POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Rare OTDR Events Retrieval With Deep Learning Approaches

Tesi di Laurea Magistrale in
Computer Science and Engineering - Ingegneria Informatica

Author: **Amirsalar Molaei**

Student ID: 940137
Advisor: Prof. Giacomo Boracchi
Co-advisors: Antonino Maria Rizzo, Ph.D. Student
Academic Year: 2022-23

Abstract

Similarity retrieval is one of the significant problems in Computer Science with a wide range of applications in industry. It aims to find similar instances to a query object from a large database. With the advances in deep learning, the capability and performance of retrieval systems have improved by a large margin. In this thesis, we aim to address the problem of one-dimensional signal retrieval over OTDR events by adopting deep learning methods. The OTDR events represent failures along optical fiber links and in our case, there are nine different types of events that were acquired by Cisco Photonics in Italy. We attempt to adopt deep learning vision techniques and architectures in a way to be suitable for one-dimensional signals. Our contribution is to design a retrieval system where we can query similar events from a large collection of signals. We propose three deep learning architectures based on Convolutional Autoencoder, Convolutional Neural Network for classification and Triplet Semi-hard Network for different training scenarios namely supervised and unsupervised. There are two principal challenges that we need to take into consideration for designing our methods: (i) our solutions have to generalize on unknown categories of events, that are not presented in the training set (ii) and methods are required to handle varying size signals in inference time. Our solutions demonstrate considerable performance and promising results in the retrieval of OTDR events.

Keywords: Retrieval, signal, deep learning, neural network, OTDR signals, autoencoder, CNN, triplet semi-hard

Abstract in lingua italiana

La ricerca di similarità mira a trovare i segnali che sono più simili a un oggetto in ingresso in un database di grandi dimensioni. È un problema significativo in Informatica con un'ampia gamma di applicazioni industriali. Sebbene sia comunemente usato per recuperare immagini, lo scopo di questo lavoro di tesi è affrontare il problema del recupero degli eventi OTDR, che rappresentano guasti sulle fibre ottiche. Cisco Photonics (Vimercate, Italia) è attualmente in grado di distinguere tra nove diversi tipi di eventi e ci ha fornito il set di dati OTDR utilizzato in questo studio. Il nostro contributo è progettare un sistema di recupero che ci permetta di cercare eventi simili in un grande set di dati. Proponiamo tre soluzioni basate su diverse architetture tipiche dell'Apprendimento Profondo: (i) Autoencoder, (ii) CNN per la classificazione e (iii) Triplet semi-hard networks, per affrontare gli scenari di addestramento sia supervisionato che non supervisionato. Ci sono due sfide principali che dobbiamo affrontare durante la progettazione delle nostre soluzioni: (i) i nostri metodi devono essere in grado di generalizzare su categorie sconosciute di eventi che non sono incluse nel set di allenamento e (ii) sono necessari metodi per gestire segnali di dimensione variabile che possono presentarsi in fase di inferenza. Le nostre soluzioni dimostrano prestazioni considerevoli e risultati promettenti nel recupero di eventi OTDR.

Parole chiave: Segnali, Apprendimento Profondo, Reti Neurali, OTDR Events, Autoencoder, Reti Neurali Convoluzionali, Triplet semi-hard

Contents

Abstract **i**

Abstract in lingua italiana **iii**

Contents **v**

Introduction **1**

1 Problem Formulation **3**

2 Background and Related Work **5**
2.1 Convolutional Neural Network(CNN) 5
2.2 Autoencoder Architecture 7
2.3 Triplet Siamese Networks 8

3 Proposed Method **11**
3.1 Learning Embedding Network 11
3.1.1 Autoencoder Architecture 11
3.1.2 CNN Trained for Classification 12
3.1.3 Triplet Semi-hard Network 13
3.2 Retrieval 15

4 Experiments **17**
4.1 OTDR Dataset 17
4.2 Experiment Settings 18
4.3 Evaluation Metric 19
4.4 Retrieval Performance in Baseline Setting 20
4.5 Retrieval Performance for Unknown Classes 23
4.6 Frameworks 26

5 Conclusions and Future Developments **27**

Bibliography **29**

A Appendix A **31**
A.1 Self-supervised learning 31

List of Figures **33**

List of Tables **35**

Acknowledgements **37**

Introduction

Similarity retrieval is a challenging and crucial problem in the field of Computer Science. In order to find similarities between data, we need to compare data points (e.g. vectors). Depending on the problem at hand, we may opt for different methods that can be used to measure similarity distance between data points such as cosine, dot product, etc. . For instance, in the case of dealing with images, the classical approaches consisted of hard-coded hand-crafted features from images to create representations in lower dimension then compare resulting feature vectors with each other using statistical distance measurement techniques.

Furthermore, in the past scientists used to employ hard-coded programs or dimensionality reduction algorithms to get features of data. As data is ever-changing, those systems needed constant updates to output desired results. One of the famous algorithms for dimensionality reduction is Principal Component Analysis (PCA), a statistical analysis technique with a broad range of use from visualization of high-dimensional data to information compression. For instance, in the past people used to perform PCA on images to reduce dimensionality and use those features for similarity search which is not a wise way to do. Because fundamentally PCA is a linear transformation, since images are unstructured data, we would like to have a learning algorithm that is able to capture complex patterns and at the same time be good at dimensionality reduction.

With the emergence of machine learning and in particular deep learning, the task of similarity search and retrieval has been given a new life. These techniques attempt to learn the defined problem from the data itself with or without supervision, in contrast to previously used statistical approaches. In this thesis, we investigate the problem of similarity retrieval of the Optical Time Domain Reflectometer (OTDR) events by utilizing deep learning methods. Our solutions are taken inspiration from deep learning techniques being practiced in computer vision applications. We essay to adopt vision solutions in a way to be applicable on one-dimensional signal data. Our experiments are carried out over OTDR events which is a collection of traces that are reflected lights captured in fiber links. The dataset consists of nine different events No event, Pass-through, Fiber-cut, Fiber-end, Face-Plate, Bulk-Attenuator, Fiber-bended, Fiber-knotted and Amplifier and

it is obtained by Cisco Photonics in Italy.
To design a similarity retrieval solution, we try to address the problem into two components: (i) feature extractor and (ii) similarity retrieval search. The first component is responsible to extract meaningful features of data. In other words, it has to preserve worthwhile features that contribute to the discrimination of particular data and discard irrelevant features(noises). Concerning similarity retrieval search, the algorithm takes as input a query embedding and calculates distance measure between a database of embeddings and query instance to retrieve alike examples. In the following paragraphs, we touch upon our solutions.
We propose three solutions with applications in different scenarios. In case of unsupervised scheme where data is unlabeled, we make use of the Autoencoder model to train and obtain embeddings. On the other hand, when data is labeled we propose two architectures to extract features, CNN and Triplet semi-hard networks. Note that, all three deep learning solutions are being applied to computer vision problems and are among the most successful methods. These techniques do not need engineers to define hard-coded rules for feature extraction part, instead, they rely on end-to-end algorithms to learn patterns from the data itself.
This thesis is organized into five chapters, starting with problem formulation and related works, then detailed elaboration of solutions followed by experiments and culminating in conclusion and future developments.

1 | Problem Formulation

In this chapter, we present a formal formulation for the problem of one-dimensional signal retrieval on events contained in the OTDR dataset. We are given a dataset D that includes signals of events belonging to nine different classes. We define X as a signal such that $X \in \mathbb{R}^n$ where n is length of the signal and note that signals can be from varying lengths. All signals of D are labeled with a corresponding class y where $y \in \{0, \ldots, 8\}$. We split D into two sets, known and unknown signals denoted by D_K and D_U respectively. $D_K \subseteq D$ where all signals belonging to the first 5 classes $y_K \in \{0, 1, 2, 3, 4\}$ and D_U contains signals from all classes such that $D_K \cap D_U = \emptyset$ but $D_K \cup D_U = D$ holds.

We aim to design a retrieval system that relies on a trained neural network model f_θ to learn embedding (feature vectors) of signals and later use these embeddings to retrieve similar instances. In depth, the system takes as an input query signal E_q where $E_q \in D$ and retrieves the K most similar signals from a database of embeddings, such that, the K retrieved signals are similar to E_q and all belong to the same class of E_q. We intend to maximize average overall precision and precision for each class over retrieved signals, we will discuss more this topic in evaluation metric Section 4.3.

There are some challenges to be noted. Firstly, the query signal E_q might be from varying lengths. In other words, as a neural network model trained on fixed-size of signals, in the retrieval phase we may have E_q which has a different size with respect to signals in the training set of the model. Secondly, the model has to be able to generalize on classes that it has not been trained on, meaning not all the query signals belong to D_K rather they would come from a different distribution. And most importantly, the training dataset is relatively small, and it suffers from imbalanced classes.

2 | Background and Related Work

As outlined before, we consider three solutions on how to approach similarity retrieval problem with deep learning. In this chapter, we present related work and literature regarding feature learning and retrieval of data. In particular, we provide a review of related research on three areas of convolutional neural networks, autoencoders and lastly triplet semi-hard scheme.

2.1. Convolutional Neural Network(CNN)

In deep learning literature, a convolutional neural network is a class of architectures that are mostly applied for computer vision problems and related tasks. In dense neural networks applied to images, we treat pixels as independent units, however, adjacent pixels are highly correlated with each other and carry relevant information, so here is where CNNs come to play. CNNs learn to extract patterns from images in a hieratical manner through convolution and pooling operations, outperforming dense neural networks. These models are applied in computer vision tasks such as classification, segmentation, object detection, etc. where input data are 2D or 3D such as images and videos. Although CNNs demonstrated promising results in the aforementioned areas, in recent years they have been gaining in popularity to be exerted on one-dimensional data specifically timeseries like signals. In the following paragraphs, we review famous de-facto CNN architectures ResNet and VGG, followed by several research papers about the application of 1D convolutions for timeseries. Our solutions throughout this thesis have drawn inspiration from architectural designs and methods proposed in these works.

Simonyan et al.(2014)[12] introduced VGG, a deep architecture where they attempted to increase the depth of the network. They used 3x3 convolution filters (smaller receptive field) and ReLU activation function. This work showed deeper models generalize and learn better features of data which consequently contributes to better performance. Figure 2.1 shows the underlying architecture of VGG19 with approximately 144 million parameters. As it can be seen from the architecture, when an input image goes through the network its spatial size is reduced by convolutional and pooling layers while the number of channels

increases. The CNN backbone is followed by three fully-connected layers that account for most of the parameters of the network.

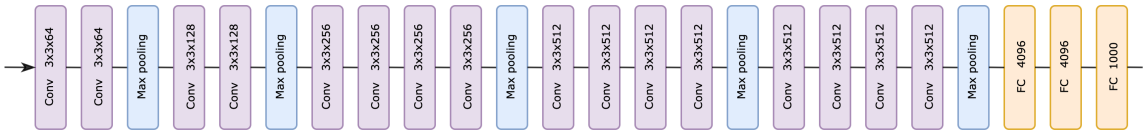


Figure 2.1: VGG19 architecture

Later in this field, a deep residual network was introduced by He et al.(2015)[6] with the aim of providing a reliable framework for training very deep neural networks which are easier to optimize and are capable of learning deep representations. This model mitigates the problem of vanishing/exploding gradients and degradation. The former is solved through extensive use of normalization both in initialization and throughout intermediate layers, and the latter happens when accuracy gets saturated, thus the authors resolved this issue by proposing residual blocks and shortcut connections. As a result of this research, it was made possible to train deep models up to 152 layers without loss of accuracy. The core piece of ResNet design is the use of residual blocks as represented in Figure 2.2, these blocks explicitly try to approximate residual function rather than original function $F(X)$ with the motivation that the training error of a deeper model can not be larger than its shallower counterpart, so the degradation problem can be mitigated.

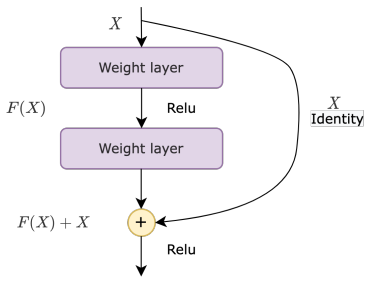


Figure 2.2: Residual block

As it is stated already, 1D convolutions have shown impressive results when applying on time series like signals. Since, we would like to adopt these techniques for OTDR events we review two related papers with regard to application of 1D convolutions. Chen et al.(2020)[2] proposed 1D convolutional autoencoder based architecture for feature learning with the purpose of fault diagnosis in multivariate processes. Because autoencoders enable the learning of discriminant features in an unsupervised way, they trained end-to-end autoencoder on unsupervised historical data. Once trained, a classifier is added at

the head of the network in order to fine-tune the network on fewer labeled examples. Furthermore, in the field of material science, Schuetzke et al. (2020)[11] used a convolutional Siamese network for the classification of 1D signals generated by X-ray diffraction over different materials. They showed that the network could even generalize for materials that were not presented in the dataset, achieving high accuracy. The proposed architecture was taken by VGG16 paper, however, it was designed in a way to suit one dimensional signals.

2.2. Autoencoder Architecture

Autoencoders had been introduced during the 1980s, however, the researchers could not simply train models to perform properly. The first successful attempt to train deep autoencoders to get better representation was carried out by Hinton et al. (2006)[7] in which their model outperformed the PCA method (back then PCA was popular algorithm with considerable performance). In detail, Autoencoder is a type of neural network architecture that is used to reconstruct input data on the output. It consists of two stacks of layers called encoder and decoder. The encoder shrinks down the input by downsampling and the decoder expands the learned representation to reconstruct the input. In other words, the encoder squeezes the input by preserving relevant information that is important for the reconstruction of output and discards irrelevant features(noises).

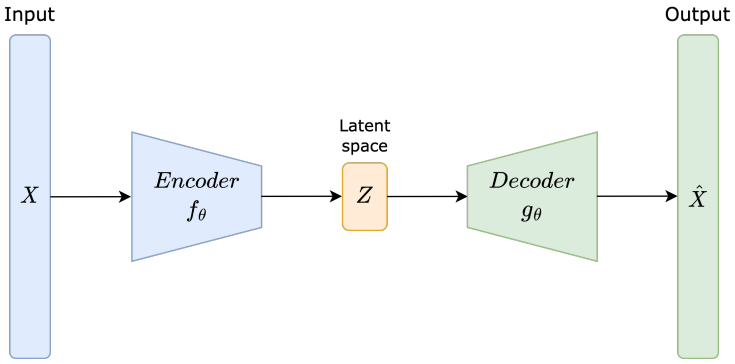


Figure 2.3: Autoencoder architecture consists of encoder and decoder stacks denoted by f_θ and g_θ respectively. The latent vector Z is a representation of input X in a lower dimension.

Nowadays, with the advances in research and availability of computing power, autoencoders are trained in an end-to-end fashion. They are able to capture complex features and patterns of input and try to recover them in the output. The network aims to minimize the reconstruction error $L(X, \hat{X})$, which is the discrepancy between the input and

its reconstruction on output.

Autoencoders are useful for learning representation, they can be used for various types of data such as images, audio, text and so on. For instance, We pass an image through a network, and it bottlenecks the image into a latent vector by reducing its dimensionality and then expanding it to the original dimensionality. In this scenario, the input image acts as its label meaning that for a given input image we expect the network to reproduce it in the output. We can interpret that, a successive layer of encoder compresses the input into a latent vector which is a thorough representation of input data in a lower dimension.

Indah et al. (2019)[13] proposed a solution for content-based image retrieval by using convolutional autoencoders. They trained a model on images of the Corel dataset. Once trained, the decoder part is discarded, keeping only the encoder as a feature extractor. To query similar images, the network takes an image extracts its features by running through the encoder stack, and then compares the embedding to all the embeddings in a database to retrieve similar cases. To measure the similarity distance between embeddings they used Euclidean distance 2.1 where p and q are two points in Euclidean n-space.

$$d(p,q) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2} \tag{2.1}$$

Furthermore, Wen et al. (2018)[14] utilized convolutional autoencoders as an unsupervised feature extractor to classify EEG signals. They achieved an accuracy of more than 92% which was more significant than previous methods such as principal component analysis and sparse random projection in terms of feature effectiveness and dimensionality reduction. For classification purposes, extracted features were then fed to a classifier, in this case, they used several classifiers including k-NN, SVM, etc.

2.3. Triplet Siamese Networks

Schroff et al. (2015)[10] from Google proposed a novel end-to-end learning system for face verification called FaceNet, which uses a triplet loss function to adjust the weights of the learning algorithm. Its backbone architecture for feature learning is Inception architecture. During the training phase, the network requires three images: Positive, Negative and Anchor. According to the loss function 2.2, pair of positive and anchor images are instances of the same class and negative from a different class. As shown in Figure 2.4, the goal of loss function is to make the square distance between the positive and anchor smaller in Euclidean space, whilst the distance of the negative sample with

anchor has to be larger.

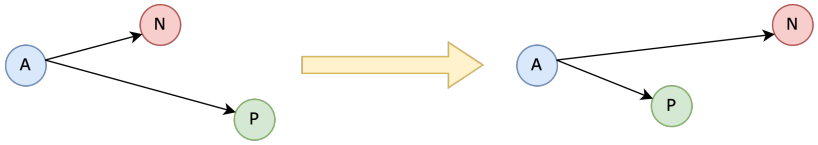


Figure 2.4: Through the learning process, the **triplet loss function** minimizes the distance between anchor and positive samples which belong to the same class and maximizes the distance between anchor and negative samples.

We want to be certain that positive and anchor images are closer together in latent space and distant from a negative image. The loss is formulated below for a given triplet of anchor, positive, negative denoted by x^a, x^p and x^n respectively, and α is a margin that we impose between positive and negative pairs.

$$\sum_{i}^{N} \Big[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \Big] \tag{2.2}$$

Figure 2.5 illustrates the structure of the model proposed in FaceNet[10] paper. The model has a CNN backbone for feature learning followed by *L2* normalization which culminates in embeddings. Finally, triplet loss function is used for training the network.

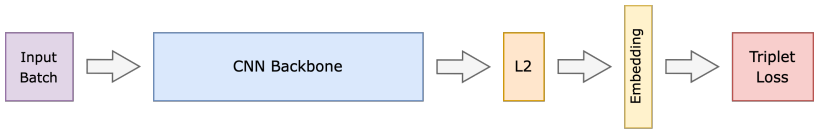


Figure 2.5: Model structure proposed in FaceNet

3 | Proposed Method

This chapter elaborates in detail the proposed solutions for the problem of one-dimensional signal retrieval specifically their corresponding architectural and configurational choices. We consider two components: embedding network and similarity retrieval algorithm. The first component acts as a feature extractor resulting in the embedding of the input. For this purpose we propose an autoencoder network for an unsupervised scheme where data is not labeled, followed by CNN and triplet semi-hard networks for supervised cases. Moreover, the similarity retrieval algorithm is accountable for searching and retrieving similar samples of a given query signal E_q from the database of embeddings.

Recall that, we need to take into consideration two important key challenges. First, our solutions should be able to handle signals with varying sizes during the retrieval phase, meaning the query signal E_q may have a different length with respect to the training set of the model. This matter is required to be addressed in network architecture itself. On top of that, models are envisaged to be able to generalize on never-before-seen classes. In other words, we might use models to retrieve signals from different data distributions.

As a rule of thumb, henceforth when we refer to a convolutional neural network we intend models built with 1D convolutions that are well-suited for timeseries like signals.

3.1. Learning Embedding Network

3.1.1. Autoencoder Architecture

As the first solution for learning representation of signals, we introduce a convolutional autoencoder model for unsupervised scenarios where data is not labeled. Our model consists of two stacks of layers; encoder and decoder. The architecture is inspired by ResNet paper as discussed in Section 2.1. The reason behind this choice is that ResNet architecture demonstrated to be effective in preserving relevant features through deep networks due to residual blocks and skip connections. Also, based on our experiments autoencoder built with residual blocks outperformed its counterpart VGG network.

We denote encoder and decoder with f_θ and g_θ respectively. Hence, $f(X)$ is the latent representation of input X and $g(f(X))$ is the output of the network which is the reconstruction of input. We train f_θ and g_θ to minimize mean square error (MSE) as shown in 3.1 which is a reconstruction loss of X.

$$L_{\text{MSE}}(X, g(f(X))) = \|X - g(f(X))\|_2^2 \tag{3.1}$$

Our proposed architecture has overall 19 convolutional layers of which 9 layers are for the encoder and 10 for the decoder and all the layers used in the decoder are transposed convolutions. We train the model in an end-to-end fashion intending to minimize reconstruction loss. Once training is completed, we discard the decoder subnetwork and keep only the encoder as a feature extractor. Note that one of the challenges we need to address is that, the network should be able to handle varying sizes of signals during inference. For this reason, as is evident from Figure 3.1 we fused a global average pooling layer that outputs a latent vector.

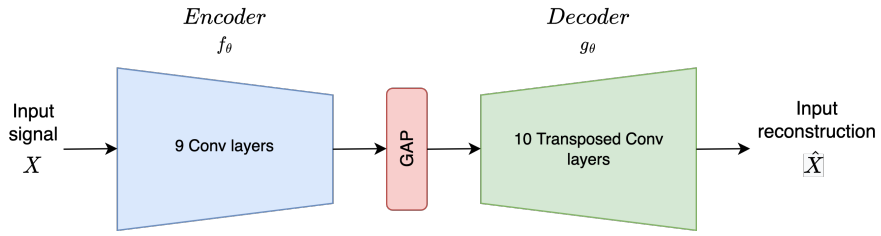


Figure 3.1: Our proposed autoencoder network

GAP is originally designed to replace fully connected layers to overcome the limitation of input size fed to the network. As it is illustrated in Figure 3.2, GAP calculates the average of each feature map over input volume, thus, no matter what is the size of the input signal, the encoder subnetwork always outputs embeddings with the same size.

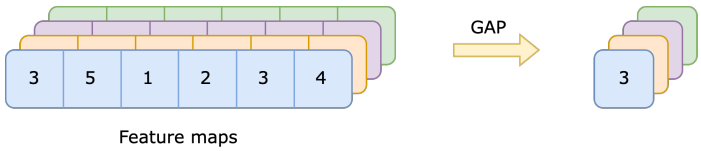


Figure 3.2: Global Average Pooling operation

3.1.2. CNN Trained for Classification

The second model that we use for representation learning of signal data is a CNN network. Our model is inspired by VGG19 2.1 architecture and it is trained with supervision,

meaning that training data is labeled. In deep learning literature, it is widely practiced to train a model for classification and after training get rid of the classification head and use the backbone network for downstream tasks. Likewise, we want to train our model for the classification of one-dimensional signals and later remove the network's classification head and use the backbone as a feature extractor for the retrieval system.

Figure 3.3 depicts our proposed CNN network, the architecture consists of consecutive layers of convolution followed by max pooling layers. And as we go deeper through the network number of convolution filters increases up to 512 filters. All convolutional layers are ReLU activated and initialized with He normal initializer. We tailored VGG19 network in a way that suits our needs, so we replaced fully connected layers which impose a limitation on input size of the network with convolutional layers that have filter size of one. And we added a global average pooling as the penultimate layer followed by a dense layer for classification.

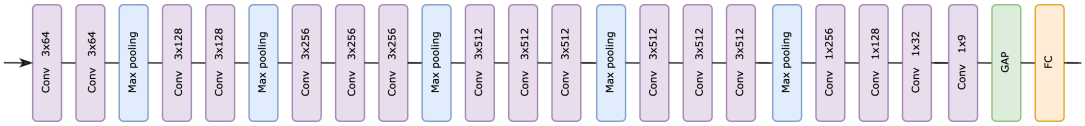


Figure 3.3: Our proposed CNN network

Once the model is trained, we freeze layers from the beginning up to GAP and use them to extract embeddings of signals. As mentioned earlier GAP enables us to use the varying size of signals in inference time so we will not have limitations in this matter.

3.1.3. Triplet Semi-hard Network

The third approach we are going to introduce for representation learning of signals is based on the triplet loss function and model structure proposed in Section 2.3. All in all, the goal of triplet loss function is to ensure examples of the same class are closer to each other in the latent space and far away from examples belonging to different classes. There are three requirements for the triplet loss function:

1. an **anchor**;
2. a **positive** sample same class as the anchor;
3. and a **negative** sample from a different class.

According to (2.2), we try to minimize the distance between anchor and positive while pushing pairs of anchor and negative to be distant from each other. The parameter α

is a margin defined between positive and negative pairs. The foremost matter is how to select triplets because if we want to generate all triplets, there will be many triplets which would fulfill the constraint in (3.2):

$$\|f(x_i^a) - f(x_i^p)\|_2^2 + \alpha < \|f(x_i^a) - f(x_i^n)\|_2^2 \tag{3.2}$$

Thus, triplet selection or so-called triplet mining is vital for achieving fast convergence while respecting the above constraint. Considering loss function we can define three categories for triplets:

- **Easy triplets:** triplets result in loss of 0, due to $distance(x^a, x^p) + \alpha < distance(x^a, x^n)$.
- **Hard triplets:** triplets in which x^n is closer to x^a than x^p.
- **Semi-hard triplets:** triplets where the negative is not closer to the anchor than the positive, however, at the same time their outcome is a positive loss.

Figure 3.4 depicts three categories mentioned before:

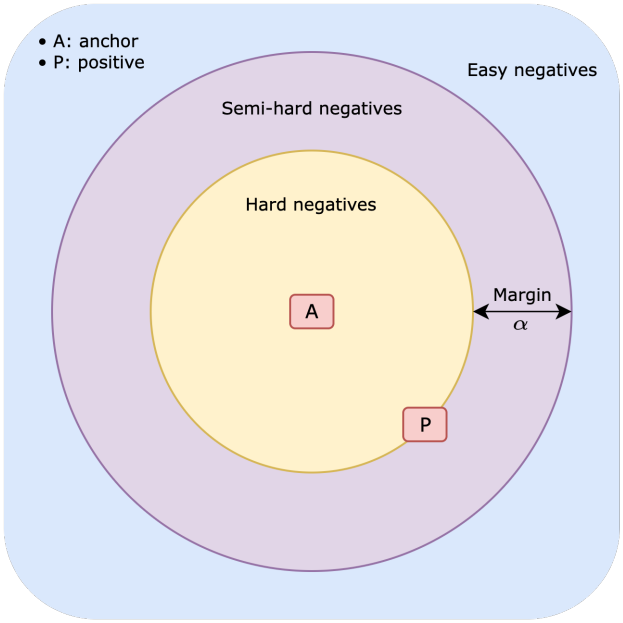


Figure 3.4: Triplet selection categories

Furthermore, there are two ways to mine samples: offline and online. In offline mining, triplets are produced at the beginning of each epoch. In particular, we compute embeddings of the training set and then choose proper samples for creating triplets. This method is not computationally efficient since it needs a full pass through the entire training set to produce triplets. The most sophisticated approach is online mining with the

idea of creating triplets straight away on every batch of inputs. In detail, we compute embeddings of a batch of inputs and pick out hard or semi-hard samples.

With respect to our solution, we follow the model structure as shown in Figure 2.5, where as a backbone we use the same architecture introduced in Section 3.1.2, but we replace the global average pooling with a global max pooling layer to get a better convergence. We make use of semi-hard mining as it is proven to yield the best results, in addition to the use of an online mining strategy that boosts the training procedure. Once the model is trained on events of the OTDR dataset, it is ready to use as an embedding function.

3.2. Retrieval

The second component in a retrieval system is similarity search in which we try to find similar data points to E_q from a database of embeddings. For this reason, we take advantage of the Unsupervised Nearest Neighbor (NN) algorithm which can efficiently retrieve similar instances to our query signal.

In order to carry out the retrieval phase, we obtain embeddings (feature vectors) of all signals $f_\theta(D)$ where f_θ is a learned feature extractor and it can be one of the previously proposed deep learning solutions. Once we have the embeddings, we store them so later it would be more convenient to compute nearest neighbors. We want to retrieve K most similar signals, for a given query E_q such that K retrieved signals are the closest to $f_\theta(E_q)$ in the latent space. To wrap up, the one-dimensional signal retrieval procedure can be represented as follows in Algorithm 3.1.

Algorithm 3.1 One-dimensional Signal Retrieval

Require: Compute embedding of signals $f_\theta(D)$, query signal E_q, number of retrieved samples K where $K \in \mathbb{N}$

Ensure: $Results \subseteq D$ and K retrieved signals belong to the same class as E_q

1: Compute embedding of a query signal $f_\theta(E_q)$
2: Opt for K nearest neighbors of $f_\theta(E_q)$ from $f_\theta(D)$
3: Store K similar signals as $Results$

Figure 3.5 illustrates the high-level architecture of a similarity retrieval system. In this case, $K = 3$ refers to the three most similar events with respect to E_q that are retrieved from dataset D. If we notice only two out of three events belong to the same class of E_q. To elaborate this procedure, first, we take dataset D which consists of known (D_K) and unknown (D_U) events, we run f_θ over D to obtain embeddings of signals. Then, we select

a query event E_q and feed it to f_θ to get the embedding. Once we have the embedding, we search for the three closest neighbors to E_q from the database of embeddings obtain from D.

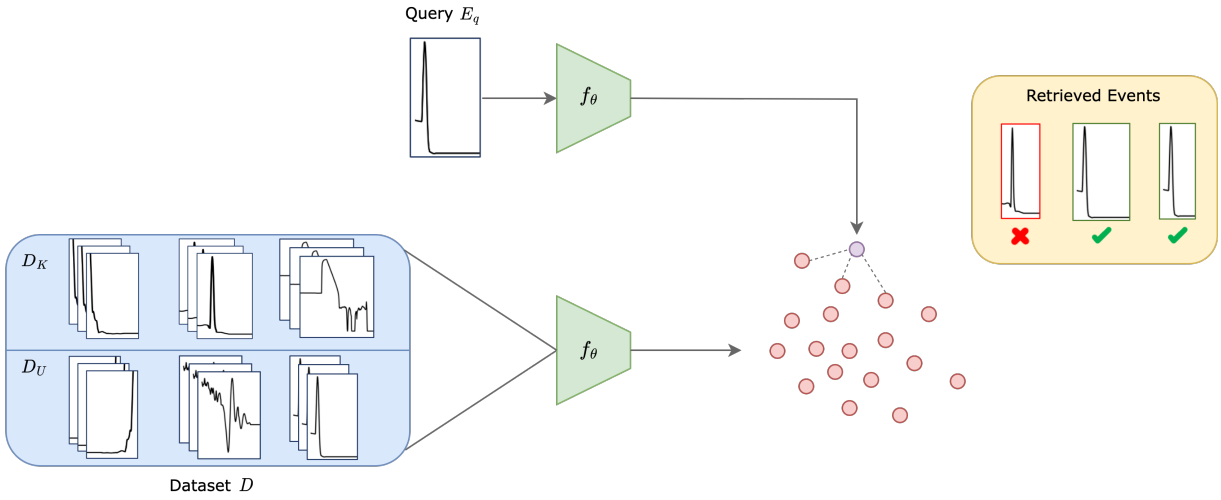


Figure 3.5: High-level architecture of a retrieval system

4 | Experiments

In this section we provide comprehensive information about our experiments. We start with the OTDR dataset and then we introduce two settings in which we evaluated our models' performance. Next, we present evaluation metric for retrieval being used in experiments followed by a report on final results and technical frameworks.

4.1. OTDR Dataset

We conduct our experiments and verify our solutions on a dataset of OTDR events which is collected by the Cisco team through experimenting with physical devices and fiber links.

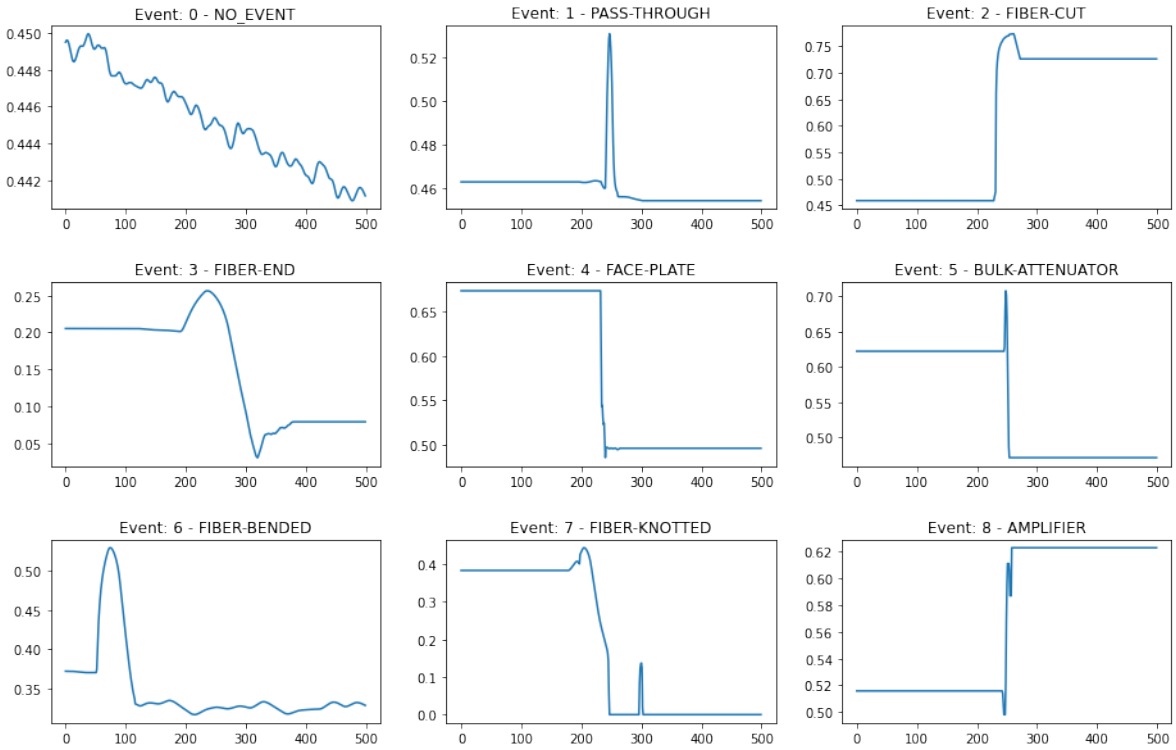


Figure 4.1: Nine types of events in OTDR dataset

OTDR is an optoelectronic apparatus that is used to measure reflection of light along op-

tical fiber links. This instrument generates optical pulses into the fiber link and measures the light that is reflected or scattered back to the source. Through this process the device produces OTDR traces that are characterized by multiple different events. At the time of writing, our dataset includes nearly 6300 events and it consists of 9 types of events as shown in Figure 4.1. The distribution of events is demonstrated in Figure 4.2, as can be seen, the dataset suffers from imbalance classes, especially on the following events: FIBER-CUT, BULK-ATTENUATOR, FIBER-BENDED and FIBER-KNOTTED.

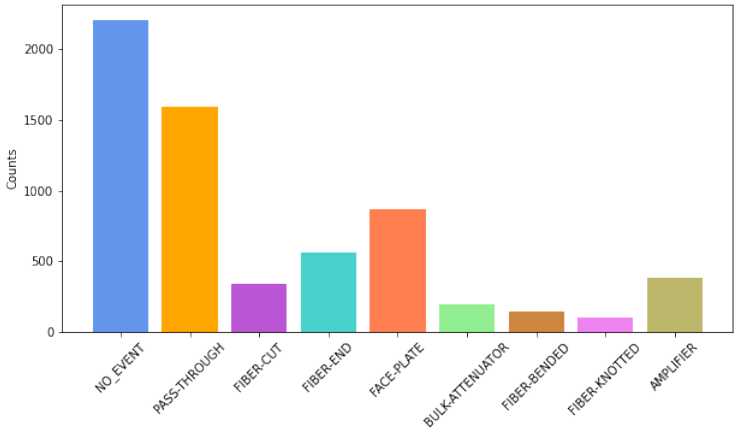


Figure 4.2: Distribution of events in OTDR dataset

As deep learning methods are data-hungry, we require more data in order to obtain better results and somehow alleviate the problem of imbalanced classes. Additionally, we cannot take advantage of generative models to increase the size of the dataset, since this might introduce artifacts that are not of our interest and must be avoided. In this regard, the process of data acquisition is being undertaken by the Cisco team, since producing OTDR events needs special physical devices and requires subtleties. To prepare the dataset for training and testing, we use the same methods for preprocessing as proposed in the recent paper by Rizzo et al. (2022)[9].

4.2. Experiment Settings

We conduct two experiments for each solution to demonstrate their effectiveness in different settings. Models are tested in the following settings:

- **Baseline:** This is the simplest setting in which we split dataset D into two separate sets Training T and Retrieval R. Training a network is done by T, and retrieval performance evaluation is performed over R. During retrieval, we calculate $f(E_q)$ where $E_q \in R$ and then we search for K closest samples from $f(R)$.

- **Unknown Classes:** As previously stated, it is crucial for our system to be able to generalize on data that it has not been trained on and still be effective in retrieving similar instances. To measure the performance of the model in this setting, we split D into D_K and D_U such that $y_K \in \{0, 1, 2, 3, 4\}$ and $y_U \in \{0, ..., 8\}$, most importantly $D_K \cap D_U = \emptyset$. In the course of retrieval, we use a large portion (nearly 80%) of D_U to compute the embedding database and use the 20% remainder as E_qs query signals.

Note that, to have a better understanding of results and to make sure we are testing methods in the same settings, we use an identical *seed* for splitting the dataset. Moreover, the retrieval set is identical for both baseline and unknown generalization settings. However in the matter of training set, as it has been mentioned before classes from 5 to 8 are removed for unknown generalization setting.

4.3. Evaluation Metric

In our experiments we quantifiably measure performance of our solutions for the problem of OTDR events retrieval. We remark Average Precision which in the context of retrieval is the portion of correctly retrieved instances. Formally, for a given query E_q as we retrieve K samples we compute average precision with the formula 4.1 where $\mathbb{1}$ denotes the characteristic function.

$$AvgP(K, E_q) = \frac{\sum_{k=1}^{K}(P(k, E_q) \; . \; \mathbb{1}\{y_k = y \; of \; E_q\})}{K} \tag{4.1}$$

Depict a scenario in which $K = 5$, for a give E_q we retrieve 5 samples, from these 5 samples those who belong to the same class of E_q are considered as True Positives and remainder are treated as False Positives. Indeed, sum of TP and FP are equal to K for a given query signal, so the denominator is always K. Thus, intuitively we can use the above formula and loop through all query signals to calculate the average precision.

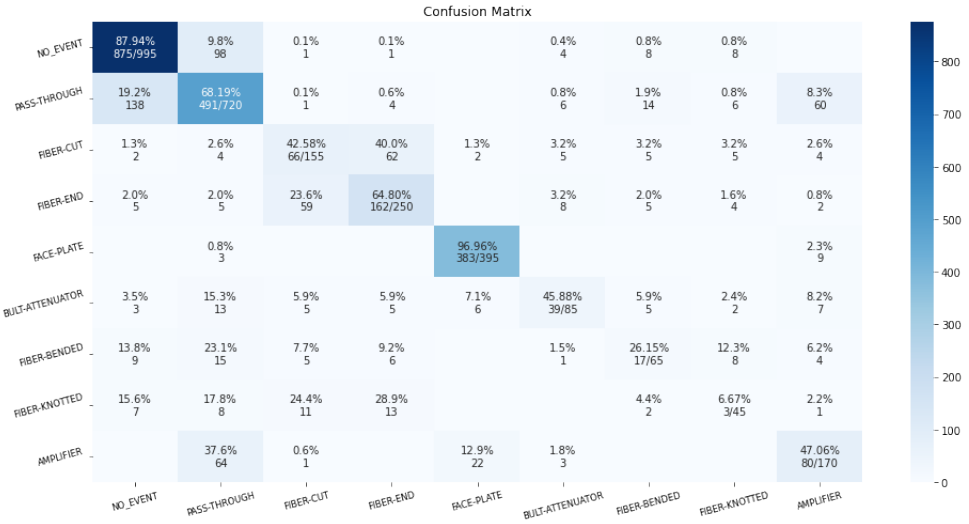
Concerning confusion matrix, we use confusion matrix functionality provided by scikit-learn[8]. According to the official API we need to specify two arguments y_true and y_pred that are ground truth labels and prediction labels respectively. In our experiments we determine $K = 5$ and for a given E_q we retrieve 5 similar samples. The y_true is the label of E_q and y_pred consists of 5 labels which belong to retrieved samples of the query E_q. So, through computing confusion matrix for all query signals we can obtain the percentage of correctly retrieved samples for each class.

4.4. Retrieval Performance in Baseline Setting

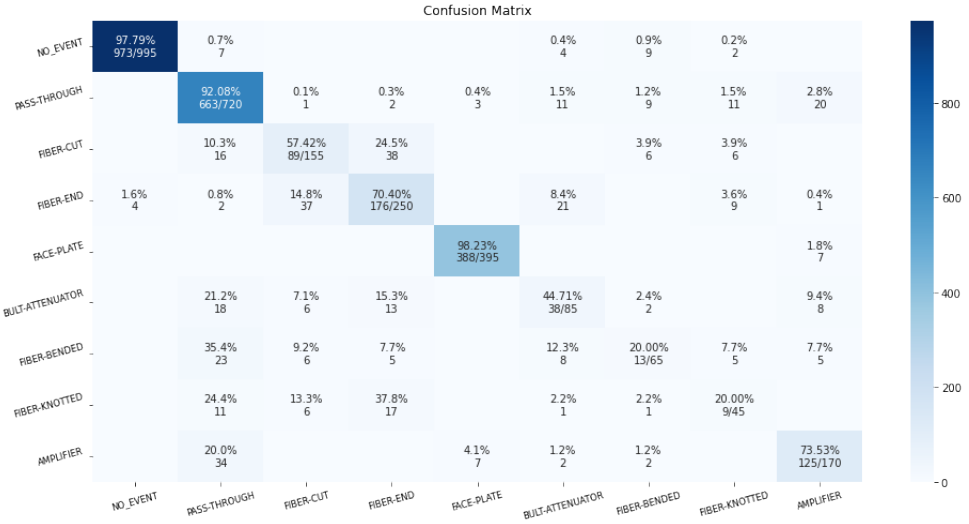
We begin evaluating our solutions against the baseline setting in which we train models using all events presented in the dataset D. The models are evaluated in completely identical environments to make the comparison fair, so we ensure that training and retrieval sets are the same for them. The average precision for each method is outlined in Table 4.1. It is evident that CNN and triplet semi-hard networks show significant performance in retrieval of similar signals compared to autoencoder model. This is no surprise since these two models are trained with supervision using labeled data. Nevertheless, the result of the autoencoder model is still promising by achieving 74% average precision.

	Learning Scenario	Avg. Precision
Autoencoder	Unsupervised	0.74
CNN	Supervised	**0.86**
Triplet semi-hard	Supervised	0.8

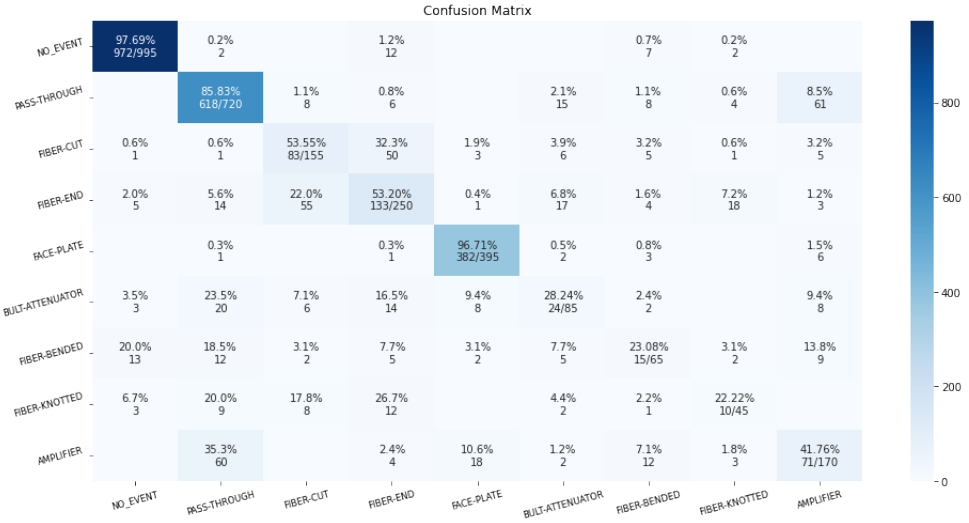
Table 4.1: Reported average precision for each model in baseline setting where $K = 5$.



(a) Confusion matrix of Autoencoder model where $K = 5$



(b) Confusion matrix of CNN model where $K = 5$



(c) Confusion matrix of Triplet semi-hard model where $K = 5$

Figure 4.3: Reported average precision of each event(class) for baseline setting.

Figure 4.3 represents confusion matrixes providing the average precision corresponding to each event for three models: autoencoder, CNN and triplet semi-hard. Additionally, we can observe what percentage of samples were incorrectly retrieved for query signal selected from different classes. The precisions of the last four events are lower than rest of the events, this is due to the fact that we are dealing with imbalanced classes in the dataset.

To get a better understanding of retrieval performance, we consider the CNN model and we retrieve 5 most similar samples from the embedding database. Figures 4.4 and 4.5 demonstrate retrieval results for two distinct events. The query signal E_q is plotted with the color blue and retrieved samples are plotted with magenta. From Figure 4.4 we can see all samples are retrieved correctly and they all belong to the same class of query E_q.

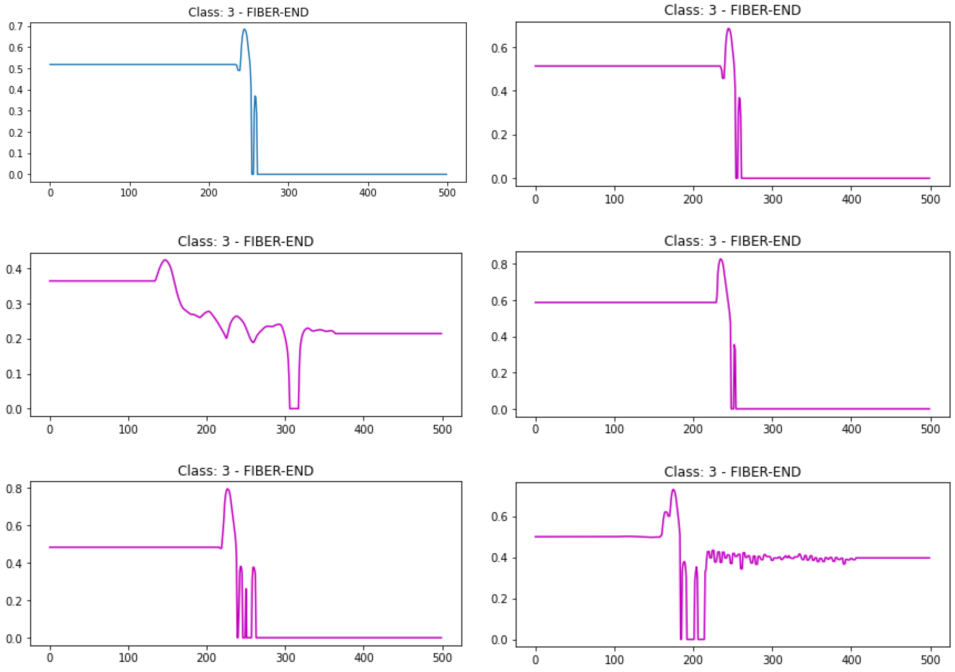


Figure 4.4: Samples retrieved for query signal belonging to event FIBER-END.

On the other hand, when we take a closer look at Figure 4.5 we can observe that only two samples belong to the same class of query signal. As highlighted previously some classes suffer from imbalanced data and class 6 (Fiber-bended) is one of them. Another potential problem with the dataset is that a few samples belonging to different classes overlap with each other and are quite similar. For instance, the sample belonging to class 7 (Fiber-knotted) is very similar to class 6. This issue has to be addressed in a future update of the dataset.

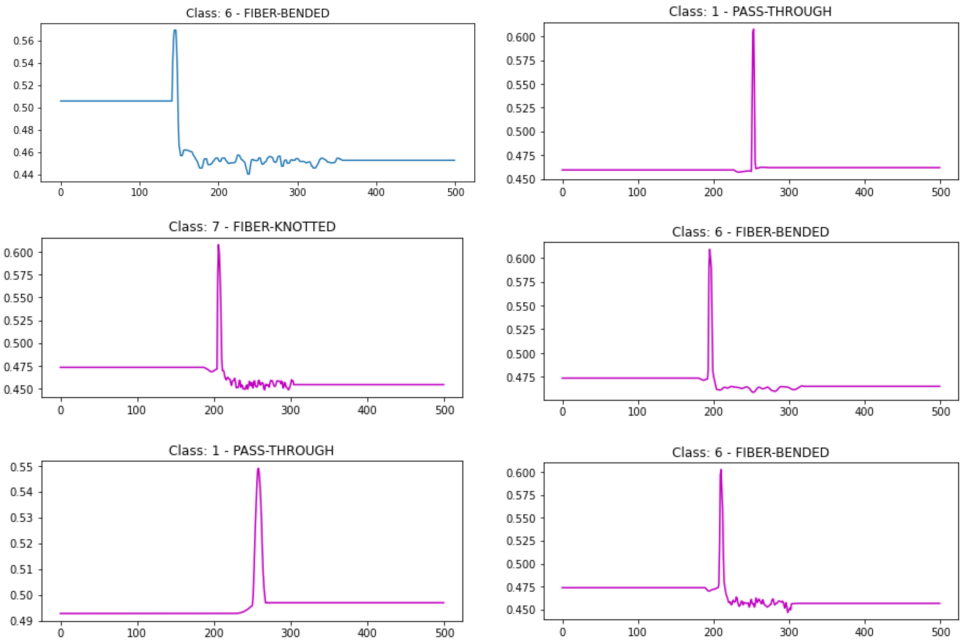


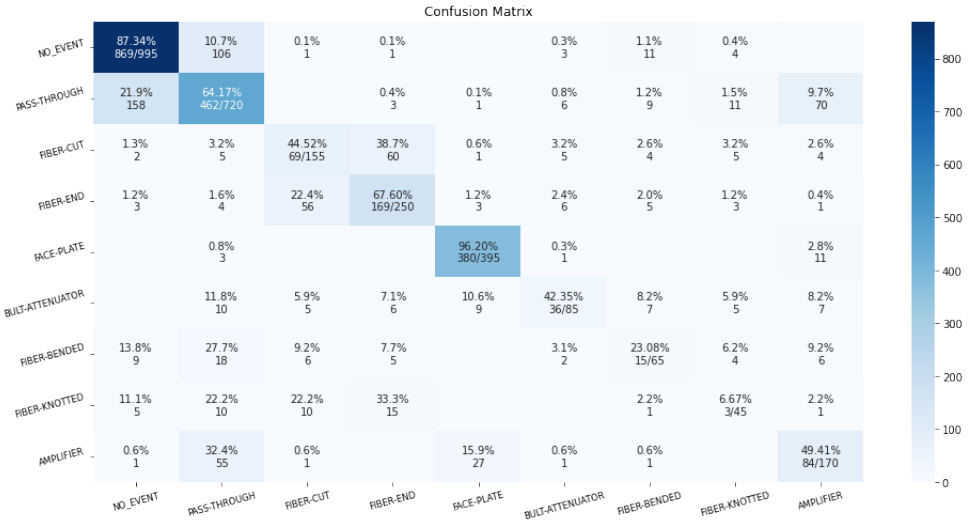
Figure 4.5: Samples retrieved for query signal belonging to event FIBER-BENDED.

4.5. Retrieval Performance for Unknown Classes

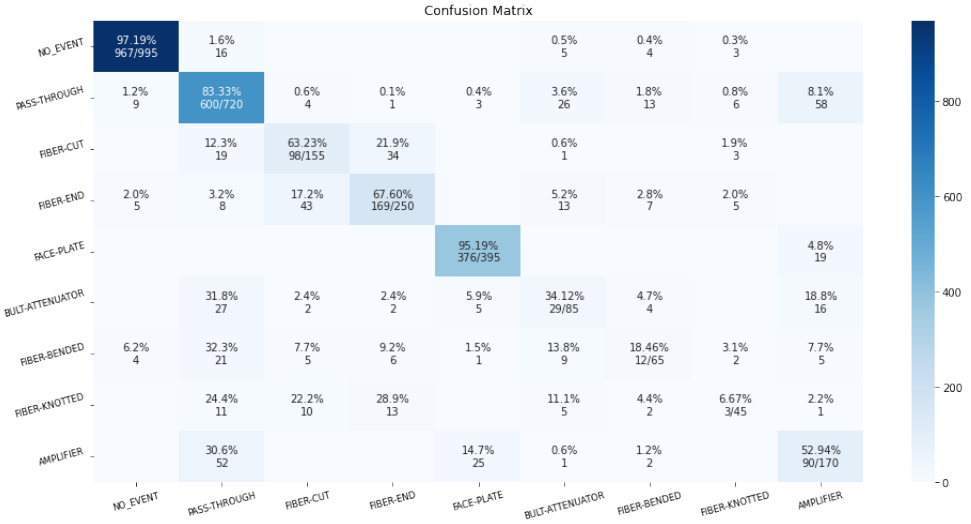
In this section, we report the results of experiments regarding unknown classes setting. We only train models on the first 5 events and during retrieval we use signals from all the classes with the aim of evaluating models' generalization on never-before-seen events. We re-emphasize that identical sets of data for training and retrieval are used. In Table 4.2 we provide figures corresponding to the average precision for each model. It is apparent that we see slight degradation in the overall performance of models since we have less training data due to the removal of signals belonging to the last 4 classes.

	Learning Scenario	Avg. Precision
Autoencoder	Unsupervised	0.72
CNN	Supervised	**0.81**
Triplet semi-hard	Supervised	0.79

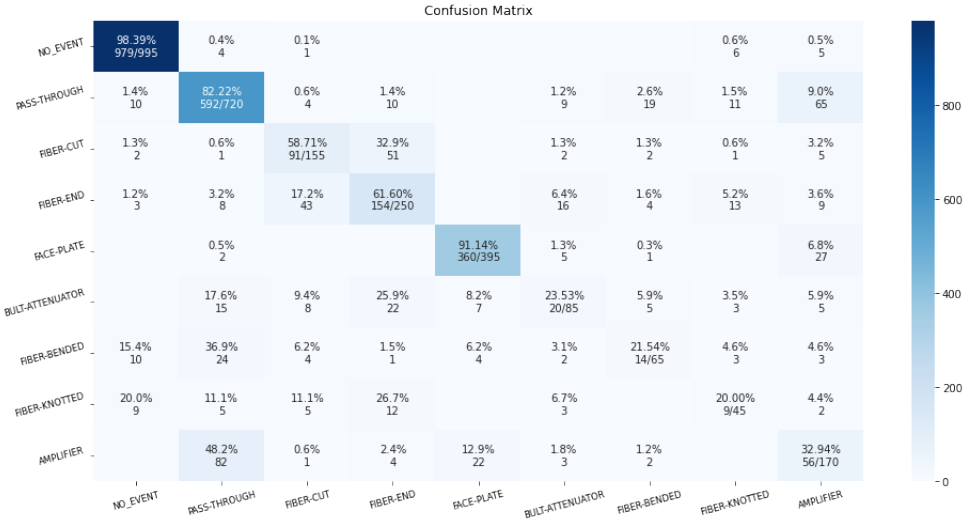
Table 4.2: Reported average precision for each model in unknown classes setting where $K = 5$.



(a) Confusion matrix of Autoencoder model where $K = 5$



(b) Confusion matrix of CNN model where $K = 5$



(c) Confusion matrix of Triplet semi-hard model where $K = 5$

Figure 4.6: Reported average precision of each event(class) for unknown generalization setting.

The Figure 4.6 provides confusion matrixes of models evaluated in unknown classes setting. The generalization of models on unknown events Bulk-attenuator and Amplifier are more acceptable than classes Fiber-bended and Fiber-knotted, unfortunately, if we look at class distribution in image 4.2 we can see these two events severely suffer from imbalanced data. Now let's consider the autoencoder model to examine retrieval performance by using query signal from class 5(Bulk-attenuator). Even if the model is not trained with signals from class 5 it was able to retrieve 4 out of 5 samples correctly.

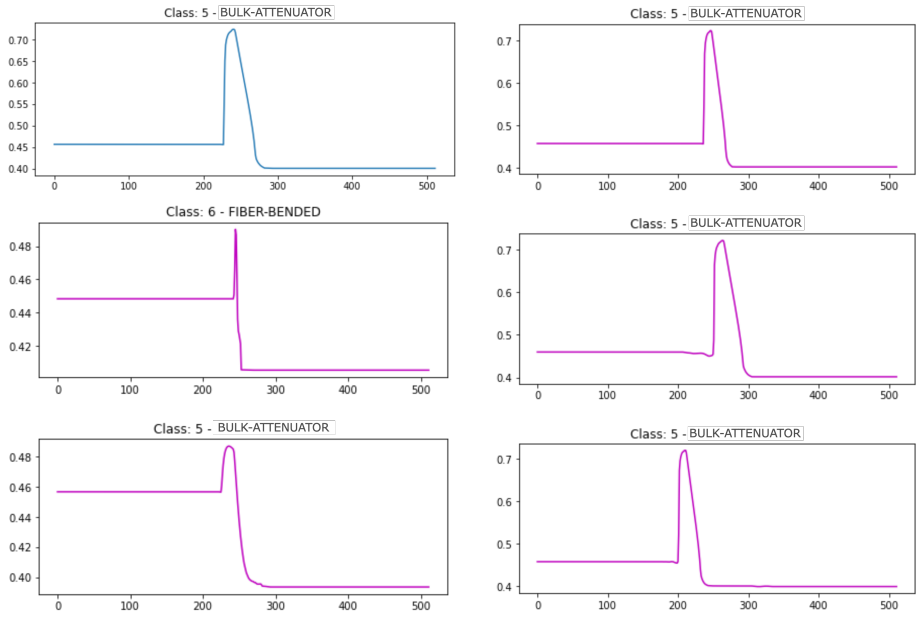


Figure 4.7: Samples retrieved for query signal belonging to event BULK-ATTENUATOR.

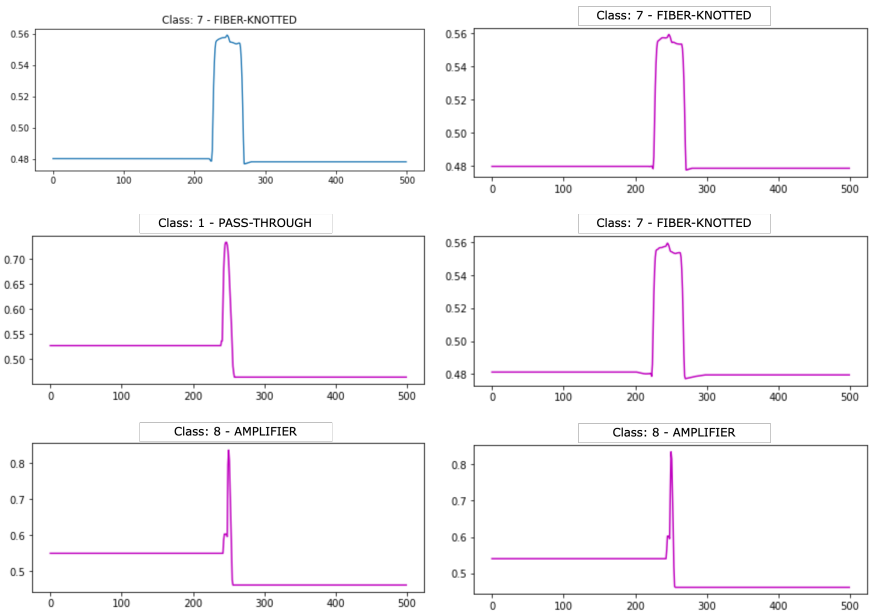


Figure 4.8: Samples retrieved for query signal belonging to event FIBER-KNOTTED.

In this current setting triplet semi-hard model has a better average precision on class Fiber-knotted than its counterparts. Let's consider this model and retrieve similar samples to E_q belonging to class Fiber-knotted. In image 4.8 we can see only two samples were retrieved with the correct class, this is expected as the average precision is 20%.

4.6. Frameworks

Deep learning architectures proposed in this thesis are implemented using Tensorflow[1] and Keras[5] frameworks. From training to testing and inference they use underlying standard APIs provided in these libraries. Concerning the retrieval part we use scikit-learn's[8] Unsupervised Nearest Neighbor API which supports three nearest neighbor algorithms namely BallTree, KDTree and Brute-Force. We pass algorithm argument as *auto*, so it will automatically determine which algorithm is suited based on the training set.

5 | Conclusions and Future Developments

In this thesis, we intended to address one-dimensional signal retrieval over the OTDR dataset. Our three solutions indicate promising results in different training scenarios depending on whether annotations for data are available or not. We used a convolutional autoencoder model for learning representation of unsupervised data. Next, we introduced CNN and triplet semi-hard networks for annotated data which consequently resulted in higher performance with respect to autoencoder model. The methods are evaluated on OTDR events and can be effectively used in industrial applications.

For future development, we want to improve existing solutions and explore their performance on different data distributions. It is also worthwhile to explore new deep learning architectures and learning techniques. In particular, we would like to investigate self-supervised methods and their possibilities for representation learning of data. In Appendix A, we provide a summary of self-supervised learning approach and briefly cover two important papers in this field.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. {TensorFlow}: a system for {Large-Scale} machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 265–283, 2016.

[2] S. Chen, J. Yu, and S. Wang. One-dimensional convolutional auto-encoder-based feature learning for fault diagnosis of multivariate processes. *Journal of Process Control*, 87:54–67, 2020.

[3] T. Chen, S. Kornblith, M. Norouzi, and G. Hinton. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*, pages 1597–1607. PMLR, 2020.

[4] X. Chen and K. He. Exploring simple siamese representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 15750–15758, 2021.

[5] F. Chollet et al. Keras. `https://keras.io`, 2015.

[6] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[7] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[8] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] A. M. Rizzo, L. Magri, D. Rutigliano, P. Invernizzi, E. Sozio, C. Alippi, S. Binetti, and G. Boracchi. Known and unknown event detection in otdr traces by deep learning networks. *Neural Computing and Applications*, pages 1–19, 2022.

[10] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015.

[11] J. Schuetzke, A. Benedix, R. Mikut, and M. Reischl. Siamese networks for 1d signal identification. In *Proceedings-30. Workshop Computational Intelligence: Berlin, 26.-27. November 2020*, volume 26, page 17. KIT Scientific Publishing, 2020.

[12] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[13] I. A. Siradjuddin, W. A. Wardana, and M. K. Sophan. Feature extraction using self-supervised convolutional autoencoder for content based image retrieval. In *2019 3rd International Conference on Informatics and Computational Sciences (ICICoS)*, pages 1–5. IEEE, 2019.

[14] T. Wen and Z. Zhang. Deep convolution neural network and autoencoders-based unsupervised feature learning of eeg signals. *IEEE Access*, 6:25399–25410, 2018.

A | Appendix A

A.1. Self-supervised learning

Self-supervised learning also known as predictive learning is a family of algorithms that learn representation from data itself meaning we do not provide labels to models as a form of supervision. Rather the idea is to design tasks according to the data modalities we are dealing with. For instance, given sequences of a video to a model, we expect the model to predict the next sequence. In the following we briefly explain two well-known research in the field of self-supervised learning SimCLR and SimSiam:

1. SimCLR was proposed by Chen et al. (2020)[3], a simple framework for learning visual representation relying on contrastive learning. They presented a scheme in which there are not any requirements for using specialized architecture or memory banks like in previous works. SimCLR outperformed state-of-the-art in Self-supervised field and even achieved matching performance of a supervised ResNet-50 on ImageNet. The representation is learned through maximizing similarity(agreement) between two different augmentations of an image via contrastive loss. Two augmented samples of an image are created which we consider as a positive pair. Then representations are extracted through a neural encoder. The projection head $g(.)$ maps the output of the encoder to space where we can apply contrastive loss, this head consists of MLP and ReLU non-linearity. As regard to negative pairs, they considered all the other augmented samples in a drawn mini batch $2(N-1)$. They carried out training with large batch sizes and bigger models, as for data augmentation they used a composition of random cropping and random color distortion.

2. Later, Chen et al. (2020)[4] used a simple Siamese network to learn representations in a self-supervised framework, and they called the new architecture SimSiam. This work improved the training procedure in several ways, it does not need negative pairs, large batch sizes or momentum encoder. In detail, this architecture takes two augmented images from random views. These images go through encoder f and then only one view is fed to a MLP head denoted by h. Next, they minimized the negative cosine similarity between $h(f(x1))$ and $f(x2)$.

List of Figures

2.1 VGG19 architecture 6
2.2 Residual block 6
2.3 Autoencoder architecture consists of encoder and decoder stacks denoted by f_θ and g_θ respectively. The latent vector Z is a representation of input X in a lower dimension. 7
2.4 Through the learning process, the **triplet loss function** minimizes the distance between anchor and positive samples which belong to the same class and maximizes the distance between anchor and negative samples. . . 9
2.5 Model structure proposed in FaceNet 9

3.1 Our proposed autoencoder network 12
3.2 Global Average Pooling operation 12
3.3 Our proposed CNN network 13
3.4 Triplet selection categories 14
3.5 High-level architecture of a retrieval system 16

4.1 Nine types of events in OTDR dataset 17
4.2 Distribution of events in OTDR dataset 18
4.3 Shorter caption 21
4.4 Samples retrieved for query signal belonging to event FIBER-END. 22
4.5 Samples retrieved for query signal belonging to event FIBER-BENDED. . 23
4.6 Shorter caption 24
4.7 Samples retrieved for query signal belonging to event BULK-ATTENUATOR. 25
4.8 Samples retrieved for query signal belonging to event FIBER-KNOTTED. 26

List of Tables

4.1 Reported average precision for each model in baseline setting where $K = 5$. 20
4.2 Reported average precision for each model in unknown classes setting where $K = 5$. 23

Acknowledgements

First and foremost, I would like to express my gratitude to Professor Giacomo Boracchi, for giving me this immense opportunity to accomplish my research thesis in his group. I appreciate his advice and guidance throughout the thesis.

I'd like to extend my thanks to co-advisor Antonino Maria Rizzo for his support and insight, certainly, his suggestions and comments helped a lot.

I'm also thankful to Politecnico di Milano for providing such an exceptional and memorable experience that one could ever ask for.

I'd like to acknowledge the Cisco Photonics in Italy for providing data to support this research project.

Finally, I could not have undertaken this journey without the unconditional love and support of my parents, I'm truly grateful for their encouragement and blessings throughout my entire life.