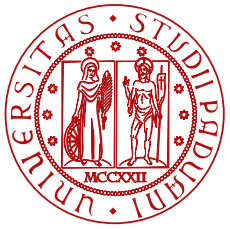


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in Computer Engineering

Error Cause Analysis of Laboratory Results with the Help of AI

February 27, 2023

Master Candidate

Andrea Matteazzi

Student ID 2010655

Supervisor

Prof. Gian Antonio Susto

University of Padova

External Supervisors

Sandra Mack

Infineon

Dr. Anja Zernig

KAI

Academic Year
2022/2023

To my family
and friends

Abstract

In the electronics laboratory, a large amount of different devices need to be tested, and the characterization of each of them generates a large amount of data. Classical root cause analysis is inherently inefficient because it requires manual inspection by experts, turning out to be costly and time consuming.
Furthermore, automatic evaluations through sequences of conditions for the signals, ending up in hard-coded logical formulas, are still inappropriate. This is due to the further necessity for experts, in order to design such formulas specifically for each different product, and to the complexity of the data itself, which may lead to the infeasibility of such approach.
For these reasons, in this thesis, first steps towards a machine learning (ML) approach are investigated, laying the foundation into the transition to a ML root cause analysis approach.

Contents

List of Figures **xi**

List of Tables **xiii**

List of Acronyms **xix**

1 Introduction **1**
1.1 Motivation 1
1.1.1 Industrial Problem 1
1.1.2 Understanding the Data 3
1.2 Research Questions 5

2 Machine Learning Theory **7**
2.1 Machine Learning 7
2.1.1 Supervised and Unsupervised Learning 7
2.1.2 Learning Algorithm and Cost Function 8
2.1.3 Parameters and Hyperparameters 9
2.1.4 Data Split and Generalization Error 10
2.1.5 Overfitting and Underfitting 11
2.2 Anomaly detection 13
2.3 Performance Measures 18
2.3.1 Confusion Matrix 18
2.3.2 F1 Score 20
2.3.3 Matthews Correlation Coefficient 21
2.4 PCA 22
2.5 DBSCAN 24
2.6 Isolation Forest 27
2.7 K Nearest Neighbors 30

2.8 Artificial Neural Network 31
2.8.1 Deep Feed-Forward Neural Network 32
2.8.2 Activation Function 35
2.8.3 Regularization 36
2.9 Convolutional Neural Network 38
2.9.1 Convolution 38
2.9.2 Convolutional and Locally Connected Layers 42
2.10 Autoencoder 42
2.10.1 Autoencoders in Anomaly Detection 43
2.10.2 Dense Autoencoder 46
2.10.3 Convolutional and Locally Connected Autoencoders . . . 46
2.10.4 Variational Autoencoder 47

3 Model Analysis and Experimental Results **51**
3.1 Unsupervised Approach 53
3.2 Semi-Supervised Approach 53

4 Conclusions and Future Works **67**

References **69**

List of Figures

1.1 Error cause analysis of different devices from a specific product . 1
1.2 Error cause analysis of different products 2
1.3 Case study device and testing procedure 3
1.4 Outcomes of the case study testing procedure 3
1.5 Example of two "Not suspicious" samples 4
1.6 Example of "Not suspicious" vs "Suspicious" samples obtained with same test parameters from two different DUTs 5
1.7 ML error cause analysis of different devices from a specific product 6
1.8 ML error cause analysis of different products 6

2.1 Training-validation-test split 10
2.2 Bias-variance trade off [15] 12
2.3 Underfitting and overfitting [15] 12
2.4 Anomalies [6] 13
2.5 Using classification for anomaly detection [6] 16
2.6 Confusion matrix in binary classification 19
2.7 PCA with n=2 23
2.8 Example of 3 sample datasets [12] 24
2.9 For $MinPts$ = 6. q = core point, p = border point [12] 25
2.10 For $MinPts$ = 6. p is directly density-reachable from q [12] 25
2.11 For $MinPts$ = 6. p is density-reachable from q [12] 26
2.12 For $MinPts$ = 6. p and q are density-connected each other [12] . . 26
2.13 Isolating points via iForest [20] 28
2.14 Averaged path lengths of x_i and x_o converge when the number of trees increases [20] 29
2.15 Example of KNN classification [28] 30
2.16 Artificial neural network 31

2.17 Unit of an NN . . . 32
2.18 An intuitive, geometric explanation of the exponential advantage of deeper piecewise linear activation function networks [15] . . . 33
2.19 Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses [15] . . . 34
2.20 Logistic sigmoid [15] . . . 35
2.21 ReLU [15] . . . 36
2.22 Fully connected vs sparse connections . . . 37
2.23 Convolutional as sparse fully-connected NN [17] . . . 38
2.24 Convolution operation [15] . . . 40
2.25 Sparse interactions with a 3 size filter [15] . . . 40
2.26 Efficiency of convolution [15] . . . 41
2.27 Convolutional layer . . . 42
2.28 AE architecture [15] . . . 42
2.29 AE building blocks [15] . . . 43
2.30 AE in anomaly detection . . . 45
2.31 DAE architecture . . . 46
2.32 CAE architecture . . . 46
2.33 VAE architecture . . . 48
2.34 Stochastic AE [15] . . . 49

3.1 Preprocessing steps . . . 52
3.2 PCA of the preprocessed data . . . 54
3.3 Confusion matrix KNN . . . 55
3.4 Histogram MAE reconstruction error of DAE . . . 56
3.5 Confusion matrix DAE . . . 57
3.6 MAE of autoencoders based on convolutional and locally connected layers . . . 58
3.7 Confusion matrices of CAE and LCAE . . . 60
3.8 PCA of the hidden space of DAE . . . 61
3.9 PCA of the hidden space of DVAE . . . 62
3.10 KL divergence loss and MAE + KL divergence losses for DVAE and CVAE . . . 63
3.11 Confusion matrices of DVAE and CVAE . . . 65

List of Tables

3.1 Formula performances 51
3.2 iForest and DBSCAN performances 53
3.3 KNN performances 54
3.4 DAE performances 56
3.5 CAE and LCAE performances 59
3.6 DVAE and CVAE performances 64

List of Acronyms

ML	machine learning
DUT	device under test
MSE	mean squared error
TP	true positives
TN	true negatives
FP	false positives
FN	false negatives
ACC	accuracy
TPR	true positive rate
TNR	true negative rate
PPV	positive predictive value
NPV	negative predictive value
MCC	Matthews correlation coefficient
PCA	principal component analysis
DBSCAN	density based spatial clustering of applications with noise
iForest	isolation forest
KNN	k nearest neighbors
NN	neural network

FNN	feed-forward neural network
ReLU	rectified linear unit
CNN	convolutional neural network
AE	autoencoder
DL	deep learning
DAE	dense autoencoder
CAE	convolutional autoencoder
LCAE	locally connected autoencoder
VAE	variational autoencoder
KL	Kullback-Leibler
MAE	mean absolute error
DVAE	dense variational autoencoder
CVAE	convolutional variational autoencoder

1 Introduction

1.1 Motivation

1.1.1 Industrial Problem

In the electronics laboratory, devices from a specific product need to be inspected, whereas for each of them, multiple tests are performed in order to check the compliance and reliability with the requirements of such a product. For each product, an expert with background knowledge about the specific product is needed in order to evaluate each single test result. In particular, each single sample, i.e. each single test result per device, is labelled as "Not suspicious" if it satisfies the technical requirements, or as "Suspicious" otherwise. In Figure 1.1, different devices of a specific product are tested with a particular testing procedure and evaluated by an expert.

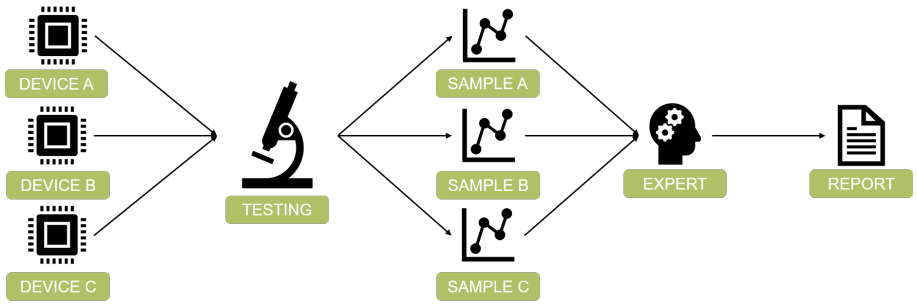


Figure 1.1: Error cause analysis of different devices from a specific product

While in Figure 1.2, different products are tested and evaluated by different experts, each of them with background knowledge about the specific product.

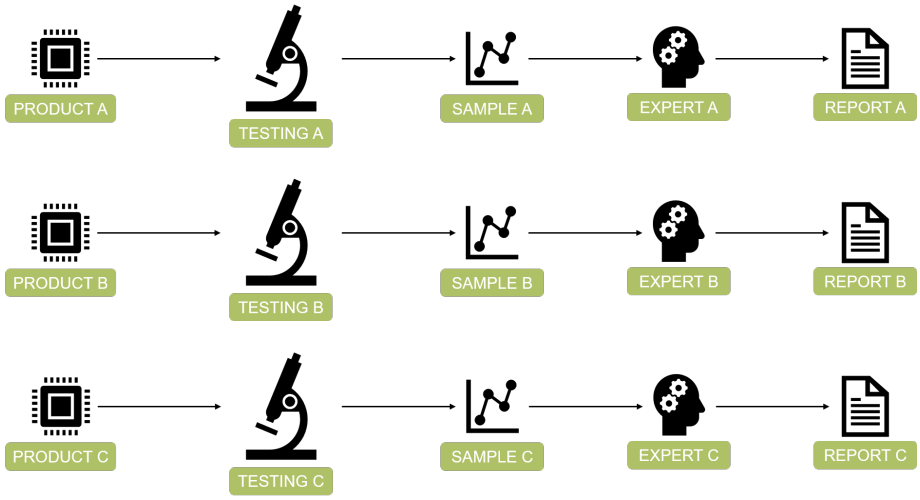


Figure 1.2: Error cause analysis of different products

Furthermore, since for each product type, a huge amount of devices need to be tested, such manual root cause analysis turns out to be infeasible and an automatic evaluation is needed instead. The state of the art in this automatic approach is given by a hard-coded logical formula, in which many product-specific measurement conditions are provided. However, there are intrinsic issues in the use of a formula because first of all, it has to be designed specifically by an expert with background knowledge about the product, secondly, due to the complexity of the devices and hence, of the data, it may be hard to design such a formula or it may have poor performances. Finally, since the formula aims to detect the "Suspicious" samples, it may be not robust to future anomalies of the devices, since it is designed based on the actual knowledge of the product and hence, the actual knowledge of the possible anomalies.

1.1.2 Understanding the Data

The work is developed on a particular product and in a particular testing procedure, resulting in a case study on a specific dataset.
Specifically, for each device under test (DUT) of the specific product, 156 measurements are performed by varying some parameters of the testing procedure, resulting in 156 different samples, each of them to be then labelled either as "Not suspicious" or "Suspicious" as shown in Figure 1.3.

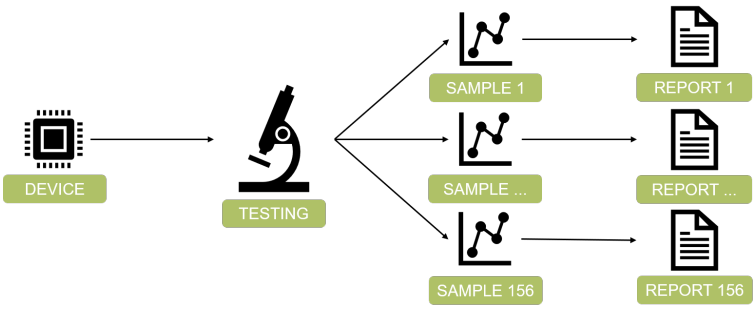


Figure 1.3: Case study device and testing procedure

Furthermore, in this particular root cause analysis, for each measurament of a DUT, two signals are inspected. From Figure 1.4, it is possible to see that a sample is indeed composed by two time series which correspond to signals: "IS voltage" and "OUT voltage", ending up in a dimension of 3646 x 2.

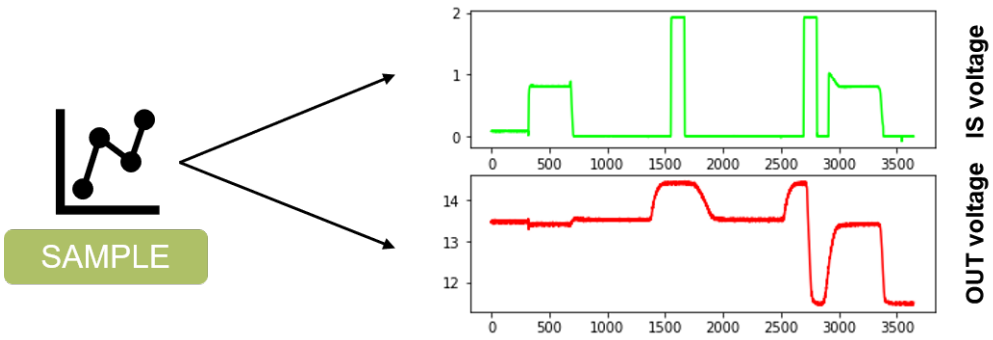


Figure 1.4: Outcomes of the case study testing procedure

However, since the time range of each sample is the same, it is possible to ignore the time dependence and treat all samples as vectors with one-to-one correspondence between entries. Then, for each DUT, the resulting 156 samples differ each other by mean, variance and waveform, for each of the two corresponding signals, as shown in Figure 1.5.

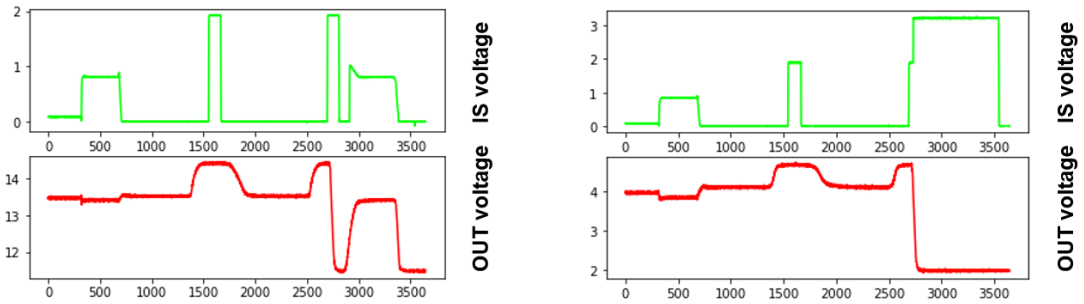


Figure 1.5: Example of two "Not suspicious" samples

Additionally, the patterns looked at in each sample are just intra-signal patterns, meaning that it would be possible even to split the two signals and analyze them separately. Another interesting characteristic of the data is that there is a one-to-one relation between each sample and the testing parameters of the testing procedure, through which such sample has been obtained. Indeed, a particular combination of the testing parameters corresponds to a particular combination of mean, variance and waveform of the sample's signals.
An important aspect is given by the testing procedure through which the samples are obtained. In fact, for each combination of the testing parameters, the resulting samples, corresponding to each DUT, are obtained in a systematic way, meaning that the behavior of such samples is expected to be the same for each timestamp, given that they are "Not suspicious".
Figure 1.6 shows an example of "Suspicious" sample. In fact, by looking at the "OUT voltage" plot on the right, it can be noticed that the waveform presents much less variations with respect to the one on the left. Further in this example, the patterns are just intra-signal and can be also noticed that a sample may be "Suspicious" just if one of the two signals is "Suspicious". Indeed, in this case, "IS voltage" has a normal behavior in both samples.

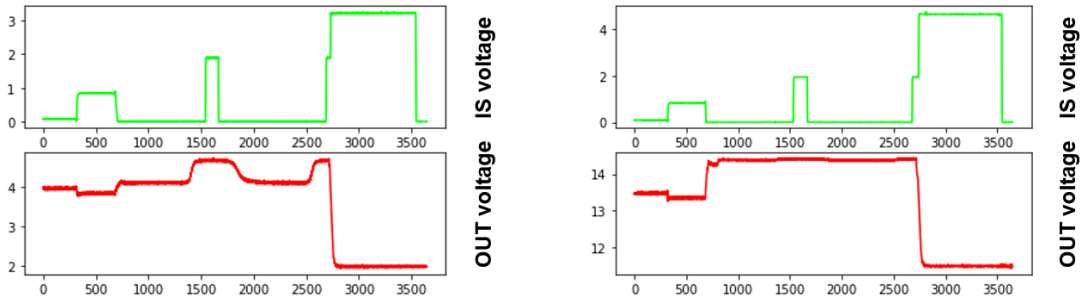


Figure 1.6: Example of "Not suspicious" vs "Suspicious" samples obtained with same test parameters from two different DUTs

The resulting dataset is composed by 50 DUTs, ending up in a dimension of 50 x 156 x 3646 x 2. Finally, it turns out to be unbalanced, with 5.8% of "Suspicious" samples.

1.2 Research Questions

From the previous chapter, it is evident that the hard-coded approach presents a lot of disadvantages. Therefore, the research questions of this thesis are formulated with the aim of bridging these pitfalls.

1) Can an ML approach be used?
   Figure 1.7 shows a possible ML error cause analysis pipeline which could substitute the one in Figure 1.1. The samples need a preprocessing step in order to be used as training data for the ML model, which will then evaluate future samples.

2) Can the formula's baseline performances be improved with an ML model?
   Indeed, the case study devices have been previously labelled with a product-specific formula providing the ground truth. A trained ML model can then be tested and compared with the formula's performances.

3) Can an ML approach be more robust to possible future anomalies?
   This question is very hard to answer because of the lack of knowledge of future anomalies but nevertheless, it is important to keep this question in mind during the choice of ML approaches.

4) Can the developed ML approach be generalized to other similar products?
   In Figure 1.8, the same data preprocessing and ML model architecture are used for different products, while, in Figure 1.2, different products are evaluated by different experts.

The content of the following chapters provides answers to these research questions.

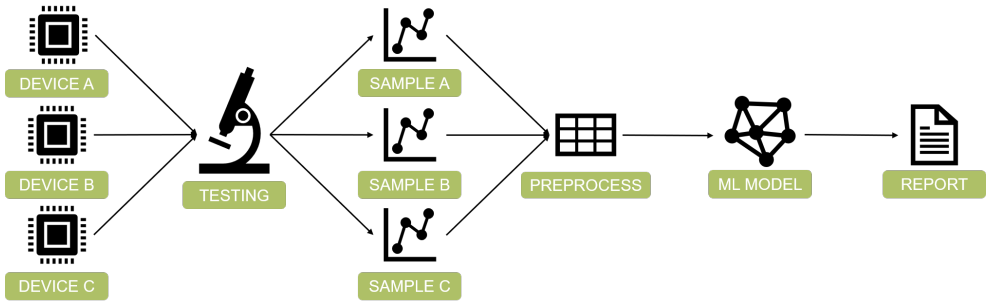


Figure 1.7: ML error cause analysis of different devices from a specific product

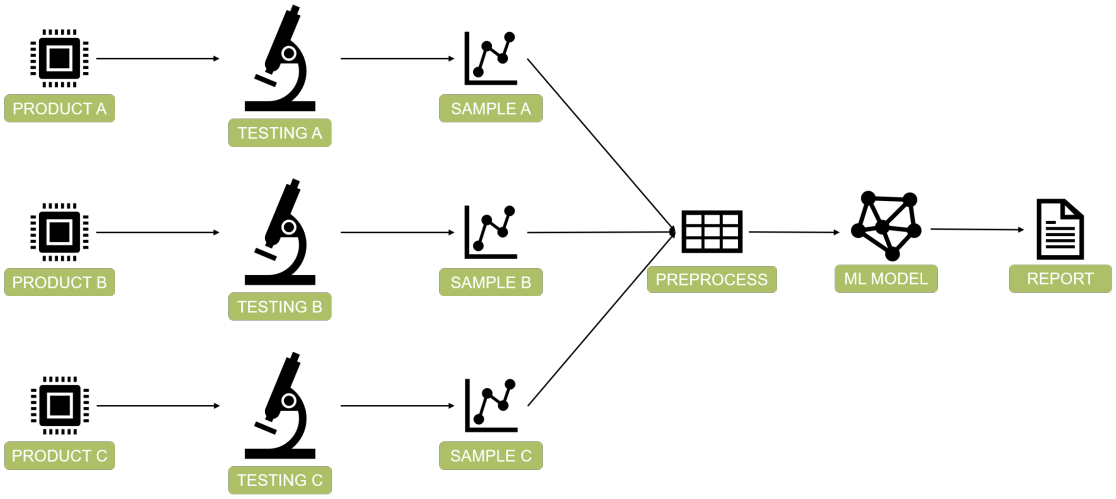


Figure 1.8: ML error cause analysis of different products

2

Machine Learning Theory

2.1 Machine Learning

"The field of study that gives computers the ability to learn without being explicitly programmed."
Arthur Samuel

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at task in T, as measured by P, improves with experience E."
Tom Mitchell

2.1.1 Supervised and Unsupervised Learning

Since learning involves an interaction between the learner and the environment, one can divide learning tasks according to the nature of that interaction. Such interaction is practically given by the experience E [26].
Viewing learning as a process of using experience to gain expertise, supervised learning describes a scenario in which the experience, a training example, contains significant information that is missing in the unseen test examples to which the learned expertise is to be applied. In this setting, the acquired expertise is aimed to predict that missing information for the test data. In such cases, we can think of the environment as a teacher that supervises the learner by providing the extra information (labels).

In unsupervised learning, however, there is no distinction between training and test data. The learner processes input data with the goal of coming up with some summary, or compressed version of that data.
In the middle of these two learning paradigms, there is semi-supervised learning. Such learning paradigm involves the use of both, labeled and unlabeled data. Hence, it is of great interest in machine learning and data mining because it directly uses available unlabeled data to improve supervised learning tasks when the labeled data are scarce or expensive.

2.1.2 Learning Algorithm and Cost Function

In the basic statistical learning setting [10][14], the learner has access to:

- Domain set X - set of all possible objects to make predictions about
- Label set Y - set of possible labels
- Training data S - learner's input, for supervised learning: $S \subseteq X \times Y$, for unsupervised learning: $S \subseteq X$
- Learner's output h - hypothesis $h : X \to Y$. The hypothesis h is produced by learning algorithm A when training set S is given to it: $h = A(S)$
- Data-generation model - instances $x \in X$ are generated by some probability distribution $x \sim D$ and labelled according to a function $f : X \to Y$ both not known to the learner
- Measure of success - probability the learner doesn't predict the correct label on random $x \sim D$, hence, a function $error_D(h)$

Each ML model is associated a hypothesis space H and learning means choosing an hypothesis $h \in H$. The aim of the learning algorithm is to pick the hypothesis $h \in H$ which minimizes the measure of success, however the learner doesn't know the data-generation model and has access just to $S \sim D^n$ where $n = |S|$. Therefore, in practice the learning algorithm cannot minimize the measure of success, also called generalization error, but it can just minimizes the so called empirical error, which is an error defined through a given cost function (or loss function) and based on S. The loss function is then a fundamental ingredients in ML, since based on its choice, it may correspond different hypothesis picked and such choice is task dependent.

2.1.3 Parameters and Hyperparameters

ML can be summarized as learning a function (a hypothesis) that maps instances of the domain set to output variables in the label set [10][26]. The form of the function is unknown and different ML models make different assumptions about the form of the function and how it can be learned. Assumptions can greatly simplify the learning process, but can also limit what can be learned.
A learning algorithm that simplifies the function to a known form and that summarizes data with a set of parameters of fixed size (independent of the number of training examples) is called parametric ML model. A parametric ML model defines a corresponding parametric function, whose parameters are then learned during the training, through the training set S, ending up in an hypothesis. On the other hand, learning algorithms which do not make particular assumptions about the kind of mapping function are known as non-parametric ML models. These algorithms do not accept a specific form of the mapping function between input and output data as true and they have the freedom to choose any functional form from the training data. In the case of non-parametric models, the number of parameters is dependent on the amount of training data.
In any case, the learning procedure involves the setting of parameters of the model, which therefore define the hypothesis picked in the hypothesis space H and whose final values depend then on the training set S.
However, an ML model may have some parameters, which define the space of possible hypothesis that the model may represent and hence, which can not be learned during training, called hyperparameters. These hyperparameters have then to be defined during the definition of the ML model itself as input parameters and they may affect the learning procedure and the choice of the final hypothesis, hence, their tuning needs particular attention since it may lead to over simplistic or over sophisticated hypothesis spaces.

2.1.4 Data Split and Generalization Error

As written in the previous section, the goal of a learning algorithm is the choice of a hypothesis which minimizes the generalization error $error_D(h)$, but in practice it can just pick a hypothesis which minimizes the empirical error $error_S(h)$ [26]. Moreover it turns out that:

$$error_D(h) = error_S(h) + generalization(h) \tag{2.1}$$

Then, it is hard to estimate the generalization error from the empirical error since h could be too much biased to the training set S. In order to derive a good estimate of the generalization error it is sufficient to compute the loss function with unseen data, that means, data not seen by the learning algorithm during the training step. Also, in order to solve a task, usually multiple ML models are tried and for each of them, multiple experiments varying the hyperparameters are performed. Therefore, some estimation of the generalization error is needed in order to choose the best ML model based on the resulting performances. Such process is called model selection. For these reasons, in machine learning it is common to perform the so called training-validation-test split in which, the original dataset is randomly partitioned into three sets in such a way, that most of the samples go in the training set and the remaining samples are equally partitioned in validation and test set. Hence, all the ML models are trained on the

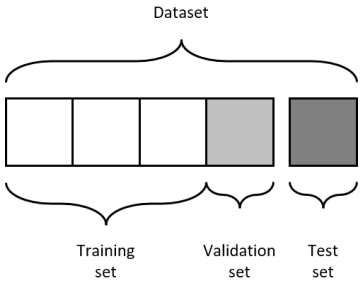


Figure 2.1: Training-validation-test split

training set and then, model selection is performed based on the validation set. Finally, since the chosen ML model has been picked among all the experimented ML models and with hyperparameters tuned based on the validation set, the validation error turns out to be biased. Even if based on unseen data during training, the validation error is not longer a good estimate of the generalization error and the test set is used for such estimate instead.

2.1.5 Overfitting and Underfitting

The previous chapter motivated the need of having a good estimate of the generalization error. It turns out that just considering the empirical loss may lead to phenomena of underfitting and overfitting [15]. Suppose that the data-generation model is composed by a function $f(x)$ such that $y = f(x) + \epsilon$, with $x \in X$, $y \in Y$ and ϵ normal noise with zero mean and variance σ^2. An ML model is then an estimator $\hat{f}(x, S)$ of $f(x)$ whereas $\hat{f}(x; S)$ is the hypothesis of the ML model trained with S. In order to measure the degree of approximation, the mean squared error (MSE) between $\hat{f}(x; S)$ and y can be computed as:

$$MSE = \mathbb{E}_{S,\epsilon}[(y - \hat{f}(x; S))^2] = (Bias_S[\hat{f}(x; S)])^2 + Var_S[\hat{f}(x; S)] + \sigma^2 \tag{2.2}$$

whereas:

- $Bias_S[\hat{f}(x, S)] = \mathbb{E}_S[\hat{f}(x; S)] - f(x)$ is the bias of the ML model
- $Var_S[\hat{f}(x; S)] = \mathbb{E}_S[(\mathbb{E}_S[\hat{f}(x; S)] - \hat{f}(x; S))^2]$ is the variance of the ML model
- σ^2 is the irreducible error

The bias term represents the error caused by the simplifying assumptions made by the model, the variance term instead represents the error related to the over-complexity of the model with respect to S.
Hence, the nature of these two terms is opposite. While too simple models lead to the incapacity to approximate the real function well (ending up in high bias and low variance, called underfitting), too complex models may lead to fit even the noise in the training set S (ending up in low bias and high variance, called overfitting).
The complexity of an ML model is given by the complexity of its hypothesis space. Therefore, through model selection, it is searched the best compromise by tuning the hyperparameters. Additionally, some form of regularization is usually performed, which is a way to limit the expressiveness of the models, often by adding to the loss function some terms, which penalizes more complex hypothesis.

Such compromise is called bias-variance trade off [23], Figure 2.2.

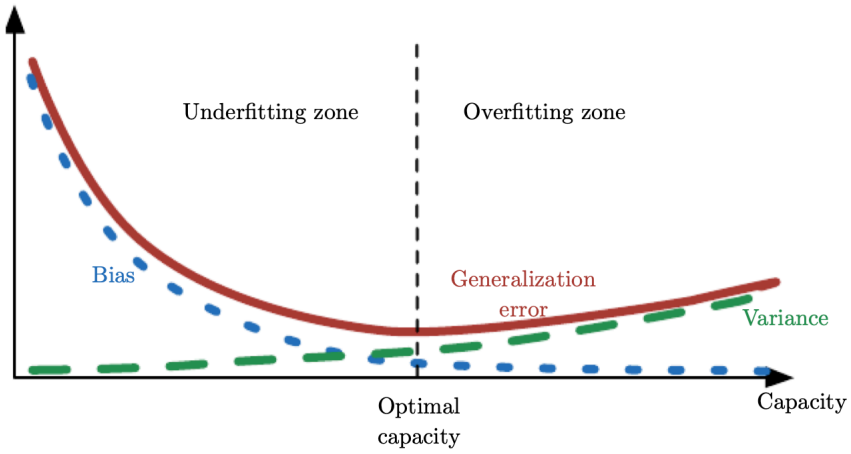


Figure 2.2: Bias-variance trade off [15]

In practice, to understand the complexity of the model, the training and validation errors are compared and it turns out that, if both of them are high, it means that the model is not complex enough and therefore, it is a case of underfitting. While, if the training error is low but the validation error is still high then, it is a case of overfitting, Figure 2.3.

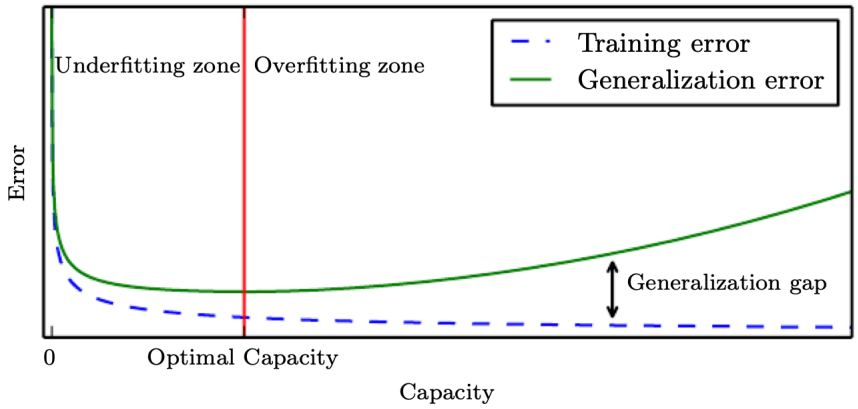


Figure 2.3: Underfitting and overfitting [15]

2.2 Anomaly detection

Anomaly detection refers to the problem of finding patterns in data that do not match the expected behavior [6]. Such patterns are often referred to as anomalies or outliers.

Figure 2.4 illustrates anomalies in a simple 2-dimensional data set. The data has two normal regions, N_1 and N_2, since most observations lie in these two regions. Points that are sufficiently far away from the regions, e.g., points o_1 and o_2, and points in region O_3, are anomalies.

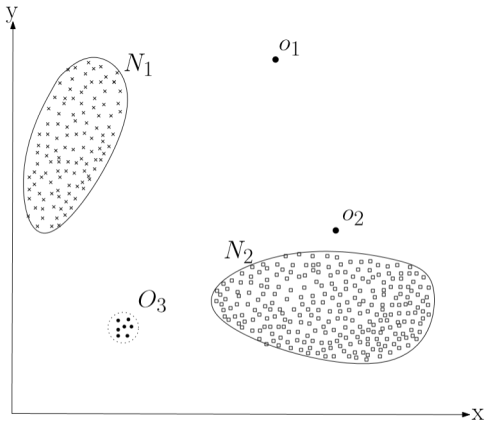


Figure 2.4: Anomalies [6]

At an abstract level, an anomaly is defined as a pattern that does not conform to expected normal behavior.

Therefore, a straightforward anomaly detection approach is to define a region representing normal behavior and declare any observation in the data which does not belong to this normal region as an anomaly.

Unfortunately, several factors make this apparently simple approach very challenging:

- Defining a normal region which encompasses every possible normal behavior is very difficult. In addition, the boundary between normal and anomalous behavior is often not precise. Thus, an anomalous observation, which lies close to the boundary, can actually be normal, and vice-versa
- When anomalies are the result of malicious actions, the malicious adversaries often adapt themselves to make the anomalous observations appear like normal, thereby, making the task of defining normal behavior more difficult
- In many domains, normal behavior keeps evolving and a current notion of normal behavior might not be sufficiently representative in the future

- The exact notion of an anomaly is different for different application domains. For example, in the medical domain a small deviation from normal (e.g., fluctuations in body temperature) might be an anomaly, while similar deviation in the stock market domain (e.g., fluctuations in the value of a stock) might be considered as normal. Thus, transferring a technique developed in one domain to another is not straightforward
- Availability of labeled data for training/validation for models used by anomaly detection techniques is usually a major issue
- Often, the data contains noise which tends to be similar to the actual anomalies and hence, is difficult to distinguish and remove

Due to the above challenges, the anomaly detection problem, in its most general form, is not easy to solve. In fact, most of the existing anomaly detection techniques solve a specific formulation of the problem. The formulation is induced by various factors such as nature of the data, availability of labeled data, type of anomalies to be detected, etc. Often, these factors are determined by the application domain in which the anomalies need to be detected.
The labels associated with a data instance denote if that instance is normal or anomalous. It should be noted that, obtaining labeled data which is accurate as well as representative for all types of behaviors, is often prohibitively expensive. Labeling is often done manually by a human expert and hence, requires substantial effort to obtain the labeled training data set. Typically, getting a labeled set of anomalous data instances which covers all possible type of anomalous behavior is more difficult than getting labels for normal behavior. Moreover, the anomalous behavior is often dynamic in nature, e.g., new types of anomalies might arise, for which there is no labeled training data available yet. In certain cases, such as air traffic safety, anomalous instances would translate to catastrophic events, and hence, will be very rare.

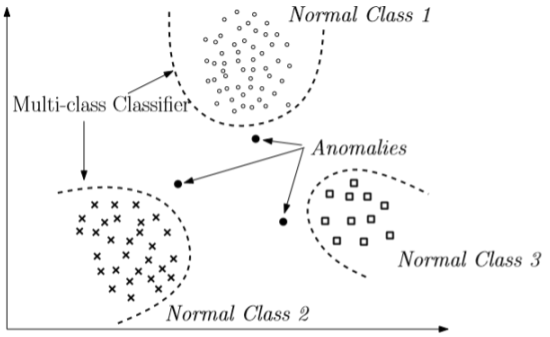
Based on the extent to which the labels are available, anomaly detection techniques can operate in one of the following three modes:

- Supervised anomaly detection - techniques trained in supervised mode assume the availability of a training data set, which has labeled instances for normal as well as anomaly class. Typical approach in such cases is to build a predictive model for normal vs anomaly classes. Any unseen data instance is compared against the model to determine which class it belongs to. There are two major issues that arise in supervised anomaly detection. First, the anomalous instances are far fewer compared to the normal instances in the training data. Issues that arise due to imbalanced class distributions. Second, obtaining accurate and representative labels, especially for the anomaly class is usually challenging. Other than these two issues, the supervised anomaly detection problem is similar to building predictive models
- Semi-supervised anomaly detection - techniques that operate in a semi-supervised mode, assume that the training data has labeled instances for only the normal class. Since they do not require labels for the anomaly class, they are more widely applicable than supervised techniques. The typical approach used in such techniques is to build a model for the class corresponding to normal behavior, and use the model to identify anomalies in the test data
- Unsupervised anomaly detection - techniques that operate in unsupervised mode do not require labeled training data, and thus are most widely applicable. The techniques in this category make the implicit assumption that normal instances are far more frequent than anomalies in the data. If this assumption is not true, then such techniques suffer from high false alarm rates. Many semi-supervised techniques can be adapted to operate in an unsupervised mode by using a sample of the unlabeled data set as training data. Such adaptation assumes that the training data contains very few anomalies and the model learnt during training is robust to these few anomalies

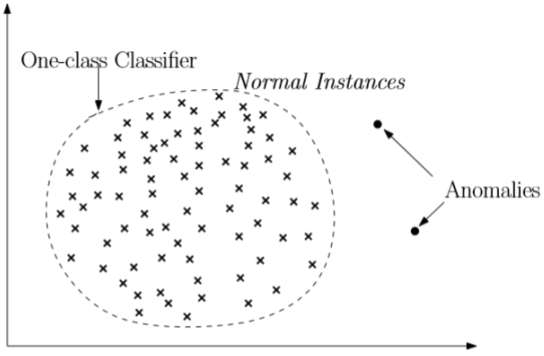
The focus of the majority of research activities on anomaly detection is about point anomalies. Such type of anomaly is present if an individual data instance can be considered as anomalous with respect to the rest of data.
For example, in Figure 2.4, points o_1 and o_2 as well as points in region O_3 lie outside the boundary of the normal regions, and hence, are point anomalies since they are different from normal data points. Furthermore, based on the assumptions made on the data, several anomaly detection techniques can be distinguished:

- Classification based anomaly detection techniques - classification is used to learn a model (classifier) from a set of labeled data instances (training) and then, classify a test instance into one of the classes using the learnt model

(testing). Classification based anomaly detection techniques operate in a similar two-phase fashion. The training phase learns a classifier using the available labeled training data. The testing phase classifies a test instance as normal or anomalous using the classifier. Classification based anomaly detection techniques operate under the following general assumption: a classifier that can distinguish between normal and anomalous classes can be learnt in the given feature space.
Based on the labels available during the training phase, classification based anomaly detection techniques can be grouped into two broad categories: multi-class and one-class anomaly detection techniques.
Multi-class classification based anomaly detection techniques, Figure 2.5a, assume that the training data contains labeled instances belonging to multiple normal classes. Such anomaly detection techniques learn a classifier to distinguish between each normal class against the rest of the classes. A test instance is considered anomalous if its not classified as normal by any of the classifiers. If none of the classifiers are confident in classifying the test instance as normal, the instance is declared to be anomalous.
One-class classification based anomaly detection techniques, Figure 2.5b, assume that all training instances have only one class label. Such techniques learn a discriminative boundary around the normal instances, using a one-class classification algorithm. Any test instance that does not fall within the learnt boundary is declared as anomalous



(a) Multi-class Anomaly Detection



(b) One-class Anomaly Detection

Figure 2.5: Using classification for anomaly detection [6]

- Nearest neighbor based anomaly detection techniques - such techniques are based on the following key assumption: normal data instances occur in dense neighborhoods, while anomalies occur far from their closest neighbors.
  Nearest neighbor based anomaly detection techniques require a distance or similarity measure defined between two data instances. Distance (or similarity) between two data instances can be computed in different ways and is typically required to be positive-definite and symmetric, but is not required to satisfy the triangle inequality.
  Nearest neighbor based anomaly detection techniques can be broadly grouped into techniques that use the distance of a data instance to its k^{th} nearest neighbor as the anomaly score and techniques that compute the relative density of each data instance to compute its anomaly score

- Clustering based anomaly detection techniques - clustering is used to group similar data instances into clusters. Clustering is primarily an unsupervised technique, though semi-supervised clustering has also been explored.
  Even though clustering and anomaly detection appear to be fundamentally different from each other, several clustering based anomaly detection techniques have been developed.
  Clustering based anomaly detection techniques can be grouped into three categories.
  First category of clustering based techniques rely on the following assumption: normal data instances belong to a cluster in the data, while anomalies do not belong to any cluster.
  Second category of clustering based techniques rely on the following assumption: normal data instances lie close to their closest cluster centroid, while anomalies are far away from their closest cluster centroid.
  Note that if the anomalies in the data form clusters by themselves, the above techniques will not be able to detect such anomalies. To address this issue a third category of clustering based techniques have been proposed that rely on the following assumption: normal data instances belong to large and dense clusters, while anomalies belong to small or sparse clusters

- Statistical anomaly detection techniques - statistical anomaly detection techniques are based on the following key assumption: normal data instances occur in high probability regions of a stochastic model, while anomalies occur in the low probability regions of the stochastic model.
  Statistical techniques fit a statistical model (usually for normal behavior) to the given data and then apply a statistical inference test to determine if an unseen instance belongs to this model or not. Instances that have a low probability to be generated from the learnt model, based on the applied test statistic, are declared as anomalies. Both parametric as well as non-parametric techniques have been applied to fit a statistical model. While parametric techniques assume that the underlying distribution is known and estimate the parameters from the given data, non-parametric techniques do not generally assume knowledge about the underlying distribution

- Spectral anomaly detection techniques - spectral techniques try to find an approximation of the data using a combination of attributes that capture the bulk of variability in the data. Such techniques are based on the following key assumption: data can be embedded into a lower dimensional subspace in which normal instances and anomalies appear significantly different.
  Thus, the general approach adopted by spectral anomaly detection techniques is to determine such subspaces (embeddings, projections, etc.) in which the anomalous instances can be easily identified. Such techniques can work in an unsupervised as well as semi-supervised setting

2.3 Performance Measures

Usually, the loss function used by a learning algorithm doesn't really correspond to the performance measure which one is interested in optimizing. Indeed, often such performance measure is not easy to directly optimize, and another performance measure is used instead. This alternative loss function is called surrogate loss function and it is an approximation of the original one, but easier to optimize and usually convex.
So, in practice, first an ML models is found, based on the surrogate loss, and then model selection and validation are performed, based on the original performance measure [10][26].
In this section, some performance measures are explored.

2.3.1 Confusion Matrix

In machine learning and specifically in the classification task, a confusion matrix is a specific table layout that allows visualization of the performance of an algorithm [13]. Each row of the matrix represents the instances in an actual class while each column represents the instances in a predicted class. Hence, taking the row i and the column j, the corresponding entry represents the number of instances with true label i predicted by an ML model as class j. Therefore, the diagonal of this matrix represents all the instances correctly classified by an ML model, while the other entries represents all the combinations of wrongly classified instances.
In the case of binary classification, where each instance X is mapped to one element of the set $\{p, n\}$ of positive and negative class labels, the corresponding confusion matrix is then a squared matrix of dimension 2, Figure 2.6.

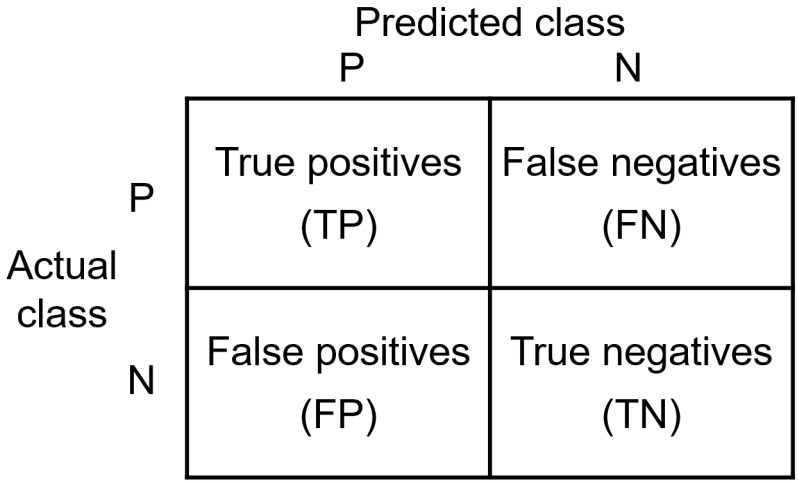


Figure 2.6: Confusion matrix in binary classification

In the particular case of binary classification, as it is also possible to see in Figure 2.6, the four entries of the confusion matrix have specific names:

- True positives (TP) - the number of positive instances correctly classified as positives
- True negatives (TN) - the number of negative instances correctly classified as negatives
- False positives (FP) - the number of false instances wrongly classified as positives
- False negatives (FN) - the number of positive instances wrongly classified as negatives

It is also possible to represent the confusion matrix in percentage, by normalizing each row of the matrix by the total number of instances of the corresponding actual class.
Starting from the confusion matrix for binary classification, several important metrics can be computed:

- Accuracy (ACC) - the proportion of correctly classified instances over all the instances, computed as follows:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.3}$$

- Recall or true positive rate (TPR) - the proportion of correctly classified positive instances over all the positive instances, computed as follows:

$$TPR = \frac{TP}{TP + FN} \tag{2.4}$$

- Specificity or true negative rate (TNR) - the proportion of correctly classified negative instances over all the negative instances, computed as follows:

$$TNR = \frac{TN}{TN + FP} \tag{2.5}$$

- Precision or positive predictive value (PPV) - the proportion of correctly classified positive instances over all the classified positive instances, computed as follows:

$$PPV = \frac{TP}{TP + FP} \tag{2.6}$$

- Negative predictive value (NPV) - the proportion of correctly classified negative instances over all the classified negative instances, computed as follows:

$$NPV = \frac{TN}{TN + FN} \tag{2.7}$$

2.3.2 F1 Score

Still in the scenario of binary classification, usually the positive instances are more relevant than the negative ones. Therefore it is possible to compare ML models based on the performance of just the positive instances. In particular, the two metrics which measure the performance of an ML model on positive instances are recall and precision, provided in Eq. (2.4) and Eq. (2.6), respectively. Since these two metrics are optimizing in opposite directions, that means that if you improve one, the other gets worse, some trade off is needed. Further, usually a unique metric is preferred in order to compare models, hence, the so called F1 score is introduced:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{2.8}$$

Basically, Eq. (2.8) is the harmonic mean between precision and recall and ranges in [0, 1] interval, whereas 0 indicates poor performances and 1 perfect precision and recall.
Equivalently, the F1 score can be defined for the negative instances by using specificity and negative predictive value instead, provided in Eq. (2.5) and Eq. (2.7), respectively.

2.3.3 Matthews Correlation Coefficient

Another interesting metric, to evaluate binary classifications and their confusion matrices, is the Matthews correlation coefficient (MCC) [8]:

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}} \tag{2.9}$$

MCC ranges in [-1, 1] interval, whereas -1 indicates poor performances, 1 perfect performances and 0 means that the binary result is not better than a random flip of a fair coin.
Accuracy and F1 score computed on confusion matrices have been (and still are) among the most popular adopted metrics in binary classification tasks. However, these statistical measures can dangerously show overoptimistic inflated results, especially on imbalanced datasets. MCC instead, is a more reliable statistical rate which produces a high score only if the prediction obtained good results in all of the four confusion matrix categories (true positives, false negatives, true negatives, and false positives), proportionally both to the size of positive elements and the size of negative elements in the dataset. In fact, when the dataset is unbalanced (the number of samples in one class is much larger than the number of samples in the other classes), accuracy cannot be considered a reliable measure anymore, because it provides an overoptimistic estimation of the classifier ability on the majority class.

2.4 PCA

Dimensionality reduction is the process of taking data in a high dimensional space and mapping it into a new space whose dimensionality is much smaller [26][10]. There are several reasons to reduce the dimensionality of the data. First, high dimensional data impose computational challenges. Moreover, in some situations high dimensionality might lead to poor generalization abilities of the learning algorithm. Finally, dimensionality reduction can be used for interpretability of the data, for finding meaningful structure of the data, and for illustration purposes.

Principal component analysis (PCA) is one of the most commonly used dimensionality reduction algorithm [18]. In PCA, both the compression and the recovery are performed by linear transformations and the method finds the linear transformations for which the differences between the recovered vectors and the original vectors are minimal in the least squared sense.

In particular, let $x_1, ..., x_m$ be m vectors in $\mathbb{R}^d$. A matrix $W \in \mathbb{R}^{n,d}$, where $n < d$, induces a mapping $x \mapsto Wx$ where $Wx \in \mathbb{R}^n$ is the lower dimensionality representation of x. Then, a second matrix $U \in \mathbb{R}^{d,n}$ can be used to approximately recover each original vector x from its compressed version. That is, for a compressed vector $y = Wx$, $y \in \mathbb{R}^n$, it is possible to reconstruct $\tilde{x} = Uy$, $\tilde{x} \in \mathbb{R}^d$.

In PCA, the compression matrix W and the recovering matrix U are found, so that the total squared distance between the original and recovered vectors is minimal:

$$\arg \min_{Wx \in \mathbb{R}^n, U \in \mathbb{R}^{d,n}} \sum_{i=1}^{n} ||x_i - UWx_i||^2 \tag{2.10}$$

Let $X \in \mathbb{R}^{m,d}$ the matrix whose ith row is x_i^T and $A = X^T X$, $A \in \mathbb{R}^{d,d}$. Then, being $u_1, ..., u_n$ the n the n eigenvectors corresponding to the n largest eigenvalues $\lambda_1, ..., \lambda_n$ of A, the solution to the PCA optimization problem given in Eq. (2.10) is to set U to be the matrix whose columns are $u_1, ..., u_n$ and to set $W = U^T$. Hence, the dimensionality reduction is performed as $Y = XW^T \in \mathbb{R}^{m,n}$.

PCA is defined as an orthogonal linear transformation that transforms the data to a new coordinate system, such that the greatest variance, by some scalar projection of the data, comes to lie on the first coordinate (called the first principal component), the second greatest variance on the second coordinate, and so on, Figure 2.7.

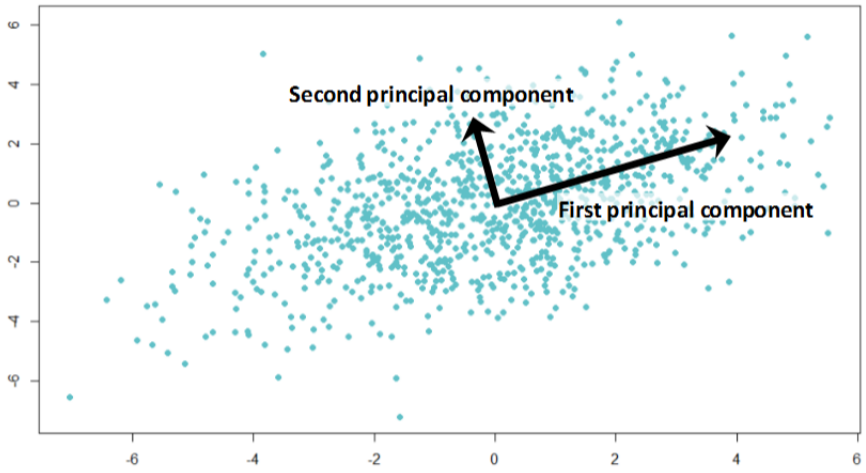


Figure 2.7: PCA with n=2

Since each eigenvalue corresponds to a particular dimension, and the higher its magnitude, the more is the variance associated to that dimension, by letting $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_d$, reducing the dimension to n means to pick the first n dimension of greater variance. To understand how much variance a particular direction explains and hence, how good a dimensionality reduction of dimension n approximates the original space of dimension d, the explained variance ratio can be used.
For the ith dimension, its explained variance ratio is:

$$\pi_i = \frac{\lambda_i}{\sum_{j=1}^{d} \lambda_j} \tag{2.11}$$

And by selecting the first n dimensions, the total explained variance ratio is:

$$\Pi_n = \sum_{j=1}^{n} \pi_j \tag{2.12}$$

2.5 DBSCAN

Density based spatial clustering of applications with noise (DBSCAN) is a density-based clustering non-parametric algorithm [26][10][12]. The assumption of a density-based clustering is that clusters are high-density regions separated by low-density regions.
When looking at the sample sets of points depicted in Figure 2.8, it can easily and unambiguously detect clusters of points and noise points not belonging to any of those clusters. The main reason why it is possible to recognize the clusters, is that within each cluster there are typical density of points, which is considerably higher than outside of the cluster. The key idea in DBSCAN is that, for each

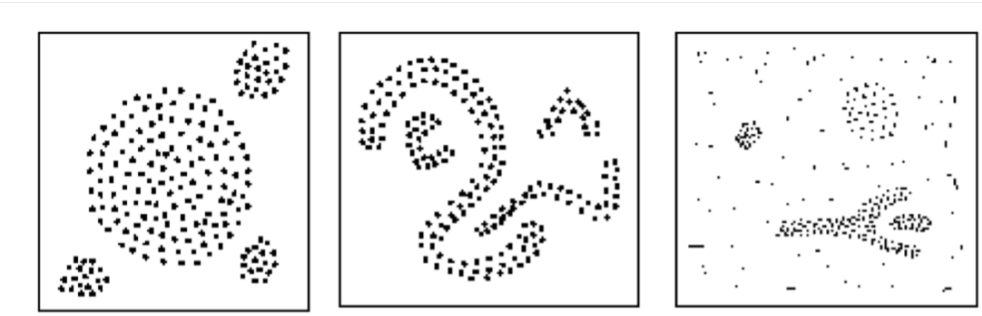


Figure 2.8: Example of 3 sample datasets [12]

point of a cluster, the neighborhood of a given radius ϵ has to contain at least a minimum number of points, i.e. the density in the neighborhood has to exceed some threshold. The shape of a neighborhood is determined by the choice of a distance function for two points p and q, denoted by $dist(p, q)$.
Given a dataset D, the ϵ-neighbourhood of a point p, denoted by $N_\epsilon(p)$, is defined by:

$$N_\epsilon(p) = \{q \in D \mid dist(p, q) \leq \epsilon\} \tag{2.13}$$

A naive approach could require, for each point in a cluster, that there are at least a minimum number($MinPts$) of points in an ϵ-neighborhood of that point. However, this approach fails because there are two kinds of points in a cluster, which are points inside of the cluster (core points) and points on the border of the cluster (border points). In general, an ϵ-neighborhood of a border point contains significantly less points than an ϵ-neighborhood of a core point, Figure 2.9. Therefore, one would have to set the minimum number of points to a relatively low value, in order to include all points belonging to the same cluster. This

value, however, will not be characteristic for the respective cluster, particularly in the presence of noise. Therefore, other requirements are needed. Defining a

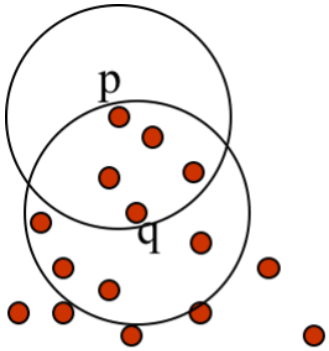


Figure 2.9: For $MinPts = 6$. q = core point, p = border point [12]

core point as a point q, such that condition (2.14) with respect to ϵ and $MinPts$ holds,

$$|N_\epsilon(q)| \geq MinPts \tag{2.14}$$

a point p is directly density-reachable from a point q with respect to ϵ and $MinPts$ if (2.15) holds.

$$\begin{aligned} &(1)\ p \in N_\epsilon(q) \\ &(2)\ |N_\epsilon(q)| \geq MinPts \end{aligned} \tag{2.15}$$

Directly density-reachability is not symmetric. In Figure 2.10, the q is not directly density-reachable from p, since p is border.

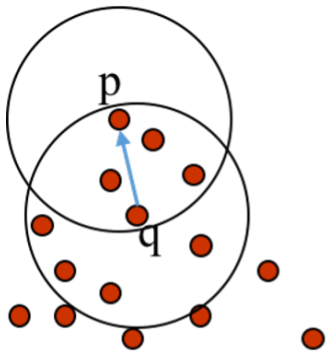


Figure 2.10: For $MinPts = 6$. p is directly density-reachable from q [12]

A point p is density-reachable from a point q with respect to ϵ and $MinPts$ if there is a chain of points q_i such that q_{i+1} is directly density-reachable from q_i, $i \in [1, nq]$, q_1 is directly density-reachable from q and p is directly density-reachable from q_{nq}, and q is a core point.

Density-reachability is not symmetric as well. In Figure 2.11, q is not directly density-reachable from p, since p is border.

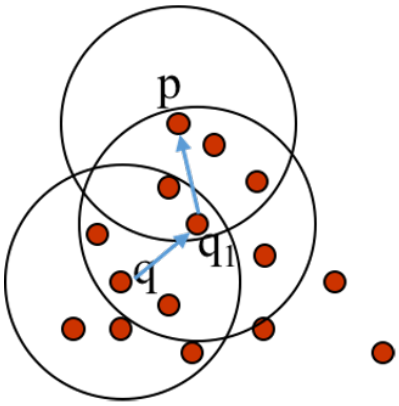


Figure 2.11: For $MinPts = 6$. p is density-reachable from q [12]

A point p is density-connected to point q if there is a point s such that p and q are density-reachable from s.
Density-connectivity is symmetric, Figure 2.12.

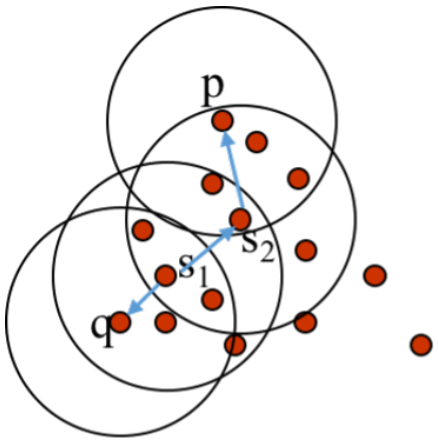


Figure 2.12: For $MinPts = 6$. p and q are density-connected each other [12]

Then in DBSCAN, a cluster is a maximal set of points density-connected. Border points, which are not connected by density to any core point, are labeled as noise.
DBSCAN is very sensitive to the values of ϵ and $MinPts$, decreasing ϵ and increasing $MinPts$ reduces the cluster size and increases the number of noise points.
Furthermore, DBSCAN doesn't require to specify the number of clusters to be returned as output (hyperparameter) and it can find clusters of any shape.

2.6 Isolation Forest

Most existing model-based approaches to anomaly detection, construct a profile of normal instances, then identify instances that do not conform to the normal profile as anomalies. Notable examples such as statistical methods, classification-based methods, and clustering-based methods, all use this general approach [6]. Two major drawbacks of this approach are:

- The anomaly detector is optimized to profile normal instances, but not optimized to detect anomalies, as a consequence, the results of anomaly detection might not be as good as expected, causing too many false alarms (having normal instances identified as anomalies) or too few anomalies being detected
- Many existing methods are constrained to low dimensional data and small data size because of their high computational complexity

Isolation forest (iForest) instead, explicitly isolates anomalies rather than profiles normal instances [20][27]. To achieve this, this method takes advantage of two anomalies quantitative properties:

- They are the minority consisting of fewer instances
- They have attribute-values that are very different from those of normal instances

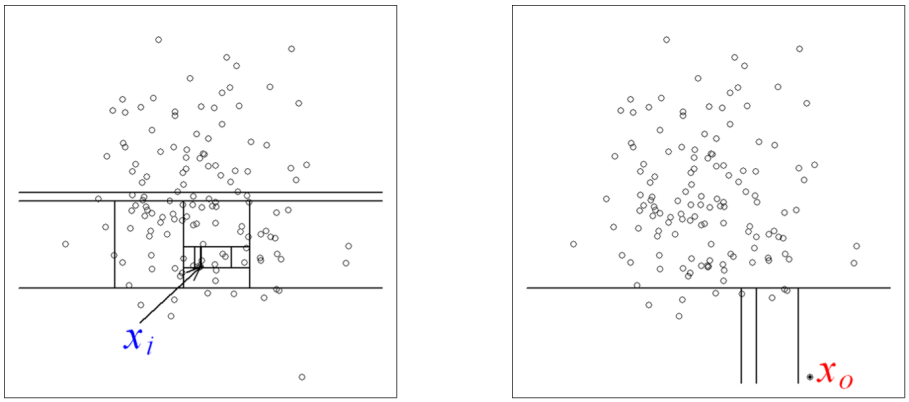
In other words, anomalies are few and different, which make them more susceptible to isolation than normal points. In particular, this method builds an ensemble of random trees for a given dataset (iTrees), then anomalies are those instances which have short average path lengths on the iTrees.
Apart from the key difference of isolation versus profiling, iForest distinguishes from existing model-based, distance-based and density-based methods in the following ways:

- The isolation characteristic of iTrees enables them to build partial models and exploit sub-sampling to an extent that is not feasible in existing methods. Since a large part of an iTree that isolates normal points is not needed for anomaly detection, it does not need to be constructed
- iForest utilizes no distance or density measures to detect anomalies. This eliminates major computational costs of distance calculation compared to all distance-based methods and density-based methods
- iForest has the capacity to scale up to handle extremely large data sizes and high-dimensional problems with a large number of irrelevant attributes

Isolation means separating an instance from the rest of the instances. Since anomalies are few and different, they are more susceptible to isolation. In a data-induced random tree, partitioning of instances is repeated recursively until all instances are isolated. This random partitioning produces noticeable shorter paths for anomalies since:

- The fewer instances of anomalies result in a smaller number of partitions and hence, shorter paths in a tree structure
- Instances with distinguishable attribute values are more likely to be separated in early partitioning. Hence, when a forest of random trees collectively produce shorter path lengths for some particular points, then they are highly likely to be anomalies

In Figures 2.13a and 2.13b, partitions are generated by randomly selecting an attribute and then, randomly selecting a split value between the maximum and minimum values of the selected attribute. Anomalies are more susceptible to isolation and hence have short path lengths. Given a Gaussian distribution (135 points), a normal point x_i requires twelve random partitions to be isolated; (b) an anomaly x_o requires only four partitions to be isolated.



(a) Isolating normal point x_i (b) Isolating anomalous point x_o

Figure 2.13: Isolating points via iForest [20]

From Figure 2.14, it can be seen that anomalies are more susceptible to isolation and hence have short path lengths. Given a Gaussian distribution (135 points), a normal point x_i, Figure 2.13a, requires twelve random partitions to be isolated; an anomaly x_o, Figure 2.13b, requires only four partitions to be isolated.

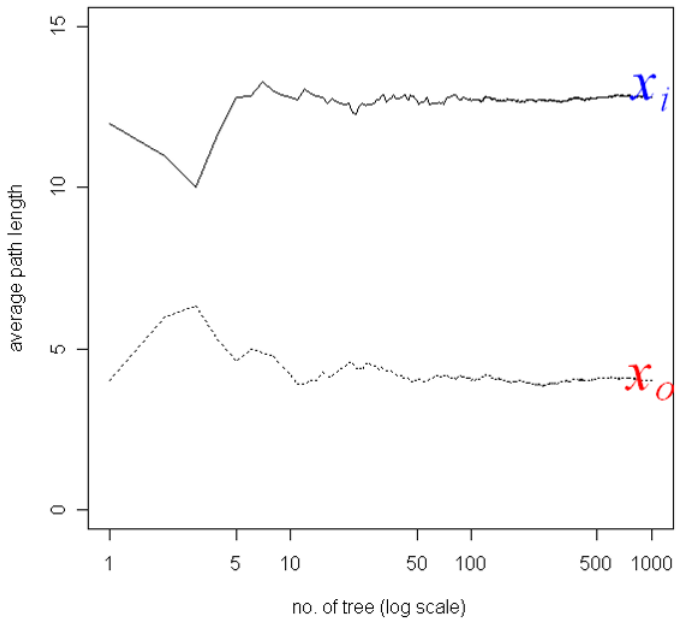


Figure 2.14: Averaged path lengths of x_i and x_o converge when the number of trees increases [20]

Since recursive partitioning can be represented by a tree structure, the number of partitions required to isolate a point, is equivalent to the path length from the root node to a terminating node. In this example, the path length of x_i is greater than the path length of x_o. Since each partition is randomly generated, individual trees are generated with different sets of partitions. It is possible to average path lengths over a number of trees to find the expected path length. Figure 2.14 shows that the average path lengths of x_o and x_i converge when the number of trees increases. Using 1000 trees, the average path lengths of x_o and x_i converge to 4.02 and 12.82, respectively. This shows that anomalies have path lengths shorter than normal instances.

Anomaly detection using iForest is a two-stage process. The first (training) stage builds isolation trees using sub-samples of the training set. The second (testing) stage passes the test instances through isolation trees, to obtain an anomaly score for each instance, based on the expected path length.

2.7 K Nearest Neighbors

K nearest neighbors (KNN) algorithm is a non-parametric classification method [9][25][28].
Given a training set $S \subseteq X \times Y$ of vectors in a multidimensional feature space, each with a class label, the training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, an unlabeled instance is classified by assigning the label which is most frequent among the k training samples nearest to that instance. In particular, k is a hyperparameter of the ML model. The word "nearest" implies a distance metric and its choice is another hyperparameter of the model. Typically, distances are measured with a Minkowski distance (L^p norm).
The Minkowski distance of two vectors x_j and x_q is defined as:

$$L^p(x_j, x_q) = (\sum_i |x_{j,i} - x_{q,i}|^p)^{\frac{1}{p}} \tag{2.16}$$

In Figure 2.15, the test sample (green dot) should be classified either to blue squares or to red triangles. If k = 3 (solid line circle) it is assigned to the red triangles because there are 2 triangles and only 1 square inside the inner circle. If k = 5 (dashed line circle) it is assigned to the blue squares (3 squares vs. 2 triangles inside the outer circle).

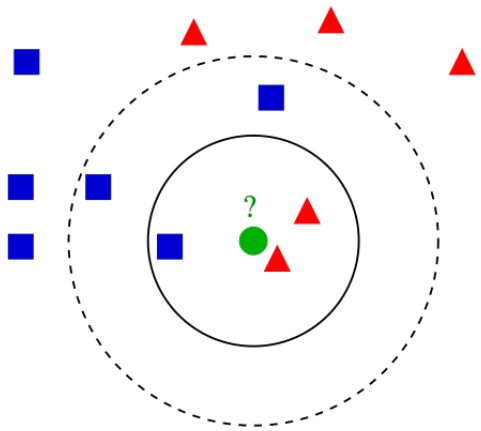


Figure 2.15: Example of KNN classification [28]

2.8 Artificial Neural Network

An artificial neural network, or simply neural network (NN), is a system consisting of interconnected units that compute nonlinear functions, called activation functions:

- Input units represent input variables and constitute the input layer
- Output units represent output variables and constitute the output layer
- Hidden units represent internal variables which codify, after learning, correlations among input and desired output variables and constitute one or more hidden layers

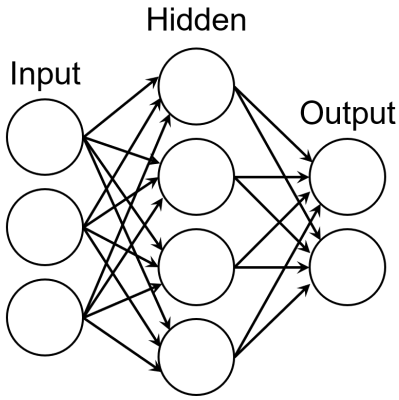


Figure 2.16: Artificial neural network

Artificial neural network is a parametric model whose parameters are the weights w associated to connections among units and bias b associated to each unit itself [15][16][25].

In Figure 2.17, it is possible to see a unit of the NN, where w_i and b represent the parameters of such unit, while the x_i represent the inputs of the unit. The output of the unit is then the output of a nonlinear function, f, with as input the linear combination of the parameters and inputs of the unit.

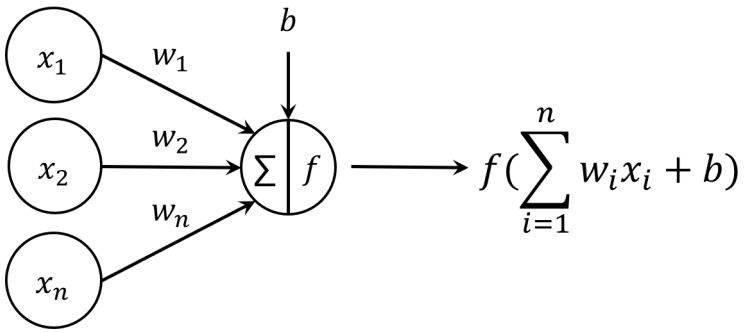


Figure 2.17: Unit of an NN

2.8.1 Deep Feed-Forward Neural Network

A feed-forward neural network (FNN), Figure 2.16, is a directed acyclic graph in which the connections are directed from input to hidden and from hidden to output units.
In this context, some architectural hyperparameters are important to state:

- Depth - the total number of layers in the FNN except input layer and hence, since the output layer is fixed, related to the number of hidden layers
- Width - the maximum number of units in a layer of a FNN
- Activation function - the activation function to use for the units of each layer

The use of this architecture introduces a bias in the hypothesis space. In fact the functions parametrized with this model are complex functions which are composition of simpler activation functions f.
The Figure 2.16 represents the simplest feed-forward (or dense) neural network, since it is composed by just one hidden layer.

Nevertheless, already for this architecture it holds the universal approximation theorem:

A feed-forward network with a linear output layer and at least one hidden layer with any squashing activation function, can approximate any continuous function and any function mapping from any finite dimensional discrete space to another, with any desirable amount of error, given enough hidden units.

This theorem guarantees that a network exists, however it doesn't guarantee that the training algorithm will be able to learn it. In fact, in the worst case, an exponential number of hidden units is required.
In this scenario, the use of deep FNN may reduce the number of units required to represent a function. Indeed, for example piecewise linear FNNs, can represent functions with a number of regions that is exponential in the depth of the network. Figure 2.18 illustrates how a network, with a particular piecewise linear activation function, creates mirror images of the function computed on top of some hidden unit, with respect to the input of that hidden unit. Each hidden unit specifies where to fold the input space in order to create mirror responses. By composing these folding operations, we obtain an exponentially large number of piecewise linear regions which can capture all kinds of regular (e.g., repeating) patterns.

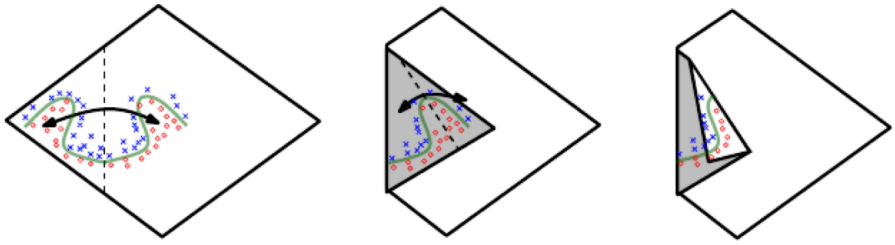


Figure 2.18: An intuitive, geometric explanation of the exponential advantage of deeper piecewise linear activation function networks [15]

Also, it turns out that empirically, a shallow (one hidden layer) neural network may overfit more than a deeper NN, Figure 2.19. Indeed, in this case the test set accuracy consistently increases with increasing depth.

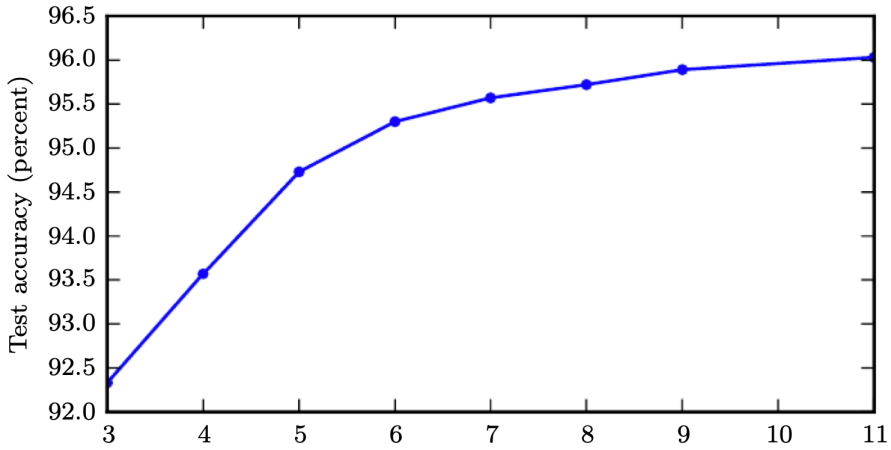


Figure 2.19: Empirical results showing that deeper networks generalize better when used to transcribe multi-digit numbers from photographs of addresses [15]

Any time a specific machine learning algorithm is chosen, there is an implicitly stating of some set of prior beliefs about what kind of function the algorithm should learn. Choosing a deep model encodes a very general belief that the function to learn should involve composition of several simpler functions. This can be interpreted, from a representation learning point of view, as the belief that the learning problem consists of discovering a set of underlying factors of variation, that can in turn be described in terms of other simpler underlying factors of variation.

2.8.2 Activation Function

From the universal approximation theorem and from Figure 2.18, it is evident the importance of the choice for the activation function. Furthermore, it is fundamental the choice of a nonlinear activation function. Indeed, it turns out that even the more complex deep FNN, with just linear activation functions, is equivalent to a shallow linear FNN. This is evident by the fact that an FNN is just a composition of functions, each corresponding to each of its layers. Hence, a composition of linear functions is equivalent to the simple linear function computed by a shallow linear FNN [15]. The choice of the activation function for the output layer instead, is tied to the task to solve and in particular, to the specific output $y \in Y$ of the problem.
A common choice in the past for the hidden activation functions was the logistic sigmoid (2.17), Figure 2.20.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.17}$$

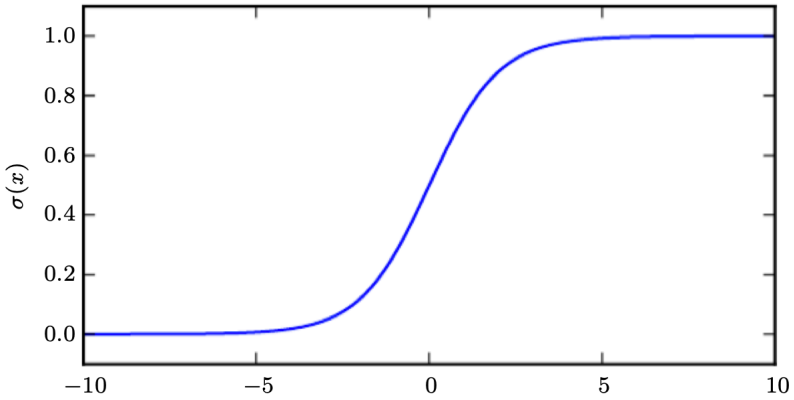


Figure 2.20: Logistic sigmoid [15]

However nowadays, for several optimization problems, it is unlikely used as hidden activation function but nevertheless, it is widely used as output activation function for modeling probability distributions. Indeed, a drawback of the universal approximation theorem is that neural networks are very complex models, whose associated parametric functions are a very complex nonlinear functions and hence, hard to optimize. Since that this complex functions are a composition of the hidden activation functions, it turns out that the use of piecewise activation functions lead to better performance because of the more guarantees associated to the use of convex optimization techniques.

In particular an extensively used function of this family is the rectified linear unit (ReLU), defined as (2.18), Figure 2.21.

$$g(z) = max\{0, z\} \tag{2.18}$$

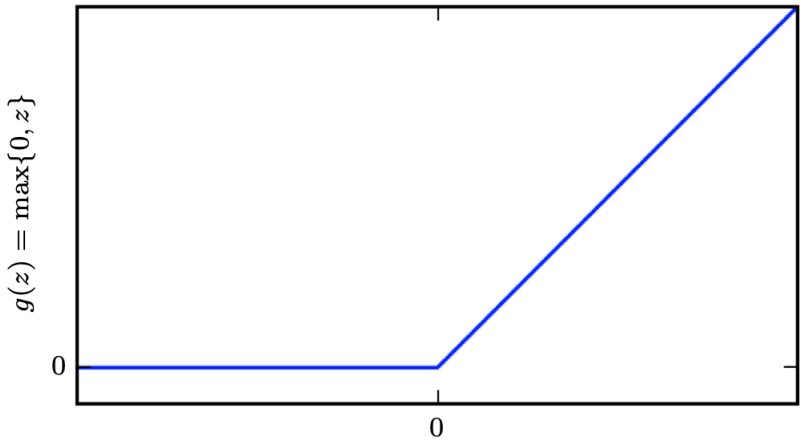


Figure 2.21: ReLU [15]

2.8.3 Regularization

Since the complexity of NNs, often the problem of overfitting arises. In fact, NNs have a large hypothesis space H associated and hence, it is very likely that the learning algorithm may find an hypothesis with high variance with respect to the training set. For these reasons, it is fundamental to limit the expressiveness of H by adding some regularization to the loss function [15]. In NNs, the hypothesis space is associated with the weights and biases of the NNs and adding a regularization, means to introduce some preference about the choice of the parameters of the network.
One of the oldest regularization techniques is the parameter norm penalties. Let be θ the set of parameters of the network and $J(\theta; S)$ the original loss function for the problem. Then the parameter norm penalties regularized loss function is:

$$\tilde{J}(\theta; S) = J(\theta; S) + \alpha\Omega(\theta) \tag{2.19}$$

whereas $\Omega(\theta)$ is a weight norm and α is a hyperparameter for the trade off between the optimization of the original loss and the parameter norm penalties, hence, for the bias and variance trade off.

This regularization, is also a form of the more general sparse representation regularization, which leads to sparse representation of the hidden space [17], Figure 2.22.

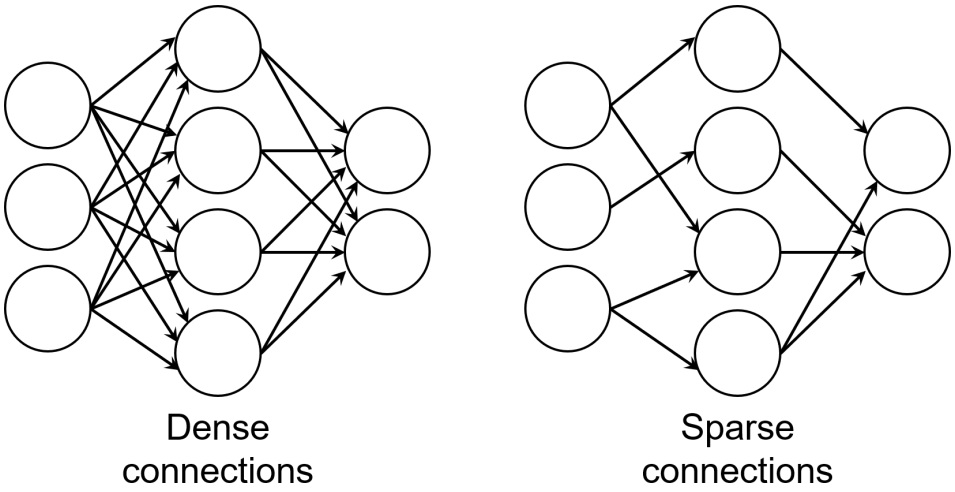


Figure 2.22: Fully connected vs sparse connections

Given h to be the hidden layers representation, that is, the set of all the hidden unit pre-activations (linear combinations for the hidden units, before the activation function is applied), the sparse representation regularization is:

$$\tilde{J}(\theta; S) = J(\theta; S) + \alpha\Omega(h) \tag{2.20}$$

Induce sparsity in the network indeed, it turns out to improve generalization and robustness but also to improve performance for inference and/or training. Another form of regularization is the early stopping. The idea underlying this approach is to return the model with the lowest validation loss. In particular, it stops the training if no improvement of the validation loss is obtained after N consecutive epochs of the training procedure, with N as hyperparameter. The regularization, in this case, is given by limiting the parameter space to a neighbourhood of the initial parameter values, that means, imposing a gaussian prior in the parameter space with respect to the initial values.

2.9 Convolutional Neural Network

As said previously, already a shallow FNN, in principle, has expressiveness enough to solve any problem [15]. However, in practice, it is hard to find a good hypothesis without overfitting. Regularization techniques may help by introducing some bias in the hypothesis space but still, for some problems this turns out to be not enough. Indeed, since the number of parameters of a feed-forward neural network depends by the dimension of the input feature space, this number may become huge for high dimensional problems.
Convolutional neural network (CNN) is an example of NN, in which some architectural constraints are introduced, in order to limit the expressiveness of the hypothesis space H in such a way to prefer the more problem-dependent hypothesis [11][15][17].
As shown in Figure 2.23, the convolution operator itself and its variants can be seen as a sparse version of fully connected layers. Instead of connecting every pair of neurons in the input and output layers, the connections are pruned in order to contain only local surroundings.

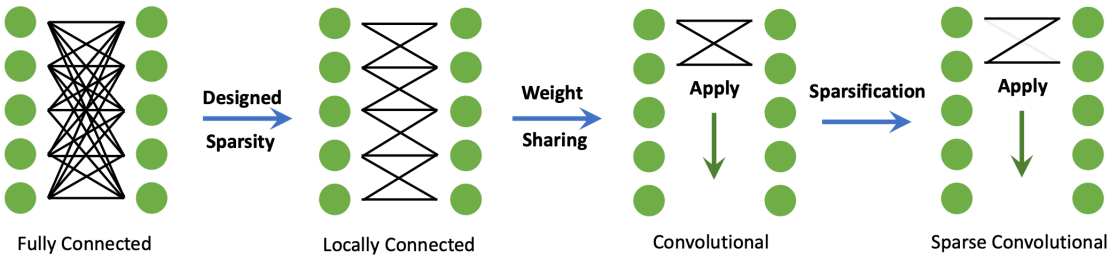


Figure 2.23: Convolutional as sparse fully-connected NN [17]

This architecture introduces a preference in learning local features of the input and hence, turns out to be state of the art in those problems whose input contains such features like image classification.

2.9.1 Convolution

The basic operation in dense neural networks is affine transformations: a vector is received as input and is multiplied with a matrix, to produce an output (to which a bias vector is usually added before passing the result through a non-linear function). This is applicable to any type of input, be it an image, a sound clip or an unordered collection of features. Whatever their dimensionality, their

representation can always be flattened into a vector before the transformation [11]. However, images, sound clips and many other similar kinds of data have an intrinsic structure. More formally, they share these important properties:

- They are stored as multi-dimensional arrays
- They feature one or more axes for which ordering matters (e.g., width and height axes for an image, time axis for a sound clip)
- One axis, called the channel axis, is used to access different views of the data (e.g., the red, green and blue channels of a color image, or the left and right channels of a stereo audio track)

These properties are not exploited when an affine transformation is applied; in fact, all the axes are treated in the same way and the topological information is not taken into account. Still, taking advantage of the implicit structure of the data may prove very handy in solving some tasks, like computer vision and speech recognition, and in these cases it would be best to preserve it. This is where discrete convolutions come into play. A discrete convolution is a linear transformation that preserves this notion of ordering.
For filter (or kernel) f and input x the convolution is defined as:

$$s(t) = (f * x)(t) = \sum_{a=-\infty}^{\infty} x(t-a)f(a) \tag{2.21}$$

Whereas $s(t)$ is called feature map.
In practice, the filter is defined to be zero everywhere except for a finite set of points and (2.21) for a m size filter becomes (2.22), Figure 2.24.

$$s(t) = (f * x)(t) = \sum_{a=0}^{m-1} x(t-a)f(a) \tag{2.22}$$

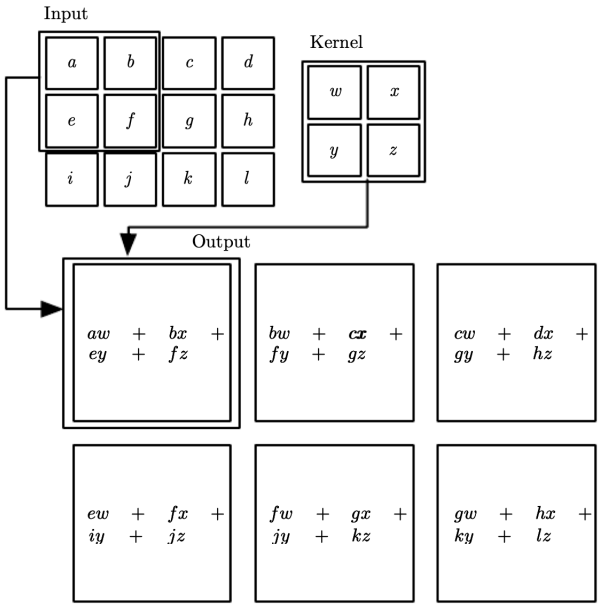


Figure 2.24: Convolution operation [15]

The main properties of convolution operator are:

- Sparse interactions. Achieved by using a filter smaller than the input. From Figure 2.25 indeed, it is possible to see that each single input feature contributes to just a portion on the size of the filter to the output. While each output depends by just a portion on the size of the filter to the input
- Parameter sharing. Each member of the filter is used at every position in the input. Instead of learning a separate set of parameters for each location, only one set is learnt

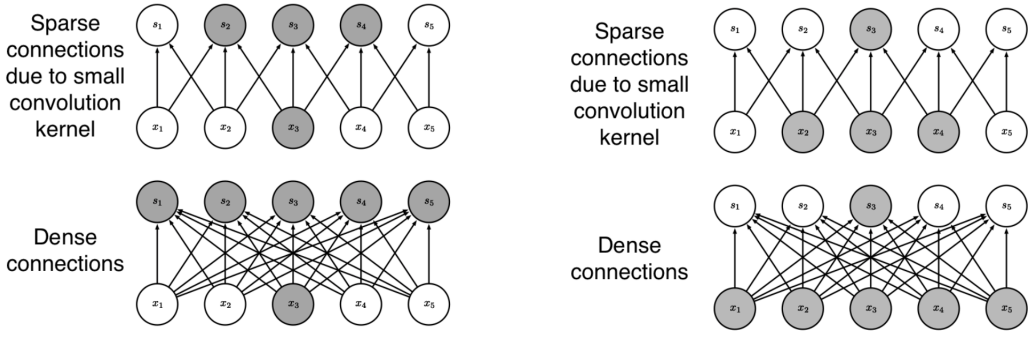


Figure 2.25: Sparse interactions with a 3 size filter [15]

These two properties lead to computational and memory efficiency. In particular, with the sparse interaction property, each output is computed by using just a small portion of the input: computational efficiency. While, with the parameter sharing property, the number of parameters to store in a layer doesn't depend by the input, but just by the size of the kernel: memory efficiency [15].

Input size: 320 by 280
Kernel size: 2 by 1
Output size: 319 by 280

	Convolution	Dense matrix	Sparse matrix
Stored floats	2	319*280*320*280 > 8e9	2*319*280 = 178,640
Float muls or adds	319*280*3 = 267,960	> 16e9	Same as convolution (267,960)

Figure 2.26: Efficiency of convolution [15]

In general, the use of convolution leads to an output with a different shape with respect to the input. Transposed convolution is a transformation going in the opposite direction of a normal convolution, i.e., from something that has the shape of the output of some convolution to something that has the shape of its input, while maintaining a connectivity pattern that is compatible with said convolution [11].

2.9.2 Convolutional and Locally Connected Layers

A convolutional layer consists in the application of the convolution operation to its input by a learnable filter, whose dimension is an hyperparameter. The resulting feature map is finally the input of an activation function, Figure 2.27. If the parameter sharing property of the convolution is removed, then such operation is called unshared convolution. Locally connected layer is therefore like a convolutional layer which performs unshared convolution [21].

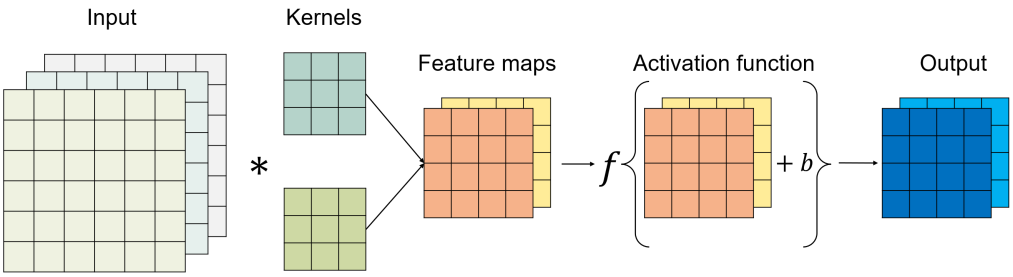


Figure 2.27: Convolutional layer

2.10 Autoencoder

The general idea underlying autoencoder (AE) is to learn a compressed and/or sparse representation h of the input x by learning the identity function $r = g(f(x))$, $r \approx x$ with some constraints [1][15], Figure 2.28.

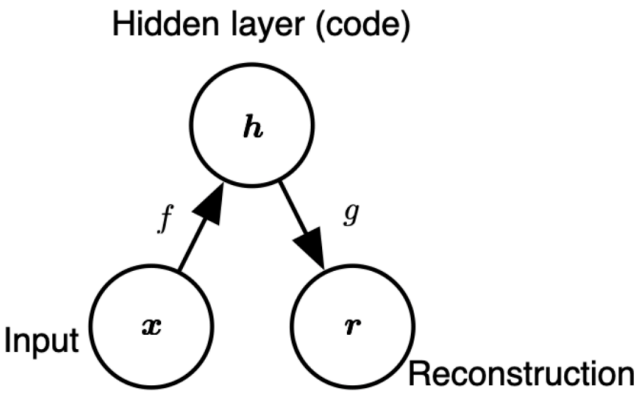


Figure 2.28: AE architecture [15]

This can be achieved by imposing a loss function of the type:

$$\mathcal{L} = Reconstruction\ error \tag{2.23}$$

where the reconstruction error can be defined as any function which measures the dissimilarity between x and r.
In particular the mapping $f : x \mapsto h$ is called encoder and the mapping $g : h \mapsto r$ is called decoder and both these functions turn out to be NNs, Figure 2.29.

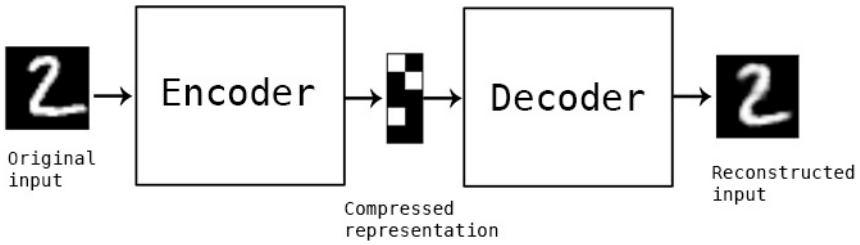


Figure 2.29: AE building blocks [15]

The constraints in the identity function can be imposed:

- On the architecture of the network: undercomplete AE
- Adding a regularizing term to the loss: overcomplete AE

In undercomplete AEs in particular, the hidden space h has a lower dimension than the input x. Therefore, the mapping $f : x \mapsto h$ has to discard some information and hence, has to learn what are the relevant informations.
Furthermore, it turns out that, if both the encoder and the decoder are linear neural networks, then $f(x) \approx Wx$ and $g(h) \approx Uh$ where W and U are respectively the compression and recovering matrices of PCA. Hence, the linear undercomplete AE mimics the PCA behavior. Therefore, the use of nonlinear activation functions in AEs allows to learn nonlinear dimensionality reductions of the input [3].

2.10.1 Autoencoders in Anomaly Detection

Deep learning (DL) is a subset of ML that achieves good performance and flexibility by learning to represent the data as a nested hierarchy of concepts within layers of the deep neural network. DL outperforms the traditional ML as the scale of data increases. In recent years, DL-based anomaly detection algorithms have become increasingly popular and have been applied for a diverse set of tasks; studies have shown that deep learning completely surpasses traditional methods [5].

The motivations and challenges in the use of deep anomaly detection techniques are:

- Performance of traditional algorithms in detecting outliers is sub-optimal on the image and sequence datasets since it fails to capture complex structures in the data
- Need for large-scale anomaly detection. As the volume of data increases then, it becomes nearly impossible for the traditional methods to scale to such large scale data to find outliers
- Deep anomaly detection techniques learn hierarchical discriminative features from data. This automatic feature learning capability eliminates the need of developing manual features by domain experts, therefore advocates to solve the problem end-to-end taking raw input data in domains such as text and speech recognition
- The boundary between normal and anomalous (erroneous) behavior is often not precisely defined in several data domains and is continually evolving. This lack of well-defined representative normal boundary poses challenges for both conventional and deep learning-based algorithms

Since the labels of normal instances are far more easy to obtain than anomalies, as a result, semi-supervised deep anomaly detection techniques are more widely adopted, these techniques leverage existing labels of single (normally positive class) to separate outliers. One common way of using deep AE in anomaly detection, is to train them in a semi-supervised way on data samples with no anomalies. With sufficient training samples of normal class, AE would produce low reconstruction errors for normal instances, over unusual events. Semi-supervised deep anomaly detection techniques methods rely on at least one of the following assumptions to score a data instance as an anomaly:

- Proximity and continuity. Points which are close to each other both in input space and learned feature space are more likely to share the same label
- Robust features are learned within hidden layers of deep neural network layers and retain the discriminative attributes for separating normal from outlier data points

AEs represent data within multiple hidden layers by reconstructing the input data, effectively learning an identity function. The AEs, when trained solely on normal data instances (which are the majority in anomaly detection tasks), fail to reconstruct the anomalous data samples, therefore, producing a large reconstruction error. Given an input x and let $\tilde{x}$ be the reconstruction obtained from an AE. Then, the reconstruction error can be used in order to discriminate

between normal and abnormal data [22].
In Figure 2.30, an AE, trained with just normal instances, learns an hidden space based on the training samples (prototypical normal patterns). Then, in the testing procedure, a new normal input is mapped close to a similar prototypical normal pattern. While an abnormal input is mapped far from any prototypical normal patterns, since different to any sample seen during the training procedure. Hence, since the AE maps back the closest prototypical normal pattern with respect to the input, the reconstruction error of a normal point will be lower than the one of an abnormal point. This because a normal point is closer than an abnormal point to a prototypical normal point in the hidden space.
Several variants of AE architectures are proposed, whose choice depends on the nature of data, producing promising results in anomaly detection.

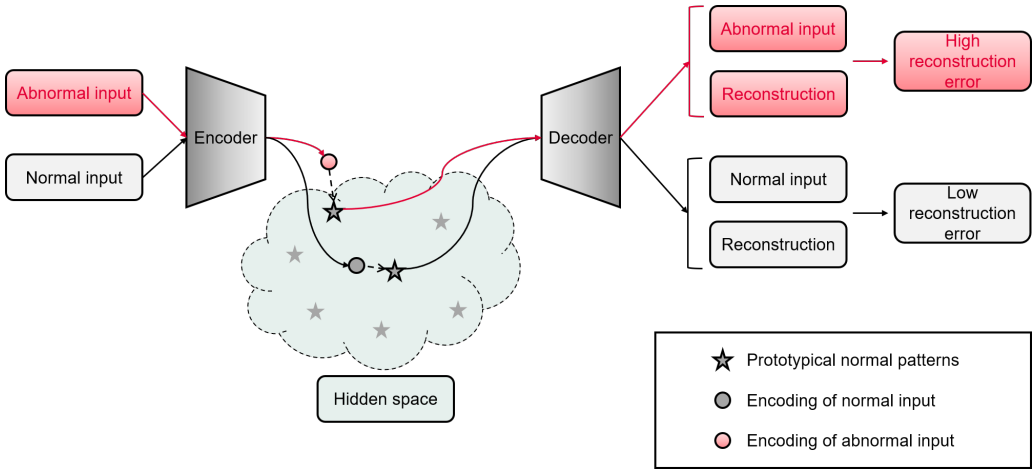


Figure 2.30: AE in anomaly detection

2.10.2 Dense Autoencoder

Dense autoencoder (DAE) is a particular architecture of undercomplete AE in which, both the encoder and the decoder are implemented with dense neural networks [24], Figure 2.31.

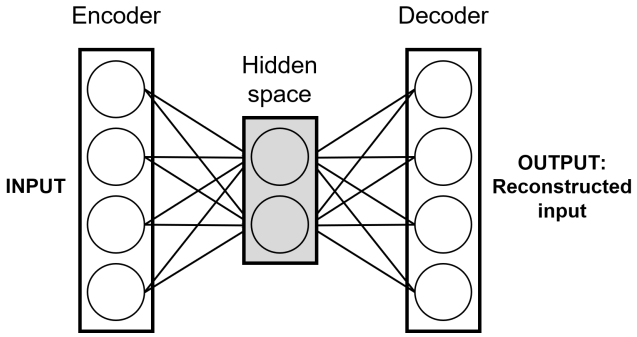


Figure 2.31: DAE architecture

2.10.3 Convolutional and Locally Connected Autoencoders

Convolutional autoencoder (CAE) is a particular architecture of undercomplete AE in which, the encoder is implemented with convolutional and dense layers. While the decoder is implemented with dense and transposed convolutional layers, in order to reconstruct the original input with the original shape [24], Figure 2.32.

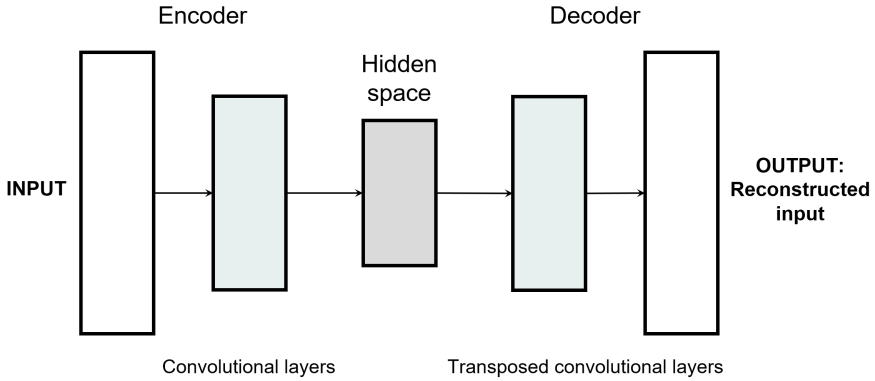


Figure 2.32: CAE architecture

Locally connected autoencoder (LCAE) is a variant of CAE in which the convolutional layers are implemented as unshared convolutions (locally conected layers).

2.10.4 Variational Autoencoder

One major division in ML is generative versus discriminative modeling.
Discriminative models learn the probability of a label y based on a data point x. In mathematical terms, this is denoted as $p_{model}(y|x)$. In order to categorize a data point into a class, it is needed to learn a mapping between the data and the classes. This mapping can be described as a probability distribution where each label will compete with the other ones for probability density over a specific data point.
Generative models, on the other hand, learn a probability distribution over the data points without external labels. Mathematically this is formulated as $p_{model}(x)$. This probability density effectively describes the behavior of the training data and enables to generate novel data by sampling from the distribution. Ideally, the model has to learn a probability density $p_{model}(x)$ which will be identical to the density of the data $p(x)$, hence, $p_{model}(x) \simeq p(x)$.
In this class of models, it can be distinguished the explicitly density estimation models, which explicitly define and solve for $p_{model}(x)$ and the implicit density estimation models, which learn to sample from $p_{model}(x)$ without explicitly define it.
Explicit density models can either compute exactly the density function or try to model it with latent variables. Latent variables are variables that are not directly observed and which describe or explain the data in a simpler way. In mathematical form, data points x that follow a probability distribution $p(x)$, are mapped into latent variables z that follow a distribution $p(z)$ [2][7][19]. In particular:

- The prior distribution $p(z)$ models the behavior of the latent variables
- The likelihood $p(x|z)$ defines how to map latent variables to the data points
- The joint distribution $p(x, z) = p(x|z)p(z)$ is the multiplication of the likelihood and the prior and describes the model
- The marginal distribution $p(x)$ is the distribution of the original data and it is the ultimate goal of the model
- The posterior distribution $p(z|x)$ describes the latent variables that can be produced by a specific data point

Then, in latent variables models, generation refers to the process of computing the data point x from the latent variable z and hence, it is defined by the likelihood $p(x|z)$. While, inference is the process of finding the latent variable z from

the data point x and is formulated by the posterior distribution $p(z|x)$.
Given a latent variables model, which has to learn a probability distribution $p_{model}(x) = p_\theta(x)$ parametrized over θ, its objective is to learn θ such that $p_\theta(x) \simeq p(x)$.
This could be achieved through maximum likelihood estimation, a well established technique of estimating the parameters of a probability distribution so that the distribution fits the observed data. In particular this can be accomplished maximizing the log-likelihood function:

$$\hat{\theta} = \arg\max_\theta \sum_i \log p_\theta(x_i) \tag{2.24}$$

In order to optimize (2.24), it is necessary to compute the marginal:

$$p_\theta(x) = \int p_\theta(x, z) dz \tag{2.25}$$

However, such integral turns out to be intractable. The intractability of $p_\theta(x)$ is related to the intractability of the posterior distribution $p_\theta(z|x)$ since the joint distribution $p_\theta(x, z)$ is tractable to compute:

$$p_\theta(z|x) = \frac{p_\theta(x, z)}{p_\theta(x)} \tag{2.26}$$

Approximate inference techniques allow to approximate the posterior $p_\theta(z|x)$ and hence, the marginal likelihood $p_\theta(x)$.
Variational autoencoder (VAE), Figure 2.33, is a deep latent variable model, which is a latent variable model $p_\theta(x, z)$ whose distributions are parameterized by neural networks.

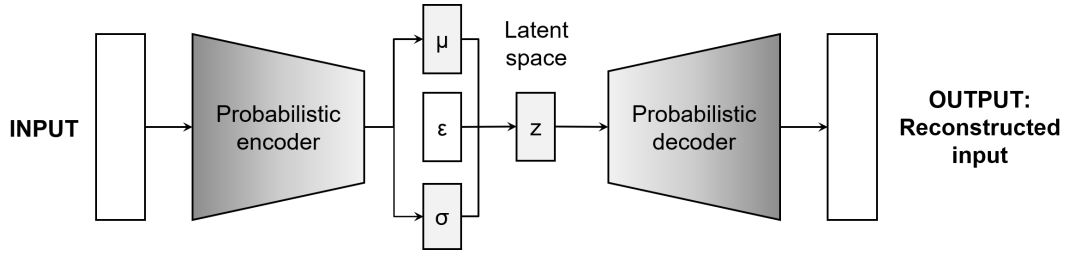


Figure 2.33: VAE architecture

Furthermore, VAE implements approximate inference techniques by introducing a parametric inference model $q_\phi(z|x)$. This model, also called encoder, is a

neural network which is optimized in order to have:

$$q_\phi(z|x) \simeq p_\theta(z|x) \tag{2.27}$$

While the decoder is a neural network which implements the likelihood $p_\theta(x|z)$. In particular, as can also be seen in Figure 2.33, a common choice is the factorized Gaussian encoder, in which the encoder computes mean $\mu_{z|x}$ and standard deviation $\sigma_{z|x}$ so that, given $\epsilon \sim \mathcal{N}(0, I)$:

$$z = \mu_{z|x} + \sigma_{z|x} \odot \epsilon \tag{2.28}$$

or equivalently:

$$z \sim \mathcal{N}(\mu_{z|x}, diag(\sigma^2_{z|x})) \tag{2.29}$$

VAE is a special case of AE because, even if its architectural affinity with AEs, there are significant differences both in the goal and in the mathematical formulation. Indeed, VAE is a generative model and differently from the architecture in Figure 2.28, in which the encoder learns two deterministic functions, generative models learn probability distributions, Figure 2.34.

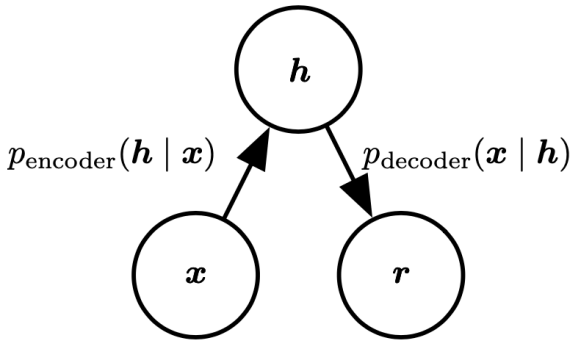


Figure 2.34: Stochastic AE [15]

Indeed, the loss function of VAE turns out to be:

$$\mathcal{L}_{\theta,\phi} = -\mathbb{E}_{z\sim q_\phi(z|x)}(log(p_\theta(x|z))) + D_{KL}(q_\phi(z|x) \,||\, p_\theta(z)) \tag{2.30}$$

whereas:

- $-\mathbb{E}_{z\sim q_\phi(z|x)}(log(p_\theta(x|z)))$ is the negative log-likelihood of the original input being reconstructed and hence, it is the reconstruction error

- $D_{KL}(q_\phi(z|x) \,||\, p_\theta(z))$ is the Kullback-Leibler (KL) divergence and forces $q_\phi(z|x)$ to be $q_\phi(z|x) \simeq p_\theta(z)$. Hence, it can be seen as a regularization term over the latent space, which forces the hidden representation to follow the prior distribution $p_\theta(z)$

Usually, the prior is assumed to be factorized unit Gaussian $p_\theta(z) = \mathcal{N}(0, I)$ and since the factorized Gaussian encoder $q_\phi(z|x) = \mathcal{N}(\mu_{z|x}, diag(\sigma^2_{z|x}))$, the loss function becomes:

$$\mathcal{L} = Reconstruction\ error + D_{KL}(\mathcal{N}(\mu_{z|x}, diag(\sigma^2_{z|x})) \,||\, \mathcal{N}(0, I)) \tag{2.31}$$

As in the case of AEs, several variants of VAE architectures are proposed, based on the implementations of the encoder and decoder parts, in order to adapt the architecture to different domains and improve its performance.
An important variant of VAE is the β-VAE [4]. Indeed, it is a modification of VAE framework that introduces an adjustable hyperparameter β to the original VAE loss:

$$\mathcal{L}_{\theta,\phi} = -\mathbb{E}_{z\sim q_\phi(z|x)}(log(p_\theta(x|z))) + \beta\ D_{KL}(q_\phi(z|x) \,||\, p_\theta(z)) \tag{2.32}$$

Well chosen values of β, result in more disentangled latent representations z. A disentangled representation can be defined as one where single latent units are sensitive to changes in single generative factors, while being relatively invariant to changes in other factors. For example, a model trained on a dataset of 3D objects might learn independent latent units sensitive to single independent data generative factors, such as object identity, position, scale, lighting or color. A disentangled representation is therefore factorized and often interpretable, whereby different independent latent units learn to encode different independent ground-truth generative factors of variation in the data.
The stronger pressure for the posterior $q_\phi(z|x)$ to match the factorized unit Gaussian prior $p_\theta(z)$, introduced by the β-VAE objective, puts extra constraints on the implicit capacity of the latent space z and extra pressures for it, to be factorized while still being sufficient to reconstruct the input data . Higher values of β necessary to encourage disentangling often lead to a trade-off between the fidelity of β-VAE reconstructions and the disentangled nature of its latent space. It turns out that, reconstructing under this constrained latent space, encourages embedding the data points on a set of representational axes, where nearby points on the axes are also close in data space.

3

Model Analysis and Experimental Results

In order to answer to the research questions [1.2], it is important to define first for which metrics the novel approach should be optimized in addition to a corresponding baseline. Since the goal of this anomaly detection problem is to detect suspicious samples, while keeping the number of false suspicious samples low, the main metric for which the different models will be compared is the F1 score (2.8) between TNR (2.5) and NPV (2.7). Although, for similar F1 score, the model with lower TNR is preferable, since the importance in detecting anomalies.

Model	TNR	NPV	MCC	F1-SCORE
FORMULA	0.961	0.889	0.921	0.923

Table 3.1: Formula performances

Before training any model, it is important to preprocess the data in order to make them suitable as input for an ML approach. Here, the crucial point is the downsampling step, because it allows to really simplify the problem by reducing drastically the size of each signal from 3646 to 156 and hence, the number of input features for an ML model. In particular, the applied downsampling technique is based on Fourier transformations and it has been empirically chosen and validated by experts. Such downsampling can be performed because it turns

out that the sampling rate, through which the signals are measured, is too high for the nature of such signals. Indeed, it can be noticed that the original signals present many flat regions where the variations are minimal, therefore, a small amount of points is needed in order to represent such signals. Furthermore, the patterns of interest are related to more global characteristics, hence, some possible small variations introduced in the signals with the downsampling method do not affect the patterns quality. Then, a simple smoothing window operation is performed in order to remove the noise introduced by downsampling. These first two operations are performed signal-wise. Finally, it is important to scale the signals to a fixed range of variations. Therefore, first the mean is subtracted and then, the signals are normalized to the [0, 1] interval through min-max normalization. Such two operations are still performed signal-wise:

$$\begin{aligned} &(1)\ x' = x - \mathbb{E}[x] \\ &(2)\ x'' = \frac{x' - min(X')}{max(X') - min(X')} \end{aligned} \tag{3.1}$$

whereas X' is the training set after the zero mean operation and $max(X')$ and $min(X')$ operations are computed along all features of the instances of X'.

The preprocessing procedure has been kept as simple and general as possible, with the idea to possibly generalize it to other similar products, as required by the fourth research question [1.2].

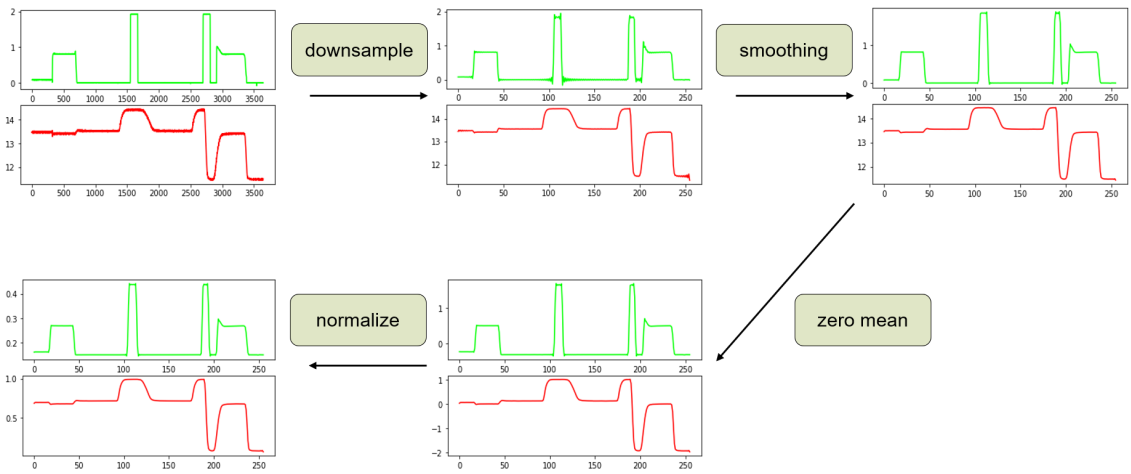


Figure 3.1: Preprocessing steps

3.1 Unsupervised Approach

Since one aim of the thesis is to keep the procedure as general as possible and to reduce the effort on building a model, in order to possibly be generalized to other similar product datasets, an unsupervised approach looks very interesting.

Model	TNR	NPV	MCC	F1-SCORE
IFOREST	0.996	0.123	0.26	0.219
DBSCAN	0.996	0.514	0.694	0.678

Table 3.2: iForest and DBSCAN performances

It turns out that an unsupervised approach doesn't provide good results, since an high number of false negative is provided. Indeed, from Table 3.2, the NPV is low with both the models.
This behavior is fully explainable by the data, because inside the "Not suspicious" samples, there are as many different variations with different distributions of patterns, so that the ones with a lower probability distributions are marked as "Suspicious" variations.

3.2 Semi-Supervised Approach

With labelled data, the idea underlying the semi-supervised approach is to train a supervised model with just the normal, in this case "Not suspicious", data in order to learn the normal behavior and then test the likelihood of a test instance to be generated by the utilized model. With this approach in fact, first of all it is possible to overcome the unbalancedness of the data and also, it is possible to be more robust to future anomalies, since the "Suspicious" samples are not used during the training procedure. Hence, the models are not biased to the "Suspicious" samples seen just so far. In this scenario, to test the feasibility of such approach, a simple model is first experimented: KNN.
In fact, by looking at both "Not suspicious" and "Suspicious" samples, after the preprocessing, in a two dimensional space through PCA, Figure 3.2, it is possible to see that the "Not suspicious" samples are somehow clustered, while the

"Suspicious" samples are slightly separated from the "Not suspicious" ones.

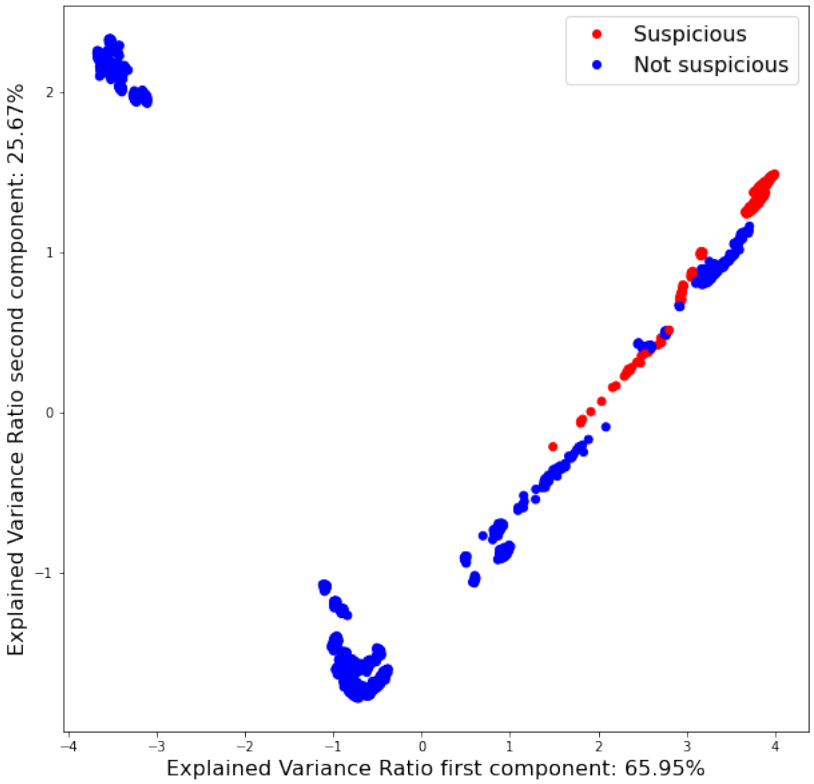


Figure 3.2: PCA of the preprocessed data

Therefore, by training a KNN with just "Not suspicious" data, the idea is that then, during the testing, the distance to the neighbors of a "Not suspicious" sample should be lower than the one of a "Suspicious" sample. In particular, given a sample, the average distance between its k=5 nearest neighbors is computed and then a threshold on such a distance is used in order to discriminate between "Not suspicious" and "Suspicious".

Model	TNR	NPV	MCC	F1-SCORE
KNN	0.961	0.944	0.943	0.952

Table 3.3: KNN performances

The results of KNN are satisfactory in terms of F1 score, however the TNR remains the same as the formula in Table 3.1. This means that KNN is not complex enough to capture all the features of the normal data and then is not able to discriminate the "Suspicious" data properly.

Indeed, from Figure 3.3, it is possible to see that 3.9% of "Suspicious" data are wrongly classified as "Not suspicious".

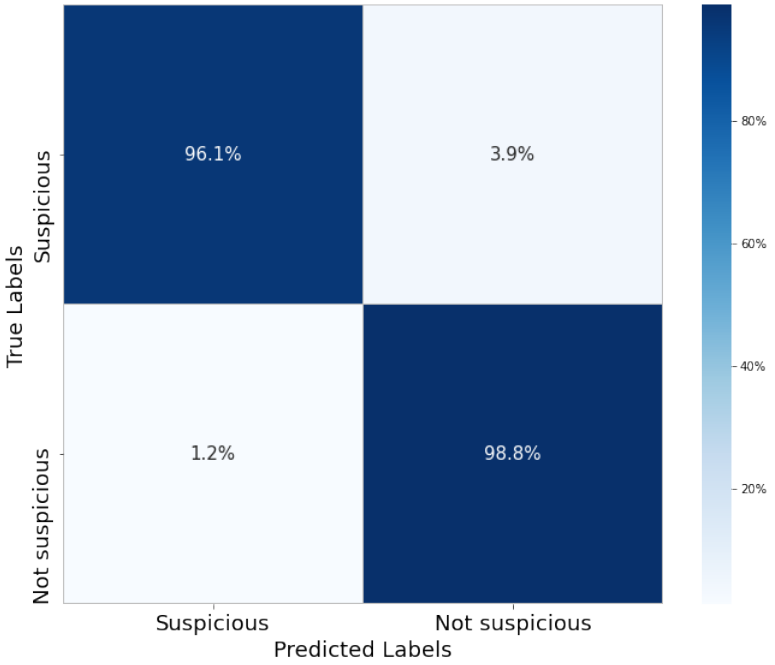


Figure 3.3: Confusion matrix KNN

In order to capture more features of the "Not suspicious" samples, more complex models are needed, but at the same time too complex models may lead to overfitting since there are not so many examples for each type of variation.
For these reasons, an undercomplete autoencoder looks appealing, since, thanks to its architecture constraints, it doesn't need to learn many parameters and this it may prevent overfitting.
Exploiting the fact that there are not inter-patterns between the two signals of a sample, it is possible to instantiate two DAEs with the same architecture, each trained with a different input signal, and then combine the two resulting reconstruction errors in order to discriminate among "Not suspicious" and "Suspicious" samples. With mean absolute error (MAE) between the original signal x and the reconstructed signal $\tilde{x}$ as reconstruction error:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|x_i - \tilde{x}_i| \tag{3.2}$$

In Figure 3.4, it is possible to see the distribution of the combined MAE losses of the two DAEs in the validation data. All the data are bounded within a certain MAE value, meaning that the threshold for discriminate among "Not suspicious" and "Suspicious" samples has to be looked at in such range of values.

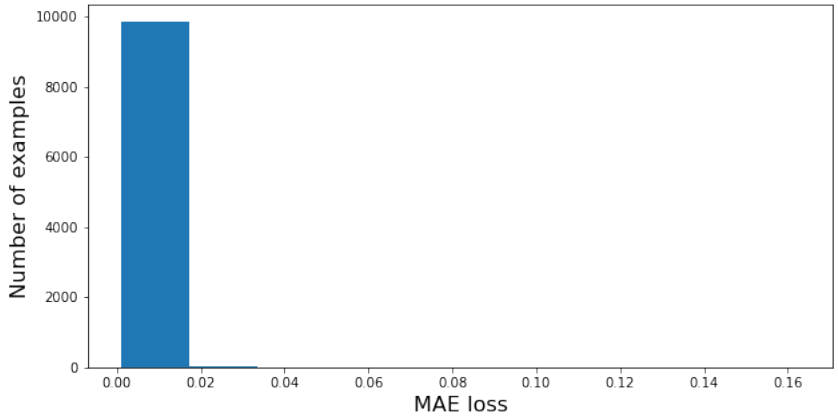


Figure 3.4: Histogram MAE reconstruction error of DAE

Model	TNR	NPV	MCC	F1-SCORE
DAE	0.995	0.969	0.978	0.982

Table 3.4: DAE performances

The use of two separated autoencoders leads to good results because this permits to keep the architecture simpler in terms of parameters to train, hence, avoiding overfitting.

Indeed, the percentage of wrongly classified "Suspicious" samples is drastically decreased, as can be seen in Figure 3.5, compared with the one of KNN in Figure 3.3.

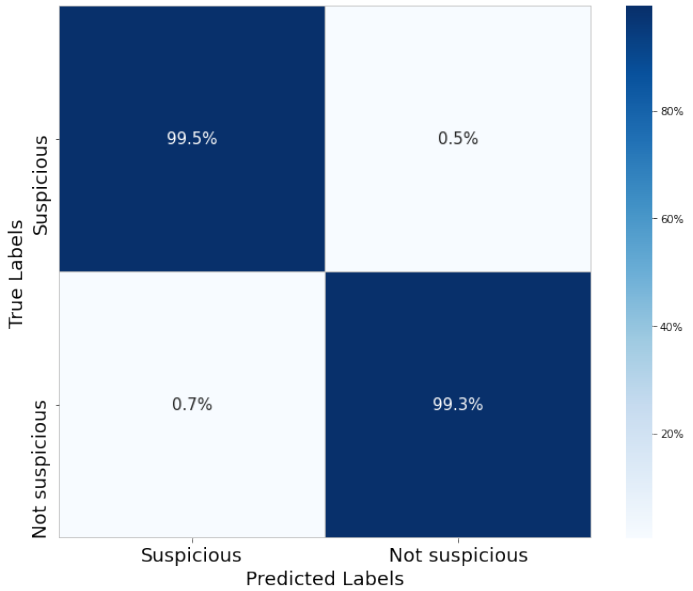
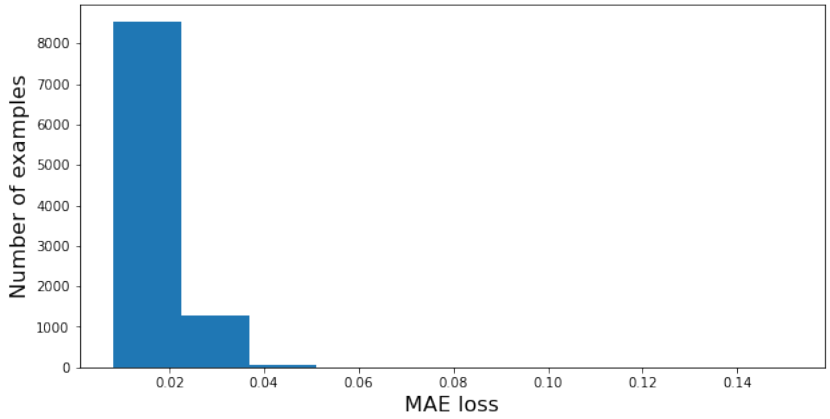


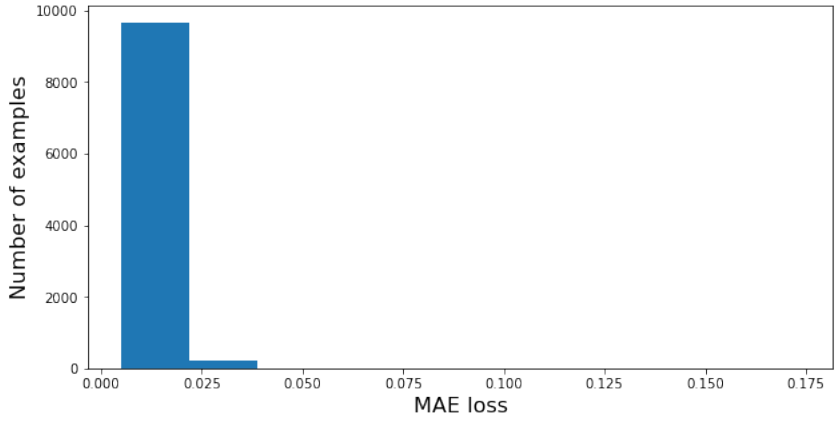
Figure 3.5: Confusion matrix DAE

The main drawback of a DAE is that, for example in the first hidden layer, each unit takes as input the whole signal, whereas the patterns looked at are local. Therefore convolutional and locally connected layers, hence, CAE and LCAE, can be used instead. In this way, it is possible to instantiate just a single model for the samples, by using each signal as a channel, and at the same time, to reduce the number of parameters to train, thanks to parameters sharing and local connectivity properties.

In Figure 3.6, it is possible to see the distributions of the MAE loss of CAE and LCAE in the validation data. All the data are bounded within a certain MAE value, meaning that the threshold for discriminate among "Not suspicious" and "Suspicious" samples has to be looked at in such range of values.



(a) Histogram MAE reconstruction error of CAE



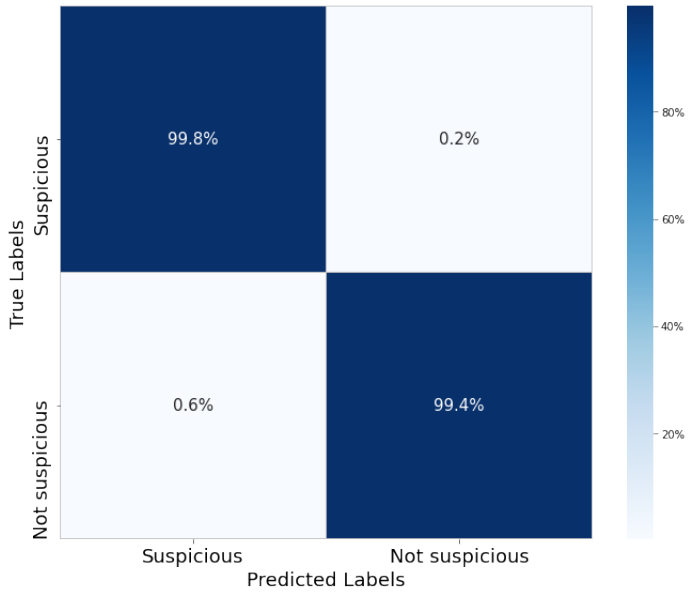
(b) Histogram MAE reconstruction error of LCAE

Figure 3.6: MAE of autoencoders based on convolutional and locally connected layers

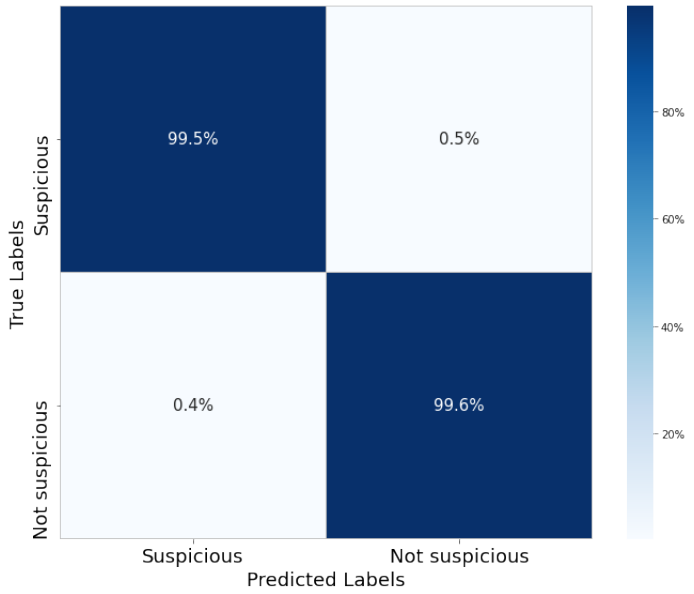
Model	TNR	NPV	MCC	F1-SCORE
CAE	0.998	0.973	0.982	0.985
LCAE	0.995	0.982	0.986	0.989

Table 3.5: CAE and LCAE performances

The bias encoded with the convolutional layers properties leads to an improvement in the performances, further decreasing the percentage of wrongly classified samples, Figure 3.7. Despite the better F1 score given by LCAE, the percentage of wrongly classified anomalies in CAE is lower. This makes CAE more preferable, since it is better in detecting "Suspicious" samples.



(a) Confusion matrix CAE



(b) Confusion matrix LCAE

Figure 3.7: Confusion matrices of CAE and LCAE

A limitation of an autoencoder is that there are not guarantees about its hidden space, because it just optimizes for the reconstruction error, in this case MAE. This means that the manifold is not taken as optimization target. In fact in Figure 3.8, the hidden space results very discrete and bad clustered.

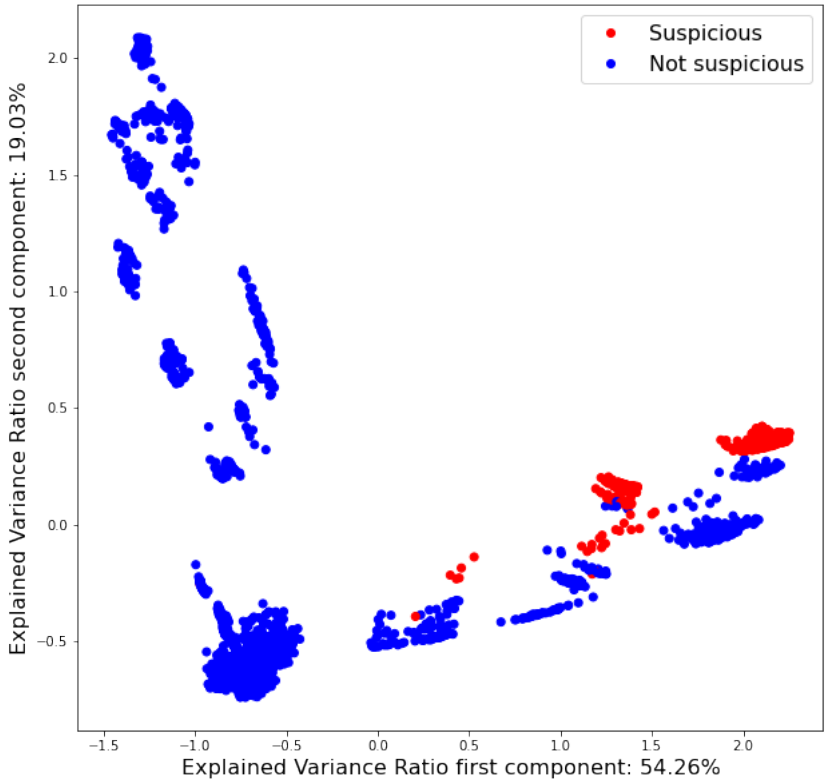


Figure 3.8: PCA of the hidden space of DAE

By using a variation of β-VAE, it is possible to add a hidden space regularization term to the loss function (3.3) whereas α and β are two hyperparameters for the trade off between the reconstruction error and the regularization of the hidden space.

$$\mathcal{L} = \alpha \cdot MAE + \beta \cdot D_{KL}(\mathcal{N}(\mu_{z|x}, diag(\sigma^2_{z|x})) \,||\, \mathcal{N}(0, I)) \tag{3.3}$$

Two types of VAE are tested, one based on dense layers: dense variational autoencoder (DVAE), which then discriminates for just KL divergence and the other based on convolutional layers: convolutional variational autoencoder (CVAE), which then discriminates for both MAE and KL divergence.

In Figure 3.9, it is possible to see the hidden space of DVAE. In this case the explained variance ratio is low in each dimension because the hidden space has a distribution very close to a multivariate normal distribution, hence all dimensions have more or less the same variance.

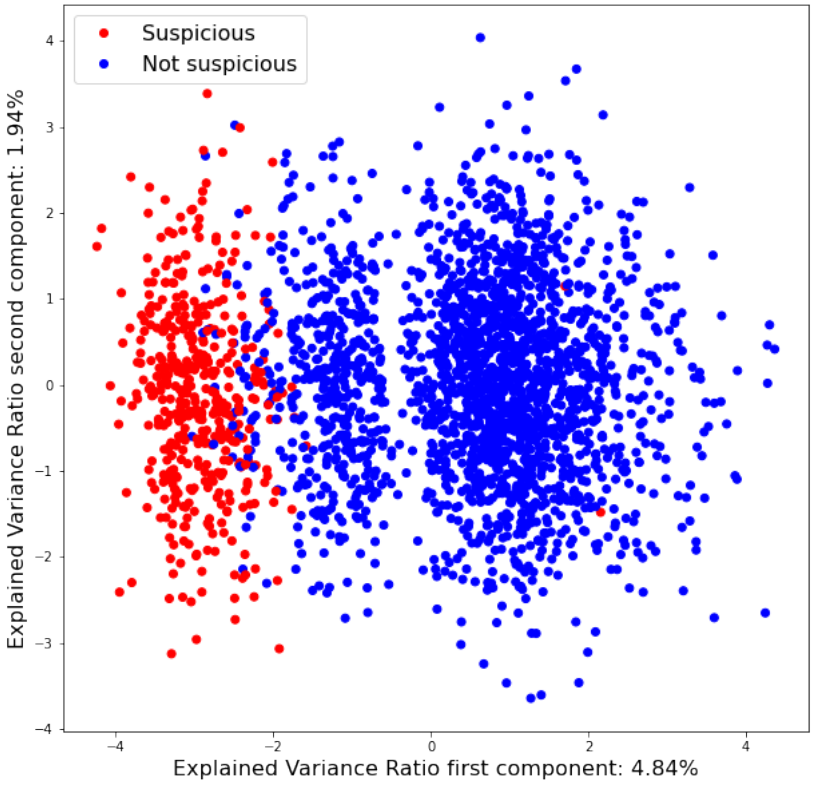
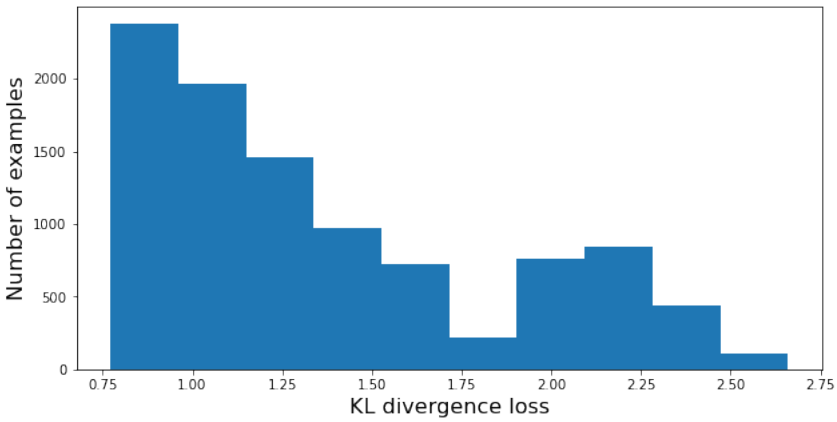
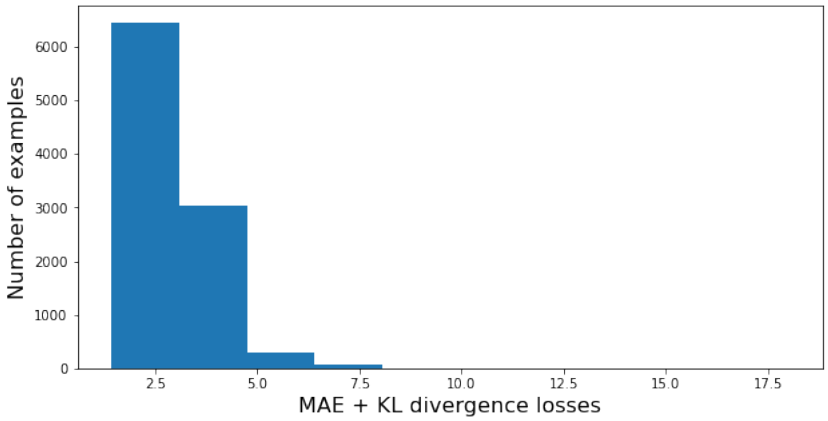


Figure 3.9: PCA of the hidden space of DVAE

In Figure 3.10, it is possible to see the distribution of the loss for DVAE and CVAE in the validation data. All the data are bounded within a certain loss value, meaning that the threshold for discriminate among "Not suspicious" and "Suspicious" samples has to be looked at in such range of values.



(a) Histogram KL divergence loss of DVAE



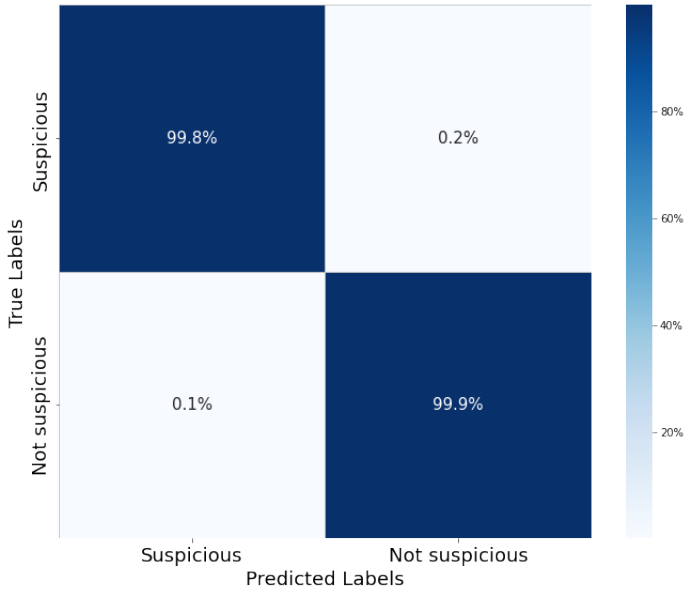
(b) Histogram MAE + KL divergence losses of CVAE

Figure 3.10: KL divergence loss and MAE + KL divergence losses for DVAE and CVAE

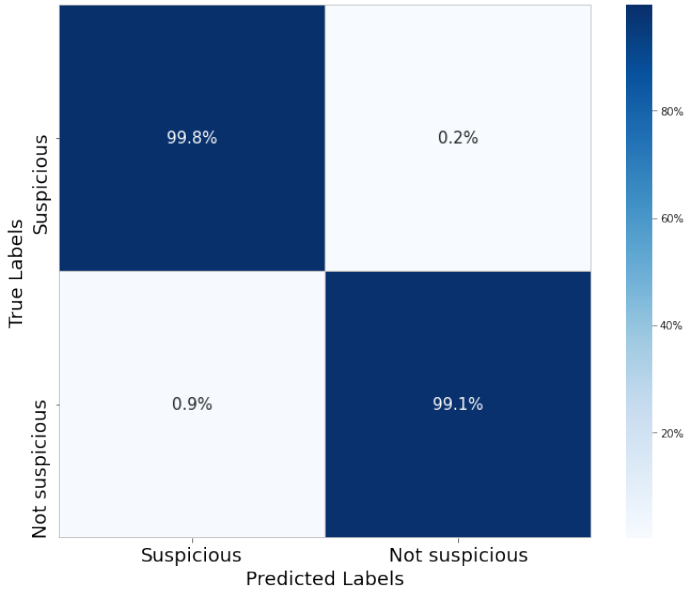
Model	TNR	NPV	MCC	F1-SCORE
DVAE	0.998	0.993	0.994	0.995
CVAE	0.998	0.967	0.978	0.982

Table 3.6: DVAE and CVAE performances

It is possible to see a net improvement of the performances in the DVAE, compared to the previous models. In fact, from Figure 3.9, the "Suspicious" points are mapped in the tail of a multivariate Gaussian distribution, hence the probability of such points is low in the imposed multivariate normal distribution and then, it is possible to discriminate by the KL divergence.
Indeed, from Figure 3.11, the percentage of wrongly classified samples in DVAE is the lowest achieved.



(a) Confusion matrix DVAE



(b) Confusion matrix CVAE

Figure 3.11: Confusion matrices of DVAE and CVAE

4

Conclusions and Future Works

In this thesis, several models have been experimented and it turns out that models based on semi-supervised learning have the best results. The success in the performances provided by those models surely confirm the first two research questions [1.2].
Since the extremely positive results given by the DVAE, further work has been done for the deployment of such a model. Furthermore, additional experiments have been done, with the same preprocessing procedure combined with DVAE, using samples coming from another similar product, and due to the positive outcomes, even the fourth question can be confirmed with some extent. The nature of the semi-supervised learning, combined with the distributional constraints in the hidden space imposed by DVAE, reasonably ensure robustness to future anomalies as the third question requires.
To conclude, the semi-supervised learning approach combined with the use of an autoencoder architecture is definitely the right approach for this particular anomaly detection problem.

References

[1] Devansh Arpit et al. *Why Regularized Auto-Encoders learn Sparse Representation?* 2015. DOI: 10.48550/ARXIV.1505.05561.

[2] Andrea Asperti. *Variance Loss in Variational Autoencoders*. 2020. DOI: 10.48550/ARXIV.2002.09860.

[3] Dor Bank, Noam Koenigstein, and Raja Giryes. *Autoencoders*. 2020. DOI: 10.48550/ARXIV.2003.05991.

[4] Christopher P. Burgess et al. *Understanding disentangling in -VAE*. 2018. DOI: 10.48550/ARXIV.1804.03599.

[5] Raghavendra Chalapathy and Sanjay Chawla. *Deep Learning for Anomaly Detection: A Survey*. 2019. DOI: 10.48550/ARXIV.1901.03407.

[6] Varun Chandola, Arindam Banerjee, and Vipin Kumar. "Anomaly Detection: A Survey". In: *ACM Comput. Surv.* 41.3 (2009). DOI: 10.1145/1541880.1541882.

[7] Taoli Cheng et al. *Variational Autoencoders for Anomalous Jet Tagging*. 2020. DOI: 10.48550/ARXIV.2007.01850.

[8] Jurman G. Chicco D. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 27 (2020), pp. 861–874. DOI: 10.1186/s12864-019-6413-7.

[9] Taurus T. Dang, Henry Y.T. Ngan, and Wei Liu. "Distance-based k-nearest neighbors outlier detection method in large-scale traffic data". In: *2015 IEEE International Conference on Digital Signal Processing (DSP)*. 2015, pp. 507–510. DOI: 10.1109/ICDSP.2015.7251924.

[10] Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. *Mathematics for Machine Learning*. Cambridge University Press, 2020. DOI: 10.1017/9781108679930.

[11] Vincent Dumoulin and Francesco Visin. *A guide to convolution arithmetic for deep learning*. 2016. DOI: 10.48550/ARXIV.1603.07285.

[12] Martin Ester et al. "A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. Portland, Oregon: AAAI Press, 1996, pp. 226–231.

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

[14] Trevor Hastie Gareth James Daniela Witten and Robert Tibshirani. *An Introduction to Statistical Learning*. Springer New York, 2013. DOI: 10.1007/978-1-4614-7138-7.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. DOI: 10.1007/s10710-017-9314-z.

[16] Kevin Gurney. *An Introduction to Neural Networks*. CRC Press, 1997. DOI: 10.1201/9781315273570.

[17] Torsten Hoefler et al. *Sparsity in Deep Learning: Pruning and growth for efficient inference and training in neural networks*. 2021. DOI: 10.48550/ARXIV.2102.00554.

[18] Ian T. Jolliffe and Jorge Cadima. "Principal component analysis: a review and recent developments". In: *Royal Society* (2016). DOI: 10.1098/rsta.2015.0202.

[19] Diederik P. Kingma and Max Welling. "An Introduction to Variational Autoencoders". In: *Foundations and Trendső in Machine Learning* 12 (2019), pp. 307–392. DOI: 10.1561/2200000056.

[20] Fei Tony Liu, Kai Ming Ting, and Zhi-Hua Zhou. "Isolation Forest". In: *2008 Eighth IEEE International Conference on Data Mining*. 2008, pp. 413–422. DOI: 10.1109/ICDM.2008.17.

[21] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[22] Manpreet Singh Minhas and John Zelek. "Semi-supervised Anomaly Detection using AutoEncoders". In: (2020). DOI: 10.48550/ARXIV.2001.03674.

[23] Brady Neal. *On the Bias-Variance Tradeoff: Textbooks Need an Update*. 2019. DOI: 10.48550/ARXIV.1912.08286.

[24] Alexandrine Ribeiro et al. *Deep Dense and Convolutional Autoencoders for Unsupervised Anomaly Detection in Machine Condition Sounds*. 2020. DOI: 10.48550/ARXIV.2006.10417.

[25] Russell and S. Norvig. "Artificial Intelligence: A Modern Approach". In: *Prentice Hall, Englewood Cliffs, NJ* (2010).

[26] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. DOI: 10.1017/CBO9781107298019.

[27] Gian Antonio Susto, Alessandro Beghi, and Seán McLoone. "Anomaly detection through on-line isolation Forest: An application to plasma etching". In: *2017 28th Annual SEMI Advanced Semiconductor Manufacturing Conference (ASMC)*. 2017, pp. 89–94. DOI: 10.1109/ASMC.2017.7969205.

[28] Kashvi Taunk et al. "A Brief Review of Nearest Neighbor Algorithm for Learning and Classification". In: *2019 International Conference on Intelligent Computing and Control Systems (ICCS)*. 2019, pp. 1255–1260. DOI: 10.1109/ICCS45141.2019.9065747.