Anomaly detection from video streams in reconnaissance missions

Group 3

December 22, 2022

1 Introduction

This chapter will give an introduction to the domain of the problem. Additionally, it will provide an explicit description of the problem including delimitation and possible solutions.

1.1 Problem domain

The customer Combitech works with the project WARA Public Safety (PS) which is a project within Wallenberg Autonomous Systems and Software Program (WASP). The objective of the project is to provide a research arena for public safety, enabling large scale demonstration environments. In terms of the research areas, there is quite a large diversity including (but not limited to) [20]:

1. Image processing.
2. 3D-tracking.
3. Video compression.

One application of importance, which is related to all of the research areas above, is autonomous search-and-rescue for usage close to the sea. Due to the diverse nature of the task, there is a need for cross-functionality of the tool used to solve it. One of the solutions is by the usage of drones, in principle, the drones are used to monitor an area (for example an archipelago) by making periodic autonomous flights, and upon the discovery of an anomaly (for example a missing person) it reports to some central unit that sends out a manned mission to investigate. It is the task of anomaly detection from drone footage that is to be investigated.

1.2 Problem description

Combitech has tasked us with providing one or more AI models that may be used to detect anomalies in video streams. In particular, the video streams will be provided from drone footage of a coastal region with an almost exclusively native environment (for example Gränsö in Västervik), generated from the monitoring missions performed by the drones. The definition of an anomaly for this purpose is very contextual, that is, something which changes from one observation to another. However, since the environment is exclusively native, any man-made objects are to be considered anomalies in this instance, therefore a list of possible anomalies will include: a person, a car, a balloon, and other man-made objects. However, data labeled with such anomalies does not exist and thus creating such data is considered part of the problem.

1.3 Delimitation

The task of anomaly detection, especially for video, is not trivial and in order to make it feasible for this project, some delimitions needs to be done. Firstly, in accordance with the directions of Combitech, it has been chosen to start by considering anomaly detection for images, as this can (with a fair bit of work) be further developed into videos in a natural manner. Secondly, due to a lack of data, the data used for the model is not restricted to drone footage of the archipelago in Västervik.

1.4 Possible solutions

When considering possible solutions for the problem of anomaly detection in video streams, the main aspects for consideration were:

1. Performance.
2. Complexity of implementation.
3. Data requirements.
4. Existing literature.

Judging from the above mentioned aspects, the following methods were investigated further.

1.4.1 CNN and outlier detection algorithm

Note: Outlier/Anomaly is used as synonyms in this text and represents the same thing.

For the problem of anomaly detection in images, a combination of deep feature extraction and an unsupervised classification method could be used. For this a pretrained *Convolutional Neural Network* (CNN) could be used for feature extraction and an outlier detection algorithm, like *One-Class Support Vector Machine* (OCSVM), for unsupervised classification [3].

1.4.2 Transformers

A transformer in machine learning is in general a model which maps sequences-to-sequences, and was firstly used in natural language processing. However, due to the temporal nature of videos, they can be applied for this purpose as well. In particular, tubelets (which is a fixed region of a set of overlapping images within a video sequence) can be used as input to the transformer in order to capture long-term spatial-temporal dependencies (eg. anomalies contextualised by time) [12].

1.4.3 Autoencoder

An autoencoder is a *neural network* which learns to recreate the input by learning a representation of the underlying set of data the input was taken from. Anomaly detection can be performed by combination of comparing the input and output of the neural network as well by finding the probability that the input was drawn from the learned representation (which should be as close to the real underlying set of data as possible) [1].

1.5 Chosen solutions

From the possible solutions two were selected for further exploration.

1. CNN and an outlier detection algorithm was selected, since the problem domain is comparable to the problem domain described in [3] which uses a CNN and OCSVM, and because the results presented in [3] show good results.
2. Transformers was not selected despite good performance [12] since the complexity was thought to become a hindrance to development.
3. Autoencoders was selected, since the experimental results show good performance, and because the model complexity seemed simple when compared to anomaly detection with transformers [1].

1.6 Hypothesis

The use of a pretrained CNN and a outlier detection algorithm is expected to produce good results for the problem of anomaly detection in images. However it is expected that given a pretrained CNN used for feature extraction, some anomalies may be missed due to the CNN not being trained to extract features that capture the difference between anomalies and non-anomalies well.

Autoencoders appears more complex than the CNN and OCSVM method and thus is expected to require more development. Nevertheless it is expected to perform good on the task of anomaly detection in images.

2 Method

This chapter gives an introduction to both the datasets used, as well as the algorithms used for solving the problem. Additionally, it will provide the necessary context needed to understand the algorithms on a fundamental level.

2.1 Datasets

The datasets used were constructed based on a dataset acquired from University of California San Diego (UCSD). The dataset provided by UCSD contains video-sequences from a stationary camera overlooking a pedestrian walkway. In order to use the dataset for anomaly detection in images, the video-sequences were partitioned into sets of images. UCSD provides images from two different perspectives and the dataset is thus partitioned into two subsets, Peds1 and Peds2, each containing images from a specific perspective [19].

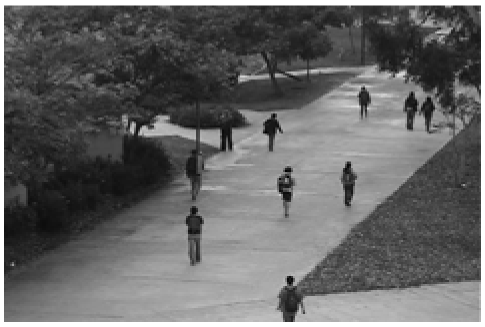


Figure 1: Image from the pedestrian dataset (Peds1 subset).

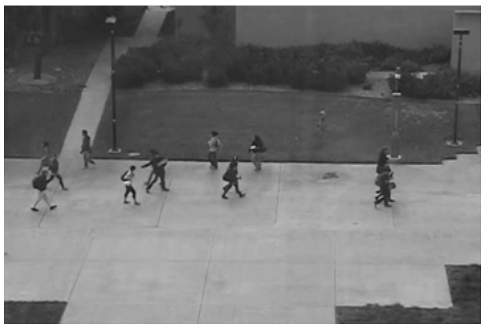
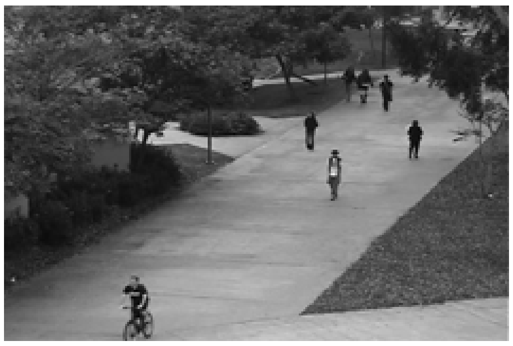
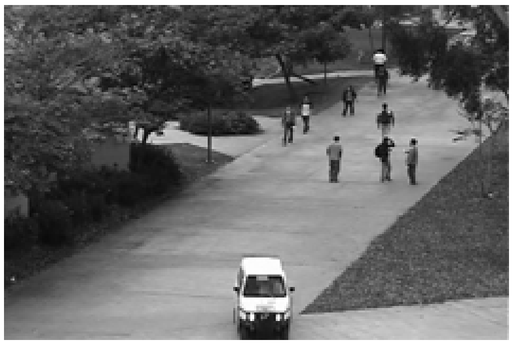


Figure 2: Image from the pedestrian dataset (Peds2 subset).

As the stationary cameras are located at an elevation towards the walkway, they provide a good alternative to videos captured by a drone. As to be expected, the walkways mostly contains pedestrians, thus a pedestrian on the walkway does not classify as an anomaly. The anomalies are instead all other objects, such as bikers and vehicles. Since anomalies are not defined in advance, the contextual definition of an anomaly is satisfied. Examples of anomalies in the datasets are shown below.



(a) Anomaly (bike) from the pedestrian dataset.



(b) Anomaly (van) from the pedestrian dataset.

Figure 3: Anomalies

Based on the dataset obtained from UCSD two new datasets were created as follows:

- **UCSD Baseline:** Training images are comprised of training images from the Peds1 subset, these do not contain any anomalies. For testing and validation, images containing large anomalies (cars and vans) were manually selected from the Peds1 subset. Testing and validation also require some normal images and these are selected from the Peds1 subset as well. This dataset provides a good baseline for anomaly detection in images and is therefore used to test model performance.
- **UCSD Combined:** This dataset is an extension to the UCSD Baseline dataset and thus contains all the images that the UCSD Baseline dataset contains. This dataset further provides all the training images from the Peds2 subset. Testing and validation images from the Peds2 subset were also added to this dataset with no restriction on the size or type of anomaly. This dataset provides a framework for testing the generalization properties of the model as the dataset contains images from multiple angles.

2.2 Preliminary

Some background knowledge of neural networks is assumed from the reader, however, in order to fully grasp how neural networks are used in this context, there is a need for additional knowledge.

2.2.1 Function Approximation

Neural networks can be used in the context of function approximation, and a motivation for this is given by theorem 1 of [11], which is shown below.

Theorem 2.1.
Let $(\mathbb{R}^n, \mathcal{M}, \mu)$ be a measure space. It holds that for every $\epsilon > 0$ and every $f \in L^p(\mathbb{R}^n, \mathbb{R}^m)$, where $p \in [1, \infty)$, there exists a neural network F, with ReLU as the activation function, such that

$$\int_{\mathbb{R}^n} \|f(x) - F(x)\|^p d\mu(x) < \epsilon$$

In conclusion of Theorem 2.1, any function of practical use can be modelled using a neural network with a ReLU activation function.

2.2.2 Conditional Distribution

Neural networks can also be used in the context of regression (or classification). Disregarding dimensions of the components, consider a sample of predictors $(x_i)_{i\in I}$ and corresponding responses $(y_i)_{i\in I}$, where it is assumed that

$$y_i = F(x_i; W) + \varepsilon$$

for some weights W and error $\varepsilon \sim N(0, \sigma^2)$. Under the assumption that the neural network is trained (thus W being fixed), the responses are stochastic, and have the following conditional distribution

$$Y \mid X, W \sim N(F(X; W), \sigma^2)$$

Under this probabilistic framework, the objective is to maximize the likelihood $\mathbb{P}[X, Y \mid W]$, which corresponds to minimizing the following expression [2]

$$\frac{1}{2} \sum_{i\in I} \|F(x_i; W) - y_i\|^2$$

2.3 CNN and outlier detection algorithm

This method uses a *Convolutional Neural Network* (CNN) together with a *outlier detection algorithm.* The CNN is used for feature extraction whilst the outlier detection algorithm is used to classify an image as containing an anomaly or not.

2.3.1 CNN

A CNN is a neural network that specializes in machine learning that analyze images.

A CNN generally contains convolutions layers and pooling layers. Shortly explained, a convolutional layer uses a kernel which "sweeps" over the input image and perform convolutions which it will send the results of to the next layer.

A pooling layer summarizes information by e.g. taking the maximum of four values to represent those four values in the next layers.

The last layers of a CNN are usually fully dense layers where the CNN uses the features it has learned to return values that can be used for prediction.

2.3.2 GoogLeNet

"GoogLeNet" is a variant of an *Inception network* which is a type of CNN architecture that uses inception layers. An inception layer is a layer in a CNN where multiple different types of convolutions are done and also pooling. The idea is that instead of having to choose one specific convolution or to do pooling, all is done and then the network can learn how to best handle the resulting information which is concatinated as output from the layer. For feature extraction, a pre-trained GoogLeNet is used. GoogLeNet has 1024 features in the last feature layer, which are extracted and used in the next step [16].

2.3.3 Normalization

After the CNN and before the outlier detection algorithm, the normalized features are extracted from the CNN using min-max normalization, which can be described by

$$x_{i,norm} = \frac{x_i - max(x)}{max(x) - min(x)} \tag{1}$$

2.3.4 EfficientNet

EfficientNet is a CNN model that proposes scaling up the architecture in an efficient way [17]. The upscaling process is often done by increasing the network horizontally or vertically to increase precision [5], [6]. It is also possible to increase the image size for better precision [15], [7], [9]. *EfficientNet* makes use of a technique that is called compound coefficient to scale up the model in an efficient way. This is done by scaling up every dimension of the network and the image based on a set of scaling coefficients. There exist seven models of *EfficientNet*, where *Efficientnet_B7* is the version that is most up-scaled and has the highest accuracy and training time [17].

Efficientnet_V2 is a more updated version of *EfficientNet* and is both faster and makes more efficient use of its parameters than its predecessor [18]. This is possible by using a combination of training-aware neural architecture search (NAS) and scaling. NAS is an automatic way to design a neural network [22]. *Efficientnet_V2* shows state-of-the art performance in comparison to other models on ImageNet ILSVRC2012 [1]. *Efficientnet_V2_m/l* are more up-scaled versions than *Efficientnet_V2_s* [18].

[1]https://www.image-net.org/challenges/LSVRC/2012/

2.3.5 Outlier detection algorithms

An outlier/anomaly detection algorithm is used to detect outliers in data. To find the best possible model, three different outlier detection algorithms were used and evaluated. Even though some of the models can be trained with both anomaly and non-anomaly data points, only non-anomaly data was used in order to not make the models biased towards certain kinds of anomalies.

2.3.6 One-class support vector machine

One-class support vector machine (OCSVM) is a outlier detection algorithm. An OCSVM specializes in novelty detection. It only trains on data that is considered "normal" and will then classify data as either belonging to the "normal" kind of data or being abnormal. In this case, the OCSVM will classify the normalized feature vector extracted from GoogLeNet as abnormal or normal. A OCSVM works by finding a boundary to enclose non-outliers and then, if a new point is inside this boundary, it is considered as a non-outlier and if it is outside, it is conisdered an outlier [13].

2.3.7 Isolation Forest

The *Isolation forest* algorithm works by building a multiple isolation trees[10]. For each tree, the training data gets partitioned until each data sample is isolated and the main idea is that when using the tree on an outlier, the path should be much shorter since outliers are few and more distinguishable than non-outliers [10].

2.3.8 Local Outlier Factor

Local outlier factor (LOF) finds outliers by finding what data points has a significantly lower density than it's neighbors. So in simple terms, the algorithm calculates a density for each data point and then compares this density to the k nearest neighbors. Any data point that has a significantly lower density than its neighbors is considered an outlier [4].

2.3.9 Evaluation Metrics

To compare different model architectures the F1-metric was used. The F1-metric is the harmonic mean of the *Recall* and *Precision* [21].

The following terminology is used. A *True positive (TP)* is a correctly classified anomaly, *true negative (TN)* is a correctly classified non-anomaly image.

$$\begin{aligned} Precision &= \frac{TP}{TP + FP} \\ Recall &= \frac{TP}{TP + FN} \end{aligned} \tag{2}$$

$$F1 = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{3}$$

2.4 Deep Autoencoder

2.4.1 Image Representation

An image X_i can be represented by a (real-valued) vector, composed of the following elements:

1. $\mathbb{Z}^w$ - Pixel-value in the width-dimension.
2. $\mathbb{Z}^h$ - Pixel-value in the height-dimension.

thus $X_i \in \mathbb{Z}^w \times \mathbb{Z}^h$ (for simplicity it is assumed that the image is gray-scale).

2.4.2 Deep Autoencoder

Let $(X_i)_{i\in I}$ be a collection of vectors representing a sample of images (all assumed to be drawn from the same distribution, or more informally all assumed to have the same underlying features). On a fundamental level, the objective of the autoencoder is to find a representation of the underlying distribution of the images, this representation is found using a combination of two approaches: *Image Recreation* and *Latent Density Estimation*. Before the approaches can be properly explained, the architecture of the network must be presented.

2.4.3 Architecture

Figure 4 shows a sketch of networks architecture.

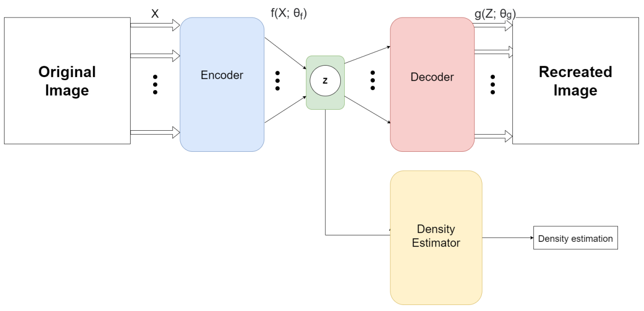


Figure 4: Network Architecture

Note that both the Encoder (blue) and the Decoder (red) are actually dense neural networks. Furthermore, the latent vector $\mathbf{Z}$ is d-dimensional. Using the previous notation and from Figure 4, the following holds.

$$\mathbf{Z} = f(\mathbf{X}; \theta_f)$$
$$\mathbf{Y} = g(\mathbf{Z}; \theta_g)$$

where θ_f and θ_g corresponds to weights in the encoder and decoder respectively.

2.4.4 Encoder

The encoder is a CNN that performs regression, the response are the latent variables (green in Figure 4). Figure 5 shows the encoder structure. The encoder starts by expanding the image channels via convolution, then a downsampling block [1] is used to reduce the image size; downsampling is repeated three times. The downsampled image is then flattened and passed to a fully-connected dense neural network that outputs latent variables. In Figure 5 the sizes of the different layers are shown and the values marked with an underscore show tuneable hyperparameters. The tuneable hyperparameter in the middle of the figure is know as L1 size and determines the size of the whole network, the last tuneable hyperparameter is latent size and thus defines the number of latent variables.

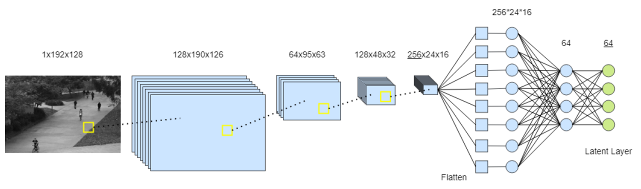


Figure 5: Encoder Architecture. Layer sizes are shown and tuneable hyperparameters are marked with an underscore.

2.4.5 Latent

The latent layer is a lower-dimensional representation of the original image. Under the assumptions that the sample images are drawn from a distribution, the representation is a random vector. In section 2.2.8 it will be further elaborated on how the latent random vector is utilized.

2.4.6 Decoder

The decoder is the inverse of the encoder structure presented in Section 2.4.4. The decoder is therefore a CNN with the same architecture as the encoder but with the order reversed. It starts with a fully-connected dense neural network taking latent variables as input, it is then connected to three upsampling blocks [1] and lastly a convolution to reverse the channel expansion in the encoder structure. However, in order to reverse convolutions, transposed convolutions need to be used.

2.4.7 Image Recreation

Feeding an image through the network, the image can be used as both the regressor and response, thus allowing optimization of the weights in the network to be done by classical back-propagation. By only training the network using images without anomalies, the network will learn to recreate these.

2.4.8 Density Estimation

In general, density estimation is the process of finding the probability distribution of a random variable. In this particular case, the density of the latent vector $\mathbf{Z}$ is of interest. One common approach for this is to assume a given distribution, such as $\mathbf{Z} \sim N_d(\cdot, \cdot)$. However, in this context this assumption is not preferable and a more flexible alternative is needed. One such alternative is a neural network as proposed in [1]. From now on, this neural network will be denoted h and is parameterized by θ_h. The overall objective of the network is that, given a sample Z, be a able to estimate the probability $\mathbb{P}[Z \in \mathbf{Z}]$, assuming that

$$\mathbb{P}[Z \in \mathbf{Z}] = h(Z; \theta_h).$$

More specifically, in [1] an autoregressive neural network (ANN) was used. The key principle of an ANN is the autoregressive part, which implies that it uses previous values to model current values. In this context, previous and current values correspond to components of a vector (ordered by the index). For $Z \in \mathbb{R}^d$, an ANN models the distribution of the second component Z_2 using the conditional distribution

$$\mathbb{P}[Z_2 \in \mathbf{Z}_2 \mid Z_1] = h(Z_1, Z_2; \theta_h).$$

More practically the autoregressive part was accomplished by applying a mask to a fully connected network, ensuring that the predicted variable only depends on the relevant input. To capture the probability of component n of sample Z the network has multiple outputs called bins. The network uses 100 bins per variable, where each bin is denoted by $B_i^{Z_n}$, where i is the bin index and n is the component of the vector. The bin indexes corresponds to the interval the bin covers, expressed as $[\frac{i}{B}, \frac{i+1}{B}]$ where B is the number of bins and i is an integer in the interval $[0, B-1]$.

Each bin represents the probability of a value for a variable and the sum of all bins (for a latent variable) is 1. The distribution of the latent variables is captured by the weights through backpropagation. To assign a probability to a (component of a) sample Z_n, the value of the bin corresponding to the interval containing Z_n is used.

A high value implies that the value for Z_n is part of the estimated distribution and therefore a normal input. A low value implies that it is not part of the estimated distribution and therefore an anomaly. So for example since 100 bins are used, a value $Z_1 = 0.068$ would mean that the bin $B_7^{Z_1}$ would be chosen and the value in the bin represents how well this latent variable matches the distribution. The architecture for the density estimator can be seen in figure 6.

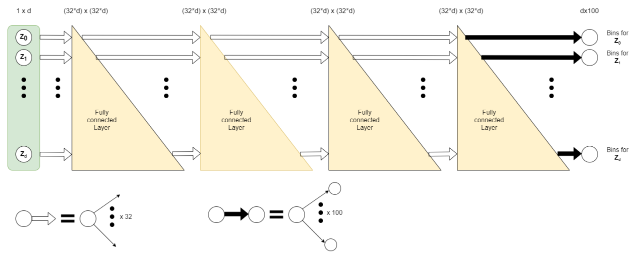


Figure 6: Density estimator architecture

2.4.9 Loss function

Since the model consists of both a neural network and a density model, both needs to be optimized during training. In [1], the following loss function was minimized during training.

$$\mathcal{L}(\mathbf{X}, \mathbf{Y}, \mathbf{Z}; \theta_f, \theta_g, \theta_h) = \mathbb{E}[\|\mathbf{X} - \mathbf{Y}\|_2 + \lambda \log(h(\mathbf{Z}; \theta_h))]$$

where $\mathbb{E}$ is the expectation-operator and $\lambda \in \mathbb{R}$ is a hyper-parameter. It can be noted that this loss function corresponds to minimizing the Kullback-Leibler divergence between the density model and the (true) latent distribution. However, instead of using the combined error, each of the error metrics was used separately. These were defined as the following.

$$\mathcal{L}_1(\mathbf{X}, \mathbf{Y}; \theta_f, \theta_g) = \mathbb{E}[\|\mathbf{X} - \mathbf{Y}\|_2]$$
$$\mathcal{L}_2(\mathbf{Z}; \theta_h) = \mathbb{E}[h(\mathbf{Z}; \theta_h)]$$

2.4.10 Training

The training and validation parts of the dataset were used for model training. At the start of training, only the autoencoder was trained using $\mathcal{L}_1$ as the loss function. After that, the trained autoencoder was used to generate latent variables for the density estimator to train on, using $\mathcal{L}_2$ as the loss function. The following method was used for training both the autoencoder and the density estimator.

First we decided a batch size, the number of training samples that will be used in a forward and backward pass. The training set was then split up into batches of the predetermined size. Running all batches is called a epoch. After a batch was run the weights of the model were updated. To decide if the weight update was an improvement of the model a validation test was performed. For the validation test we used two different approaches, No Ratio and Use Ratio described below.

No Ratio
This approach only uses normal images from the validation data to compute the loss for the part of the model being trained. A lower score implied a better model so if the current score was lower than the previous best then the weights were saved and the best score updated.

Use Ratio
This approach uses both normal images and images containing anomalies from the validation data to compute the loss for the part of the model being trained.

The score was defined as the loss value for the anomaly images divided by the loss value for the normal images. Here a score value implied a better model so if the current score was higher than the previous best the weights were saved and the best score updated.

2.4.11 Classification

Let $\theta = (\theta_f, \theta_g, \theta_h)^T$ be the final weights after training, and let $(X_i)_{i \in I}$ be a set of validation images. For each sample, a reconstruction $Y_i = g(f(X_i; \theta_f); \theta_g)$ is generated, and the density $Z_i = h(X_i, \theta_h)$. For each sample in the validation set, both the reconstruction- and the density estimation-error can now be determined. Using this information, an optimal partition of the space is found using Quadratic Discriminant Analysis (QDA).

2.4.12 Quadratic Discriminant Analysis

Under the assumption that the respective errors are drawn from a class specific (bivariate) normal distribution, where anomalies and non-anomalies have their own respective mean and covariance [8] (thus each is forming a cluster in a sense). More formally, let the (bivariate) error be defined as

$$E_i = (\mathcal{L}_1(X_i, Y_i; \theta_f, \theta_g), \mathcal{L}_2(Z_i; \theta_h))^T$$

and C being the random variable corresponding if the sample is an anomaly or not. Then, for $i \in I$ it is assumed that

$$E_i \mid C, \mu_C, \Sigma_C \sim N_2(\mu_C, \Sigma_C)$$

The classification is performed by estimating μ_C, Σ_C and C from the data, and then classifying a sample based on the maximum posterior probability [14]

$$\max_{c \in \{A,N\}} \mathbb{P}[C = c \mid E_i, \mu_C, \Sigma_C]$$

2.4.13 Classification, continued

In contrast to the regular use of QDA, the classification is performed using a threshold for the posterior probability, instead of maximizing it. This provides additional resistance to outliers. The threshold is set by a grid search during model validation. Tie breaks were solved by selecting the smallest threshold, this provides a model bias towards anomaly classification which is preferred.

3 Results

3.1 CNN + Outlier detection algorithm

3.1.1 Separate- and cross-validation

The following table shows the F1-score and accuracy of different outlier detection algorithms using either cross-validation or a separate validation set for the tuning of hyperparameters. This is further discussed in the discussion chapter. All of the following results is from using the *GoogLeNet* CNN and testing on the baseline dataset.

Outlier detection algorithm	CV	F1	Accuracy
OCSVM	No	0.8353	0.8818
OCSVM	Yes	0.5684	0.7637
LOF	No	0.8536	0.8991
LOF	Yes	0.8755	0.9049
iForest	No	0.8924	0.9222
iForest	Yes	0.8376	0.8905

Table 1: Results from using either CV or a seperate validation set on different outlier detection algorithms.

3.1.2 Different combinations of CNN and outlier detection algorithm

The following table present the result when testing two different CNN architectures other than *GoogLeNet* combined with the three outlier detection algorithms. The outlier detection algorithms were all tuned using a separate validation set and tested on the test set. The dataset used for these results was the baseline dataset.

Outlier detection algorithm	CNN	F1	Accuracy
OCSVM	Efficientnet_v2_s	0.9783	0.9827
OCSVM	Efficientnet_v2_m	0.9189	0.9308
iForest	Efficientnet_v2_s	1.0000	1.0000
iForest	Efficientnet_v2_m	0.9429	0.9539
LOF	Efficientnet_v2_s	1.0000	1.0000
LOF	Efficientnet_v2_m	0.9603	0.9683

Table 2: Results from using Efficientnet as a cnn model.

3.1.3 Results on combined dataset

From the results in the previous section it can be noted that both iForest and LOF performed the same when used with the Efficientnet_v2_s CNN. To further evaluate these and the confirm that Efficientnet_v2_s works better than Efficientnet_v2_m additional tests were performed on the combined dataset. The results can be see below.

Outlier detection algorithm	CNN	F1	Accuracy
LOF	Efficientnet_v2_s	0.8417	0.9323
LOF	Efficientnet_v2_m	0.7302	0.8878
iForest	Efficientnet_v2_s	0.8123	0.8993
iForest	Efficientnet_v2_m	0.6770	0.8284

Table 3: Results from combined dataset.

3.2 Deep autoencoder

3.2.1 Baseline Quantitative

Results on the baseline dataset using the two autoencoder training methods is presented in tables below. The results show that apart from some outliers, both methods and most parameters seem to create well performing models. It is clear that performance on validation data is not sufficient for model comparison using this dataset and therefore a qualitative analysis is appropriate.

Parameters				Validation		Test	
Ratio	L1 Size	LR	Latent Size	Accuracy	F1-score	Accuracy	F1-score
No	512	1e-2	64	1.0	1.0	0.9914	0.9891
No	512	1e-2	32	0.9812	0.9799	0.9827	0.9784
No	512	1e-2	16	1.0	1.0	0.9942	0.9927
No	512	1e-3	64	1.0	1.0	0.9971	0.9963
No	512	1e-3	32	1.0	1.0	0.9856	0.9819
No	512	1e-3	16	1.0	1.0	0.9942	0.9927
No	512	1e-4	64	1.0	1.0	1.0	1.0
No	512	1e-4	32	0.925	0.9104	0.9741	0.9663
No	512	1e-4	16	1.0	1.0	0.9827	0.9784
No	256	1e-2	64	1.0	1.0	1.0	1.0
No	256	1e-2	32	1.0	1.0	1.0	1.0
No	256	1e-2	16	0.8688	0.8712	0.8905	0.8766
No	256	1e-3	64	1.0	1.0	0.9971	0.9963
No	256	1e-3	32	0.9812	0.9799	0.9798	0.9749
No	256	1e-3	16	1.0	1.0	0.9856	0.9819
No	256	1e-4	64	1.0	1.0	0.9971	0.9963
No	256	1e-4	32	0.9938	0.9932	0.9769	0.9710
No	256	1e-4	16	1.0	1.0	0.9971	0.9963
No	128	1e-2	64	0.975	0.9733	0.9452	0.9324
No	128	1e-2	32	0.9875	0.9863	0.9769	0.9704
No	128	1e-2	16	1.0	1.0	1.0	1.0
No	128	1e-3	64	1.0	1.0	0.9971	0.9963
No	128	1e-3	32	1.0	1.0	0.9971	0.9963
No	128	1e-3	16	0.9938	0.9931	1.0	1.0
No	128	1e-4	64	1.0	1.0	0.9885	0.9855
No	128	1e-4	32	1.0	1.0	0.9971	0.9963
No	128	1e-4	16	0.9938	0.9932	0.9971	0.9963

Table 4: Results from baseline dataset not using ratio

Parameters				Validation		Test	
Ratio	L1 Size	LR	Latent Size	Accuracy	F1-score	Accuracy	F1-score
Yes	512	1e-2	64	0.9938	0.9932	0.9827	0.9776
Yes	512	1e-2	32	0.85	0.8481	0.8415	0.8297
Yes	512	1e-2	16	1.0	1.0	0.9971	0.9963
Yes	512	1e-3	64	1.0	1.0	0.9971	0.9963
Yes	512	1e-3	32	0.975	0.9733	0.9654	0.9577
Yes	512	1e-3	16	1.0	1.0	1.0	1.0
Yes	512	1e-4	64	1.0	1.0	0.9942	0.9927
Yes	512	1e-4	32	1.0	1.0	0.9942	0.9927
Yes	512	1e-4	16	1.0	1.0	0.9914	0.9891
Yes	256	1e-2	64	1.0	1.0	0.9971	0.9963
Yes	256	1e-2	32	1.0	1.0	0.9827	0.9783
Yes	256	1e-2	16	0.9812	0.9793	0.9769	0.9704
Yes	256	1e-3	64	1.0	1.0	1.0	1.0
Yes	256	1e-3	32	0.9938	0.9932	0.9885	0.9854
Yes	256	1e-3	16	0.9562	0.9504	0.9366	0.9134
Yes	256	1e-4	64	1.0	1.0	0.9914	0.9891
Yes	256	1e-4	32	1.0	1.0	1.0	1.0
Yes	256	1e-4	16	1.0	1.0	0.9914	0.9891
Yes	128	1e-2	64	0.95	0.9481	0.9424	0.931
Yes	128	1e-2	32	1.0	1.0	1.0	1.0
Yes	128	1e-2	16	1.0	1.0	0.9971	0.9963
Yes	128	1e-3	64	1.0	1.0	1.0	1.0
Yes	128	1e-3	32	0.9875	0.9865	0.9798	0.9749
Yes	128	1e-3	16	0.6875	0.6875	0.7032	0.6997
Yes	128	1e-4	64	0.9938	0.9932	0.9798	0.9744
Yes	128	1e-4	32	1.0	1.0	0.9942	0.9927
Yes	128	1e-4	16	1.0	1.0	1.0	1.0

Table 5: Results from baseline dataset using ratio

3.2.2 Baseline Qualitative

Since the values were very high for most models a qualitative analysis was performed where the $\mathcal{L}_1$ and $\mathcal{L}_2$ values were plotted for validation data along with the trained classification boundary. These plots were then evaluated based on the separation of the two dataclasses and the simplicity of classification. From these plots it could be concluded that the best performing models seemed to be the ones using an L1 size of 256, a latent size of 64 and a learning rate of 10^{-4}; this patterns was concluded for both training methods.

For the autoencoder portion of the model it is possible to calculate pixelwise $\mathcal{L}_1$ loss, it is therefore possible to create a heatmap of where the image reconstruction is incorrect. Such a heatmap is shown in Figure 7. This figure shows that the model is able to detect and locate the anomaly but it also shows that people are viewed by the autoencoder as anomalies, this is not intended.

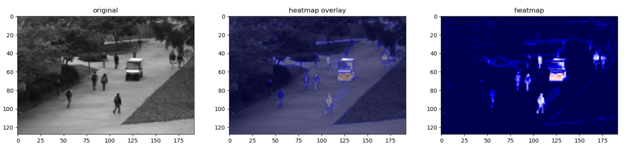


Figure 7: Heatmap of pixelwise $\mathcal{L}_1$ loss in image containing anomaly. Model parameters: (Ratio: No, L1 Size: 256, LR: 10^{-4}, Latent Size: 64)

3.2.3 Combined Quantitative

Results on the combined dataset are shown in the table 6. The results indicate good generalization performance for the models tested and show that there may be a slight difference between the training methods used. For the smaller L1 size tested, the training method using both normal and anomaly images during model selection seems to perform better but this does not hold for the larger L1 size. Therefore, a qualitative analysis was also justified for this dataset as the models performance is very similar between the models.

Parameters				Test	
Ratio	L1 Size	LR	Latent Size	Accuracy	F1-score
No	512	1e-4	64	0.8482	0.8598
No	256	1e-4	64	0.7838	0.7911
Yes	512	1e-4	64	0.8465	0.8584
Yes	256	1e-4	64	0.8382	0.8492

Table 6: Results from combined dataset

3.2.4 Combined Qualitative

Similar to the process for the baseline dataset, $\mathcal{L}_1$ and $\mathcal{L}_2$ values were plotted for validation data along with the trained classification boundary. From these plots it was concluded that the method using both normal and anomaly images during model selection was preferred as it produced a plot were normal images contained smaller $\mathcal{L}_2$ errors which is preferred.

4 Discussion

4.1 CNN and Outlier Detection

This chapter will provide the discussion of the model using a CNN and outlier detection algorithms.

4.1.1 Outlier detection algorithms

The first step of evaluating the outlier detection algorithms was to choose how to tune the hyperparameters of the different algorithms. This was done using both cross-validation and using a separate validation set.

One significant difference in these two methods is that cross-validation uses the same data as the model uses for training for evaluation, which in this case means that only normal data points are used. When using a separate validation set, anomalies were included. The hypothesis was that the separate validation set would make the model generalize better since it would actually configure it's hyperparameters with regard to anomalies and not only normal samples. It was assumed that when using CV, the model might tune the parameters in favor of classifying a data point as normal since that would always be the correct classification when doing evaluation with CV. The downside of a separate validation set with anomalies is that the model might only configure to the specific anomalies that in the validation set and fail to detect anomalies that are vastly different.

As can be seen in the results, it does seem that the recall got significantly worse in two out of three algorithms when using CV, which in words translate to the model not detecting a lot of the anomalies present in the test set. When testing the *Local Outlier Factor* algorithm it does however perform a bit better when using cross-validation. Since it does seem to perform better most of the time with a separate validation set, it was chosen to be the primary method for tuning the hyperparameters when comparing the different models with different CNN:s.

When comparing CV to a separate validation set, all test runs used the *GoogLeNet* CNN. This was chosen due to the paper which the architecture was based on used GoogLeNet. Due to the time-constraint it was not feasible to test all different combinations of CNN and outlier detection algorithm in order to choose a method for parameters tuning.

4.1.2 Efficientnet_v2_s

The results when using the *Efficientnet_v2_s* CNN and either the *LOF* or *iForest* outlier detection algorithms shows the surprisingly good results of 1.0 F1-score (meaning 0 miss-classifications) on the baseline dataset. This is most likely because the baseline dataset used is rather simple compared to most datasets and contains rather big anomalies. All images being taken from the same camera and angle is also a factor that might show results that look "too good". Nevertheless the results show that most likely the use of Efficientnet_v2_s is more beneficial than using GoogLeNet since the results show that using Efficientnet_v2_s gives better results than GoogLeNet no matter which of the three outlier detection algorithms were used. It is to be expected that both EfficientNet models perform better than the GoogLeNet model. This is due to the fact that EfficientNet-V2 is the latest state-of-the art model from the CNN architecture family and has shown better classification results on the ImageNet-1K dataset [2] [3]. The results also shows that the small version of Efficientnet gives better results than the medium-sized version (*Efficientnet_v2_m*). That Efficientnet_v2_s shows better results than Efficientnet_v2_m is unexpected as the latter mentioned is a more up-scaled version than the earlier one. As Efficientnet_v2_m is larger, it should be better at extracting relevant features and thereby show higher classification results. One reason for this behavior could be that Efficientnet_v2_m is better at generalizing as it has more parameters. Therefore, it could perform better if the anomalous images contained more fine grained anomalies compared to the vehicles that are used in this experiment. Another reason could be due to randomness, which in just this particular dataset Efficientnet_v2_s showed better classification results. As well, it could be a combination of these two reasons. In order to get a more trustworthy answer about which architecture that performs best, a variety of different datasets would have to be tested. Some further on the combined dataset were performed, and which showed that the small version of Efficientnet is the best performing version.

[2]https://pytorch.org/vision/stable/models.html
[3]https://image-net.org

4.1.3 Comparison to paper

The paper by Chriki et al. [3], which this architecture was based on, showed very promising results of 0.93 F1-score. It is however not very informative to make a direct comparison to the results obtained by our models, since the paper used a completely different dataset.

4.1.4 Further work

For further work, the main focus should be to test the model with different datasets. Due to the very limited dataset, it is hard to say how well the model generalize to different environments. It is also hard to know how many different environments it could be trained on at the same time.

Further work should also try to use this method to detect anomalies in videos. In the original paper [3], they for example classify a video sequence as an anomaly if 40% of the frames are classified as outliers.

4.2 Deep Autoencoder

This chapter will provide the discussion of the model using a deep autoencoder.

4.2.1 Reconstruction

From the results presented it is clear that for the task of anomaly detection in images the model is good. However, the heatmap shown in Figure 7 indicates that the autoencoder may not solve the problem in the intended way. The heatmap shows that anything that is not background has a large $\mathcal{L}_1$ loss and will therefore perhaps be considered an anomaly. This means that people in general are considered anomalies which is not the intended outcome, as people are considered normal in the datasets used. What this means for classification is that an image containing a sufficient amount of people may be considered an anomaly, due to people being considered anomalies and increasing the $\mathcal{L}_1$ loss. It is unclear whether this effect is reduced by the $\mathcal{L}_2$ loss factor, because it is not possible to extract pixelwise $\mathcal{L}_2$ loss. Further research is needed to determine the impact the $\mathcal{L}_2$ loss has on the model and extract some metric for image area specific $\mathcal{L}_2$ loss.

4.2.2 Classification

The main deviation of the autoencoder compared to the model by Abati et al. [1] were the choice to use QDA on the space of errors separately (instead of adding the errors and using a scalar as a threshold). This provided significantly better performance compared to the proposed method. However, it is not certain that the generalization of the chosen approach is better, as it is possible that the assumptions of QDA (each class has a normal distribution) are too strong. This is to be considered for further evaluation, for example when using video sequences as the dataset.

4.2.3 Comparison to paper

Since the foundations of the deep autoencoder were inspired by Abati et al. [1], it provides a suitable comparison for discussion. However, the authors used AUROC for evaluating their model, and the results for the model are only given by the classification accuracy and F1-score. The reason for the used metrics is due to the direct applicability to the problem description, as it provides an indicator of both the direct classification error, but also an insight into the misclassification of the model. Additionally the authors in [1] used video sequences for anomaly detection in the UCSD Peds dataset, which also impairs the comparison. However, for this particular task, the accuracy is more important, it can be noted that the accuracy for the model and the AUROC for the model in [1] are similar.

4.2.4 Further work

For further work, the main challenge is to develop support for anomaly detection in video sequences instead of images. Not only would this provide a better suited solution for the problem at hand, but it may also enable better results, as video sequences inherently has more information due to the spatio-temporal relations. Furthermore using colors for images (and also video sequences) may also be useful in terms of accuracy, as these also contains more information. Another aspect to consider for further study is the memory and time complexity of the model as these metrics are crucial for the real-life application.

4.3 Comparison

It can be noted that both models provide a very good accuracy and F1-score for both the baseline and combined datasets. Thus it is hard to conclude the model which is to prefer for the task at hand from the aspect of accuracy. For further study it would therefore be useful to evaluate both models using videos, as this would provide a better comparison of the generalization of the models and thus allow for more differentiation of the properties for each model. It would also be useful to evaluate the models on other datasets, since this might show which one of the models generalizes better. It might also show that one model works better than the other in different environments. Another aspect to consider for further study is the complexity in terms of both memory and time.

References

[1] Davide Abati, Angelo Porrello, Simone Calderara, and Rita Cucchiara. Latent space autoregression for novelty detection. arXiv, 2018.

[2] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 1 edition, 2007.

[3] Amira Chriki, Haifa Touati, Hichem Snoussi, and Farouk Kamoun. Uav-based surveillance system: an anomaly detection approach. In *2020 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6, 2020.

[4] Wikipedia contributors. Local outlier factor, November 2022. [Online; accessed 17-November-2022].

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[6] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[7] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, et al. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *Advances in neural information processing systems*, 32, 2019.

[8] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning: with Applications in R.* Springer, 2013.

[9] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie.

Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[10] Fei Tony Liu, Kai Ting, and Zhi-Hua Zhou. Isolation forest. pages 413 – 422, 01 2009.

[11] Sejun Park, Chulhee Yun, Jaeho Lee, and Jinwoo Shin. Minimum width for universal approximation. *CoRR*, abs/2006.08859, 2020.

[12] Jin Pu, Mou Lichao, Xia Gui-Song, and Zhu Xiao. Anomaly detection in aerial videos with transformers. In *IEEE Transactions on Geoscience and Remote Sensing*, pages 1–13, 2022.

[13] Bernhard Schölkopf, Robert C Williamson, Alex Smola, John Shawe-Taylor, and John Platt. Support vector method for novelty detection. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999.

[14] Oleg Sysoev. Lecture notes in machine learning, October 2020.

[15] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[16] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions, 2014.

[17] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR, 2019.

[18] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In *International Conference on Machine Learning*, pages 10096–10106. PMLR, 2021.

[19] V. Bhalodia V. Mahadevan, W. Li and N. Vasconcelos. Anomaly detection in crowded scenes. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, San Francisco, CA, 2010.

[20] Wallenberg. WASP Research Arena - public safety, 2022.

[21] Wikipedia contributors. F-score — Wikipedia, the free encyclopedia, 2022. [Online; accessed 1-December-2022].

[22] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.