Convolutional neural networks for automated seismic interpretation

Anders U. Waldeland[1], Are Charles Jensen[1], Leiv-J. Gelius[2], and Anne H. Schistad Solberg[1]

https://doi.org/10.1190/tle37070529.1.

Abstract

Deep-learning methods have proved successful recently for solving problems in image analysis and natural language processing. One of these methods, convolutional neural networks (CNNs), is revolutionizing the field of image analysis and pushing the state of the art. CNNs consist of layers of convolutions with trainable filters. The input to the network is the raw image or seismic amplitudes, removing the need for feature/attribute engineering. During the training phase, the filter coefficients are found by iterative optimization. The network thereby learns how to compute good attributes to solve the given classification task. However, CNNs require large amounts of training data and must be carefully designed and trained to perform well. We look into the intuition behind this method and discuss considerations that must be made in order to make the method reliable. In particular, we discuss how deep learning can be used for automated seismic interpretation. As an example, we show how a CNN can be used for automatic interpretation of salt bodies.

Introduction

Interpretation of seismic data is an important part of the seismic workflow, but it can be tedious and difficult. Due to large amounts of 3D data in modern seismic, manual interpretation is highly time-consuming and subject to human bias. Tools for automated seismic interpretation can speed up the process and, to some extent, reduce the subjective bias.

Convolutional neural networks (CNNs) have become increasingly popular for solving problems in image analysis (Goodfellow et al., 2016). CNNs now hold the record in the ImageNet Large Scale Visual Recognition Challenge (Russakovsky et al., 2015), pushing the error rates for classification from 25.8% (2011) to 3.0% (2016). CNNs differ from traditional feedforward neural networks, which have long been used to analyze seismic attributes (van der Baan and Jutten, 2000). CNNs are applied directly to the input image, not to a set of attributes/features. During the training phase, the network learns how to compute the best attributes to solve a given task. This makes CNNs fundamentally different from many classical image analysis approaches in which feature selection/engineering is an integral part of the methodology.

Despite successful applications within natural and medical image analysis, only a few publications exist in which CNNs are applied to seismic data (e.g., Araya-Polo et al., 2017; Huang et al., 2017). However, we expect an increased interest in this topic due to the success of applying CNNs for natural image analysis. CNNs are complex and may appear as a black-box algorithm where the user lacks intuition of what is happening inside the algorithm. As there are many details that need to be correct, using CNNs as a black-box algorithm can easily result in overfitting or make the algorithms not perform as well as expected.

In this paper, we explain CNNs and how the different parts of CNNs work together to solve a given classification task. We give examples with natural images to better explain the general concepts. We then explain how CNNs are commonly trained in a robust and efficient manner. We will discuss how this method can be used in the context of seismic interpretation. As an example, a CNN is applied to seismic data for salt body classification. Finally, we discuss how this method can be applied to other problems in automated seismic interpretation and what limitations can be expected when applying CNNs to seismic images.

Convolutional neural networks

To explain how CNNs work, we will consider face detection as an example. In a CNN, classification problems are solved by dividing a problem into multiple smaller problems. We know that a face consists of two eyes, a mouth, a nose, and two ears. The nose, eyes, ears, and mouth can be decomposed into primitive features like edges, curves, circles, or blobs of colors. These objects can be detected using simple edge detectors with different orientations or Gabor-like filters (Gabor, 1946). The CNN utilizes this hierarchical structure of objects by first detecting low-level features and then putting them together to detect higher level features (Figure 1).

We recognize that objects other than faces can be decomposed into the same low-level features. CNNs use the same detectors to find the shared simple features and then build specific detectors for the different classes on top of these. We observe this property in the Alex-net (Krizhevsky et al., 2012) in which simple features are put together to form higher level features (Figure 2).

CNN architecture for classification. A CNN mainly consists of two types of layers — convolutional layers and fully connected layers. For classification networks, it is common to first have some convolutional layers followed by some fully connected layers (Figure 3). The convolutional layers act as feature extractors, and the fully connected layers act as a classifier. When the network is trained, it is the (filter) weights of the convolutional and fully connected layers that are optimized. This means we do not have to specify how the object classes should be detected; the network will learn how to do this from the training data.

Convolutional layers. The convolutional layers in the neural network consist of nodes that perform a convolution between an input image and a filter (Figure 4a). The convolutions act as feature extractors since each filter is tuned to extract a specific characteristic from the image. The nodes in the first convolutional layer receive the raw image as their input. Each node applies one filter to the input image, which generates one channel in the

[1]University of Oslo, Department of Informatics, Oslo, Norway. E-mail: anders.u.waldeland@gmail.com; arej@ifi.uio.no; anne@ifi.uio.no.
[2]University of Oslo, Department of Geosciences, Oslo, Norway. E-mail: l.j.gelius@geo.uio.no.

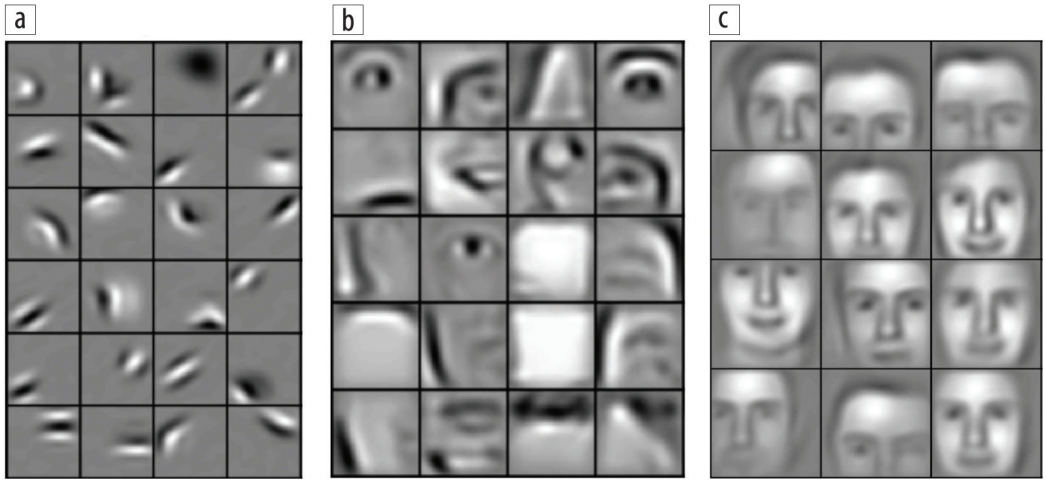

Figure 1. A visualization of how faces may be represented in a CNN. We observe that (a) edges with different orientations form (b) eyes, noses, and mouths that can be used to represent (c) faces. This figure originally appeared in Lee et al. (2011) and is printed with permission from the author of the original article.

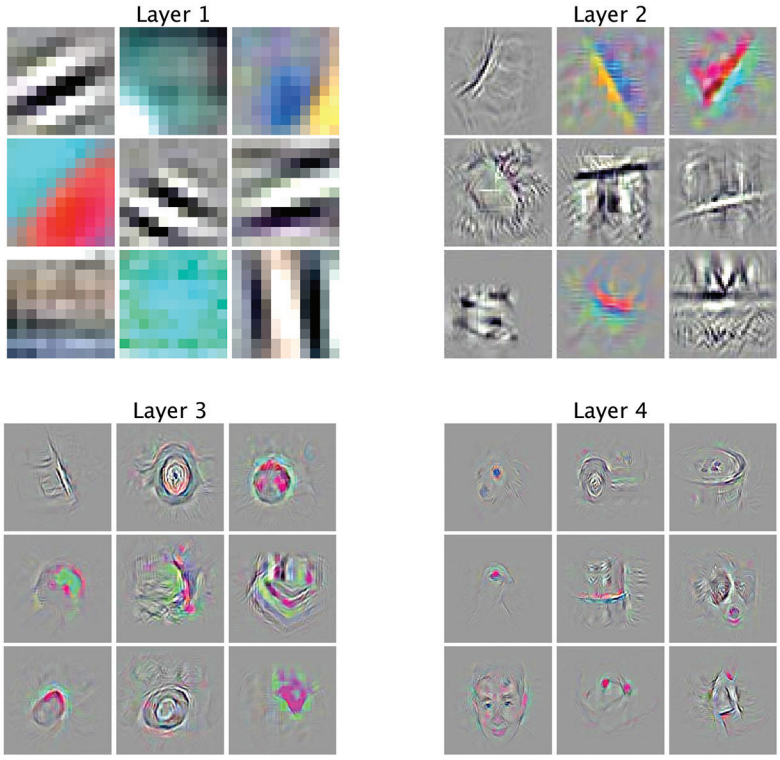

Figure 2. Filter responses for some of the nodes of the first four layers of the Alex-net (Krizhevsky et al., 2012) visualized using the technique by Zeiler and Fergus (2014). This figure originally appeared in Zeiler and Fergus (2014) and is printed with permission from the authors of the original article.

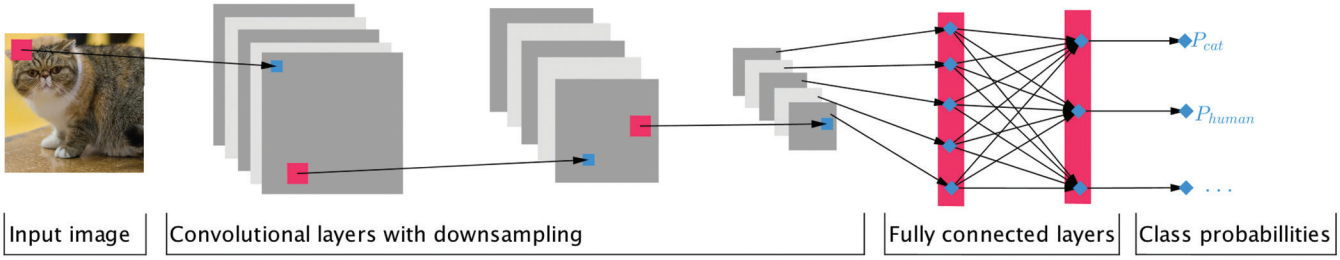

Figure 3. A CNN applied for image classification. The input is an image, and the output is a probability vector in which each element contains the estimated probability of the image containing an object of the given class.

multichannel output image from this layer. Each channel in the output image highlights a different feature in the input image depending on the filter (Figure 5).

In the second layer, multichannel filters are applied to the output of the first layer (Figure 4b). The nodes are analyzing the simple features detected in the first layer and combining them to form higher level features.

We can describe these operations mathematically. The input image to an arbitrary layer i is denoted by $X_i(x,y,c)$ and a filter j in this layer is denoted by $W_{i,j}(x,y,c)$. Here, x and y are the image coordinates, and c is the channel. The input image to layer i has C_{i-1} channels, corresponding to the number of filters in the previous layer. The output channel j from a convolutional layer is then

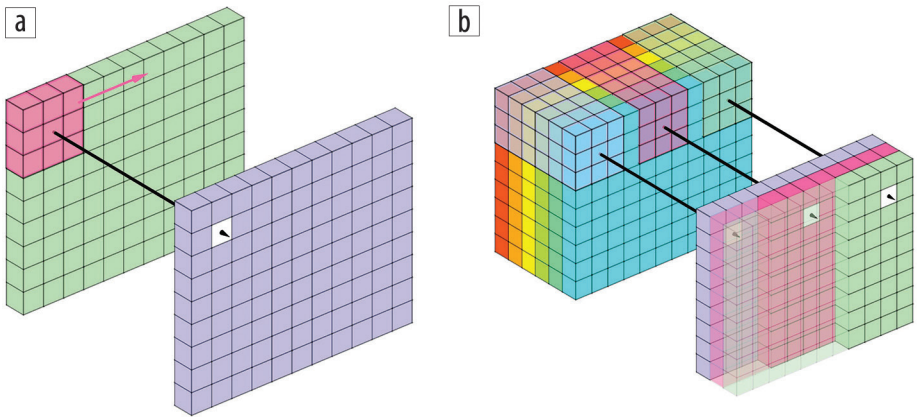

Figure 4. (a) A convolution between the input image (green) and a single filter (red). The filter is slid across the input image to compute the output image. (b) If the previous layer had multiple filters the input image has multiple channels. The nodes, therefore, apply multichannel filters to the input image. In this case, the layer has three filters, which produce three output channels.

$$X_{i+1}(x,y,j) = f\left(\sum_{k=1}^{C_{i-1}} X_i(x,y,k) * W_{i,j}(x,y,k)\right), \quad (1)$$

where $*$ is the 2D convolution operator in x,y space and f is a nonlinear activation function. We use zero padding on the input image so that the output image has the same size.

The activation function introduces nonlinearity to the network. One of the most common activation functions is the rectified linear unit (RELU) (Nair and Hinton, 2010):

$$f(x) = \max(0,x). \quad (2)$$

Conceptually, we can think of it as a soft classifier that is applied to the extracted feature image. The RELU is determining if the previous filter detected the given feature (input is higher than zero) and, if detected, returns how strongly the feature is present. By applying convolution plus activation, each node becomes a small feature extractor and classification unit.

Between some of the convolutional layers, the spatial size of the images is sometimes reduced by downsampling or pooling. This gradually introduces spatial invariance as it helps the network to move information from the spatial arrangement of pixel values (image domain) to features containing information relevant for the classification task (feature domain).

Fully connected layers. After the final convolutional layer, we have hopefully detected high-level features that will help identify the class of the object. The next step is to predict the class of the image based on these features. For example, if we have detected a mouth and eyes, we can predict that the image contains a human face. If we have detected eyes and fur, we can predict that the image contains an animal. Such hierarchical decision rules are modeled in the fully connected part of the network.

The output vector from the final convolutional layer is the input to the first fully connected layer. Each of the subsequent layers receives the output vector from the previous layer. Each node applies a weight to each of the input elements, sums, and applies the activation function. The output of the final fully connected layer is a vector ($\mathbf{x}$) with one element for each class. Each element in this vector then indicates the probability of the input image belonging to a given class. Finally, the output is subject to a softmax function to scale and normalize the probabilities:

$$y_i = \frac{e^{x_i}}{\sum_k e^{x_k}}, \quad (3)$$

where the summation runs over all elements in $\mathbf{x}$.

If we consider the fully connected part of the CNN, we see that this is the same as a conventional (nonconvolutional) neural network.

Training the network

Now that we have investigated how a trained CNN can solve classification tasks, the most prominent question is: how is the network trained? Defining the filters manually is not an option since there might be thousands of filters in a network. Instead, we train the CNN by iteratively updating the filter weights to minimize the error on the training set. For simplicity, we use the term *weights* for all tunable parameters in the network. This includes the filter coefficients in the convolutional layers and the weights in the fully connected layers.

Cost function. The network is trained by minimizing a differentiable cost function. The cost function is used instead of the error rate because the error rate is discontinuous and will be harder to minimize numerically. Because the softmax function gives a "soft" label with values between 0 and 1, the output of this layer is used to construct a continuous cost function. A common cost function, especially for multiclass problems, is cross entropy:

$$E = -\sum_{\forall j} y'_j \log(y_j), \quad (4)$$

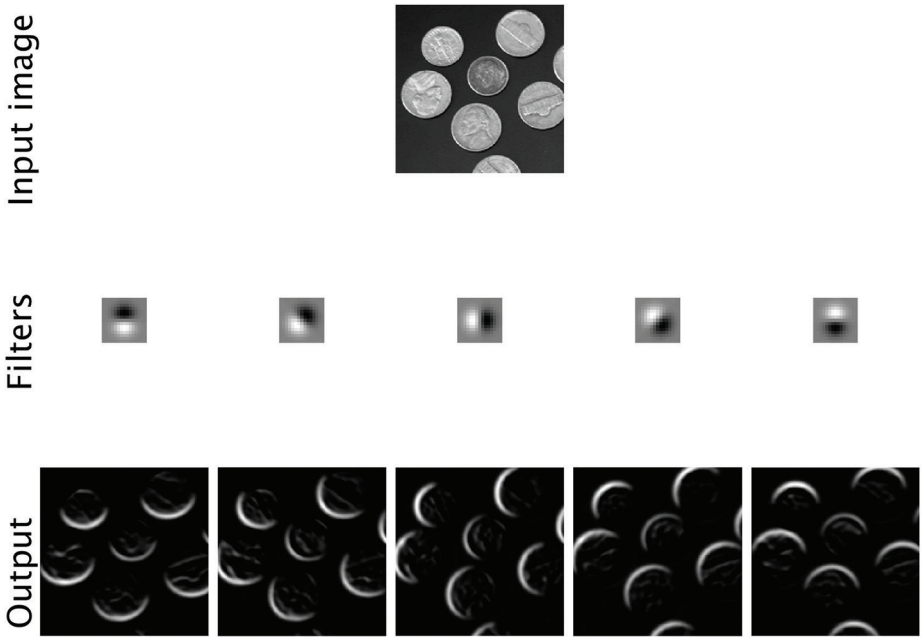

Figure 5. An input image is convolved with orientation filters. This produces new images where edges with different orientations are highlighted.

where y_j is the (predicted) softmax output for class j, and y_j' is 1 if the (true) label for the sample belongs to class j and 0 otherwise.

Stochastic gradient descent and back-propagation. An efficient method for minimizing the cost function is by using the stochastic gradient descent (SGD) method with (mini) batches. A batch is a small random subset of the training set. Batch sizes typically vary between eight and 256 samples. Iteratively, a random batch of samples is drawn from the full training set and fed through the network. Using the method of back propagation (Rumelhart et al., 1986; Simard et al., 2003), the gradients of the cost function with respect to the weights are computed. The gradients are efficiently computed using vectorization and averaged over the samples in the batch to approximate the gradients for the full data set ($\widehat{\frac{\partial E}{\partial w_i}}$). Each weight ($w_i$) is then updated by taking a step in the direction of the negative gradient:

$$w_i^* = w_i - \tau \widehat{\frac{\partial E}{\partial w_i}}. \quad (5)$$

The τ parameter is the learning rate and controls how much the weights are updated at each iteration and is often set to a fixed value. Adaptive schemes for determining τ can lead to quicker convergence (i.e., Kingma and Ba, 2015).

Training, validation, and test set. Due to the high number of weights, CNNs are prone to "overfitting." Overfitting occurs when the network becomes so specialized in classifying the samples in the training set that it does not perform well on new samples. To detect possible overfitting, we need to withhold some data from the training set to use for validation. If the error on the training data decreases but the error rate on the validation set increases, it is a sign of overfitting. If this happens, we must decrease the number of layers or nodes in our network and/or apply some regularization to the network. When the final network is trained, including experiments with different network architectures and hyperparameters, a third independent test set is used to obtain a final error estimate. This indicates how well the network will perform on new previously unseen data.

Recommendations for successful training

Training neural networks robustly without overfitting has always been a major problem. This is also true for CNNs, and many solutions have been proposed to improve the training (Bengio, 2012). Here we have collected some of the most important methods and recommendations for training modern CNNs successfully.

Large amount of training data. Training large networks requires large amounts of training data to avoid overfitting. The amount of training data needed is always relative to the number of weights in the model. If we have a small model, we can avoid overfitting even with a small training set. However, to solve a complex classification task, a larger model (and thus more training data) may be needed.

Initialization of weights. The initial values for the weights are important. If all filter coefficients are set to the same value (for example zero), all filters will be trained using similar gradients and become almost identical. In addition, the initialization also affects how fast the network is trained. It is common to initialize the weights with random values — i.e., by the approach proposed by Glorot and Bengio (2010).

Choice of activation function. Conventional neural networks often use a sigmoid or tanh function as the activation function. These activation functions may cause the network to converge slowly. Modern deep networks often use the RELU activation function (equation 2) or variants of this. By using such activation functions, the networks are often trained faster.

Batch norm. When training a deep network, the weights become increasingly dependent on weights in previous layers. When a weight changes in a shallow layer, it may imply a large change in the input to a deeper layer (covariate shift). This makes it hard to train the weights in deeper layers. The batch norm algorithm (Ioffe and Szegedy, 2015) has proven to be a powerful strategy to cope with this challenge.

Balancing class labels. When training a network with SGD, the weight updates will be biased if the different classes are unevenly represented in the training set (Hensman and Masko, 2015). To avoid this, we have to either compensate for uneven distributions when computing the cost function or ensure that the batches (on average) have an equal number of samples from each class.

Dropout. Dropout (Srivastava et al., 2014) is another popular regularization technique designed to prevent overfitting

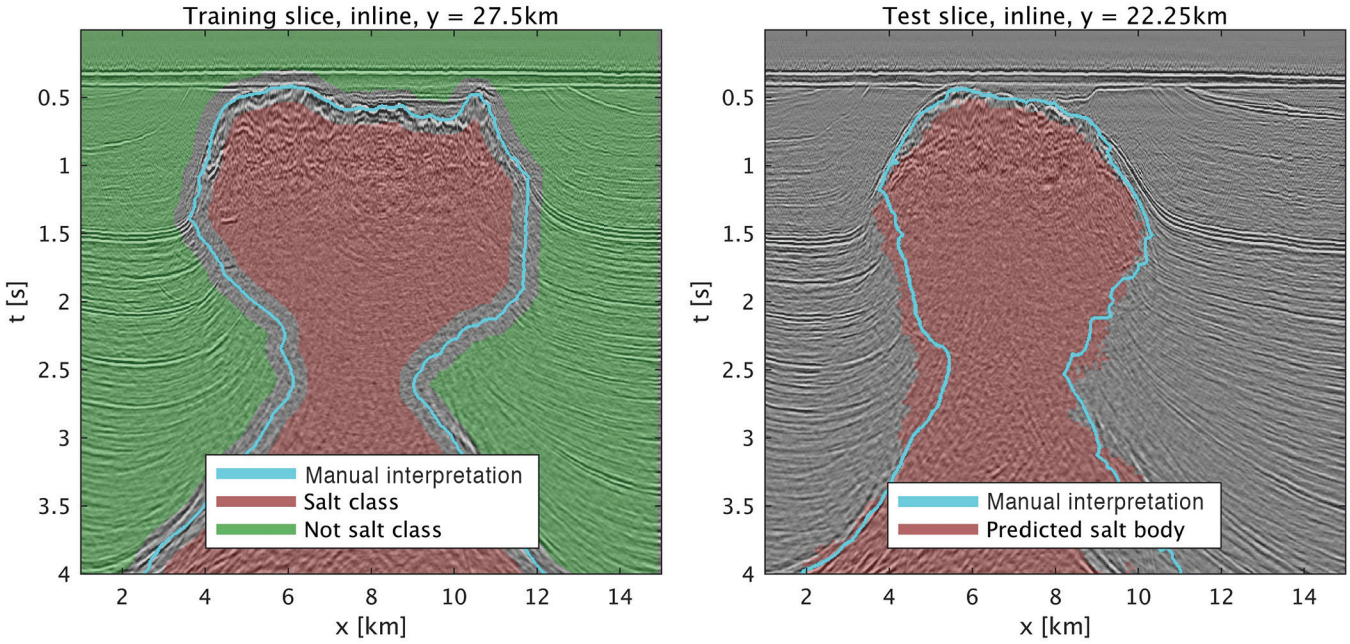

Figure 6. (a) The inline slice used for training. The manual interpretation was reproduced from Rojo et al. (2016). (b) The test slice. No postprocessing was applied to the predicted salt pixels.

during training. During the training phase, 50% of the nodes of each layer are randomly dropped (put to zero) at each iteration. The weights in each of the nodes then become less dependent on the other nodes, which prevents overfitting. It also forces the network to learn redundant features and increases the robustness.

Augmentation. Given that we have gathered the largest training set practically possible, random augmentation can be used to simulate a larger training data set. Small random geometrical transforms (rotation, flipping of axes, scaling, etc.) are added to the training images, which artificially generate more training samples. The technique prevents the network from overfitting to a limited set of training examples and makes it invariant to the types of augmentation that are applied. The amount of augmentation should be chosen such that we get samples with as much variety as possible but should not alter a sample so much that it is not representative of its original class.

Example: Salt classification

In this section, a CNN is applied in the context of automated seismic interpretation for delineation of salt bodies. Interpretation of salt bodies is of interest for two reasons. Due to the low permeability associated with salt bodies, they may form seals for reservoirs. In addition, they have a relatively high sound velocity, which makes it important to obtain an accurate velocity model in the vicinity of the salt bodies.

Data set and network configuration. In our example, we use a 3D data set acquired in the Barents Sea at the 7228/7 block, which contains one salt wall and four salt stocks (Rojo et al., 2016). We define two classes — "salt" and "not salt." We select small 65 × 65 × 65 cubes of seismic amplitudes from the full cube. The goal is to have the network predicting the class of the center pixel of the small cubes.

The network is trained on one manually labeled inline slice (Figure 6a). We select 3D cubes around the pixels in this slice (including amplitudes from the neighboring inline slices) and use these cubes as training samples. When the network has been trained, we go through the full 3D volume and select all possible 65 × 65 × 65 cubes and apply the network to predict the class at each location.

We used the network configuration presented in Figure 7. This is a further development of the network we proposed in Waldeland and Solberg (2017). Batch norm was applied before the RELU activations. The network was trained for 2000 iterations, with a batch size of 32 with 16 samples from each class. Having only one training slice does not give enough training data to avoid overfitting. To remedy this, we applied random augmentation to simulate a larger training set. The augmentation was random scaling (± 20%), random flipping of the nondepth axes, random rotation (± 180°), and random tilting (± 15°). The chosen network and training configurations were a result of some experimentation.[3] The training lasted for approximately 20 minutes on a Nvidia Titan GPU (2013 model).

Results and discussion. To assess the quality of the automatic interpretation, we compare it with a manual interpretation of an inline test section (Figure 6b). The automatic interpretation generally coincides well with the interpreted salt body. At the flanks, the manual interpretation is slightly more conservative than the

[3]Initially, we investigated how the number of layers and number of nodes affected the performance. The results were very similar for network configurations with everything between four and 10 layers and with 40–80 nodes in each layer. The network seemed to converge after 1000–2000 training iterations and did not improve notably when training for more iterations. (Up to 20,000 iterations were tested.) We also tested dropout, max, and average pooling, ELU activation function, but the results did not change much. The network did not perform well, however, without augmentation and batch norm.

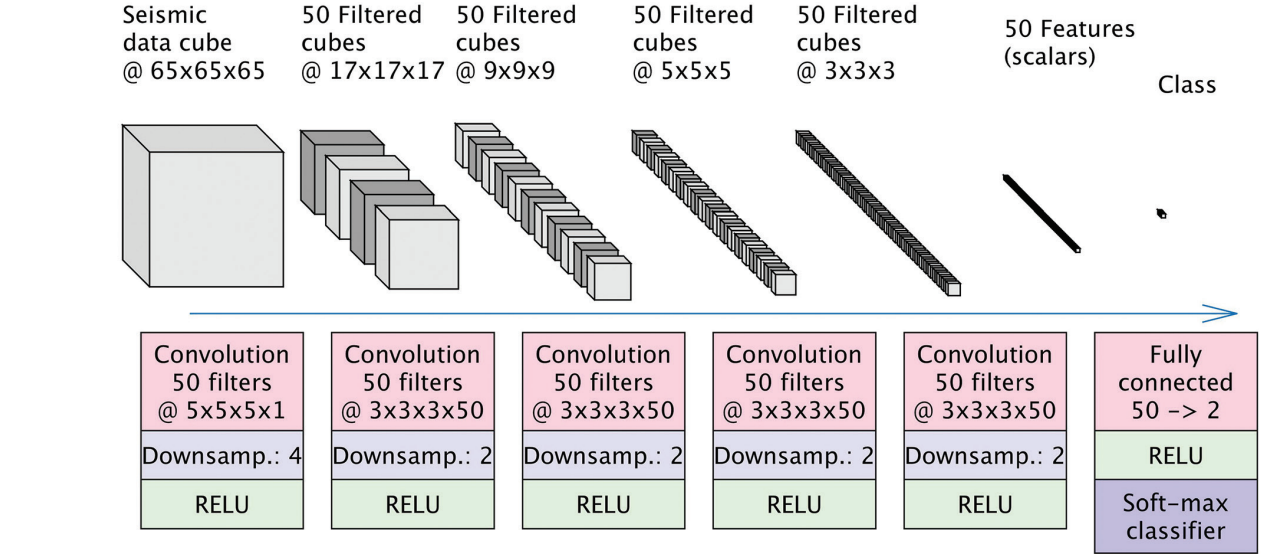

Figure 7. The full 3D cube is partitioned into small amplitude cubes of 65 × 65 × 65 samples, which are input into the salt classification network. The network then predicts the class of the center pixel.

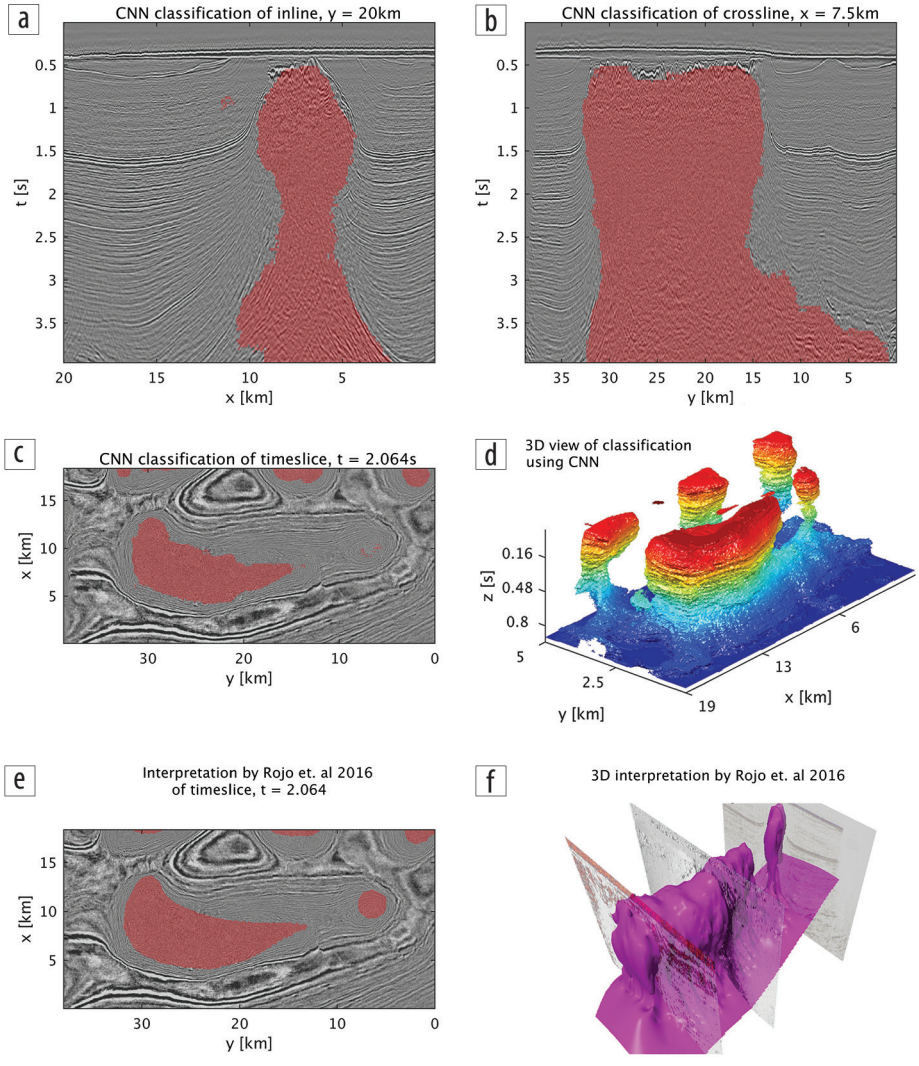

Figure 8. (a–c) The predicted salt body is marked with red. (d) The predicted full 3D salt body is visualized with color, indicating the time. No postprocessing was applied to the predicted salt pixels. Panels (e) and (f) show the manual interpretation done in Rojo et al. (2016) and are printed with permission from the authors of the original article. Note that the manual 3D interpretation was conducted only for the salt wall and one of the four salt stocks.

automatic interpretation using the CNN. It should be noted that it is not trivial to determine the boundaries at the salt flanks and that manual interpretations might vary from interpreter to interpreter. The resulting classification of the full 3D data set shows that the network has successfully delineated the salt wall and the four salt stocks (Figure 8). Although the automatic interpretation is somewhat more conservative for the four salt stocks (visible in the time slices), the automatic interpretation is very close to the manual 3D interpretation. In our example, we only trained on one inline containing the salt wall. It is likely that by including training slices from the four salt stocks, the classification would improve.

Learned attributes. We can construct sections with the learned attributes to gain insight into how the classification task is solved. To compute the attribute sections, we select the small 3D cube for a given center pixel and run it through the network. The output from a given node can then be collected as the attribute value at this location. Figure 5 shows five of the learned attributes for the test slice. The first attribute (Figure 9a) is sensitive to horizontally layered geology. This is a good indicator of regions outside the salt but does not give a good separation close to the boundary. In these regions, we often have dipping salt flanks. The next attributes (Figures 9b and 9c) are sensitive

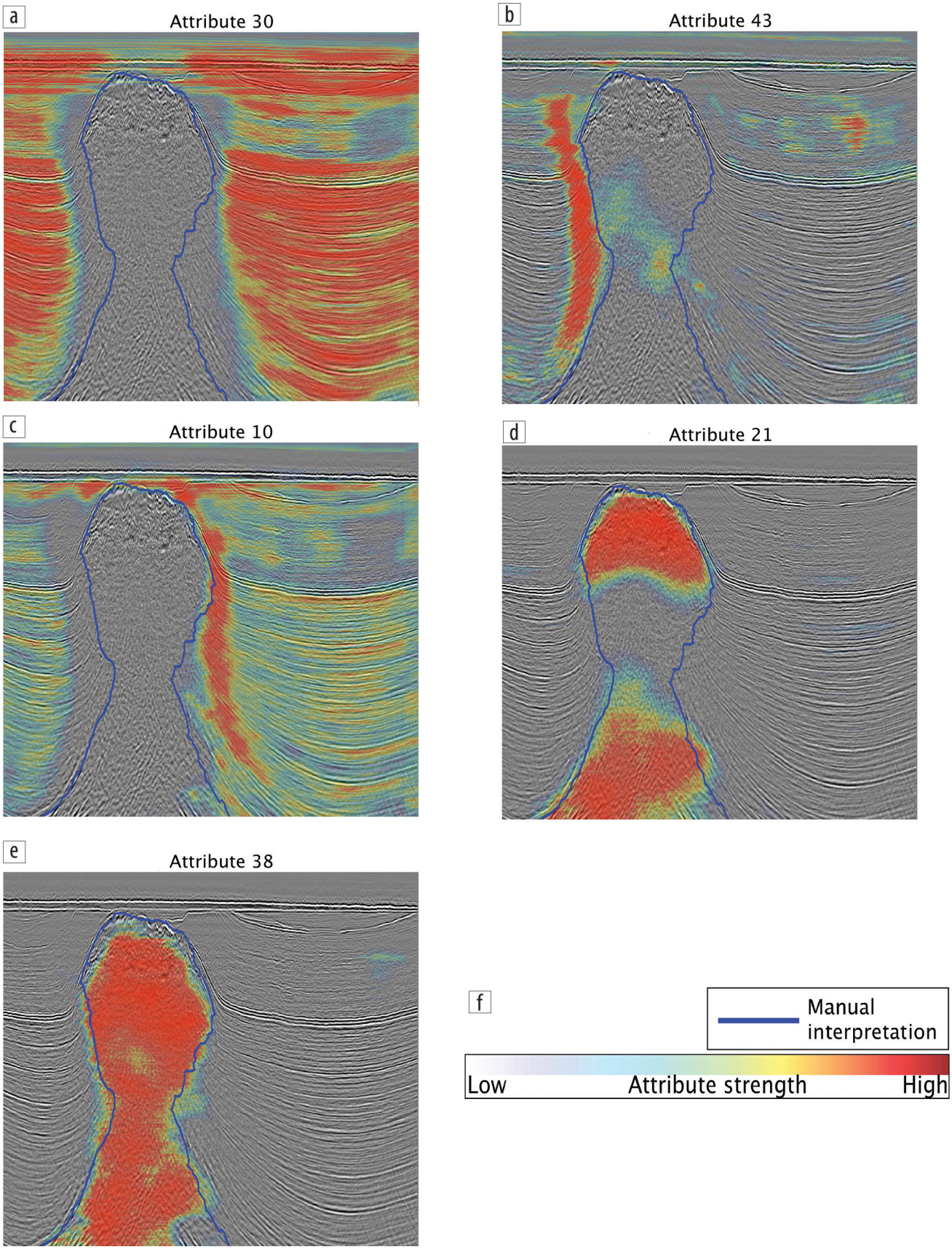

Figure 9. Some of the learned attributes from the last convolutional layer computed for the test inline slice.

to the dipping reflectors and complement the first attribute. The attributes in Figures 9d and 9e are sensitive to different parts of the salt body. In the final fully connected layers, these attributes are combined to give the final classification.

Implications of moving from natural to seismic images

One difference between natural images and seismic data is that the objects we typically want to detect in seismic images are less complex than objects we want to detect in natural images. This means the network architecture needed to solve seismic interpretation problems can have fewer layers than the architectures used for natural images, which is beneficial because smaller networks are more easily trained with less data.

Much of the success of CNNs has come due to efficient implementation of convolutions on powerful GPUs, which have made it possible to train networks with more iterations within a feasible time frame. For 3D seismic data, this poses a challenge due to the limited amount of memory on GPUs. For 3D data, this quickly limits the sizes of networks, samples, and batches that can be used.

In the context of seismic, we need to be aware that the underlying geology might be very different depending on the region where the data were acquired. Also, the data quality is dependent on the processing workflow, sampling interval, and eventual errors in the velocity model. If we want to train a network that should generalize to new data sets, we need to have training samples from data sets covering a wide range of these differences. However, when a new seismic data set is processed and interpreted, a large amount of manual work and quality control is invested in each data set. Therefore, the cost of labeling some parts (e.g., some slices) of a data set is relatively small compared to the entire workflow. This makes it possible to train the network on a few slices from the data set and use it to interpret the remaining.

Suggested reading

The network architecture we used is rather simple and is based on texture classification. State-of-the-art models for classification (Szegedy et al., 2015; He et al., 2016) should be considered for complex classification tasks and segmentation networks (Ronneberger et al., 2015) for non-texture-based problems. In cases with little training data, reuse of existing networks (Razavian et al., 2014) or domain adaption (i.e., Ganin and Lempitsky, 2015) could improve results, but it is not trivial to succeed with such methods. Visualization of trained networks (i.e., Zeiler and Fergus, 2014; Mordvintsev et al., 2015) is important in order to understand how a network solves a given classification task. It should be noted that the field of deep learning is actively being developed and that best practices for network configurations and choosing hyperparameters and training strategies change rapidly.

Conclusions

The use of CNNs on seismic data is promising and may lead to increased accuracy for automated seismic interpretation, as it has done for tasks in natural and medical image analysis. CNNs consider the spatial aspect of images and exhibit the hierarchal structure for detecting complex objects through layers of convolutional nodes. The filter weights are optimized during the training phase, which means CNNs can be used without the need for predefined attributes. This means that the network may detect which characteristics of the classes are useful for classification. The CNNs are commonly trained by iterative optimization using the SGD algorithm. This requires a sufficient amount of training data and the use of the correct regularization techniques. We have demonstrated the use of CNNs in the context of seismic images by using them to delineate salt bodies. One manually labeled slice was used to train the network, and the network was used to successfully delineate the full 3D salt body. This was confirmed by a comparison with a manual interpretation. TLE

Acknowledgments

This work is funded by the Norwegian Research Council, Grant 234019. The CNN used in this paper was implemented using the TensorFlow framework by Google. Example code is available at https://github.com/waldeland/CNN-for-ASI.

Corresponding author: anders.u.waldeland@gmail.com

References

Araya-Polo, M., T. Dahlke, C. Frogner, C. Zhang, T. Poggio, and D. Hohl, 2017, Automated fault detection without seismic processing: The Leading Edge, **36**, no. 3, 208–214, https://doi.org/10.1190/tle36030208.1.

Bengio, Y., 2012, Practical recommendations for gradient-based training of deep architectures, *in* G. Montavon, G. B. Orr, and K.-R. Müller, eds., Neural networks: Tricks of the trade, second edition: Springer Heidelberg, 437–478.

Gabor, D., 1946, Theory of communication. Part 1: The analysis of information: Journal of the Institution of Electrical Engineers — Part III: Radio and Communication Engineering, **93**, no. 26, 429–457, https://doi.org/10.1049/ji-3-2.1946.0074.

Ganin, Y., and V. Lempitsky, 2015, Unsupervised domain adaptation by backpropagation: Proceedings of the International Conference on Machine learning, 1180–1189.

Glorot, X., and Y. Bengio, 2010, Understanding the difficulty of training deep feedforward neural networks: Proceedings of the 13th International Conference on Artificial Intelligence and Statistics, 249–256.

Goodfellow, I., Y. Bengio, and A. Courville, 2016, Deep learning: MIT Press.

He, K., X. Zhang, S. Ren, and J. Sun, 2016, Deep residual learning for image recognition: IEEE Conference on Computer Vision and Pattern Recognition, 770–778, https://doi.org/10.1109/CVPR.2016.90.

Hensman, P., and D. Masko, 2015, The impact of imbalanced training data for convolutional neural networks: PhD thesis, KTH Royal Institute of Technology.

Huang, L., X. Dong, and T. E. Clee, 2017, A scalable deep learning platform for identifying geologic features from seismic attributes: The Leading Edge, **36**, no. 3, 249–256, https://doi.org/10.1190/tle36030249.1.

Ioffe, S., and C. Szegedy, 2015, Batch normalization: Accelerating deep network training by reducing internal covariate shift: Proceedings of 32nd International Conference on Machine Learning.

Kingma, D., and J. Ba, 2015, Adam: A method for stochastic optimization: Proceedings of the 3rd International Conference on Learning Representations, 1–15.

Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012, ImageNet classification with deep convolutional neural networks: Proceedings of the 25th International Conference on Neural Information Processing Systems, 1097–1105.

Lee, H., R. Grosse, R. Ranganath, and A. Y. Ng, 2011, Unsupervised learning of hierarchical representations with convolutional deep belief networks: Communications of the ACM, **54**, no. 10, 95–103, https://doi.org/10.1145/2001269.2001295.

Mordvintsev, A., M. Tyka, and C. Olah, 2015, Inceptionism: Going deeper into neural networks: Google research blog, https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html, accessed 4 May 2018.

Nair, V., and G. E. Hinton, 2010, Rectified linear units improve restricted Boltzmann machines: Proceedings of the 27th International Conference on Machine Learning, 807–814.

Razavian, A. S., H. Azizpour, J. Sullivan, and S. Carlsson, 2014, CNN features off-the-shelf: An astounding baseline for recognition: IEEE Conference on Computer Vision and Pattern Recognition Workshops, https://doi.org/10.1109/CVPRW.2014.131.

Rojo, L. A., A. Escalona, and L. Schulte, 2016, The use of seismic attributes to enhance imaging of salt structures in the Barents Sea: First Break, **34**, no. 11, 49–57, https://doi.org/10.3997/1365-2397.2016014.

Ronneberger, O., P. Fischer, and T. Brox, 2015, U-Net: Convolutional networks for biomedical image segmentation: Medical Image Computing and Computer-Assisted Intervention — MICCAI 2015, 234–241, https://doi.org/10.1007/978-3-319-24574-4_28.

Rumelhart, D. E., G. E. Hinton, and R. J. Williams, 1986, Learning representations by back-propagating errors: Nature, **323**, 533–536, https://doi.org/10.1038/323533a0.

Russakovsky, O., J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, et al., 2015, ImageNet Large Scale Visual Recognition Challenge: International Journal of Computer Vision, **115**, no. 3, 211–252, https://doi.org/10.1007/s11263-015-0816-y.

Simard, P. Y., D. Steinkraus, and J. C. Platt, 2003, Best practices for convolutional neural networks applied to visual document analysis: 7th International Conference on Document Analysis and Recognition, 958–963.

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, 2014, Dropout: A simple way to prevent neural networks from overfitting: Journal of Machine Learning Research, **15**, 1929–1958.

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, 2015, Going deeper with convolutions: IEEE Conference on Computer Vision and Pattern Recognition, https://doi.org/10.1109/CVPR.2015.7298594.

van der Baan, M., and C. Jutten, 2000, Neural networks in geophysical applications: Geophysics, **65**, no. 4, 1032–1047, https://doi.org/10.1190/1.1444797.

Waldeland, A. U., and A. H. S. Solberg, 2017, Salt classification using deep learning: 79th Conference and Exhibition, EAGE, Extended Abstracts, https://doi.org/10.3997/2214-4609.201700918.

Zeiler, M. D., and R. Fergus, 2014, Visualizing and understanding convolutional networks: European Conference on Computer Vision, 818–833, https://doi.org/10.1007/978-3-319-10590-1_53.