UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Fraunhofer IAIS

Understanding Convolutional Neural Networks for Seismic Features by Visualization

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

André von Landenberg

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Dr. rer. nat. Stefan Rilling (Fraunhofer IAIS)

Koblenz, im Januar 2019

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Acknowledgements

I would first like to thank my supervisors Prof. Dr. Stefan Müller and Dr. Stefan Rilling for their constant support and advice in writing this thesis. Also, my thanks goes to my second supervisor at Fraunhofer IAIS, Dr. Manfred Bogen, who gave me invaluable feedback and motivation.

Then, I would like to thank my friends and colleagues at Fraunhofer IAIS for their precious ideas and help. I am especially thankful to the Fraunhofer deep learning experts and the members of the VRGeo Consortium who gave helpful feedback and advice to my work.

Finally, I want to thank my girlfriend, my family and my friends for their ongoing support in the last months.

Without them, this thesis would not have been possible.

Abstract

In recent years, Convolutional Neural Networks (CNNs), that are most commonly applied to visual image analysis, have yielded astonishing results in a number of classification tasks. The areas of application include the recognition of text, speech and various objects and structures. However, the various activities inside of CNNs and their reactions to specific inputs remain unclear. For a better understanding of processes within a CNN, correlations between neurons and layers have to be visualized to be able to compare the reaction to different inputs afterwards. There are various approaches to visualize activities in CNNs in order to strengthen a user's confidence in the classification results by offering a look inside the networks that usually work like black-boxes.

In this thesis, five visualization methods (Activation Maximization, Deconvolution, Guided Backpropagation, Grad-CAM and Guided Grad-CAM) are applied to CNN architectures that are trained to detect seismic features in geological data. By visualizing features and other information from these CNNs, it is possible to gain knowledge about the decision-making process in the network and even derive optimization possibilities.

Zusammenfassung

In den letzten Jahren haben Convolutional Neural Networks (CNNs) erstaunliche Ergebnisse bei einer Reihe von Klassifikationsaufgaben erzielt. Die Anwendungsgebiete umfassen die Erkennung von Text, Sprache und verschiedenen Objekten und Strukturen. Die Aktivitäten innerhalb der CNNs und ihre Reaktionen auf bestimmte Eingaben bleiben jedoch unklar. Zum besseren Verständnis der Vorgänge innerhalb eines CNN können Zusammenhänge zwischen Neuronen und Schichten visualisiert und die Reaktion auf verschiedene Eingaben verglichen werden. Es gibt verschiedene Ansätze, die versuchen, Aktivitäten in CNNs zu visualisieren, um das Vertrauen in die Klassifikationsergebnisse zu stärken, indem sie einen Blick in die Netzwerke werfen, die normalerweise wie Black-Boxes funktionieren.

In dieser Arbeit werden fünf Visualisierungsmethoden (Activation Maximization, Deconvolution, Guided Backpropagation, Grad-CAM und Guided Grad-CAM) auf CNN-Architekturen angewendet, die darauf trainiert wurden, seismische Merkmale in geologischen Daten zu erkennen. Durch die Visualisierung von Features und anderen Informationen aus diesen CNNs ist es möglich, Erkenntnisse über den Entscheidungsprozess im Netzwerk zu gewinnen und sogar Optimierungsmöglichkeiten daraus abzuleiten.

Contents

1 Introduction

Working with huge amounts of data is a challenging task for humans and computers. Especially in the oil and gas industry, datasets from seismic surveys with a size of several hundred gigabytes[1] are collected and analyzed by geologists and geophysicists in order to eventually find promising locations of oil and gas reservoirs. To find possible hydrocarbon deposits, structures in the subsurface of the earth have to be interpreted by domain experts in a difficult and time-consuming process.

Before computers had become common in the industry, the workflow comprised printing out acquired volume datasets slice by slice on sheets of paper and interpreting every slice by hand. With modern technology, the procedure was revolutionized so that nowadays sophisticated software programs like OpenDTect[1] and Petrel[2] are used by domain experts in the interpretation workflow. The experts interpret the volumetric data with user-driven classification techniques that enable them to find and isolate important geological structures in a fraction of time. For example, a variety of seismic attributes helps the interpreters to distinguish seismic features from unimportant data and identified regions can be visualized through a volume rendering approach in order to see the isolated three-dimensional structures.

As user-driven classification techniques still require a lot of manual interaction, they are time-consuming and need highly experienced experts to deliver useful results. Therefore, one focus of research is the automatic detection and classification of important seismic structures to improve the interpretation workflow of oil and gas experts. The VRGeo Consortium[3] is an alliance of members from the oil and gas industry and research institutes like Fraunhofer IAIS that was founded in 1998 to provide innovation in the industry and since then is hosted by Fraunhofer IAIS or its predecessor GMD (Gesellschaft für Mathematik und Datenverarbeitung). First, innovative user-driven classification techniques like the intuitive creation of fault geometries[3] or the use of multi-dimensional transfer functions[4] for volumetric data visualization have been developed at Fraunhofer IAIS as part of the work in the VRGeo Consortium. Especially, the introduction of semi-automatic detection of anomalies in local histograms[5] enabled domain experts to concentrate on anomalous regions in the seismic datasets that are most likely to contain important structures, as the approach highlights these regions and therefore ignores irrelevant regions. As this method is still not fully-automated, the use of deep learning came into focus.

[1] `https://www.dgbes.com`
[2] `https://www.software.slb.com/products/petrel`
[3] `https://www.vrgeo.org`

In recent years, artificial intelligence in general and deep learning in particular have gained popularity not only in computer science research and large technology companies, but also in the oil and gas industry. Deep learning approaches offer the possibility to analyze and classify large amounts of data automatically in a fraction of time that was needed for a manual analysis. Thus, Convolutional Neural Networks (CNNs) are used to classify important geological structures that are mandatory to find hydrocarbon deposits. First results of geological structure detection with CNNs[15] in the VRGeo Consortium were very promising and lately the DeepGeo[4] framework has been introduced to easily execute distributed deep learning tasks from a web interface. Nevertheless, one disadvantage of CNNs is their black-box property, making it difficult to understand their decisions and classification results as most of the complex processes are hidden from the user.

Therefore, this thesis will introduce several methods to visualize a network's features in multiple layers and reveal concepts learned by the network in order to better understand the network's operations and take a look into the black-box. The key to proper automatic classification with neural networks is the use of hybrid AI which describes a combination of artificial intelligence (e.g. CNNs) with human knowledge to get the best of both worlds. While neural networks can deal with large amounts of monotonous data and learn to classify data based on training examples, humans can steer and monitor the automatic process with contribution of their domain knowledge. As part of the VRGeo project, this thesis contributes to a better understanding of CNNs in the context of seismic interpretation and shows optimization possibilities for automatic classification approaches in the oil and gas exploration workflow.

This thesis is structured as follows. In chapter 2, the process of data acquisition and seismic interpretation will be explained to understand the geological background. Chapter 3 will introduce related work about neural networks and take a deeper look into CNNs, that are especially useful for classifying image data. As the focus of this thesis lies on understanding CNNs by visualization, several visualization methods like Activation Maximization or Deconvolution will be discussed in chapter 4. Subsequently, the implementation of these methods into the Fraunhofer's DeepGeo framework will be explained in chapter 5. Finally, the implemented methods will be evaluated in detail in chapter 6 by applying them to CNNs trained on seismic datasets and comparing the visualization results. Also in this chapter, optimization possibilities for the CNNs and the training data will be discussed. At the end, the results will be summarized in chapter 7 and an outlook into the future of CNN visualization will be given.

[4] `https://www.vrgeo.org/index.php?id=639`

2 Geological Background

The exploration of the earth's crust in order to find the location of oil and gas reservoirs is a time-consuming process in which huge amounts of data have to be collected and analyzed by geophysicists and geologists to generate detailed models of the earth's subsurface and to eventually predict the location of hydrocarbon deposits.

In this chapter, the geological background and seismic interpretation will be briefly discussed in order to understand the value of applying Convolutional Neural Networks to classify seismic structures and features.

2.1 Data Acquisition

To be able to predict the location of oil and gas reservoirs, geophysicists have to know which rock layers and structures lie beneath the surface of the earth to conclude where hydrocarbons might be trapped. The most common method of obtaining subsurface data is the use of seismic reflection. Similar to an ultrasound image used by a doctor to see an embryo inside a mother's body, the echo of strong sound or shock waves are measured to deduce the composition of different rock layers in the subsurface. On land, sound waves are sent into the ground from a specifically built vehicle, called vibrator truck. The reflections of these waves are captured by a recording vehicle that carries a set of microphones called seismometers or geophones (see Figure 1). At every transition from one layer to the next, an echo can be measured by the geophones.

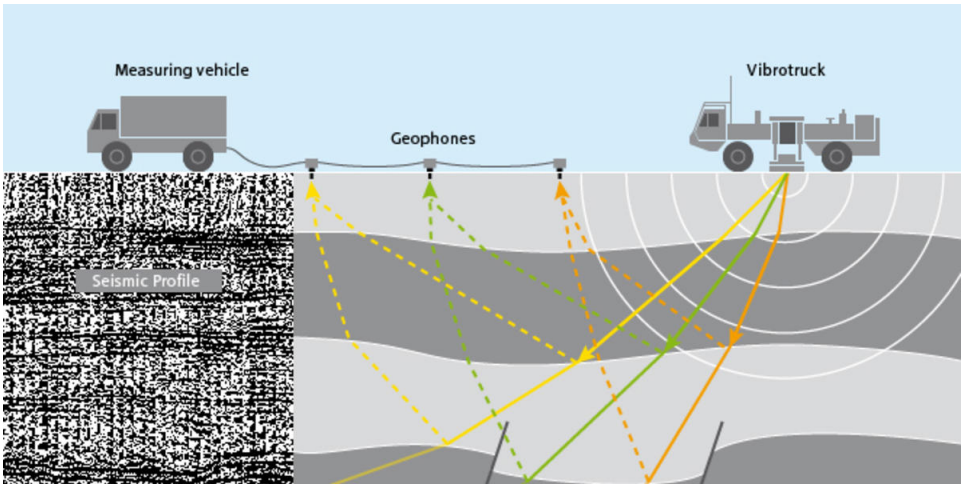


Figure 1: Data acquisition by seismic reflection on land [2]

If the area of exploration lies beneath the sea, the vibrator vehicle is replaced by a ship that uses airpulses as seismic source. The ship carries a line of seismometers that are called hydrophones in this case. These hydrophones record the echos of the pulses (see Figure 2), while being positioned on a long line that can have a length of several kilometers.

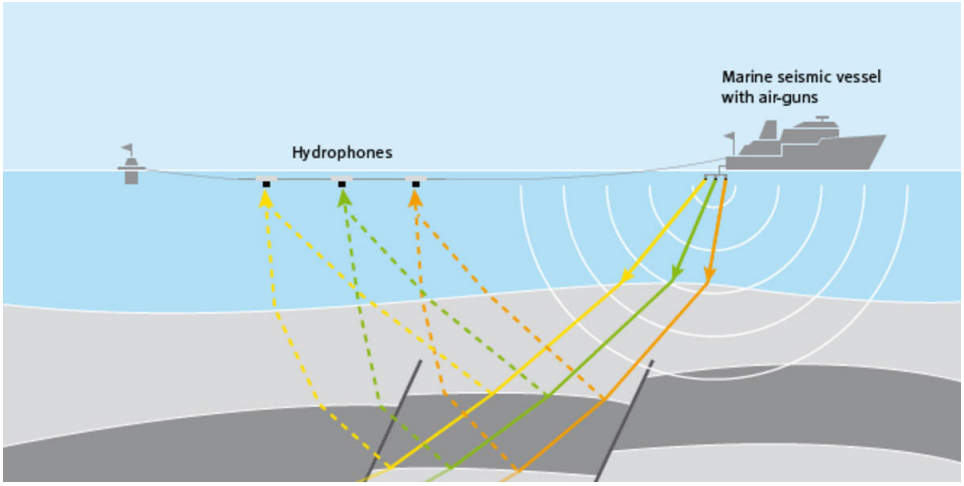
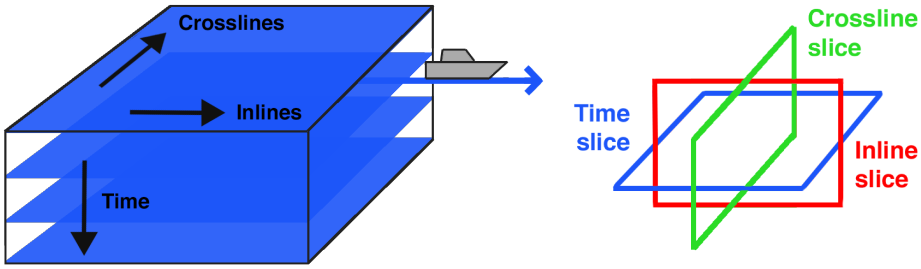


Figure 2: Data acquisition by seismic reflection on sea [2]

Apart from seismic exploration, there are other methods to create models of the subsurface using gravity, magnetism, electromagnetic waves or even radioactive minerals [17]. The use of seismic reflection still resembles the most popular method, as it is very environment-friendly and delivers high-quality results.

2.2 Seismic Interpretation

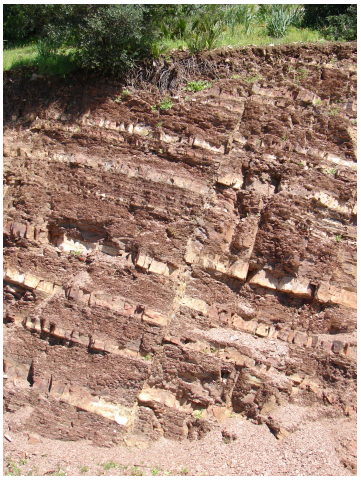
After the acquisition of seismic data, all the information is processed e.g. to filter out noise and combined with information from other sources to eventually build a three-dimensional model of the explored subsurface area. To visualize this model, the three-dimensional cube is viewed slice by slice. In seismic interpretation, there are three kinds of slices. The inline direction is always parallel to the direction in which the data was acquired. Figure 3a shows the collection of data on sea where the inline thus corresponds to the direction in which the ship is towing the hydrophones. The crossline is perpendicular to this direction while the time is measured downwards into the ground. Eventually, the resulting 3D volume can be viewed either as inline, crossline or as time slices, as depicted in figure 3b.



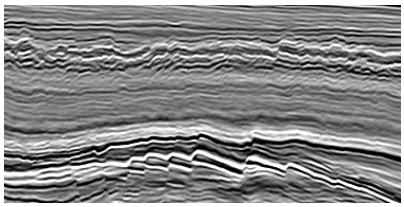
(a) **(b)**

Figure 3: Directions of inline, crossline and time slices in a 3D survey

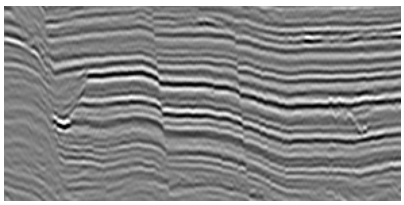
For a better visibility of seismic features and structures in the subsurface area, a set of seismic attributes can be derived or extracted from the gathered seismic data. This set includes attributes like the measured time or amplitude but also frequency, similarity or dip among many others [17, 8]. With these attributes, experienced geophysicists or geologists interpret this model in order to find structures that form a so-called trap for hydrocarbons. The Schlumberger Oilfield Glossary [29] defines a trap as a "configuration of rocks suitable for containing hydrocarbons and sealed by a relatively impermeable formation through which hydrocarbons will not migrate"[5]. These structures block the upward migration of hydrocarbons and indicate the location of an oil or gas reservoir nearby.



(a) A fault in Morocco [6]



(b) F3 dataset (inline)[7]



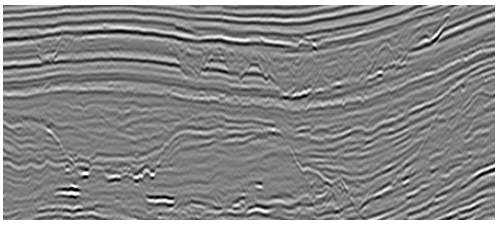
(c) Parihaka dataset (crossline)[8]

Figure 4: Pictures of faults from subsurface exploration (right) and a fault that came to the surface in Morocco (left)

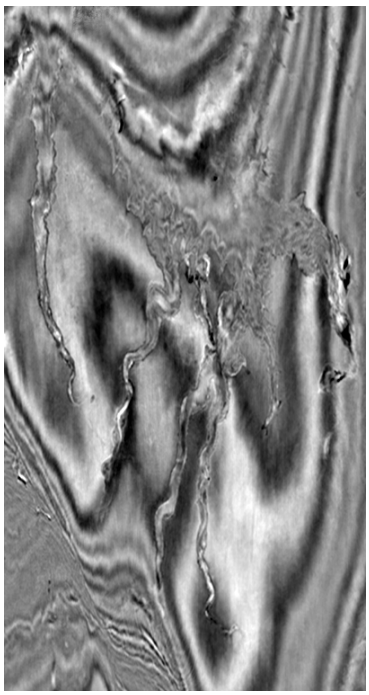
[5] https://www.glossary.oilfield.slb.com/Terms/t/trap.aspx

The most common of these structures are faults. According to the Schlumberger Oilfield Glossary [29] a fault is "a break or planar surface in brittle rock across which there is observable displacement"[6] like the displacements clearly visible in Figure 4 for faults in Morocco and the datasets F3[7] and Parihaka[8]. Faults are the result of rock-mass movements in the subsurface and can cause earthquakes if they lock in regions of high friction and suddenly release the built-up energy in form of seismic waves. Depending on the fault's nature some faults "can act as a conduit for migrating oil or gas"[6] whereas other faults "can act as a fault seal"[6] that prevents hydrocarbons from migrating.

(a) Waimakariri River, New Zealand [23]



(b) Parihaka dataset (crossline)



(c) Parihaka dataset (timeslice)

Figure 5: Pictures of a river in New Zealand causing a channel structure (a) and channels in Parihaka dataset from crossline (b) and timeslice (c)

Another important structure visible in seismic volume data is called channel. A channel is "a linear, commonly concave-based depression through which water and sediment flow and into which sediment can be deposited in distinctive, often elongated bodies. [...] The close proximity of coarse-grained and fine-grained sediments can ultimately lead to the formation

[6] `https://www.glossary.oilfield.slb.com/Terms/f/fault.aspx`
[7] F3 dataset covers a Dutch part of the North Sea and is publicly available [7]
[8] Parihaka dataset covers an area of New Zealand's coast and is publicly available [21]

of stratigraphic hydrocarbon traps".[9] These structures result from former rivers and streams which have eroded the rock beneath them. Figure 5a shows the Waimakariri River in New Zealand that has a meandering shape which is a typical indicator for a channel in subsurface volume data (see Figure 5b, 5c).

Another trap for hydrocarbons is a salt dome. The Oilfield Glossary [29] defines this as "a mushroom-shaped or plug-shaped diapir made of salt, commonly having an overlying cap rock. [...] Hydrocarbons are commonly found around salt domes because of the abundance and variety of traps created by salt movement and the association with evaporite minerals that can provide excellent sealing capabilities".[10] Thus, salt domes can be recognized in seismic volume data by their dome-shape (see Figure 6).

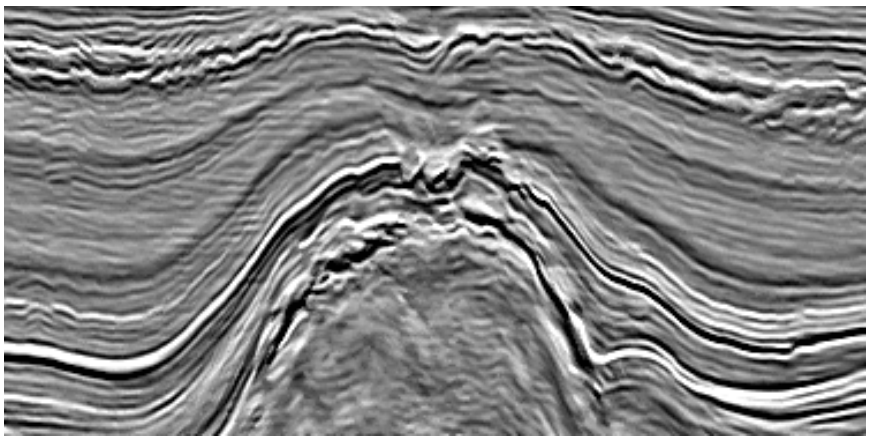


Figure 6: Salt Dome in F3 dataset

Eventually, with the help of several interpretation iterations, the additional use of well logs and geological domain knowledge, geophysicists and geologists can predict the location of oil and gas reservoirs with high confidence.

[9] https://www.glossary.oilfield.slb.com/Terms/c/channel.aspx
[10] https://www.glossary.oilfield.slb.com/Terms/s/salt_dome.aspx

3 Related Work

Neural Networks are used in a variety of tasks in research and industry such as image classification, speech and face recognition or the automatic analysis of huge amounts of data. For image data the use of CNNs has proven to deliver outstanding results in several applications. This chapter will give an overview on related work in the field of deep learning and artificial intelligence including different kinds of neural networks.

3.1 Neural Networks

The term *artificial intelligence* is widely used today and in most cases describes a computer system that has some of the abilities of the human mind, such as the ability to recognize images or languages. An *intelligent* system is able to learn how to solve its specified task on its own. In order to achieve this intelligence, scientists analyzed the human brain and tried to simulate processes inside of it by forming a neural network.

A simple artificial neuron, called perceptron, was already introduced by Rosenblatt in 1961 [27] inspired by earlier work of McCulloch and Pitts in 1941 [20]. A perceptron receives several binary inputs $x_1, x_2, ..., x_n$ and produces a single binary output a (see Figure 7).

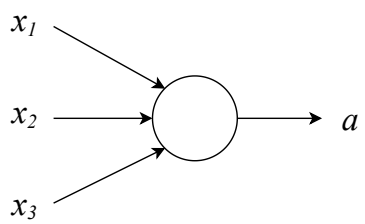


Figure 7: Representation of a single perceptron

The output a is calculated by multiplying each input x_i with a corresponding weight $w_i \in \mathbb{R}$. Then, a bias value $b \in \mathbb{R}$ is subtracted from the sum of all pairs. If the result is less or equal 0 the output will be 0, otherwise it will be 1 (see Equation 1).

$$a = \begin{cases} 0 & \text{if} \sum_i x_i w_i + b \leq 0 \\ 1 & \text{if} \sum_i x_i w_i + b > 0 \end{cases} \tag{1}$$

To overcome the need of having binary inputs, the perceptron can be extended to a sigmoid neuron. Sigmoid neurons have the ability to work

with real input values and furthermore calculate real output values instead of binary ones. The final output of a sigmoid neuron is defined as:

$$a = \sigma\left(\sum_i x_i w_i + b\right) \tag{2}$$

where $\sigma(z)$ depicts the sigmoid function:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}} \tag{3}$$

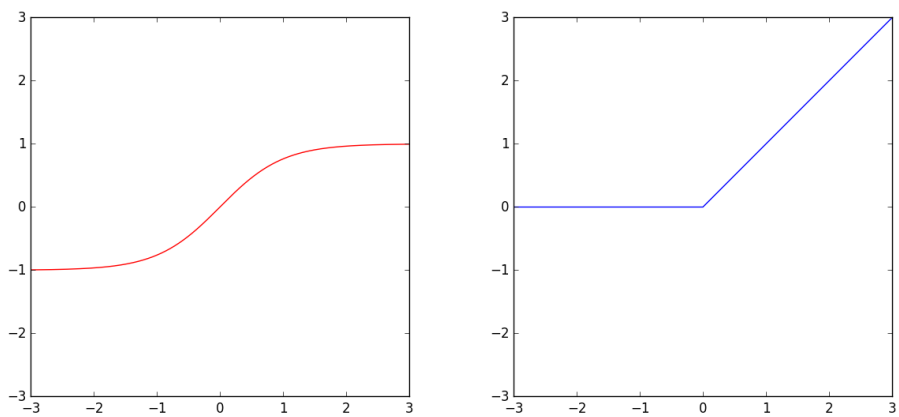
Sigmoid neurons have the advantage that small changes in their parameters only lead to small changes in the output, in contrast to perceptrons, where small changes in parameters may lead to a complete change of the output. In literature, sometimes σ is also called logistic function and this kind of neuron logistic neuron. In modern neural network architectures the sigmoid function often is replaced by the $tanh$ function defined as:

$$\sigma(z) = tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{4}$$

or the rectified linear unit (ReLU) which is defined as:

$$\sigma(z) = max(0, z) \tag{5}$$

Graphs of both functions are depicted in figure 8.



(a) tanh **(b)** ReLU

Figure 8: Comparison of *tanh* and *ReLU* functions

The advantage of the ReLU function is a significant decrease of training time, which makes it especially useful for large CNN architectures (see chapter 3.2). With the use of weights, biases and an activation function such as ReLU, a single neuron in a neural networks now looks as depicted in figure 9.

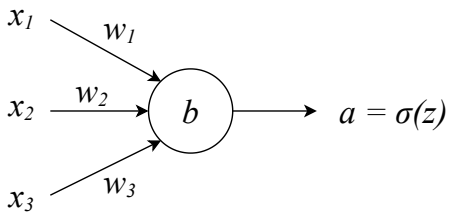


Figure 9: Neuron with weights, bias and activation function

A network of many neurons arranged in a number of layers can make more complex decisions compared to a single neuron. Figure 10 shows a network consisting of four layers. The left layer is called input layer and contains all input values. The right layer is the output layer and outputs the final decision of the network. All layers in between are named hidden layers and use the results of the entire neurons from the previous layer to make decisions. The advantage of a many-layer network is that neurons in the second layer make their decisions based on the output of the first layer neurons. Thus, with every layer the decisions are made on a more complex and abstract level. Networks that pass the output of a layer to the next layer without any loops are termed feed-forward networks. This kind of network is called multi-layer perceptrons (MLP), although it actually consists of sigmoid neurons.

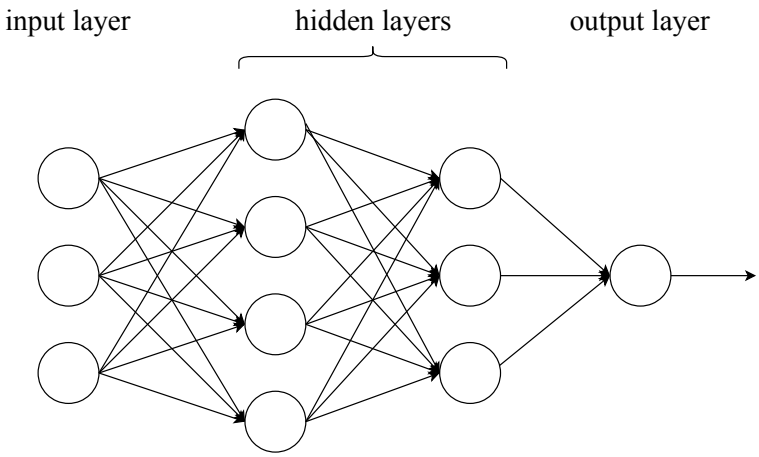


Figure 10: Multi-layer perceptrons (MLP)

3.1.1 Gradient Descent

In order to find weights and biases that lead to desired outputs, a training process with labelled training data is performed. An example input is passed to the first layer of which the wanted output is already familiar. Then, the actual output is compared to the wanted output and the network's parameters can be adapted to better fit the desired result. The improvement can be quantified by a cost function (also called loss or objective function) that can be defined as:

$$C(w, b) \equiv \frac{1}{2} \sum_{x} \|y(x) - a\|^2 \tag{6}$$

In equation 6, w and b denote the weights and biases in the network, a is a vector denoting all outputs of the network and $y(x)$ corresponds to the correct label of input x. This quadratic cost function becomes less when $y(x)$ is close to a which is the desired result after training the network.

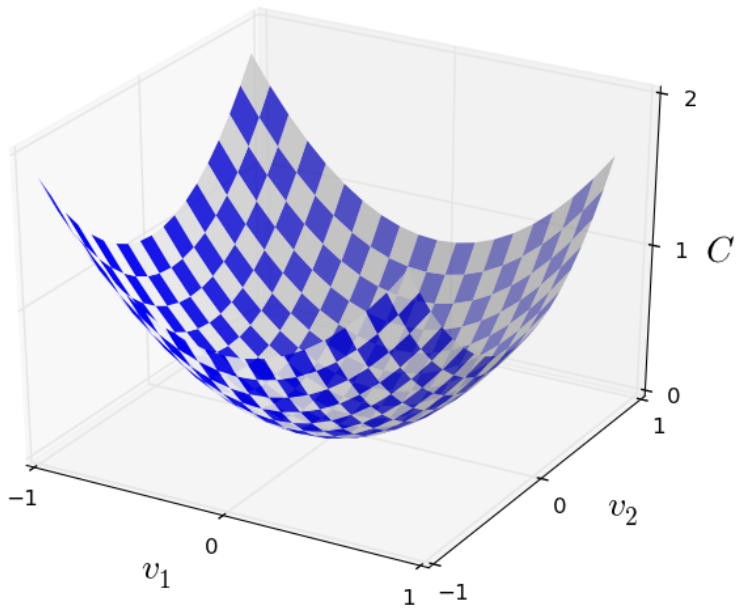


Figure 11: Visualization of a quadratic cost function with parameters v_1, v_2 [22]

In order to find parameters w and b such that $C(w, b)$ gets minimal, the gradient descent technique can be applied. The goal of gradient descent is to find the global minimum of the cost function. The analytical approach of solving this problem via the derivatives of C turns out to be far too complex for a huge number of parameters. Another approach is to imagine

the quadratic cost function as a valley (see figure 11) and placing a marble somewhere on the edge of this valley. Then, the marble will follow the direction of the steepest descent and will eventually end in its global minimum. In general, the cost function can be seen as a real-valued function $C(v)$ with an arbitrary number of parameters in a vector v. For simplicity, we look at only two parameters in this example that we call v_1, v_2. Thus, the change of the marble's position can be described as:

$$\Delta C \approx \frac{\partial C}{\partial v_1}\Delta v_1 + \frac{\partial C}{\partial v_2}\Delta v_2 \tag{7}$$

We also define the vector Δv as the vector of changes in v and the gradient of C as the vector of partial derivatives:

$$\Delta v \equiv (\Delta v_1, \Delta v_2)^T \tag{8}$$

$$\nabla C \equiv \left(\frac{\partial C}{\partial v_1}, \frac{\partial C}{\partial v_2}\right)^T \tag{9}$$

With these definitions, equation 7 can be rewritten as:

$$\Delta C \approx \nabla C \cdot \Delta v \tag{10}$$

In order to decrease the overall cost C, the vector Δv needs to be updated with proper values in every training iteration. To achieve that, we choose a small positive parameter η, which is commonly known as learning rate, and move along the gradient of C:

$$\Delta v = -\eta \nabla C \tag{11}$$

Inserted into equation 10, we get:

$$\Delta C \approx -\eta \nabla C \cdot \nabla C = -\eta \left\|\nabla C\right\|^2 \tag{12}$$

With $\|\nabla C\|^2$ and η always being positive, the change of the cost function C is therefore always negative which guarantees a decrease of cost with every training iteration.

Replacing the parameters v_1, v_2 with the neural network parameters w and b again, the rule to update every weight w_k and bias b_l is defined as follows:

$$w_j \rightarrow w'_j = w_j - \eta \frac{\partial C}{\partial w_j} \tag{13}$$

$$b_j \rightarrow b'_j = b_j - \eta \frac{\partial C}{\partial b_j} \tag{14}$$

As the gradient descent process is expensive and time-consuming, the idea of stochastic gradient descent (SGD) [26] came up to speed up the gradient calculation. This is done by calculating ∇C only from a small batch of randomly chosen inputs and repeating this step until all input samples have been consumed. With SGD, the gradient is only approximated, but the overall direction is still correct. Thus, the training can be completed in a fraction of time compared to normal gradient descent but with similar results, which makes SGD preferable.

3.1.2 Backpropagation

In the previous section the learning process of a neural network has been described using the gradient descent technique. However, the decrease of the cost function C relies on the calculation of $\frac{\partial C}{\partial w}$ and $\frac{\partial C}{\partial b}$ which is not trivial. Therefore, the Backpropagation algorithm [28] has been proposed for fast computation of these gradients.

This algorithm is about measuring how much the alteration of a weight or bias changes the output of the cost function. In the following notation, z_j^l describes the value z for the jth neuron in the lth layer, defined as:

$$z_j^l \equiv \sum_k w_{jk}^l a_k^{l-1} + b_j^l \tag{15}$$

Here, w_{jk}^l denotes the weight of the connection between the kth neuron in layer $l-1$ and the jth neuron in layer l.

Then, we imagine that a small change Δz_j^l is added to a neurons weighted input which causes the overall cost to change by $\frac{\partial C}{\partial z_j^l} \Delta z_j^l$. Thus, the error δ_j^l in the jth neuron in the lth layer is defined as:

$$\delta_j^l \equiv \frac{\partial C}{\partial z_j^l} \tag{16}$$

For the last layer L the error is given as:

$$\delta_j^L = \frac{\partial C}{\partial a_j^L} \sigma'(z_j^L) \tag{17}$$

On the right hand side of this equation, the first term $\partial C / \partial a_j^L$ describes how fast the cost is changing depending on the activation of the jth neuron. The second term $\sigma'(z_j^L)$ is the derivative of the activation function of the jth neuron and measures the change in σ at position z_j^L. Notice that $a_j^l = \sigma(z_j^l)$.

The term $\partial C / \partial a_j^L$ can be computed, depending on the cost function. For example, for the quadratic cost function $C = \frac{1}{2} \sum_j (y_j - a_j^L)^2$ the partial derivative equals:

$$\frac{\partial C}{\partial a_j^L} = a_j^L - y_j \tag{18}$$

To consider all neurons of one layer, δ^L can be defined in a matrix-based notation as:

$$\delta^L = \nabla_a C \odot \sigma'(z^L) \tag{19}$$

$$= (a^L - y) \odot \sigma'(z^L) \tag{20}$$

with $\odot$ being the element-wise product, also called Hadamard product.

From here on, we can use δ^L to compute the errors in the previous layers by passing the already computed errors backwards. The error δ^l in the lth layer is then calculated by:

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L) \tag{21}$$

The term $(w^{l+1})^T \delta^{l+1}$ weights the known error δ^{l+1} from the next layer according to each neurons weight in layer l, while the element-wise product $\odot \sigma'(z^L)$ passes this error backwards through the activation function.

In order to pass the errors through the network and calculate gradients of the cost function with respect to biases or weights, the chain rule of calculus has to be applied. For any variable z depending on y, which itself depends on x, the chain rule states:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x} \tag{22}$$

With the help of the chain rule from equation 22 and the equations 15 and 16, we can derive the change of the cost function with respect to a bias b_j^l in the network:

$$\frac{\partial C}{\partial b_j^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial b_j^l} = \delta_j^l \tag{23}$$

Thus, $\partial C / \partial b_j^l$ turns out to be exactly the error δ_j^l of the neuron the bias belongs to. With the same equations, the change of the cost function with respect to a weight w_{jk}^l is derived as:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l} \cdot \frac{\partial z_j^l}{\partial w_{jk}^l} = \delta_j^l \cdot a_k^{l-1} \tag{24}$$

In summary, the four fundamental equations for the Backpropagation algorithm are:

$$\delta^L = (a^L - y) \odot \sigma'(z^L) \tag{20}$$

$$\delta^l = ((w^{l+1})^T \delta^{l+1}) \odot \sigma'(z^L) \tag{21}$$

$$\frac{\partial C}{\partial b_j^l} = \delta_j^l \tag{23}$$

$$\frac{\partial C}{\partial w_{jk}^l} = \delta_j^l \cdot a_k^{l-1} \tag{24}$$

With these equations it is now possible to compute the gradient of the cost function in order to change the network parameters w and b such that the overall cost is decreased. The learning process consists of:

1. Passing input data forward through the network to obtain values from the output layer a^L
2. Performing the Backpropagation algorithm
   - Computing the error δ^L of the output layer with equation 20
   - Backpropagating the error for each $l = L-1, L-2, ..., 2$ with equation 21
   - Calculating the gradients of the C with respect to w^l and b^l with equations 23 and 24

3. Performing the gradient descent method by updating the parameters w^l and b^l according to equations 13 and 14

3.2 Convolutional Neural Networks

Multilayer perceptrons (MLP) have very limited capabilities concerning image classification tasks because they do not scale well with image size and they lack the ability to detect specific structures or patterns in images. Convolutional neural networks improve MLP by adding several new components like convolution and max pooling layers that take spatial structure of images into account.

3.2.1 Biological Inspiration

In fact, convolutional neural networks are inspired by the visual cortex of animals and humans. Hubel and Wiesel [13, 14] conducted experiments on the visual cortex of anaesthetized cats by stimulating visual cells with lights in different shapes, orientations and intensities and measuring the firing of the neurons.

They found out that visual cells only react to changes in a limited region on the retina, which Hubel and Wiesel called a neuron's receptive field. These receptive fields can be of a simple or a complex type. The highest reaction of neurons was achieved with shapes of narrow long rectangles and straight-line border between regions of different brightness depending on the orientation of these shapes. While the reaction of simple and complex fields did not differ concerning different shapes, the complex fields are usually larger than the simple fields and react to shapes independently of their position in the receptive field.

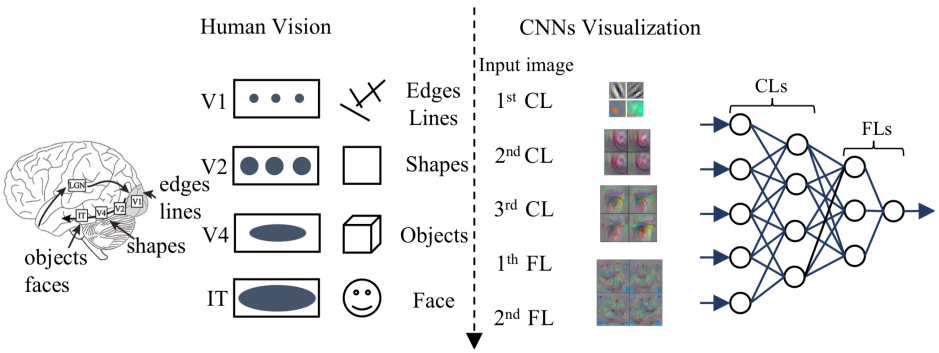


Figure 12: Human vision compared to CNN [25]

The study also showed that similar cells are spatially close to each other and are organized hierarchically in layers, with simple cells in lower layers and complex cells in higher layers. This structure offers the ability to react to increasingly abstract and generalized information. Figure 12 shows how the shapes and concepts detected by CNNs (right) resemble the levels of abstract concepts in the human vision (left).

3.2.2 Architecture

There are three basic ideas that convolutional neural networks use to improve their results in image-related tasks. The first idea is the use of local receptive fields inspired by the visual cortex of animals and humans (see chapter 3.2.1). Instead of connecting every neuron from the previous layer with every neuron of the current layer, there are convolutional layers in CNNs that only connect a small localized region. As depicted in figure 13, a small field of neurons from the input layer (left) is used to obtain the output of one neuron in the first hidden layer (right). In this case the input layer has a size of 28×28 with a receptive field of 5×5 which results in a 24×24 output for the first hidden layer given that the offset (stride) for the field is 1.

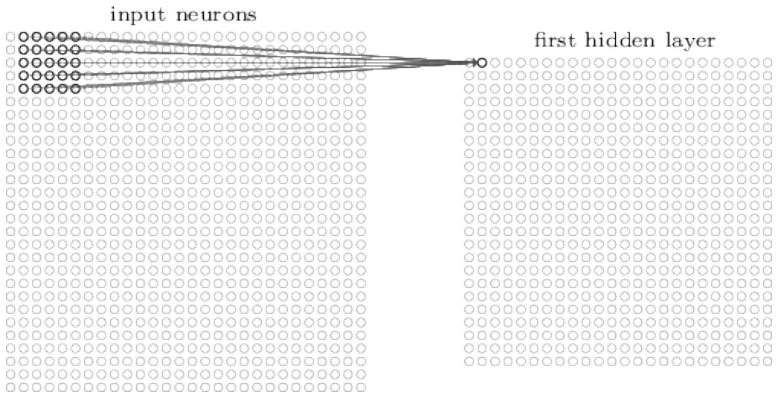


Figure 13: Receptive field in CNN [22]

The second idea is the use of shared weights and biases. Instead of using separate weights and biases for each neuron, in a convolutional layer the same bias and weights are used for every neuron of the output. For a 5×5 receptive field the output of the activation at position (j, k) is:

$$a^i_{j,k} = \sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a^{i-1}_{j+l,k+m}\right) \tag{25}$$

where σ denotes the activation function, e.g. the sigmoid or the ReLU function (see chapter 3.1). Equation 25 shows a convolution operation which can also be written as $a^1 = \sigma(b + w * a^0)$, where a^1 denotes the output activation function, a^0 the input activation function and $*$ the convolution operator.

Every neuron in the first hidden layer thus detects exactly the same feature, a straight line for example. The mapping from one layer to the next layer is called feature map and is usually used several times per convolutional layer such that one layer can detect several features. Figure 14 shows 20 feature maps of a CNN's convolutional layer that is able to detect handwritten digits. These feature maps are also referred to as kernels or filters and basically show the weights of the 5×5 local receptive field and the features it responds to.

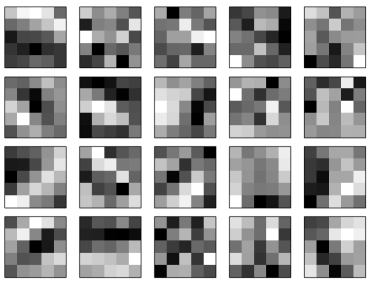


Figure 14: Feature maps in a convolutional layer [22]

The third idea is the use of pooling layers that immediately follow a convolutional layer. A pooling layer condenses feature maps by consecutively summarizing small local regions. The definition of the pooling kernel defines the size of the pooling output. Figure 15 shows the 24×24 output feature map of a hidden layer (left) and the resulting pooling output (right) with a size of 12×12 produced by a 2×2 pooling kernel.

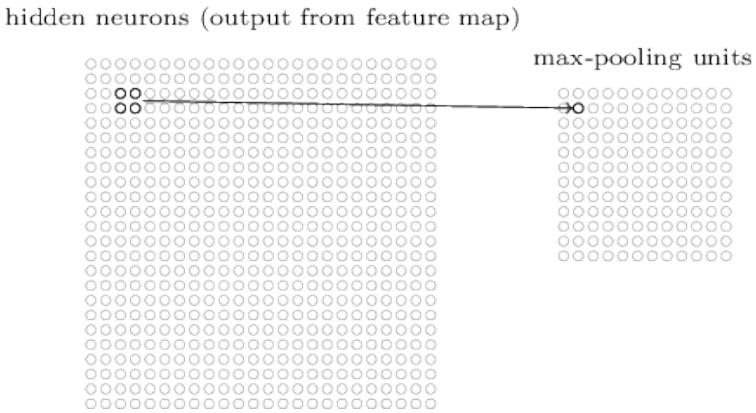


Figure 15: Max Pooling of a feature map [22]

The most commonly used type of pooling is called max pooling in which only the maximum value inside the kernel is used as output. Another type of pooling is average pooling where the average over all values inside the pooling kernel is used as output. The pooling technique decreases the number of parameters inside a convolutional neural network while keeping the relevant information about features in the input image.

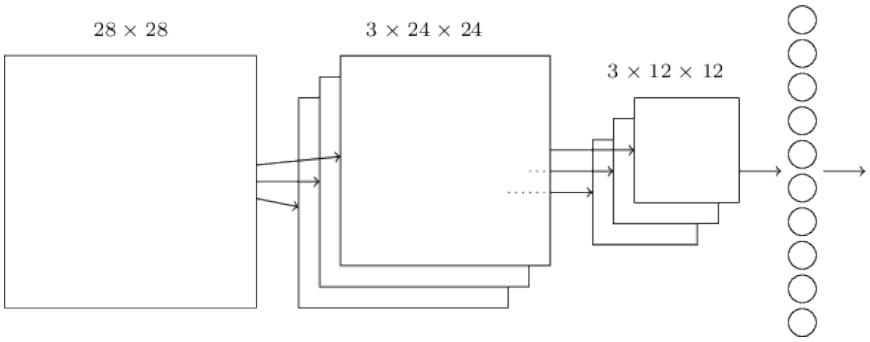


Figure 16: Structure of a common CNN [22]

In conclusion, the structure of a common convolutional neural network consists of one or more convolutional layers each followed by a pooling layer and finally processed by one or more fully-connected layers. In figure 16 the examples of this section are combined to show the general structure of a convolutional neural network. A 28×28 input image is processed by a convolutional layer with 3 kernels, resulting in 3 different 24×24 feature maps. These feature maps are then passed to a max pooling layer which summarizes them to a size of 12×12. Finally, these layers are followed by a standard fully-connected network which produces the final output of the network to classify the input image.

3.2.3 Recent Network Architectures

One of the first CNN architectures introduced by LeCun et al in 1998 is called LeNet [19]. This network is able to recognize hand-written digits and consists of seven layers excluding the input layer (see figure 17). The first four layers perform two convolutions each followed by a subsampling layer. The last three layers consist of a third convolution and two fully-connected layers which calculate the network's final prediction.

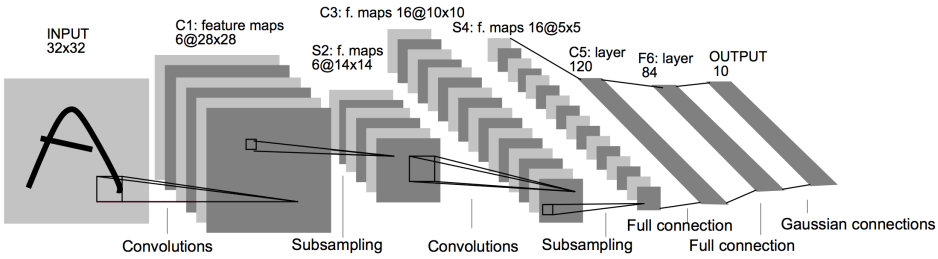


Figure 17: Architecture of LeNet [19]

In 2012, AlexNet [18] was proposed which has a very similar architecture to LeNet but is deeper in terms of the number of filters per layer. AlexNet consists of five convolutional layers followed by three fully-connected layers and uses techniques like max pooling, dropout, data augmentation and a ReLU activation function instead of the tanh or sigmoid function. It was the first CNN to reduce the top-5 error from 26% to 15.3% in the ImageNet Large Scale Visual Recognition Challenge[11] (ILSVRC).

The so-called GoogleNet [34] also known as Inception V1 was presented by Google in 2014. GoogleNet was able to achieve a top-5 error of 6.67% in ILSVRC 2014 which is close to human performance and earned them the first place in the competition. The network architecture is inspired by LeNet but uses 22 layers. Through optimization techniques like batch normalization, image distortion and RMSprop, Google was able to drastically reduce the number of parameters from 60 million for AlexNet to four million. The second place in ILSVRC 2014 was taken by VGG [32] which was developed by Simonyan and Zisserman. The VGG network architecture uses 16 convolutional layers with many filters which ends up in 138 million parameters.

The Inception V1 CNN was revised several times, resulting in Inception V3 [35] in 2016. This version of the Inception network achieved a top-5 error rate of 3.58% on the ILSVRC classification benchmark which is almost a

[11] `http://image-net.org/challenges/LSVRC/`

decrease in error rate by a factor of two compared to Inception V1. Figure 18 shows the architecture of Inception V3 which increased the number of layers from 22 to 48.

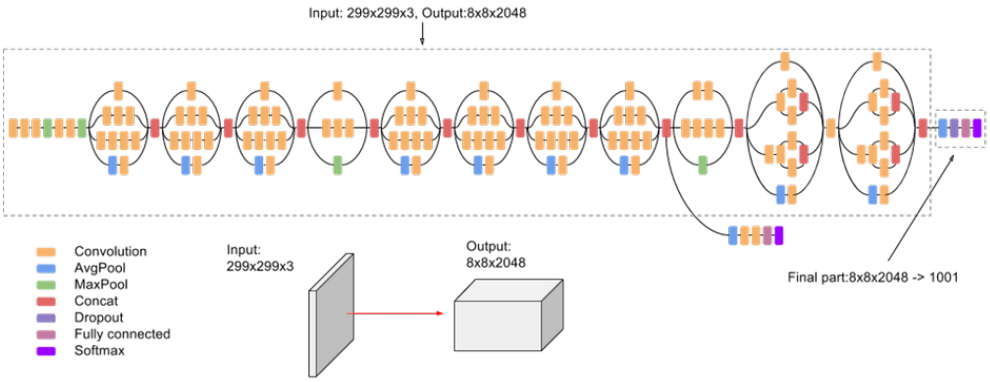


Figure 18: Architecture of Inception V3. Source: `https://cloud.google.com/tpu/docs/inception-v3-advanced`

One year later in ILSVRC 2015, He et al presented the so-called Residual Neural Network (ResNet) [11] that uses up to 152 layers but still has a lower complexity than VGG. To be able to train a network of this size the authors used shortcut connections that enable the network to skip one or more layers. Using this architecture achieved a top-5 error rate of 3.57% in ILSVRC and outperforms human-level performance in this kind of dataset.

3.2.4 3D Convolution

In some applications like medical imaging or seismic exploration, there is only three-dimensional data available although standard CNN architectures perform 2D convolutions on 2D input data. Nevertheless, it is possible to generate 2D slices from a 3D data cube and feed these slices as 2D images into a convolutional neural network. By doing this, information on the third dimension can get lost because every slice is processed separately from its adjacent slices.

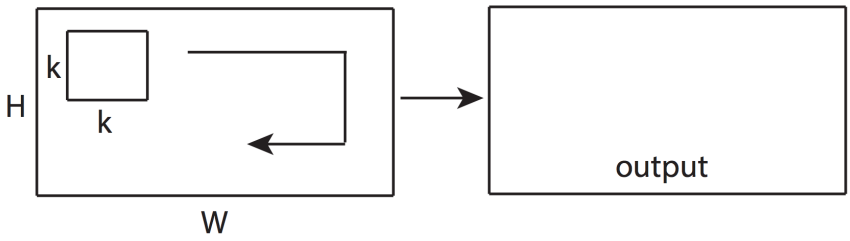


Figure 19: 2D Convolution [36]

Figure 19 shows the standard 2D convolution with 2D input data. The kernel of size $k \times k$ is moved across the input image with dimensions $H \times W$. The result of this operation is a 2D feature map whose size depends on kernel size and stride of the convolution.

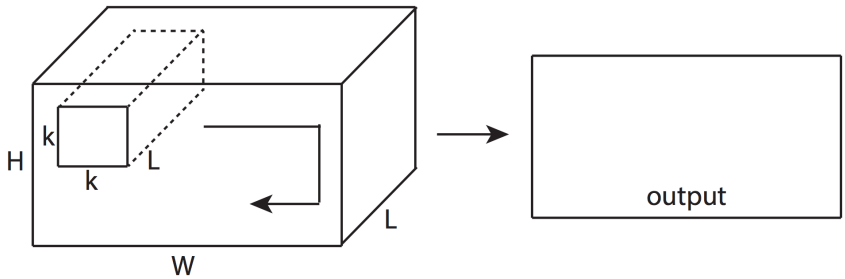


Figure 20: 2D Convolution with 3D input [36]

Figure 20 presents the process of passing 3D input data into a standard 2D convolution layer. The input is seen as a stack of L 2D slices that each have a size of $H \times W$. The kernel of size $k \times k$ is applied to all L slices and the results are summed up to form a single 2D feature map as output.

In order to capture spatial information from 3D data, a CNN can also use 3D convolutions. In this case, the kernel has to be extended by a third dimension d and is moved across the whole input volume of size $H \times W \times L$ (see figure 21). The resulting output feature map is also three-dimensional and preserves features across all three dimensions.

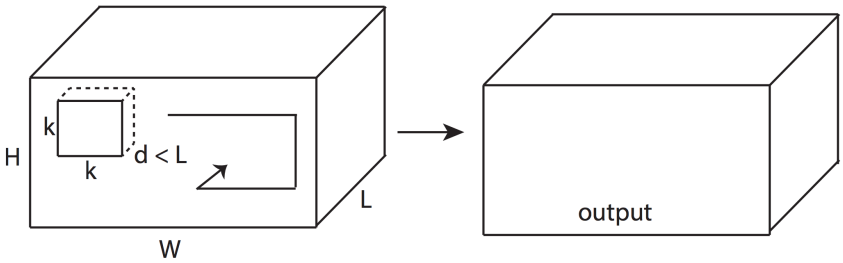


Figure 21: 3D Convolution [36]

To perform a 3D convolution, equation 25 from section 3.2.2 has to be adapted like follows to calculate the activation at position (j, k, h):

$$a^i_{j,k,h} = \sigma\left(b + \sum_{l=0}^{k}\sum_{m=0}^{k}\sum_{n=0}^{d} w_{l,m,n} a^{i-1}_{j+l,k+m,h+n}\right) \tag{26}$$

The same can be applied to the max pooling operation. Instead of choosing the maximum value in a 2×2 kernel, the 3D max pooling looks at a three-dimensional kernel e.g. of size $2 \times 2 \times 2$.

4 Visualization Methods

Visualizing features in CNNs is common practice to understand and trust decisions made by a network. While activations on the first layer can directly be projected into pixel space, correlations between higher layer features and the input image pixels are much more complex. Higher layers represent more complex and abstract concepts that can be detected by a neural network. Therefore, research focused a lot on visualizing features especially in higher layers. In this chapter, several visualization methods with different abilities and aims will be described and their usage for seismic feature detection will be analyzed.

4.1 Activation Maximization

The first visualization method is called Activation Maximization [9, 25] and is used to generate an input image that maximizes the activation of a certain neuron in any layer. Thus, Activation Maximization can generate images that certain neurons prefer most.

The aim of Activation Maximization is to synthesize an input image x^* that maximizes the activation a_i^l of the ith neuron in the lth layer:

$$x^* = \underset{x}{\operatorname{argmax}} \left(a_i^l(x) \right) \tag{27}$$

The fundamental algorithm consists of three sequential steps:

1. A random input image $x = x_0$ is generated and set as input for the algorithm.
2. The gradients $\frac{\partial a_i^l}{\partial x}$ with respect to the input image x are calculated using Backpropagation.
3. The direction of the gradient $\frac{\partial a_i^l}{\partial x}$ is used to change each pixel of the input image iteratively to maximize each neurons activation with a step size of η.

$$x \leftarrow x + \eta \frac{\partial a_i^l}{\partial x} \tag{28}$$

Steps 2 and 3 are repeated until the change in image x between two iterations is below a certain threshold. Image x then represents the optimal input image for the given neuron i and thus maximizes its activation. In order to compute a correct maximization, a_i^l denotes the unnormalized activation value of the last layer in the network before applying the softmax function.

Figure 22 shows 36 neurons from each of the three hidden layers of a Deep Belief Network [12] that was trained to detect hand-written digits from the MNIST dataset[12]. In the first hidden layer (left) only rough patterns and lines are visible. The second layer (middle) shows shapes that begin to resemble hand-written digits. In the last hidden layer the shapes of different digits are clearly visible, which proves that the neurons in this layer have been successfully trained to detect hand-written digits.

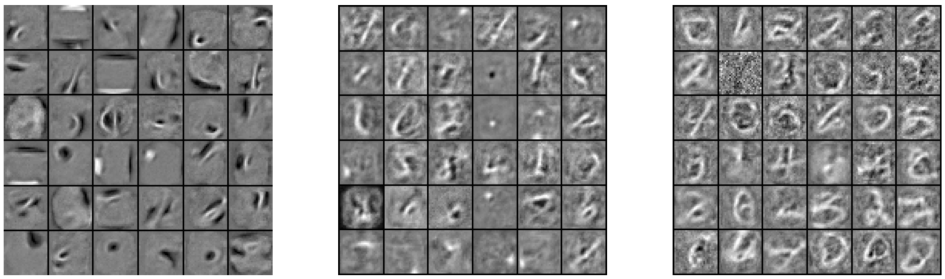


Figure 22: Results of Activation Maximization [9]

The MNIST dataset consists of 70,000 black-and-white images with a size of 28×28 pixels. Modern CNNs use much more complex input images with dimensions of e.g. 299×299 pixels for Inception V3 [35] and three color channels (RGB). In this case, Activation Maximization delivers unrealistic and not easily interpretable results.

To overcome these disadvantages of gradient ascent techniques like Activation Maximization, Yosinski et al [37] proposed the use of regularization. The regularization can be applied in step 3 of the optimization process, resulting in a change of the input image as follows:

$$x \leftarrow r_\theta \left(x + \eta \frac{\partial a_i^l}{\partial x} \right) \tag{29}$$

where r_θ denotes a regularization function.

A common regularization technique is L_2 decay which penalizes large values and prevents some extreme pixel values. To implement L_2 decay, r_θ is defined as:

$$r_\theta(x) = (1 - \theta_{\text{decay}}) \cdot x \tag{30}$$

where θ_{decay} denotes the decay parameter to control the strength of regularization.

[12] `http://yann.lecun.com/exdb/mnist/`

Another regularization technique applicable for Activation Maximization is Gaussian blur. Images produced by gradient ascent techniques often suffer from high frequencies which are neither realistic nor interpretable. Regularization via Gaussian blur is implemented as:

$$r_\theta(x) = \text{GaussianBlur}(x, \theta_{\text{width}}) \tag{31}$$

where θ_{width} specifies the width of the blurring kernel. As convolution with a blurring kernel is computationally expensive, the parameter θ_{every} is introduced to perform the blurring only every θ_{every} steps. This decreases the computational costs and does not change the results, as blurring multiple times with a small kernel is equivalent to blurring once with a larger kernel.

4.2 Deconvolution

To gain an insight into intermediate convolutional layers, the Deconvolution method [38] was introduced by Zeiler and Fergus in 2014. This technique shows patterns in the training set that activate given convolutional feature maps by reconstructing these features in the input pixel space.

The authors propose to extend an existing convolutional neural network with an inversed version of itself called DeconvNet [39] that, after a forward-pass through the original network, performs an upsampling of a certain feature map back to the input image. Figure 23 shows the architecture of DeconvNet with the standard forward-pass on the right and backward-pass on the left.

The results of a previous layer are passed to the convolution operation of the next layer, then the resulting feature map is passed through a ReLU function and in the end processed by a max pooling. As the max pooling operation decreases the size of the feature map and therefore removes values from the map, this process cannot easily be inverted. Zeiler and Fergus proposed to save each position of the highest value in the max pooling kernel and use this switch in the backward-pass to perform an unpooling. The left side of figure 23 shows the backward-pass that uses the reconstruction (or the network's output in case of the final layer) of a higher layer as input. With the previously recorded switches, the max pooling operation can be inverted by filling the max pooling kernel with zeros and restoring the maximum value in the position marked by the switch (see figure 24).

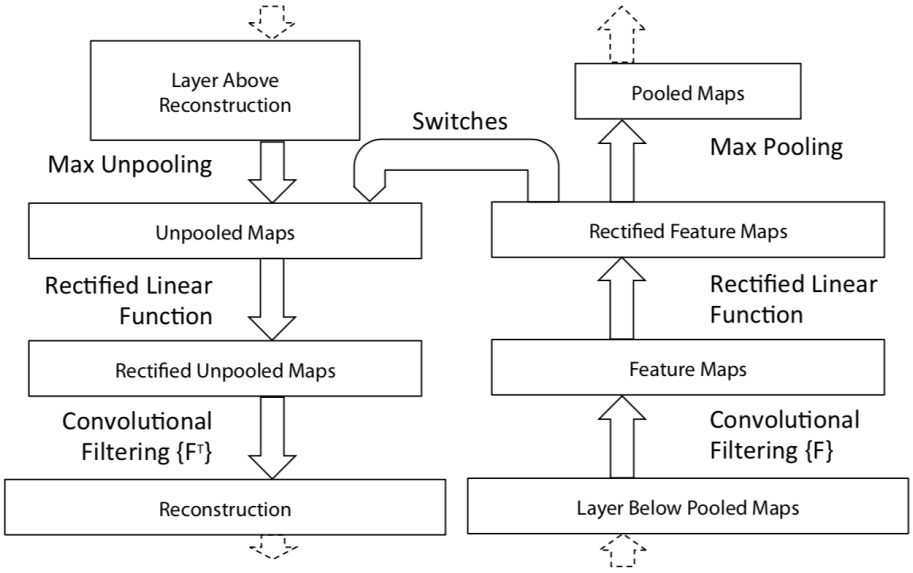


Figure 23: Architecture of DeconvNet [38]

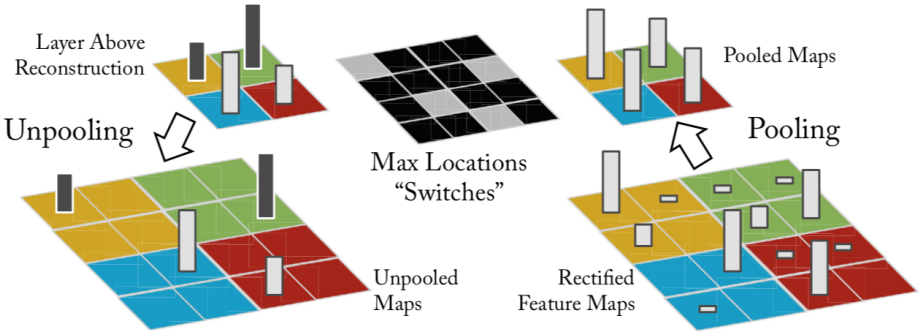


Figure 24: Unpooling via switches in DeconvNet [38]

This map is passed through a ReLU function another time and is finally being "deconvolved". The name Deconvolution might imply that this technique uses the inverse of the convolution operation. But in fact, the Deconvolution method uses a transposed convolution to upsample a feature map.

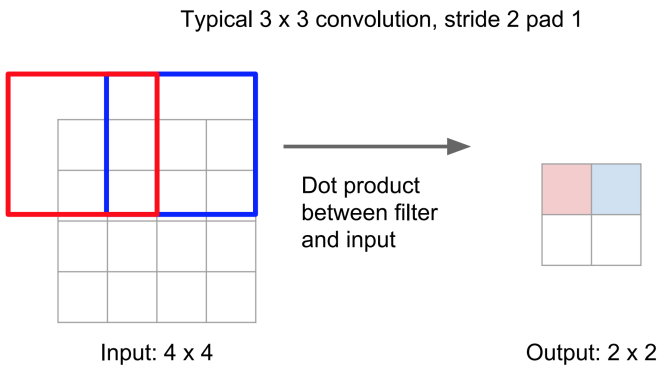


Figure 25: Forward-pass convolution [16]

In figure 25 the forward-pass of a standard convolution operation with a kernel size of 3×3 and a stride of 2 is depicted. At first, all values inside of the red box on the left are multiplied with the convolution filter which results in a single value that is written to the red field of the feature map on the right. Afterwards, the convolution kernel is moved to the right by a stride of 2 and the next value of the feature map is computed.

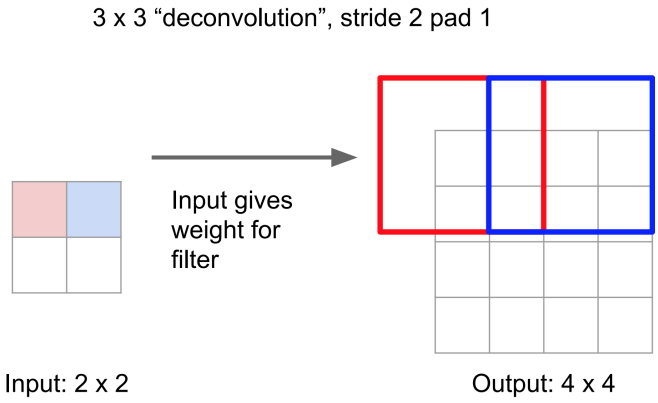


Figure 26: Backward-pass convolution [16]

In the backward-pass (see figure 26) a 2×2 input is upsampled to a 4×4 output by applying the so-called deconvolution. In this case, the convolution kernel is weighted with each of the input values and applied to the output feature map where overlapping regions are summed up. This operation is also called transposed convolution, backward-strided convolution or upconvolution [16].

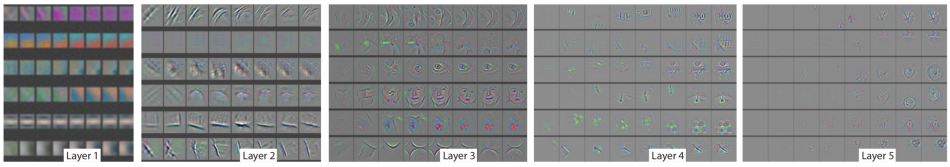


Figure 27: Evolution of features with Deconvolution [38]

Randomly chosen features from several convolutional layers obtained by the Deconvolution method are shown in figure 27. In the first layer there are usually only basic features like lines, dots or gradients visible. In higher layers the features become more complex and can comprise objects like faces, eyes or wheels.

4.3 Guided Backpropagation

Guided Backpropagation is a visualization method introduced by Springenberg et al [33] that uses a modified Deconvolution approach. This approach can be applied to a wider range of neural network structures and it leads to more accurate reconstructions of features especially from higher layers.

The authors questioned the necessity of different components commonly used in convolutional neural networks. They found out that a max pooling layer can be replaced by a convolutional layer with increased stride without losing accuracy [33].

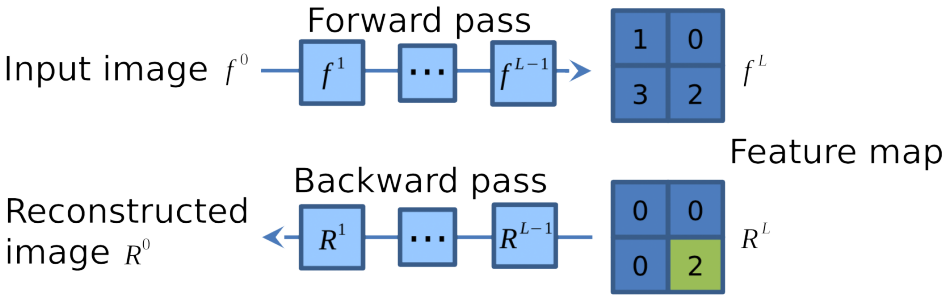


Figure 28: Scheme of forward and backward-passes [33]

To reconstruct a feature from a higher layer using Guided Backpropagation or Deconvolution, an input image is passed into the network up to a given layer, as depicted in figure 28. In the feature map of this layer, every value except one is set to zero. Then, this feature map is passed backwards

through the network to obtain a reconstruction of the activated feature at the input layer.

The important step in reconstructing features of higher layers is the backward-pass, where the ReLU function has to be applied. In a forward-pass through the network the ReLU function $\sigma(z) = max(0, z)$ sets every negative value to zero, as presented in the upper left of figure 29.

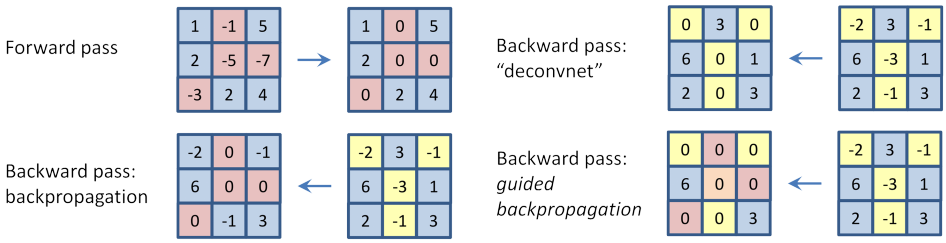


Figure 29: Comparison of different backward-passes [33]

Using the Backpropagation algorithm (see section 3.1.2), only the bottom gradient, i.e. the values from the forward-pass, are used for masking out values in the backward-pass. Thus, only those positions are set to zero that have been negative in the forward-pass (see figure 29, lower left) and the gradient of feature i in layer l denoted as R_i^l is calculated as:

$$R_i^l = \sigma(z_i^l) \cdot R_i^{l+1} \tag{32}$$

where $R_i^{l+1} = \partial a^L / \partial a_i^{l+1}$ and $\sigma(z_i^l)$ being the ReLU function that produces a feature map where every negative value has been replaced by zero.

For the Deconvolution method (section 4.2) the ReLU nonlinearity is applied in the backward-pass using the top gradient for the Backpropagation. This results in zeros at positions where the top gradient is negative and the gradient is computed as:

$$R_i^l = \sigma(R_i^{l+1}) \cdot R_i^{l+1} \tag{33}$$

The Guided Backpropagation combines the standard Backpropagation algorithm with the Deconvolution approach by using both bottom and top gradients to calculate the propagation through the ReLU nonlinearity in the backward-pass. Then, values are replaced by zero if at least one of the corresponding gradient values is zero. The gradient therefore is calculated as:

$$R_i^l = \sigma(z_i^l) \cdot \sigma(R_i^{l+1}) \cdot R_i^{l+1} \tag{34}$$

The name Guided Backpropagation originates from the additional guidance signal from the top gradient, which is added to the standard Backpropagation.

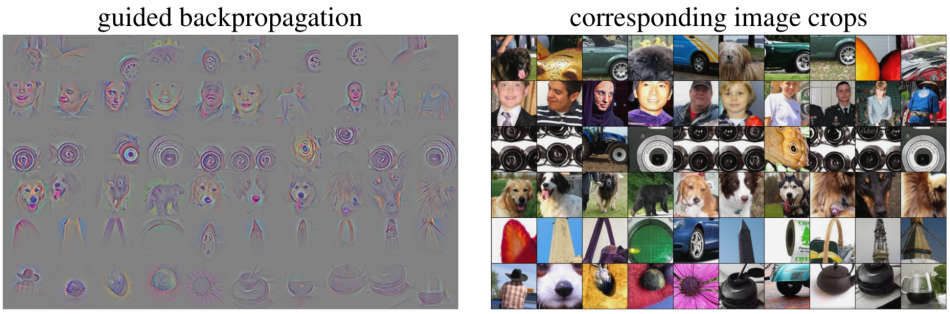


Figure 30: Features obtained by Guided Backpropagation [33]

Some results of Guided Backpropagation are depicted on the left of figure 30 with the corresponding input image crops on the right. For RGB images Guided Backpropagation delivers more accurate and visible features.

4.4 Class Activation Mapping

In 2016, Zhou et al proposed a visualization method called Class Activation Mapping [40] (CAM) that is able to produce a heatmap indicating the important regions in the input image that lead to the network's decision for a given class. In the original paper, these Class Activation Maps could only be obtained from fully convolutional neural networks that work without any fully-connected layer and use global average pooling.

The result of the global average pooling operation performed after the last convolutional layer is obtained by:

$$a_i^L = \frac{1}{Z} \sum_{x,y} f_i^L(x, y) \tag{35}$$

where $f_i^L(x, y)$ depicts the activation of unit i at spatial position (x, y) from last convolutional layer L. The input for the softmax function is then weighted by the weights w_i^c for every class c:

$$S_c = \sum_i w_i^c a_i^L \tag{36}$$

where w_i^c is an indicator for the importance of a_i^L for class c. The final output of the softmax function for every class c is then given by:

$$P_c = \frac{\exp(S_c)}{\sum_c \exp(S_c)} \tag{37}$$

The bias b is ignored in this case as it has too little influence on the classification result. By plugging together equations 35 and 36, we obtain:

$$S_c = \sum_i w_i^c \frac{1}{Z} \sum_{x,y} f_i^L(x,y) \tag{38}$$

$$= \frac{1}{Z} \sum_{x,y} \sum_i w_i^c f_i^L(x,y) \tag{39}$$

Eventually, the Class Activation Map M_c^{CAM} for class c is defined as:

$$M_c^{\text{CAM}}(x,y) = \sum_i w_i^c f_i^L(x,y) \tag{40}$$

which directly indicates the influence of an activation at spatial position (x,y) leading to a classification into a class c. Note that $S_c = (1/Z) \sum_{x,y} M_c(x,y)$ holds.

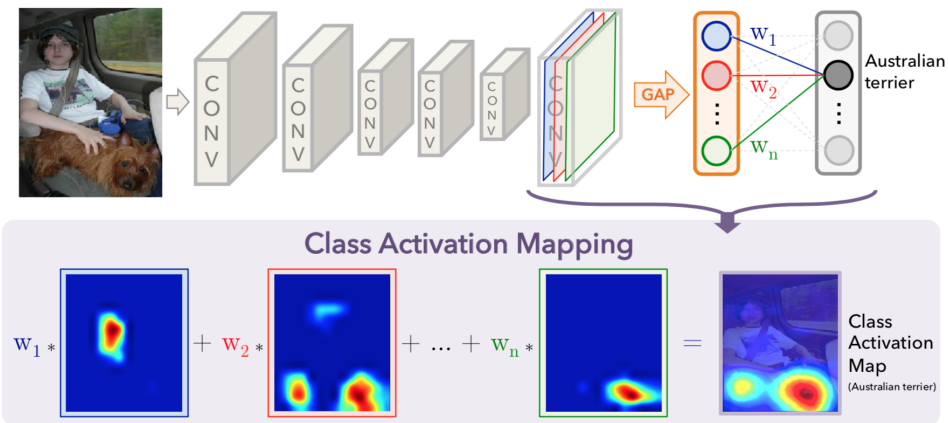


Figure 31: Mapping of class scores with CAM [40]

Figure 31 shows the process of mapping the predicted class score back to the previous convolutional layer. Thus, the CAM heatmap is generated and highlights the most discriminative regions of a given class in the input image.

4.5 Grad-CAM

To overcome the limitation of using only fully-convolutional neural networks, Selvaraju et al proposed a modification of the Class Activation Mapping method which they called Gradient-weighted Class Activation Mapping [30] (Grad-CAM). Grad-CAM can be applied to a significantly broader range of neural networks, including common convolutional neural networks with fully-connected layers at the end.

For this approach, a neuron importance weight α_i^c is defined as:

$$\alpha_i^c = \frac{1}{Z} \sum_{x,y} \frac{\partial y^c}{\partial f_i^L(x,y)} \tag{41}$$

where the first part represents the global average pooling operation and the second part the gradient of the class score y^c (before the softmax function) with respect to the feature map value f_i^L at position (x, y). To obtain a heatmap from this, a weighted combination of activation maps is performed:

$$M_c^{\text{Grad-CAM}}(x,y) = \sum_i \alpha_i^c f_i^L(x,y) \tag{42}$$

By passing the linear combination through the ReLU activation function, we can additionally assure that only positive influences for the class c are taken into account.

Comparing equations 40 and 42, we can see that in case $w_i^c = \alpha_i^c$ the maps for CAM M_c^{CAM} and Grad-CAM $M_c^{\text{Grad-CAM}}$ are identical. The authors proved that Grad-CAM is a generalization of CAM to arbitrary CNN-based architectures [30].

4.6 Guided Grad-CAM

Visualizations made with Grad-CAM can localize important regions in the input image but lack the ability to show fine-grained features like with Deconvolution or Guided Backpropagation. Therefore, Selvaraju et al also proposed a modification of their Grad-CAM method, called Guided Grad-CAM [30].

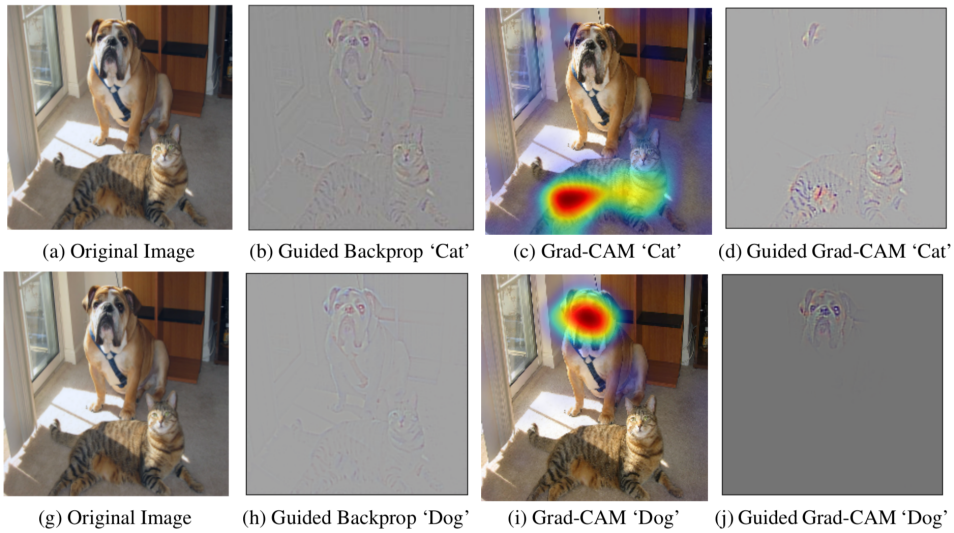


Figure 32: Results of Guided Backpropagation, Grad-CAM and Guided Grad-CAM [30]

For example, in figure 32 (c) the Grad-CAM visualization shows the region that contributes most to the decision 'Cat'. But it remains unclear which features in this region influenced this decision. With Guided Grad-CAM the features in this region are highlighted (see figure 32 (d)), e.g. showing stripes on the cat's fur.

The feature maps of the last convolutional layer are used to obtain the Grad-CAM heatmap, while Guided Backpropagation is applied in the backward-pass of the CNN to obtain features from the last fully-connected layer. Then, the Grad-CAM heatmap $M_c^{\text{Grad-CAM}}$ is upsampled to the size of the input image and element-wisely multiplied with the Guided Backpropagation features. The result shows fine-grained features only in those regions that are most discriminative for a given category, e.g. 'Cat'.

5 Implementation

In the application domain of oil and gas exploration, CNNs start to be used by the domain experts for classification purposes which lead to the hypothesis that the previously described visualization methods can also increase the understanding of and the trust in the achieved results here. In order to conduct experiments on seismic data, the algorithms had to be integrated into the existing DeepGeo[13] platform. In this chapter, the DeepGeo framework and the implementation of the visualization algorithms and the design of the new visualization tool called DeepVis will be described in detail.

5.1 DeepGeo

As user-driven classification approaches still needed a lot of manual interaction, members of the oil and gas industry started using deep learning to automate the seismic classification process. As part of the work in the VRGeo Consortium[14], the machine learning platform *DeepGeo* was created to run distributed machine learning tasks from a web interface and make neural network applications usable also for non-experts in this field.

DeepGeo consists of a scalable web interface from which Docker [15] containers, called jobs, can be started and managed. Every job can contain arbitrary code and can execute a variety of tasks like creating a training set, training a CNN or letting a CNN predict labels for given input data. In order to make CNN visualizations usable for both deep learning experts and non-experts, the visualization tool written as part of this master thesis is built on top of the DeepGeo framework.

One main application of DeepGeo is to view seismic datasets in a common internet browser. For this, DeepGeo includes a three-dimensional viewer in which volumetric seismic data can be displayed and labelled to mark visible seismic features like faults or channels. These annotations can then be used for training set generation. Figure 33 shows a three-dimensional view of the F3 dataset in DeepGeo with several annotations visible on the inline slice in green and red.

[13] `https://www.vrgeo.org/index.php?id=639`
[14] `https://www.vrgeo.org`
[15] `https://www.docker.com/`

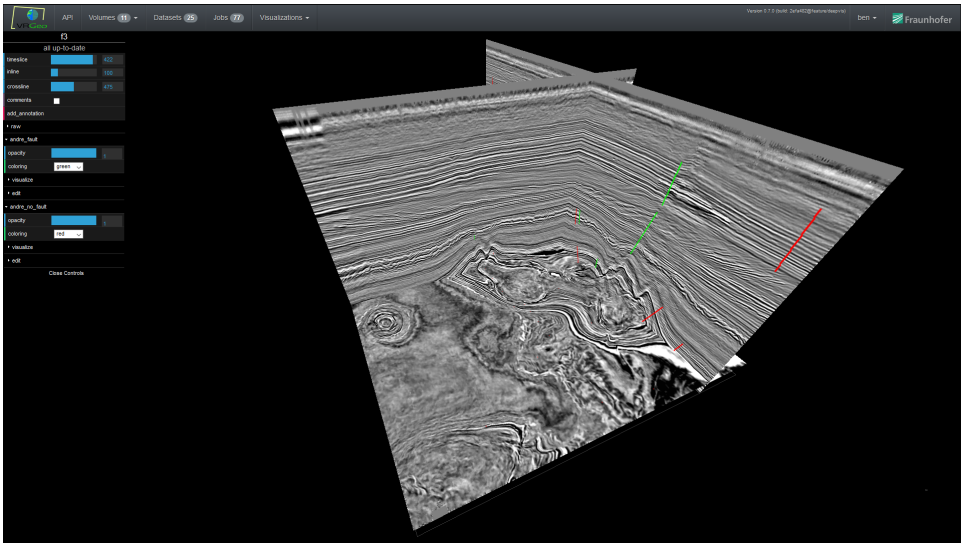


Figure 33: 3D view of F3 dataset in DeepGeo

So far, DeepGeo has the ability to run arbitrary code in a job that runs as a Docker container. Using the built-in DeepGeo API, a job can communicate with the DeepGeo server to save or edit volumes and slices. A job that runs a CNN on a given dataset can predict seismic features like faults or channels and store these as annotations in the DeepGeo database. Thus, only the input to the neural network and its outputs are known to a user. If the results are not satisfactory, a user might not be able to identify the reason for this bad performance as a CNN works like a black-box and does not reveal any information about its decision making. DeepVis fills this gap and tries to make deep learning and its decisions explainable.

5.2 Algorithms

For the implementation of the visualization algorithms from section 4 the Tensorflow framework has been used. Tensorflow[16] is a widely used open-source machine learning framework for a range of deep learning tasks. It provides definitions for neural network components like convolutional, max pooling and fully-connected layers and has several learning and optimization algorithms included. Also, Tensorflow handles the usage of one or several GPUs in order to make use of parallelizable operations and speed up the training and inference process. With Tensorflow users can concentrate on optimizing their neural network architectures instead of implementing every detail on their own.

[16] `https://www.tensorflow.org/`

Activation Maximization

Algorithm 1 shows the implementation of the Activation Maximization method in Python code. The function receives a *tensor* object from Tensorflow which represents a layer of the CNN, the number of *iterations* to repeat the optimization process and the regularization parameters *l2_decay*, *gauss_sigma* and *gauss_every*.

In line 2, the network with input (X) and output (Y) layer and logits (which describe the output of the last fully-connected layer prior to the softmax function) as well as several additional variables like input image shape or Tensorflow session object is obtained with the method *get_network()*. Following that, a random input image with values in a range of $[0.45; 0.55]$ is initialized, as input images are normalized between 0 and 1. The gradients of the given layer *tensor* with respect to the input image X are computed with Tensorflow's *tf.gradients()* method and afterwards multiplied with a step size of $gradient.std()^{-1}$ which has proven to give a fast convergence. The input image is updated on line 13 and depending on the parameters, regularization is applied.

Algorithm 1 Activation Maximization

```
1  def AM (tensor, iterations, l2_decay, gauss_sigma, gauss_every):
2      net = get_network()
3      num_features = get_num_features(tensor)
4      am_images = [np.random.uniform(0.45, 0.55, net.input_image_shape) ↲
5                  for n in range(num_features)]
6      losses = [tf.reduce_mean(tensor[..., n]) for n in range(num_features)]
7      gradients = [tf.gradients(losses[n], net.X) for n in range(num_features)]
8      for n in range(num_features):
9          for i in range(iterations):
10             # perform gradient ascent
11             gradient = net.session.run(gradients[n], feed_dict={net.X: [am_images[n]]})
12             step_size = 1.0 / (gradient.std() + 1e−8)
13             am_images[n] += step_size ∗ gradient
14             # apply regularization
15             am_images[n] = (1.0 − l2_decay) ∗ am_images[n]
16             if i % gauss_every == 0:
17                 am_images[n] = gaussian_filter(am_images[n], gauss_sigma)
18     return am_images
```

The results of the Activation Maximization algorithm, as well as of every other algorithm described in this section, are not normalized yet and should be converted to bytes (uint8) in a range between 0 and 255 before saving them as an image.

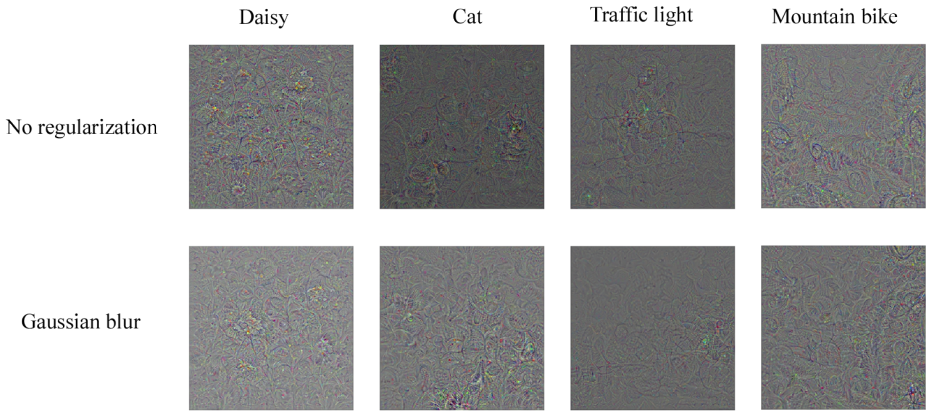


Figure 34: Results of Activation Maximization on Inception V3 with no regularization (top) and Gaussian blur regularization with *gauss_every* $= 10$ and *gauss_sigma* $= 0.5$ (bottom)

Activation Maximization can reveal concepts learned by a neural network by creating optimal input images for every neuron or class. Nevertheless, one disadvantage of Activation Maximization is the poor interpretability of the synthesized results for complex objects. In figure 34, the optimal input images for four classes (daisy, cat, traffic light, mountain bike) are shown with no regularization in the upper row and Gaussian blur with *gauss_sigma* $= 0.5$ and *gauss_every* $= 10$ in the lower row. For example, the AM results for the class *daisy* show some flower-like patterns and in the bottom image for class *mountain bike* parts of a wheel are recognizable. Although the regularization changes the appearance of the images slightly, they remain very hard to interpret and do not resemble real-life examples of the input classes. With the help of a Deep Generative Network (DGN) the results can be forced to look more realistic [25].

Backpropagation and gradients

Deconvolution and Guided Backpropagation rely on a backward-pass through the network with modified gradients after the ReLU function. Therefore, they use the same method (see algorithm 2) which uses a *tensor* object and an *input_image* as parameters. In the backward-pass the gradients of the *tensor* object with respect to the input image are computed.

Algorithm 2 Backpropagation

```
def Backprop (tensor, input_image):
    net = get_network()
    num_features = get_num_features(tensor)
    # perform backpropagation via gradients
    gradients = [tf.gradients(tensor[..., n], net.X)[0] ↲
            for n in range(num_features)]
    backprop_images = [net.session.run(gradients[n], feed_dict={net.X: [input_image]}) ↲
            for n in range(num_features)]
    return backprop_images
```

In the backpropagation process, the gradient calculation for the ReLU function has to be modified in order to create Deconvolution or Guided Backpropagation results. Therefore, algorithm 3 shows the custom gradient calculation methods for Deconvolution and Guided Backpropagation. Also, the initialization of the Tensorflow graph with modified gradients is listed in the method *initialize_graph_with_relu()*.

Algorithm 3 Custom gradients for ReLU

```
@ops.RegisterGradient('DeconvRelu')
def _DeconvReluGrad(op, gradient):
    # define custom gradient for ReLU (Deconvolution)
    return tf.where(gradient > 0.0, gradient, tf.zeros(tf.shape(gradient)))

@ops.RegisterGradient('GuidedRelu')
def _GuidedReluGrad(op, gradient):
    # define custom gradient for ReLU (Guided Backpropagation & Guided Grad−CAM)
    return tf.where(gradient > 0.0, gen_nn_ops.relu_grad(gradient, op.outputs[0]), ↲
            tf.zeros_like(gradient))

def initialize_graph_with_relu(relu):
    net = get_network()
    with net.graph.gradient_override_map({'Relu': relu}):
        saver = tf.train.import_meta_graph(net.model_file + '.meta')
    saver.restore(net.session, net.model_file)
```

Deconvolution

To apply the Deconvolution method to an input image, the network graph has to be initialized with the *DeconvRelu* gradient from algorithm 3. Then, the *Backpropagation* method is called to generate the Deconvolution results in the backward-pass (see algorithm 4).

Algorithm 4 Deconvolution

```
def Deconv (tensor, input_image):
    initialize_graph_with_relu('DeconvRelu')
    return Backprop(tensor, input_image)
```

Figure 35 shows 18 randomly chosen features of convolutional layer *conv_3_3* from VGG16 for an input image showing a daisy. In each feature the input image can be recognized as the most important shapes and edges have been detected by the CNN.

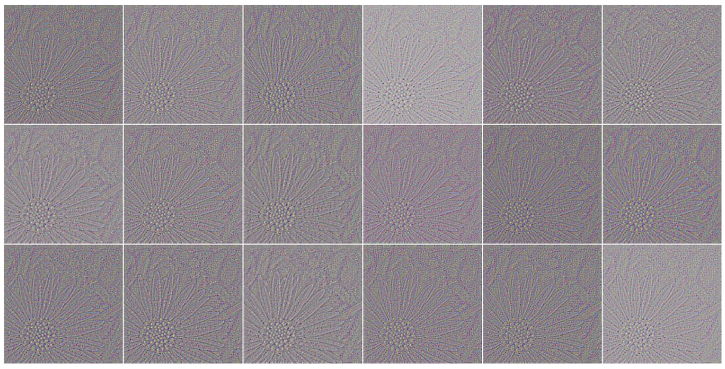


Figure 35: Results of Deconvolution on layer Conv_3_3 of VGG16

For the last layer of a CNN, the Deconvolution method generates a reconstruction of the input image with all its detected features. The original input images (top row) and their corresponding Deconvolution reconstruction are depicted in figure 36. The input images are clearly visible in the reconstructions, proving that the network has learned important features for each class.

Figure 36: Results of Deconvolution on last layer of VGG16

Guided Backpropagation

Similar to Deconvolution, the Guided Backpropagation algorithm, depicted in algorithm 5, first initializes the Tensorflow graph with the custom gradient for the ReLU function in line 2 and then performs backpropagation with these modified gradients.

Algorithm 5 Guided Backpropagation

```
def GuidedBackprop (tensor, input_image):
    initialize_graph_with_relu('GuidedRelu')
    return Backprop(tensor, input_image)
```

With Guided Backpropagation, not the whole input image but only the most discriminative features are reconstructed, resulting in much finer output images than with Deconvolution. The results of Guided Backpropagation on the last layer for six different input images are shown in figure 37.

Figure 37: Results of Guided Backpropagation on last layer of VGG16

Grad-CAM

The Grad-CAM algorithm (see algorithm 6) takes a *tensor* object, the *input_image* as array and the corresponding *input_label* as arguments. In line 3, the gradients of the logits are computed with respect to the input image. Note, that only the logits of the true label are considered, as we only want to obtain the most discriminative regions for this class. Then, a constant padding is added to the gradients to prevent high activations at the border of the image. In lines 8 and 9 the padded gradients are average pooled, summed up and passed through a ReLU function, as we are only interested in positive influences on our class probability. Because dimensions of the resulting heatmap equal the tensor's dimensions, the map has to be scaled up to the input image dimensions (lines 11-12).

Algorithm 6 Grad-CAM

```
def GradCAM (tensor, input_image, input_label):
    net = get_network()
    gradient_conv = tf.gradients(net.logits[..., np.argmax(input_label)], tensor)[0]
    # pad gradients with zero to remove high activations on the border
    paddings = tf.constant([[0, 0], [0, 1], [0, 1], [0, 0]])
    gradient_pad = tf.pad(gradient_conv, paddings, 'constant', constant_values=0)
    # perform global average pooling
    partial_lin = tf.nn.avg_pool(gradient_pad, [1, 2, 2, 1], [1, 1, 1, 1], 'valid')
    heat_map = tf.nn.relu(tf.reduce_sum(tensor * partial_lin, axis = 3, keepdims = True))
    # resize heat map to fit input image size
    resized_heat_map = tf.image.resize_bilinear(heat_map, np.shape(input_image), ↲
            align_corners = True)
    grad_cam = net.session.run(resized_heat_map, feed_dict={net.X: [input_image]})
    return grad_cam
```

Figure 38: Results of Grad-CAM on last convolutional layer of Inception V3

With Grad-CAM the ability of a CNN to locate objects can be visualized. In figure 38 the heatmaps obtained by Grad-CAM are depicted on top of six input images for the Inception V3 architecture. The results show that the network is not only able to recognize certain objects but also to locate them in the image. Note, that for example in the third image, showing two cyclists, Inception V3 can locate both bikes separately. In figure 39, a comparison between the classes *cat* and *dog* is presented. The left column shows the Grad-CAM results for the class *dog*, while in the right column the heatmaps for *cat* are visualized. We can see that the important regions change from dog to cat, showing us that Inception V3 can clearly distinguish both classes and is able to locate them.

Figure 39: Comparison of Grad-CAM heatmaps for classes cat (right) and dog (left) on Inception V3

Guided Grad-CAM

To make use of Grad-CAM's location abilities and the fine-graded features obtained by Guided Backpropagation, the Guided Grad-CAM algorithm was introduced in section 4.6. The implemented method (see algorithm 7) takes the same arguments as the Grad-CAM algorithm, being a *tensor* object, the *input_image* and the corresponding *input_label*. At the beginning, Guided Backpropgation is performed (lines 3-5) , followed by the Grad-CAM method (lines 6-12). In line 15, the results of both methods are combined by element-wise multiplication to obtain the final results of Guided Grad-CAM.

Algorithm 7 Guided Grad-CAM

```
def GuidedGradCAM (tensor, input_image, input_label):
    net = get_network()
    # Guided backpropagation
    initialize_graph_with_relu('GuidedRelu')
    gradient_X = tf.gradients(net.logits[:, np.argmax(input_label)], net.X)[0]
    # Grad−CAM
    gradient_conv = tf.gradients(net.logits[..., np.argmax(input_label)], tensor)[0]
    paddings = tf.constant([[0, 0], [0, 1], [0, 1], [0, 0]])
    gradient_pad = tf.pad(gradient_conv, paddings, 'constant', constant_values=0)
    partial_lin = tf.nn.avg_pool(gradient_pad, [1, 2, 2, 1], [1, 1, 1, 1], 'valid')
    heat_map = tf.nn.relu(tf.reduce_sum(tensor ∗ partial_lin, axis = 3, keepdims = True))
    resized_heat_map = tf.image.resize_bilinear(heat_map, np.shape(input_image), ↲
            align_corners = True)
    # Combine both methods by element−wise multiplication
    guided_grad_cam = tf.multiply(gradient_X, resized_heat_map)
    result = net.session.run(guided_grad_cam, feed_dict={net.X: [input_image]})
    return result
```

Guided Grad-CAM reveals detailed features only in regions that are important for the given class. Six input images and their corresponding visualization are depicted in figure 40. Note, that the lion for example seems to be recognized most from his eyes and nose, ignoring its mane completely.

Figure 40: Results of Guided Grad-CAM on the last convolutional layer of Inception V3

Again, we can use Guided Grad-CAM to compare the detection of the classes *dog* and *cat* as depicted in figure 41. Once more, the visualizations depict that Inception V3 can distinguish between the dog's and the cat's body. This time, not only the animal's faces are highlighted, but also the features leading to the classification results are visible. Most animals seem to be detected by their faces as this is the strongest feature in the Guided Grad-CAM results.

Figure 41: Comparison of Guided Grad-CAM results for classes cat (right) and dog (middle) on Inception V3. The original input images are shown on the left.

5.3 DeepVis

To make the previously mentioned visualization methods accessible and usable from the DeepGeo web interface, the visualization tool DeepVis was created as part of this thesis. The tool consists of several parts that will be explained in this section.

DeepVis Library

The DeepVis library is a Python library that implements all visualization methods listed in section 5.2 and handles calls to the DeepVis API. Every DeepGeo job can include the library to access visualization methods or send data to the API without explicitly implementing the visualization algorithms or the HTTP requests.

DeepGeo Job

A DeepGeo job can comprise arbitrary functionality that runs inside of a Docker container. Nevertheless, it is necessary to implement some spe-

cific lines of code in order to use this job with DeepVis. To map the whole process of working with CNNs in DeepGeo, the job must include the generation of training data, the training of the CNN and an inference step.

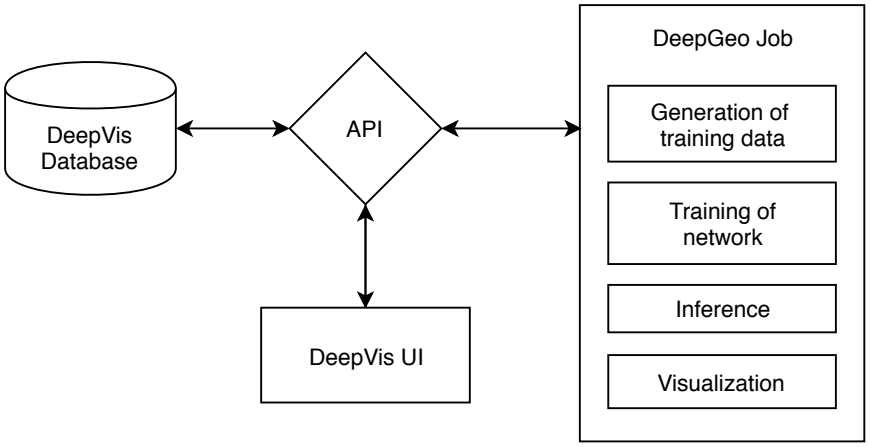


Figure 42: Connection of DeepVis database, DeepVis UI and DeepGeo jobs through DeepVis API

The connections between a DeepGeo job, the DeepVis API with its database and the DeepVis UI are depicted in figure 42. When creating training data, the obtained samples have to be saved in the DeepVis database which can be done by sending them to the corresponding endpoint of the DeepVis API. In the training step, only the network architecture needs to be saved in DeepVis to display the network in the UI. There must also be a visualization step in which visualizations for a given visualization method are created and saved via the DeepVis API.

DeepVis UI

The DeepVIs UI is a tool integrated into DeepGeo to create and view visualizations of CNN features. Figure 43 shows the user interface while using the Grad-CAM method. In the upper left corner, details about the trained CNN and its training data are listed. Beneath, the visualization method can be chosen and input images from the training data can be selected. The lower left corner contains a log for the visualization job. The architecture of the CNN is loaded from the database and displayed in 3D via WebGL in the upper right. Convolutional layers are depicted in blue, max pooling in yellow, fully-connected layers in red and dropout in black. The input is by default shown on the left of the network, the output on the right. Each visualization method generates results only for pre-defined layers. When the user clicks on such a layer, an enlarged version of the result image is being

revealed in the lower right corner of DeepVis UI. Below the architecture, the selected input images are listed with the CNN's prediction beneath every image.

Figure 43: User interface of DeepVis

During the evaluation of DeepVis (see chapter 6.5), members of the VR-Geo Consortium mentioned the need of visualizations for CNNs containing 3D convolutions, as 3D CNNs are currently used a lot in the industry. Therefore, support for 3D CNNs was added to DeepVis by using the slicing approach already known from visualizing the 3D seismic datasets.

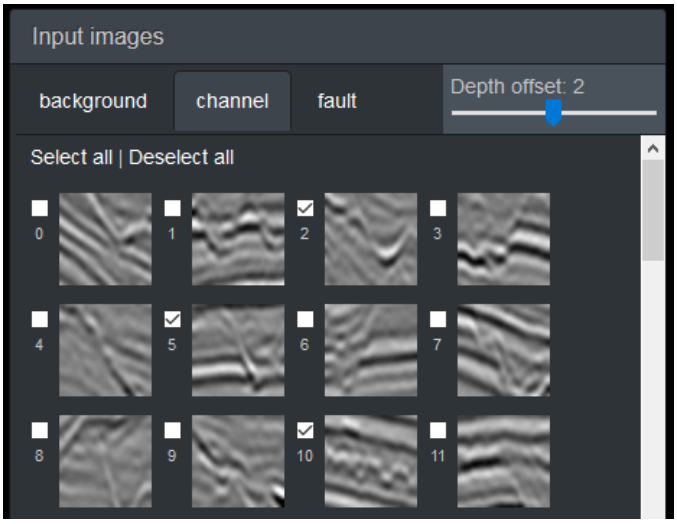


Figure 44: Selection of input images with depth slider in the upper right

Training datasets that use a patch size with a depth of more than 1 (e.g. $32 \times 32 \times 5$) can be viewed slice per slice by using a slider controlling the depth offset. The selection of input images in DeepVis UI is shown in figure 44. After changing the value of the slider in the upper right, all input images are updated dynamically to show the corresponding slice of the input volume. By default, the middle slice is shown as it always contains the labelled seismic feature. For input data that has a size of $32 \times 32 \times 5$ e.g., the depth offset ranges between 0 and 4, thus the default slice is 2.

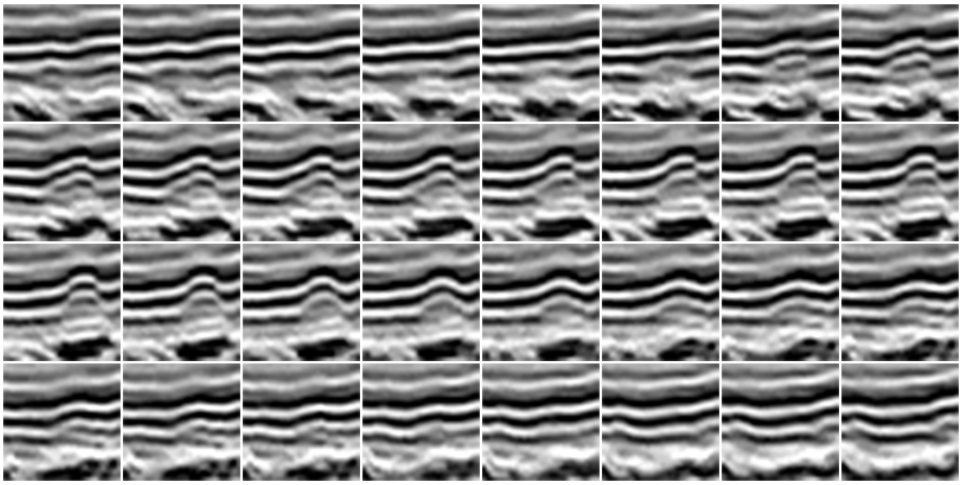


Figure 45: All 32 slices of a $32 \times 32 \times 32$ input volume

In figure 45 all 32 slices of a $32 \times 32 \times 32$ input volume are listed in ascending order from depth offset 0 to 31. The evolution of the seismic can easily be observed, showing an anomalous region appearing in rows 2 and 3.

After the visualization process for the CNN is finished, the results are added to the 3D view of the network architecture and can be selected. Similar to the input image selection, the result images e.g. from Grad-CAM can be viewed slice by slice if they are three-dimensional. Otherwise, the depth offset slider is hidden and the single 2D image is shown.

6 Evaluation

In order to evaluate the presented visualization methods for their usage in seismic interpretation, experiments have been conducted in the DeepGeo framework. For this, different CNN architectures had to be trained with different training datasets to subsequently analyze the visualization results from the implemented visualization methods presented in chapter 4. The use of these visualizations and the created tool will be discussed with deep learning experts and members of the oil and gas industry in section 6.5. Then, optimization possibilities of training data and network architecture derived from these visualizations will be evaluated in section 6.6 and discussed in the last section of this chapter.

6.1 CNN Architectures

For evaluation, several different CNN architectures have been used. The first one is called CNN7 and was adopted from Ying Jiang's master thesis [15]. It consists of seven layers plus input and output layer. The individual layers are listed in table 1. Depending on the training set, the input layer can either accept input sizes of $32 \times 32 \times 1$ or $32 \times 32 \times 5$. CNN7 comprises three 2D convolutional layers with a kernel size of 3×3 each, a stride of 1 and 32 or, respectively, 64 feature maps per layer. The first and the third convolutional layer are followed by a 2D max pooling with a kernel size of 2×2 and stride 2 to decrease the size of the feature maps. After that, a fully-connected layer with 512 neurons and a dropout of 50% follows. At the end of the network, the output layer with N neurons (N equals the number of classes the CNN is trained on) is followed by a softmax function that delivers the final classification result of CNN7.

	Operation	**Details**	**Output**
Input		Data preprocessing	32x32x1 or 32x32x5
Layer 1	2D Convolution	3x3, 32 feature maps, stride 1	32x32x32
Layer 2	2D Max Pooling	2x2 kernel, stride 2	16x16x32
Layer 3	2D Convolution	3x3, 64 feature maps, stride 1	16x16x64
Layer 4	2D Convolution	3x3, 64 feature maps, stride 1	16x16x64
Layer 5	2D Max Pooling	2x2 kernel, stride 2	8x8x64
Layer 6	Fully-Connected	512 neurons	512
Layer 7	Dropout	Dropout rate 50%	512
Output	Fully-Connected	N neurons	N

Table 1: Architecture of CNN7

Also, a three-dimensional CNN, called CNN12, was used for evaluation. The input of CNN12 is a volume of $32 \times 32 \times 32$ pixels. As listed in table 2, the architecture consists of 12 layers, including five 3D convolutional layers with a kernel size of $3 \times 3 \times 3$, a stride of 1 and 32, 64 or 128 feature maps. Three of these convolutional layers are followed by 3D max pooling layers with a kernel size of $2 \times 2 \times 2$ and stride 2. Then, CNN12 uses two fully-connected layers with 512 neurons and a dropout rate of 50% each. Eventually, the output layer consists of N neurons and uses the softmax function to compute the final prediction of the network.

	Operation	**Details**	**Output**
Input		Data preprocessing	32x32x32
Layer 1	3D Convolution	3x3x3, 32 kernels, Stride 1	32x32x32x32
Layer 2	3D Max Pooling	2x2x2 kernel, Stride 2	16x16x16x32
Layer 3	3D Convolution	3x3x3, 64 kernels, Stride 1	16x16x16x64
Layer 4	3D Convolution	3x3x3, 64 kernels, Stride 1	16x16x16x64
Layer 5	3D Max Pooling	2x2x2 kernel, Stride 2	8x8x8x64
Layer 6	3D Convolution	3x3x3, 128 kernels, Stride 1	8x8x8x128
Layer 7	3D Convolution	3x3x3, 128 kernels, Stride 1	8x8x8x128
Layer 8	3D Max Pooling	2x2x2 kernel, Stride 2	4x4x4x128
Layer 9	Fully-Connected	512 neurons	512
Layer 10	Dropout	Dropout rate 50%	512
Layer 11	Fully-Connected	512 neurons	512
Layer 12	Dropout	Dropout rate 50%	512
Output	Fully-Connected	N neurons	N

Table 2: Architecture of CNN12

Neural network with more layers usually contain a larger number of parameters that have to be optimized during training. The number of parameters for CNN7 and CNN12 are listed in table 3. CNN7 has about 2.2 million parameters for both input sizes, while CNN12 needs more than 5.3 million parameters. The high number of parameters for CNN12 leads to an extension of training time as both the size of the input data and the parameters to be optimized have been increased.

Architecture	**Input Size**	**Parameters**
CNN7	32x32x1	2.2 million
CNN7	32x32x5	2.2 million
CNN12	32x32x32	5.3 million

Table 3: Number of trainable parameters for each CNN architecture and input size

6.2 Training Data

A crucial part of training a CNN is the gathering or creation of training examples. For our experiments, the public available seismic datasets F3 [7] and Parihaka [21] have been used to detect seismic features.

F3

The F3 dataset has many areas containing faults, of which several are clearly visible. Nevertheless, some areas contain many small faults with very little space between them. In figure 46 an example from inline slice 100 of labels for *fault* (green) and *background* (red) is shown. There is a major fault in the left half of this slice as well as many smaller faults in the lower right.

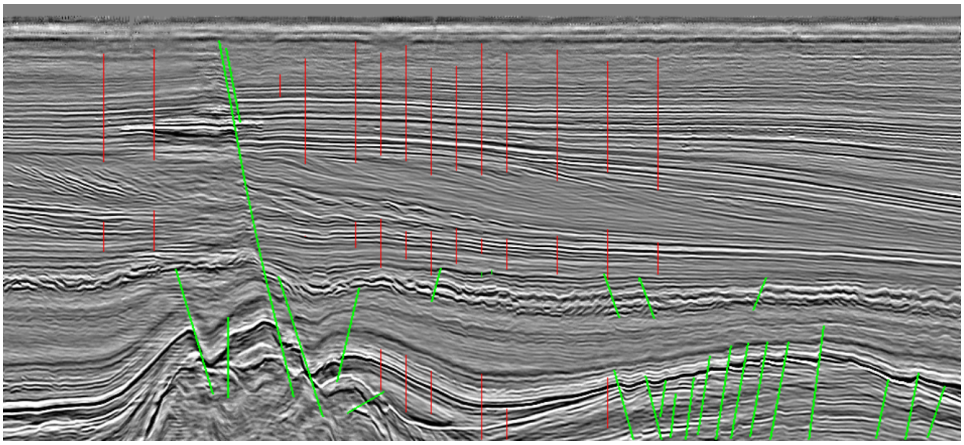


Figure 46: Annotations of classes *fault* (green) and *background* (red) on inline slice 100 in F3 dataset

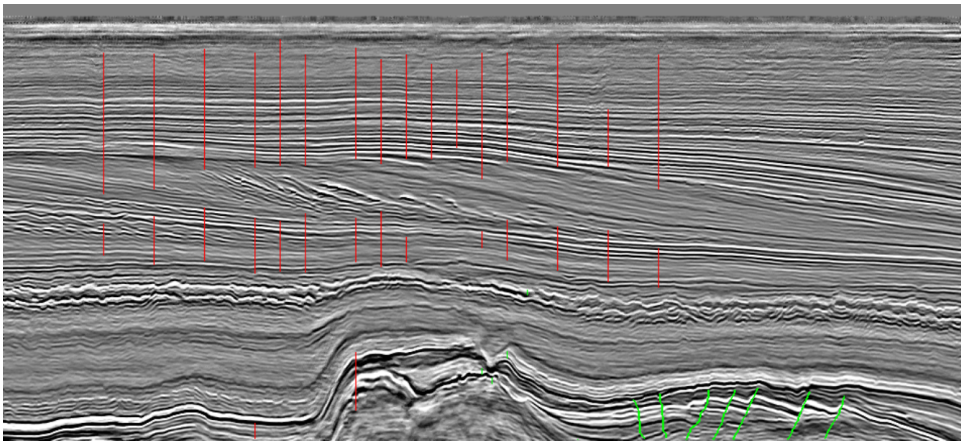


Figure 47: Annotations of classes *fault* (green) and *background* (red) on inline slice 325 in F3 dataset

On inline slice 325 (see figure 47) the major fault has vanished, but still there a many smaller faults present. There is also a salt dome present at the bottom of the dataset, which is not labelled, because the used patch size of 32×32 pixels is too small for detecting such a large-scale structure.

Parihaka

Compared to F3, the Parihaka dataset is geophysically much more complex. It also contains some large fault structures visible on the right in figure 48, as well as differently sized channel structures that are difficult to interpret. Because some channels are too large to be covered by a patch size of 32×32, only the lower boundaries of channels have been labelled. As those boundaries are not clearly distinguishable from the background, the annotation brush size has been chosen larger than for faults. Similar to F3, some regions without seismic features are labelled in red comprising the *background* class that contains neither faults nor channels.

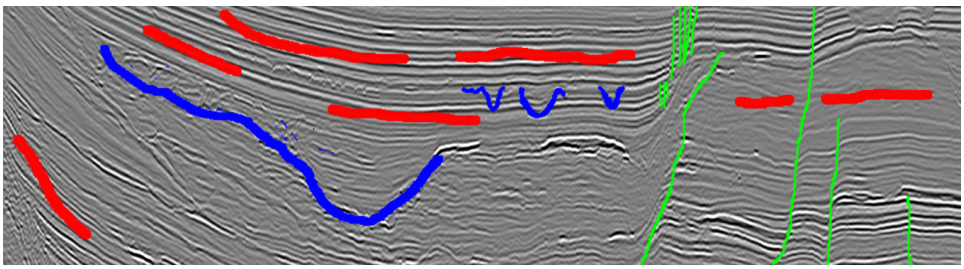


Figure 48: Annotations of classes *fault* (green), *channel* (blue) and *background* (red) on crossline slice 563 in Parihaka dataset

6.3 Classification Results

After training each CNN for 10 epochs, the validation accuracy of all architectures reached more than 99%. An epoch describes a full training cycle on the training set where every training sample is passed to the network once. Due to the fact that seismic data has to be interpreted and thus not every pixel of a slice can be labelled safely, the labelled training data is not complete. Also, the validation set size has been chosen very small with 1-3% of the whole dataset to have more samples left for training, as the amount of labelled seismic data is very limited.

F3

An example of the classification results from CNN7 with input size $32 \times 32 \times 1$ is depicted in figure 49. Apparently, the network has learned to distinguish faults from background, but the detected fault areas are not precise enough to be further used in the interpretation process. Some faults even have coalesced which renders these specific results useless for creating a fault model.

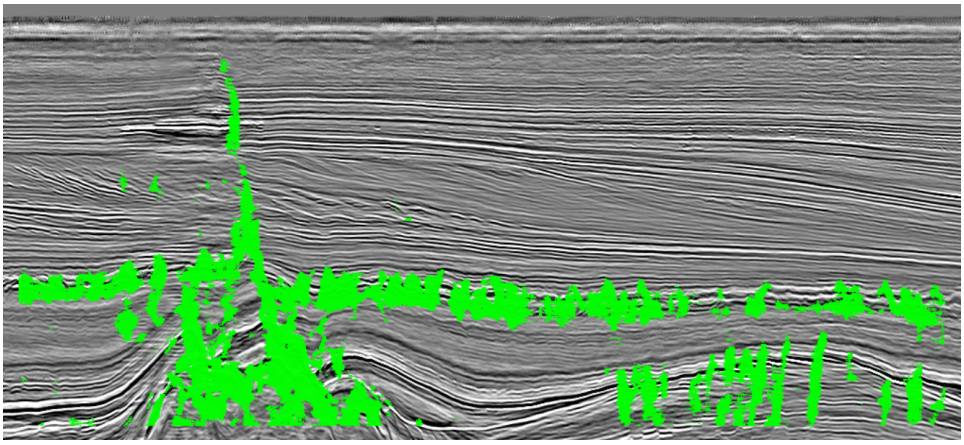


Figure 49: Detection of faults on inline slice 100 in F3 dataset with CNN7 and input patch of 32x32x1

For comparison, the classification results for CNN7 with input size $32 \times 32 \times 5$ and CNN12 are depicted in figures 50 and 51 respectively. The quality of the results leads to the conclusion that for now CNN7 with an input size of $32 \times 32 \times 1$ is best suited for detecting faults in F3, because the size of an annotation around a fault structure seems to increase proportionally with the depth of input. Nevertheless, the classifications contain a lot of

false-positives and are too coarse to be used in the seismic interpretation workflow.

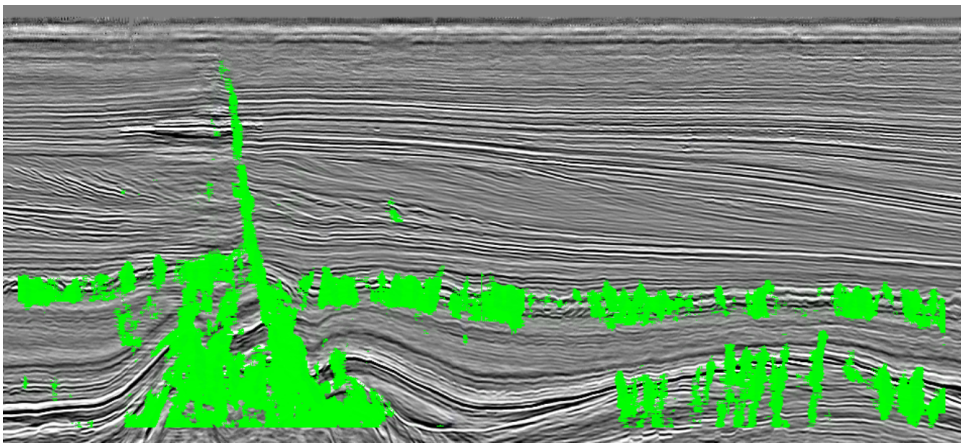


Figure 50: Detection of faults on inline slice 100 in F3 dataset with CNN7 and input patch of 32x32x5

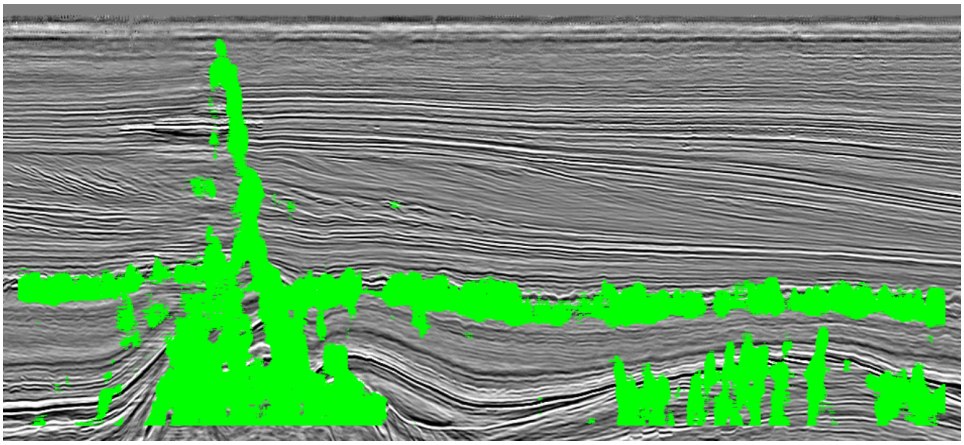


Figure 51: Detection of faults on inline slice 100 in F3 dataset with CNN12 and input patch of 32x32x32

Parihaka

In Parihaka, the classification results are even coarser than in F3. Figure 52 shows the large blue areas predicted as channels and also faults (green) are classified too wide. The classification results with an input size of $32 \times 32 \times 5$ and CNN12 are similar and far from being usable. As Parihaka is a geologically much more complex dataset than F3, this result was expectable. Since

only few regions can be interpreted and labelled with high confidence, the training data for Parihaka has bad quality.

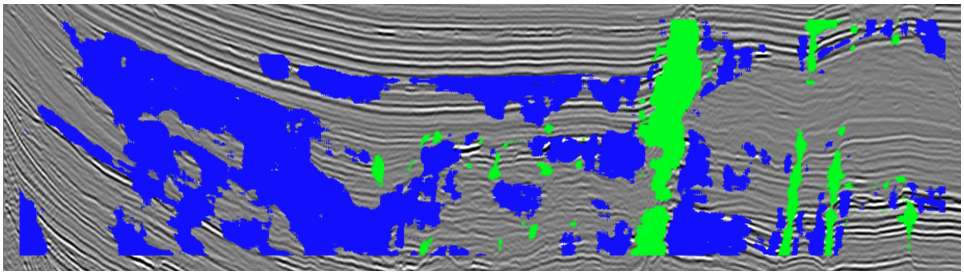


Figure 52: Detection of faults (green) and channels (blue) on crossline slice 563 in Parihaka dataset with CNN7 and input patch of 32x32x1

6.4 Visualization Results

In order to analyze and understand the decisions of the CNNs, the DeepVis tool was used to create visualizations with Activation Maximization, Deconvolution, Guided Backpropagation, Grad-CAM and Guided Grad-CAM.

6.4.1 F3

Activation Maximization

Activation Maximization was applied to all convolutional layers in CNN7 to synthesize optimal input images for each neuron. Figures 53a, 53b and 53c show the resulting input images for every neuron of the given convolutional layer. In figure 53d the optimal input for each class is depicted, with *fault* on the left and *no fault* on the right.

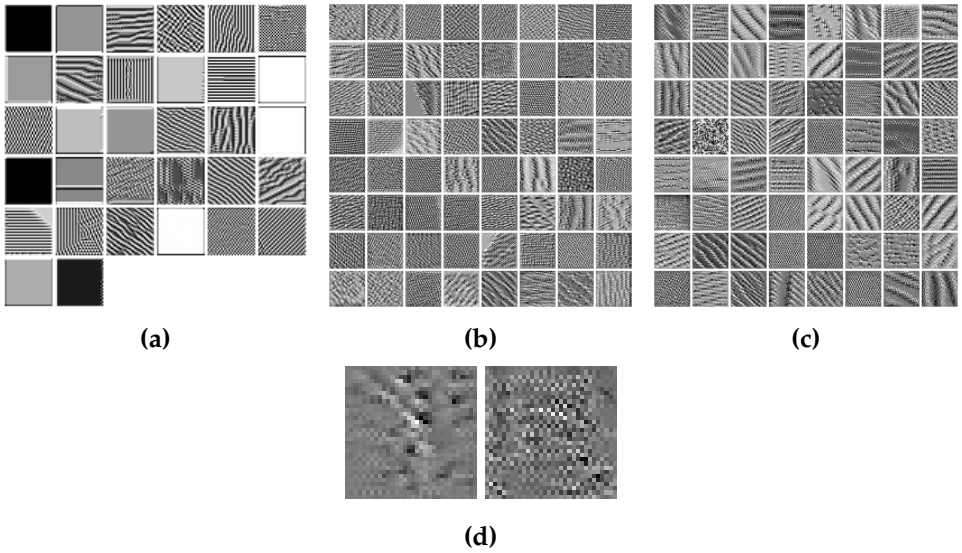


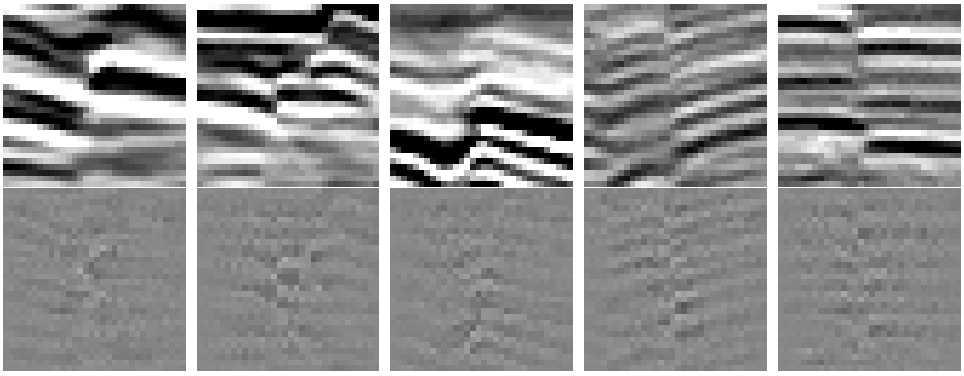
Figure 53: Results of Activation Maximization on layers 1, 3 and 4 (a)-(c) and on the final layer (d) for classes *fault* (left) and *no fault* (right) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

In figures 53a, 53b and 53c many Gabor filters are visible, which is common for convolutional neural networks. Gabor filters are used for texture analysis in image processing, analyzing specific frequencies in different directions in a local region of an image. Also, a lot of checkerboard patterns have been visualized, especially in figure 53b. These artifacts originate from inverting the max pooling layers and do not represent any patterns maximizing a neuron's activation.

In the optimal *fault* input image (figure 53d, left), several spots with high gradients are recognizable on the vertical axis in the center. The network has learned to react to horizontal gradients which indicates a fault in most cases. Nevertheless, the input for *no fault* appears noisy and has very little structure.

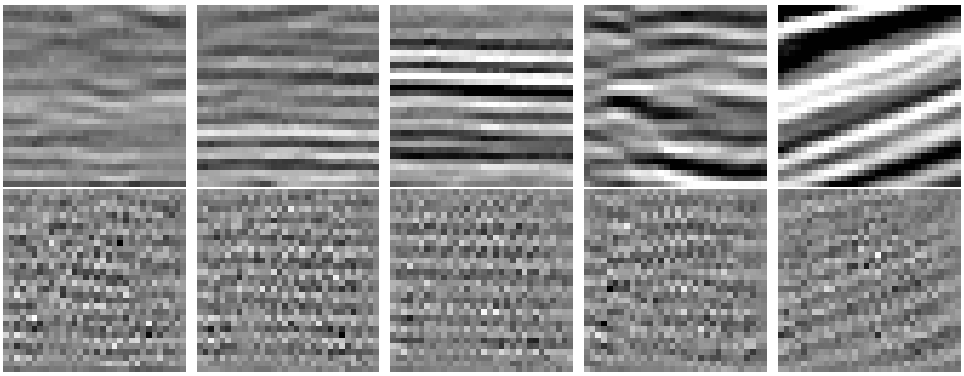
Deconvolution

The Deconvolution method is able to reconstruct features from the input image that convolutional layers react to. The results for five input images from class *fault* are depicted in figures 54a to 54e. Similar to the optimal class image from Activation Maximization, the Deconvolution results reveal a higher attention to the center of the image in case of faults.



(a) (b) (c) (d) (e)

Figure 54: Results of Deconvolution on the final layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)



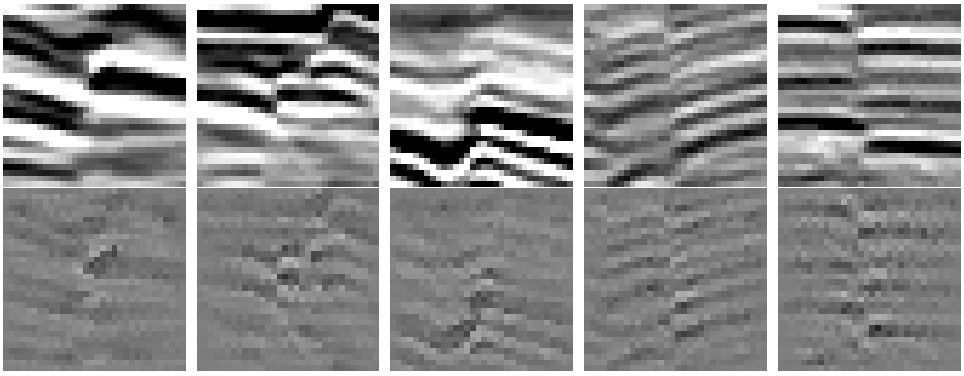
(a) (b) (c) (d) (e)

Figure 55: Results of Deconvolution on the final layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

Visualization results for the *no fault* class (see figures 55a to 55e) show that background samples seem to be detected through straight lines that run almost horizontally.

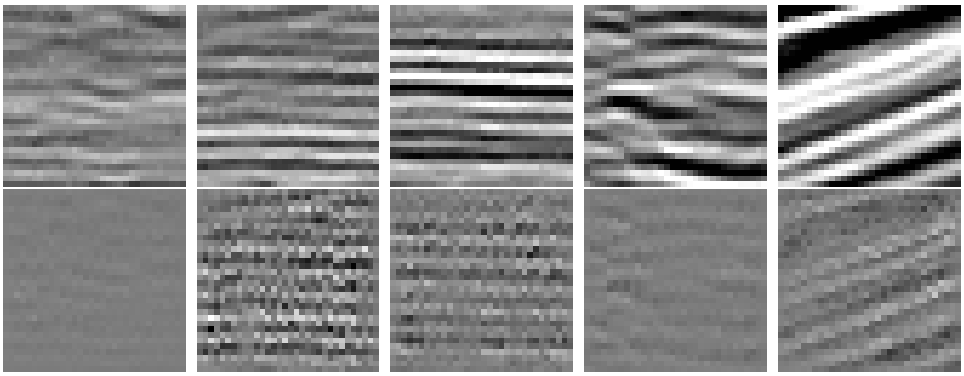
Guided Backpropagation

The visualizations generated by Guided Backpropagation (see figure 56) look similar to Deconvolution, although they are more intense.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 56: Results of Guided Backpropagation on the final layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)



(a) **(b)** **(c)** **(d)** **(e)**

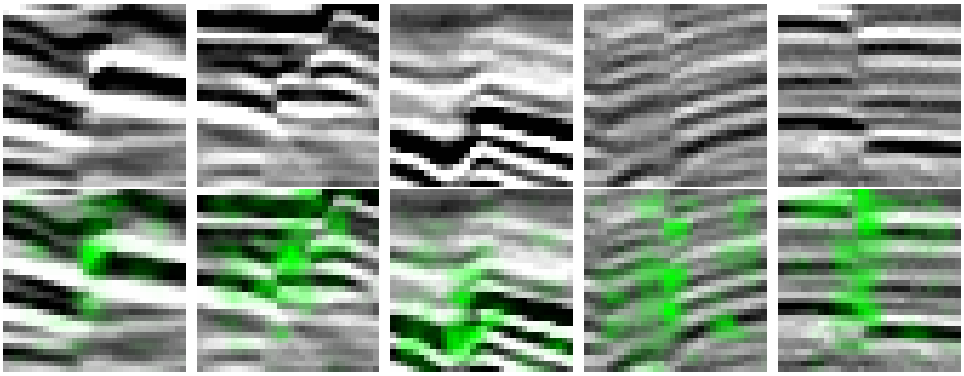
Figure 57: Results of Guided Backpropagation on the final layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

Especially the images obtained from *no fault* samples (see figure 57) show some differences to Deconvolution as there are two images (figures 57a and 57d) that possess very few visible features. Possibly, these samples have a larger distance to the mean of *no fault* samples and therefore the features are weaker.

Grad-CAM

The Grad-CAM method creates a heatmap for every input image showing the most discriminative regions for the predicted class. Figures 58a to 58e

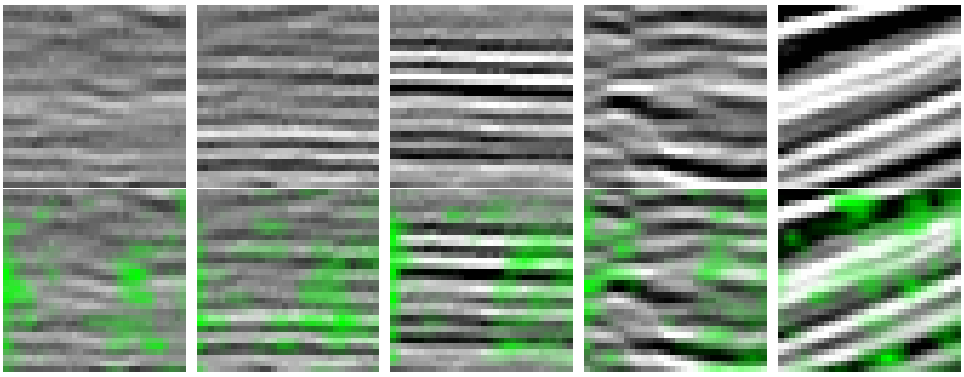
confirm the assumption that the CNNs look at high horizontal gradients in the center of the image. Nevertheless, the visualizations reveal also many activations apart from the center. As we use a sliding-window approach to classify the center pixel of each 32×32 sample, only faults that actually run through this pixel should be detected.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 58: Results of Grad-CAM on the last convolutional layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

The Grad-CAM results of *no fault* input images (see figures 59a to 59e) show discriminative regions spread across the whole image.

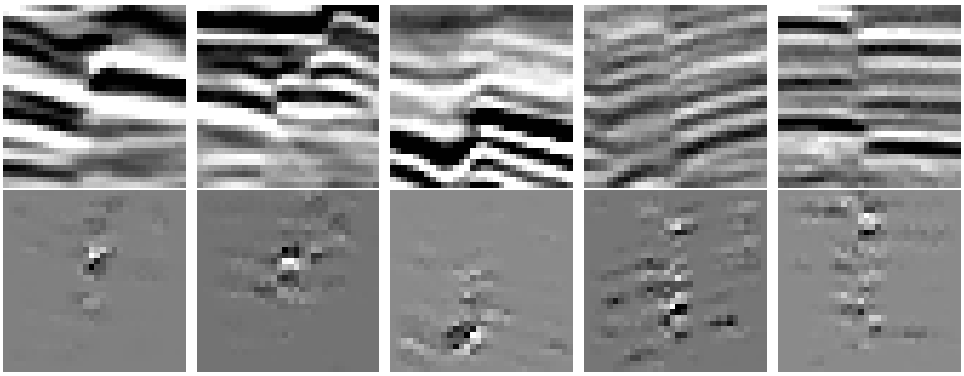


(a) **(b)** **(c)** **(d)** **(e)**

Figure 59: Results of Grad-CAM on the last convolutional layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

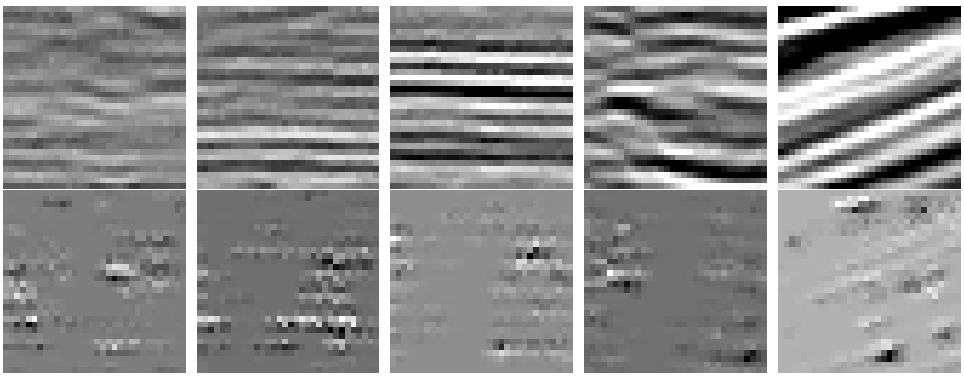
Guided Grad-CAM

The visualization results from Guided Grad-CAM (see figures 60 and 61) show fine-graded features in the most discriminative regions for the predicted class. Obviously, the results are similar to Guided Backpropagation but limited to the vertical axis in the center of the image in case of class *fault* (see figures 60a to 60e) or to various regions spread across the whole image for class *no fault* (see figures 61a to 61e).



(a) **(b)** **(c)** **(d)** **(e)**

Figure 60: Results of Guided Grad-CAM on the last convolutional layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)



(a) **(b)** **(c)** **(d)** **(e)**

Figure 61: Results of Guided Grad-CAM on the last convolutional layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

6.4.2 Parihaka

Activation Maximization

Also in the Parihaka dataset, Activation Maximization reveals Gabor filters in the convolutional layers (see figures 62a, 62b and 62c).

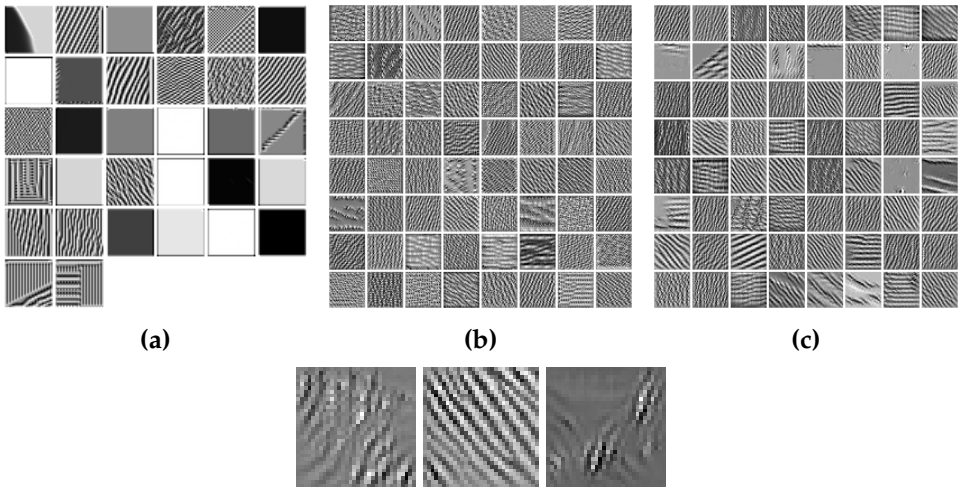
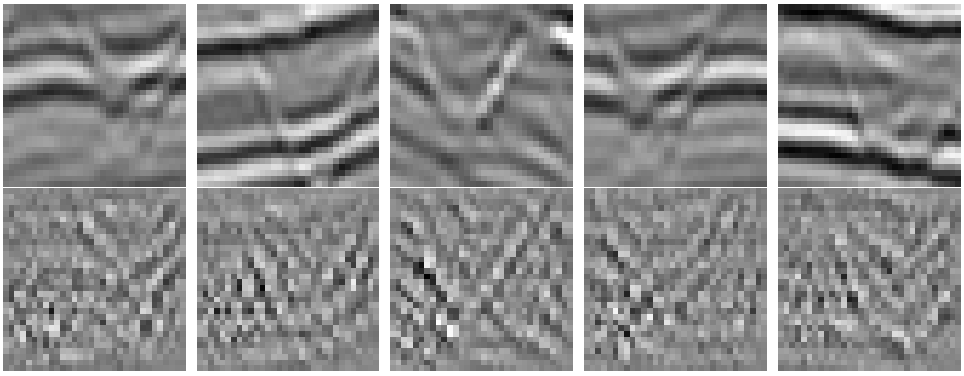


Figure 62: Results of Activation Maximization on layers 1, 3 and 4 (a)-(c) and on the final layer (d) for classes *fault* (left), *background* (middle) and *channel* (right) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

The optimal images for the classes *fault*, *background* and *channel* (see figure 62d from left to right) cannot easily be interpreted. A fault structure is not visible in the first image, while the optimal background image shows diagonal lines instead of horizontal lines. In the channel image a 'V' shape is recognizable, which usually resembles most kinds of channels.

Deconvolution

The Deconvolution for *fault* and *background* input images in Parihaka (see appendix A, figures 86 and 87 respectively) looks similar to the results from F3. In the reconstructions for *channels* depicted in figure 63, the diagonal lines forming the boundary of a channel are slightly visible.

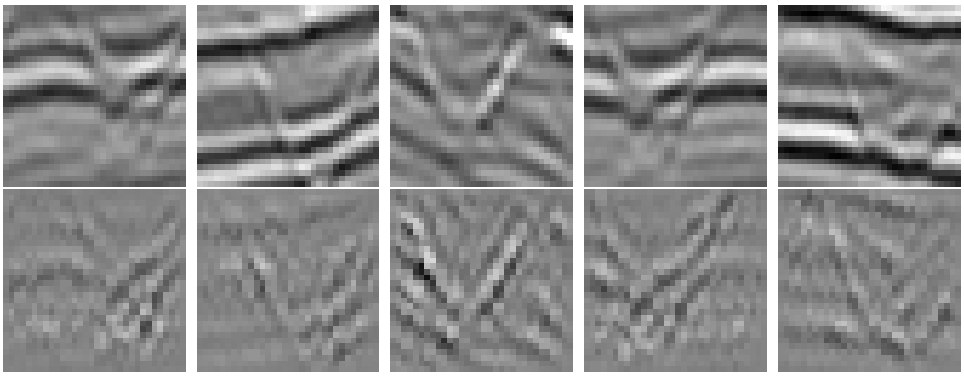


(a) **(b)** **(c)** **(d)** **(e)**

Figure 63: Results of Deconvolution on the final layer (bottom) for class *channel* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

Guided Backpropagation

The use of Guided Backpropagation in Parihaka reveals more visible boundaries for *channel* structures shown in figure 64. For *fault* and *background* input images (see appendix A, figures 88 and 89) the results again do not differ significantly from those in F3.

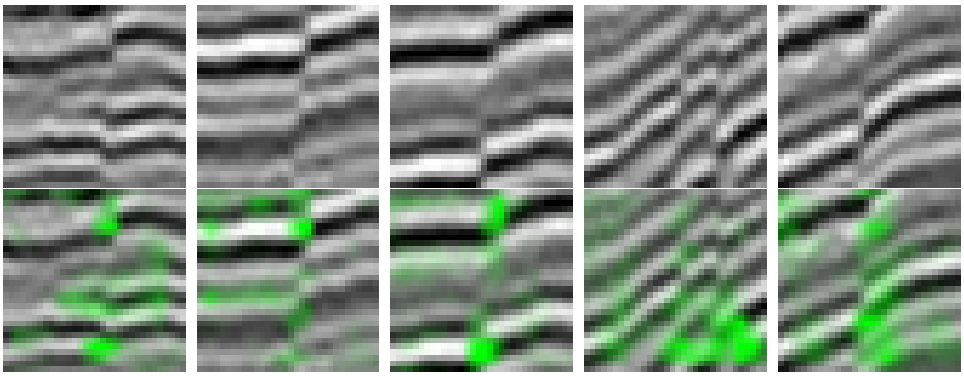


(a) **(b)** **(c)** **(d)** **(e)**

Figure 64: Results of Guided Backpropagation on the final layer (bottom) for class *channel* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

Grad-CAM

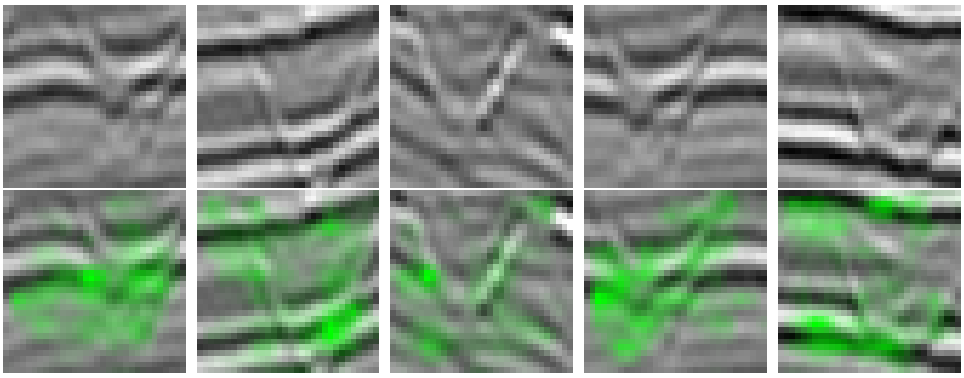
Grad-CAM shows that the important regions for the class *fault* lie mostly on the center vertical axis, although figure 65d reveals that the CNN also reacts to fault regions right of the center.



(a) **(b)** **(c)** **(d)** **(e)**

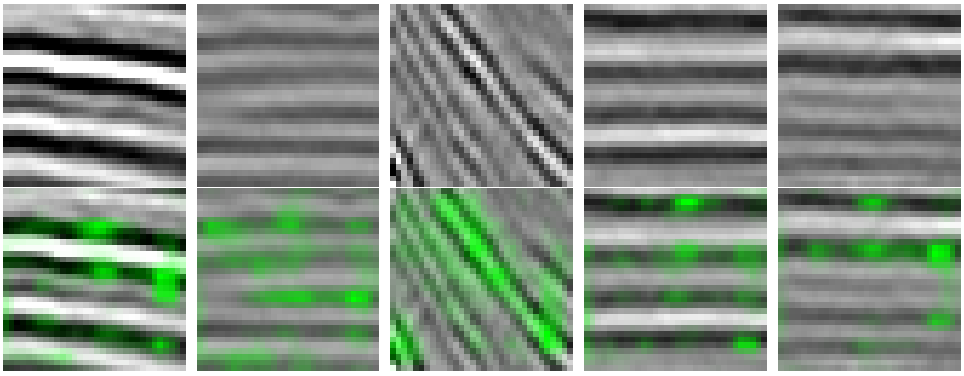
Figure 65: Results of Grad-CAM on the last convolutional layer (bottom) for class *fault* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

The most discriminative regions for *channels* represented in figure 66 are not always intuitive, as they often are positioned apart from the actual channel boundary in the input.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 66: Results of Grad-CAM on the last convolutional layer (bottom) for class *channel* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

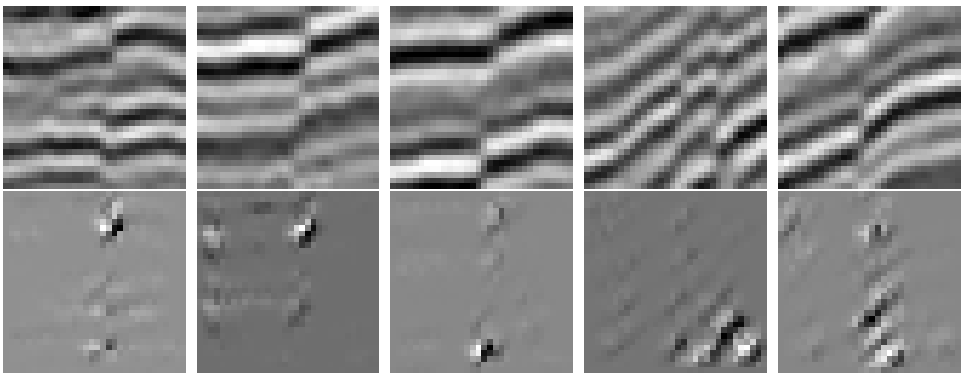


(a) (b) (c) (d) (e)

Figure 67: Results of Grad-CAM on the last convolutional layer (bottom) for class *background* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

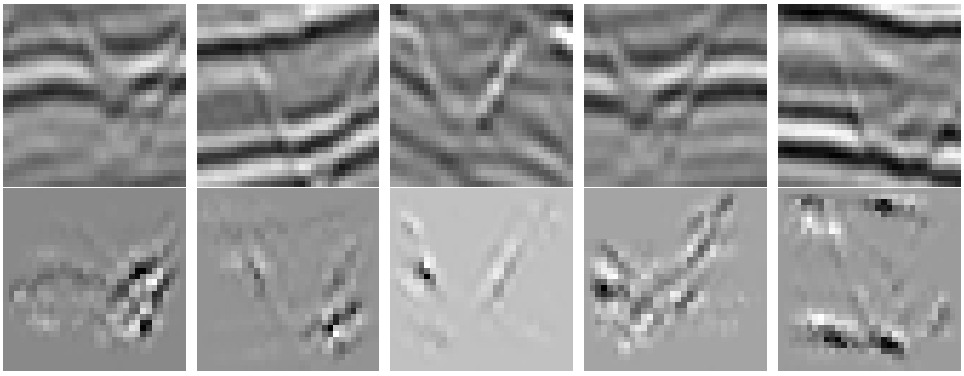
Guided Grad-CAM

With Guided Grad-CAM especially *channels* are recognizable in the resulting images depicted in figure 69. Although Grad-CAM shows regions in the heatmap that do not lie on the channel boundary, the combination with Guided Backpropagation leads to sharp diagonal lines and masked out horizontal lines.



(a) (b) (c) (d) (e)

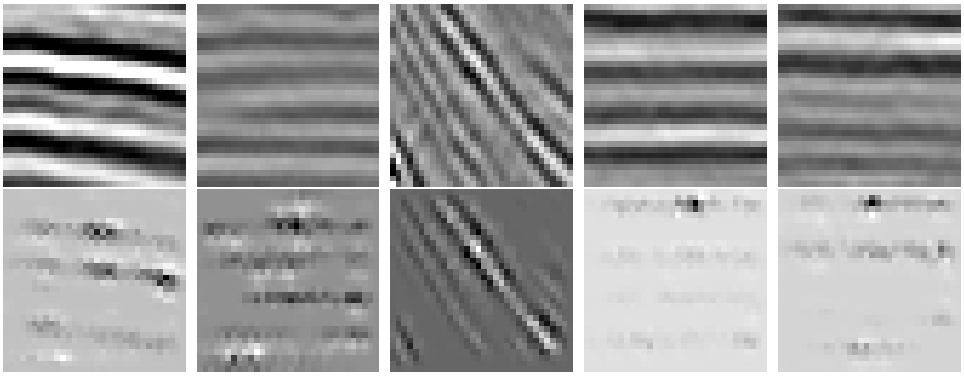
Figure 68: Results of Guided Grad-CAM on the last convolutional layer (bottom) for class *fault* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)



(a) (b) (c) (d) (e)

Figure 69: Results of Guided Grad-CAM on the last convolutional layer (bottom) for class *channel* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

Guided Grad-CAM also makes the horizontal lines visible, that the CNNs classify as background. Figure 70 demonstrates that only some of the lines from the input image are important for the classification.



(a) (b) (c) (d) (e)

Figure 70: Results of Guided Grad-CAM on the last convolutional layer (bottom) for class *background* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

6.5 Experts Feedback

After presenting the results of all introduced visualization methods, the quality and value of the visualizations in the context of seismic interpretation has to be evaluated. Therefore, a feedback session with several deep learning experts from Fraunhofer IAIS was conducted, in which the Deep-

Vis tool and the visualization results from section 6.4 have been presented and discussed.

For Activation Maximization the experts mentioned the appearance of Gabor filters in the first layers as indicator for the successful training of the CNNs. However, they agreed that the optimal class images do not resemble real-life examples and cannot be found in our available seismic datasets.

For Deconvolution and Guided Backpropagation results only the visualizations from the last layer were thought to be useful as they show high activations in important regions especially for faults. At this point, there is no real use of intermediate layer results because they do not contain much visible information for our small black-and-white input images.

Grad-CAM and Guided Grad-CAM have been considered the most valuable methods because they deliver easily interpretable results by highlighting important areas in the input images. These methods reveal if a CNN has learned wrong concepts that might lead to false-positive classifications. Also, both methods can confirm that a network precisely reacts to certain features in the input and therefore strengthen the trust in the results.

In conclusion, the deep learning experts found DeepVis a helpful tool to estimate the quality of training data and the actual accuracy of a trained CNN. With methods like (Guided) Grad-CAM and Activation Maximization the concepts learned by a network can be revealed and the adoption to the training data can be estimated, which might expose that the network has not been trained long enough for example.

A presentation of DeepVis to oil and gas experts from the VRGeo consortium delivered more feedback especially concerning the way to use DeepVis in DeepGeo. As many CNN experiments in the oil and gas industry are conducted with 3D CNNs, the experts suggest support for this in DeepVis, which was added at the beginning of evaluation to be able to compare 2D and 3D results. The feedback made clear that proper and high-quality training data is the most crucial part in order to obtain useful seismic classification results with CNNs.

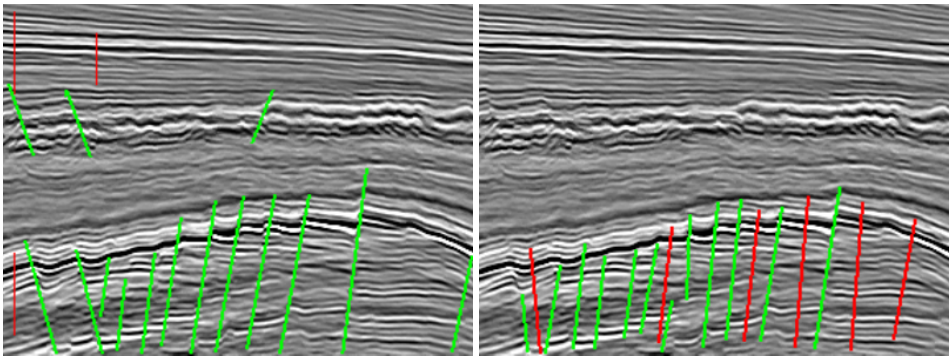
6.6 Optimizations derived from Visualizations

Resulting from the visualizations described in sections 6.4.1 and 6.4.2 and the feedback collected from experts in deep learning and the oil and gas industry, optimizations can be derived to improve the network's accuracy or performance.

6.6.1 Training Data

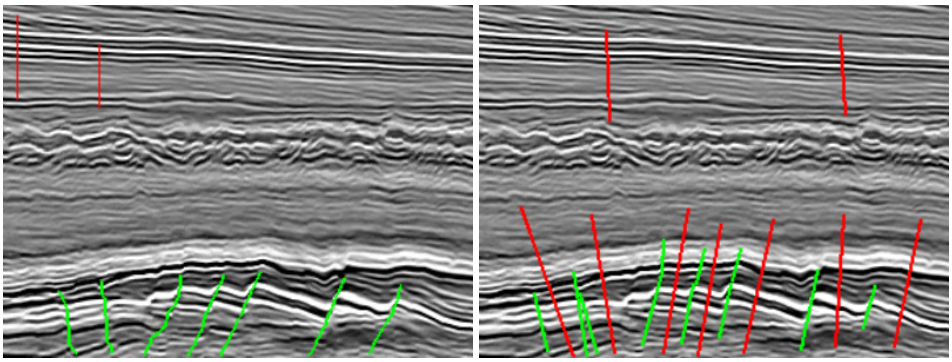
Grad-CAM and Activation Maximization visualizations revealed, that the network is trained to detect faults that lie in the center of the input image. Nevertheless, there are also some samples with high activations in regions apart from the center. In other words, the CNN detects not only the one pixel lying exactly on the fault, but also many pixels left and right of the actual true pixel.

As a consequence, the training data can be modified with additional *no fault* annotations that are positioned close to the fault annotations. By doing this, the CNN should learn to predict faults more precisely, which should also be visible in the visualizations made by DeepVis.



(a) **(b)**

Figure 71: Example of original (a) and modified (b) annotations for classes *fault* (green) and *background* (red) on inline slice 100 in F3 dataset



(a) **(b)**

Figure 72: Example of original (a) and modified (b) annotations for classes *fault* (green) and *background* (red) on inline slice 325 in F3 dataset

Figures 71 and 72 show comparisons of original and modified annotations in F3. Now there are *no fault* annotations between *fault* annotations, especially visible in the lower halves of the images. Also, the *fault* annotations have been modified to match the visible fault structure more precisely.

After training the CNN with the modified training data for 10 epochs, the inference results (see figure 73) expose finer classifications of the faults. Most faults are separated from their neighbouring faults and also the number of false-positive classifications has decreased especially in the salt dome at the bottom of the slice.

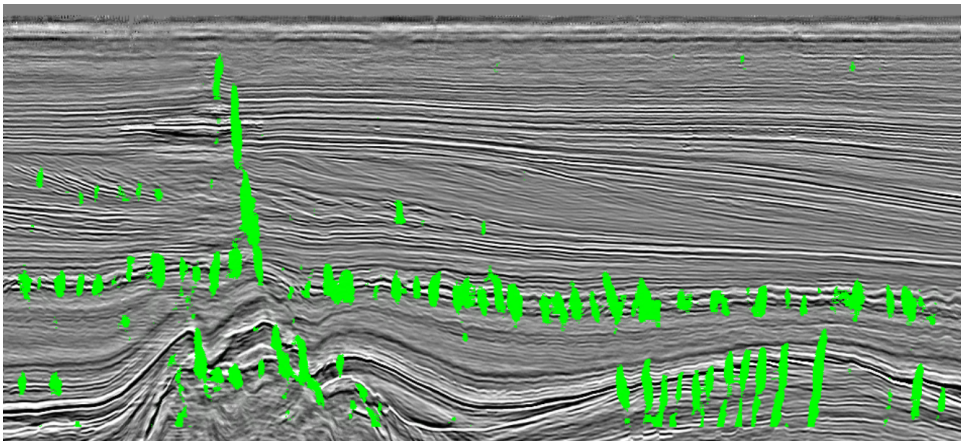


Figure 73: Detection of faults on inline slice 100 in F3 dataset with CNN7 and input patch of $32 \times 32 \times 1$

In order to understand the improvements made by the new annotations, visualizations have been performed on the trained network with DeepVis.

Activation Maximization

Activation Maximization still shows similar patterns for convolutional layers 1 to 3 (see figures 74a to 74c) but the optimal input images for every class (see figure 74d) obtained from the logits reveal changes in the detection of faults and background. The left image presents the optimal input for *fault*, with several transitions from black to white or vice versa on the vertical axis in the center of the image while the boundary regions show weak horizontal lines. The optimal input for *no fault* on the right represents almost the opposite of the fault image, as high gradients are only visible near the border while the center region is almost uniform.

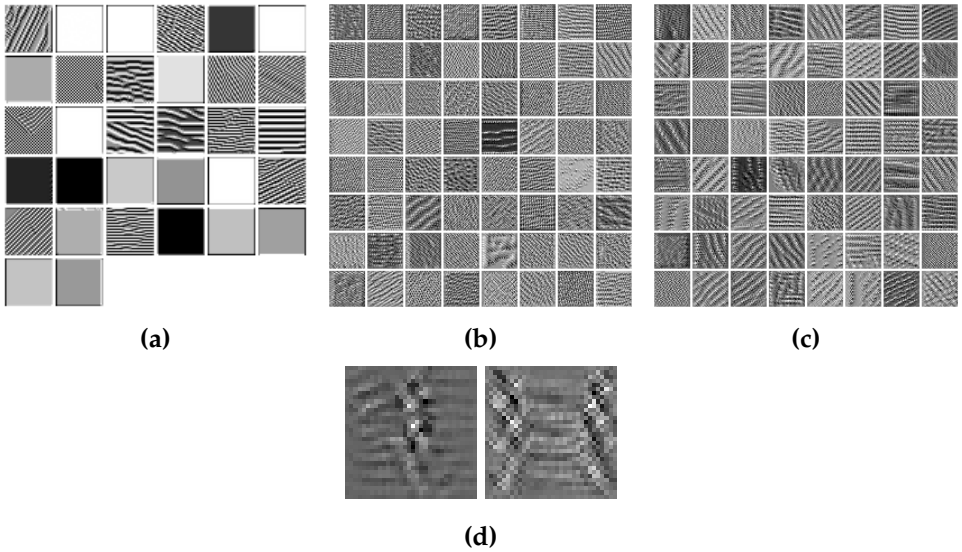
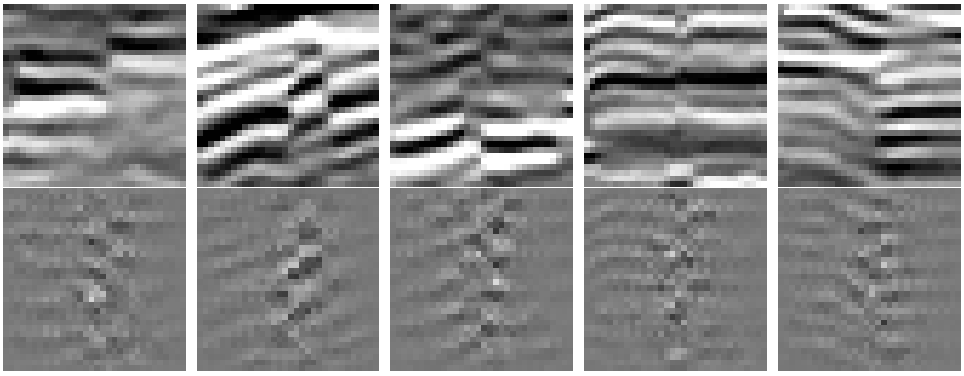


Figure 74: Improved results of Activation Maximization on layers 1, 3 and 4 (a)-(c) and on the final layer (d) for classes *fault* (left) and *no fault* (right) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

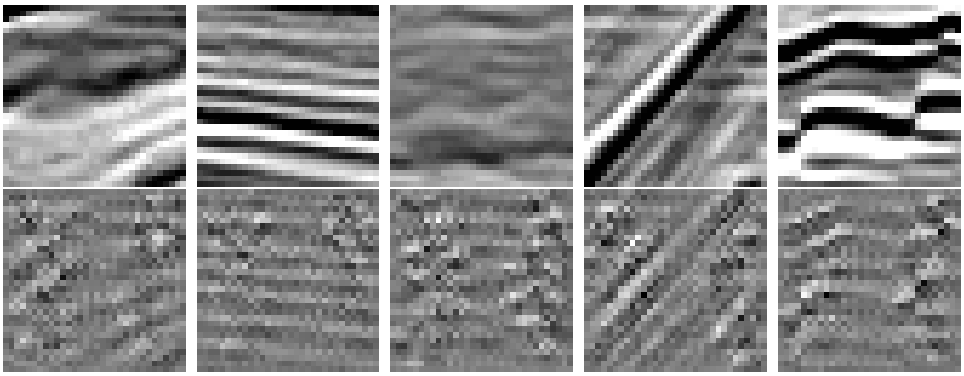
Deconvolution and Guided Backpropagation

Deconvolution and Guided Backpropagation present almost equal results with the modified training data. The features in center regions are stronger than previously and show higher contrast. The features reconstructed also confirm the optimal class images from Activation Maximization, showing high activations in the center for fault (see figure 75) and near the left and right border for *no fault* inputs (see figure 76) using Deconvolution.



(a) (b) (c) (d) (e)

Figure 75: Improved results of Deconvolution on the final layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

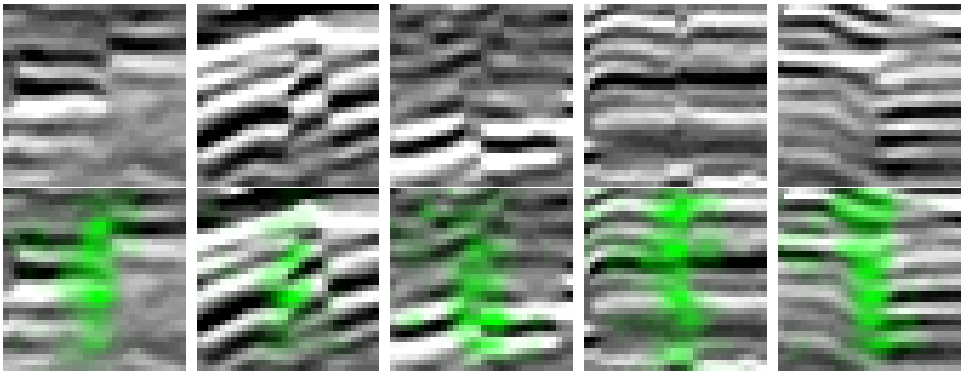


(a) (b) (c) (d) (e)

Figure 76: Improved results of Deconvolution on the final layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

Grad-CAM

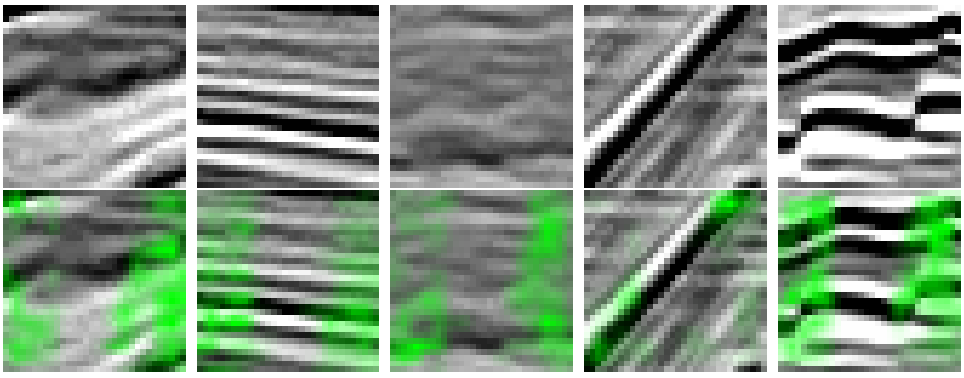
The heatmaps created by Grad-CAM expose that the important regions for classifying an input as *fault* lie completely on the vertical line through the center of the image (see figure 77). Note, that in figure 77b only the center fault is important for the decision, while the fault right to the center remains unnoticed. As with the sliding-window approach, only the center pixel of an input image is classified, this is the behaviour we want to evoke to get finer classifications of faults.



(a) (b) (c) (d) (e)

Figure 77: Improved results of Grad-CAM on the last convolutional layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

With Grad-CAM we can see that the most discriminative regions for *no fault* are apart from the center at the left and right border of the image. As depicted in figure 78e, a sample containing off-center faults is classified as *no fault* because there are no fault-like patterns in the center and therefore only the boundary regions are considered.



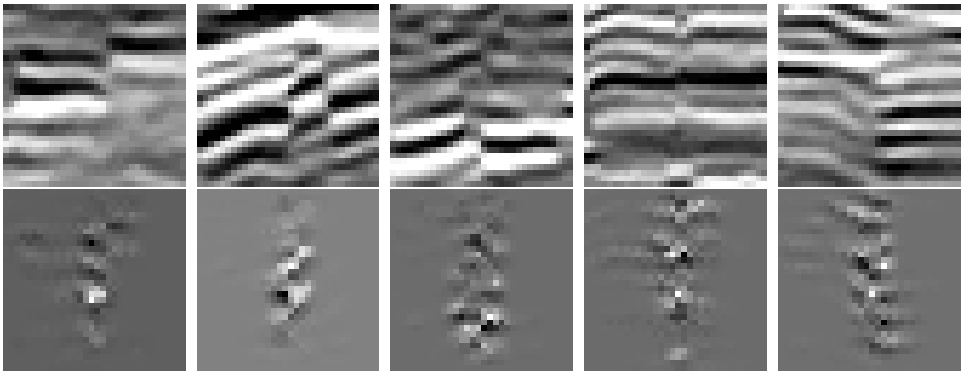
(a) (b) (c) (d) (e)

Figure 78: Improved results of Grad-CAM on the last convolutional layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

Guided Grad-CAM

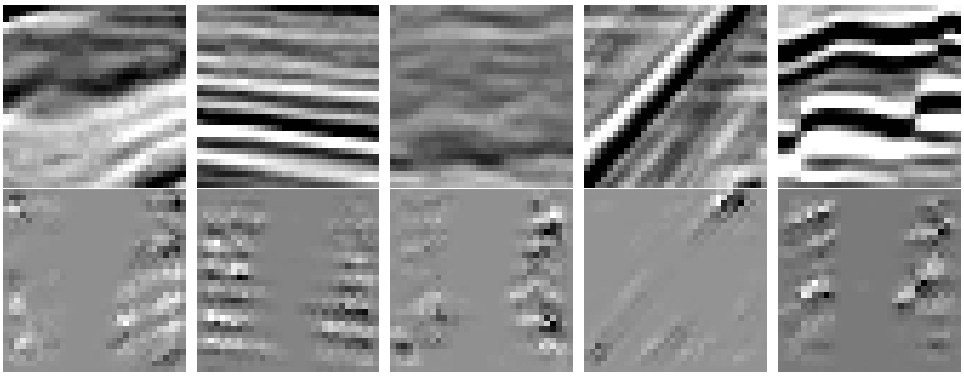
Visualizations of Guided Grad-CAM depicted in figures 79 and 80 also confirm the adaptation of CNN7 to the modified training data. Features leading to a *fault* classification (see figure 79) can only be found in the center

region, while previously this area has been significantly wider. Opposed to that, *no fault* classifications (see figure 80) are based on the whole image except the vertical line through the center.



(a) **(b)** **(c)** **(d)** **(e)**

Figure 79: Improved results of Guided Grad-CAM on the last convolutional layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

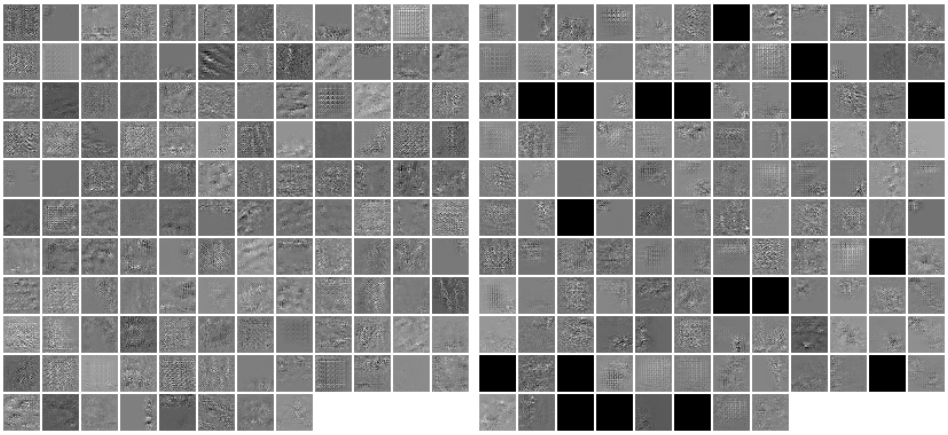


(a) **(b)** **(c)** **(d)** **(e)**

Figure 80: Improved results of Guided Grad-CAM on the last convolutional layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)

6.6.2 Network Architecture

During visualization of CNN12, noisy results with Activation Maximization visible in figure 81 were displayed in the higher convolutional layers 6 and 7. Most of the synthesized optimal input images show almost no structure except from noisy checkerboard patterns.



(a) **(b)**

Figure 81: Results of Activation Maximization for layer 6 (a) and 7(b) of CNN12

This leads to the assumption that 5 convolutional layers might be too much for the classification of images with a size of 32×32 pixels. Therefore, the architecture of CNN12 was adapted by removing layers 6,7 and 8 (2 convolutional, 1 max pooling layer). The classification results for CNN12 (left) and its modified version (right) are depicted in figure 82 which disproves the previously made assumption. The precision of the classified regions has not improved but worsened as many fault classifications have become larger and partly coalesced. To investigate the reasons for this misleading deduction, we take a look into more visualizations of the two networks.

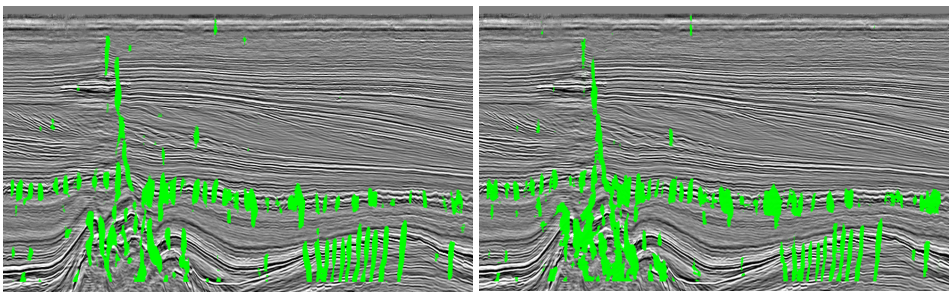


Figure 82: Classification results of CNN12 (left) and modified CNN12 (right)

A comparison between Activation Maximization results of CNN12 and modified CNN12 (see figure 83) from layer 4, which is the last convolutional layer in the modified architecture, reveals that the modified version has lost many features in this layer although the noisy layers have been removed. Also, many of the optimal input images for the modified network

stay completely black, which is an indicator that the calculated gradients are zero and therefore no optimization could be performed.

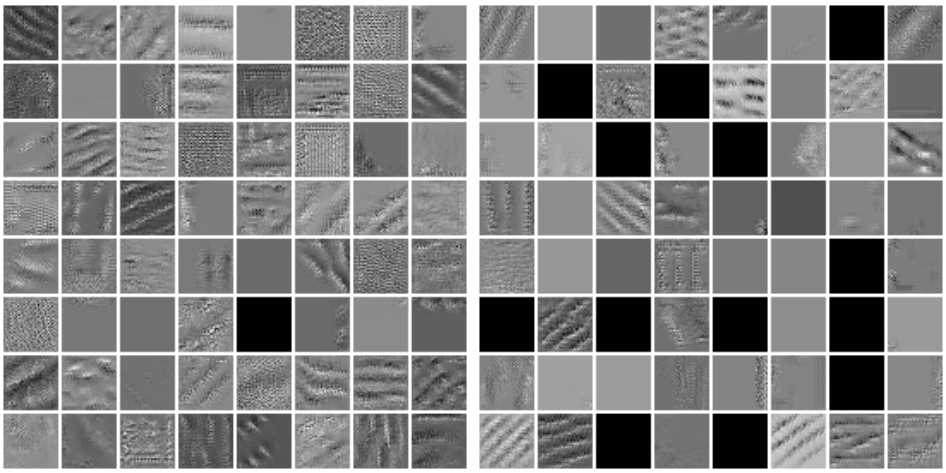


Figure 83: Results of Activation Maximization for layer 4 in CNN12 (left) and modified CNN12 (right)

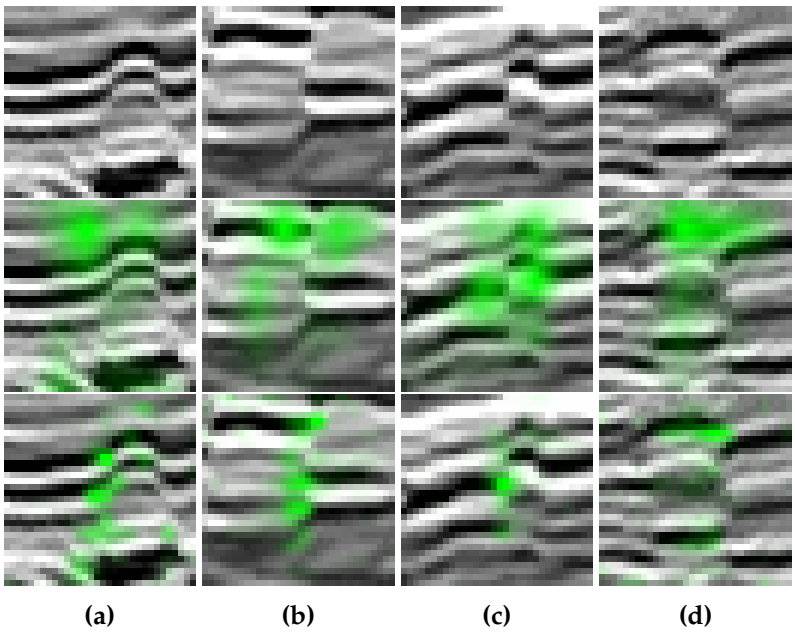


Figure 84: Results of Grad-CAM for CNN12 (middle row) and modified CNN12 (bottom row) with four input images (upper row)

The comparison of Grad-CAM results for both network architectures depicted in figure 84 shows much finer highlighted regions for the modified

network than for the original CNN12 with two more convolutional layers. This is due to the fact that the feature map of the removed layers has only half as much pixels per axis as the last feature map of the modified CNN12. Therefore, the upsampling from 8×8 to 32×32 pixels is much coarser and blurrier than from 16×16 pixels which results in a blurred and imprecise heat map and makes no statement about the quality of the classification results.

6.7 Discussion

By evaluating visualization results from seismic datasets made by DeepVis, it was discovered that at least some of the implemented visualization methods can be helpful to optimize the deep learning workflow for seismic classification purposes. Heatmaps obtained by Grad-CAM demonstrated that our CNNs reacted to fault features apart from the center. With this gained knowledge the training data could be optimized to better match the desired results and finally get more precise classifications.

The most valuable and impressive visualization method for users with or without a deep learning background seems to be Grad-CAM as it generates easily interpretable heatmaps on top of every input image. In other words, it highlights important regions for the predicted class directly in the input image and therefore leads to an understanding of the concepts that the CNN has learned. However, experiments with CNN12 revealed that in many-layer networks with max pooling layers Grad-CAM can deliver blurred and imprecise highlights that cover large areas of the image. This is due to the fact that the heatmap is generated from the last layer and has to be upsampled to match the input image dimensions. Thus, the use of Grad-CAM makes sense for CNNs that downsample its feature maps only a few times.

Activation Maximization syntehsizes optimal input images for every feature of an intermediate layer and for every class of the final output layer. The visualization with this method has shown that some features and patterns are recognizable especially in the optimal class images. However, a user still has to interpret the results because they do not represent real-life examples of training data. Also, a user must pay attention to draw proper conclusions from the visualizations, as demonstrated in section 6.6.2 where the results of Activation Maximization led to the assumption that CNN12 had to many convolutional layers. After removing the layers the classification results got worse because the noise in the visualization images in fact originated from upsampling instead of an insufficient architecture of the network.

Feature reconstruction with Deconvolution and Guided Backpropagation deliverd similar results, although images from Guided Backpropagation have a higher contrast and show finer and more detailed features. DeepVis is able to visualize features of intermediate layers as well as from the output layer, while in the evaluation only the latter have been shown. In figure 85 intermediate features of the last convolutional layer in CNN7 are depicted with the corresponding fault input image in the upper left corner. The intermediate features are hard to distinguish and very similar to each other, making it difficult to use them for understanding and optimization. However, in literature Deconvolution is a popular technique to visualize learned features of CNNs. Therefore, the assumption arises that especially Deconvolution does not work well for black-and-white images with a small size that we use in seismic classification.

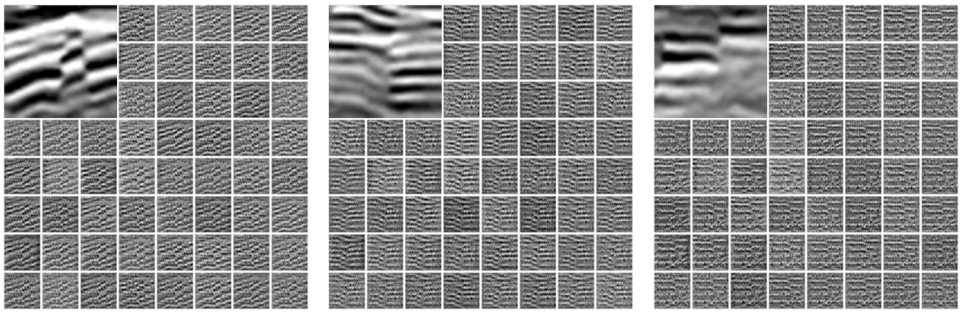


Figure 85: Intermediate features for faults with Deconvolution

Guided Backpropagation results are better than Deconvolution results concerning the visibility of features in the last layer but the approach of combining Guided Backpropagation with Grad-CAM seems to be the more viable solution as the fine-graded results are additionally localized showing only regions of high interest to the network. With Guided Grad-CAM unimportant features for the network are masked out while the most discriminative regions are highlighted.

In conclusion, the evaluation has proven that the availability of high-quality training data in large quantities is crucial for a high classification accuracy. In some cases, the labels have to be adapted to the use case, like when annotating the neighbourhood of every *fault* as *no fault* to achieve much finer classification results. With a combination of Activation Maximization, Grad-CAM and Guided Grad-CAM, a user has a rich toolkit for analyzing, understanding and optimizing the deep learning process with CNNs. DeepVis makes these tools available to users in their web browser and integrates the CNN visualization into the deep learning workflow for seismic interpretation.

7 Conclusion and Outlook

In this thesis, five methods for visualizing different aspects of convolutional neural networks have been presented with the purpose to get an insight into the decision-making process and open the black-boxes of neural networks. In order to use the visualizations and make them accessible for people in the oil and gas industry, a visualization tool called DeepVis was created and integrated into the DeepGeo platform. In addition to simple 2D CNNs, support for CNNs containing 3D convolutions has been implemented as this has been considered by experts to be particularly necessary in the oil and gas industry that uses large-scale three-dimensional datasets.

As CNNs are very complex structures, they are difficult to debug and hard to understand completely. By visualizing features from intermediate and final layers and revealing which concepts a network has learned, the black-box system is opened to some extend and its decisions become more comprehensible for a user. Activation Maximization, Grad-CAM and Guided Grad-CAM have proven to be particularly useful to understand what features in an input image lead to which decisions. With this knowledge optimization possibilities can be deduced to improve the training data or the network architecture as presented in section 6.6. However, a user still has to be cautious when interpreting those visualizations as they are always an abstraction of the complex processes inside a neural network and can sometimes mislead.

Although the automatic interpretation of seismic data with deep learning is only a small part of the oil and gas exploration process and therefore DeepVis and DeepGeo represent only one link in a long chain of tools and processes to eventually find hydrocarbon reservoirs, the use of deep learning will increase in the next years and will be a major research topic for the oil and gas industry as well as many others. Classification results can be further improved with post-processing methods or the combination of several seismic datasets and perspectives with highly precise labelling.

DeepVis cannot make every single calculation of the complex classification process visible and comprehensible as the amount of processed data and variables is too large. However, DeepVis represents a first step towards a better understanding of deep learning and the use of hybrid artificial intelligence in the seismic interpretation workflow. A computer's precision and its ability to handle large amounts of possibly monotonous data combined with a human as supervising authority that can contribute domain knowledge and a critical look from outside will be of great importance in the future.

As the quality of training data has proven to be of great importance for

a good performance of deep learning approaches, one idea for future research is the clustering of similar samples in the seismic training datasets. When DeepVis reveals a strong reaction to a certain kind of fault for example, the training data could be searched for all similar-looking samples to directly reveal which structures are detected best by the CNN and possibly deduce further optimizations from this. This represents an extension to Activation Maximization, in which images are found in the training data that maximize the activation of a class instead of synthesizing optimal inputs.

Because the training data consists of many small and similar-looking black-and-white images, the problem of a strong adaption to the training data, called overfitting, plays a major role in automatic seismic classification. So far, CNNs trained to detect seismic structures do not adapt well enough to new volume datasets as they are too overfitted. Common regularization methods like dropout and L2 regularization have shown some improvements but are not capable enough in this context. Therefore, the overfitting problem is also a challenging future research topic.

In conclusion, DeepVis is a promising solution to make decisions of CNNs more comprehensible to users and to find optimization possibilities for networks with poor performance. More visualization algorithms and optimizations of methods and results in the automatic seismic classification workflow are subject to further research.

List of Figures

List of Tables

Bibliography

[1] DEA Deutsche Erdoel AG. Data models and their interpretation, 2018. https://www.dea-group.com/en/technology/exploration/data-models (accessed: 30 December 2018).

[2] DEA Deutsche Erdoel AG. Seismics, 2018. https://www.dea-group.com/en/technology/exploration/seismik (accessed: 30 December 2018).

[3] Jan Bender. Intuitive Erstellung von Störungsgeometrien in dreidimensionalen seismischen Volumendaten. Bachelor Thesis, Technische Hochschule Köln, 2016.

[4] Jan Christoph Beutgen. Interaktive Volumenvisualisierung mit multidimensionalen Transferfunktionen. Bachelor Thesis, University of Koblenz-Landau, 2015.

[5] Jan Christoph Beutgen. Semi-Automatic Feature Detection in Volume Data through Anomalies in Local Histograms. Master Thesis, University of Koblenz-Landau, 2017.

[6] Franck Bouttemy. Fault in the Carboniferous soils, 2013. https://www.geodiversite.net/media1015 (accessed: 07 November 2018).

[7] dGB Earth Sciences B.V. Netherlands Offshore F3 Block - Complete, 1987. https://terranubis.com/datainfo/Netherlands_Offshore_F3_Block_-_Complete (accessed: 30 December 2018).

[8] dGB Earth Sciences B.V. Attributes Table, 2018. https://dgbes.com/index.php/software/attributes-table/ (accessed: 30 December 2018).

[9] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.

[10] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

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

[12] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[13] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[14] David H Hubel and Torsten N Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160(1):106–154, 1962.

[15] Ying Jiang. Detecting geological structures in seismic volumes using deep convolutional neural networks. Master Thesis, RWTH Aachen, 2017.

[16] Andrej Karpathy. CS231n: Convolutional Neural Networks for Visual Recognition, 2018. `http://cs231n.stanford.edu/` (accessed: 30 November 2018).

[17] Philip Kearey, Michael Brooks, and Ian Hill. *An introduction to geophysical exploration*. John Wiley & Sons, 2013.

[18] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[19] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[20] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[21] New Zealand Crown Minerals. Parihaka-3D. `https://wiki.seg.org/wiki/Parihaka-3D` (accessed: 30 December 2018).

[22] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[23] Greg O'Beirne. Waimakariri River, Canterbury, New Zealand, 2007. `https://commons.wikimedia.org/wiki/File:Waimakariri01_gobeirne.jpg` (accessed: 07 November 2018).

[24] OMV. What is seismic reflection?, 2018. `https://www.youtube.com/watch?v=NJ4RhpKCePM` (accessed: 07 November 2018).

[25] Zhuwei Qin, Funxun Yu, Chenchen Liu, and Xiang Chen. How convolutional neural network see the world - a survey of convolutional neural network visualization methods. *arXiv preprint arXiv:1804.11191*, 2018.

[26] Herbert Robbins and Sutton Monro. A Stochastic Approximation Method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[27] Frank Rosenblatt. Principles of neurodynamics. perceptrons and the theory of brain mechanisms. Technical report, CORNELL AERONAUTICAL LAB INC BUFFALO NY, 1961.

[28] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[29] Schlumberger. Oilfield glossary, 2018. `https://www.glossary.oilfield.slb.com/` (accessed: 07 November 2018).

[30] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra, et al. Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization. In *ICCV*, pages 618–626, 2017.

[31] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, Dhruv Batra, et al. Grad-CAM: Gradient-weighted Class Activation Mapping, 2018.

[32] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[33] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, and Martin Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

[34] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[35] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016.

[36] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[37] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579*, 2015.

[38] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[39] Matthew D Zeiler, Graham W Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2018–2025. IEEE, 2011.

[40] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning deep features for discriminative localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2921–2929, 2016.

A Visualization Results

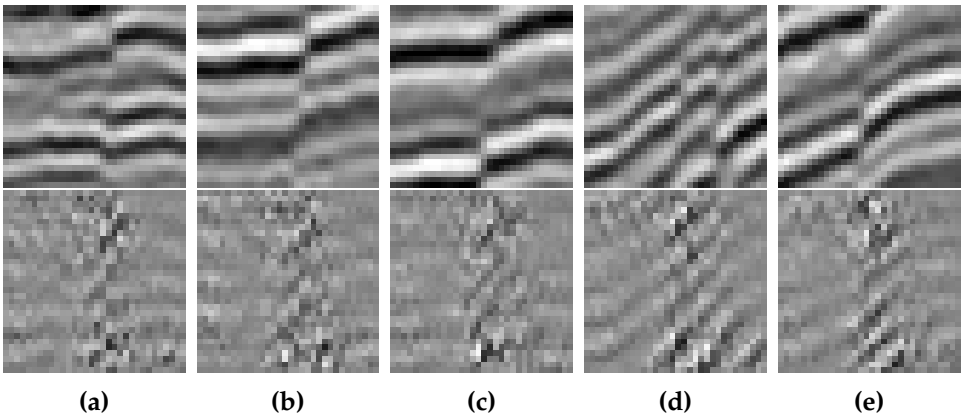


Figure 86: Results of Deconvolution on the final layer (bottom) for class *fault* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)

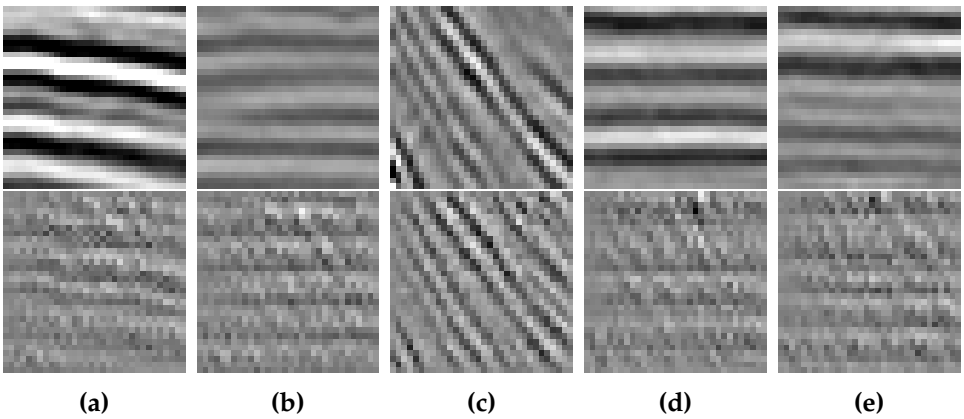
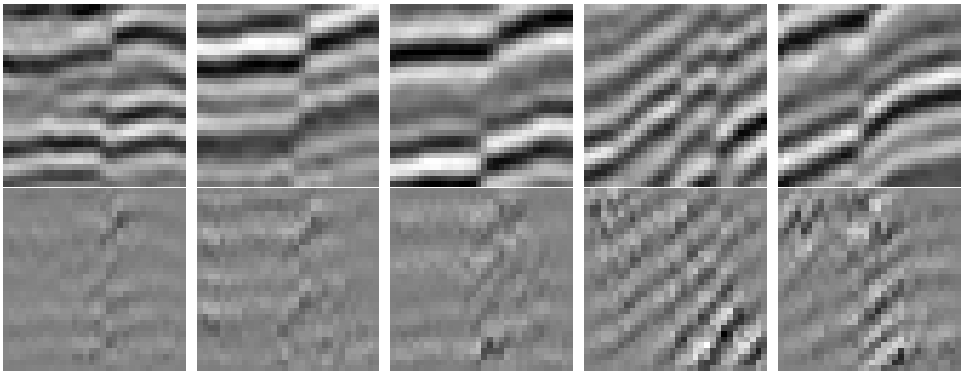
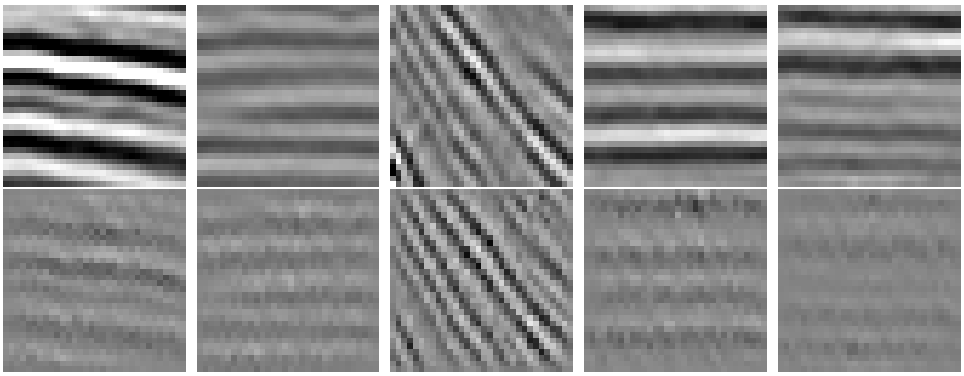


Figure 87: Results of Deconvolution on the final layer (bottom) for class *background* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)



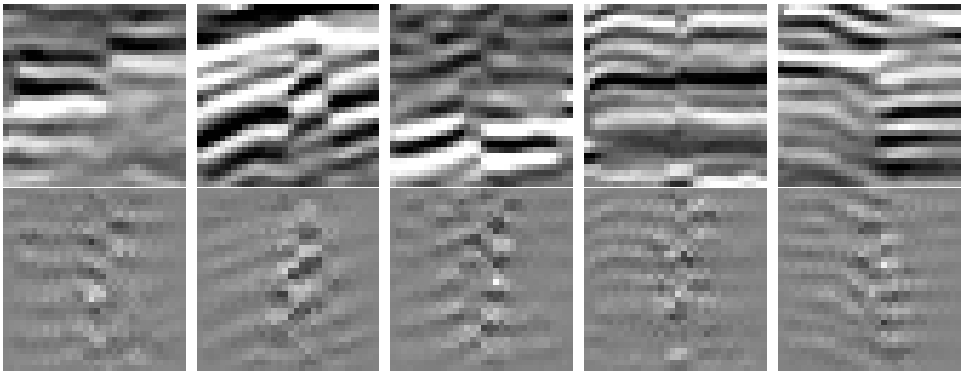
(a) **(b)** **(c)** **(d)** **(e)**

Figure 88: Results of Guided Backpropagation on the final layer (bottom) for class *fault* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)



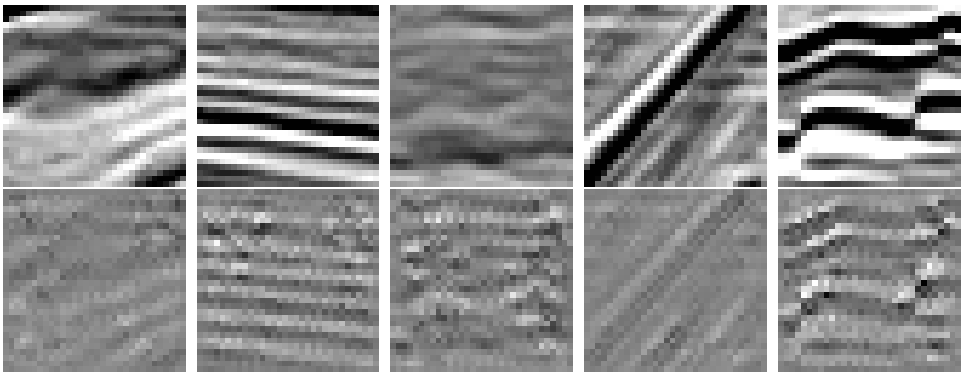
(a) **(b)** **(c)** **(d)** **(e)**

Figure 89: Results of Guided Backpropagation on the final layer (bottom) for class *background* and corresponding input images (top) in Parihaka dataset (CNN7, input size $32 \times 32 \times 1$)



(a) **(b)** **(c)** **(d)** **(e)**

Figure 90: Improved results of Guided Backpropagation on the final layer (bottom) for class *fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)



(a) **(b)** **(c)** **(d)** **(e)**

Figure 91: Improved results of Guided Backpropagation on the final layer (bottom) for class *no fault* and corresponding input images (top) in F3 dataset (CNN7, input size $32 \times 32 \times 1$)