GEOPHYSICS, VOL. 85, NO. 1 (JANUARY-FEBRUARY 2020); P. V33–V43, 15 FIGS., 4 TABLES.
10.1190/GEO2018-0870.1

Automatic velocity analysis using convolutional neural network and transfer learning

Min Jun Park[1] and Mauricio D. Sacchi[2]

ABSTRACT

Velocity analysis can be a time-consuming task when performed manually. Methods have been proposed to automate the process of velocity analysis, which, however, typically requires significant manual effort. We have developed a convolutional neural network (CNN) to estimate stacking velocities directly from the semblance. Our CNN model uses two images as one input data for training. One is an entire semblance (guide image), and the other is a small patch (target image) extracted from the semblance at a specific time step. Labels for each input data set are the root mean square velocities. We generate the training data set using synthetic data. After training the CNN model with synthetic data, we test the trained model with another synthetic data that were not used in the training step. The results indicate that the model can predict a consistent velocity model. We also noticed that when the input data are extremely different from those used for the training, the CNN model will hardly pick the correct velocities. In this case, we adopt transfer learning to update the trained model (base model) with a small portion of the target data to improve the accuracy of the predicted velocity model. A marine data set from the Gulf of Mexico is used for validating our new model. The updated model performed a reasonable velocity analysis in seconds.

INTRODUCTION

Building a velocity model is a critical seismic data processing step. Traditionally, a processor picks a geologically sound stacking velocity from the semblance (Taner and Koehler, 1969; Neidell and Taner, 1971). Velocities are estimated by semblance analysis and then used in the normal moveout (NMO) correction. In general, velocity analysis is a time-consuming task because it requires visual examination of a large number of semblance panels by a processor. Many studies have been conducted to automate velocity analysis for solving this problem (Toldi, 1985; Abbad et al., 2009; Fomel, 2009; Choi et al., 2010; Chen, 2018). Nonconvolutional neural networks (NNs) have also been suggested for velocity analysis (Schmidt and Hadsell, 1992; Fish and Kusuma, 1994; Calderón-Macías et al., 1998).

Recently, there has been a significant improvement in deep learning itself due to the advancement of computer performance (LeCun et al., 2015). The convolutional neural network (CNN) is one of the most powerful algorithms to classify images among various approaches in deep learning (LeCun et al., 1989). This is because the CNN can consider the local connectivity of the input image by adopting convolutional filters that provide invariance of geometric shifts or distortions (LeCun and Bengio, 1995). Given its superiority in extracting features from images, CNNs have been extensively used in image classification or recognition (Lawrence et al., 1997; Simard et al., 2003; Krizhevsky et al., 2012; Karpathy et al., 2014; Maturana and Scherer, 2015; He et al., 2016). Studies of the application of deep learning to seismic data processing have also been carried out. For instance, Araya-Polo et al. (2018) propose to adopt deep NN for tomographic inversion. Also, CNN applications in problems such as fault classification (Wu et al., 2018, 2019; Xiong et al., 2018), first-break picking (Yuan et al., 2018), and denoising (Liu et al., 2018) have been explored.

In this study, we propose a CNN for automatic velocity analysis. We consider the velocity picking process to be an image classification problem by assuming that small patches extracted from the semblance represent stacking velocity corresponding to a specific time. For training, we use seven synthetic custom models to build the training data set. We first compute synthetic shot gathers from those models by using the finite-difference method and calculate semblance panels after performing common midpoint (CMP) sorting. Then, we obtain the input data from the semblance panels and

Manuscript received by the Editor 18 December 2018; revised manuscript received 4 September 2019; published ahead of production 07 October 2020; published online 22 November 2019.
[1]Formerly University of Alberta, Department of Physics, Edmonton, Alberta, Canada; presently Stanford University, Department of Geophysics, Stanford, California, USA. E-mail: minjun1@ualberta.ca; minjun@stanford.edu.
[2]University of Alberta, Department of Physics, Edmonton, Alberta, Canada. E-mail: msacchi@ualberta.ca.

calculate the rms velocity from the velocity model to generate the labels. After training, the trained model can predict a consistent velocity field. We assume that the CNN model trained with the rms velocity can predict the stacking velocity because those seven synthetic models correspond to quasihorizontal structures (Yilmaz, 2001). However, if the target and training data are significantly different, the base model can barely output the correct labels. To overcome this limitation, we use transfer learning (Pan and Yang, 2010; Yosinski et al., 2014), which is an additional training process with a small portion of target data. We have found that although the new input data have features different from those of the training data set, the CNN model obtained after transfer learning can predict reasonable stacking velocities. A marine data set from the Gulf of Mexico is used for validating our method. Training the original model takes a significant amount of time (approximately 3 h), but transfer learning takes less than 5 min. Also, the time that the trained model takes to predict the velocity is negligible.

METHOD

Problem formulation

Neidell and Taner (1971) introduce the concept of velocity spectra for velocity analysis and the semblance as an energy measure to optimally determine stacking velocities from CMP gathers. In this paper, we adopt a CNN model to automatically determine stacking velocities from semblance panels. Semblance has a direct relation to velocity. Even if scanning the semblance panels requires extra time, it improves the training process by providing essential features for training (Araya-Polo et al., 2018).

The input data first need to be defined for training. Figure 1 shows the definition of the input data. The guide image G is representing the whole semblance. The target image T includes only the values in a specific τ range where τ is the zero-offset two-way traveltime of the reflection hyperbola. Each image T represents the velocity at the middle of the τ interval. We substitute all of the values in the semblance by zero except for the values in the τ interval of analysis. The number of target images T is 45 for each semblance. Thus, the input data are represented by pairs of images denoted by G_n and T_{n,t_m}, where n indicates the CMP number and the index $m = \{1, 2, 3 \ldots, 45\}$ corresponds to the intercept traveltime t_m. We unify the size of all input data by compressing them into images of 50×50 pixels to reduce the computational cost. The model trained with inputs G and T can pick a reasonable stacking velocity for each τ because it allows the trained model to focus on a specific τ time and it also considers the overall trend of the semblance. Regarding

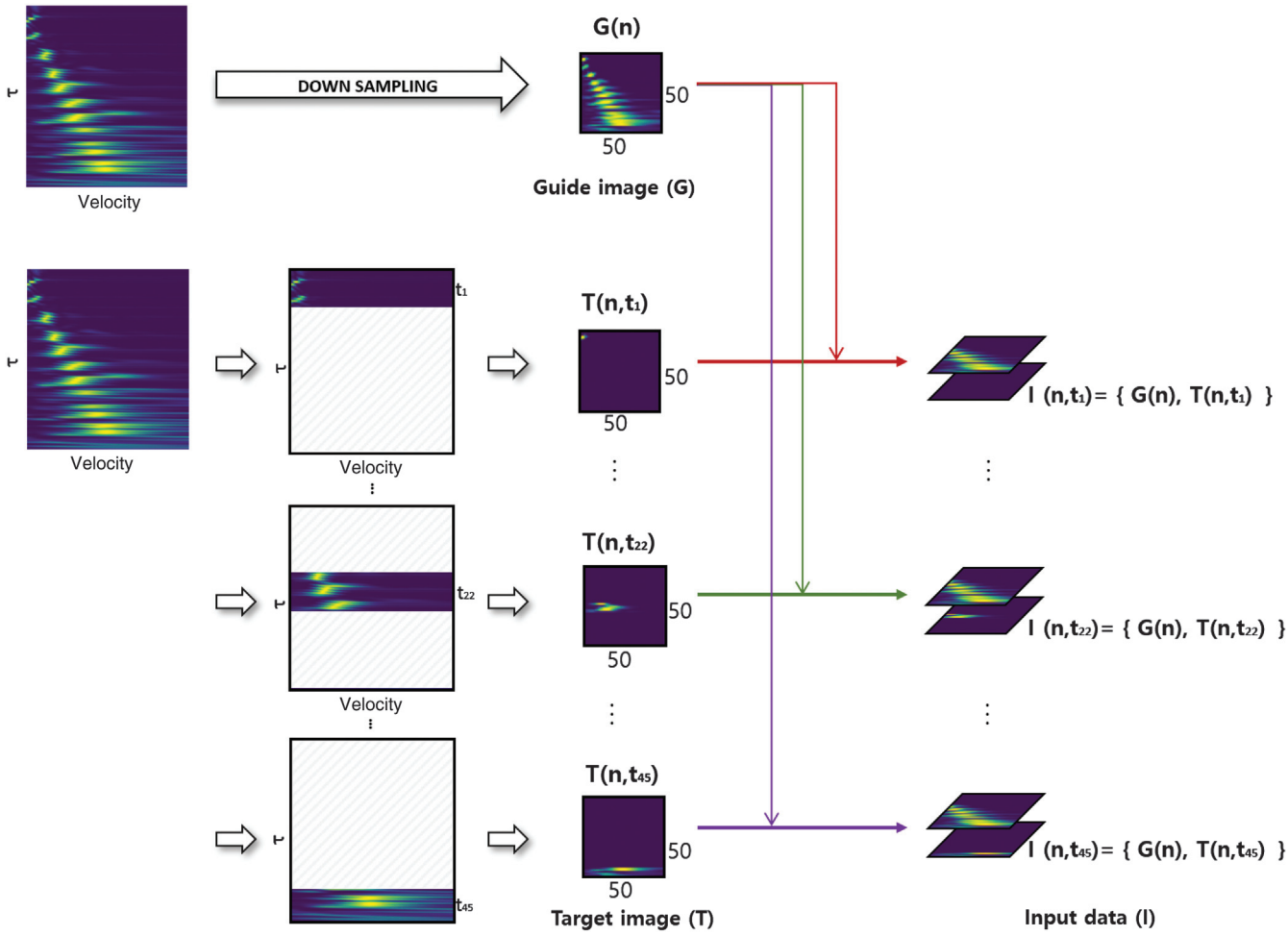


Figure 1. Input data definition. For each semblance, there is one guide image G and 45 target images T. Then, we pair each G_n and T_{n,t_m} to produce the input data (I_{n,t_m}), where n is the CMP number and the index $m = \{1, 2, 3 \ldots, 45\}$ corresponds to the intercept traveltime t_m.

the label definition, we divide the semblance velocity axis into $K = 40$ compartments and define each compartment as one class. That is, one class represents a specific velocity bin. For example, if the velocity axis of the semblance ranges from 2000 to 4000 m/s and if we assign classes with an increment of 50 m/s, all values of velocity between 2000 and 2050 m/s belong to class 1. Similarly, velocities in the bin 2050–2100 m/s belong to class 2 and so on. Labels can be represented in binary form. For instance, for class 2, the label is given by the vector of length K, $\mathbf{p}_{n,m} = [0, 1, 0, 0, 0, \ldots, 0]$ where n is the CMP number and m is the intercept time. We match all input data with the corresponding label to construct the data set for training. Our model outputs the vector $\hat{\mathbf{q}}[j]_{n,m}, j = 1 \ldots 40$. We adopt a softmax classifier to convert $\hat{\mathbf{q}}_{n,m}$ to the probability of each class j being correctly identified

$$q[j]_{n,m} = \frac{\exp\ \hat{q}[j]_{n,m}}{\sum_{i=1}^{K} \exp\ \hat{q}[i]_{n,m}}. \tag{1}$$

Thus, each $q[j]_{n,m}$ represents a predicted probability for each velocity class j, where each element of the vector $q[j]_{n,m}$ is a number in the interval [0,1]. The weights of the CNN are computed by minimizing a cost function that measures the similarity of $\mathbf{p}_{n,m}$ and $\mathbf{q}_{n,m}$, which is equivalent to say that the CNN model should be able to predict the stacking velocities provided in the training step. We have adopted the cross-entropy cost function

$$J = -\sum_{n}\sum_{m}\sum_{j=1}^{K} p_{n,m}(j)\log(q_{n,m}(j)). \tag{2}$$

It is usually applied for building CNN models because it leads to an algorithm with a high training speed (Nielsen, 2015).

Data construction

We construct a training data set using the seven custom synthetic velocity models shown in Figure 2. We also use another velocity model to generate a data set for validation. All models have a different number of layers. They all have a velocity distribution that increases with depth.

Figure 3 shows the workflow that we use to create the CNN data set. First, we simulate data from the velocity models by solving the acoustic wave equation with the finite-difference method. We also conduct CMP sorting and apply automatic gain control to the data to equalize amplitudes. Then, we compute the semblance panels for each model. We also normalize the semblance panels to a common predefined scale. For all custom models, the number of time samples of each record is 2850. The number of semblance panels is 70. The velocity for each semblance ranges from 2000 to 4000 m/s. Finally, we construct the entire input data set using the method mentioned in the last section. The total size of the input data is $22,050 \times 50 \times 50 \times 2$, where $22,050 = 7 \times 70 \times 45$ is the number

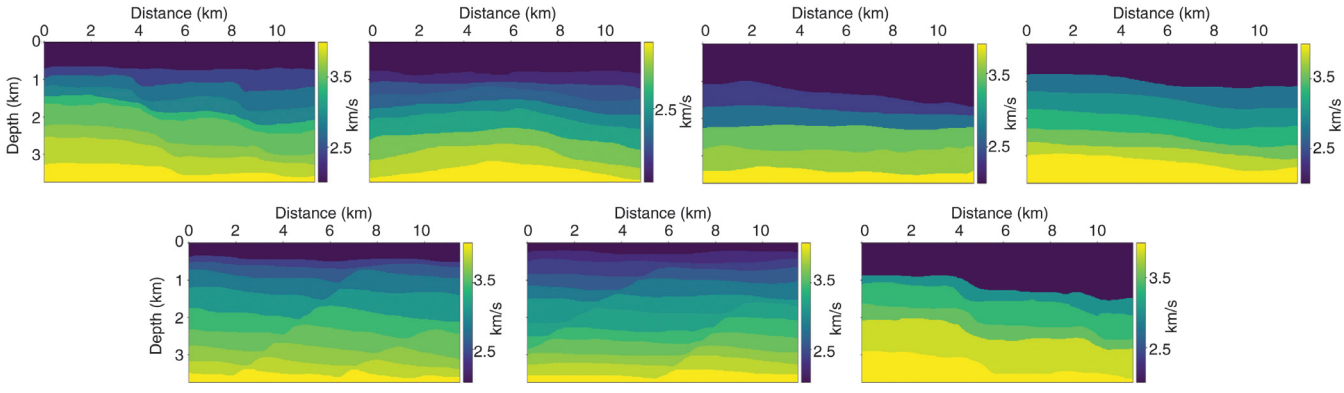


Figure 2. Seven synthetic velocity models used to generate the training data. All models have a velocity distribution that increases with depth.

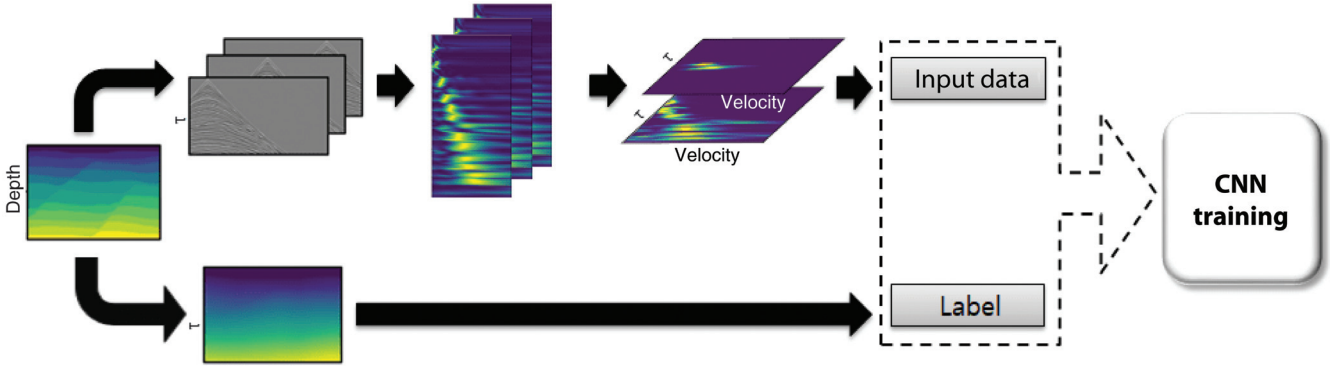


Figure 3. Workflow displaying the training step. We solve the acoustic wave equation via the finite-difference method to compute synthetic shot gathers. We calculate the semblance panels for each CMP location after CMP sorting and automatic gain control. To compute the labels, the rms velocities are computed analytically from the velocity models.

of input data and $50 \times 50 \times 2$ is the size of each input data. We also calculate the rms velocity profiles at the CMP locations where we computed the semblance. As mentioned earlier, we assume that the rms velocity and the stacking velocity are equivalent because the seven synthetic models used for training are composed of quasihorizontal layers. All of the rms velocities are mapped into $K = 40$ classes. The total size of the label is $22{,}050 \times 40$. We match all input data sets with all labels to create the entire training data set.

CNN model training

For efficiency, we decide to adopt one among the three existing CNN architectures: LeNet-5 (LeCun, 2015), AlexNet (Krizhevsky et al., 2012), and VGG16 (Simonyan and Zisserman, 2014). To find the best model for our task, we first train each model with our training data set and test the trained model with our validation data set. We evaluate each training performance using the F1 score (Powers, 2011) presented in Table 1. All of the results are from the same validation data set after 100 training epochs. We can clearly observe that VGG16 is the most superior architecture for automatic velocity analysis among the three architectures because it has the highest F1 score and accuracy as well as the lowest loss value. Thus, we decide to adopt VGG16 for our research.

Table 1. Evaluation results for three architectures (LeNet-5, AlexNet, and VGG16). All of the results are from the validation data set after 100 training process epochs.

Architecture	Loss	Accuracy	F1 score
LeNet-5	3.53	0.45	0.45
AlexNet	2.35	0.51	0.51
VGG16	1.53	0.69	0.69

Figure 4 shows the details of the VGG16 architecture. It consists of five convolution blocks and two fully connected layers followed by a softmax classification layer. First block includes two convolution layers, and both layers have 64 filters of size 3×3. The second block also contains two convolution layers, and both layers have 128 filters of size 3×3. The third one has three convolution layers, and each layer has 256 filters of size 3×3. The fourth and fifth blocks have three convolution layers each, and all of the layers have 512 filters of size 3×3. After every convolution layer, it contains the `batch-normalization` and rectified linear units (`ReLU`) activation function (Nair and Hinton, 2010; Krizhevsky et al., 2012). After each convolution block, max-pooling of size and stride 2×2 is applied for subsampling. In the `FC1` layer, the feature maps are flattened to 4096 elements. The `FC2` layer also has 4096 elements. We apply `ReLu` after `batch-normalization` and `dropout` are used to `FC1` and `FC2` layers to prevent overfitting (Srivastava et al., 2014). Through the softmax classification layer, we predict the stacking velocities. For the optimization of the CNN, we adopt the adaptive moment estimation (Adam) optimizer (Kingma and Ba, 2014) implemented in `Tensorflow` (Abadi et al., 2016).

In addition to the CNN structure, we also need to obtain optimal hyperparameters to get better performance. Based on the VGG16 we adopt, we try to find the optimal values of two hyperparameters: the `learning rate` and `batch size`. The evaluation method for each hyperparameter is the same as the method that we use for selecting the architecture. The first parameter that we try to optimize is the `learning rate`. Table 2 shows the evaluation results for five different learning rates (0.0001, 0.0005, 0.001, 0.005, and 0.01). All of the results are from the same validation data set after 100 training epochs. The batch size, in this case, is unified to 256. Among the five different `learning rates`, we choose a

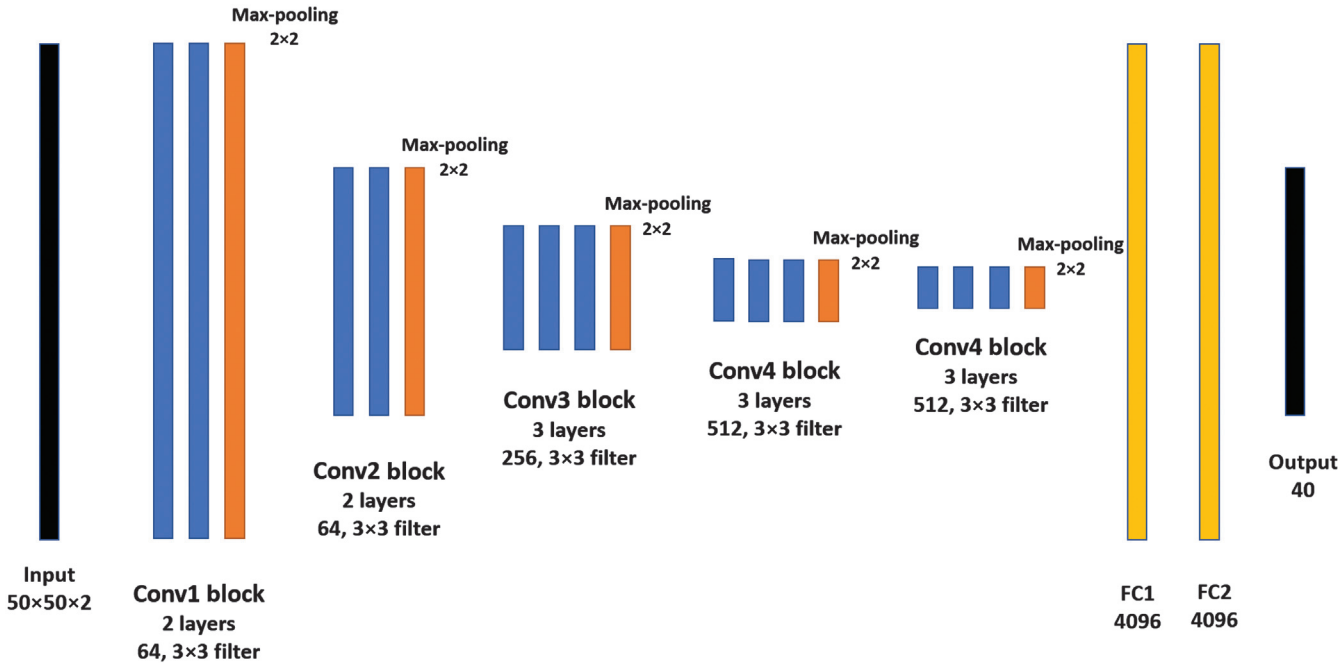


Figure 4. A VGG16 CNN architecture. It contains a total of 16 layers (13 convolutional layers and two fully connected layers followed by a softmax classifier). The model outputs 40 numbers in one array for each input data. Each output represents the probability of each class.

learning rate of 0.001 as it shows the lowest loss and the highest accuracy and F1 score.

We tune the batch size after choosing a learning rate 0.001. Table 3 shows the evaluation results for six different batch sizes (32, 64, 128, 256, 512, and 1024). We observed that the trained model with small batch size shows better prediction for the validation data set. The benefit of small batch size has also been shown in many research studies (LeCun et al., 2012; Keskar et al., 2016; Masters and Luschi, 2018). Although batch size 32 and 64 show the highest values of accuracy and F1 score, we choose batch size 64 because it requires less training time.

We are finally ready to train the CNN model using our defined data set. We randomly initialize all of the weights of the CNN and train the model with randomly shuffled training data set. We use a computer equipped with a RTX 2080 ti graphic card for training. Training takes approximately 3 h for 300 epochs. We call the trained model the "Base model."

Transfer learning

Even though our model is trained as expected, we often have insufficient data for training. Let us assume a task X and a trained CNN model through labeled data A. We expect that the trained model can predict the correct label of the new data B in task Y, which is similar to X. However, this scenario is only possible if the data A are large enough to reflect task X and Y or the data B have features that are similar to those of the data A. Unfortunately, there are not enough labeled data for training in many of our real seismic data processing applications. This data-hungry problem also occurs in our semblance analysis problem, which requires manual picking of velocities to obtain the labeled data. Thus, we use transfer learning when we have an unsatisfying result with the original model trained with synthetic data (the base model). Figure 5 shows the concept of transfer learning. The base model (trained with a labeled data A) can be updated with a small portion of target data B so that it can predict task Y well.

Table 2. Evaluation results for five different learning rates (0.0001, 0.0005, 0.001, 0.005, and 0.01). All of the results are from the same validation data set after 100 training epochs.

Learning rate	Loss	Accuracy	F1 score
0.0001	0.1	0.57	0.56
0.0005	0.02	0.65	0.65
0.001	0.03	0.69	0.69
0.005	0.05	0.64	0.64
0.01	0.29	0.66	0.66

Table 3. Evaluation results for six different batch sizes (32, 64, 128, 256, 512, and 1024). All of the results except for time/epoch are from the validation data set after 100 training process epochs.

Batch size	Loss	Accuracy	F1 score	Time/epoch (s)
32	1.35	0.73	0.73	101.3
64	1.41	0.73	0.73	50.2
128	1.45	0.72	0.72	27.8
256	1.53	0.69	0.69	15.4
512	1.95	0.62	0.62	12.1
1024	1.91	0.64	0.63	10.3

EXAMPLES

Synthetic data example

The base model (without transfer learning) is first evaluated with a simple synthetic case. We use the velocity model (Figure 6) that includes similar features to the training data for testing. The testing data are obtained in the same way as the base model training data. The test result shows 0.74 of the F1 score and 75% accuracy. Most predicted values of the remaining 25% outputs have a similar value to the label. We also obtain a mean squared error (MSE) of 550 between the prediction and the label velocity fields. This result is acceptable considering the number of classes is 40. Figure 7 shows the label model (Figure 7b) and the predicted velocity model (Figure 7a). The predicted velocities match almost exactly to the label model. We performed NMO correction and stacking using the true rms velocities and the predicted velocities. Figure 8a shows the predicted velocity (red line) and the true rms velocity (blue line) superimposed on the semblance panel. The NMO-corrected result

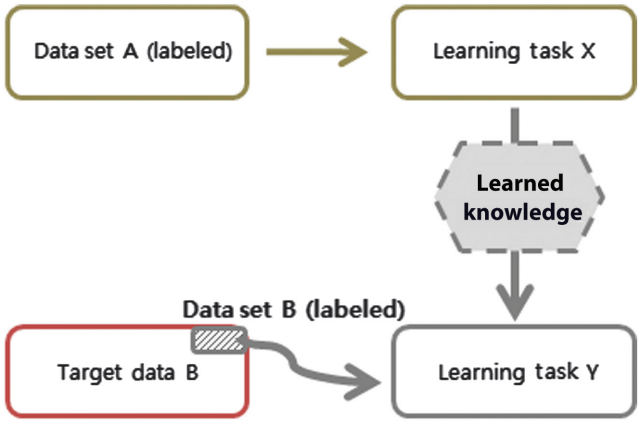


Figure 5. The concept of transfer learning.

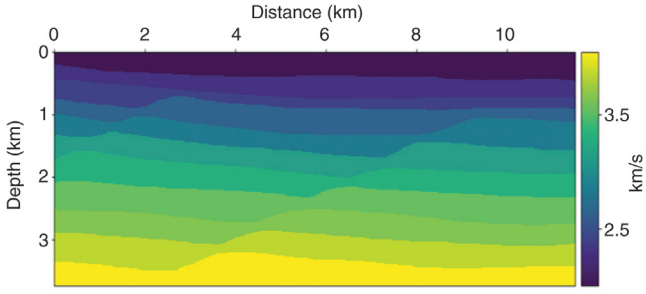


Figure 6. The velocity model used for testing. This model was not used in the training stage.

(Figure 8d) using the predicted velocity is almost identical to the result (Figure 8c) obtained by the true rms velocity. There are also some parts with a relatively high error, especially in the shallow part. In those areas, the predicted velocity is slightly higher than the true rms velocity. The NMO correction with velocity higher than the true rms velocity causes undercorrection as shown in Figure 8d. Finally, we apply the NMO correction to all CMP gathers and stack them. Figure 9 shows the stack sections for the true velocity model (Figure 9a) and the predicted velocity model (Figure 9b). As mentioned above, the trained model predicts a shallow velocity field that is higher than the true one. Thus, we can also observe the relatively large error in the shallow part of the stack (Figure 9c). We confirm that the trained model can output a consistent velocity by comparing the stack section with the near-offset section (Figure 9d).

Field data example

Figure 10 shows the near-offset traces of a data set from the Gulf of Mexico (Verschuur and Prein, 1999). The data set is contaminated with free-surface multiples. We adopt transfer learning because the

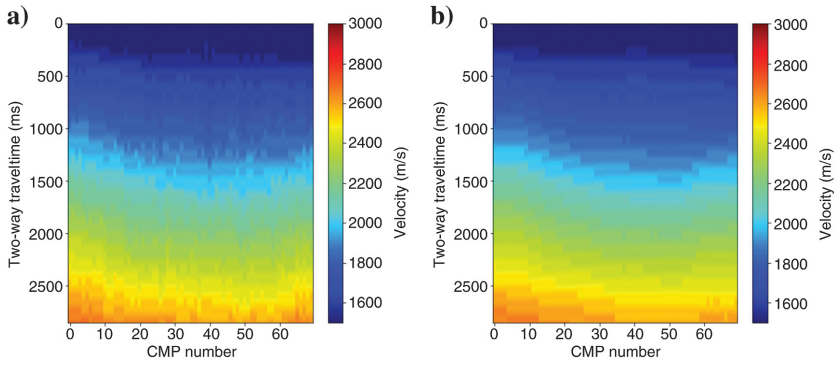


Figure 7. (a) Velocity field predicted via CNN and (b) the label velocity model.

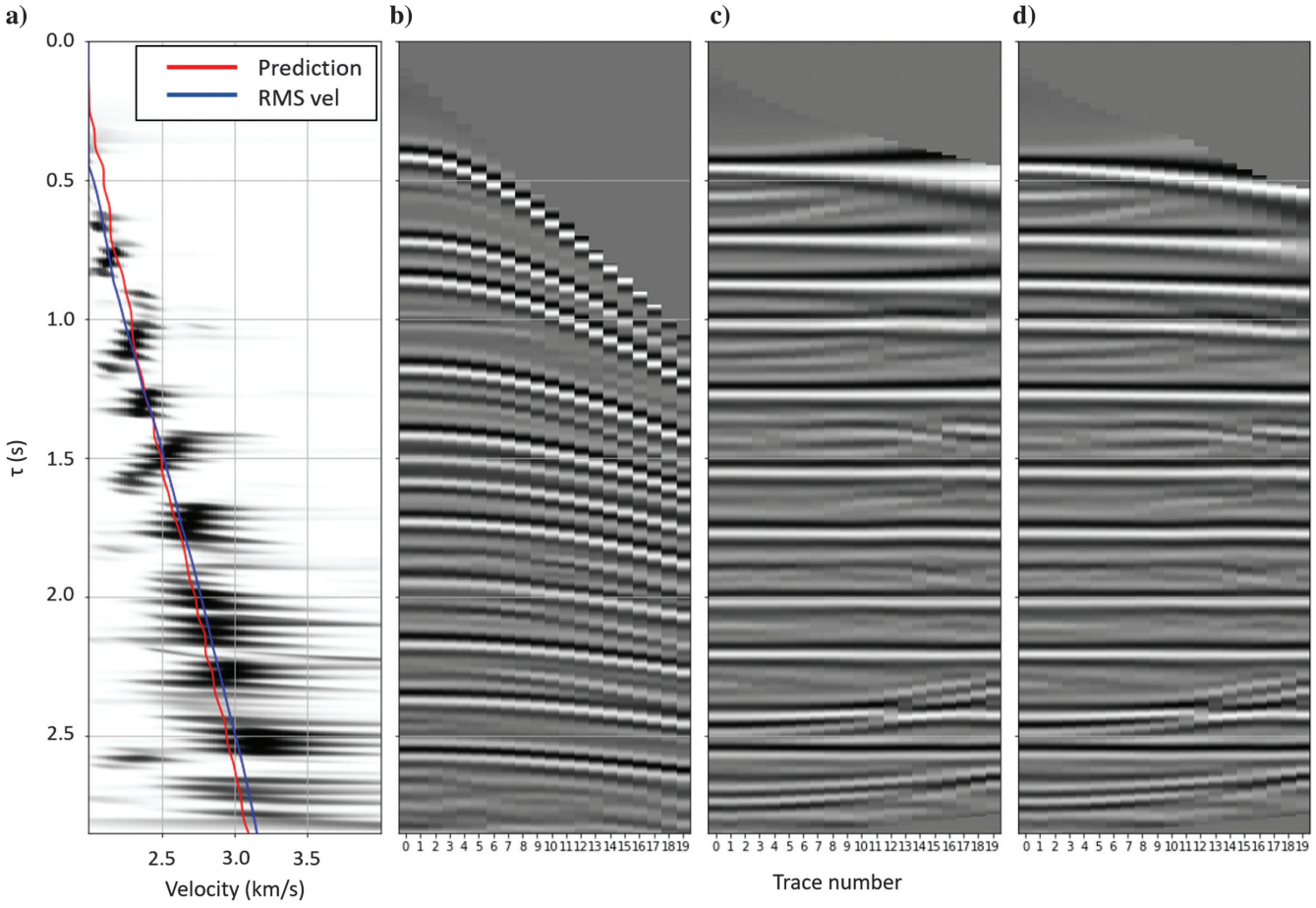


Figure 8. (a) Predicted velocity (the red line) and true rms velocity (the blue line) are shown in conjunction with the semblance. (b) NMO correction corresponding to the CMP gather. (c) The results of NMO correction with the true velocity and (d) the predicted velocity.

synthetic data that we use for the base model training is quite different from the real marine data. In other words, we extracted six semblance panels (out of 1400) and velocity profiles (dashed lines in Figures 11) that were obtained via manual velocity analysis and used them for transfer learning. The manual analysis velocity field (Figure 11a) is created using linear interpolation, with velocity analysis every 50 CMP gathers.

As shown in Figure 10, there is a severe lateral structural variation between CMP 600 and CMP 1000, which can make the manual velocity analysis more difficult. Thus, when we chose the locations of the data set for transfer learning, we excluded this area to check whether the CNN model can provide a satisfactory solution. Three velocity profiles are taken from areas outside the tabular salt body (CMP: 202, 402, 602), and another three velocity profiles are taken from CMPs above the tabular salt body (CMP: 1002, 1302, 1599).

When using transfer learning, it is important to decide which part of the model to update. If the training data set for the base model is similar to the target data, and the amount of data used for the update is small, it is common to update only fully connected layers (Huot et al., 2018). In the opposite case, we can update the entire pretrained model. To determine the update part, we perform transfer learning for three different updating targets: full model update, update after the first convolution block, and update after the second convolution block. We evaluate each case using the MSE between the predicted velocity field and the label velocity field. When we

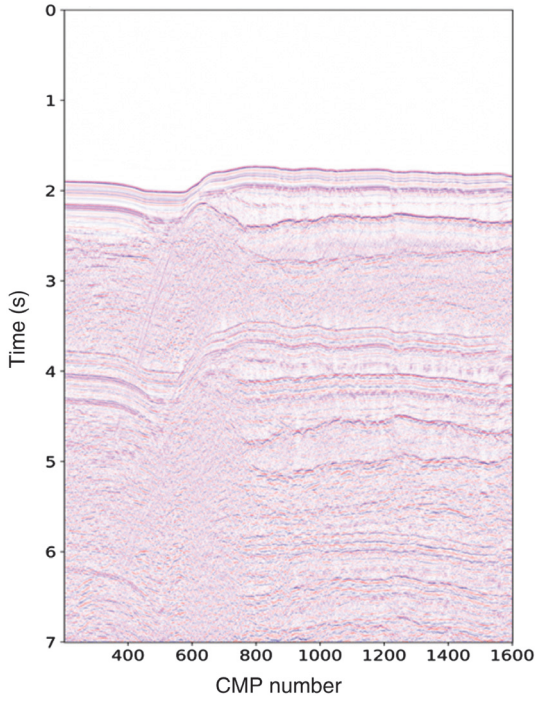


Figure 10. Near-offset traces of a data set from the Gulf of Mexico. The data include a salt body from position CMP 700 to CMP 1600.

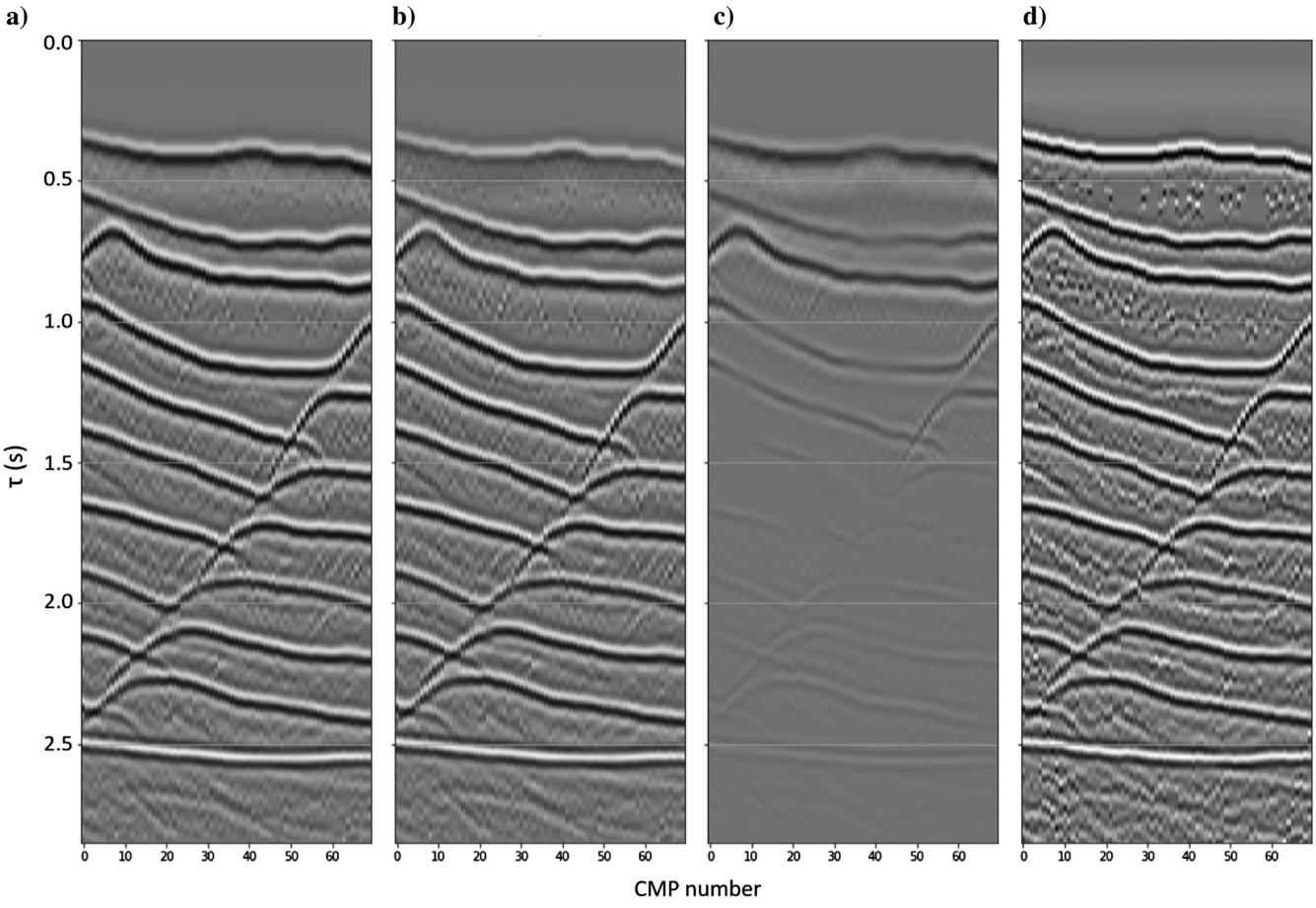


Figure 9. Stack sections. (a) Stack section obtained with the true velocity model. (b) Stack section obtained via the velocity model predicted by CNN. (c) Difference: panel (b) minus panel (a). (d) The near-offset traces.

calculate MSE, we exclude the location between CMP 600 and CMP 1000 to verify only the training performance.

Table 4 shows the results of comparison. We can see that transfer learning after the first block is the best updated model because it has the highest F1 score as well as the lowest MSE. Transfer learning, in this case, takes only approximately 3 min, and prediction for 1400 semblances takes a few seconds. Figure 11b shows the predicted velocity field. From the CNN model, we have 45 output points for each semblance panel. Then, we interpolated via a spline method the 45 points to estimate the whole velocity profile. We also apply a 2D Gaussian filter to smooth the velocity field. The white lines (dashed) indicate the positions of the CMP locations where we adopted the data for transfer learning. Figure 12 shows the difference between the manual analysis velocity field and the predicted velocity field (the difference between Figure 11a and 11b). The prediction result is quite similar to the velocity model estimated by manual analysis except for the area approximately CMP 800. As mentioned earlier, we exclude this area when we perform transfer learning. In this CMP range, the CNN model predicts velocities that are higher than those estimated by manual analysis.

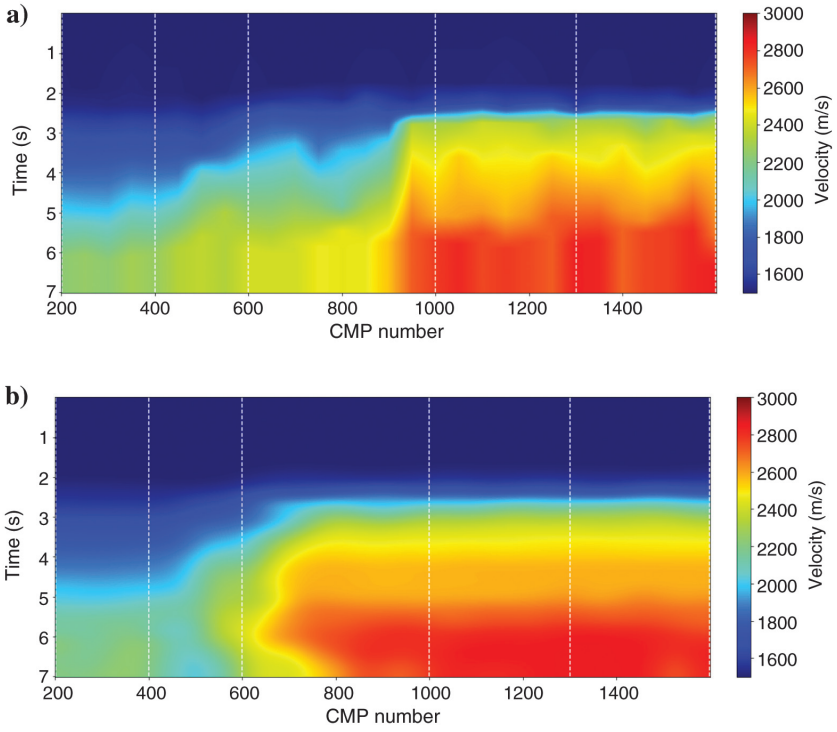


Figure 11. (a) The velocity field created using linear interpolation with velocity analysis every 50 CMP gathers and (b) the velocity field predicted by CNN. We use a total of six semblances and velocity profiles (CMP: 202, 402, 602, 1002, 1302, and 1599) to perform transfer learning.

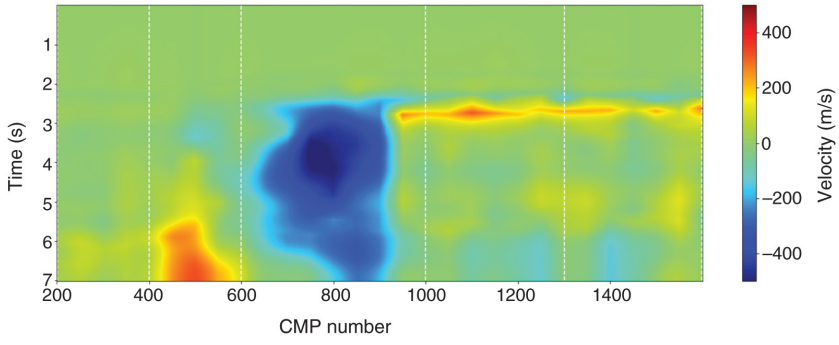
Figure 13 shows the semblance panels at CMP 350, 800, 1250, and 1400. It is important to mention that these CMPs were not used for transfer learning. The red lines indicate the manual velocity analysis, and the white lines represent the CNN predicted velocity. We point out a significant discrepancy between automatic and manually estimated velocities at CMP 800. The discrepancy occurs in an area dominated by nonhorizontal structures and diffracted energy. When analyzing the pattern of semblances from the areas outside the tabular salt body (Figure 13a) and the area above the tabular salt body (Figure 13c and 13d), it can be said that the CNN model has identified an acceptable solution.

Table 4. Evaluation results for three different transfer learning strategies. All of the results are from the manual analysis velocity field after 500 epochs.

Target of transfer learning	Accuracy	F1 score	MSE
Full model update	0.45	0.46	10,535
Update after first block	0.47	0.49	6368
Update after second block	0.47	0.47	12,444

Finally, we compare stack sections for the manual analysis (Figure 14a) and the CNN prediction (Figure 14b). Both stack sections have similar results in most areas. To verify the CNN prediction especially in the area including strong lateral variations, we magnified two parts in this area (between CMP 600 and 1000). One is from $\tau = 2.4$ to $\tau = 3.6$ s (Figure 15a and 15b), and the other is between $\tau = 5$ and $\tau = 6.2$ s (Figure 15c and 15d). The CNN prediction results have clearer reflectors than the manual analysis; these results are indicated with the red arrows.

Figure 12. The difference velocity field (Figure 11a minus Figure 11b).



DISCUSSION

Our results were satisfactory overall. We think there is room for improvement in prediction performance if we increase the input size or use a better quality of training data set. However, the CNN model proposed in this paper does not consider the surrounding semblance's analysis and its lateral continuity as a good processor would do. This is also an essential part to take into account in the future. To overcome this limitation, we need to expand our input data from a 2D semblance to a 3D semblance cube to include geologic information for training (Araya-Polo et al., 2018). In this case, semantic segmentation can be a possible solution (Long et al., 2015; Garcia-Garcia et al., 2017). Using semantic segmentation, each pixel of the image can be defined as its label. Even if this method needs more computational resources and more training data sets, it has shown enormous potential in various geophysical applications such as salt classification (Shi et al., 2018), channel detection (Pham et al., 2018), seismic facies classification (Zhao, 2018), and velocity estimation (Wang et al., 2018).

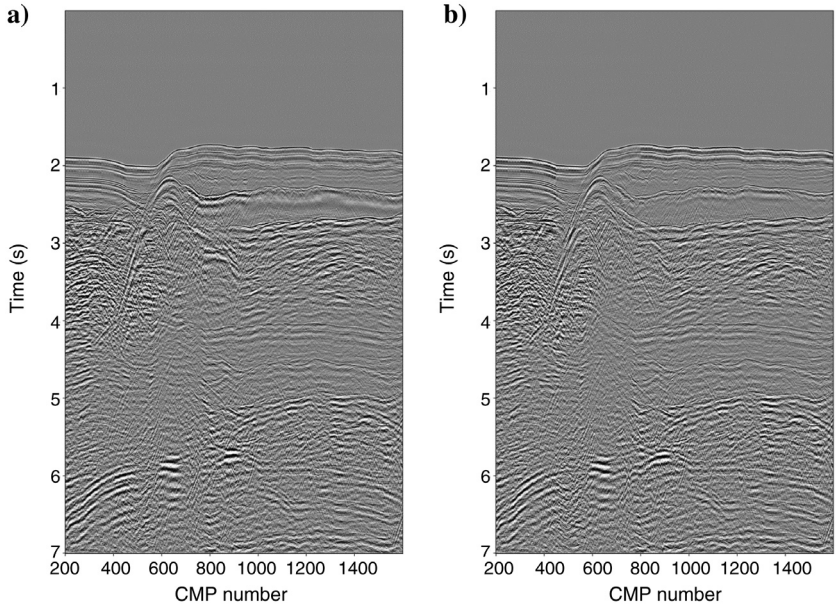


Figure 14. Stack sections. (a) Stack section obtained with the manual analysis velocity model. (b) Stack section obtained via the velocity model predicted by CNN.

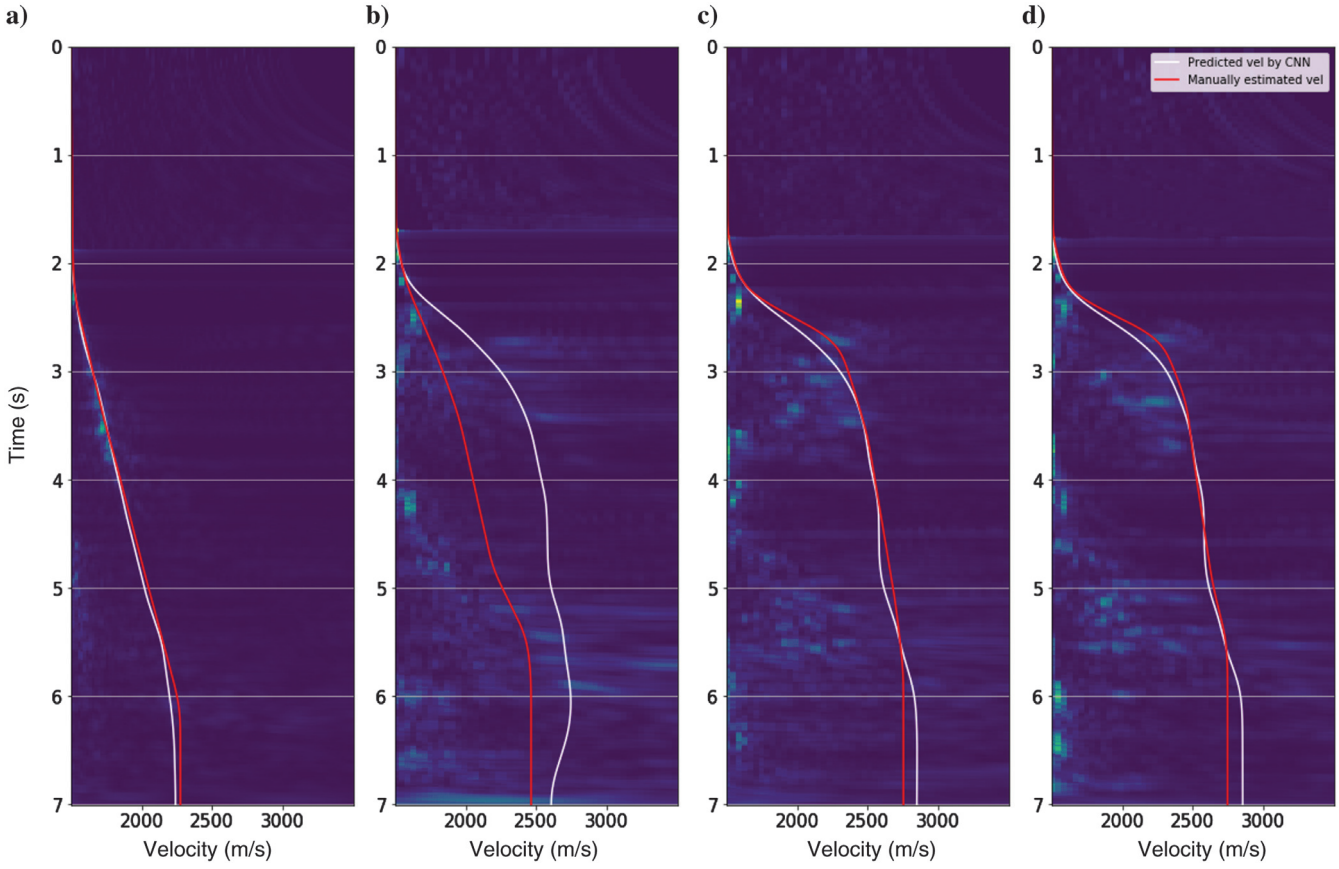


Figure 13. Semblance panels. CMP number: (a) 350, (b) 800, (c) 1250, and (d) 1400. Manual analysis velocities (red lines) and the velocities predicted via CNN (white lines). These semblance panels are not used for transfer learning.

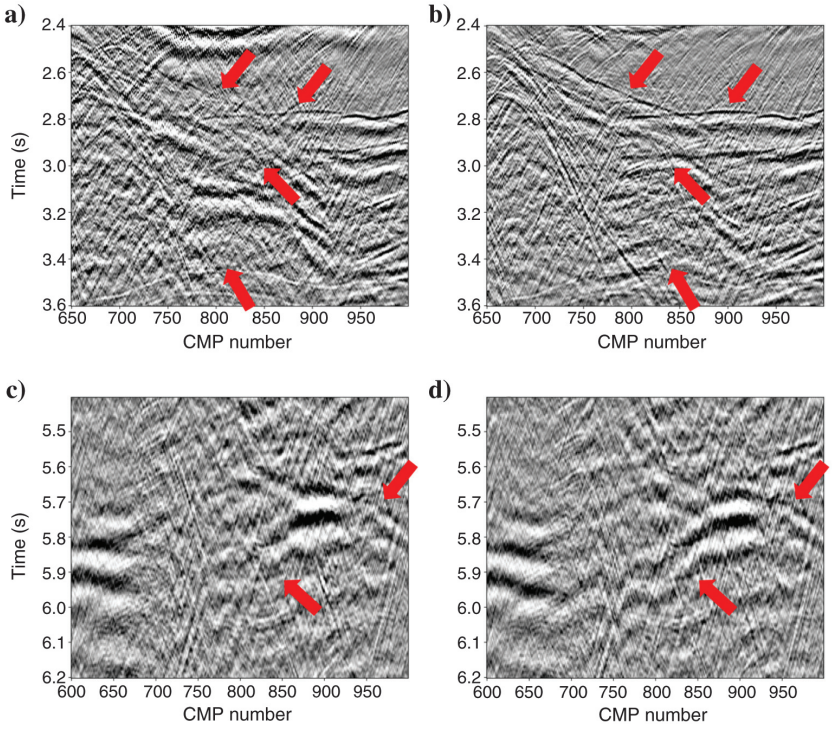


Figure 15. The magnified parts from (a and c) the manual analysis and (b and d) the CNN prediction. The red arrows indicate clearer reflectors in the magnified parts of the CNN prediction.

CONCLUSION

We have developed a method to automatically perform semblance analysis using CNN. We expect the trained CNN model to act like a professional human processor who can consider the entire trend of the semblance to avoid picking multiples. We defined the regression problem that predicts the continuous velocity as an image classification problem by dividing the velocity axis of the semblance into segments of a certain size. The latter permitted CNN to solve the problem easily. It has the potential to yield more accurate prediction using a bigger input and smaller size of compartments. Synthetic and real examples were tested with our method. The results were quite encouraging. Our method has the potential to update the base model continuously through new data. As the training data accumulate, CNN can continue to evolve. When the base model failed to predict correctly, transfer learning with a small portion of the target data was used to solve the problem.

DATA AND MATERIALS AVAILABILITY

Data associated with this research are confidential and cannot be released.

REFERENCES

Abadi, M., P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, P. Wicke, Y. Yu, and X. Zheng, 2016, Tensorflow: A system for large-scale machine learning: Symposium on Operating Systems Design and Implementation (OSDI), 265–283.

Abbad, B., B. Ursin, and D. Rappin, 2009, Automatic nonhyperbolic velocity analysis: Geophysics, **74**, no. 2, U1–U12, doi: 10.1190/1.3075144.

Araya-Polo, M., J. Jennings, A. Adler, and T. Dahlke, 2018, Deep-learning tomography: The Leading Edge, **37**, 58–66, doi: 10.1190/tle37010058.1.

Calderón-Macías, C., M. K. Sen, and P. L. Stoffa, 1998, Automatic NMO correction and velocity estimation by a feedforward neural network: Geophysics, **63**, 1696–1707, doi: 10.1190/1.1444465.

Chen, Y., 2018, Automatic velocity analysis using high-resolution hyperbolic Radon transform: Geophysics, **83**, no. 4, A53–A57, doi: 10.1190/geo2017-0813.1.

Choi, H., J. Byun, and S. J. Seol, 2010, Automatic velocity analysis using bootstrapped differential semblance and global search methods: Exploration Geophysics, **41**, 31–39, doi: 10.1071/EG10004.

Fish, B. C., and T. Kusuma, 1994, A neural network approach to automate velocity picking: 64th Annual International Meeting, SEG, Expanded Abstracts, 185–188, doi: 10.1190/1.1822888.

Fomel, S., 2009, Velocity analysis using AB semblance: Geophysical Prospecting, **57**, 311–321, doi: 10.1111/j.1365-2478.2008.00741.x.

Garcia-Garcia, A., S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia-Rodriguez, 2017, A review on deep learning techniques applied to mantic segmentation: arXiv preprint, arXiv:1704.06857.

He, K., X. Zhang, S. Ren, and J. Sun, 2016, Deep residual learning for image recognition: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 770–778.

Huot, F., B. Biondi, and G. Beroza, 2018, Jump-starting neural network training for seismic problems: 88th Annual International Meeting, SEG, Expanded Abstracts, 2191–2195, doi: 10.1190/segam2018-2998567.1.

Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei, 2014, Large-scale video classification with convolutional neural networks: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 1725–1732.

Keskar, N. S., D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. T. P. Tang, 2016, On large-batch training for deep learning: Generalization gap and sharp minima: arXiv preprint, arXiv:1609.04836.

Kingma, D. P., and J. Ba, 2014, Adam: A method for stochastic optimization: arXiv preprint, arXiv:1412.6980.

Krizhevsky, A., I. Sutskever, and G. E. Hinton, 2012, ImageNet classification with deep convolutional neural networks: Advances in Neural Information Processing Systems, **25**, 1097–1105.

Lawrence, S., C. L. Giles, A. C. Tsoi, and A. D. Back, 1997, Face recognition: A convolutional neural-network approach: IEEE Transactions on Neural Networks, **8**, 98–113, doi: 10.1109/72.554195.

LeCun, Y., 2015, LeNet-5, convolutional neural networks, http://yann.lecun.com/exdb/lenet, accessed 6 November 2019.

LeCun, Y., and Y. Bengio, 1995, Convolutional networks for images, speech, and time series, *in* M. A. Arbib, ed., The handbook of brain theory and neural networks: MIT Press 3361, 255–258.

LeCun, Y., Y. Bengio, and G. Hinton, 2015, Deep learning: Nature, **521**, 436–444, doi: 10.1038/nature14539.

LeCun, Y., B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, 1989, Backpropagation applied to handwritten zip code recognition: Neural Computation, **1**, 541–551, doi: 10.1162/neco.1989.1.4.541.

LeCun, Y. A., L. Bottou, G. B. Orr, and K.-R. Müller, 2012, Efficient backprop, *in* G. B. Orr and K.-R. Müller, eds., Neural networks: Tricks of the trade: Springer, 9–48.

Liu, D., W. Wang, W. Chen, X. Wang, Y. Zhou, and Z. Shi, 2018, Random noise suppression in seismic data: What can deep learning do?: 88th Annual International Meeting, SEG, Expanded Abstracts, 2016–2020, doi: 10.1190/segam2018-2998114.1.

Long, J., E. Shelhamer, and T. Darrell, 2015, Fully convolutional networks for semantic segmentation: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 3431–3440.

Masters, D., and C. Luschi, 2018, Revisiting small batch training for deep neural networks: arXiv preprint, arXiv:1804.07612.

Maturana, D., and S. Scherer, 2015, VoxNet: A 3D convolutional neural network for real-time object recognition: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 922–928.

Nair, V., and G. E. Hinton, 2010, Rectified linear units improve restricted boltzmann machines: Proceedings of the 27th International Conference on Machine Learning (ICML-10), 807–814.

Neidell, N. S., and M. T. Taner, 1971, Semblance and other coherency measures for multichannel data: Geophysics, **36**, 482–497, doi: 10.1190/1.1440186.

Nielsen, M. A., 2015, Neural networks and deep learning: Determination Press, 25.

Pan, S. J., and Q. Yang, 2010, A survey on transfer learning: IEEE Transactions on Knowledge and Data Engineering, **22**, 1345–1359, doi: 10.1109/TKDE.2009.191.
Pham, N., S. Fomel, and D. Dunlap, 2018, Automatic channel detection using deep learning: 88th Annual International Meeting, SEG, Expanded Abstracts, 2026–2030, doi: 10.1190/segam2018-2991756.1.
Powers, D. M., 2011, Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation: Journal of Machine Learning Technologies, **2**, 37–63, doi: 10.9735/2229-3981.
Schmidt, J., and F. A. Hadsell, 1992, Neural network stacking velocity picking: 62nd Annual International Meeting, SEG, Expanded Abstracts, 18–21, doi: 10.1190/1.1822036.
Shi, Y., X. Wu, and S. Fomel, 2018, Automatic salt-body classification using a deep convolutional neural network: 88th Annual International Meeting, SEG, Expanded Abstracts, 1971–1975, doi: 10.1190/segam2018-2997304.1.
Simard, P. Y., D. Steinkraus, and J. C. Platt, 2003, Best practices for convolutional neural networks applied to visual document analysis: IEEE International Conference Document Analysis and Recognition (ICDAR), 3.
Simonyan, K., and A. Zisserman, 2014, Very deep convolutional networks for large-scale image recognition: arXiv preprint, arXiv:1409.1556.
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.
Taner, M. T., and F. Koehler, 1969, Velocity spectradigital computer derivation applications of velocity functions: Geophysics, **34**, 859–881, doi: 10.1190/1.1440058.
Toldi, J., 1985, Velocity analysis without picking: 55th Annual International Meeting, SEG, Expanded Abstracts, 575–578, doi: 10.1190/1.1892810.
Verschuur, D., and R. Prein, 1999, Multiple removal results from Delft University: The Leading Edge, **18**, 86–91, doi: 10.1190/1.1438164.
Wang, W., F. Yang, and J. Ma, 2018, Velocity model building with a modified fully convolutional network: 88th Annual International Meeting, SEG, Expanded Abstracts, 2086–2090, doi: 10.1190/segam2018-2997566.1.
Wu, X., L. Liang, Y. Shi, and S. Fomel, 2019, FaultSeg3D: Using synthetic datasets to train an end-to-end convolutional neural network for 3D seismic fault segmentation: Geophysics, **84**, no. 3, IM35–IM45, doi: 10.1190/geo2018-0646.1.
Wu, X., Y. Shi, S. Fomel, and L. Liang, 2018, Convolutional neural networks for fault interpretation in seismic images: 88th Annual International Meeting, SEG, Expanded Abstracts, 1946–1950, doi: 10.1190/segam2018-2995341.1.
Xiong, W., X. Ji, Y. Ma, Y. Wang, N. M. BenHassan, M. N. Ali, and Y. Luo, 2018, Seismic fault detection with convolutional neural network: Geophysics, **83**, no. 5, O97–O103, doi: 10.1190/geo2017-0666.1.
Yilmaz, Ö., 2001, Seismic data analysis: Processing, inversion, and interpretation of seismic data: SEG.
Yosinski, J., J. Clune, Y. Bengio, and H. Lipson, 2014, How transferable are features in deep neural networks?: Advances in Neural Information Processing Systems, 3320–3328.
Yuan, S., J. Liu, S. Wang, T. Wang, and P. Shi, 2018, Seismic waveform classification and first-break picking using convolution neural networks: IEEE Geoscience and Remote Sensing Letters, **15**, 272–276, doi: 10.1109/LGRS.2017.2785834.
Zhao, T., 2018, Seismic facies classification using different deep convolutional neural networks: 88th Annual International Meeting, SEG, Expanded Abstracts, 2046–2050, doi: 10.1190/segam2018-2997085.1.