POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Salt Segmentation of Geophysical Images through Explainable CNNs

Tesi di Laurea Magistrale in Telecommunication Engineering - Ingegneria delle Telecomunicazioni

Author: **Francesco Maffezzoli**

Student ID: 944914
Advisor: Prof. Paolo Bestagini
Co-advisors: Dr. Vincenzo Lipari, Dr. Francesco Picetti
Academic Year: 2021-22

Abstract

Salt segmentation in seismic images is crucial to identify the correct velocity model that has to be used at migration time to ensure a correct image production. Up to now migrated images with incorrect velocity models are interpreted by an expert that has to pick salt bodies by hand, resulting in a slow and costly process. Salt segmentation process can be sped up using segmentation Convolutional neural networks (CNNs) to automate (at least partly) the process of salt picking. This thesis tackles the problem of automatic salt segmentation using deep learning. We first propose a salt segmentation CNN. In particular, we perform an extensive testing of different loss functions on a same baseline architecture. We analyze losses using multiple evaluation metrics to have different perspectives on how effective losses are. Then, we focus on the application of two eXplainable AI (XAI) techniques to inspect how salt segmentation CNN interpret different classes of horizon artifacts. In particular, we apply Activation maximization (AM) and Network inversion (NI) methods to previously trained segmentation CNNs, producing synthetic images that shows how trained CNNs interpret salt bodies, background and other seismic artifacts. AM technique has been mainly used to produce class level images, finding the input images that activates class neurons the most. NI on the other hand shows how an input image is processed by the CNN layer by layer. This technique can be used to find out what part of information is lost and what is kept from one layer to another.

Keywords: CNN, Salt segmentation, Geophysics, XAI,

Abstract in lingua italiana

L'identificazione del sale nelle immagini sismiche è un passo cruciale per la valutazione del corretto modello di velocità da applicare durante il processo di migrazione, assicurando così una corretta produzione delle immagini migrate. Allo stato attuale la correzione del modello di velocità dei migrati è affidata ad un esperto che ha il compito di identificare i corpi salini manualmente, risultando in un processo lungo e costoso. Il processo di segmentazione del sale può essere velocizzato considerevolmente usando delle reti neurali convoluzionali per automatizzare, almeno parzialmente, il l'iter di identificazione del sale. Questa tesi affronta il problema di segmentazione automatica dei corpi salini tramite deep learning. Per prima cosa proponiamo un'architettura di rete neurale convoluzionale da utilizzare per la segmentazione, in particolare focalizziamo l'attenzione sul testing e la valutazione di diverse funzioni di loss (a parità di modello) utilizzando svariate metriche, per avere prospettive differenti sull'efficacia di ciascuna di esse. Nella sezione successiva analizziamo in che modo le reti neurali convoluzionali precedentemente allenate interpretano le diverse classi di fenomeni sismici presenti nelle immagini tramite l'utilizzo di due diversi metodi di XAI. In particolare applichiamo le tecniche di AM e NI alle reti allenate, producendo immagini artificiali che permettono di comprendere (almeno in parte) come le reti interpretano le diverse classi sismiche, fra cui i corpi salini. Nello specifico la tecnica di AM è stata prevalentemente utilizzata per produrre immagini interpretabili delle classi finali: la tecnica consiste nel trovare l'immagine in input che massimizza l'attivazione di uno specifico neurone di output. La tecnica di NI al contrario mostra come una rete neurale convoluzionale processa l'immagine in input layer per layer, può essere dunque usata per comprendere quale parte di informazione viene scartata e quale ritenuta da un layer al successivo.

Parole chiave: Rete neurale convoluzionale, Segmentazione del sale, Geofisica, XAI

Contents

1 | Introduction

With the increasing need of hydrocarbons worldwide, oil and gas industries are collecting several hundreds of gigabytes of seismic data to find possible oil and gas reservoirs. Up to now, to find possible locations of underground hydrocarbons, the collected data needed to be processed by field domain experts, such as geologists and geophysicists. This manual process of analyzing huge amount of data is extremely costly and time consuming for gas industries, but that is not the only problem. Indeed, it has been known for quite a while that gas and oil reservoirs are often located near underground salt basins, which are notoriously difficult to image correctly due to their elastic properties. In order to correct the seismic images and generate decent sub-salt migrates, salt bodies position should be known a priori.

The problem of locating oil from seismic images is characterized by three time-consuming steps: identify salt basins in poorly migrated images by hands; correct those images with the previously gathered information of the salt position; interpret the newly generated images by hand to find reservoirs.

The need of a helping hand to classify and process a large amount of seismic data led oil industries to start researching on deep learning as an alternative and faster way to recognize salt basins in seismic images. Deep learning approaches offer the possibility of analyzing a large amount of data in almost no time, especially compared to human experts. The task of picking salt bodies in seismic images can in fact be considered a segmentation problem, and can be efficiently solved by deep learning algorithms (like CNNs) efficiently. CNNs are widely used in automatic image processing and can be very efficient in terms of training and test time. They have been proven to be able to outperform humans in image classification tasks, setting themselves as the perfect substitute to perform this kind of time consuming problems.

However seismic images can be very different from natural images and can vary a lot from one another depending of the subsurface lithology and acquisition method used. This unpredictable behaviour makes trained CNNs to be highly inefficient on new data. For these reasons the research procedure on salt bodies identification is still very far from perfection and cannot still substitute fully human experts in the field. However, CNNs

can still assist geologists and geophysicists in their work, substantially speeding up their work.
This thesis tackles the problem of salt segmentation in seismic images by means of deep CNNs. In particular, we first propose the use of CNNs to identify salt bodies. Then, we focus on interpreting the cognitive process of the used CNNs by means of XAI. In particular our contributions are:

- We propose our solution of salt segmentation problem using CNNs, investigating in particular on the impact of different loss functions on segmentation performances. Proposed results on the salt segmentation problem aim to improve a baseline provided by us by ENI.
- We discuss about how trained CNNs recognize salt bodies using two different XAI techniques. By comparing results obtained on classification and segmentation CNNs we reason about how differently those two CNNs interprets images.

The document is structured as follows. In Chapter 2 we provide an introduction to deep learning and geophysical imaging, explaining how the seismic images are formed. In Chapter 3 we go through a brief review of the state of the art review, both in terms of XAI and salt segmentation; in this chapter problems related to salt segmentation are introduced for the first time. In Chapter 4 we propose our own salt segmentation pipeline: first we formally introduce the problem of salt segmentation, then we describe in detail every step made from input preprocessing to metric evaluation, motivating made decisions. In the last part of Chapter 4 a description of all tested losses is provided, along with their graph and mathematical intuition. In Chapter 5 we explain in detail the chosen XAI techniques used to perform inspection on trained CNNs. Chapter 6 is perhaps the most important of the thesis: we begin with an overview of the two datasets used, followed by a description of all metrics used for segmentation performance evaluation. Then we will explain how experiments have been carried out, talking about the experimental set up for segmentation, classification, and CNN inspection. The last part of Chapter 6 reports obtained results, from segmentation performances using the above defined metrics to XAI generated images. In the last chapter we make some final consideration on the work done, adding some interesting suggestions on future works to improve both segmentation performances and CNN inspection.

2 | Background

This chapter introduces some background concepts needed to understand the rest of the work. In particular, we first introduce the main concepts behind seismic imaging. Then, we introduce the main ideas behind deep learning.

2.1. Geophysical imaging

To understand the relevance of the proposed work it is important to understand how geophysical data is collected and how images are formed. In this section we first go through a brief recap about how data is gathered with the help of geophones, then we introduce some techniques which transform the geophysical data into migrated images, introducing the problem of correct velocity model identification and how salt bodies interfere with this process. An interested reader that wants to dig deeper into this concepts can refer to Etienne Robein book on Seismic imaging [33].

Reflectors and Diffractors

The underling idea below Geophysical Imaging is pretty simple: known the velocity model of the media, one can reconstruct the underground structure by looking at how reflected and diffracted waves propagate in the subsurface.

First thing to do is to define the two main geophysical events present in the underground: reflectors and diffractors. Reflectors are the interface with two geological layers with different elastic properties. Since the two materials have different impedances, an incoming wave traveling from the surface will be partially reflected and partially refracted. Since reflectors usually have some kind of lateral continuity, the resulting reflected wave will have a straight shape in seismic images. Diffractors are usually small-sized objects with respect to the wavelength or points that correspond to the limit of reflectors. Diffractors reflect incoming waves with a hyperbolic-like shape. Figure 2.1 shows an example of reflector and diffractor.

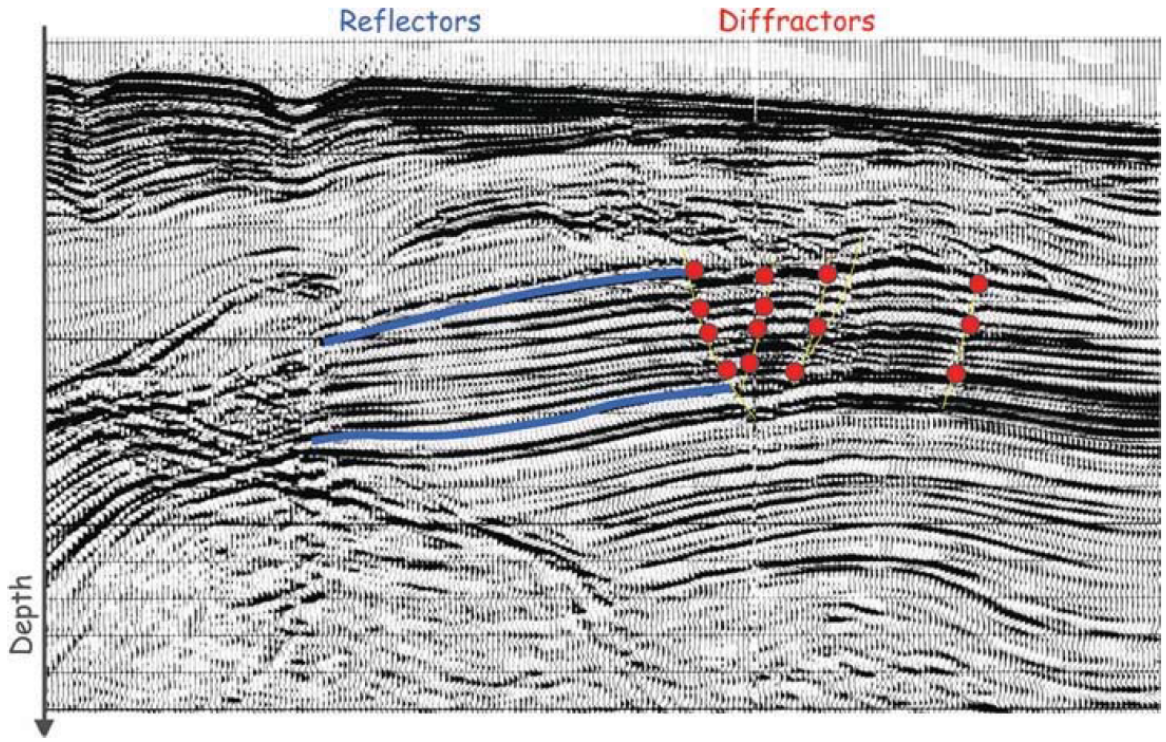


Figure 2.1: In blue, reflectors with their characteristic lateral continuity. In red, Diffractors points.

Knowing how reflected and diffracted waves prapagate theorugh the media, one can exploit this knowledge to map the subsurface morghology by generating incoming waves from the surface and recording the reflected wavefronts over time.

Data gathering

A typical setup used to obtain seismic data is the following: an source is placed in a specific position on the ground, followed by a set of receivers according to a pre-designed acquisition geometry.
When a signal is released from the source, the receivers (usully geophones) capture the pressure coming from the subsurface, creating a sequence of traces of the pressure level over the surface for a period of time t. The resulting raw data obtained from this process is a series of reflected waves recorded by each geophone at every instant t.

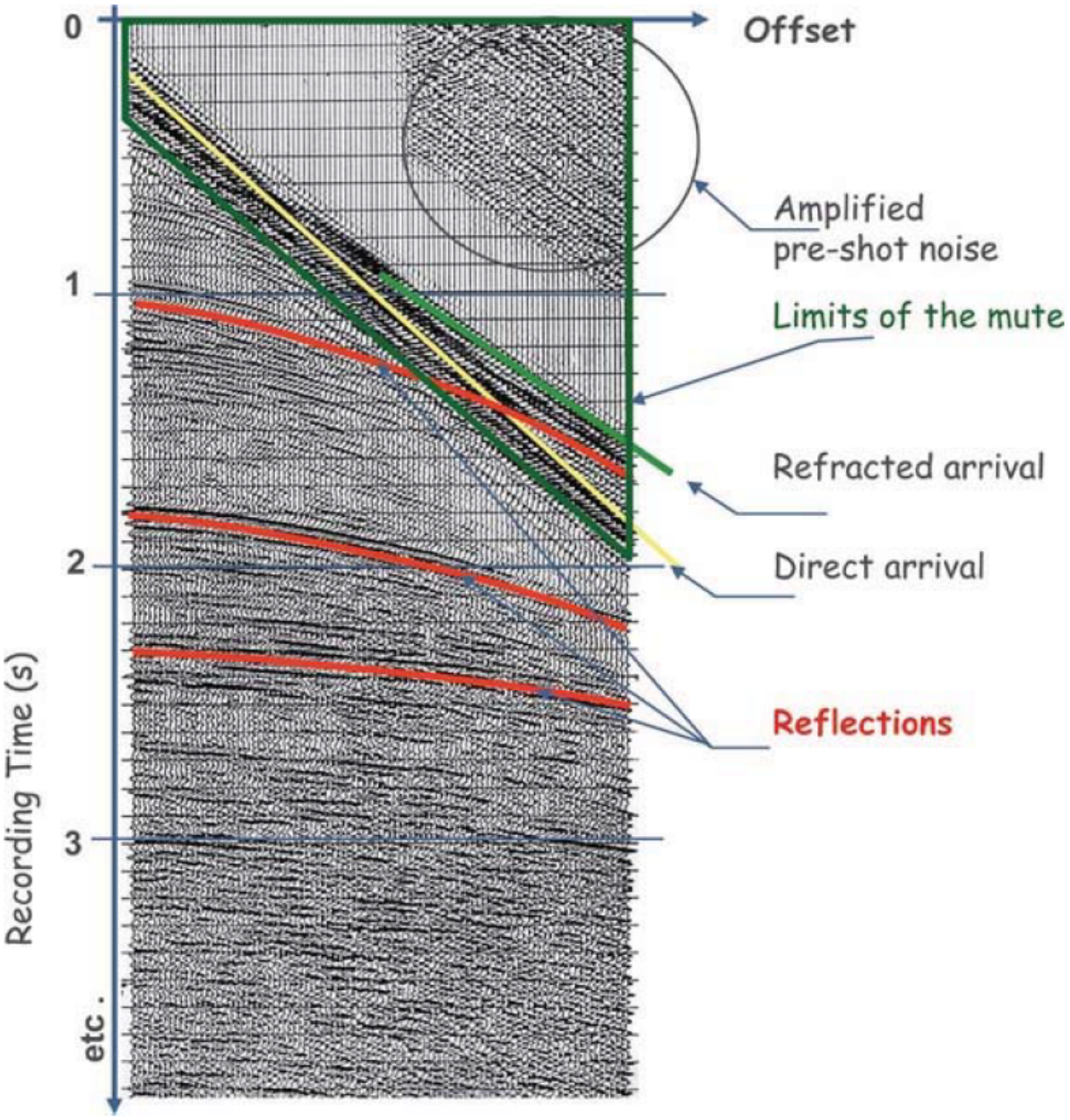


Figure 2.2: Typical data collected by a geophone. The first recorded signal is the direct arrival of the shot, followed by the underground reflections. Note how reflections arrive with a small delay from one another, giving us an information about the reflection event depth. All the noise recorded before the shot is muted.

Figure 2.2 shows how different types of waves are registered at the receiver. Since we are interested only in reflected waves, pre-shot noise and direct arrivals are muted. Note also the definition of the axis: the y axis represents the elapsed time while the x axis

represents the offset, that is the distance between the source and the receiver. Note that the elapsed time gives us the information about reflection depth: at same offset, the more time has passed the deeper the reflection has occurred.

Data migration

If the velocity model of the media is known, waves propagation through the media has a predictable behavior: known the distance between the emitter and receiver, the angle with the vertical (group angle) and the total elapsed time between the shot and the detection, one can compute the reflection depth. The process of reconstructing the correct reflector position given the set of wavefields at the surface is called migration. Knowing the velocity model of the medium is crucial to perform a correct migration: the velocity model provides information about the propagation speed and direction of the reflected waves. Furthermore, the medium is usually anisotropic and heterogeneous (wave propagation velocity depends on its location and direction), .
Migration can be performed using several different moethods, in this chapter we briefly discuss about two of them: Kirchhoff migration and Reverse Time Migration (RTM). The common migration technique known as Kirchhoff Migration is based on the hypothesis that all seismic points can be considered as diffractors. Diffractor points, for constant propagation velocity, reflect seismic waves in all direction. Those waves are recorded by geophones with different delays, causing the seismic profile to be hyperbola-shaped, as shown in Figure 2.3.

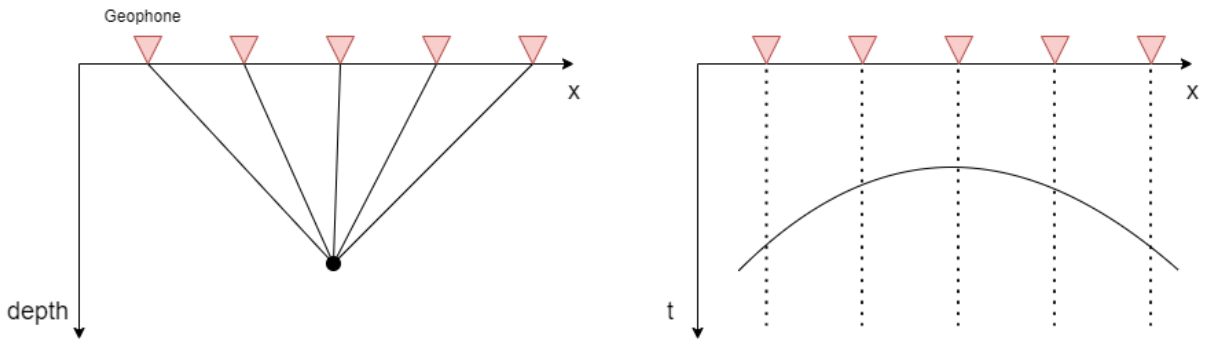


Figure 2.3: Hyperbolic seismic profile of a diffractor point. The image on the right shows how a diffracted waves propagates through the media reaching the geophones with different delays.

This process now have to be reverted using migration, that is, finding the location of the diffractor point by looking at his hyperbolic seismic profile.
For each diffractor point the hyperbolic profile is computed and summed with those of all other points in the image.
Reflectors, which can be considered as multiple diffractor points close to each other, appear as multiple overlapping hyperbolas that sum up generating an horizontal line in the reflector position. On that line all points sums thanks to constructive interference, hyperbolas tails on the other hand sums in noisy ways.
RTM technique directly solves acoustic wave equations to simulate down-going and up-going wavefield propagation, constructing in this way the so called "Wavefield Movie". To find reflectors locations, RTM exploits the famous Imaging principle [10]: "Reflector exists at points in the ground where the first arrival of down-going wave is coincident with an up-going wave". This means that if we manage to reconstruct down-going and up-going waves, we can find the position of reflectors by finding points in time where the two waves coincide. The workflow for each shot is the following:

- Simulate the down-going wavefield: knowing the velocity model of the media, one can compute the down-going wavefield by simply solving the wave equation of the shot recursively from shot time. While solving down-going wave equations, up going waves disturbance is neglected.
- Reconstruct the up-going wavefield: Given a shot gather, that is the trend of the up-going wavefield pressure over time (Example in Figure 2.2), the process of up-going wavefield reconstruction runs in reverse time. Starting from the maximum recording time and going backwards, wave equations are solved iteratively imposing shot gather amplitudes as boundary conditions for the problem. The wavefield that we extrapolate in this way is the up-going wavefield that generated the shot gather.
- Compare the two reconstructed wavefields imposing the imaging principle to find reflectors position.

RTM migration results to be more precise when compared to Kirchhoff migration, at the cost of computational complexity. As mentioned, the underling assumption that has been made is that the subsurface velocity model is known. Only in this way it is possible to compute diffraction curves for every point and sum them up correctly. For non-homogeneous media the diffraction curves of seismic points are not perfect hyperbolas but complex shaped curves. How is it possible to know if a velocity model fits the seismic problem or not?
To check if the chosen velocity model at a specific point is correct, one can use multiple

measurements performed with different offsets and compare them together: the set of traces collected using different offset ranges is called Common Image Gather (CIG). CIG represent the image obtained at a specific points with different offsets using the same velocity function.

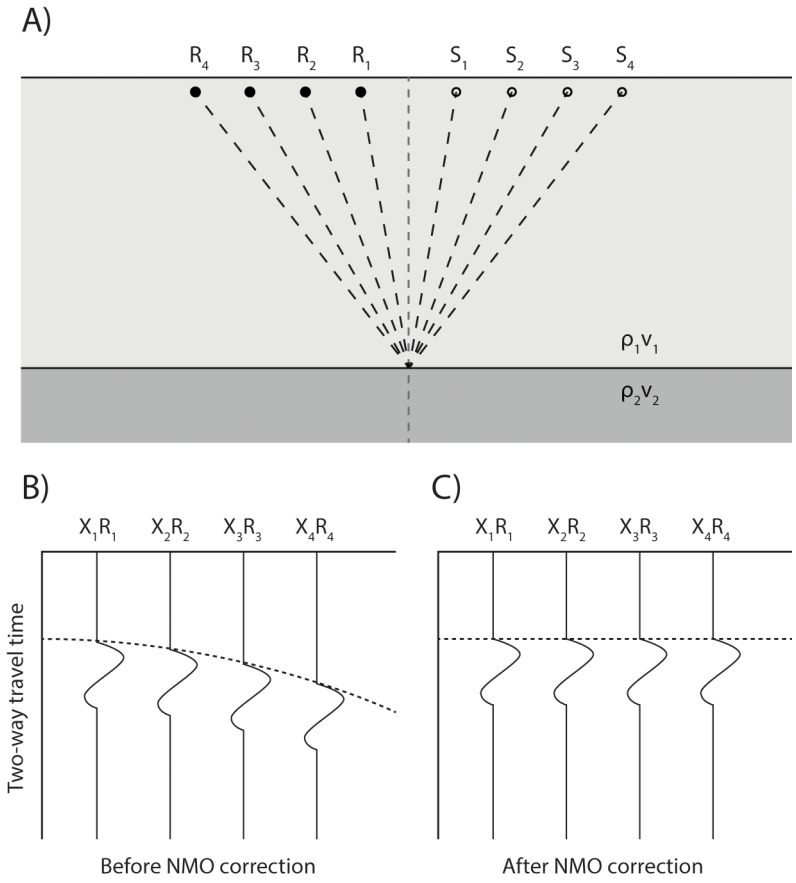


Figure 2.4: A) Series of shots taken with different offsets that form a CIG.
B) Signals belonging tho the same reflection event captured at different offset (the reflection event detection is represented as a dashed line). The vertical time is different for each offset. C) CIG stack after Normal move out (NMO) correction. Note how vertical time coincide for every offset, resulting in a flat CIG (flat dashed line).
Image courtesy of [17].

Since all measurements taken are the result of the same geophysical event, all CIG traces must migrate to the same point (Figure 2.4 A)). Once the velocity model is applied to all the CIG traces, a resulting reflector should appear at the same "vertical time" for all

offsets used. After correcting the vertical times using the selected velocity model (NMO correction [1]) all the obtained reflection signals are horizontally juxtaposed: if the velocity model is correct, signals belonging to the same reflection event should arrive at the same time for all offsets, causing CIG to generate a flat line (image 2.4 C). A CIG made only of flat lines is ideal, and tells us that the velocity model used is universally correct for all offset, thus the migrated image is reliable. Salt bodies introduce a sudden variation in the subsurface velocity model, this happens because elastic properties of the salt are usually very different from all the other geological materials. If the salt velocity model is not correctly associated with salt bodies at migration time, everything below salt is wrongly migrated.

To cope with this problem, one can first produce migrated images without considering salt at all. In this way the migrated image will be correct only up to the top salt perimeter. Once the upper perimeter is known, salt velocity model is assigned from the upper perimeter to the end of the image. Following the same principle, the migrated image will now be correct up to the bottom salt perimeter.
Now that the whole salt body has been identified one last migrated image is produced, assigning the correct velocity model to the whole image.

2.2. Deep learning

Machine learning (ML) is the field of study that gives machines the ability to solve complex problems without having to code them [36].
In standard ML algorithms, the features of the dataset are usually extracted by humans and then fed to a learned classifier to compute the output. This approach not only requires time-consuming handcrafted feature extraction, but there is another problem: optimal features for humans are not necessarily optimal algorithms. Deep learning differs from standard ML algorithms because feature extraction is performed by the machine itself (2.5).

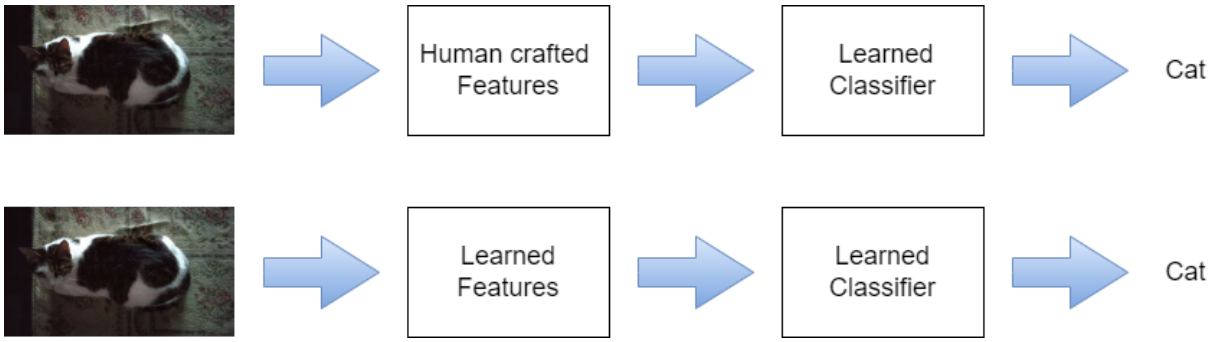


Figure 2.5: Top and bottom pipeline show respectively the common ML and Deep learning approach. What differentiate the two pipelines is the feature extraction step: standard supervised ML requires external feature extraction while Deep learning algorithms learn features from the data.

2.2.1. Artificial Neural Networks

Artificial neural networks (ANNs) are Deep learning algorithm inspired to human brain: the building block of ANNs is the perceptron, a binary classifier that, given an input, outputs a binary output. [35]. More specifically, a weight vector w is multiplied by scalar product with an input vector x. The result is then fed into an activation function f that provides the perceptron output (equation 2.1). Formally, the output of a perceptron is defined as

$$P(x|w,b) = f(w^T x + b), \tag{2.1}$$

where x is the input vector, w the weight vector, b the bias vector, and f is known as activation function,

The so called Activation function f is usually a threshold function that either set the output to zero or not depending on the input value, moreover f must be differentiable to correctly perform gradient descend at learning time. Perceptrons (neurons) are the building blocks of ANNs, enabling complex processing by stacking them together. ANNs networks are organized in layers that process the input sequentially: in dense networks, each neuron output is an input for all the next layer neurons, processing the input and extracting more and more complex feature up to the last decision layer.

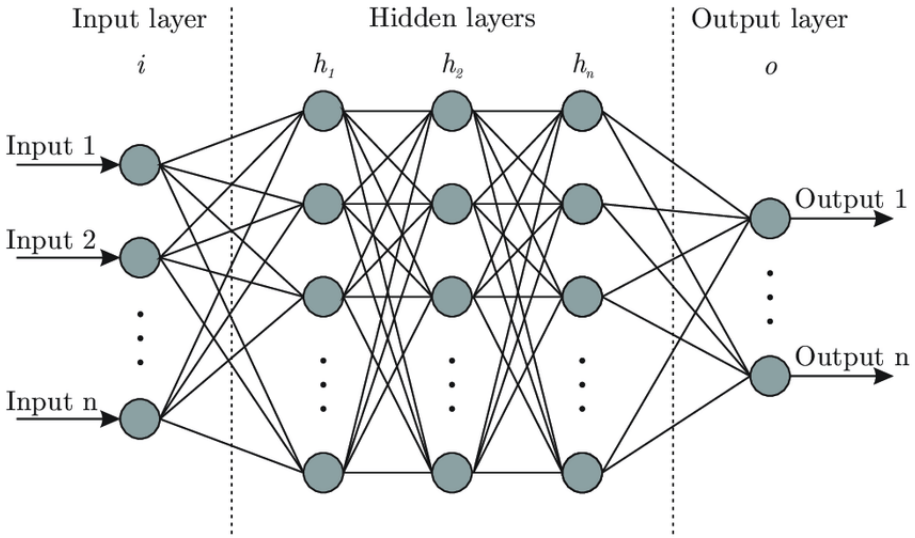


Figure 2.6: General ANN architecture [7], composed of one input and output layer with many hidden layers in between.

How can ANNs produce a meaningful output?
At the beginning all ANN parameters, namely weight vectors w and bias vectors b, are randomly initialized. With random initialization the computed output will probably be meaningless, but through learning the ANN gets better and better by tuning weights and biases until good enough performances are reached. Learning is the process of updating weight and biases iteratively by minimizing or maximizing a specific cost function.
Cost function can be chosen depending on the problem that has to be solved, and affect ANN performances greatly. In general we can separate an ANN task in two categories, regression and classification, depending on the desired output: classification tasks expect discrete outputs while regression tasks expects continuous ones. Once the loss function is computed, ANN parameters are updated using gradient descend technique. If the cost function is differentiable, one can compute partial derivatives of the cost function with respect to all weights and biases and update them iteratively. Given a loss function L with input x_i^k and target value t_i^k the iterative update is called backpropagation and is computed using gradient descend formula as [41]:

$$w_i^{k+1} = w_i^k + \Delta w = w_i^k + \eta \frac{\partial L(x_i^k, t_i^k)}{\partial w_i} \tag{2.2}$$

Equation 2.2 represents backpropagation at time k with learning rate η.

2.2.2. Convolutional Neural Networks

Although ANNs allow us to process a vast diversity of data more complex computations requires bigger and deeper networks, this phenomenon greatly impact ANN performances both in terms of training time and performances: for example, processing an image with a standard ANN would require an input neuron for every pixel, causing the number of trainable parameter to grow bigger and bigger.
CNNs cope with this problem introducing the concept of convolutional layers, trainable filters that perform image convolution on the input image, and pooling layers.

- **Convolutional layer:** A convolutional layer is a stack of n trainable filters of dimension $d \times k1 \times k2 \times n$, where d is the depth of the input and $k1 \times k2$ is the filter kernel size [3]. The filtering operation performed by convolutional layers project an input (typically an image) of size $d \times l1 \times l2$ onto a feature map of size $n \times k1 \times k2$. Each filter of the convolutional layer learns to extract a specific feature from the input; intermediate states between convolutional layers are in fact called feature maps since each processing stage extracts more and more complex features from the previous one. The advantage of convolutional layers over dense layers is the number of trainable parameters: convolutional layers only require $n \times k1 \times k2 + n$ parameters to process an input image of arbitrary dimension.
- **Pooling layer:** Pooling layers are used to reduce the spatial dimensionality of the input by summarizing features extracted in previous convolutional layers using a variety of methods. Max Pooling layers [26] for example substitute portion of non overlapping region of the input image of size $(K1 \times K2)$ with their maximum value. Pooling layers effectively reduce the spatial dimensionality of the input without requiring any trainable parameters.

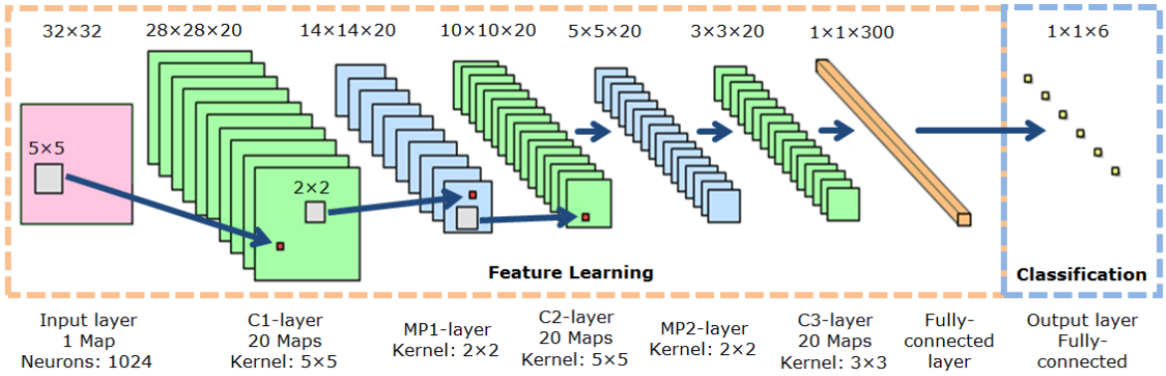


Figure 2.7: CNN architecture, including an encoder and a classification head. Convolutions with different kernel size are stacked with pooling layers. The last convolutional layer is then flattened and connected to a dense classification layer.

Figure 2.7 is an example of a CNN used for classification purposes. By alternating convolutional and pooling layers the CNN can extract more and more complex features from the input image while at the same time reduce its spatial dimensionality. The convolutional and pooling part of a CNN is called Encoder, which outputs a highly compressed and information dense stack of images that can now be processed using standard dense layers by applying a flattening or Global Average Pooling (GAP) operation.

3 | State of the art

This chapter contains an overview of state-of-the-art techniques for salt segmentation and the main XAI tools used in this work.

3.1. State of the art for salt segmentation

The identification of salt basins in migrated images is a crucial step to determine the correct velocity model to be applied at migration time.
Currently a field expert has to pick salt manually on incorrectly migrated images, as the process has not yet been fully automated. Deep learning can be used to automate the process of salt identification in images, consistently speeding up the process of salt identification. Fist, let us differentiate the three different steps of salt segmentation both with human and CNNs approach.
As stated in Chapter 2, salt segmentation is composed of two different salt picking phases, after each of them the mask salt is provided to the migration algorithm that adjust the velocity model of the migrated image taking into account the salt perimeter.

- Top salt body recognition: top salt perimeter is picked on wrongly migrated images. Then, a new velocity model is built considering as salt all the pixels below the top perimeter. Once the new velocity model is available, a new migrated image is produced.
- Base salt body recognition: bottom salt is picked from new migrated images. Now that the whole salt body has been identified, a last correctly migrated image is produced using the salt velocity model only in salt bodies positions.

CNNs for salt segmentation can achieve human level accuracy in some specific situations but greatly suffer dataset domain shifting [18], meaning that to effectively use CNNs for salt segmentation, humans must first interpret a portion of the data set to create an ad hoc training set for fine tuning. This preliminary salt picking process is greatly time consuming and severely decrease the utility of CNNs for salt picking, since heavy human intervention is still required.

Domain shifting is already a known enemy for CNN generalization, but in the case of Salt segmentation this problem is even more pronounced. For example, seismic images generated with different migration techniques (or that differ greatly in noise distribution and contrast) can lead to very different segmentation results. Figure 3.1 [37] shows salt segmentation on images migrated with two different migration techniques.

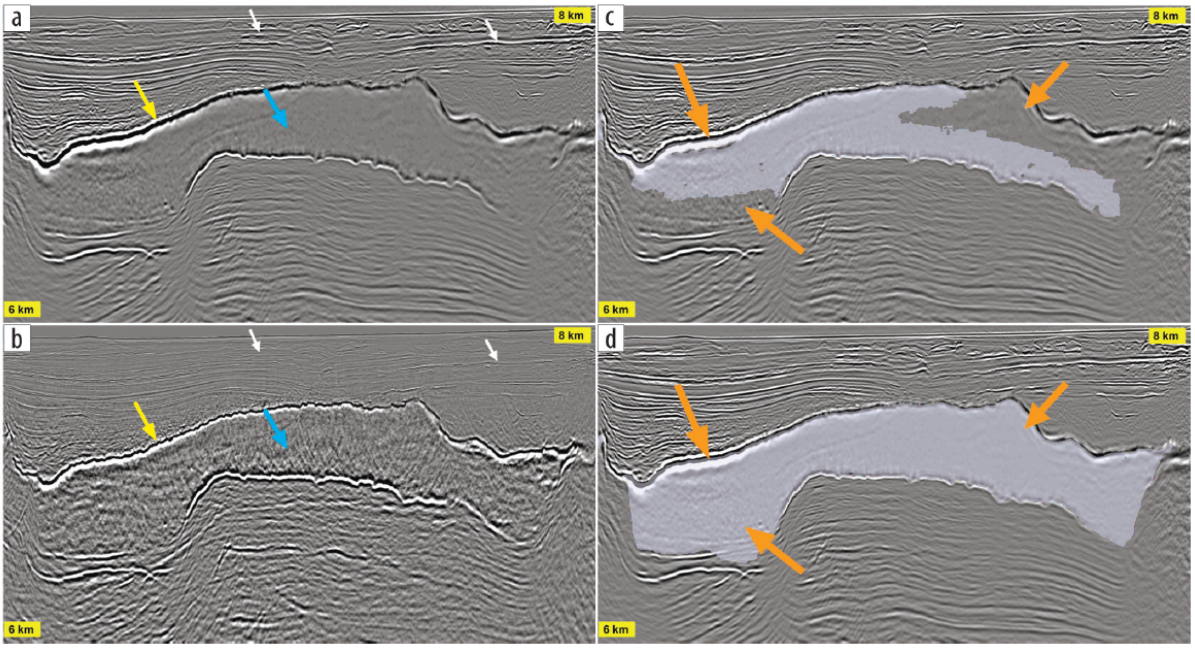


Figure 3.1: Example of the same salt body migrated with two different migration algorithms, respectively RTM (a) and KDM (b). Images (c, d) shows how the same CNN performs on the two different images. The orange arrows on the right show portions of the salt body that have been classified differently when using the two different migration techniques. Arrows on the left highlight the seismic image composition: white arrows represent the background, the yellow and the blue arrow delineate the top and bottom perimeter of the salt body.

Note how texture in images (a) and (b) differ due to different contrast and noise distribution. To tackle the problem of domain shift a full learning pipeline called SaltNet [37] have been proposed. SaltNet copes with the absence of unlabeled data from the new dataset by generating proxy labels for CNN training using semisupervised learning: an ensemble of five pretrained CNNs with different architectures is used to generate an initial prediction on the new, unlabeled dataset. Using CNNs ensembles usually provides better

performances than single a single CNN, in this case different architectures have been used to provide different views on the target dataset.
Once proxy labels are available, better performing CNNs are selected and retrained on the new augmented dataset. To add diversity during re-training the selected CNN architecture is slightly modified, to introduce novelty in the predictions and improve domain shifting performances. The idea of self training was at first introduced by Babakhin ed al. [4], presenting a semi-supervised training approach and winning the TSG Salt identification challenge. The proposed self training procedure is divided into K iterative rounds divided in tree steps:

- CNNs training: Train an ensemble of CNNs using the available labels. During the first round only ground truth labels will be available.
- Pseudo-label prediction: Predict a set of pseudo-labels using previously trained CNNs in an ensemble fashion by averaging. A confidence level is computed for each pseudo-label in order to discard those that are too unreliable.
- Dataset extension: Extend the previous labeled dataset with the new pseudolabels.

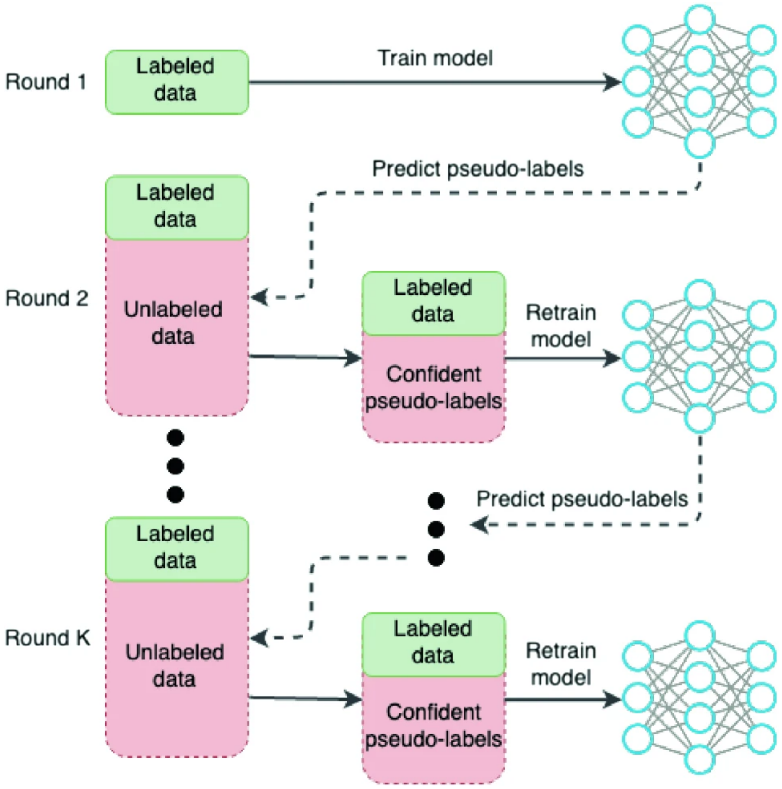


Figure 3.2: Self training pipeline for salt segmentation CNN. [4] Every round is made of two steps: first the model is trained on the labelled part of the dataset, then the trained CNN predicts new data generating pseudo labels.

On the architectural side, Sen et al. [37] performs an extensive test of multiple architecture during ensemble predictions. The best resulting CNN architecture comes up to be ResNet34 [19], the same architectural choice was made by Babakhin ed al. [4], that used U-ResNet34 and U-ResNet50 to perform self-training.

3.2. XAI state of the art

The reason why Deep learning is so widely used is that it does not require direct coding of hand-crafted features. Trained algorithms are therefore often considered "black boxes" since developers do not need to explicitly code the the solution of a problem but just need to take care of the training procedure. This black box approach is for sure convenient since it enables flexible solutions for complex problems but makes it hard to interpret and understand why and how CNNs work. The field of XAI studies methods and techniques to shed some light on those black boxes, making them more reasonable and interpretable

to human eyes.
XAI comprehends various kind of techniques, each of one providing a human friendly way to visualize how CNNs learns. It can be used in CNNs to understand how layers and features are linked: network visualization is the XAI process of translating internal CNN features into human understandable images. The further an input image is digested through a CNN, the more complex the extracted features will be. The convolutional part of a CNN extracts low level features (like edges and colours) in shallow layers and high level features (objects, motions) in deep layers, as shown in Figure 3.3.

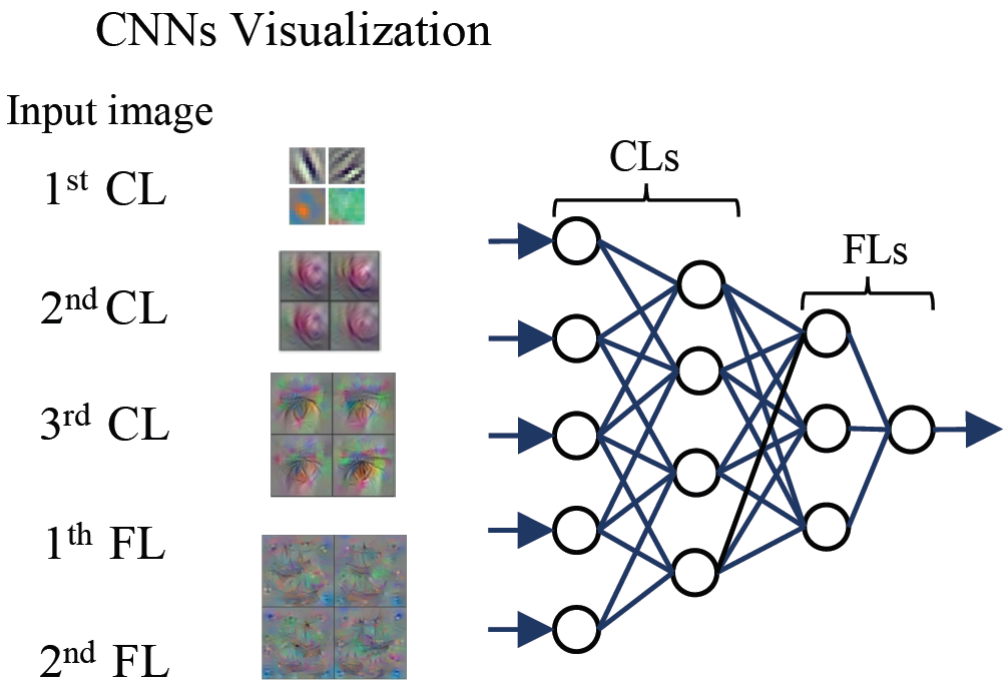


Figure 3.3: Hierarchical feature representation of a CNN. Images on the left show example of features extracted by progressively deeper layers. [31]

Multiple methods are available to inspect CNNs working principles, but reviewing all of them is beyond the scope of this section. We rather focus on a significant example to showcase XAI capabilities. We leave a deeper description of the techniques we will use in our analysis to Chapter 5.

CAM and Grad-CAM

Class Activation Mapping (CAM) [44] was the first XAI technique to introduce the concept of saliency maps: a saliency map is a heat map built over the original image that highlight what part of the image contributed most to the final prediction. CAM algorithm uses the property of the GAP layer to address which feature map influences the most the final decision.
The GAP layer projects every channel of the CNN latent representation onto a dense layer by performing an average on each channel. If a dense layer is built over the GAP layer, the resulting value of the output neurons will be a weighted average of all the GAP layer neurons: each weight states in some way the importance of a specific feature map on the final prediction. The class score for every output neuron is in fact computed as:

$$S_k = \sum_k w_k \sum_{x,y} f_k(x, y), \tag{3.1}$$

where w_k is the weight associated to the k_{th} GAP layer neuron, and $f_k(x, y)$ is the k_{th} feature map of the latent representation. The saliency map is obtained by simply weighing the feature maps $f_k(x, y)$ with their weight value w_k as

$$C(x, y) = \sum_k w_k f_k(x, y) \tag{3.2}$$

The weighted sum obtained, a linear combination of all feature maps, is then upsampled to match the input image shape. Figure 3.4 shows an example of saliency map obtained using CAM algorithm.

CAM present one big limitation: the target CNN architecture must have a GAP layer between the last dense layer and the latent representation. Grad-CAM overcomes this problem by generalizing CAM to any kind of network architecture. A weight is computed by applying GAP to the gradient of each feature map with respect to the target class score. In this way one can assign a weights for each feature map, and by upsampling and averaging them together the saliency map is computed.

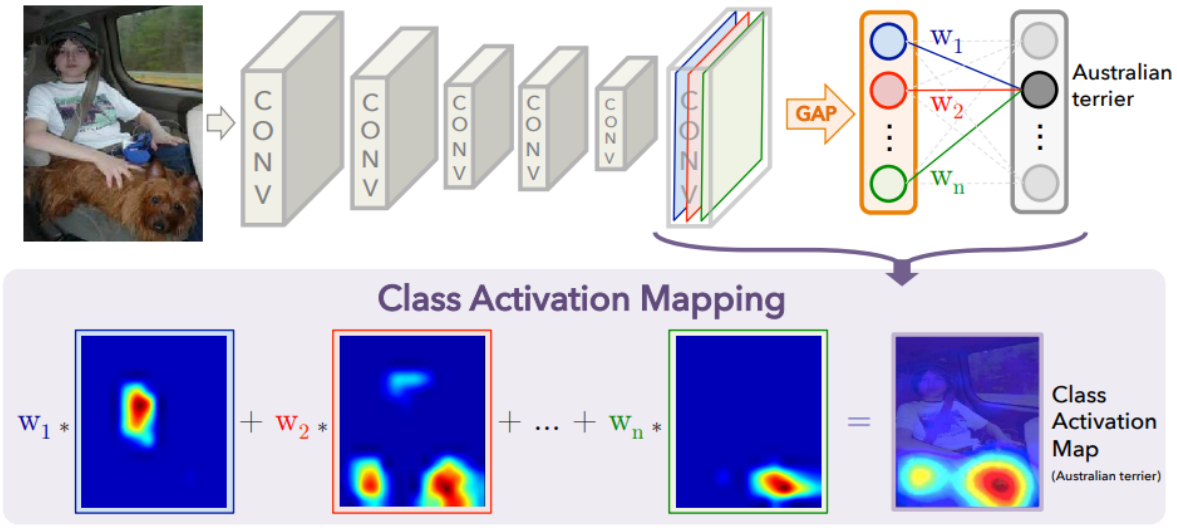


Figure 3.4: Example of how CAM works [44]: every feature map generate a saliency map, combining them together by weighting provides the Class Activation Map of the input image.

4 | Salt Segmentation

In this chapter we tackle all the theoretical work addressed for salt segmentation purposes. The segmentation task is the result of the ensemble of many operations both regarding training and dataset manipulation. In this section, we first introduce the the problem of salt segmentation, specifying why it is a challenging one. Then we will go through all details regarding different steps that have been followed to reach our solution. The last part of the chapter contains an overview on the proposed loss functions, including a mathematical intuition for each of them.

4.1. Problem Formulation

Salt recognition task can be considered a binary segmentation problem: performing binary segmentation on an image means assigning a binary label to each pixel depending on his content. In the case of salt segmentation, we can define our two labels as **background** and **salt**. To be more specific, given two labels (S, B) representing class salt and background, performing segmentation on in image I means assigning to each of his pixel $I(x, y)$ one of the two labels producing a binary mask M with the same dimension of I. Formally, we can define the mask as

$$M(x, y) = f(I(x, y)) \ , \forall (x, y) \in I \tag{4.1}$$

$$f(x, y) = \begin{cases} 0, & \text{if } I(x, y) \in B \\ 1, & \text{if } p_{x,y} \in S \end{cases} \tag{4.2}$$

where the function $f(x, y)$ associate a pixel to the corresponding label.

Segmentation task can be performed by training a CNN to produce a mask M using a labelled dataset. A segmentation dataset associates each sample image with a binary ground truth mask; in the case of salt segmentation, the sample image is a seismic image obtained with migration techniques as in Chapter 2 and the ground truth mask is a $\{0, 1\}$ mask that states the real position of salt domes in the target image.

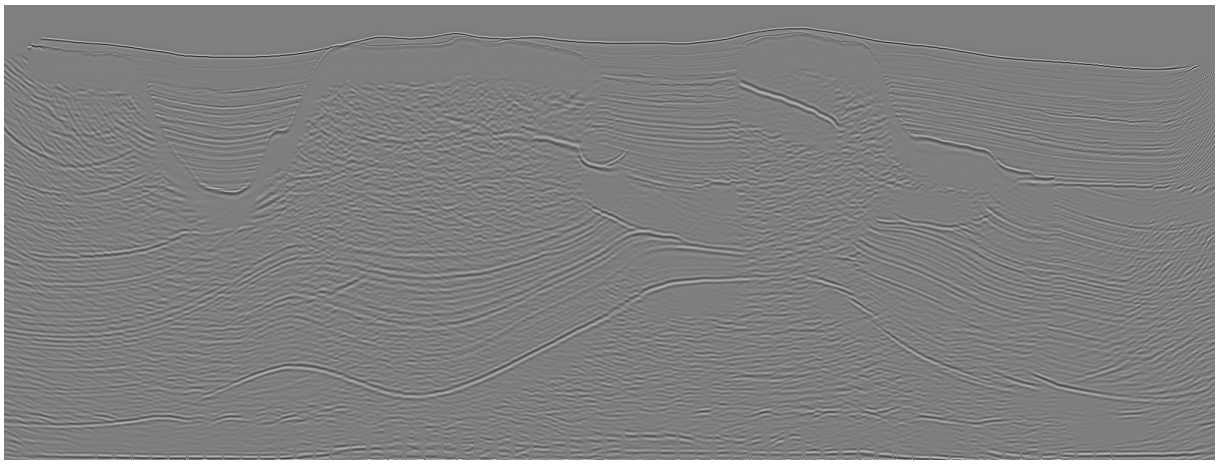


Figure 4.1: The 3D volume needs to be sliced in order to be used. Example of a 2D slice of SEAM dataset.

Figure 4.2: Corresponding mask of the 2D slice reported in Figure 4.1.

The produced CNN should be able, given an input image I, to produce a mask M as close as possible to the ground truth mask.
The dataset used for segmentation is called SEAM dataset, it is the result of a huge 3D migration of seismic images. Since CNNs can perform segmentation on 2D images the original dataset cannot be used as a whole 3D volume, for this reason training and testing operation are performed on 2D slices along the width dimension. Figures 4.1 and 4.2 show an example of 2D image and mask sliced from the original 3D volume.

4.2. Proposed Salt Segmentation Method

Our proposed method to effectively perform salt segmentation is made of different steps linked together, each taking care of a different aspect of data preprocessing, training, and evaluation. First, the 3D dataset is sliced in multiple 2D images and masks, some used for training and validation, and others for testing. The produced 2D images undergo a preprocessing step that comprise patch extraction and normalization. After preprocessing the training and validation step is performed, which provides a trained CNN ready for testing. At test time, the trained CNN is fed with testing patches and is required to produce a binary mask for each of them: once all masks have been computed, those are stitched together forming a complete 2D mask. The last step is metric evaluation, the process of comparing ground truth masks and CNN produced masks to compute performance scores for the tested CNN.

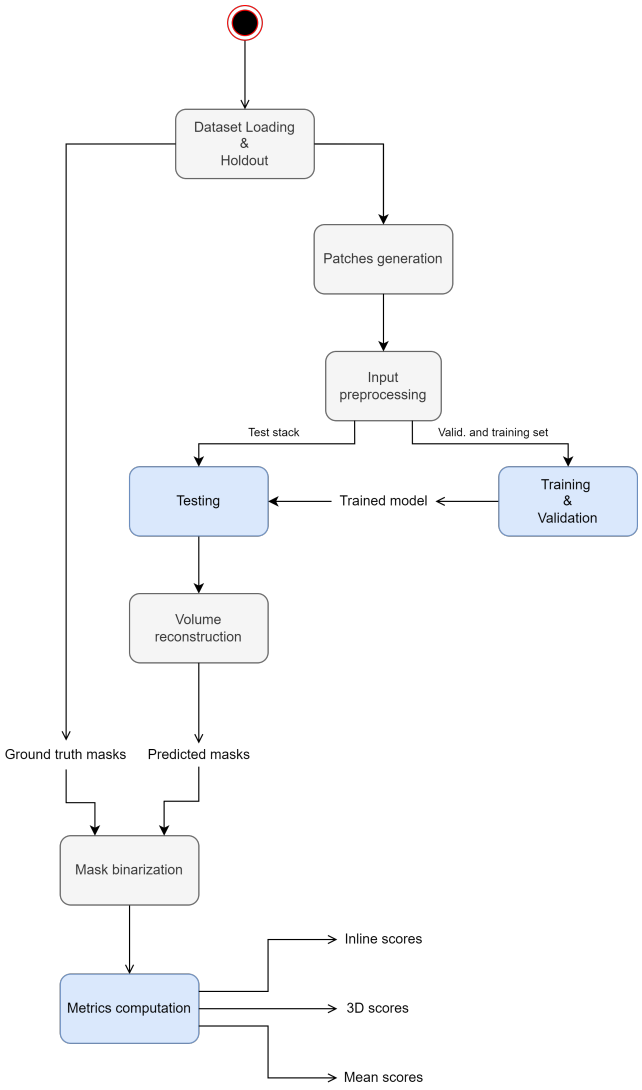


Figure 4.3: Segmentation process pipeline. Blocks coloured in blue are the core part of the segmentation process.

Figure 4.3 shows the complete segmentation pipeline, follows the detailed description of each block:

- **Dataset Loading & Holdout**: This is the very first phase of dataset preprocessing. The 3D volume is loaded into the memory as a whole and then split into several 2D slices. Follows the holdout process, which group 2D images into three categories for training, validation, and testing steps.
- **Patches generation**: Training, validation, and test images undergo patch extraction, along with training and validation ground truth masks. Patching is necessary for both memory performance (using whole 2D images would require too much RAM) and to reduce CNNs input spatial dimensionality.

- **Input preprocessing**: Input preprocessing is crucial to achieve good performances at training time. Preprocessing step include normalization and augmentations, all details about this step are reported in Section 4.2.1.
- **Training & Validation**: Training operations are performed, alternating training and validation steps multiple times. All operations related to this phase (such as training performance metrics, Learning rate (LR) and loss values) are logged for future inspection.
- **Testing**: During testing the CNN is asked to evaluate multiple unseen image patches and produce a binary mask for each of them. At testing time, only image patches are provided to trained CNN, avoiding masks that will be used for metric evaluation.
- **Volume reconstruction**: Once all mask patches are available, it is time to put them back together forming 2D images. Note that only the testing and validation masks have been split into patches. The produced 2D masks are then paired with the corresponding images an stacked together forming a 3D volume.
- **Metric Evaluation**: To compute CNN performances the previously predicted masks are compared with ground truth masks from the testing set, producing several scores (see Section 6.2). Scores are divided into 3 categories: 3D scores are computed considering masks as 3D volumes, Inline scores are computed considering masks as 2D images, Mean scores are computed by averaging all the obtained Inline scores scores together.

4.2.1. Input Pre-Processing

Input pre-processing is the first step for CNN training. The main objective of training CNNs is to teach them how to generalize the training dataset to be used in other contexts: input preprocessing helps to achieve this objective by standardizing the input distribution to fixed values.
The first step of input preprocessing is patch extraction, 2D images are in fact too big to be used directly for training for mainly one reason: too big images require huge memory space, since images are loaded in batches. To overcome this problem, a tunable patch extraction has been implemented; in this way one can decide how big a patch should be and how close patches overlap (namely patch shape and stride). The last thing that needs to be taken care of is the padding: extracting patches at the border of the image can lead to border trespassing. To cope with this problem, the exceeding parts of the image have been filled with the mean value of the valid part, leading to a full-shaped patch

(Figures 4.4 and 4.5 show an example of an extracted patch with the relative ground truth mask). The second part of the pre-processing is the z-score normalization step. Z-score normalization improves CNNs learning capabilities by standardizing the input to fixed range values: given an image patch I with mean value $E[I]$ and standard deviation σ_I, the whitened version of I is given by:

$$I_w = \frac{I - E[I]}{\sigma_I} \tag{4.3}$$

The result of the normalization step is an image with mean value equal to zero and standard deviation equal to one. Since this process is applied to all patches separately, every patch will be processed with his own mean and standard deviation.

4.2.2. Segmentation CNN

The proposed segmentation CNN has the role of performing segmentation, that is provide an output mask given an input seismic patch. The output mask is a 0, 1 image that associate salt pixels with label 1 and background pixels with label 0.

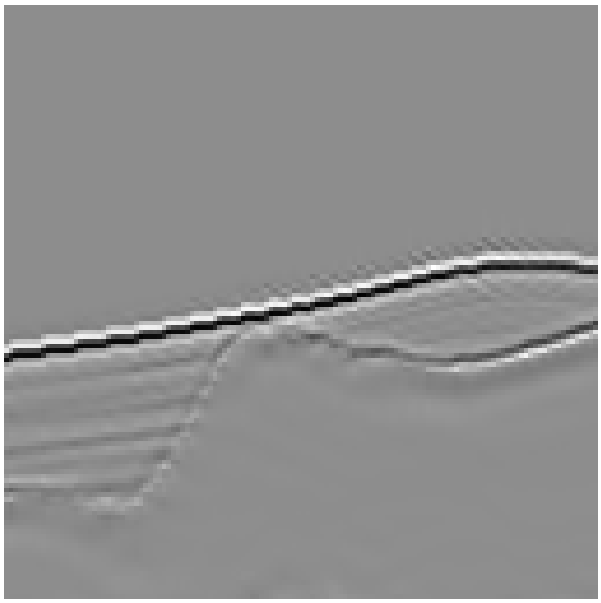


Figure 4.4: Example of extracted patch for CNN training

Figure 4.5: Ground truth mask of the image patch on the left.

The input patch, after being whitened, is fed to the CNN input layer and processed through an encoder and decoder. The chosen network backbone is Unet [34].

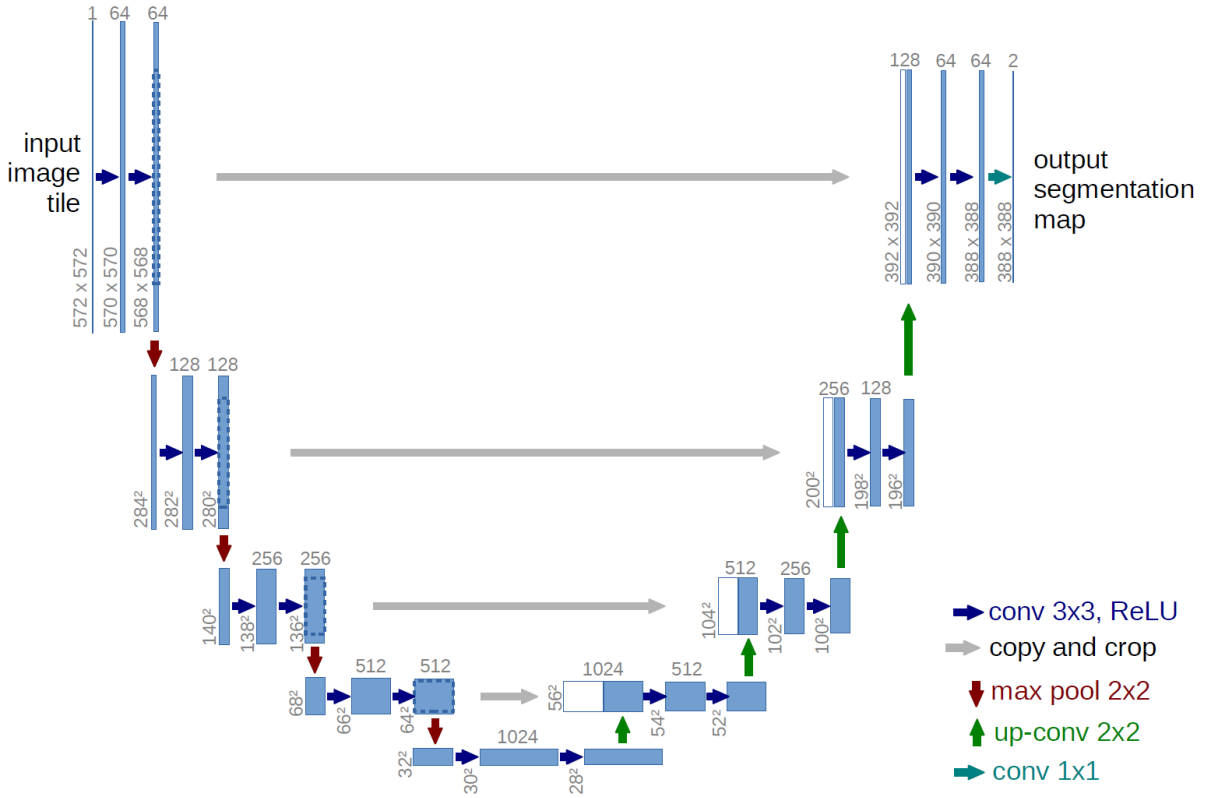


Figure 4.6: Unet network architecture, characterized by its typical "U" shape.

Unet architecture is widely known to be very effective for segmentation tasks. Unet architecture is made on an encoder and decoder part: the encoder contracts the input image, reducing its dimensionality to a small latent representation, while the decoder upsamples the compressed data to match the input image dimension, generating a mask. At the end of the decoder a sigmoid function can be present or not depending on what output values are needed. The sigmoid function squeezes all values to the range $[0, 1]$, without using it the output mask will be made of raw logits.

The contracting path (encoder) follows the standard architecture of any desired CNN, stacking convolutional layers, ReLu functions and pooling layers. The expanding path (decoder) performs up-convolutions to halve the number of feature channels at each step, concatenating also the corresponding feature map of the encoder part of the network. The concatenation between encoder and decoder at every step is the key to Unet performances. By using the same CNN architecture in both encoder and decoder each feature map have the same number of channels and image dimensionality in both paths. This enables horizontal feature map concatenation granting a huge gain in performances, the

concatenation in fact propagate fine details of the input image to the decoder, granting the output mask a detail level precision.

On the encoder and decoder architectural side, the selected model is ResNet34 [19]. Reasons that led to this decision are mainly two: ResNet34 has already been shown to be optimal for salt segmentation problems in the literature [4][37], also the same CNN architecture has been selected by our reference baseline, ensuring a faithful comparison of results.

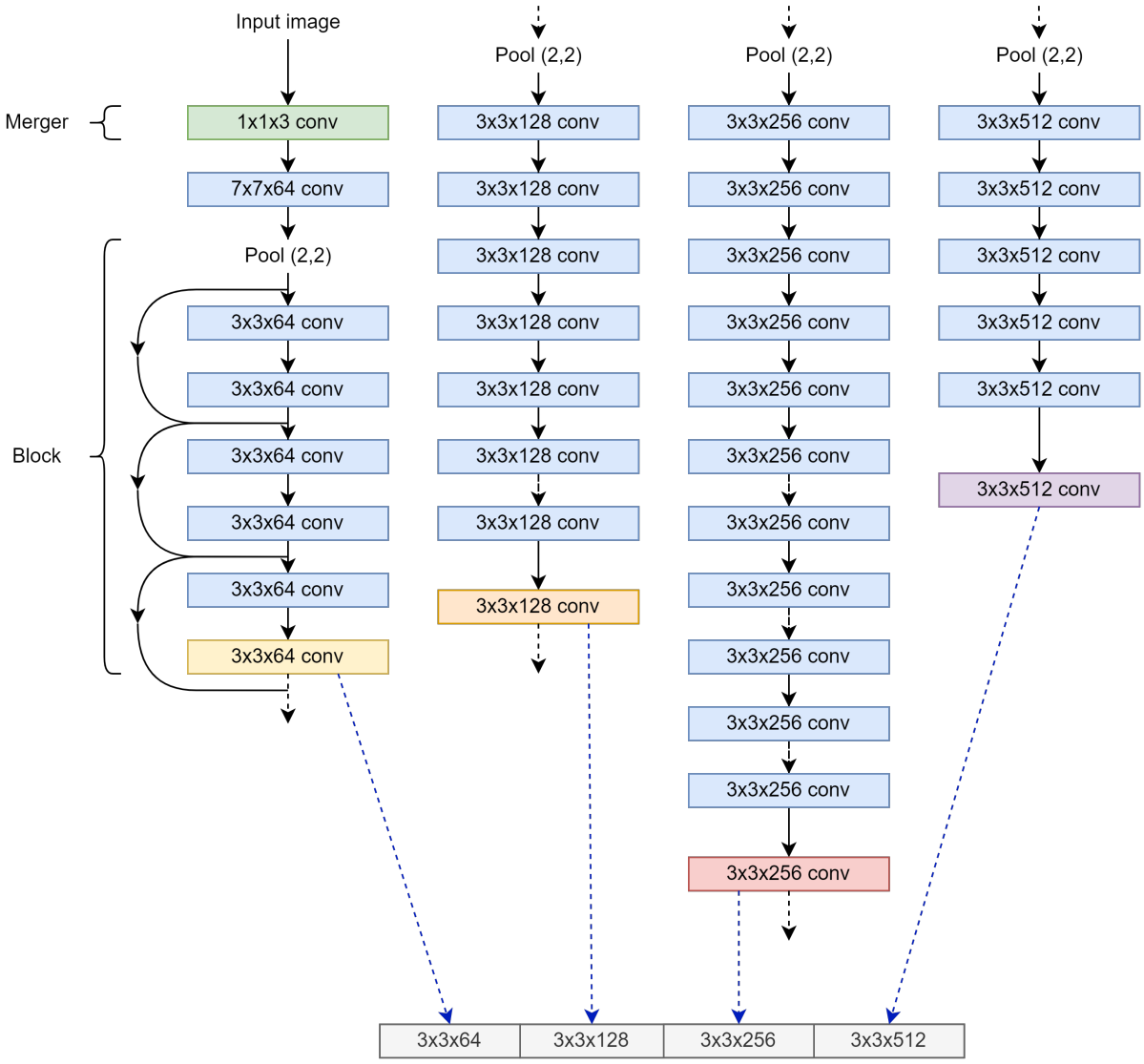


Figure 4.7: ResNet34 encoder architecture. For simplicity skip connection have been added only on the first block. Coloured layers at the end of each blocks are concatenated to create the final latent representation. The first green coloured block is the "merger".

Figure 4.7 shows the ResNet34 encoder architecture. ResNet34 is made of four convolutional blocks connected together, at the end of each block the number of channels doubles and a pooling layer halves the image dimensionality. In order to use ResNet34 with mono-channel images like ours, two options are available: replicate three times the input image to fit the three channels of ResNet architecture or add a learnable convolutional layer that expands dimensions of the input image form one to three. We decided to follow the second option by adding a convolutional layer called "merger" made of three $1x1$ convolutions. The main point of strength of ResNet34 are skip connections, that connect directly previous convolutional layer to prevent vanishing gradient during training (for clarity purposes in Figure 4.7 skip connections were added only on the first block). Thanks to skip connections ResNet architecture can be extended by concatenating blocks to form CNNs with arbitrary depth.
Every block of ResNet architecture outputs a stack of channels with different dimensionality: shallow blocks produces images with less channels and bigger image size (e.g. block 1 produces 64 images of dimension 256×256) while deep blocks outputs smaller images with many channels (e.g. block 4 produces 512 images of dimension 32×32). The final latent representation, gray in Figure 4.7, is formed by concatenating the final output of all four blocks, encapsulating information in four different processing stages.
The chosen decoder architecture is still ResNet34, following Unet principle, stacking upsampling and transposed convolution layers instead of pooling and convolutional layers.

4.2.3. Selected loss functions

The main focus of the proposed study on salt segmentation is loss testing. Many different losses are available for segmentation tasks, each of one leads the optimization process to a different minimum. Now follows an overview of chosen loss functions, with their mathematical definition and a brief intuition on how they works. When is possible a pixel-wise graph of the loss function is reported, plotting loss function values for a single pixel input y when the ground truth $\hat{y}$ is fixed to 1.

Binary Crossentropy (BCE) Loss Cross entropy loss is a widely known loss used mainly for classification purposes. Given two probability distributions $(y, \hat{y})$ and supposing that y is an approximation of $\hat{y}$, the cross-entropy is the additional information needed to represent an event drawn from the distribution $\hat{y}$ using the distribution y. For the binary case, cross-entropy is defined as [42]:

$$BCE(y, \hat{y}) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y})) \tag{4.4}$$

Where $y, \hat{y} \in 0, 1$. If the two distributions are identical no excessive information is needed to encode an event coming from one distribution using the other, hence their cross-entropy is zero.

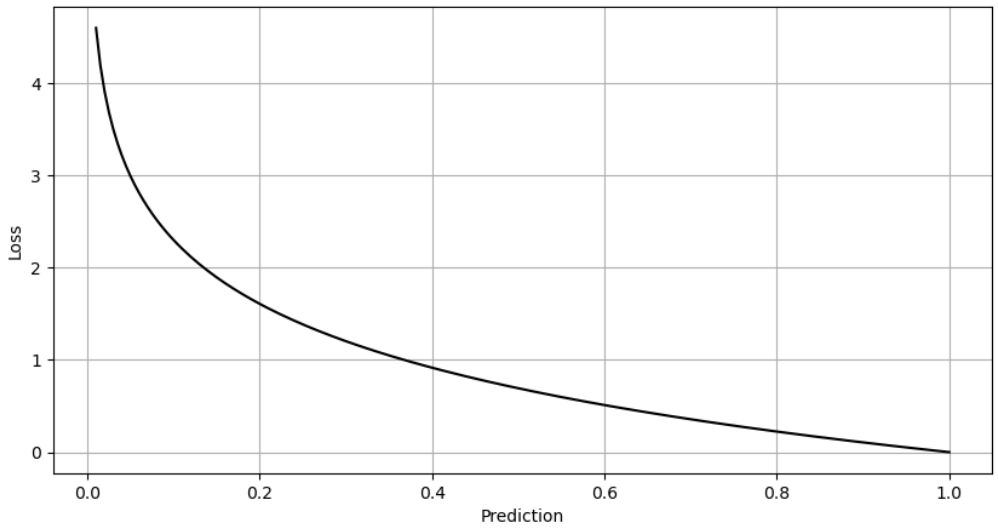


Figure 4.8: BCE loss function values for ground truth equal to 1.

Figure 4.4 shows the trend of BCE loss for a single pixel when the ground truth value is set to 1.

Contour Loss Contour loss offers a fresh point of view of image segmentation. In image segmentation the most important and difficult pixels to identify are boundaries: for salt segmentation, the main goal of the segmentation process is to find the perimeter of the salt domes. Contour loss gives greater importance to border pixels during training by weighting. Contour loss is defined as [9]:

$$C(y, \hat{y}) = -M_y(ylog(\hat{y}) + (1 - y)log(1 - \hat{y})) \tag{4.5}$$

Contour loss is the weighted version of BCE loss (BCE loss equation 4.4) using a weighting parameter M_y). M_y is obtained as the difference of an erosion $(y, S)^+$ and a dilation $(y, S)^+$

operation on the predicted mask y using a structuring element S:

$$M_y = K((y, S)^+ - (y, S)^-) + 1 \tag{4.6}$$

Equation 4.6 states that all pixels that are not set to zero by the previously defined morphological operations will be weighted by a value K, all the others will be weighted as 1 (with $K > 1$).
A "single pixel" loss graph is not available for contour loss since to be applied it requires complex masks with boundaries.

Dice Loss Dice loss originates from Sørensen–Dice coefficient[8], a score used to compute the similarity between two samples. Dice score is computed as:

$$D = \frac{2\sum y_i \hat{y}}{\sum y_i^2 + \sum \hat{y}_i^2} \tag{4.7}$$

y_i and $\hat{y}_i$ represent a prediction and ground truth pixel that can assume values of 0, 1. The denominator represent the set of all pixels, while the numerator is the sum of the correctly predicted ones. Dice coefficient (DIC) score can reach a maximum value of 1 when the two sets overlap, and can be used as a minimization function by simply setting the loss function as $1 - D$.

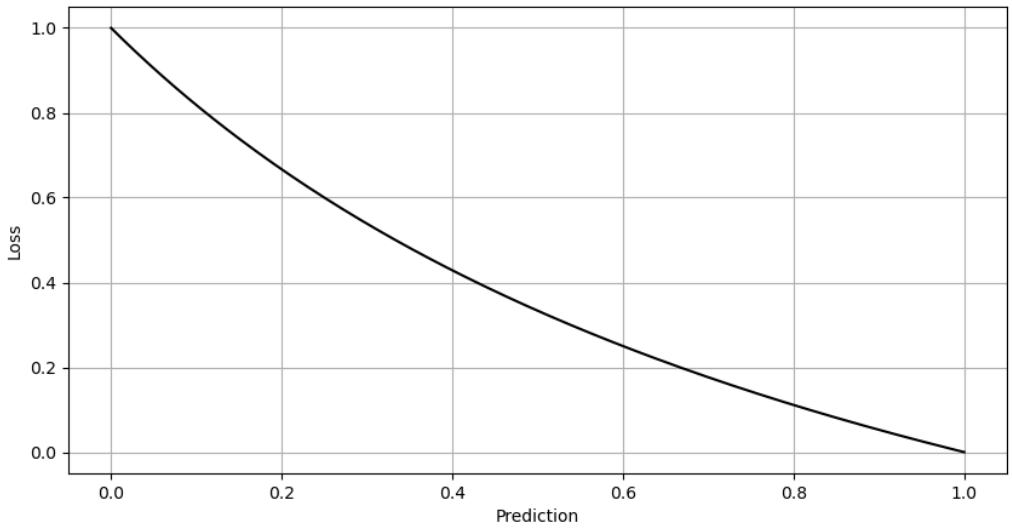


Figure 4.9: Dice loss function values for ground truth equal to 1.

Figure 4.9 shows the trend of DIC loss function for different input values. The graph has

been obtained in on pixel keeping the ground truth value equal to 1 and iterating the prediction pixel over the range $[0, 1]$. Coherently with what just said, the loss function value $1 - D$ is equal to one when the prediction and the ground truth matches.

Focal Loss Focal loss has been created to solve the problem of class imbalance in segmentation datasets: in binary segmentation problems usually the background class represents the vast majority of pixels; one other remark that must be made in binary classification is that the difficulty of classifying a background pixel is not the same for all pixels; in fact, some background pixels (usually those that are close to the object of interest) are harder to classify and should weight more on gradient calculation. Focal loss is defined as [22]:

$$F(y, \hat{y}) = -(y^{\gamma} log(\hat{y}) + (1 - y)^{\gamma} log(1 - \hat{y})) \tag{4.8}$$

Focal loss uses a gamma factor γ to further increase the probability distribution weights of BCE loss. For $\gamma = 1$ Focal loss coincides with BCE loss, for $\gamma = 0$ focal loss can be considered a non-weighted version of BCE loss.

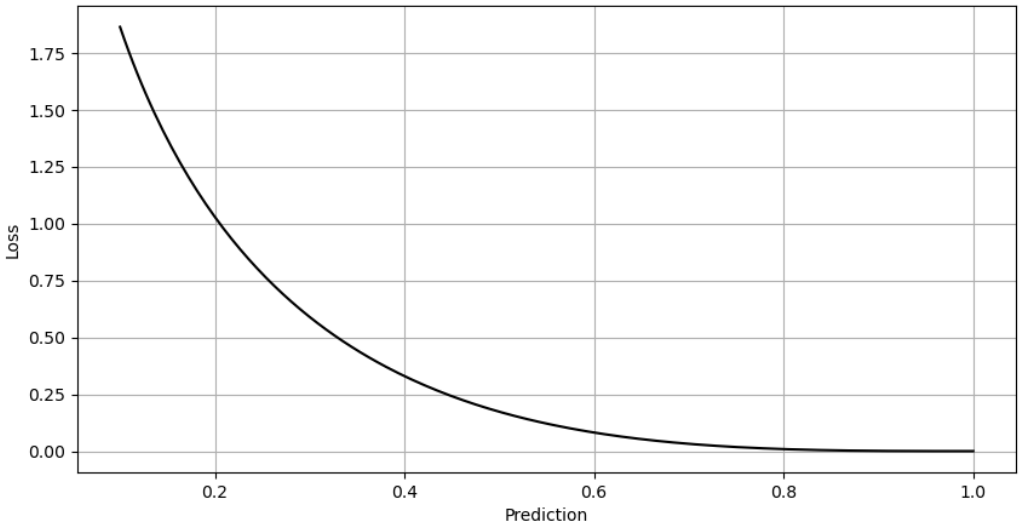


Figure 4.10: Focal loss function values for ground truth equal to 1 and $\gamma = 2$.

Figure 4.10 shows the trend of focal loss function for increasing values of y when $\hat{y} = 1$. Lin et al.[22] performed an extensive test on different values of γ, finding the best results for $\gamma = 2$.

Hausdorff Loss Hausdorff loss is yet another geometric loss function, based on the concept of Hausdorff distance. Hausdorff distance is not "commutative", given two sets $Y, \hat{Y}$, the Hausdorff distance from Y to $\hat{Y}$ is defined as [20]:

$$hd(Y, \hat{Y}) = \max_{y \in Y} \min_{\hat{y} \in \hat{Y}} ||y - \hat{y}|| \tag{4.9}$$

The bidirectional version of Hausdorff distance is defined as:

$$HD(Y, \hat{Y}) = \max(hd(Y, \hat{Y}), hd(\hat{Y}, Y)) \tag{4.10}$$

Hausdorff loss aims to minimize the bidirectional Hausdorff distance between two distributions to achieve optimality. A possible problem with Hausdorff distance is that it does not aim to minimize the pixel-wise distance of the whole image but only looks at the maximum values.
For this reason Hausdorff optimized CNNs might have overall great performances on the whole image except for few isolated locations, where segmentation error is high.

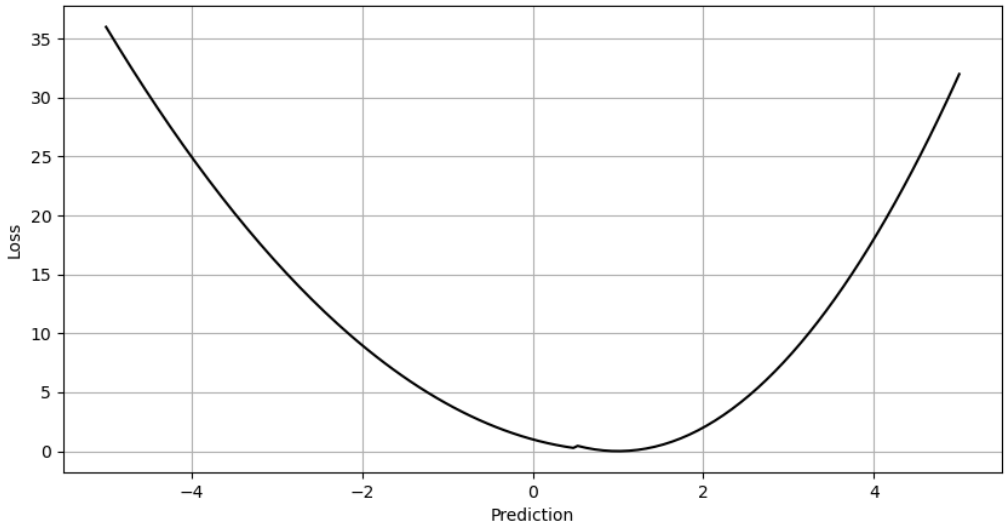


Figure 4.11: Hausdorff distance based loss function values for ground truth equal to 1. Hausdorff loss is defined over the whole real axis.

Figure 4.11 shows the trend of Hausdorff distance loss function for different prediction values. Note that Hausdorff distance is not only defined in the interval [0, 1] but spans over the whole real axis. For this reason one can either use Hausdorff loss function with

both probability input ($y \in (0, 1)$) or logits ($y \in (-\inf, +\inf)$).

Jaccard Loss Jaccard loss function [13] is a geometric based loss function, hence assume his value depending on the Jaccard distance between to sets. For discrete sets Jaccard distance coincide with Intersection over Union (IoU) (check equation 6.2), if the two sets are identical their Jaccard distance is equal to zero. For continuous sets (prediction $y \in [0, 1]$, ground truth $\hat{y} = \{0, 1\}$) Jaccard loss is defined as:

$$J_d(y, \hat{y}) = 1 - \frac{y \cdot \hat{y} + \epsilon}{(y + \hat{y} - y \cdot \hat{y}) + \epsilon} \tag{4.11}$$

The numerator part of Jaccard loss computes the intersection between the two sets, while the denominator computes the union as $cardinally - intersection$. The ϵ therm is tunable, his main role is to avoid getting infinite values when the two sets $y, \hat{y}$ are empty.

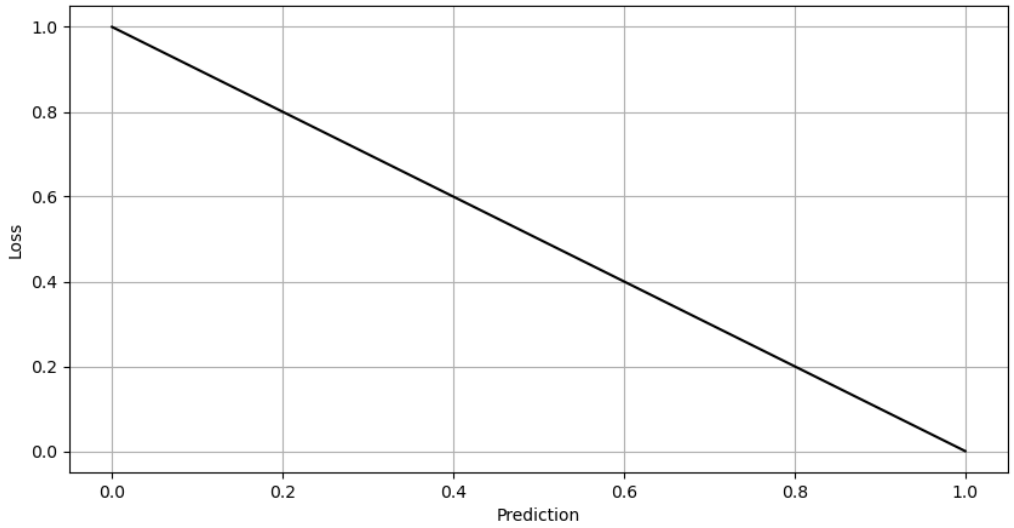


Figure 4.12: Jaccard loss function values for ground truth equal to 1.

Jaccard loss function is defined for only probability input values, Figure 4.12 shows the trend of Jaccard loss value for a single pixel value.

Kullback-Leibler (KL) - divergence Loss Given two probability distributions $(p(y), p(\hat{y}))$ their KL - divergence is defined as [38]:

$$KL(Y, \hat{Y}) = \sum_{y,\hat{y}} p(\hat{y}) \cdot log \frac{p(\hat{y})}{p(y)} \tag{4.12}$$

Intuitively, KL divergence can be considered as a "distance" between the two distributions: it measures how similar the distributions $p(y), p(\hat{y})$ are in therms of bits needed to encode one when the other is given. When the two random variables (RVs) are identically distributed, their KL divergence is equal to zero.
An important remark to make is how KL divergence and Mutual information (MI) are related. Mutual information is defined as the KL divergence between the joint and the product of two distributions:

$$MI(Y, \hat{Y}) = KL(p(y, \hat{y}), p(y)p(\hat{y})) = \sum_{y,\hat{y}} p(y, \hat{y}) \cdot log \frac{p(y, \hat{y})}{p(y)p(y, \hat{y})} \tag{4.13}$$

MI measures reduction of uncertainty of one RV once we know the other, the minimum value of MI (zero) is obtained when two RVs are independent, hence knowing one does not give any information about the other [11]. When two RVs are independent their joint and product distributions coincide, which means that their KL is zero, since there is no "distance" between the two distributions:

$$p(y, \hat{y}) \sim p(y)p(y, \hat{y}) \implies KL(p(y, \hat{y}), p(y)p(\hat{y})) = MI(Y, \hat{Y}) = 0 \tag{4.14}$$

For this reason, KL divergence can be seen as a sort of "inverse" of mutual information: if two RVs are independent then their KL divergence is high (the "distance" between two probability distributions is big) while their MI is zero (since 4.14 holds). On the other hand, if the two RVs have the same distribution their KL divergence will be zero (no distance between the two RVs distributions) while their MI will be maximum (knowing one RV gives us full information about the other).
Plotting the pixel-wise loss function of KL - divergence, as we have done with previous losses, will output only values equal to zero:

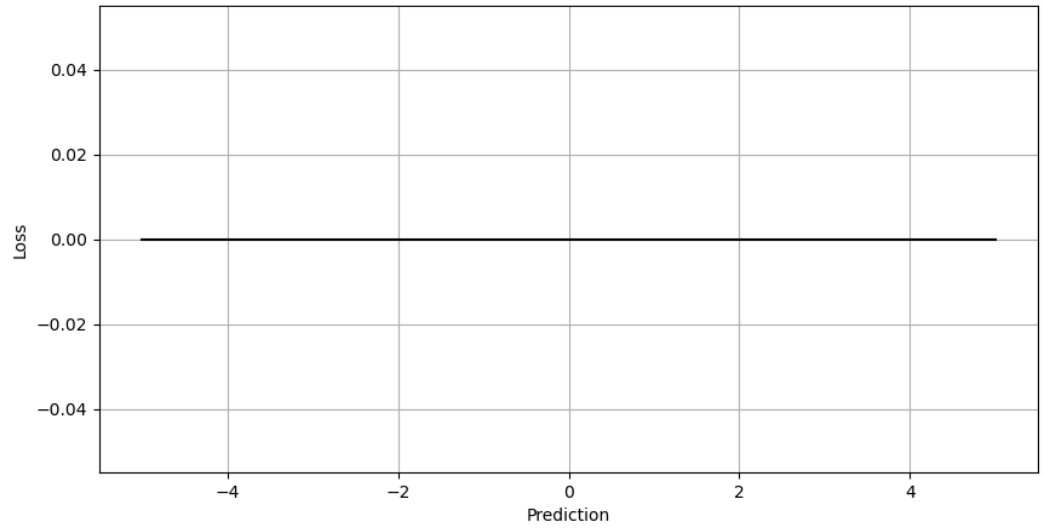


Figure 4.13: Pixel wise KL - divergence loss function values for ground truth equal to 1. Since the two pixels represent the same distribution their KL - divergence is zero for every input value.

KL - divergence loss does not take into account single pixel values but looks at the entire pixel image distribution. If the two images are made by one pixel each they will for sure have the same distribution, meaning that KL - divergence loss will output zero for all values of y.

Lovasz Loss Lovasz loss have been created to optimize the already present Jaccard loss to increase performances and speed up training [6]. As we said in Jaccard paragraph, Jaccard loss is basically a continuous version of IoU score, used as a loss function in CNNs. Lovasz loss exploits Jaccard loss submodularity property to optimize a surrogate version of it, making the new loss function (the Lovasz loss) convex with polynomial convergence time.

Mean square error (MSE) Loss MSE loss (also called L2 loss) is simply defined as the Mean Squared Error between ground truth and predicted pixels. Given two images $Y, \hat{Y}$ the MSE loss between the two is defined as:

$$MSE(Y, \hat{Y}) = \frac{1}{n} \sum_{i,j} (Y_{i,j}^2 - \hat{Y}_{i,j}^2) \tag{4.15}$$

MSE loss is fairly intuitive and widely used in many applications for regression problems.

An example of MSE loss function trend for probability input y is reported in Figure 4.14:

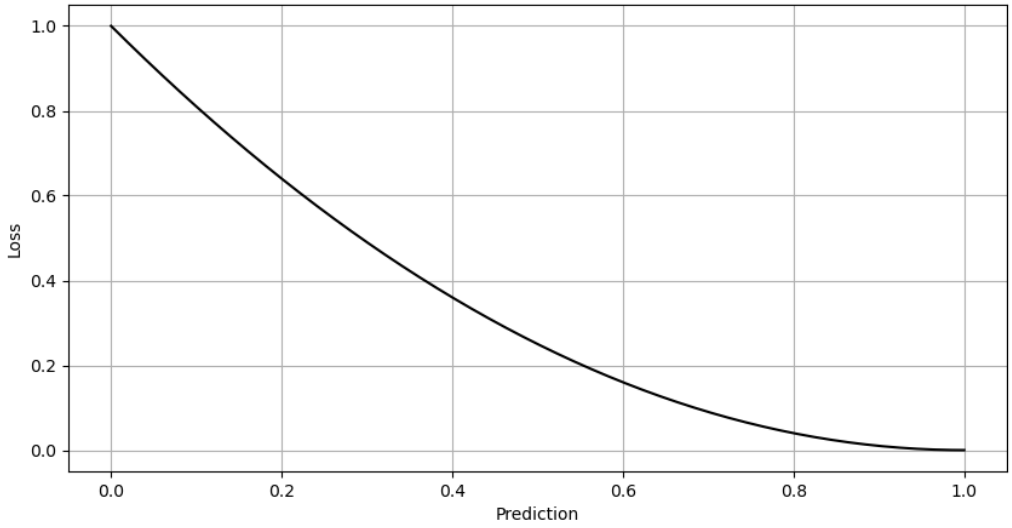


Figure 4.14: MSE loss function values for ground truth equal to 1.

Wing Loss Wing loss has been created to overcome some optimization problem related to the standard L1 and L2 losses. Wing loss is defined as [16]:

$$Wing(y, \hat{y}) = \begin{cases} \omega ln(1 + |\frac{y-\hat{y}}{\epsilon}|), & \text{if } |y - \hat{y}| < \omega \\ |y - \hat{y}| - C, & \text{if } |y - \hat{y}| \geq \omega \end{cases} \tag{4.16}$$

Where ω is a set of points $(-\omega, \omega)$ where we want our function to behave logarithmically and C is a constant factor used to smooth the connection between the linear and nonlinear parts. With respect to standard L1 and L2 loss, wing loss function can assume two different behaviour depending on the error magnitude $|y-\hat{y}|$: if the error is small $|y-\hat{y}| < \epsilon$ the wing loss behaves like a logarithm (with a gradient equal to $\frac{1}{|y-\hat{y}|}$) outputting large gradient values for lower errors. For larger errors, on the other hand, the gradient is constant. This adaptive behavior increases the weight of small errors on the gradient: L1 and L2 losses work well with large errors, but struggle to fine-tune for smaller ones; Wing loss can enforce decent gradient magnitude in both cases.

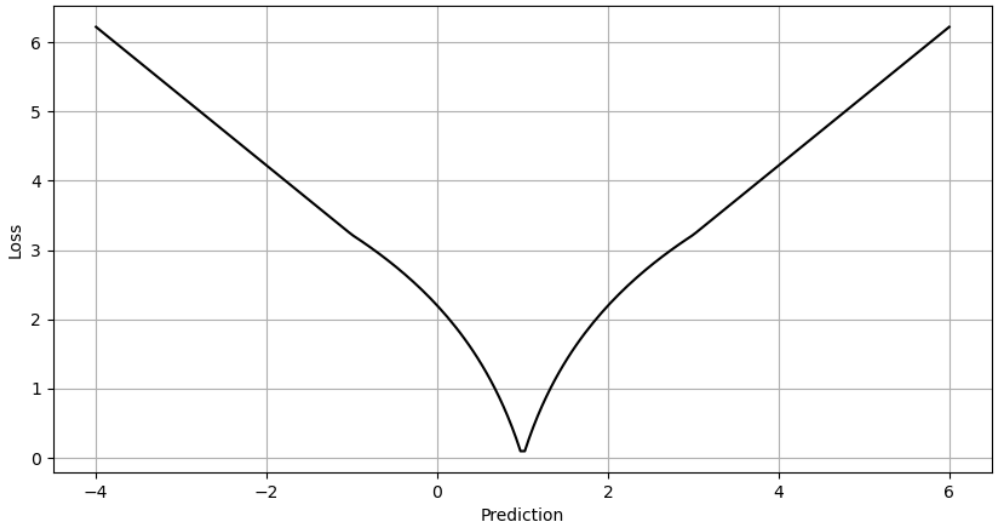


Figure 4.15: Wing loss function values for ground truth equal to 1.

Figure 4.15 clearly shows the different wing loss behaviour for small and large error values. In the reported graph ω assumes a value of 2: note how for larger error values wing loss switches behaviour, shifting from a logarithmic-like function to a linear function.

5 | CNN Interpretation

XAI comprehend a set of techniques that can be used to shed a light into the black box of a CNN. Every technique has its pros and cons, and the results it produces can sometimes be misleading (or hard to interpret) for the human eye. This chapter dives into the details of two different techniques, AM and NI: each one has a different scope and tries to inspect how a CNN works under a different perspectives.

5.1. Activation Maximization

Activation maximization technique is based on a very simple yet powerful idea: to understand the meaning of a specific neuron of a CNN, look for the input image that maximizes the activation of that neuron; the resulting image could be a good representation of what that neuron is doing. Since CNN are differentiable with respect to their input, it is possible to find a target input image by tuning it iteratively [14].

Given a neuron $\mathbf{h}_{ij}$ of a neural network with parameters θ, the input image $\mathbf{x}$ we are looking for will be[14]:

$$\arg\max_{x} h_i j(\theta, x) \tag{5.1}$$

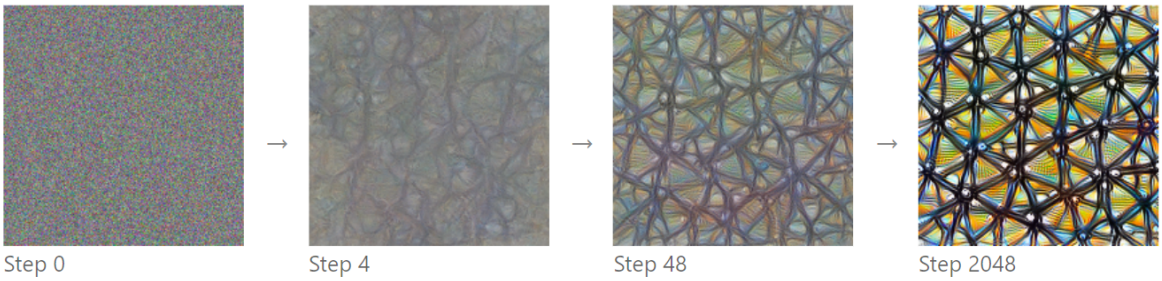


Figure 5.1: Figure shows how the AM target image is formed. At step zero the generated image is random noise, by progressing in the optimization procedure clearer images are formed.

To perform the optimization, perform a training operation on the input image. Choose a number of epochs and then iterate.

- Initialize an image $\mathbf{x}_0$ as random noise for all pixel values
- Compute the gradient on the input image using backpropagation as

$$\frac{\partial h_{ij}}{\partial x} \tag{5.2}$$

- Update each pixel of the image to maximize neuron activation by moving on the previously computed gradient: the update iteration is known as gradient ascent of step η for each step k

$$x_k \leftarrow x_{k-1} + \eta \frac{\partial h_{ij}}{\partial x_{k-1}} \tag{5.3}$$

Iterate this process for all the previously defined epochs. Figure 5.1 shows how, by performing more and more optimization iterations, a synthetic image evolves form pure noise to a clear image.
A shortcoming of the standard AM method is that in this way it is possible to inspect only one neuron at the time.
Looking at only one neuron activation might be enough when inspecting a specific class neuron, but will not be able to capture how intermediate layers work. CNNs layers are made by a large number of neurons that interact with each other in a complex manner: it is impossible to inspect how an entire layer activates by looking at each of his neuron because they usually are way too much.
A solution could be tweaking our objective function to maximize an entire layer or channel of the CNN, instead of a neuron only. To inspect how a classification network interprets a specific class, one can set as objective function the specific output neuron that represents the class of interest and apply AM to that neuron. To do that, the neuron value can be taken before or after the softmax.
It has been proved that [39] that optimizing post-softmax probabilities rather than logits might lead to worse results: this happens because the optimization, instead of maximizing the output probability for that specific class, finds easier to push down the probability of all other classes.

Transformation Robustness

The optimization process of AM, even if it looks easy in theory, can be quite hard.
The objective function to be optimized has many local minimum, some shallower than others. Without introducing any constrictions or parametrizations, the optimal input

image that activates a neuron might be covered with random noise.
This happens because, for some still not sure reasons [28], the target neuron strongly respond to this high-frequency patterns that covers the image. To cope with this problem, the first thing that comes to mind could be introducing a penalization over high frequencies during the optimization process.
This method surely reduce the noise that affect the image, but might also dump high frequencies that might be valid and useful information for the optimization process. By adding some sort of regularization, the objective function to be optimized becomes:

$$\arg\max_x(h_i j(\theta, x) - \lambda(x)), \tag{5.4}$$

where $\lambda(x)$ is the regularization therm.
A number of different $\lambda()$ functions have been proposed in literature, with their pros and cons. Some examples are l_2 decay [39] and gaussian blur [43].

Regularized Activation Maximization
Transformation robustness substantially increases the quality of target images without penalizing high frequencies components [29].
The intuition behind transformation robustness is that, since the CNN is usually trained to identify targets at different scales and positions, if an image maximizes activation of a neuron also a slightly transformed version of it does. This statement usually does not hold for the high frequency noise that might affect the image, that is not robust to transformations: by adding transformations to the noisy image, the new transformed noise will not activate the neuron anymore but will probably do the opposite.

In Figure 5.2 the first row has been obtained by performing optimization with no transformation robustness. Note how high frequency noise dominates the image.
The second row shows images obtained by imposing jitter and scale transformation robustness on the target image.

5.2. Network Inversion

NI shares some similarities with AM method, the objective though is totally different. AM mainly focus on a specific neuron or channel, sometimes lacking a general prospective. Even though AM can be very effective on logits class neurons, it is not that useful when looking at internal convolutional layers, made of hundreds of channels stacked together. NI can be used to effectively inspect a layer as a whole, both convolutional or fully connected [24].

Figure 5.2: Images generated without imposing transformation robustness (top) compared with same images generated imposing transformation robustness. Note how high frequency noise disappear when transformations are introduced.

To do so a sample image x is fed to the CNN, and his representation is computed up to the layer A to be inspected obtaining the so called "feature map" $A(x)$. The algorithm goal is to find an input image x^* whose feature maps $A(x^*)$ is close to $A(x)$, adding also a regularization therm such that x^* resembles in some ways the dataset sample x.
The optimal image x^* is obtained iteratively by optimizing an input image x such that [14, 39]:

$$x^* = \arg\min_x(||A(x) - A(x^*)||^2 - \lambda(x)) \tag{5.5}$$

The find x^* again perform a training operation over a starting image x_0:

- Initialize the input image x_0 as noise.
- Compute the feature maps $A(x_0)$ and $A(x)$ of respectively the image to be optimized and the target image.
- Compute the gradient $\frac{L(A(x_0),A(x))}{\partial x}$ and update x_0 by gradient descend.

The algorithm tweak iteratively the input noise x_0 until the minimum distance between feature maps $A(x_0)$ and $A(x)$ is reached.
The main goal of NI is to find out what part of input image information is lost and what is kept at a specific network layer. CNNs learn to generalize by encoding an input image, transforming it and removing useless details the more the image is processed: first layers

elaborate the input and extract low level features, deeper layers combine already processed inputs together to extract features of higher level. With NI is possible to inspect CNN layers one by one, and find out what features are preserved and what have been discarded the more we go deep into the network; the similarity with the input image is imposed by the regularization therm $\lambda(x)$.

As shown in the Figure 5.3, the input image gets more and more processed the deeper we

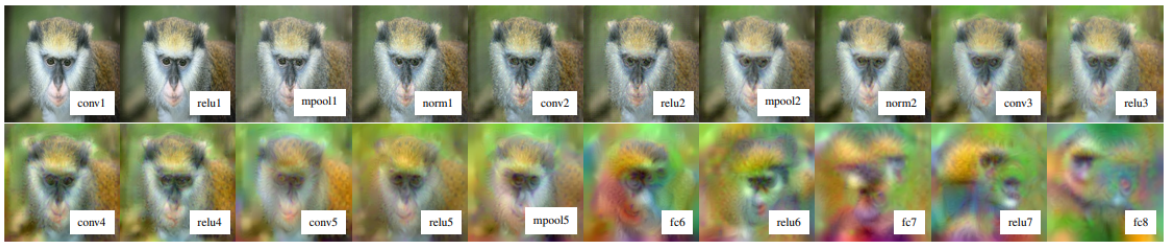


Figure 5.3: Reconstruction of an image using NI. From left to right, NI is computed on deeper layers. Shallow layer representations retain several details of the natural input image, last convolutional and dense layers are much harder to interpret.

inspect into the network. NI applied on the first convolutional layers result in very similar images with respect to the original; deeper layers tends to discard fine details of the input image, leaving only generic, high-level generic features. In the last fully connected layer the synthesized image is almost impossible to associate with the input image, representing only a generic idea of a baboon. One last aspect to consider network invariance: CNNs are usually trained to be robust with respect to rotations, translations, scales and many more transformations. Network invariance is visible in NI generated images, in deeper layers synthesized images might present replicas of the input with different scales or rotations at different positions.

6 | Results

This chapter reports all the details of the experimental validation we carried out. First, we describe the used datasets. Then, we explain how the selected evaluation metrics work. It follows the the experimental setup used to validate the proposed segmentation solution as well as the used XAI techniques. Finally, we report numerical results and visual examples.

6.1. Datasets

In this thesis two datasets have been used in total. One, the SEAM dataset [15], has first been used to train CNNs for segmentation purposes. SEAM dataset has then been processed to perform classification on segmentation networks for the purpose of CNN explainability.
The second dataset used is called LANDMASS [2], a classification dataset that contains images belonging to four different types of seismic artifacts.

SEAM Dataset
The first dataset we describe is the used segmentation dataset called SEAM [15].
A peculiarity of this dataset is that it is not flooded; this means that the velocity model used to obtain all dataset samples does not take into account salt bodies.
For this reason only the top part of salt bodies will be easy to visualize. This happens because the velocity model is correct up to the beginning of salt blocks, so clearly revealing the discontinuity between standard reflectors and salt.
The bottom perimeter of the salt will be much harder to exploit: since the velocity model is incorrect from the top of the salt down to the end of the image, all the information that comes to us is somewhat distorted.
This dataset is made of two parts: migrated images and associated masks. Migrated images and masks are organized in huge 3D volumes of 1751 Inlines (ILs), 2001 Xlines, 751 Depth. The IL direction is usually parallel to the direction of the data acquisition process. The crossline is perpendicular to the IL direction. Depth (or time) in the end is

the axis that goes downward into the ground. To obtain 2D images from the 3D volume, the dataset has been processed as IL "slices". The first thing to do is to separate the dataset in training, validation and testing set along the IL direction.

The first 735 ILs have been used for training and validation, the rest are for testing. To

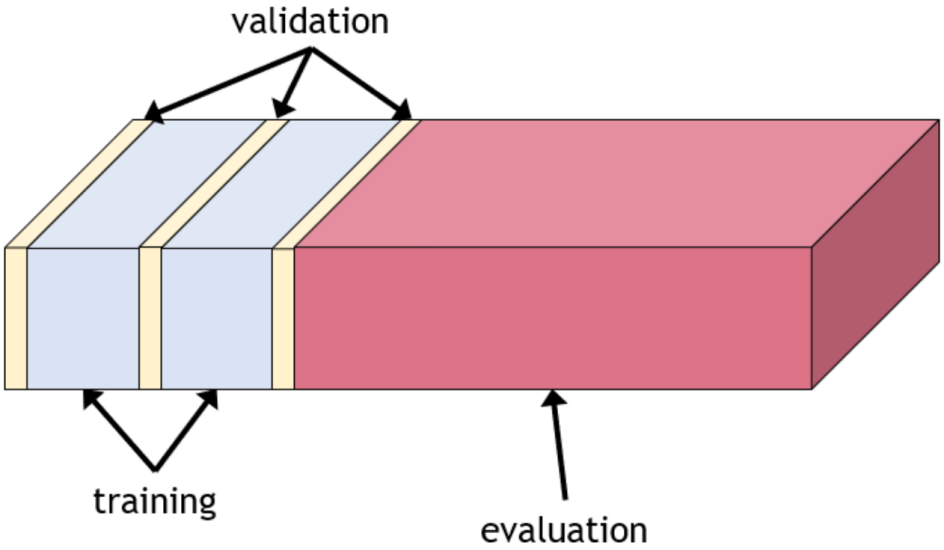


Figure 6.1: Th above figure shows how the 3D block of SEAM dataset has been divided for training, validation and test set along the IL axis

be fair in the separation of training and validation, the validation set has been obtained by picking ILs fairly at the beginning, middle and end of the first 735 ILs as shown in Figure 6.1. Slicing the SEAM dataset along ILs results in the creation of multiple images with a dimension of 2001×751, clearly too much to be fed in a segmentation CNN. For this reason, smaller patches have been extracted from ILs, used for training, validation, and testing. For details, see Section 6.3.

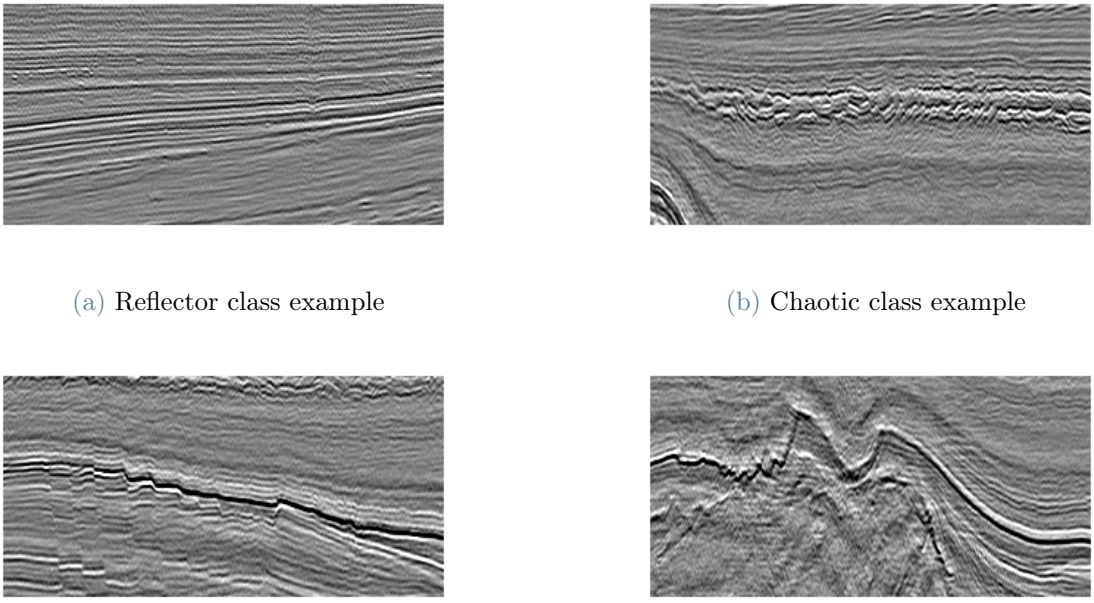
LANDMASS Dataset

LANDMASS (LArge North-Sea Dataset of Migrated Aggregated Seismic Structures) dataset [2] is a classification dataset made of 4000 images of size 150×300 with values in the range range $[0, 1]$.

This dataset contains 4 classes, each one of 1000 images, labelled as follows:

- Reflectors: Smooth horizontal formations (Example in Figure 6.2a).
- Chaotic Horizons: Horizons with noisy, chaotic-like appearance. All the artifacts that the interpreter could not classify with confidence belong here. For this reason, chaotic class might be the harder to classify since all sorts of different horizons belong to this class. An example of class Chaotic is reported in Figure 6.2b

- Faults: Images with one or more faults in them. As shown in figure 6.2c Faults have the shape of vertical lines, they can hard to pick up for non expert eyes.
- Salt Domes: Images that include parts of salt domes in them. Since the dataset is flooded, the salt shape should be much more easy to recognize compared to SEAM dataset (example in Figure 6.2d).



(a) Reflector class example

(b) Chaotic class example

(c) Fault class example

(d) Salt class example

Figure 6.2: Example of the four classes of LANDMASS dataset

6.2. Metrics

CNN salt recognition is a segmentation problem made of two classes: salt domes and background.
An ideal way to evaluate the performances of a model would be to consult an expert who can tell us which is, among all trained CNNs, the best network for salt segmentation purposes. Obviously this is infeasible due to both the large amount of data to be examined and the large amount of CNNs that have been trained.
Metrics can greatly help coping with this problem: even if none of them is perfect, each one can provide a different prospective on how effective the trained CNN is at performing segmentation. All the proposed metrics refer to a binary classification mask detection, where the two classes are divided in salt (class 1) and background (class 0). The available data is obviously made of two sets: the ground truth S_g and the CNN prediction S_p.

Lastly, some few metrics for classification problems will be reported: they have been used to train classification networks for the purpose of CNN inspection.

6.2.1. Confusion metrics

This paragraph will go through basic metrics that will than be used to compute more complex ones in next sections.
The four building blocks are:

- TP - *True positives*: Salt pixels in Sp correctly identified by the CNN
- TN - *True negatives*: Background pixels in S_p correctly identified by the CNN
- FP - *False positives*: Background pixels in S_g that have been classified as salt in S_p
- FN - *False positives*: Salt pixels in S_g that have been classified as background in S_p

6.2.2. Segmentation metrics

Segmentation metrics are many, each one tries to explain performances of our trained CNN in a different way. They take as input S_g and S_p and compute a score through confusion metrics.

DIC DIC[12] is an index that reveals how much two sets overlaps. it can range from 0 (no overlapping) to 1 (complete overlapping). It is computed as:

$$DIC = \frac{2TP}{2TP + FP + FN} = \frac{2\left|S_g^1 \cap S_p^1\right|}{\left|S_g^1\right| + \left|S_p^1\right|} \tag{6.1}$$

In fact, the second equality of the above equation computes how much intersection between S_g and S_p of class 1 is present in their sum.

Jaccard index (JAC) The Jaccard index [21] computes IoU on the two sets.

$$JAC = \frac{TP}{TP + FP + FN} = \frac{\left|S_g^1 \cap S_p^1\right|}{\left|S_g^1 \cup S_p^1\right|} \tag{6.2}$$

JAC and DIC fundamentally computes the same quantity, in fact they are related through the following formula:

$$JAC = \frac{DIC}{2 - DIC} \tag{6.3}$$

Hence DIC, JAC and IoU are strongly related to each other.

True positive rate (TPR) and True negative rate (TNR) TPR is also called Sensitivity or Recall and computes what portion of positive samples in the ground truth has been identified as same during prediction. It is computed as:

$$TPR = \frac{TP}{TP + FN} \tag{6.4}$$

TNR on the other hand measures the same aspect but on the negative class. It is also called Specificity, and is computed as:

$$TNR = \frac{TN}{TN + FP} \tag{6.5}$$

Global Consistency error (GCE) GCE [25] considers one of the two segmentations as a refinement of the other. To reach a high GCE score, the two segmentation masks do not need to coincide: for each pixel p extract two subsets of both masks $R(Sg,p), R(Sp,p)$ whose elements belong to a neighborhood of p. If a set is a proper subset of the other then one set can be seen as a refinement of the other, granting GCE equal to zero. On the other hand, if the two set overlapping is inconsistent GCE will be close to one. This process has to be repeated for each pixel p, taking a final average[40]:

$$GCE = \frac{1}{n} min(\sum_{i}^{n} E(S_g, S_p, p), \sum_{i}^{n} E(S_p, S_g, p)) \tag{6.6}$$

Where $E(S_g, S_p, p)$ is the error computed considering $R(S_g, p)$ as a subset of $R(S_p, p)$. This value is different from $E(S_p, S_g, p)$ where instead the subset relationship between the two subsets is inverted. $E(S_g, S_p, p)$ is computed as:

$$E(S_p, S_g, p) = \frac{|R(S_g, p)/R(S_p, p)|}{|R(S_g, p)|} \tag{6.7}$$

Volumetric similarity (VS) VS Computes how similar two volumes of segmentations are. Is defined as $1 - VD$ were VD measures the volumetric distances between masks[32]. VS can be computed as:

$$VS = 1 - VD = 1 - \frac{|FN - FP|}{2TP + FP + FN} \tag{6.8}$$

Perimetric similarity (PS) PS is simply defined as VS computed only on the perimeter of salt bodies.

MI MI measures how much information of one mask is retained in the other [5]. MI is measured in terms of Entropy $E(S)$, an estimate on how "unexpected" mask pixel values are, and joint mutual information $H(S_p, S_g)$ that measures how much one mask is predictable knowing the other. In equations:

$$H(S) = -\sum_i p(S^i) log[p(S^i)] \tag{6.9}$$

$$H(S_g, S_p) = -\sum_i p(S_t^i, S_p^i) log[p(S_t^i, S_p^i)] \tag{6.10}$$

$$I(S_g, S_p) = H(S_g) + H(S_p) - H(S_g, S_p) \tag{6.11}$$

Where $p(S^i)$ and $p(S_t^i, S_p^i)$ can be computed in therms of confusion metrics. $I(S_g, S_p)$ is an estimate of how much uncertainty is lost over S_g if S_p known and vice versa.

Area under Curve (AUC) AUC means Area Under the ROC Curve. The ROC curve is usually defined in classification as the curve plot of TPR vs. TNR once a threshold is set. In case of Segmentation, is defined as [30]:

$$AUC = 1 - \frac{1}{2}\left(\frac{FP}{FP + TN} + \frac{FN}{FN + TP}\right) \tag{6.12}$$

6.2.3. Classification metrics

Classification metrics still use the same confusion metrics defined above, with the difference that they are now computed on binary labels instead of pixels [23].

Precision In binary classification tasks, the Precision metric determines how well CNN identify True positives over all predicted positives. It is a good metric to trust when it is vital to avoid False positives, and is computed as:

$$Precision = \frac{TP}{TP + FP} \tag{6.13}$$

Recall Recall follows the same principles of Precision: it evaluate how well the CNN identify true positives over all the real positives. A high recall score ensures resistance to

False negatives:

$$Recall = \frac{TP}{TP + FN} \tag{6.14}$$

F1 Score F1 Score is computed as a function of Precision and Recall. It can be seen as as sort of balance between them, very usefull if the dataset is unbalanced.

$$F1 = 2 * \frac{Precision\, Recall}{Precision + Recall} = \frac{TP}{2TP + FN + FP} \tag{6.15}$$

By substituting Precision and Recall with confusion metrics, F1 Score formula ends up being the same as DIC score for segmentation. They are indeed the same score.

Accuracy Accuracy metric computes how often the prediction is equal to the ground truth at test time. It is defined as the fraction between the number of correct predictions and the number of total predictions. In formulas:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{6.16}$$

Accuracy (rather than F1 score) does not take into account how data are distributed and therefore is not reliable for unbalanced classification problems.

6.3. Experimental setup: Segmentation

In this section we go through all the details about how CNNs have been trained for segmentation purpose.
All experiments have been coded in Python using Pytorch library. Pytorch provides great freedom for parameter selection and allows the user to customize training and validation steps as desired. All trainigs have been performed over GPU to speed up the process, only one GPU has been used at the time.
The process of CNN training can be divided into four parts: Training and validation, testing, metrics evaluation. The first thing to do is splitting the dataset into 3 parts for training, validation and testing. As stated in dataset section, the first 735 ILs have been used for training and validation, the rest for testing. Furthermore, 20% of the first ILs stack have been used for validation, bringing us to the following partition :

- 588 ILs for Training - 34%
- 147 ILs for Validation - 8%
- 1016 ILs for Testing - 58%

Since ILs slices are too large to be fed directly into CNNs, smaller patches have been extracted from every IL. The selected patch dimension is 512×512, with a stride of 512×512. Note that, since patch shape and stride have the same value, there is no overlapping between adjacent patches. This decision, applied to both stack images and masks, has been made to avoid excessive GPU memory usage during training and testing, since only one GPU has been used at the time. If a patch exceed a border of the IL a padding is applied, filling missing values with the mean value of the selected IL: the relative mask is filled with zero values. After extracting patches augmentations have been applied to both masks and patches. Among all possible augmentations only Horizontal flip and label smoothing have been applied to the newly generated patch dataset, with a probability of 0.5.
This choice have been made for two reasons:

- Training set is already sufficiently big to ensure quality CNN training, flipping patches with a probability of 0.5 provides even more data.
- Adding other augmentations could be dangerous since they could denature the meaning of some patches. Horizons rotation, brightness, and contrast are all fundamental clues for salt recognition. Applying random perturbation to those features could hence invalidate CNN decision.

All segmentation networks trained have some common hyperparameters: training process have been performed over 100 epochs with a LR of 0.01, using Adam as optimizer and a batch size of 8. What differentiate all CNNs is the different loss function (for details see Chapter 4) and the last activation layer, that have to be chosen accordingly: if the loss required it CNN have been trained with a final Sigmoid layer, otherwise raw logits are directly passed to the loss function (Figure 6.3).

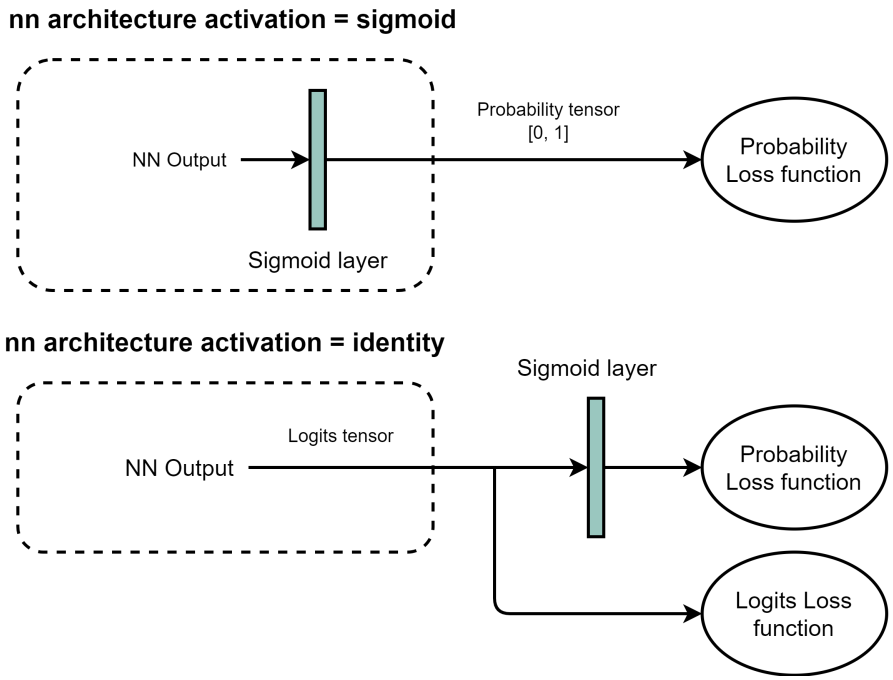


Figure 6.3: If CNN architecture do not include a final sigmoid layer but loss function requires it, sigmoid function is applied externally before passing output values to the loss function. Note that this process is not needed if CNN architecture already include a sigmoid activation, but in this case it is not possible to use losses that requires raw logits as input.

6.4. Experimental setup: Classification

CNN Inversion techniques discussed in Chapter 5 investigate on how a CNN interprets different classes: to apply AM for example, we need some specific target to optimize that embodies how our CNN sees one class or another. It is impossible to apply those techniques directly to segmentation networks since it maps 2D inputs in 2D outputs, there is no specific objective that resembles one class or another. Even if we can't inspect the whole segmentation network, we can still try to understand how networks manipulate the input by looking at its latent representation, extracting meaningful features on seismic images for segmentation purposes. There are two possible ways to inspect segmentation models with CNN inspection techniques:

- Remove the decoder from a pre-trained segmentation CNN, attach a classification head, and train the new CNN by lowering the encoder LR to 5% of the chosen LR.
- Remove the decoder from a segmentation CNN architecture, append a classification head and train the whole network from scratch.

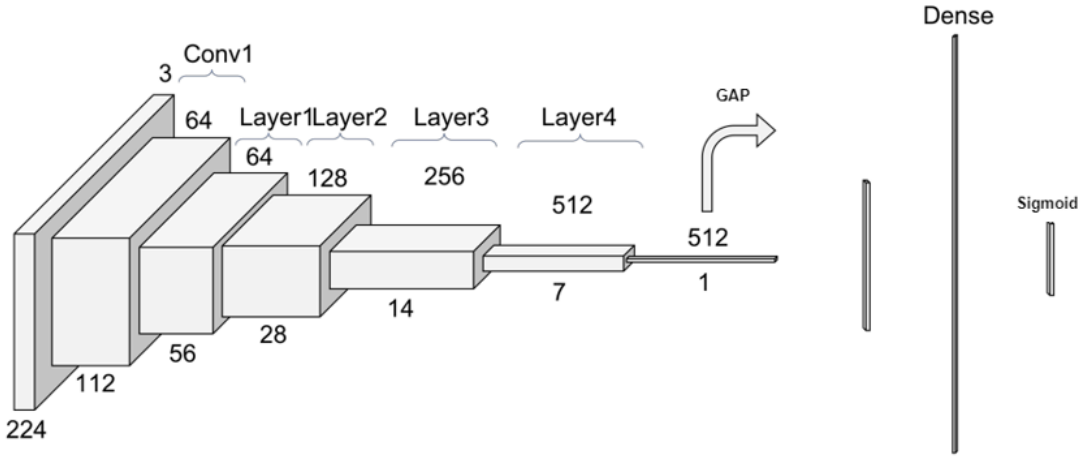


Figure 6.4: Starting from a segmentation CNN model, the decoder has been discarded and a classification head has been attached at the end, with a final sigmoid layer for binary classification purposes.

Figure 6.4 shows the selected classification CNN architecture, made of an encoder followed by a GAP layer and a final dense classification layer. In the first case, we are effectively inspecting a CNN trained for segmentation purposes by keeping the encoder almost the same. One could still argue that since the objective of CNN changed from regression to classification, keeping the same encoder should not be the best choice to get the best performance in a classification problem. This is probably true, but remember that our goal is not building an efficient classification network but inspecting how a segmentation CNN works.

SEAM Dataset for classification

Next problem to address is how to use SEAM segmentation dataset for classification: first, compute the percentage $S_{\%}$ of salt pixels in each patch, then assign a label to that patch depending on a predefined threshold. Two policies have been applied:

- Extract patches of dimension 512×512, set only one threshold to 20% such that for every patch if $S_{\%} < 20\%$ the patch is labeled as background, else it will be labeled as salt.
- Extract patches of dimension 128×128, label a patch as background if $S_{\%} < 5\%$ or as salt if $S_{\%} > 50\%$. Discard the patch for all middle values.

The first policy has been applied for pre-trained segmentation networks re-adapted for classification. Patch shape is the same for both classification and segmentation training, ensuring that the encoder works on the same input dimensions for both tasks. To get

a big enough dataset patches have been labelled with a 20% salt threshold with no gap between the two classes: in this way no patches are discarded but the definition of salt and background is ambiguous. In both cases, the result is a newly generated dataset containing patches labeled as salt or background depending on the salt percent present. Figure 6.5 shows an example of a patch classified as background in both cases, having $S_{\%} < 20\%$. Figure 6.6 has a percentage of salt such that $20\% < S_{\%} < 50\%$, meaning that will be classifiead as salt if the first policy is applied, otherwise it will be discarded.

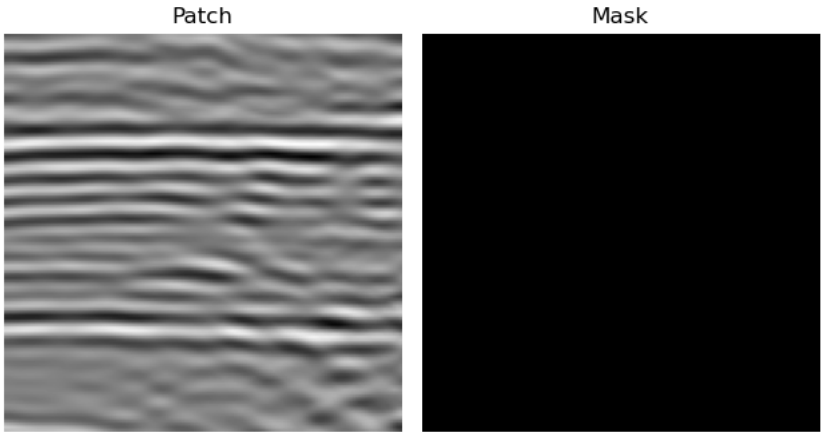


Figure 6.5: Patch and mask of a sample labelled as background

The second policy have been applied from CNNs trained from scratch: by setting a patch shape of 128×128 much more patches are produced. In this way, it is feasible to set distinct thresholds for the two classes without loosing too many dataset samples. In total 4 CNNs have been trained for classification with the purpose of CNN inspection, keeping the same architecture described in Chapter 4, as shown in the figure 6.1.

6.5. Experimental setup: CNN inspection

CNN inspection is not an easy job since used techniques are optimization based. This means that finding the right parameters to ensure a decent image quality is a process that requires many attempts. In this chapter we will go through some implementation choices that have been made to make selected CNN inspection techniques work, bearing in mind that there could be better and more elegant ways to do so.

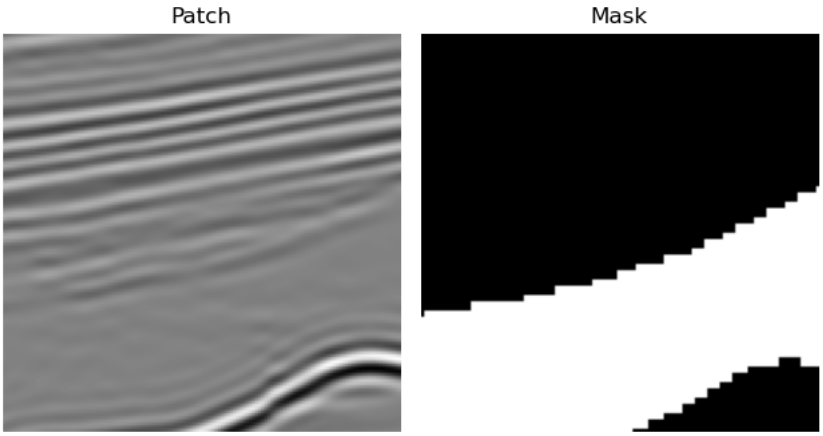


Figure 6.6: Patch and mask of a sample labelled as salt

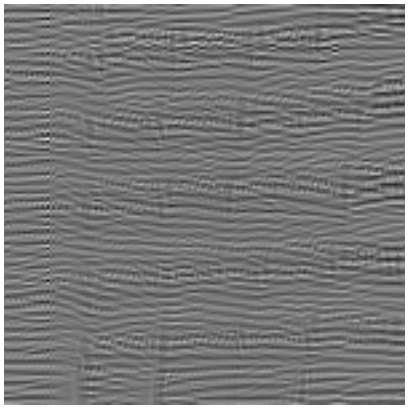
Dataset	Patch shape	Loss function	Lower threshold	Upper threshold
SEAM	512 x 512	BinaryCrossEntropy	20%	20%
SEAM	512 x 512	Dice	20%	20%
SEAM	128 x 128	BinaryCrossEntropy	5%	50%
LANDMASS	150 x 300	BinaryCrossEntropy	-	-

Table 6.1: Summary of CNNs trained for XAI purposes.

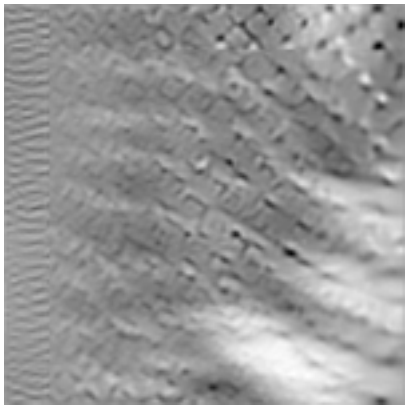
6.5.1. Activation Maximization setup

The first thing to consider when implementing AM is what layers and channels we are interested in inspecting.
The networks to be inspected are made of a convolutional encoder, a GAP layer and a final dense layer whose number of neurons depends on the dataset number of classes (2 for SEAM, 4 for LANDMASS dataset). For sure we are interested in understanding how CNNs understand classes just before outputting a result, this translates to performing AM on the last dense layer, maximizing the activation of one neuron at the time. What else can we do with this technique? Since we can look at one channel or neuron at the time, sadly not much.

If we want to inspect what the encoder part of the network has learned we should optimize every channel for every layer one by one, but then we will end up with a bunch of AM generated images that are somehow related to each other in an unknown manner. Visualizing deep layers of a CNN, even if only some channels will be selected, can still be interesting for academic purposes.
For this reason AM have been applied only the last dense layer, on each class neuron separately. Note that class neurons have been optimized pre-activation. This choice have been made since it is known that applying AM on after-softmax neurons might lead (not always) to misleading results. This happens (as stated in Section 5) because, for post-softmax AM, optimization process might prefer pushing other class activation down instead of maximizing the activation of the desired class: for this reason post-activation AM might result in poorly understandable images. Sometimes, on the other hand, optimization process could go well for post-softmax activation as well.



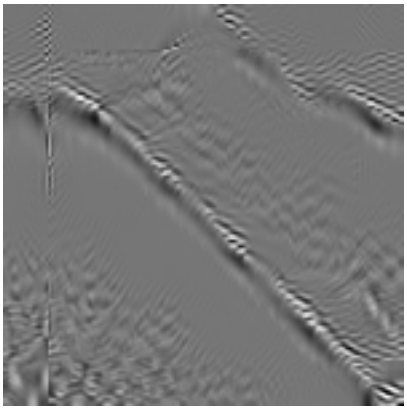
(a) Pre-softmax AM on class Reflector



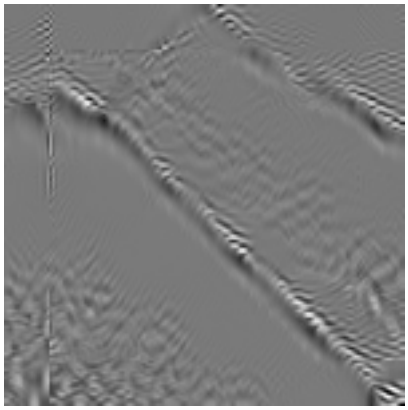
(b) Post-softmax AM on class Reflector

Figure 6.7: Example of how optimization can go wrong when applying AM method on pre or post softmax activation on last dense layer. Note how in this case post-softmax optimization leads to noisy results.

Figure 6.7 shows how optimization post softmax could be difficult to interpret. On the other hand, Figure 6.8 shows how the optimization procedure can get nice results for both the pre- and post-softmax layer. Probably the optimization process in Figure 6.8 landed on the same minimum in both cases, there are some small differences between images probably due to the difference in initialization. AM optimization has been carried out optimizing images of size 150×150, initialized as white noise, with a LR of $5 \cdot 10^{-3}$ for 2048 steps. For every neuron or channel, different combinations of transformations and parametrizations have been applied:



(a) Pre-softmax AM Background class



(b) Post-softmax AM on Background class

Figure 6.8: Another example of pre and post softmax application of AM. In this example both pre or post softmax optimization procedure produces good results.

- Case 1; No parametrization - No transformation
- Case 2: No parametrization - jitter only transformation of 8 pixel width and height.
- Case 3: No parametrization - padding, jitter, random rotation around 10 deg, random scaling from 80% to 120% with respect to original image size.

No regularization therm has been used during AM optimization. As stated previously in Chapter 5 regularization can help reduce high-frequency noise that may affect generated images, but may also discourage high frequencies that should naturally belong to the synthetic image. Transformation robustness can provide high random noise resistance without imposing frequency restrictions, enabling the image to reach high frequency peaks if needed.

6.5.2. Network Inversion setup

NI setup is overall similar to AM as they are based on a similar optimization process but with different loss function.
NI requires an input image as a target, since both the synthetic image and the input image must be coherent with CNN pre-processing, the first step is to apply z-score normalization to the input. Next thing to consider is what layers should we inspect with NI techniques: to give a comprehensive view on how the image is processed through the network, both shallow and deep layers have been selected.

Figure 6.9 shows what layers have been selected for NI visualization, one layer has been

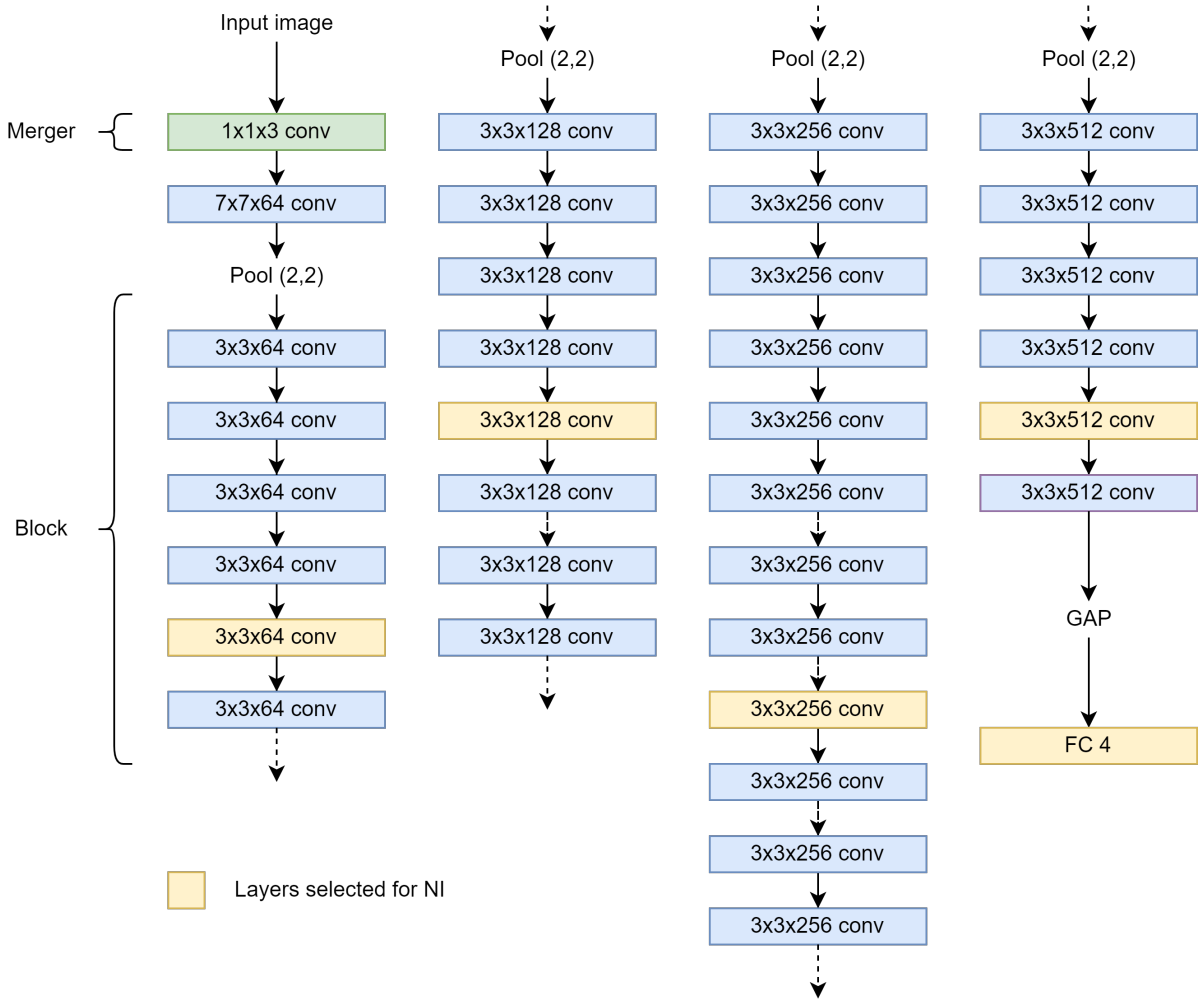


Figure 6.9: Simplified ResNet34 architecture with highlighted in yellow selected layers for NI. The Merger layer is in green because is not a proper part of ResNet34.

selected for every ResNet block, layer decision making has been made by testing.
The optimization process has been carried out using the loss function defined in equation 5.5. The regularization has been performed using cosine similarity as distance metric, elevated to a power that determine how similar the synthetic image have to be with respect to the input image:

$$\lambda(x) = \frac{x \cdot x^*}{\|x\| \, \|x^*\|}^{p} \tag{6.17}$$

For $p = 0$ x and x^* only have similar activation maps. By increasing p one can enforce similarity also on the two images.
The optimization process have been carried out with the following parameters:

- $LR = 10^{-3}$
- Adam optimizer.
- 1500 optimization steps.
- Transformation robustness including padding, jitter, random rotation around 10 degrees, random scaling from 80% to 120%.

Using a too high LR or training for a too small number of epochs might result in images with scattered blurred patches.

6.6. Segmentation results

Here results on the segmentation tass will be discussed, showing how trained CNNs perform with different loss functions with the addition of some predicted masks of best or worst inlines. Average metrics can be computed in two ways:

- First compute scores for each IL, then perform an average. In this way the error distribution along the IL influences the final average scores.
- Compute scores on all the IL as a volume. This method provides a more general information on segmentation performances.

The last raw of the table represents the "baseline", that is a CNN trained with the same Unet-Resnet34 architecture but with combined loss function. The baseline loss is a linear combination of BCE and DIC loss functions, with weights equal to one. Baseline details have been given to us by ENI, the purpose of this testing session is to see if some other, less common losses can perform better.
Note that best performances for mean metrics (Table 6.2) have been reached with JAC loss, even though JAC does not perform the best on all scores. Performances on mean

Loss	DICE	JAC	TPR	TNR	GCE	VS	PS	MI	AUC
BCE	0.812638	0.721974	0.949435	0.975198	0.046727	0.846757	0.690128	0.237378	0.962317
Contour	0.848063	0.758775	0.948355	0.981226	0.042707	0.884685	0.733327	0.235857	0.96479
Dice	0.871489	0.802302	0.956411	0.986096	0.032658	0.896936	0.858734	0.251895	0.971254
Focal	0.803855	0.706648	0.952539	0.975561	0.047399	0.83405	0.665581	0.236086	0.96405
Hausdorff	0.83674	0.749267	0.868182	0.9806	0.046255	0.910073	0.900011	0.227698	0.924391
Jaccard	0.888863	0.816848	0.933841	0.987378	0.03564	0.927317	0.781028	0.242153	0.96061
KL	0.85725	0.777592	0.951931	0.980296	0.040149	0.892909	0.767319	0.243421	0.966113
Lovasz	0.861183	0.780118	0.946661	0.983673	0.039956	0.893133	0.876121	0.239472	0.965167
MSE	0.708325	0.598997	0.949385	0.950032	0.076974	0.736457	0.643612	0.213206	0.949708
Wing	0.867064	0.790348	0.947193	0.980141	0.041123	0.903478	0.907862	0.240946	0.963667
BCE + Dice	0.865285	0.787707	0.942216	0.980308	0.041086	0.90311	0.849794	0.241284	0.961262

Table 6.2: Summary of the mean scores obtained from segmentation experiments with different losses. The last raw represent the baseline to improve.

scores are usually lower than volumetric scores: mean score metrics are heavily influenced by the distribution of scores along IL. If the performance gap among ILs is large, mean scores will probably be lower, on the other hand if the error distribution is close to uniform mean scores, it will be similar to 3D scores. Another important metric is PS, as it determines how well CNN is able to recognize the salt body perimeter: The best loss for both volumetric and mean scores resulted to be Hausdorff. Generally good performing losses have high scores on more or less all scores, with some exceptions. Since there are plenty of different scores to reason how good a CNN is, we will take into consideration just two of them for now and show how different CNNs perform in best and worst case scenario. Scores taken into account are:

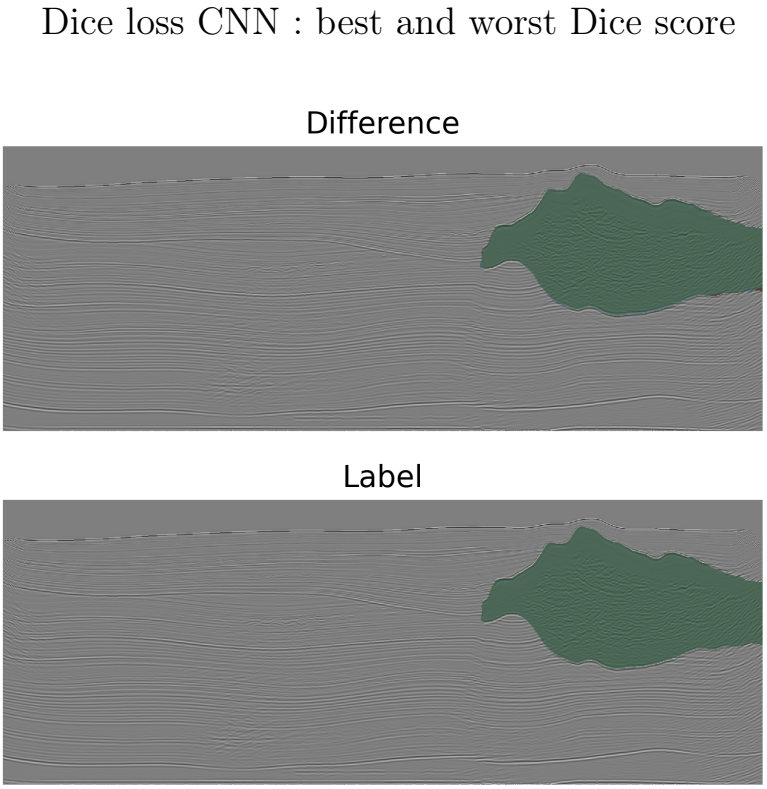
- DIC: Dice score is the most common score used for segmentation purposes, and it can provide a good all around view of how a CNN performs.
- PS: Perimeter score is particularly important in this application. The reason behind salt segmentation is identifying with as much accuracy as possible where the salt body begins and ends. PS focuses on this aspect by computing the score function only on the salt perimeter.

Now, for the best performing losses in DIC and PS score reported in Table 6.3, the best and worst IL prediction will be reported, along with the true salt mask. The following

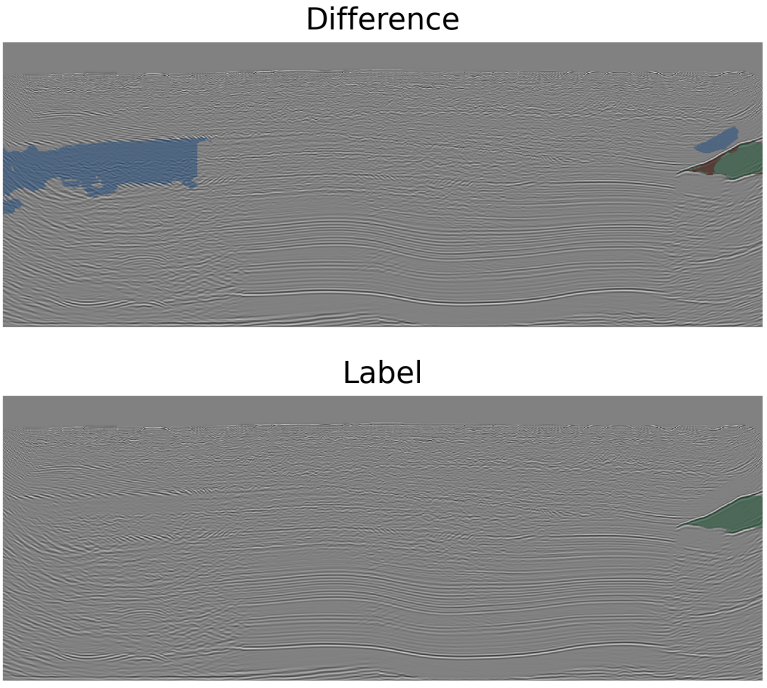
Loss	DICE	JAC	TPR	TNR	GCE	VS	PS	MI	AUC
BCE	0.8914	0.8041	0.9513	0.975834	0.050586	0.937081	0.748591	0.267418	0.963542
Contour	0.902134	0.821717	0.934523	0.981874	0.04518	0.965342	0.773261	0.26849	0.958199
Dice	0.923534	0.857931	0.944873	0.98662	0.035307	0.977416	0.88268	0.283316	0.965746
Focal	0.890961	0.803364	0.947971	0.976235	0.050733	0.939862	0.727423	0.2662	0.962103
Hausdorff	0.893505	0.80751	0.919971	0.981612	0.04892	0.971232	0.936658	0.260388	0.950791
Jaccard	0.917728	0.847964	0.924482	0.988086	0.037641	0.992694	0.784634	0.275303	0.956284
KL	0.907488	0.830644	0.949337	0.981134	0.04293	0.955918	0.798635	0.2753	0.965236
Lovasz	0.907314	0.830352	0.92867	0.984367	0.042619	0.977004	0.903268	0.270009	0.956518
MSE	0.824938	0.702038	0.960891	0.951318	0.083183	0.858514	0.692863	0.240208	0.956105
Wing	0.90502	0.826518	0.945128	0.981053	0.044022	0.957564	0.923319	0.272822	0.96309
BCE + Dice	0.905626	0.827529	0.94467	0.98131	0.043728	0.958669	0.862574	0.273016	0.96299

Table 6.3: Summary of 3D scores obtained from segmentation experiments with different losses

figures (example in 6.10a) show the predicted mask and the ground truth superimposed. Green pixels means that the prediction is correct (prediction and ground truth overlap), Blue pixels means false positives (FP) (background identified as salt), Red pixels means false negatives (FN) (Salt identified as background).



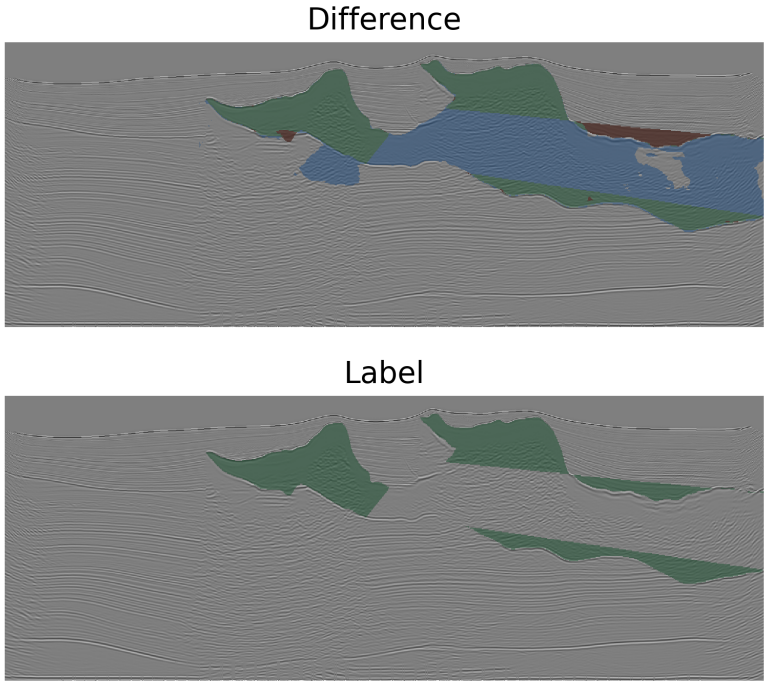
(a) Best DIC score obtained at IL 494



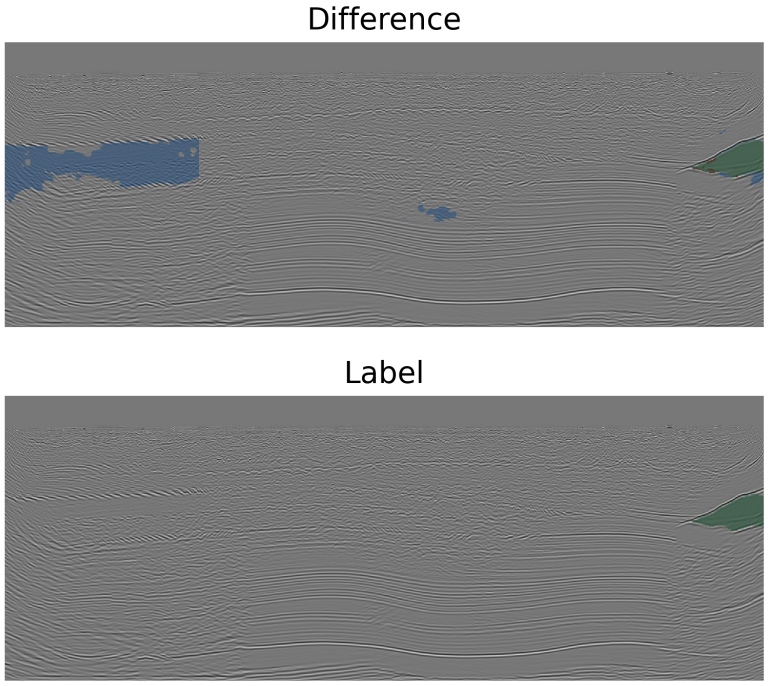
(b) Worst DIC score obtained at IL 995

Figure 6.10: Best and worst DIC score obtained by a CNN trained with Dice loss.

Dice loss CNN : best and worst Perimeter score

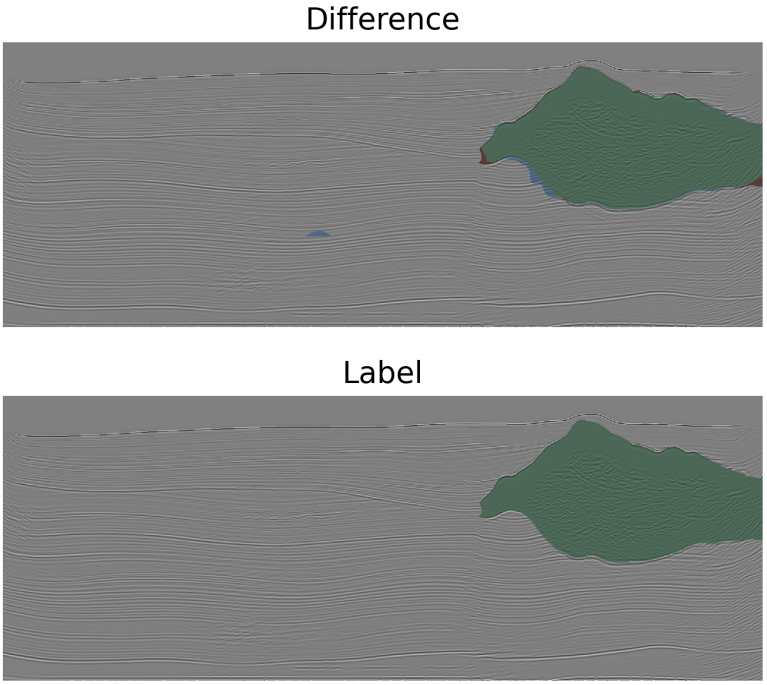


(a) Best PS score obtained at IL 345

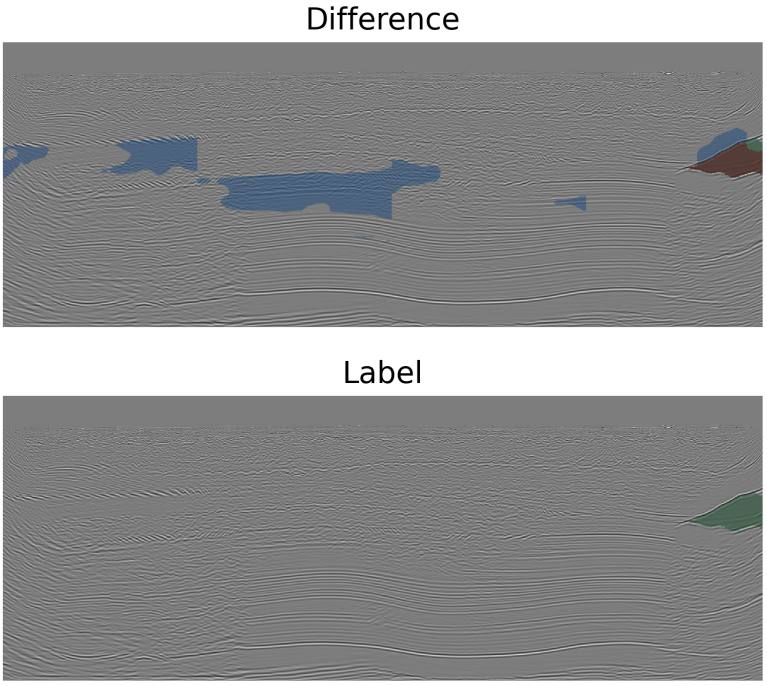


(b) Worst PS score obtained ad IL 1011

Figure 6.11: Best and worst PS score obtained by a CNN trained with Dice loss



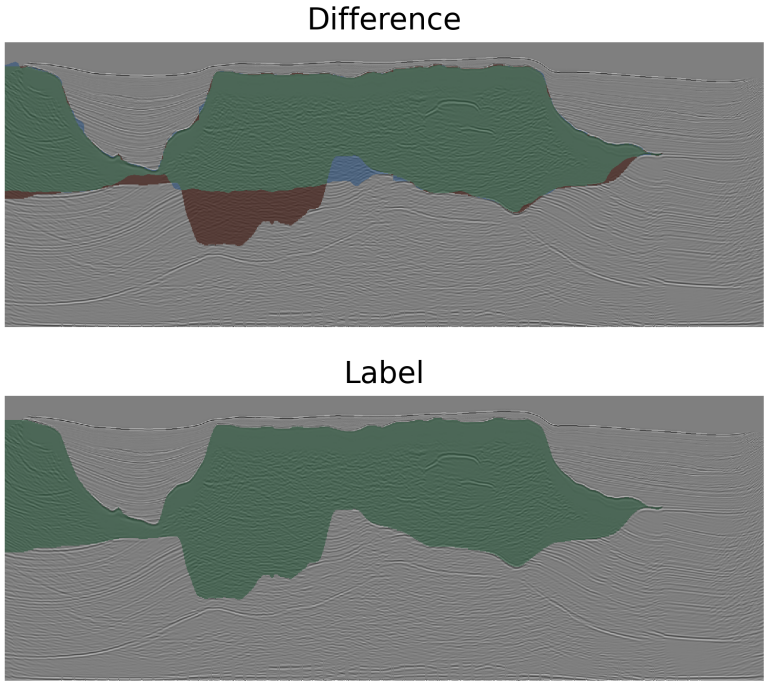
(a) Best DIC obtained at IL 503



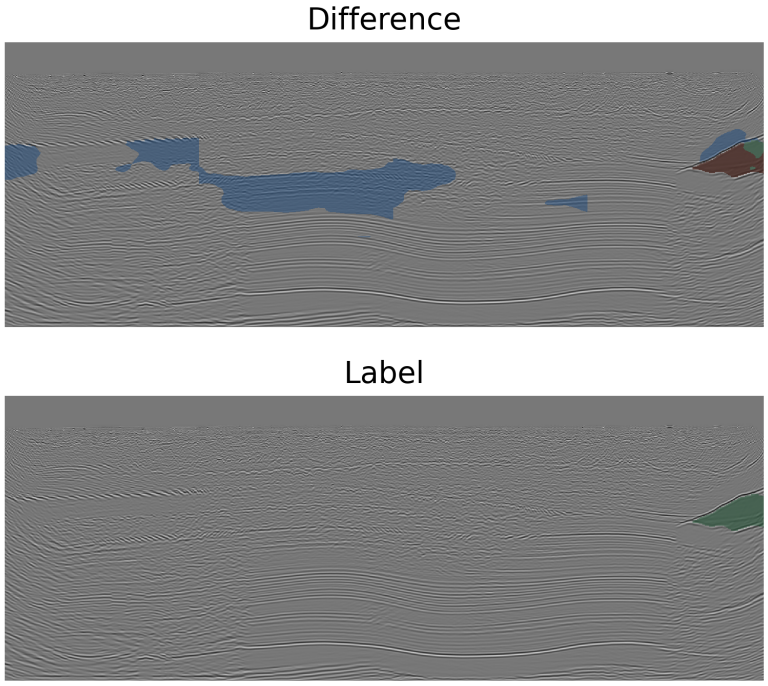
(b) Worst DIC score obtained at IL 1009

Figure 6.12: Best and worst DIC score obtained by CNN trained with Hausdorff loss.

Hausdorff loss CNN : best and worst Perimeter score



(a) Best PS score obtained at IL 30



(b) Worst PS score obtained at IL 1011

Figure 6.13: Best and worst PS score obtained by CNN trained with Hausdorff loss.

Among all the losses tested, only the two best performing are reported: Dice loss, which resulted in having the best DIC score, and Hausdorff loss, which reached the maximum PS score.

Both CNNs (as shoin in Figures 6.10 and 6.12) achieve the lowest and highest value of DIC score for about the same ILs. Those specific IL ranges (around 500 and around 1000) resulted to be respectively very easy and hard to segment for almost all the proposed losses. Note how around IL 1000 both Dice and Hausdorff CNNs have plenty of false positives that bring down both perimeter and dice score.
It is interesting to see how IL 345 (Figure 6.11a) has the best PS score of the whole test set. The CNN prediction is filled with false positives, but since these pixels are not on the salt perimeter, they do not contribute to PS computation. The prediction on the salt perimeter is, on the other hand, overall accurate.
Figure 6.13a is another example of high PS; the CNN trained with Hausdorff loss was able to recognize with great precision the general border of the large salt body present, missing only the bottom lobe. Note how the two losses behave differently for IL around 1010: Dice CNN correctly recognized the salt body but added some false positives (figure 6.11b), Hausdorff CNN on the other hand missed completely the salt body (figure 6.13b).

6.7. Interpretation results

In this section, all results regarding XAI will be reported. First we will go through SEAM dataset CNNs, with and without pretrainig, and see how segmentation networks interpret classes salt and background using AM technique imposing different transformation robustness and parametrization, also some images obtained with NI will be provided, to show how the encoder part of CNN elaborates the input image. Then we will go through LANDMASS dataset CNNs, showing how those techniques work with CNNs trained directly for multi-class classification.

SEAM Dataset CNNs

The first CNN to be inspected uses an encoder trained for segmentation purposes, the classification head is then trained on a new classification task splitting salt patches with a threshold of 20%. Note that there is no gap between background and salt class, so definition of salt and background can be ambiguous at times. Summary training details can be found in 6.4

Patch shape	Encoder loss function	Pretraining	Upper threshold	Lower threshold	Encoder LR
512 x 512	BinaryCrossEntropy	True	20%	20%	5%

Table 6.4: CNNs training for XAI purposes.

By keeping the encoder LR low, we should be able to inspect in the most faithfully way possible how the segmentation encoder interprets the two output classes.

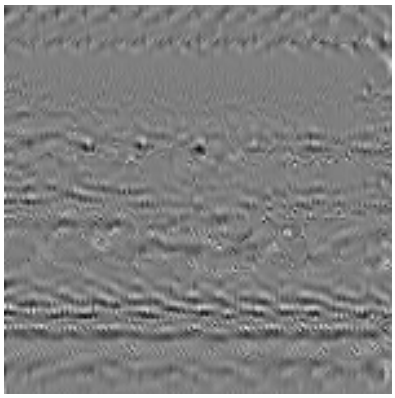


Figure 6.14: No transformation robustness

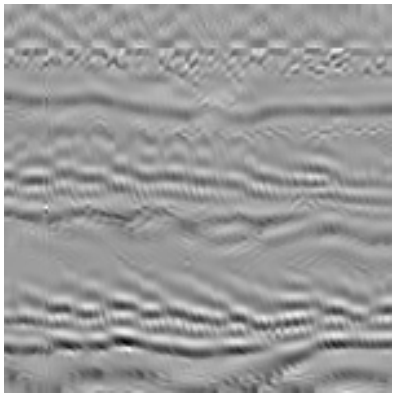


Figure 6.15: Jitter only

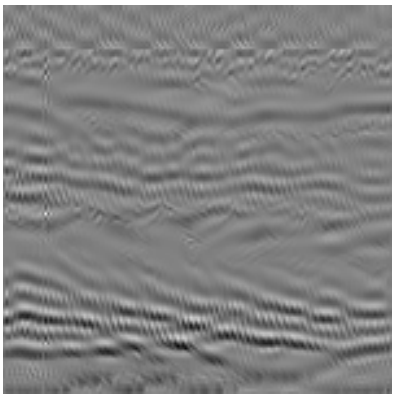


Figure 6.16: Padding, jitter, rotation, scaling robustness

Figure 6.17: Class Background interpretation for BCE pretrained CNN

Figure 6.17 is the result of AM application on last dense neuron representing class background of CNN trained as stated in Table 6.4: 6.14 shows unparametrized AM with no transformation robustness applied, the result is a little more noisy image with respect to the other two. In all three images horizontal reflectors are easy to spot; they faithfully retain features of reflectors usually found in seismic images. We will see later that this fact does not hold for non-pretrained networks, where resulting images will be much harder to interpret.

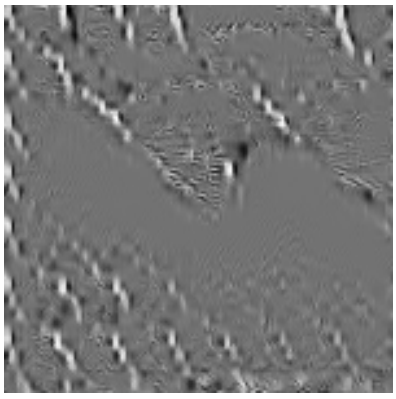


Figure 6.18: No transformation robustness

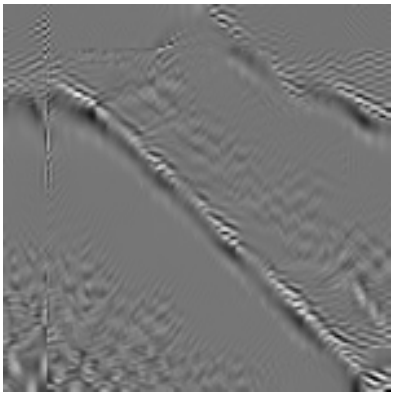


Figure 6.19: Jitter only

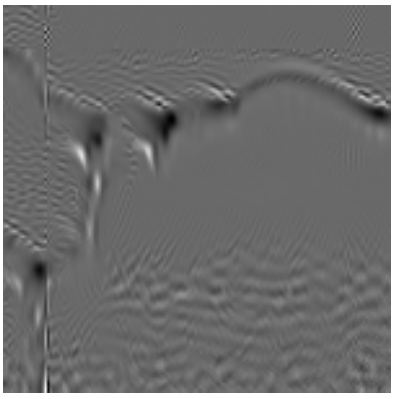


Figure 6.20: Padding, jitter, rotation, scaling robustness

Figure 6.21: Class Salt interpretation for BCE pretrained CNN

Figure 6.21 shows the interpretation of class Salt instead. Note that, especially in the last two images, salt body discontinuity is sharp and stands out with respect to the rest of the image: this tells that the CNN has effectively learned to recognize salt bodies by looking at the surface discontinuity between background and salt. The first Figure (6.18) is clearly more noisy than the other two due to the lack of transformation robustness.
To inspect how the encoder elaborated the input image as a whole, an example of NI is provided for some selected layers (yellow boxes in diagram 6.9)

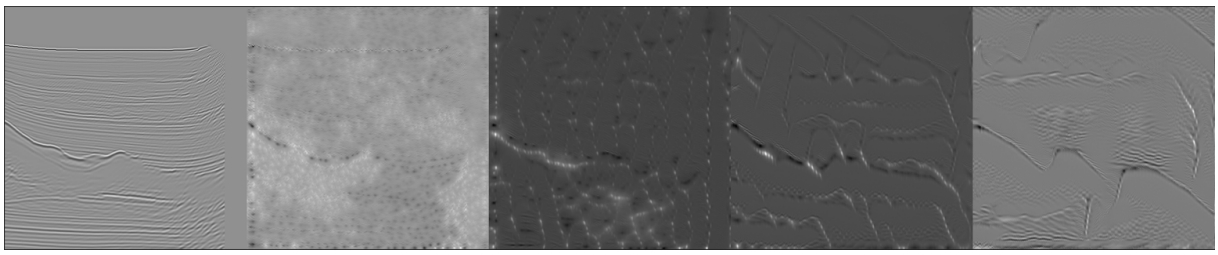


Figure 6.22: NI with no constraints applied to a pretrained CNN. The image on the left represent the input image.

Figure 6.22 shows NI with no regularization therm. Note how the deeper layer we inspect, the more the input image gets modified by the encoder. Since the regularization therm is

absent, contrast in different layers can vary a lot.

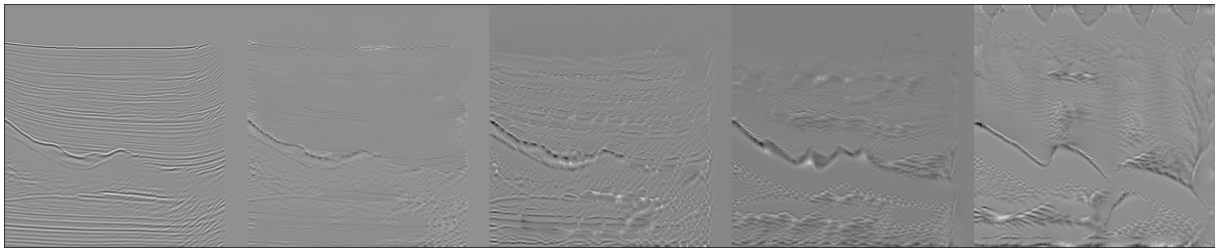
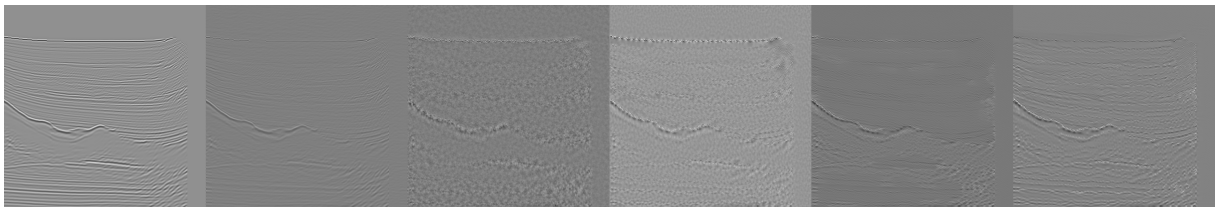
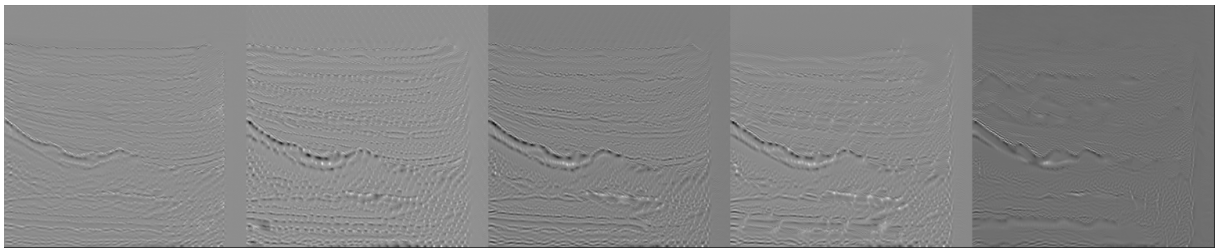


Figure 6.23: NI with constraints applied to a pretrained CNN

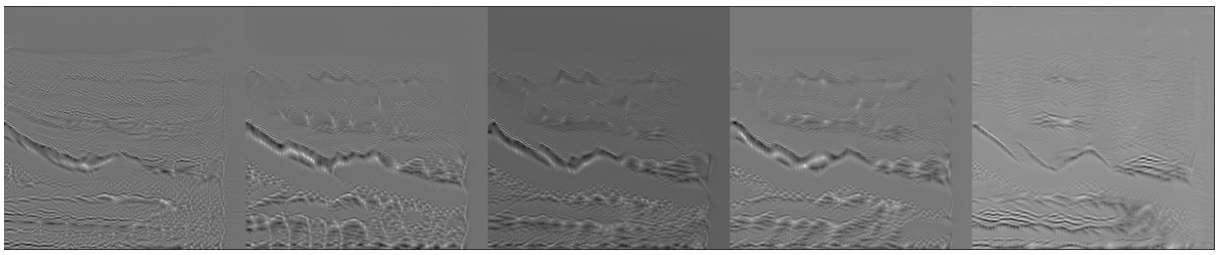
Figure 6.23 shows NI, respectively, with a strong regularization therm. The biggest impact in regularization is always the contrast: strongly regularized images usually have a contrast similar to the original image. Also here, like in the previous case, it is clear how the input image almost remains the same through the first layer while is heavily distorted in the last layer.
The most important point to make with NI for this CNN is that details of input images gets distorted but not discarded. This fact is crucial for segmentation CNN: one possible explanation for why it happens is that segmentation CNNs have to retain some sort of spatial cognition of where different parts of the image are located in order to perform segmentation correctly. We will see that for classification CNNs this does not hold, since their job is just to classify the input image without caring about where details are spatially located.
For this CNN only we will show a more complete view of encoder part of the network by applying NI to five convolutional layers per block (4 blocks in total, see Figure 6.9 for details).



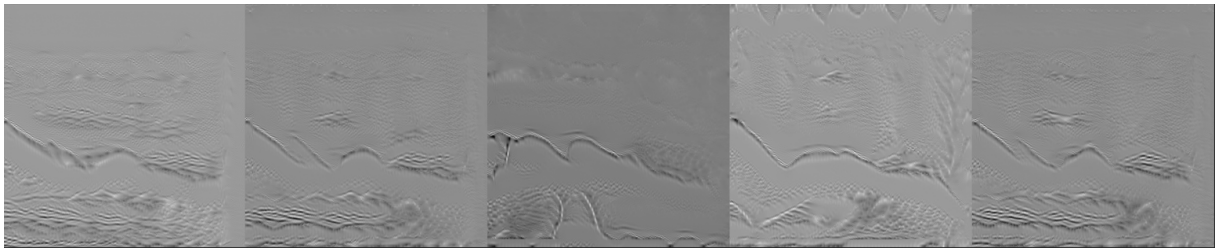
(a) Block 1. First image on the left shows the input image.



(b) Block 2



(c) Block 3



(d) Block 4

Figure 6.24: NI applied to all four blocks of the encoder.

Note how the input image is processed more and more from left to right and from up to down; Figure 6.24 shows how gentle the transformation of the input image is. For all future CNNs we will avoid applying NI to the whole encoder.

Next CNN to be inspected has been trained from scratch for classification purposes using smaller patches (128 x 128) in order to obtain classes with less ambiguity, with a threshold

of $< 5\%$ for background and $> 50\%$ for salt. For training summary see Table 6.5.

Patch shape	Encoder loss function	Pretraining	Upper threshold	Lower threshold
128 x 128	BinaryCrossEntropy	False	50%	5%

Table 6.5: CNNs training for XAI purposes.

In this CNN the encoder has also been trained for classification purposes, this will lead to very different result with respect to segmentation - pre-trained CNN.

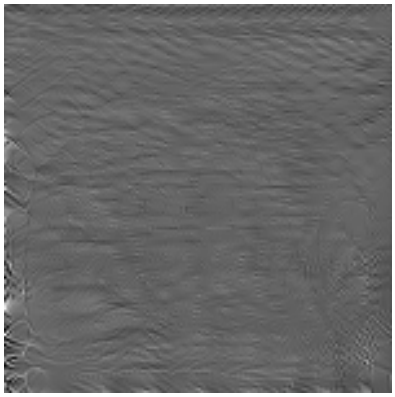


Figure 6.25: No transformation robustness

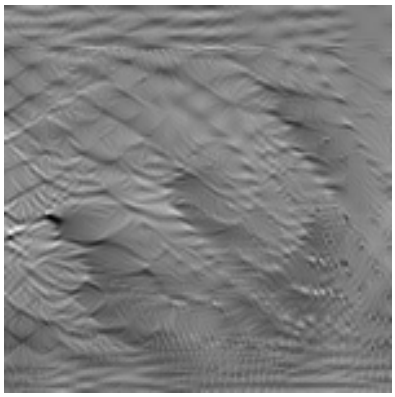


Figure 6.26: Jitter only

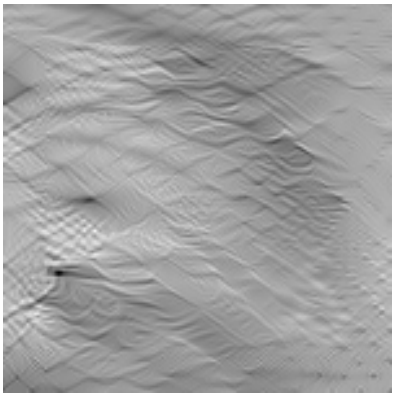


Figure 6.27: Padding, jitter, rotation, scaling robustness

Figure 6.28: Class Background interpretation for BCE non - pretrained CNN

All three images in Figure 6.28 are very different from those in Figure 6.17. In this case reflectors are still visible but much more distorted, also each image contains much more replicas of reflector class, this could be due to smaller patch size. Lastly, Figure 6.25 is extremely noisy, transformation robustness is in this case crucial to obtain decent results.

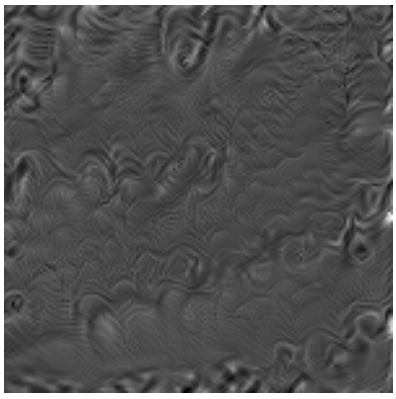


Figure 6.29: No transformation robustness

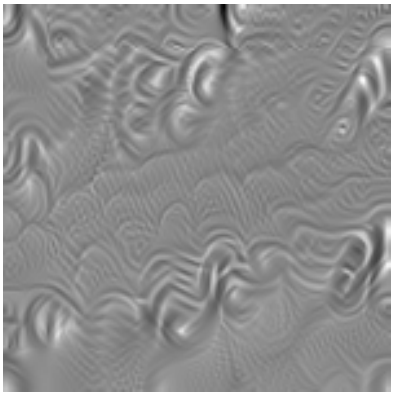


Figure 6.30: Jitter only

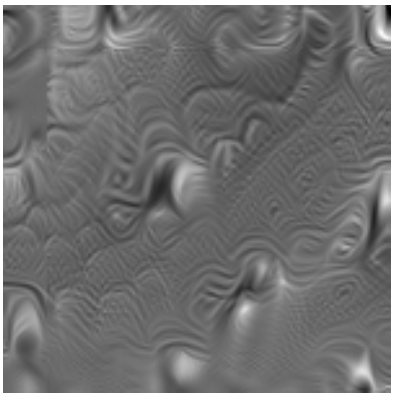


Figure 6.31: Padding, jitter, rotation, scaling robustness

Figure 6.32: Class Salt interpretation for BCE non - pretrained CNN

Also for salt class one can do the same reasoning that have been done for background class. In Figure 6.32 the only information retained by class salt is the contrast gap between different parts of the image. Now is not possible anymore to recognize salt borders as in Figure 6.21.
Why does the two CNNs described in Tables 6.4 and 6.5 have such a different interpretation of the two classes? The CNN trained for scratch is only for classification purposes and classification task resulted to be much easier than segmentation, in fact training process ended up in about 20 epochs with very high scores. Probably the CNN trained appositely for classification does not need to "understand" salt bodies in such a detailed manner since the classification resulted to be very easy. On the other hand, segmentation training took about 100 epochs, resulting in a much harder process. Also the segmentation CNN has to retain information about spatial location of salt perimeter to perform segmentation, probably resulting in a more "sharp" interpretation of classes salt and background.
NI applied on this CNN could confirm this hypotheses, showing how differently input images are processed with respect to the previous case.

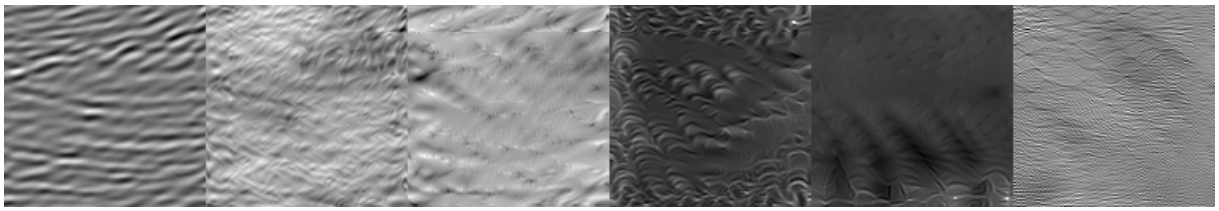


Figure 6.33: NI of a class background image for non pretrained CNN, the input image is the first on the left.

Figure 6.33 shows regularized NI on background class image. Note how details about the position and shape of the horizons get lost quickly as we go deeper into CNN. Last image on the right shows class neuron NI: in this layer the position and features of horizons is totally lost, substituted by the general interpretation of class background. Note in fact the similarity with Figure 6.28.

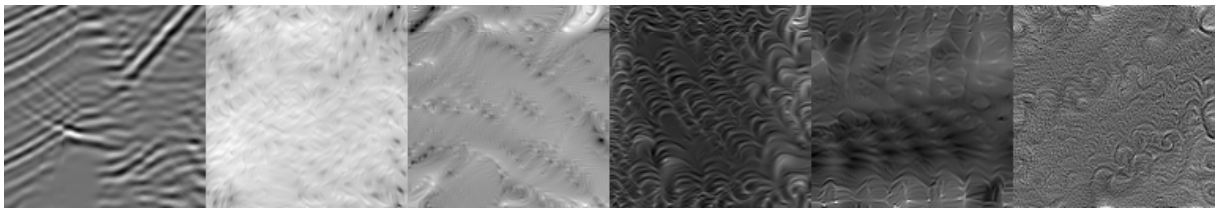


Figure 6.34: NI of a class salt image for non pretrained CNN.

Also, for NI of class salt (Figure 6.34) features of salt bodies present on the original image on the left quickly fades the deeper we go into the network. NI applied on the last dense layer (image on the right) only shows the general understanding of class salt, as in Figure 6.32.

LANDMASS Dataset CNNs

LANDMASS dataset has the only purpose of classification, for this reason we can't make a comparisons between segmentation and classification encoders, since only classification CNNs were trained.
On the other hand, since LANDMASS dataset were created for classification, there is neither ambiguity or class imbalance in the dataset. LANDMASS dataset labels images with four classes, which means that classification CNNs will have four output neurons. The CNN to be inspected was trained with BCE loss over images of size 150×150, note that LANDMASS dataset contains already split images, while SEAM dataset required

patching with arbitrary shape. A confusion matrix is provided to show classification performance on a test set, made of 15% of the total dataset samples (4000 samples):

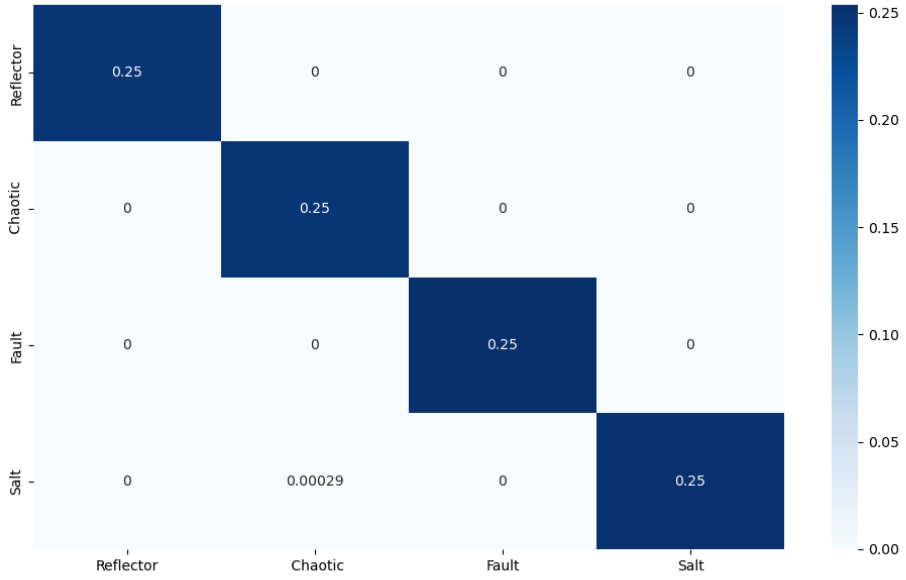


Figure 6.35: Confusion matrix of LANDMASS classification CNN

The confusion matrix in Figure 6.35 clearly shows how classification task was performed successfully with great results, with only one incorrect classification on chaotic class. Now an example of AM of every class will be provided:

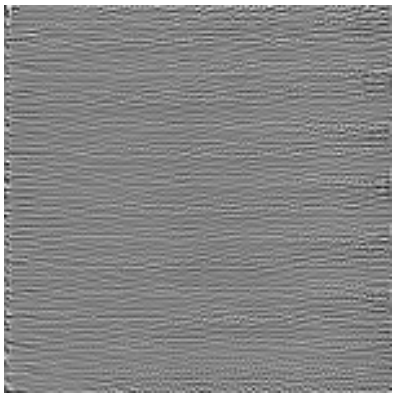


Figure 6.36: No transformation robustness

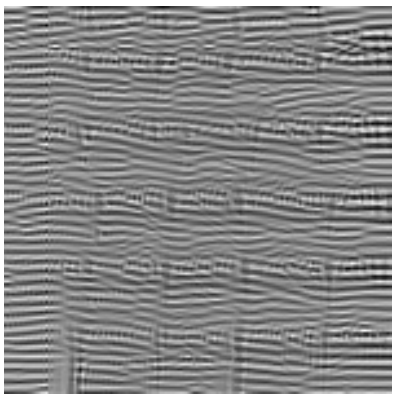


Figure 6.37: Jitter only

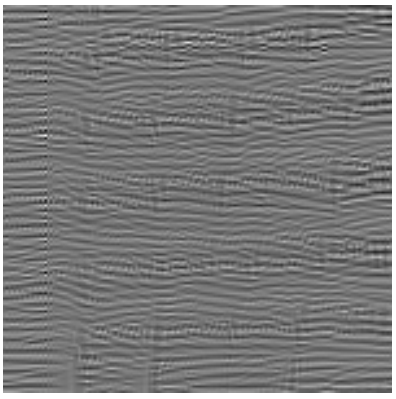


Figure 6.38: Padding, jitter, rotation, scaling robustness

Figure 6.39: Class Reflector Interpretation of CNN trained on LANDMASS dataset

Reflector class represent the "general" reflector in seismic images. AM representation on class reflector (Figure 6.39) clearly captures the regular horizontal pattern that characterizes this class.

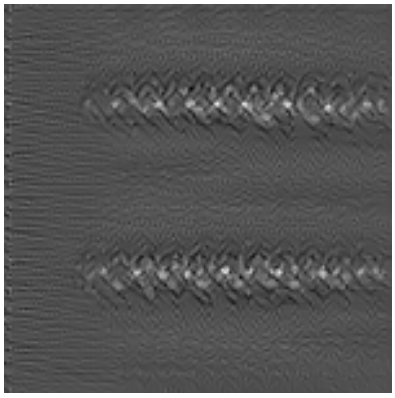


Figure 6.40: No transformation robustness

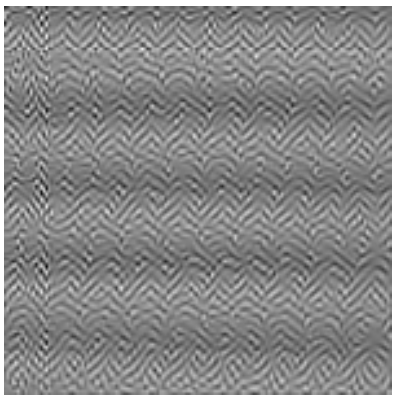


Figure 6.41: Jitter only

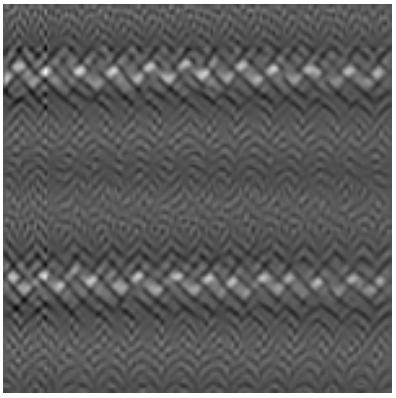


Figure 6.42: Padding, jitter, rotation, scaling robustness

Figure 6.43: Class Chaotic Interpretation of CNN trained on LANDMASS dataset

Chaotic is not a well defined class per se: it contains all types of artifacts that the exert could not really classify, resulting in a class with a large number of very different images.

The AM process, as show in Figure 6.43 somehow captured a contrast discontinuity with the background, that resembles the class reflector.

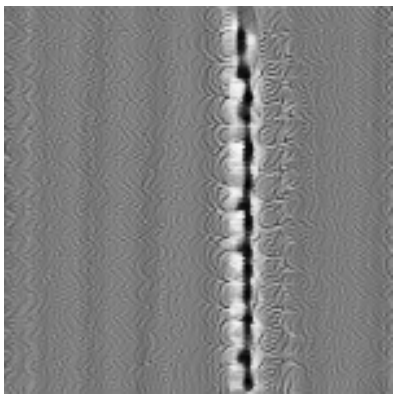


Figure 6.44: No transformation robustness

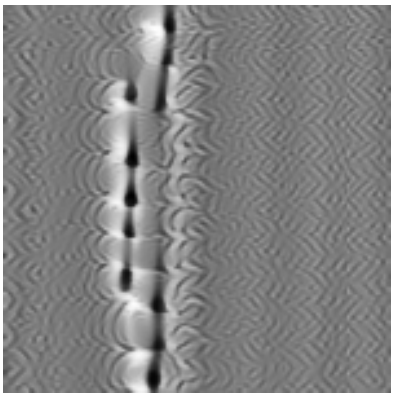


Figure 6.45: Jitter only

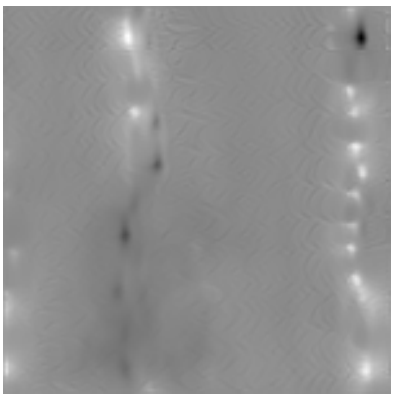


Figure 6.46: Padding, jitter, rotation, scaling robustness

Figure 6.47: Class Fault interpretation of a CNN trained on LANDMASS dataset

Fault class if perhaps the most interesting: AM generate images in Figure 6.47 shows how the CNN interpret fault class as vertical cuts. A fault is indeed a vertical fracture that propagates vertically through the whole image. Note how in this case imposing fewer transformation robustness resulted in more clear images. Another interesting detail is that the background present resembles a lot the AM interpretation on class chaotic in Figure 6.43

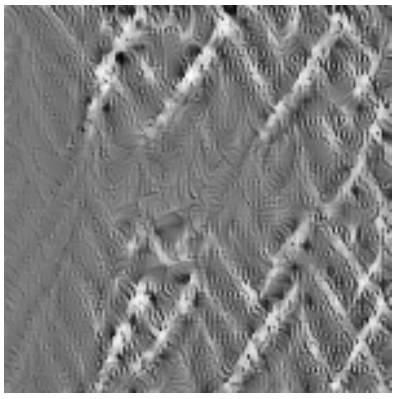


Figure 6.48: No transformation robustness

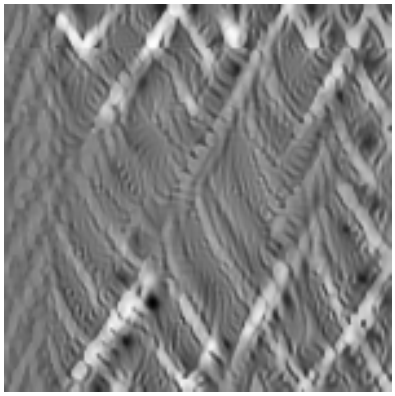


Figure 6.49: Jitter only

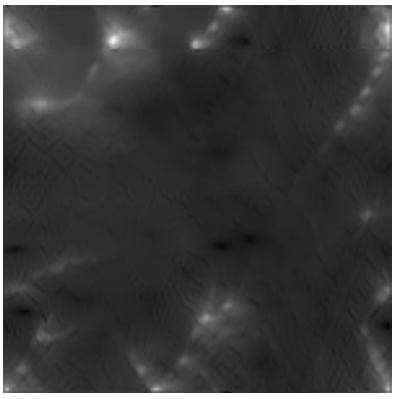


Figure 6.50: Padding, jitter, rotation, scaling robustness

Figure 6.51: Class Salt Interpretation of CNN trained using LANDMASS dataset

Lastly, salt class (Figure 6.51 is represented as triangular shaped artifacts across the image with different contrasts with respect to the background. This could in some way capture the general shape of salt bodies, which are usually jagged in shape with great discontinuity with the rest of the image.
The same consideration done for NI applied SEAM classification CNN (detains in Table 6.5) are valid here. Also in this case spatial details about the input images get rapidly lost leaving only a general representation of the input image. This loss of precision is unacceptable for the segmentation CNN, where details are preserved through the encoder. Let now see how encoder process images using NI:

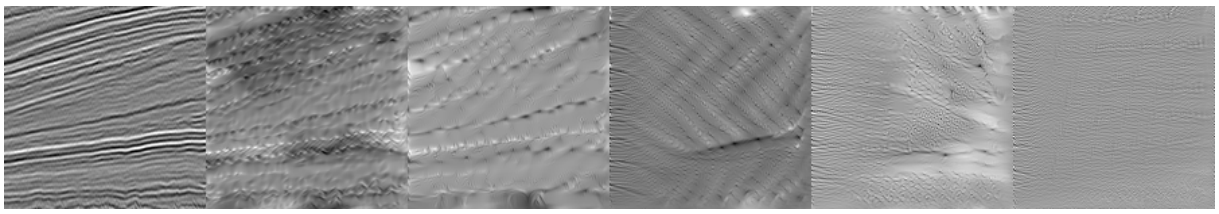


Figure 6.52: NI of a class Reflector on LANDMASS dataset

NI on reflector class in Figure 6.52 simply shows how, stating from the original image, details get lost more and more until the last dense layer, where only the general representation of class reflector remain (as in Figure 6.39).

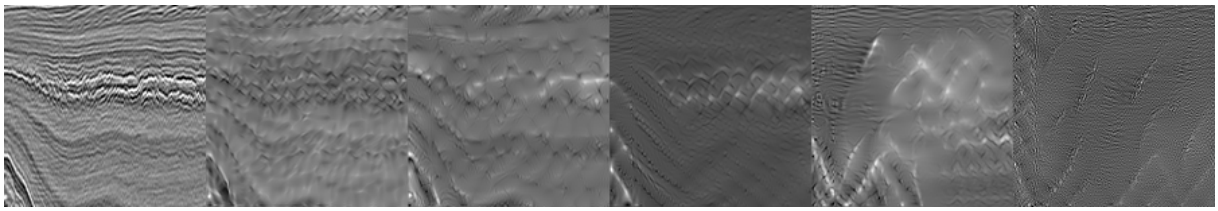


Figure 6.53: NI of a class Chaotic on LANDMASS dataset

Figure 6.53 shows how this chaotic class input image is processed through the network by "enhancing" the middle horizon that dominates the image. As always, the last dense layer retains almost no details about the original image, but why? This happens also because the last dense dense layer is made of only four neurons, each one giving a score on the probability of a class. When NI is applied on this layer, the feature map of the optimized image has to resemble as much as possible that of one of the input image (see Chapter 5 for NI details), but the dense layer feature map only contains four values indicating the probability of each class. In this case, since the input image is (and was classified as) chaotic, the four values composing the feature map will be three low values (those representing the other three other classes) and one high value (representing class chaotic). Only four neurons cannot incorporate too much information about the input image, for this reason NI process ends up obtaining images very similar to AM: in fact, what the NI algorithm is doing is effectively finding an input image that has a high score on the class chaotic and low scores on other classes. Similarly AM on class chaotic will find the input image that activates most the neuron of class chaotic. What differentiate the two optimization process is the regularization: AM is not bounded to any image, while NI forces a similarity between the generated image and the input image.

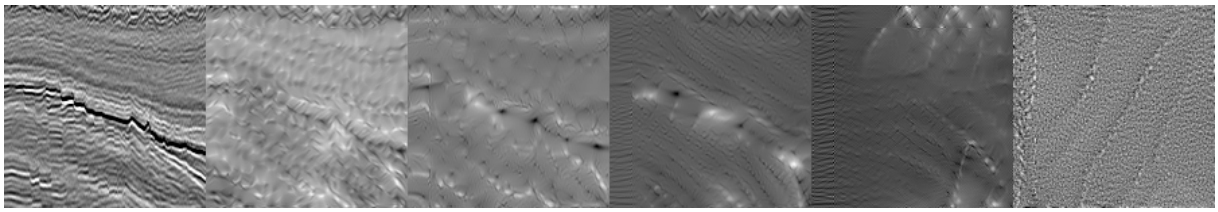


Figure 6.54: NI of a class Fault on LANDMASS dataset

Fault class NI (Figure 6.54) is probably the most interesting among the four.
The last image, that represent NI over the dense layer, shows clearly vertical lines that resembles fault formations over the image. Note that the input image only has one fault in it, while NI generated image has more in different positions: the presence of these replicas suggests that the CNN learned how to generalize the recognition of faults with

different shapes and sizes, placed in different parts of the image.

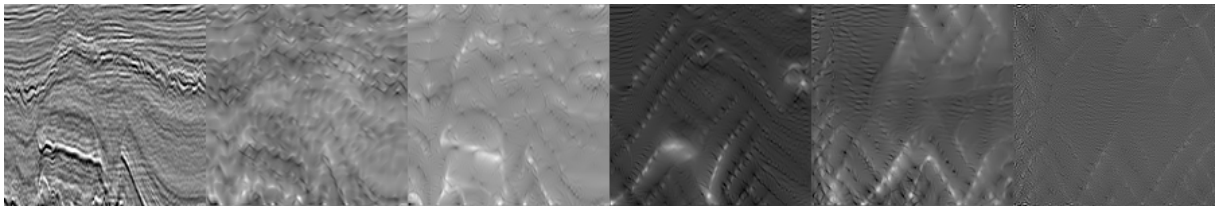


Figure 6.55: NI of a class Salt on LANDMASS dataset

Lastly, NI over the Salt class simply shows how the salt body input image is processed, highlighting the salt formations the deeper we go into the network. An example of NI on class Salt is reported in Figure 6.55

7 | Conclusions and future works

In this thesis we tackled the problem of salt segmentation in two different ways: In the first part, we proposed a baseline segmentation CNN and took care of all the steps, from data preparation to testing and metric evaluation. In the second part we inspected network behaviours using two XAI techniques called Network Inversion and Activation Maximization.
During the segmentation process multiple loss functions have been implemented and tested equally under the same architecture and training parameter, to give a general and fair overview of their performances for salt segmentation task. By comparing the obtained results with the provided baseline, a significant improvement has been obtained by simply selecting the best performing loss function.
Now that all losses have been successfully implemented, the next step to make to reach even higher segmentation performances is hyperparameter tuning: the fact that one loss performs better than another with the proposed baseline does not mean that by changing parameters like batch size, patch shape and size or LR the same will hold. To reach optimal results one should try different hyperparameter combination and find the more suitable for ResNet34 architecture. One further step to make to reach state of the art performances could be implementing ensemble-based predictions using multiple architectures. Now that all losses have been implemented and tested, performing hyperparameter tuning on different architectures can lead to an ensemble of highly performing CNNs that can be used for joint segmentation.

On the side of CNN inspection we successfully implemented AM and NI techniques to ResNet34 architecture achieving highly interpretable results. CNNs are still highly complex and hard to understand structures, implemented XAI techniques help understanding what a CNN has learned and how images are progressively processed the deeper we go into the network. With the help of this newly implemented methods we managed to shed a light into the black box of CNNs, potentially opening new strategies for training and evaluation. AM and NI techniques showed how different a classification and segmentation encoder interpret the world. By looking ad AM produced images we explained how a

segmentation encoder interprets class salt and background: the produced images contain usually only one very detailed replica of the selected class, embodying the concept of salt or background label. To reach decent segmentation performances the segmentation encoder must retain as much information as possible to produce a precise mask. On the other hand AM generated images on classification encoders show how different the understanding of the two labels can be. In this case, since the spatial location of details is not important, the learned meaning of the two classes is much more simple and details-free, with multiple replicas of class background and salt scattered all over the image.
NI technique confirmed the made hypotheses by showing how differently the two encoders process the input image: the segmentation encoder barely modify the image shape and size, retaining as much details as possible and enhancing those that are more important for segmentation purposes (such as salt perimeter). Classification encoders modify the input dramatically already from from shallower layers, progressively loosing image details leaving behind only its general concept.
One possible future step to make could be implementing grad-CAM based techniques and apply them to AM produced images to see what part of the synthetic images are considerate as highly informative for the CNN. One bigger step could be implementing a generator network using Generative adversarial network (GAN) training: as stated in Nyugen et al. article [27] AM generated images can be used as preferred initialization to train a generator network using GAN training, resulting in impressively realistic images. By reducing the number of optimization steps of the already implemented AM algorithm one can produce perfect starting images for generator networks.

Bibliography

[1] D. Adams and D. Markus. Time-frequency transform based nmo correction with auto-mute. 06 2014. doi: 10.3997/2214-4609.20140867.

[2] Y. Alaudah, Z. Wang, Z. Long, and G. AlRegib. LANDMASS Seismic Dataset, 2015. URL `http://cegp.ece.gatech.edu/codedata/LANDMASS/index.html`.

[3] S. Albawi, T. A. Mohammed, and S. Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6, 2017. doi: 10.1109/ICEngTechnol.2017.8308186.

[4] Y. Babakhin, A. Sanakoyeu, and H. Kitamura. Semi-supervised segmentation of salt bodies in seismic images using an ensemble of convolutional neural networks. In G. A. Fink, S. Frintrop, and X. Jiang, editors, *Pattern Recognition*, pages 218–231, Cham, 2019. Springer International Publishing. ISBN 978-3-030-33676-9.

[5] S. Basheera and M. Ram. A hybrid enhanced independent component analysis approach for segmentation of brain magnetic resonance image. *Defence Life Science Journal*, 3(3):285–292, Jun. 2018. doi: 10.14429/dlsj.3.11499. URL `https://publications.drdo.gov.in/ojs/index.php/dlsj/article/view/11499`.

[6] M. Berman and M. B. Blaschko. Optimization of the jaccard index for image segmentation with the lovász hinge. *CoRR*, abs/1705.08790, 2017. URL `http://arxiv.org/abs/1705.08790`.

[7] F. Bre, J. Gimenez, and V. Fachinotti. Prediction of wind pressure coefficients on building surfaces using artificial neural networks. *Energy and Buildings*, 158, 11 2017. doi: 10.1016/j.enbuild.2017.11.045.

[8] A. Carass, S. Roy, A. Gherman, J. C. Reinhold, A. Jesson, T. Arbel, O. Maier, H. Handels, M. Ghafoorian, B. Platel, A. Birenbaum, H. Greenspan, D. L. Pham, C. M. Crainiceanu, P. A. Calabresi, J. L. Prince, W. R. G. Roncal, R. T. Shinohara, and I. Oguz. Evaluating white matter lesion segmentations with refined sørensen-dice analysis. *Scientific Reports*, 10(1):8242, 2020. ISSN 2045-2322. URL `https://doi.org/10.1038/s41598-020-64803-w`.

[9] Z. Chen, H. Zhou, X. Xie, and J. Lai. Contour loss: Boundary-aware learning for salient object segmentation. *CoRR*, abs/1908.01975, 2019. URL `http://arxiv.org/abs/1908.01975`.

[10] J. F. Claerbout. Toward a unified theory of reflector mapping. *GEOPHYSICS*, 36 (3):467–481, 1971. doi: 10.1190/1.1440185. URL `https://doi.org/10.1190/1.1440185`.

[11] T. M. Cover and J. A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, USA, 2006. ISBN 0471241954.

[12] L. R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945. ISSN 00129658, 19399170. URL `http://www.jstor.org/stable/1932409`.

[13] D. Duque-Arias, S. Velasco-Forero, J.-E. Deschaud, F. Goulette, A. Serna, E. Decencière, and B. Marcotegui. On power Jaccard losses for semantic segmentation. In *VISAPP 2021 : 16th International Conference on Computer Vision Theory and Applications*, Vienne (on line), Austria, Feb. 2021. URL `https://hal.archives-ouvertes.fr/hal-03139997`.

[14] D. Erhan, Y. Bengio, A. Courville, and P. Vincent. Visualizing higher-layer features of a deep network. *Technical Report, Univeristé de Montréal*, 01 2009.

[15] M. Fehler and K. Larner. SEG Advanced Modeling (SEAM) : Phase I first year update. *The Leading Edge*, 27(8):1006–1007, 08 2008. ISSN 1070-485X. doi: 10.1190/1.2967551. URL `https://doi.org/10.1190/1.2967551`.

[16] Z.-H. Feng, J. Kittler, M. Awais, P. Huber, and X.-J. Wu. Wing loss for robust facial landmark localisation with convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[17] P. B. Flemings. *Pressure and Stress from Seismic Velocity*, page 142–166. Cambridge University Press, 2021. doi: 10.1017/9781107326309.007.

[18] O. Gramstad and M. Nickel. Automated interpretation of top and base salt using deep-convolutional networks. pages 1956–1960, 08 2018. doi: 10.1190/segam2018-2996306.1.

[19] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition.

In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016. doi: 10.1109/CVPR.2016.90.

[20] D. Karimi and S. E. Salcudean. Reducing the hausdorff distance in medical image segmentation with convolutional neural networks. *IEEE Transactions on Medical Imaging*, 39(2):499–513, 2020. doi: 10.1109/TMI.2019.2930068.

[21] J. Kur, M. Mioduchowska, and A. Kilikowska. Distribution of cyclopoid copepods in different subterranean habitats (southern poland). *Oceanological and Hydrobiological Studies*, 49:255–266, 09 2020. doi: 10.1515/ohs-2020-0023.

[22] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar. Focal loss for dense object detection. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[23] A. Luque, A. Carrasco, A. Martín, and A. de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition*, 91:216–231, 2019. ISSN 0031-3203. doi: https://doi.org/10.1016/j.patcog.2019.02.023. URL `https://www.sciencedirect.com/science/article/pii/S0031320319300950`.

[24] A. Mahendran and A. Vedaldi. Understanding deep image representations by inverting them. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5188–5196, 2015. doi: 10.1109/CVPR.2015.7299155.

[25] D. Martin, C. Fowlkes, D. Tal, and J. Malik. A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 416–423 vol.2, 2001. doi: 10.1109/ICCV.2001.937655.

[26] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 342–347, 2011. doi: 10.1109/ICSIPA.2011.6144164.

[27] A. M. Nguyen, A. Dosovitskiy, J. Yosinski, T. Brox, and J. Clune. Synthesizing the preferred inputs for neurons in neural networks via deep generator networks. *CoRR*, abs/1605.09304, 2016. URL `http://arxiv.org/abs/1605.09304`.

[28] A. Odena, V. Dumoulin, and C. Olah. Deconvolution and checkerboard arti-

facts. *Distill*, 2016. doi: 10.23915/distill.00003. URL `http://distill.pub/2016/deconv-checkerboard`.

[29] C. Olah, A. Mordvintsev, and L. Schubert. Feature visualization. *Distill*, 2(11):e7, 2017.

[30] D. M. W. Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *CoRR*, abs/2010.16061, 2020. URL `https://arxiv.org/abs/2010.16061`.

[31] Z. Qin, F. Yu, C. Liu, and X. Chen. How convolutional neural network see the world - a survey of convolutional neural network visualization methods. *Math. Found. Comput.*, 1:149–180, 2018.

[32] A. R. Reddy, E. V. Prasad, and L. S. S. Reddy. Article: Abnormality detection of brain mr image segmentation using iterative conditional mode algorithm. *International Journal of Applied Information Systems*, 5(2):56–66, January 2013. Published by Foundation of Computer Science, New York, USA.

[33] E. Robein. *Seismic imaging: A review of the techniques, their principles, merits and limitations*. EAGE publications, 2010.

[34] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*, pages 234–241, Cham, 2015. Springer International Publishing. ISBN 978-3-319-24574-4.

[35] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65 6:386–408, 1958.

[36] A. L. Samuel. Some studies in machine learning using the game of checkers. ii—recent progress. *IBM Journal of Research and Development*, 11(6):601–617, 1967. doi: 10.1147/rd.116.0601.

[37] S. Sen, S. Kainkaryam, C. Ong, and A. Sharma. Saltnet: A production-scale deep learning pipeline for automated salt model building. *The Leading Edge*, 39:195–203, 03 2020. doi: 10.1190/tle39030195.1.

[38] J. Shlens. Notes on kullback-leibler divergence and likelihood. *CoRR*, abs/1404.2000, 2014. URL `http://arxiv.org/abs/1404.2000`.

[39] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep inside convolutional networks:

Visualising image classification models and saliency maps. In *In Workshop at International Conference on Learning Representations*, 2014.

[40] R. Unnikrishnan, C. Pantofaru, and M. Hebert. A measure for objective evaluation of image segmentation algorithms. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Workshops*, pages 34–34, 2005. doi: 10.1109/CVPR.2005.390.

[41] B. J. Wythoff. Backpropagation neural networks: A tutorial. *Chemometrics and Intelligent Laboratory Systems*, 18(2):115–155, 1993. ISSN 0169-7439. doi: https://doi.org/10.1016/0169-7439(93)80052-J. URL `https://www.sciencedirect.com/science/article/pii/016974399380052J`.

[42] M. Yeung, E. Sala, C.-B. Schönlieb, and L. Rundo. Unified focal loss: Generalising dice and cross entropy-based losses to handle class imbalanced medical image segmentation. *Computerized Medical Imaging and Graphics*, 95:102026, 2022. ISSN 0895-6111. doi: https://doi.org/10.1016/j.compmedimag.2021.102026. URL `https://www.sciencedirect.com/science/article/pii/S0895611121001750`.

[43] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 818–833, Cham, 2014. Springer International Publishing.

[44] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

List of Figures

List of Tables

Glossary

AM Activation maximization. i, iii, 41–43, 55, 58–60, 69, 70, 77–79, 81, 83, 84, 93

ANN Artificial neural network. 10–12, 91

AUC Area under Curve. 52

BCE Binary Crossentropy. 31, 32, 34, 62, 76

CAM Class Activation Mapping. 20, 84

CIG Common Image Gather. 8, 9, 91

CNN Convolutional neural network. i, 1, 2, 12, 13, 15–20, 23–31, 35, 38, 41–45, 47–50, 52–60, 62, 63, 65–81, 83, 84, 91–95, 97

DIC Dice coefficient. 33, 50, 51, 53, 62, 63, 65, 67, 69, 94

FN false negatives. 64

FP false positives. 64

GAN Generative adversarial network. 84

GAP Global Average Pooling. 13, 20, 56, 58

GCE Global Consistency error. 51

IL Inline. 47, 48, 53, 54, 62, 63, 65–69, 93

IoU Intersection over Union. 36, 38, 50, 51

JAC Jaccard index. 50, 51, 62

KL Kullback-Leibler. 36–38, 93

LR Learning rate. 27, 54, 55, 59, 62, 70, 83

MI Mutual information. 37, 52

ML Machine learning. 9, 10, 91

MSE Mean square error. 38, 39

NI Network inversion. i, iii, 41, 43–45, 60, 61, 69, 71–73, 75, 76, 80–84, 93–95

NMO Normal move out. 8, 9, 91

PS Perimetric similarity. 52, 63, 66, 68, 69, 94

RTM Reverse Time Migration. 6, 7, 16, 92

RV random variable. 37

TNR True negative rate. 51

TPR True positive rate. 51

VS Volumetric similarity. 51, 52

XAI eXplainable AI. i, iii, 2, 15, 18–20, 41, 47, 58, 69, 70, 74, 83, 97