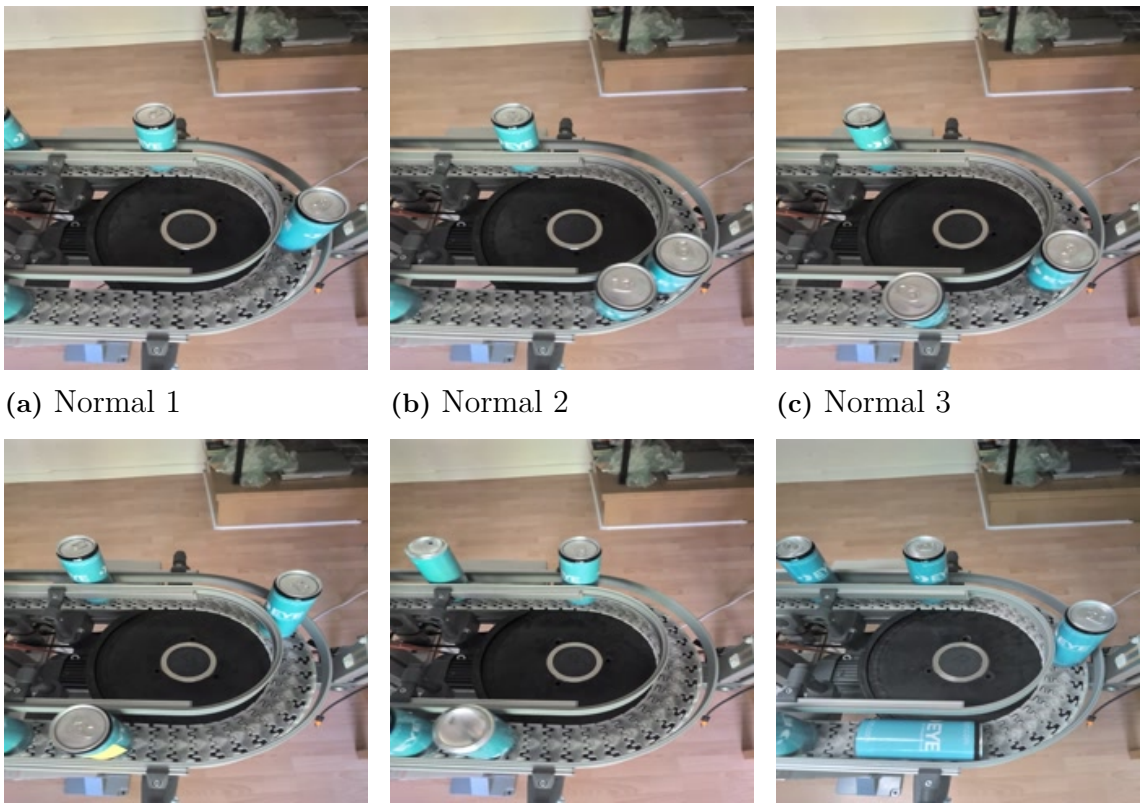
[46] E. Duman and O. A. Erdem, "Anomaly detection in videos using optical flow and convolutional autoencoder," *IEEE Access*, vol. 7, pp. 183 914–183 923, 2019.

A

Appendix 1

A.1 Datasets

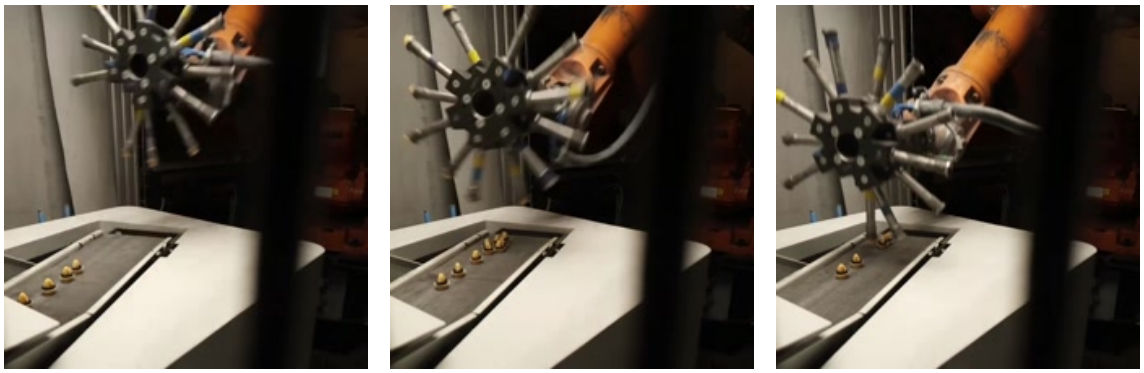
Below are some sample frames from the three datasets used for evaluation, as well as some from the UCSD ped2 dataset used for benchmarking. For the feeder set and the foam set, the training and evaluation was performed on ROI frames, while the can set and the ped2 set were used directly.



(a) Normal 1 **(b)** Normal 2 **(c)** Normal 3

(d) Anomaly, post-it note **(e)** Anomaly, upside-down **(f)** Anomaly, fallen can

Figure A.1: Can set: images (a)-(c) are normal frames, images (d)-(f) are anomalous frames.



(a) Normal 1

(b) Normal 2

(c) Normal 3

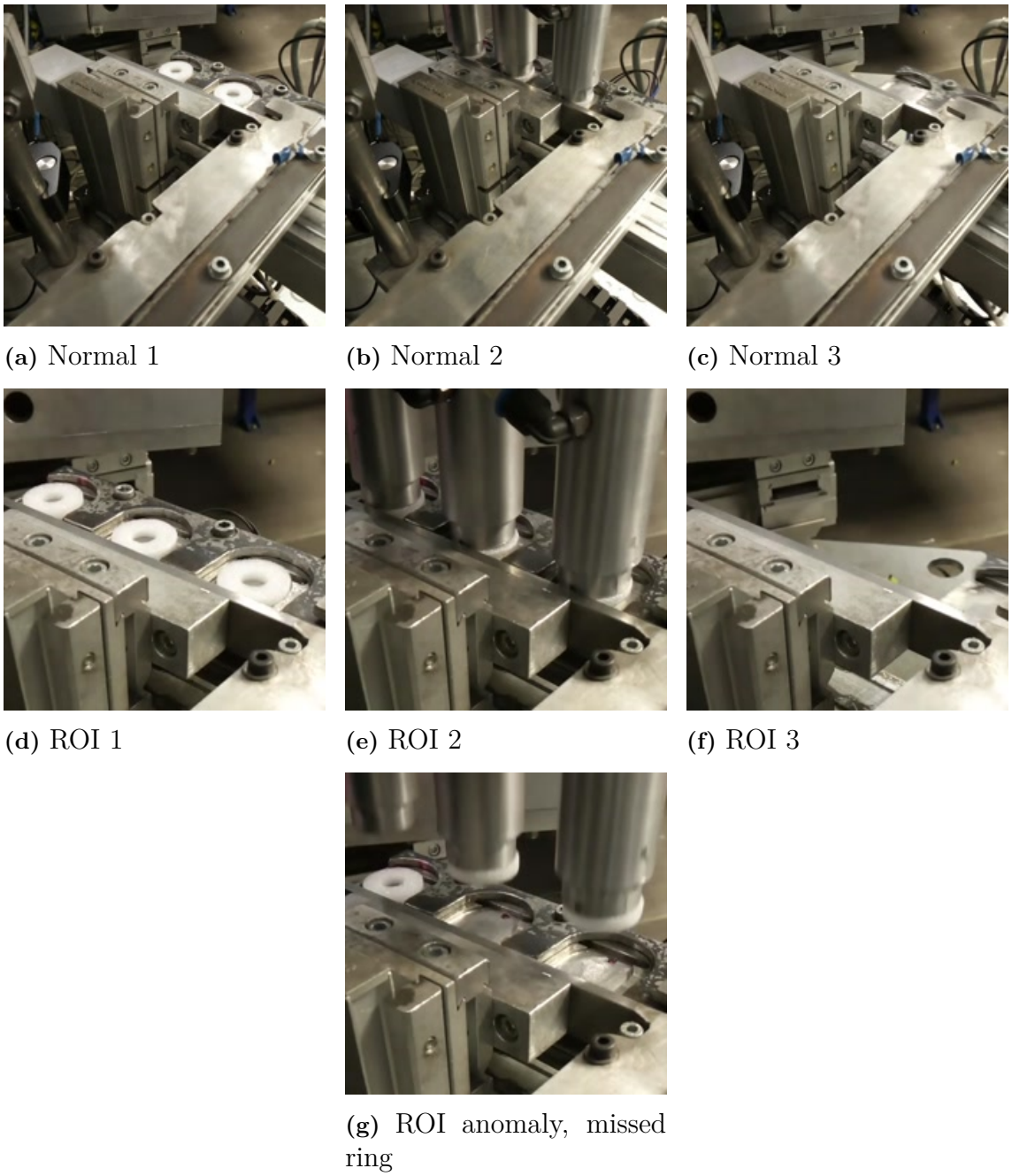
(d) ROI 1

(e) ROI 2

(f) ROI 3

(g) ROI anomaly, clumped caps

Figure A.2: Feeder set: images (a)-(c) are normal full-sized frames, (d)-(f) are normal ROI frames, and (g) is a typical anomalous frame.



(a) Normal 1 **(b)** Normal 2 **(c)** Normal 3

(d) ROI 1 **(e)** ROI 2 **(f)** ROI 3

(g) ROI anomaly, missed ring

Figure A.3: Foam set: images (a)-(c) are normal full-sized frames, (d)-(f) are normal ROI frames, and (g) is a typical anomalous frame.

(a) Normal 1

(b) Normal 2

(c) Normal 3

(d) Anomaly, biker

(e) Anomaly, golf cart

(f) Anomaly, skateboarder and biker

Figure A.4: UCSD ped2 set [2]: images (a)-(c) are normal frames, images (d)-(f) are anomalous frames.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

CHALMERS
UNIVERSITY OF TECHNOLOGY