Detecting Images Outside Training Distribution for Fingerprint Spoof Detection

Daniel Holmkvist

Master's thesis
2023:E25

LUND INSTITUTE OF TECHNOLOGY
Lund University

Centre for Mathematical Sciences
Mathematics

CENTRUM SCIENTIARUM MATHEMATICARUM

Abstract

Artificial neural networks are known to run into issues when given samples that deviate from the training distribution, where the network may confidently provide an incorrect answer. Out-of-distribution detection methods aims to provide a solution to this issue, by detecting data that deviates from the distribution used to train the model. This thesis looks at the possibility of using out-of-distribution methods in a more challenging context, where the data looked at is more semantically similar than what is often looked at in the literature. Three out-of-distribution detection methods are tested and evaluated at separating fingerprint images from out-of-distribution images, including spoof molds of fingerprints, generated fingerprints using GANs, and non-finger images. The results on the non-finger dataset are in line with the literature, and all methods give promising results on the non-finger dataset. None of the methods are however able to separate the more challenging spoof molds dataset from the in-distribution images. However, on the generated dataset the methods are able to provide somewhat encouraging results, though the performance is significantly lower than on the semantically different datasets.

Keywords: Machine learning, Out-of-Distribution, Deep Neural Network, spoof detection, Neural Network

Contents

Chapter 1

Introduction

1.1 Background

Precise Biometrics is a company in Lund working with developing fingerprint detection methods for various applications, such as for smartphones, cars, and even credit cards. More and more systems are using fingerprint scanners for security today, and with that there is a growing importance in preventing malicious attempts [17]. One of the main challenges is to be able to recognize spoofs, that is if a fingerprint is real or artificial. Malicious actors may use spoofs to impersonate target victims that have some desired authorization, thereby hoping to bypass the security. Spoof fingerprints can be generated using a variety of methods, and as detection methods have been getting more advanced, so have the spoofing methods. A simple spoofing method may be to take a photo with a high-resolution camera of the fingertip and then printing it on paper, but there are also more sophisticated methods. For example, engraving the fingerprint on a material which is soft and flexible, such as silicon or Play-Doh, may make the spoof more difficult to detect [17].

Because of the many different ways of spoofing fingerprints it is difficult to cover all possibilities in a training data set. This creates a problem as classification algorithms are known to give strange results for data outside of the training distribution [1]. In this case classifiers using deep neural networks often ends up giving high confidence predictions, which makes it difficult to rely upon prediction probabilities as a confidence estimate [7]. For safety critical systems it is therefore important for the system to recognize if a given image is too far from the training data, such that it can not give a good classification of the image [1].

Out-Of-Distribution (OOD) detection tries to solve this by using methods to detect data deviating from the underlying distribution that the model was trained on. Data outside the training distribution is referred to as OOD data, whereas data inside of the training

distribution is referred to as ID, or in-distribution data.

The Out-Of-Distribution (OOD) detector will function as an addition to a pre-trained fingerprint classifier. This matcher will often scan through a large amount of images, and it is therefore desirable that the OOD detector is fast, and does not cause a significant overhead to the system.

1.2 Goals

The aim with this Master's thesis work is to explore, implement, and analyze fast methods for out-of-distribution detection for fingerprints using the models developed by Precise Biometrics. Different models for OOD detection will be implemented using TensorFlow and PyTorch, and they will be evaluated on a test data set. In particular, the goal is to explore the following research question:

- How well does different modern state-of-the-art OOD detection methods perform in a more challenging context?

1.2.1 Limitations

- The methods compared and presented do not make up an exhaustive list of modern out-of-distribution detection methods.
- The results presented are not necessarily generalizeable to other datasets.
- The neural network models and datasets used are from Precise Biometrics and will therefore not be explained in detail. This also means that fingerprint images will not be present in the report.

1.3 Terminology

The terminology regarding OOD detection in the literature can be confusing, as there are many areas pointing to a similar problem but with some variations in the assumptions and problem description [18]. Here, we use the definition that the aim of OOD detection is to identify test samples that come from a distribution that is different from the distribution of the training data. The goal of OOD detection is to flag such samples as being out-of-distribution, so that appropriate action can be taken, such as rejecting the sample or assigning it a lower confidence score.

Chapter 2

Theory

This chapter aims to present the relevant theoretical basis the thesis is dependent upon. This chapter will first offer a short introduction to neural networks. That is followed by an introduction to the OOD detection field where the terminology regarding OOD samples will be discussed and defined. Next the methods chosen for evaluation will be described. Finally, methods of analyzing the results on real data will be discussed and presented.

2.1 Neural Networks

Artificial Neural Networks (ANNs) are a form of machine learning algorithms that are loosely inspired by the human brain. In an ANN, a neuron (or node), receives some input which it processes before generating an output. In a simple feed-forward network, neurons are organized into layers, where the output of one layer serves as the input to the next layer [5]. An example of a neural network can be seen in Figure 2.1.

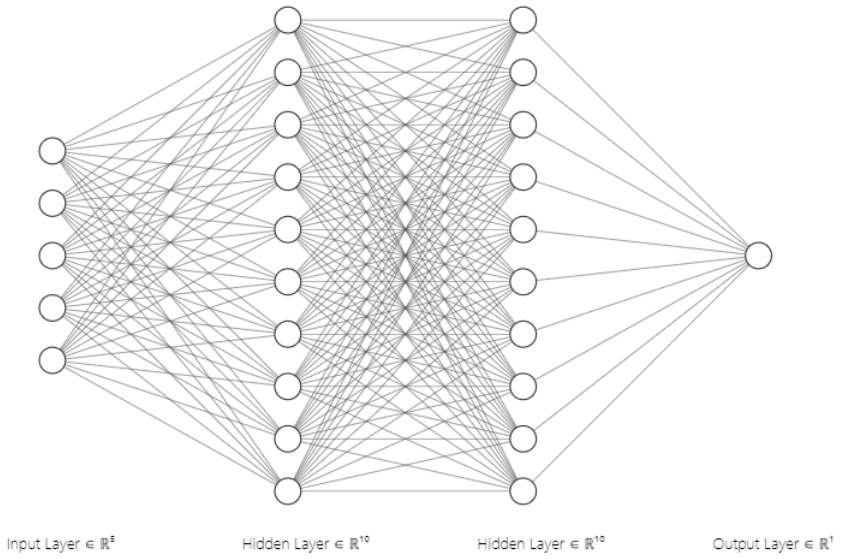


Figure 2.1: Figure of a fully connected DNN with two hidden layers.

the Figure shows an example of a fully connected Deep Neural Network (DNN), a type of ANN that have multiple "hidden" layers, that is layers between the input and output layers. They are designed to learn increasingly complex representations of the input data as it passes through the network, with each layer learning higher-level features than the previous layer.

The DNN can be seen a nonlinear mapping as $f : X \rightarrow Y$ where X represents the input data and Y the output. The function $f(x)$ can be described as a composition of functions $g_i(x)$ where $g_i(x)$ represents the function of the previous layers nodes, which themselves may be composition of other functions. Generally, a nonlinear weighted sum is used, which is described by [5]:

$$f(x) = K\left(\sum_i w_i g_i(x)\right), \tag{2.1}$$

where K is called the activation function. The activation function introduces a nonlinearity that allows for more complex learning by the network. Without the introduction of the nonlinearity all layers of neurons would function as one, as any linear combination of linear functions is itself linear. There are many suitable functions that may be used as activation functions, such as the sigmoid, the ReLU, and tanh [5].

For classification tasks, it is often desirable to create a probability distribution over the classes. This is done by taking the outputs of the final layer, called the logits, and normalizing by use of a final activation function. The class with the highest conditional probability is then picked as the predicted class for the input. For classification tasks, a common choice of final activation function is the softmax function.

During training, an ANN adjusts the weights of the connections between neurons so that the output of the network matches the desired output, according to some loss function. This is

done by comparing the predicted output of the network to the actual output and adjusting the weights to minimize the difference between them. This process is called backpropagation, and is used to update the weights in the network based on the some loss function dependant on the predicted output and the expected output [5]. The update is scaled by a hyperparameter, the learning rate μ, that determines the step size at which the parameters of a model are updated. A high learning rate may result in faster convergence, but it can also cause the model to overshoot the optimal solution or even diverge. On the other hand, a low learning rate may lead to slow convergence or getting stuck in suboptimal solutions. For gradient-based optimization methods like stochastic gradient descent (SGD) another hyperparameter is often introduced, the momentum. Momentum allows the optimization method to remember its previous updates and use them as a factor in the current update. This helps accelerate the convergence by dampening oscillations and providing stability in the optimization process [5].

For image analysis, Convolutional Neural Networks (CNNs) are of particular importance. CNNs use convulutional layers, which allow small matrices ("filter") to slide over image, creating multiple results in a new "image" [13]. During training, these filters are adjusted such that it detects features relevant according to the loss function.

Another important aspect of CNNs is pooling layers, which are used to downsample the output of the convolutional layers. Pooling reduces the spatial dimensions of the feature maps and makes the network more robust to small changes in the input image. Commonly max pooling is used, where the maximum value in each region of the feature map is kept, or average pooling, where the average value in each region is kept [5].

2.2 OOD Methods

This section provides a description of all methods for OOD detection evaluated. First, the Max Softmax Probability method is presented, a simple method looking only at the softmax probabilities of the output of a neural network classification model. That is followed by looking at more sophisticated methods: Mahalanobis Distance, Gram Matrix and FOOD.

2.2.1 Max Softmax Probability

In a neural network classification model, the final output is typically a vector of class probabilities, where each entry corresponds to the probability of the input belonging to a specific class. This vector is typically obtained by applying the softmax activation function to the final layer's outputs. The softmax function $\sigma(\mathbf{z})_i$ is defined as:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \text{ for } i = 1, \ldots, K \text{ and } \mathbf{z} = (z_1, \ldots, z_K) \in \mathbb{R}^K, \tag{2.2}$$

where K denotes the number of classes. The function maps each output to the resulting vector $\mathbf{u}$ with probability value between 0 and 1 and ensures that the probabilities sum to 1. Each entry corresponds to the conditional class probability $P(y = k|x)$ for class $k = 1, ..., K$.

The predicted class $\hat{y}$ is then determined by selecting the class with the highest probability, given by $\arg\max_k \mathbf{u}$ [5].

The max softmax probability (MSP) method is a simple method that makes use of this. The idea is that the network is more likely to be confident in a correct prediction belonging to some class and it therefore tends to have a greater maximum softmax probability than for OOD samples [6]. Therefore, the maximum softmax value can be used as an indication on whether the sample is OOD. However, many studies have shown the max softmax method to be heavily outperformed by more sophisticated methods [3].

2.2.2 Mahalanobis Distance

The Mahalanobis Distance (MD) method was presented in 2018 and achieves state-of-the-art OOD detection problems [10] [1]. The method operates by assuming that in each layer's representation in the DNN, the samples of each class are distributed as a Gaussian, with a shared covariance matrix $\mathbf{\Sigma}$ among all classes. The class-conditional Gaussian distributions are defined as:

$$P(f(\mathbf{x}) \mid y = c) = \mathcal{N}\left(f(\mathbf{x}) \mid \boldsymbol{\mu}_c, \mathbf{\Sigma}\right), \tag{2.3}$$

where $\boldsymbol{\mu}_c$ is the mean of multivariate Gaussian distribution of class $c \in \{1, ..., C\}$ [10].

The covariance matrix is found by computing the class mean and covariance of the training data according to:

$$\widehat{\boldsymbol{\mu}}_c = \frac{1}{N_c} \sum_{i:y_i=c} f\left(\mathbf{x}_i\right), \quad \widehat{\Sigma} = \frac{1}{N} \sum_{c} \sum_{i:y_i=c} \left(f\left(\mathbf{x}_i\right) - \widehat{\boldsymbol{\mu}}_c\right)\left(f\left(\mathbf{x}_i\right) - \widehat{\boldsymbol{\mu}}_c\right)^{\top}, \tag{2.4}$$

where N_c is the number of training samples of label c.

Consider some test sample $\boldsymbol{x}$ with corresponding feature map $\boldsymbol{z}' = f(\mathbf{x})$. The Mahalanobis distance $\mathbf{MD}_c$ between $\boldsymbol{z}'$ and the C fitted Gaussian distributions is calculated as [11]:

$$\mathbf{MD}_c\left(z'\right) = \left(z' - \boldsymbol{\mu}_c\right)^T \mathbf{\Sigma}^{-1} \left(z' - \boldsymbol{\mu}_c\right). \tag{2.5}$$

The Mahalanobis distance can then be used to get a confidence score for how OOD a sample is by taking the negative of the minimum over all classes [10]:

$$C\left(z'\right) = -\min_c \left\{\mathbf{MD}_c\left(z'\right)\right\}. \tag{2.6}$$

Additionally, a technique for separating the ID and OOD samples is presented [10]. During preprocessing, a small noise can be added to each test sample x according to:

$$\widehat{\mathbf{x}} = \mathbf{x} - \varepsilon \operatorname{sign}\left(\nabla \mathbf{x}\left(f(\mathbf{x}) - \widehat{\boldsymbol{\mu}}_{\widehat{c}}\right)^{\top} \widehat{\Sigma}^{-1} \left(f(\mathbf{x}) - \widehat{\boldsymbol{\mu}}_{\widehat{c}}\right)\right), \tag{2.7}$$

where ϵ is the magnitude of the noise, and $\widehat{c}$ is the closest class index to the sample. The authors state that the addition of such a noise increases the confidence score, see equation 2.6, and that the perturbation can lead to a separation of the ID and OOD samples [10].

For each layer ℓ the class means and covariances can be calculated, i.e. $\widehat{\boldsymbol{\mu}}_{\ell,c}$ and $\hat{\boldsymbol{\Sigma}}_{\ell}$. Then the confidence scores are obtained according to equation 2.6. The confidence scores are then combined by a weighted average where the weight of each layer α_ℓ is selected from training a logistic regression detector on validation samples of OOD data.

Adjustment

A recent paper proposes an adjustment to the MD method to improve near-OOD detection rates[15]. They propose using a distance relative to the entire training distribution, the Relative Mahalanobis distance (RMD), without considering the specific class. The RMD is defined as:

$$\mathrm{RMD}_c(\boldsymbol{z}') = \mathrm{MD}_c(\boldsymbol{z}') - \mathrm{MD}_0(\boldsymbol{z}')\,. \tag{2.8}$$

The same calculations to obtain the confidence score are carried out as before, but the $\mathbf{RMD}_c$ is used instead of $\mathbf{MD}_c$ in 2.6.

2.2.3 FOOD: Fast Out-Of-Distribution Detector

The Fast Out-Of-Distribution Detector (FOOD) method has been shown to provide a fast inference time for some datasets [1]. Furthermore, FOOD, as opposed to the MD method does not require any out-of-distribution data samples, which can be thought to increase the reliability of the method.

The method uses a Gaussian Layer that is connected to the penultimate layer of the network. The input to the Gaussian layer is assumed to be a multivariate Gaussian distribution [1]. Each class c is modeled by a trainable positive-semidefinite covariance matrix $\boldsymbol{\Sigma}_c \in \mathbb{R}^{d\times d}$ and mean $\boldsymbol{\mu}_c \in \mathbb{R}^d$ where d is the number of classes. During training, the parameters for the Gaussian layer are adjusted to fit the distribution of each output class [1]. For each class the Gaussian Layer assigns a score corresponding to the log-likelihood of a multivariate Gaussian function as:

$$\log(p(f(x) \mid c)) = \log(\mathcal{N}(x;\boldsymbol{\mu},\boldsymbol{\Sigma})) = -\frac{d}{2}\log(2\pi) - \frac{1}{2}\log(|\Sigma|) - \frac{1}{2}\|x-\boldsymbol{\mu}\|^2_{\Sigma^{-1}}, \tag{2.9}$$

where $f(x)$ is the output of the networks penultimate layer. The covariance matrix is diagonal to improve the speed of computation as well as numerical stability.

The method, despite connecting a new layer to the network does not require it to be retrained. Rather the Gaussian Layer can be fine-tuned after being connected to an already trained model. The Gaussian Layer is initialized from the penultimate layer of the previously trained network as 2.10:

$$\begin{aligned} \boldsymbol{\mu}_c &= \frac{1}{|S_c|} \sum_{x \in S_c} f(x), \\ \boldsymbol{\Sigma}_c &= \frac{1}{|S_c|} \sum_{x \in S_c} (\boldsymbol{\mu}_c - f(x)) (\boldsymbol{\mu}_c - f(x))^T , \end{aligned} \tag{2.10}$$

where S_c are the samples in the training data for class c, and $f(x)$ is the representation of the sample in the penultimate layer of the network.

A final output neuron is connected to the Gaussian Layer, the output of which corresponds to the log-likelihood of the Gaussians corresponding to the different classes in the penultimate representation. Therefore, the log-likelihood ratio ($\mathcal{LLR}$) test can be applied on the log-likelihood values. The score over the predicted class $p_{\text{pred}}(x)$, and the log-likelihood over all other classes $p_{\text{other}}(x)$ can be found as in equation 2.11.

$$\begin{aligned} p_{\text{pred}}(x) &= \max_{c \in \{1,\dots,C\}} \log \left(\mathcal{N}(f(x); \boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c) \right) \\ p_{\text{other}}(x) &= \frac{1}{C-1} \sum_{\substack{c' \neq c \\ c' \in \{1,\dots,C\}}} \log \left(\mathcal{N}(f(x); \boldsymbol{\mu}_{c'}, \boldsymbol{\Sigma}_{c'}) \right) \end{aligned} \tag{2.11}$$

Then, the $\mathcal{LLR}$ value can be calculated as:

$$\mathcal{LLR}(x) = p_{\text{pred}}(x) - p_{\text{other}}(x). \tag{2.12}$$

Note that ID samples have higher $\mathcal{LLR}$ values as compared to OOD samples.

2.2.4 Artifical OOD samples

The FOOD method uses a method for crafting artifical OOD samples. Given a training set, some samples have lower $\mathcal{LLR}$ than others, meaning that they are more likely to be classified as an OOD sample. This can be used to find boundaries of the ID data such that everything within the boundary is considered ID and everything outside the boundary is considered OOD [1]. A threshold is selected for how much of the training data is to be considered as ID, typically 95%, with a corresponding threshold value. During creation of artificial OOD samples the samples are iteratively perturbed towards a lower $\mathcal{LLR}$ score such that they end up below the threshold, and would therefore be considered OOD. The perturbation of a sample x given by:

$$x_{\text{per}} = x - \epsilon \cdot \nabla_x \mathcal{LLR}(x), \tag{2.13}$$

where the perturbation size ϵ is a hyperparameter.

2.2.5 Gram Matrix method

The Gram Matrix method uses extensive analysis of the correlation between features in each layer [16]. For this method (and this method only), "layer" refers to the values immediately obtained after applying convolution, as opposed to all layers in the network. The method looks at feature co-occurrences is the network, that is to say how features in different layers of

the network covary with each other. Given a deep convectional neural network with L layers where layer ℓ has n_ℓ channels co-occurrences of features is looked at between the feature maps of the $\sum_{1<=l<=L} \frac{n_\ell*(n_\ell+1)}{2}$ layers.

To do this, the method computes the Gram Matrices for each layer ℓ, defined as:

$$G_\ell = F_\ell F_\ell^\top, \tag{2.14}$$

where F_ℓ is an $n_\ell \times p_\ell$ size matrix, corresponding to the feature map for an arbitrary input image $\mathbf{x}$, i.e. $F_\ell = F_\ell(\mathbf{x})$, where p_ℓ the number of pixels per channel and n_ℓ is the number of channels for the layer ℓ.

A Gram Matrix of a higher order p, G_ℓ^p, can be used to obtain more prominent feature maps:

$$G_\ell^p = \left(F_\ell^p F_\ell^{p^\top}\right)^{\frac{1}{p}}. \tag{2.15}$$

where the power and root are element wise operations [1]. Experimentally, it has been shown that use of higher order Gram Matrices can significantly improve results [16].

Compared to the MD method discussed in the previous section the Gram Matrix method goes further than just computing channel-wise means for each layer. A Gram matrix of order p contains the *raw moment* of order p of the element-wise product between the channels in its off-diagonal element and the moments of order 2p of the individual channels along the diagonal [16]. The n-th raw moment $\mu_X(n)$ of a random variable X refers to the expected value of the n-th power:

$$\mu_X(n) = \mathrm{E}\left[X^n\right], \tag{2.16}$$

where $n \in \mathbb{N}$. Note that the first moment corresponds to the mean, the second the variance, and the third the skewness of the distribution of X [14].

The flattened upper triangular matrix along with the diagonal entries, denoted as $\overline{G_l^p}$, is computed during training for each layer and order. The minimum and the maximum value of feature co-occurrences is also stored for each class, layer and order in the corresponding vectors $\mathbf{Mins}$ and $\mathbf{Maxs}$.

During testing a deviation from the training data δ is computed as the percentage change in the maximum or minimum values of feature co-occurances according to equation 2.17

$$\delta(\min, \max, \text{ value }) = \begin{cases} 0 & \text{if } \min \leq \text{ value } \leq \max \\ \frac{\min - \text{value}}{|\min|} & \text{if value } < \min \\ \frac{\text{value} - \max}{|\max|} & \text{if value } > \max \end{cases} \tag{2.17}$$

The deviation of a layer with regards to a test image $\mathbf{x}$ is computed as:

$$\delta_l(\mathbf{x}) = \sum_{p=1}^{P} \sum_{i=1}^{\frac{1}{2}n_l(n_l+1)} \delta\left(\mathbf{Mins}\left[\mathbf{x}_c\right][\ell][p][i], \mathbf{Maxs}\left[\mathbf{x}_c\right][\ell][p][i], \overline{G_\ell^p(\mathbf{x})}[i]\right), \tag{2.18}$$

where P is the maximal order Gram-matrix considered.

The total deviation of an image $\mathbf{x}$, $\Delta(\mathbf{x})$, can then be computed by normalizing the deviations from the different layers (to prevent larger layers from having too much of an impact). The normalization factor is found by calculating the "expected deviation" for layer ℓ using the validation data, $\mathbb{E}_{\text{Va}}$. The total deviation is given by:

$$\Delta(\mathbf{x}) = \sum_{\ell=1}^{L} \frac{\delta_\ell(\mathbf{x})}{\mathbb{E}_{\text{Va}}[\delta_\ell]}. \tag{2.19}$$

Finally, the maximum softmax probability (see Section 2.2.1) is used on the total deviation $\Delta(\mathbf{x})$ to further improve the result.

2.3 Principal Component Analysis

Principal component analysis (PCA) is a technique for reducing the dimensionality of a dataset [8]. A large dataset with many variables is transformed into a smaller dataset with fewer variables, while still retaining the majority of the variability in the data. This is done by creating new, uncorrelated variables called principal components, that successively maximize variance. The principal components are linear combinations of the original variables. Each principal component is chosen to capture as much of the remaining variance in the data as possible, under the constraint that they are orthogonal to any previous principal components [8].

2.4 Metrics

A confusion matrix can be seen in Table 2.1, describing the true positives (TP), false positives (FP), false negatives (FN) and true negatives (TN). The row indicates the predicted value according to the model and the columns indicate the actual value, the ground truth [9].

Table 2.1: Confusion Matrix for binary classification.

		Actual	
		Positive	Negative
Predicted	Positive	TP	FP
	Negative	FN	TN

2.4.1 TPR

The true positive rate (TPR), or the *recall*, is the fraction of positive samples correctly classified as such, i.e. [9]:

$$TNR = \frac{TN}{TN + FP} = 1 - FPR. \tag{2.20}$$

2.4.2 TNR

The true negative rate (TNR), or the *specificity*, is the fraction of negative samples correctly classified as such, i.e. [9]:

$$TNR = \frac{TN}{TN + FP} = 1 - FPR. \tag{2.21}$$

2.4.3 Detection Accuracy

One accuracy measure can be obtained by taking the correct classifications divided by the total number of classifications, as seen in equation 2.22.

$$Acc = \frac{TN + TP}{TN + FN + TP + FP} \tag{2.22}$$

For imbalanced datasets, as is often the case with OOD detection, the accuracy can often be misleading so other metrics may be more suitable [9].

The detection accuracy, or simply accuracy, will here refer to the maximum possible classification accuracy for all thresholds. It can be calculated as

$$\max_{\tau} \{0.5TPR(\tau) + 0.5TNR(\tau)\}. \tag{2.23}$$

Note that this is adjusted for the imbalanced datasets, so we get around the issue with the accuracy measure in equation 2.22.

2.4.4 Thresholded metrics

All the methods discussed in Section 2.2 use a threshold to discriminate between ID and OOD samples once a metric for the deviation of an image have been reached [16]. The mathematical description for the threshold τ is:

$$\text{isOOD}\,(\mathbf{x}) = \begin{cases} \text{True} & \text{if } \Delta(\mathbf{x}) > \tau \\ \text{False} & \text{if } \Delta(\mathbf{x}) \leq \tau \end{cases} \tag{2.24}$$

where $\Delta(\mathbf{x})$ describes the total deviation of an input image $\mathbf{x}$.

Often, this threshold is put at where the TPR is 95%, such that 95% of the training data is detected as ID. A natural performance metric for such a threshold is then how many OOD samples are correctly detected at such a threshold. This is the same as the TNR when the TPR is 95%, and this metric will be referred to as *TNR95* [1].

2.4.5 ROC

The receiver operating characteristic curve (ROC) is created by plotting the TPR against the FPR over varying thresholds [4].

2.4.6 AUROC

The area under the ROC (AUROC), is a measure of performance for the detector over all possible detection thresholds. It holds an important property for analyzing OOD classifiers in that, assuming that the method assigns a higher score to ID samples, the AUROC is equivalent to the probability that a randomly chosen OOD sample is assigned a lower score than an ID sample [4].

While ROC curves are useful for evaluation of classifiers it is not sufficient to use them alone to make conclusions about which classifier is the best. Simply seeing which classifier performs the best in ROC space disregards the variance of the methods, which is necessary for useful comparison. This can however be done by averaging ROC curves [4].

For OOD detection we can easily fix the FP rate, and therefore vertical averaging is deemed suitable. The FP rate is fixed and vertical samples of the ROC curve are taken, and then averaged. This can be done using k-fold cross-validation, where the data is randomly divided into k equally large sets, of which k-1 are used as training data, and the last one is left for validation [2]. The process is then repeated k times, such that each of the sets have been used as validation data once. Finally, the result can be averaged and we can get a measure of the standard deviation of the set. A confidence interval can be computed using the assumption of a binomial distribution [4]. Some research suggest that 10-fold cross-validation presents a comparatively low-bias [12], and it will therefore be used in this report.

Chapter 3

Method

This chapter will first give an overview of the data used for the different tests used to evaluate the methods. Then, implementation details regarding the different methods will be given. This will include a discussion about the assumption of a Gaussian Distribution of the layer representations made by both the Mahalanobis Distance and FOOD methods. Finally, methods for out-of-distribution detection such that only one person, the user, is considered in-distribution will be discussed.

3.1 Models and Data

All data used for evaluating and pre-trained models are from Precise Biometrics. This includes fingerprint databases with real and spoof images of fingerprints, and non-finger databases that contains significantly semantically different data. All datasets used was captured with the same sensor.

Note that it is very difficult to visually separate real fingerprint from spoof images or from fingerprint images from other people. All pre-trained network models have been trained to separate spoof images from real fingerprint images on some data.

The data used consists of four different sets, *ID*, *Nonfinger*, *Generated*, and *Molds*. ID describes the in-distribution data the model has been trained on, this includes both real and spoof images. Nonfinger is a set of non-fingerprint image, and will be considered far-OOD. The dataset *Generated* consists of generated images trained on the ID dataset. This includes both real and spoof images. Visually, the difference between them and a real fingerprint can be hard to make out. An example of a generated fingerprint image can be seen in Figure 3.1.

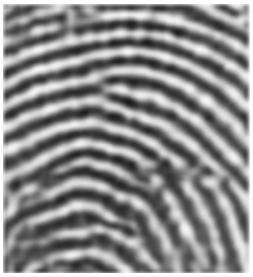


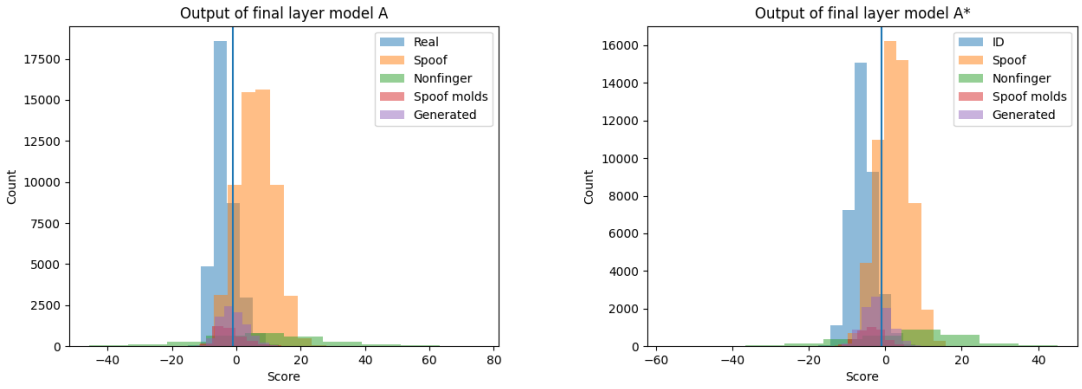
Figure 3.1: Generated fingerprints.

Finally, the *Molds* dataset describes a dataset of intentionally constructed spoof molds. The number of training, validation and test samples used for all datasets can be seen in Table 3.1.

Table 3.1: Number of training, validation, and test samples for each dataset used in the tests for the OOD methods.

Dataset	# of training samples	# of validation samples	# of test samples
ID	50000	25000	25000
Nonfinger	N/A	3000	3000
Generated	N/A	5000	5000
Molds	N/A	5000	5000

The output of the models on the different datasets can be seen in the following Figure



(a) Model A **(b)** Model A*

Figure 3.2: Figure showing the output of the last layer of the model on the different datasets.

Here, we can see that the model is good at distinguishing between the spoof and real fingers. However, for the nonfinger, dataset the model gives a varied result, and it is likely to recognize the spoof molds (OOD) as real fingerprints. For the spoof molds dataset the trained model performs somewhat better, but both models perform quite poorly on that data.

3.2 Tests

The different tests that will be carried out to evaluate methods for OOD detection will be described here. The result of each test can be seen in the corresponding results section.

3.2.1 Test 1: OOD Detection for far OOD Distribution

For this test, the OOD detection methods are applied to images fundamentally different to the training data, i.e. non-fingerprint images. It looks at how the OOD detection methods perform in a real-world setting on a dataset that is more in line with what has been looked at in the literature. Because of this, we would expect similarly good performance on this dataset [3].

3.2.2 Test 2: OOD Detection for Spoof Distribution outside of the Training Data

For test 2, a comparison will be made between a model A trained on the same large set of fingerprint data (including real and spoof images), *ID*, without a particular set of spoof molds, and model A* trained on both the ID data and the spoof molds. It can be seen in Figure 3.2 that model A* significantly outperforms model A on the spoof molds. Note that the spoof molds are not used for training the OOD detection methods for either model, so we look at differences in OOD detection performance from having trained the network on this data versus not having trained it on that data. All images were taken using the same sensor as the one that was used for obtaining the training data. Furthermore, the OOD detection methods will be evaluated on a set of generated fingerprints.

3.3 Assumption of a Gaussian Distribution

The Mahalanobis Distance and FOOD methods both assume the layer representations of an class-specific ID samples to be distributed as a Gaussian. There are reasons to doubt this to be the case. An image of the distributions of some features in the penultimate layer can be seen in Figure 3.3 for the different datasets.

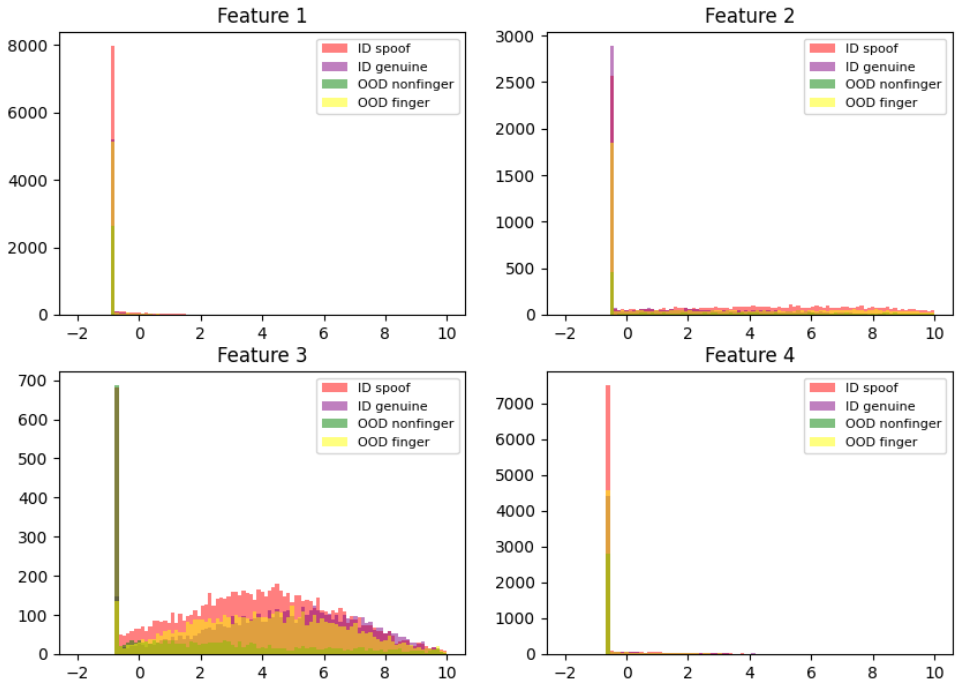


Figure 3.3: Figure showing the distribution of the first 4 features in the penultimate layer for a pre-trained model.

By visual inspection, the features do not appear to be well approximated by a normal distribution. This can be thought to occur since the different features are linearly dependant on each other. Using PCA we can however extract orthogonal components, which may make this assumption more in line with reality, and therefore improve results. Figure 3.4 shows the first 4 principal components after PCA.

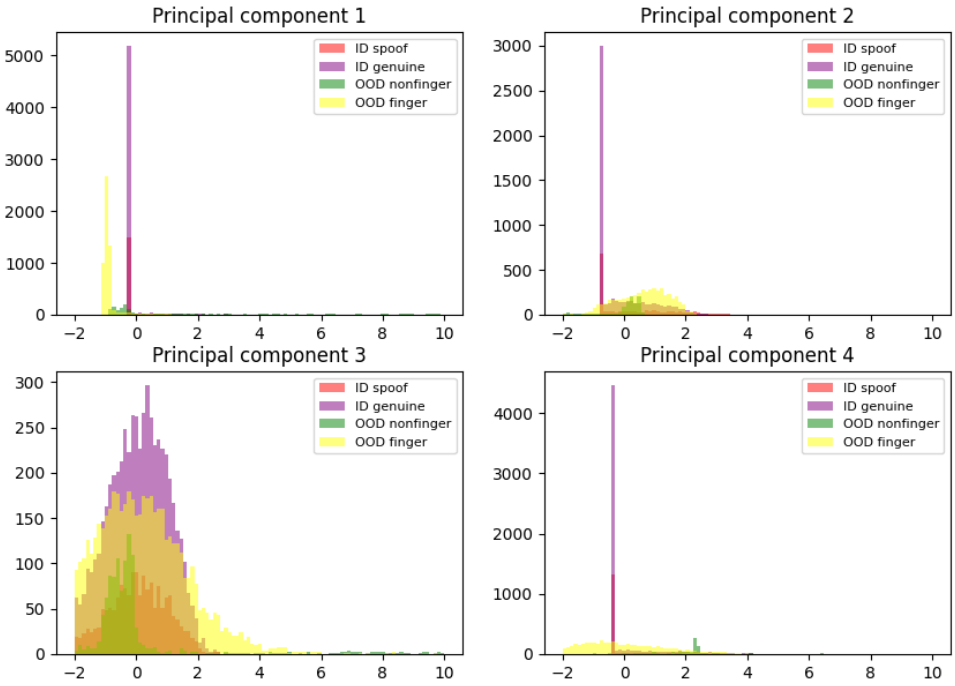


Figure 3.4: Figure showing the distribution the distribution of one feature in the penultimate layer for a pre-trained model after PCA.

Visually we can see that the principal components appear significantly more Gaussian than what the layer representation previously was.

3.4 Implementation

In this section implementation details for all OOD methods will be presented. All test were run on NVIDIA GeForce RTX 2080 Ti using Python 3.8 with TensorFlow 2.12.0. No hyper-parameter tuning was performed in the implementation of each method, other than what is specified in this section.

3.4.1 Mahalanobis Distance method

The Mahalanobis Distance method is non-invasive and works by looking at the representation of a sample throughout the network. In total, 6 layers were selected. To train the logistic regression detector the method for generating artificial OOD samples for training was used with $\epsilon = 0.01$ in addition to the OOD validation dataset. Again, the OOD samples were assigned the label 1, with the ID samples 0.

3.4.2 FOOD

FOOD got a significantly improved performance from using PCA. The number of principal components to be included was decided from performance on the validation dataset. The result on the Nonfinger dataset with and without PCA can be seen in Figure 3.5.

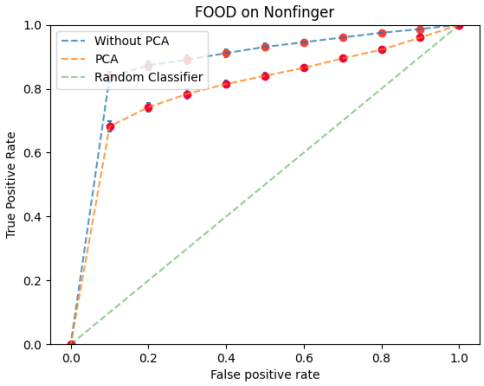


Figure 3.5: ROC curve for FOOD with and without PCA on the nonfinger dataset.

For crafting artificial OOD samples the perturbation size $\epsilon = 0.01$ was used. The OOD samples were assigned the label 1, with the ID samples 0. During finetuning, the Gaussian layer was trained for 1000 epochs, with learning rate $\mu = 0.001$, and momentum 0.9.

3.4.3 Gram Matrix

Similarly to the MD method the Gram Matrix method is non-invasive. In total, 6 layers were selected, each after a convolutional layer as in [16]. The maximum order of the Gram Matrix was selected empirically as $P = 2$ as a compromise between performance on the validation dataset and inference speed.

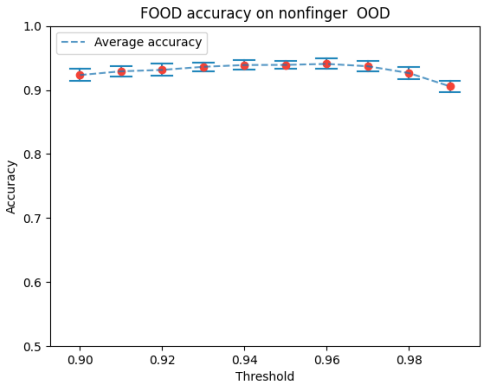
Chapter 4

Results

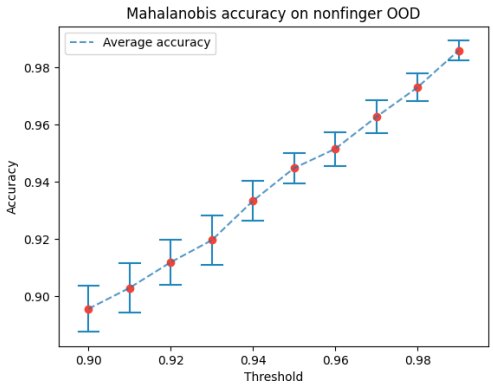
This chapter will present the results of the different methods on the different datasets. First, comparisons will be made for FOOD before and after PCA pre-processing. Then, vertically averaged curves will be presented for the different methods to offer a form of performance comparison for different datasets. These curves will include the threshold on the ID dataset as the x-axis, with the y-axis being the averaged accuracy on the validation dataset for both the ID and OOD data, see equation 2.23. In all of the curves, the points were evaluated between 0.9 and 0.99 (inclusive) and spread out evenly. These will only be presented for thresholds greater than 0.9 as a higher false positive rate (corresponding with a lower threshold) was deemed unreasonable for this application. For the Gram Matrix method the performance depending on the amount of layers selected will be presented.

4.1 Test 1: Performance on Nonfinger dataset

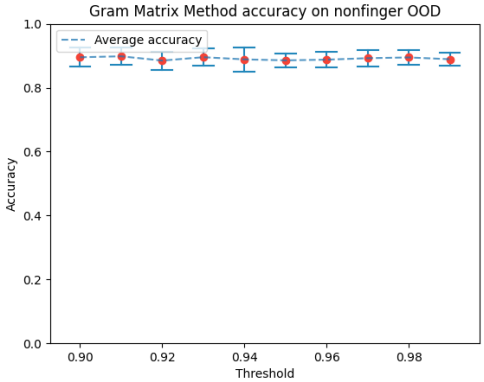
The results on the Nonfinger dataset for the different methods using model A can be seen in Figure 4.1



(a) FOOD



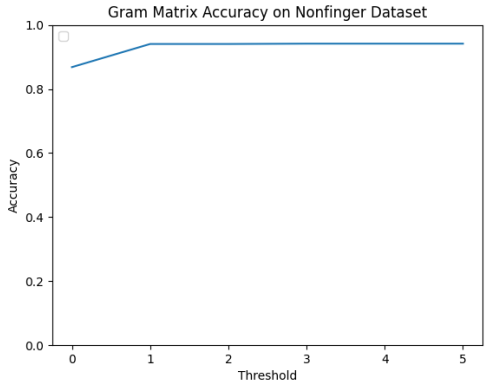
(b) Mahalanobis



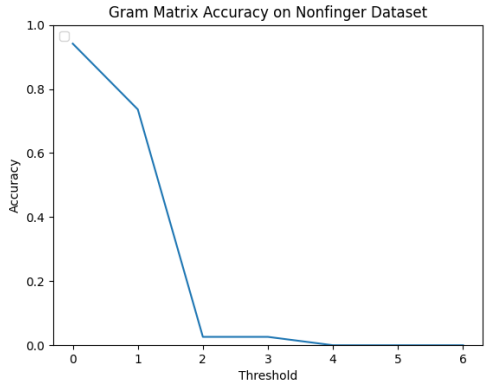
(c) GM

Figure 4.1: The methods results on the Nonfinger dataset. The 95% confidence interval was obtained from using 10-fold cross-validation.

We can see that the FOOD method reaches and accuracy of about 93%, the Gram Matrix method 97.6% and the Mahalanobis method reaches an accuracy of almost 99%. For the Gram matrix method the result (TNR95) depending on how many layers were selected can be seen in the following Figure 4.2.



(a) Performance with additional layers starting at the root of the network.



(b) Performance with additional layers from the end of the network.

Figure 4.2: Gram Matrix method performance on the nonfinger dataset depending on how many layers are selected.

Figure 4.2a displays the result as more layers are added from the root of the network towards the end of the network. We can see that the increase in accuracy close to plateaus after the 2nd layer is added. Figure 4.2b displays it from the other direction, with "0" representing all layers being present, and "5" only the outermost layer.

The summarized results (TNR95) on the nonfinger dataset can be seen in Table 4.1.

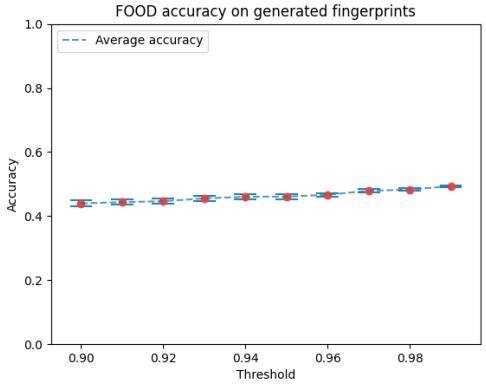
Table 4.1: Averaged result (TNR95) for the different methods.

Dataset	Mahalanobis	FOOD	Gram Matrix
Nonfinger OOD	**99.7**%	93.4%	97.6%

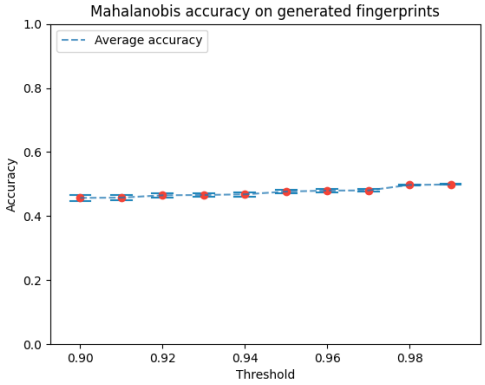
4.2 Test 2: Performance on near-OOD datasets

4.2.1 Generated Dataset

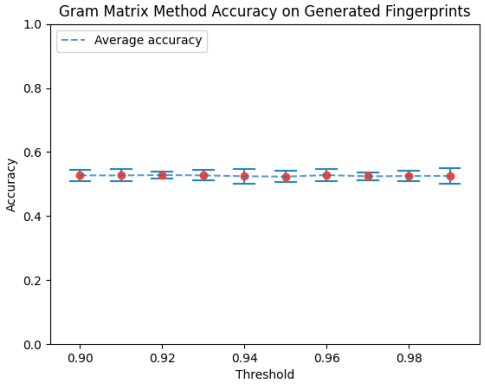
The performance on the generated fingerprint dataset using model A can be seen in Figure 4.3



(a) FOOD



(b) Mahalanobis



(c) GM

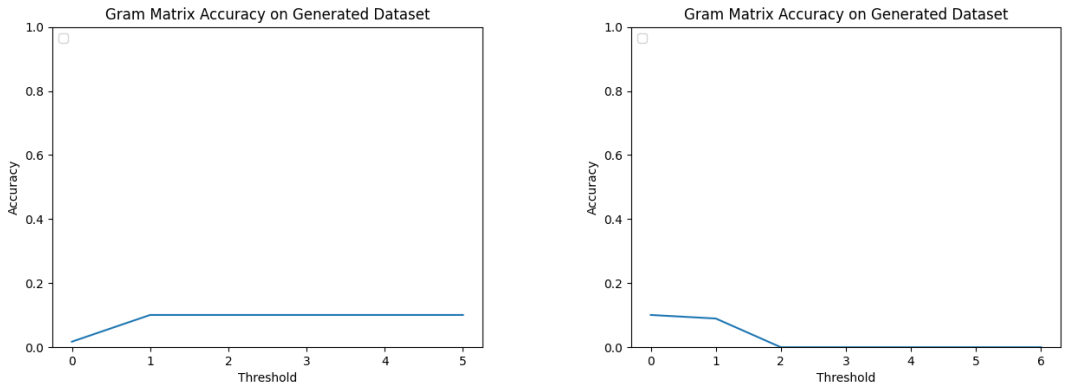
Figure 4.3: The methods results on the Generated dataset. The 95% confidence interval was obtained from using 10-fold cross-validation.

All methods perform significantly worse on the generated dataset as compared to the non-finger dataset. A more detailed view of the result can be seen in table 4.2.

Table 4.2: Averaged result (TNR95) for the different methods.

Dataset	Mahalanobis	FOOD	Gram Matrix
Nonfinger OOD	99.7%	93.4%	97.6%
Generated OOD	11.7%	6.1%	14.3%

We can see that all methods struggle with the generated dataset as compared to the nonfinger dataset. Both the Gram Matrix method, and the Mahalanobis distance method however outperform that of a random classifier (TNR95 5%). The results from the Gram Matrix method depending on how many layers are selected can be seen in Figure 4.4.



(a) TNR95 with additional layers starting at the root of the network.

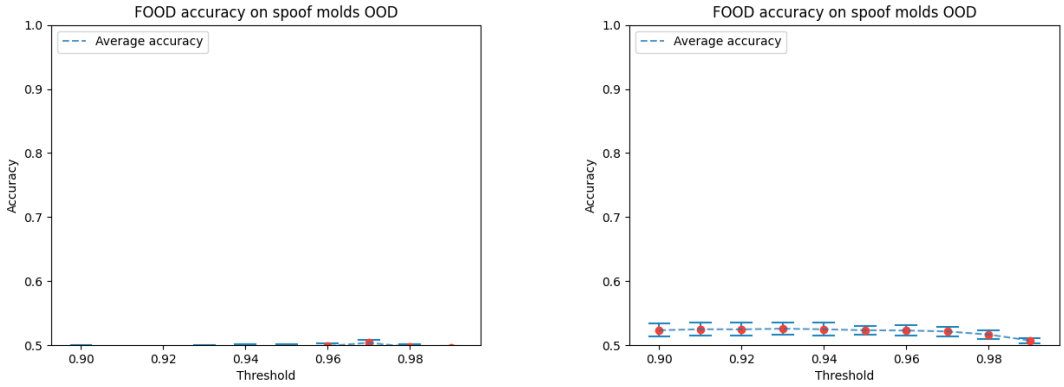
(b) TNR95 with additional layers starting at the end of the network.

Figure 4.4: Gram Matrix method performance (TNR95) on the generated dataset depending on how many layers are selected.

We can see that the first two layers selected are the reason for almost all of the accuracy from the Gram Matrix method, and adding the additional layers have very little effect on the result.

4.2.2 Spoof Molds

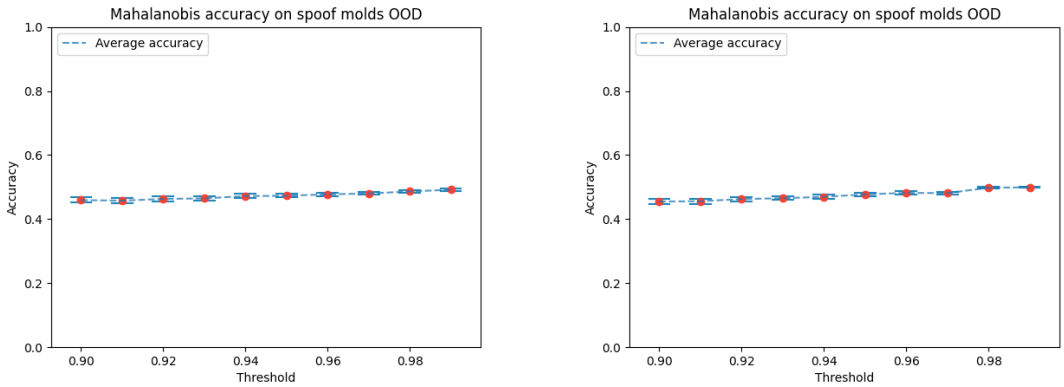
The performance of FOOD for the two different models on the spoof molds dataset can be seen in Figure 4.5



(a) Model A* (trained) **(b)** Model A

Figure 4.5: FOOD accuracy on the spoof molds dataset.

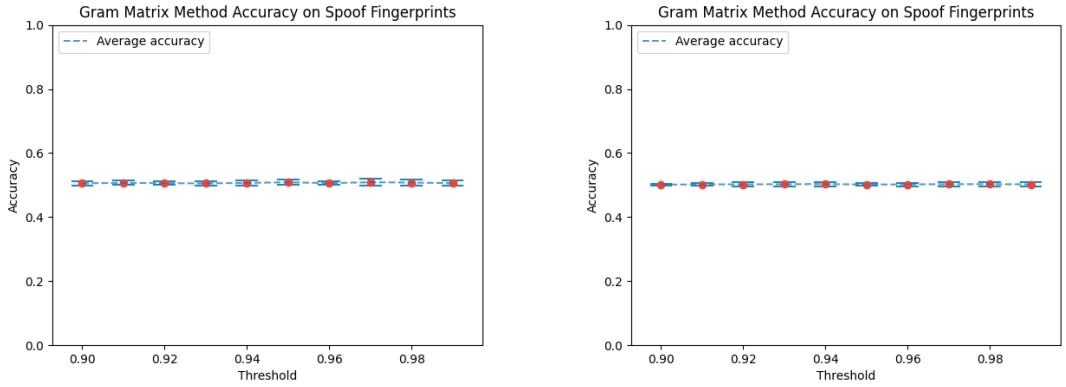
We can see that the untrained model performs slightly better than the trained model, reaching a TNR95 of 7.8%. The performance of the Mahalanobis distance method for the two different models on the spoof molds dataset can be seen in Figure 4.6.



(a) Model A* (trained) **(b)** Model A

Figure 4.6: Mahalanobis accuracy on the spoof molds dataset.

Again, the untrained model appears to perform slightly better, though the difference is very slight and both models perform poorly. The performance of the Gram Matrix distance for the two different models on the spoof molds dataset can be seen in Figure 4.7.



(a) Model A* (trained)

(b) Model A

Figure 4.7: Gram Matrix method accuracy on the spoof molds dataset.

Since the result is similar to that of a random classifier it does not vary with layers for the Gram Matrix method and such plots are therefore left out.

The results on the molds dataset from the two different models can be seen in Table 4.3.

Table 4.3: Averaged result (TNR95) for model A and A* on the molds dataset.

Model	Mahalanobis	FOOD	Gram Matrix
A	5.2%	**6.2**%	5.7%
A*	4.3%	4.7%	**5.1**%

4.2.3 Summary

The results on all the datasets from model A are summarized in table 4.4.

Table 4.4: Averaged result (TNR95) for the different methods on the different datasets for model A.

Dataset	Mahalanobis	FOOD	Gram Matrix
Nonfinger OOD	**99.7**%	93.4%	93.6%
Generated OOD	11.7%	6.1%	**14.3**%
Molds OOD	5.2%	**6.2**%	5.7%

Chapter 5

Discussion

From looking at the summary Table 4.4 we can see that all methods performs well on the more easier task of the Nonfinger dataset. This result is in accordance with the literature, where the methods tend to be applied on far-OOD samples. On the more difficult datasets all of the methods struggle. Interestingly, the methods looking further into the network, i.e. the Mahalanobis and Gram Matrix methods, are able to reach an accuracy a meaningful amount over that of a random classifier on the generated dataset. On the more difficult molds dataset none of the methods are able to perform much above that of a random classifier. Doing a simple threshold on the value of the final layer would give a lot worse result, see Figure 3.2, where it is noticeable that many of the nonfinger samples got classified as real images of fingerprints.

The results on the near-OOD samples are significantly worse, and none of the methods are able to significantly distinguish the dataset of spoof molds from the ID set. However, the performance on the generated dataset is significantly better, and the best methods performance (TNR95) is 14.3%. It is difficult to tell why the Mahalanobis and Gram matrix method are able to detect more of these as OOD as compared to the FOOD network. Intuitively it might be expected that the early layers of the network would correspond more closely to the pixels in an image, whereas the features later in the network would tell us more about whether the fingerprint is a spoof or not, the "spoofiness". This seems to make sense when looking at the results on the spoof molds dataset, where FOOD performs the best, which only looks at the penultimate layer. However, for the generated data there is something that the Gram Matrix method is able to pick up at the early layers of the network, see Figure 4.4, that FOOD is not able to see. Considering the network detects all the generated images between the real and spoof distributions, see Figure 3.2, it is expected that this dataset would be more difficult to classify as OOD.

The model that was trained on the molds dataset got slightly lower OOD detection scores

for all methods, despite the molds dataset not being selected as ID for the purpose of training the OOD detection model. This may be because in the network trained on the molds A* the "easy" ones, that the network are able to notice, are correctly classified as spoof images so the OOD detection method will not pick up on those, whereas for the untrained model A the OOD detection method may be able to see that something is off with these samples resulting in higher OOD detection scores.

5.1 Implementation

In this section reasoning for implementation details will be discussed, and suggestions for improvements will be given where applicable.

5.1.1 Mahalanobis Distance

The Mahalanobis Distance method showed very promising results on the nonfinger dataset classifying almost all OOD samples correctly as such. The method is non-invasive and does not require any form of retraining of the model. Selecting a different amount of different layers would however likely affect results, and a deeper study of how that choice affects the result could improve performance. It is unlikely that experimenting with changing the perturbation size ϵ for generating the adversarial samples would have much effect, as the method is claimed to be rigorous to such changes [1].

The possibility of using PCA for each layer representation could be investigated. This would however be a more computationally costly undertaking as the deeper layer representations are of much higher dimensionality than the penultimate layer, where this method showed improvements for FOOD. It is also not entirely obvious how to select the number of principal components, and this issue would be much further highlighted in this scenario. Explaining the variance through the network in the traditional sense would perhaps not be what we are looking for, as this would only explain what makes a sample be classified as a spoof or real image, whereas we are interested in whether it is out-of-distribution.

5.1.2 FOOD

The result of the FOOD method was highly improved with the addition of PCA, but it still struggles to compete with the more computationally heavy alternatives on the nonfinger dataset. Experimenting with finetuning the Gaussian layer for longer with different hyperparameters may have some effects on the result, but because of the way the layer is initialized it is thought to be quite small.

Overall, the performance of FOOD is quite satisfactory as the amount of data the method requires is small, considering we're only looking at the penultimate layer and doing computationally reasonable calculations from that, as compared to the other methods that do significantly heavier calculations [1].

5.1.3 Gram Matrix

The performance of the Gram Matrix method can be seen to be heavily impacted by the selection of layers, see Figure 4.2 and 4.4. Therefore, performance could likely be improved by selecting a different amount of layers. However, adding layers would increase the inference speed, for which the Gram Matrix method already performs the poorest out of the studied methods [1]. Because the performance on the current datasets can be seen to come from the early layers in the network, it would make sense to attempt adding more layers there.

5.1.4 Selecting the Threshold

The threshold can be adjusted according to a specific user as the fingerprint detector gets more and more images to work with. Since, in a real world scenario, we cannot expect the user to put in a lot of images before wanting the detector to work it is likely not possible to only use the fingerprint images from a specific user. We may however fine-tune the detector to set the boundary for the ID more in line with the specific user. For the FOOD and Mahalanobis distance methods we can easily adjust the threshold such that we take the new images into account. However, the new images are in some sense "better" than the ones we have previously possessed as these images are of the actual person using the detector rather than random fingerprints. Therefore, it makes sense to weigh these fingerprints more heavily. This could result in slightly improved detection rates, as we are able to decrease the variance of the in-distribution fingerprints, so we may end up with a better bound for what to consider OOD. However, since these changes are likely small, we may run into risks with too many real fingerprints being detected as OOD, particularly since also ID fingerprints can vary within themselves depending on their condition.

Chapter 6

Conclusion

The sophisticated methods for out-of-distribution detection were tested on far- and near-OOD datasets for fingerprint spoof detection. The performance of all the methods were much lower on the near-OOD datasets, while the performance on the far-OOD datasets were in line with the literature. The Mahalanobis Distance method, as well as the Gram Matrix method were able to perform meaningfully above that of a random classifier on a generated set of fingerprint images from the ID distribution, but the performance was drastically reduced from that on the far-OOD dataset.

Future work may look at other methods for OOD detection not considered here, or at improvements possible for the methods considered. Particularly, it would be interesting to study usage of PCA for the Mahalanobis distance method, and to look at a more varied set of near-OOD datasets.

References

[1] Guy Amit et al. "FOOD: Fast Out-Of-Distribution Detector". In: *2021 International Joint Conference on Neural Networks (IJCNN)* (2021), pp. 1–8.

[2] Daniel Berrar. "Cross-Validation". In: *Encyclopedia of Bioinformatics and Computational Biology*. Ed. by Shoba Ranganathan et al. Oxford: Academic Press, 2019, pp. 542–545. ISBN: 978-0-12-811432-2. DOI: https://doi.org/10.1016/B978-0-12-809633-8.20349-X.

[3] Peng Cui and Jinjia Wang. "Out-of-Distribution (OOD) Detection Based on Deep Learning: A Review". In: *Electronics* 11.21 (2022). ISSN: 2079-9292. DOI: 10.3390/electronics11213500.

[4] Tom Fawcett. "An introduction to ROC analysis". In: *Pattern Recognition Letters* 27.8 (2006). ROC Analysis in Pattern Recognition, pp. 861–874. ISSN: 0167-8655. DOI: https://doi.org/10.1016/j.patrec.2005.10.010.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org [Accessed 2023-06-05]. 2016.

[6] Dan Hendrycks and Kevin Gimpel. "A Baseline for Detecting Misclassified and Out-of-Distribution Examples in Neural Networks". In: (Oct. 2016).

[7] Dan Hendrycks, Mantas Mazeika, and Thomas Dietterich. *Deep Anomaly Detection with Outlier Exposure*. 2018. DOI: 10.48550/ARXIV.1812.04606.

[8] Ian Jolliffe and Jorge Cadima. "Principal component analysis: A review and recent developments". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 374 (Apr. 2016), p. 20150202. DOI: 10.1098/rsta.2015.0202.

[9] Ajay Kulkarni, Deri Chong, and Feras A. Batarseh. "5 - Foundations of data imbalance and solutions for a data democracy". In: *Data Democracy*. Ed. by Feras A. Batarseh and Ruixin Yang. Academic Press, 2020, pp. 83–106. ISBN: 978-0-12-818366-3. DOI: https://doi.org/10.1016/B978-0-12-818366-3.00005-8.

[10] Kimin Lee et al. "A Simple Unified Framework for Detecting Out-of-Distribution Samples and Adversarial Attacks". In: *Advances in Neural Information Processing Systems*. Ed. by S. Bengio et al. Vol. 31. Curran Associates, Inc., 2018.

[11] Prasanta Chandra Mahalanobis. “On the generalized distance in statistics”. In: *Proceedings of the National Institute of Sciences (Calcutta)* 2 (1936), pp. 49–55.

[12] Annette Molinaro, Richard Simon, and Ruth Pfeiffer. “Prediction error estimation: A comparison of resampling methods”. In: *Bioinformatics (Oxford, England)* 21 (Sept. 2005), pp. 3301–7. DOI: 10.1093/bioinformatics/bti499.

[13] M. Ohlsson and Patrik. Edén. *Introduction to Artificial Neural Networks and Deep Learning*. 2021.

[14] A. Papoulis and S.U. Pillai. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill series in electrical engineering: Communications and signal processing. Tata McGraw-Hill, 2002. ISBN: 9780070486584.

[15] Jie Ren et al. *A Simple Fix to Mahalanobis Distance for Improving Near-OOD Detection*. June 2021.

[16] Chandramouli Shama Sastry and Sageev Oore. “Detecting out-of-distribution examples with in-distribution examples and gram matrices”. In: 2019, arXiv–1912.

[17] Diaa M. Uliyan, Somayeh Sadeghi, and Hamid A. Jalab. “Anti-spoofing method for fingerprint recognition using patch based deep learning machine”. In: *Engineering Science and Technology, an International Journal* 23.2 (2020), pp. 264–273. ISSN: 2215-0986. DOI: https://doi.org/10.1016/j.jestch.2019.06.005.

[18] Jingkang Yang et al. “Generalized Out-of-Distribution Detection: A Survey”. In: *arXiv preprint arXiv:2110.11334* (Oct. 2021).

Master's Theses in Mathematical Sciences 2023:E25
ISSN 1404-6342

LUTFMA-3503-2023

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lth.se/