Scale-invariant image-recognition using convolutional neural networks and wavelet analysis

Heather Hardeman*, Matt McDonald† Michael P. Lamoureux*

ABSTRACT

We begin with a discussion of machine learning and its applications to seismic interpretation. We provide a brief overview of convolutional neural networks as well as tree-structured wavelet transforms. We introduce a new wavelet transform called the inverted tree-structured wavelet transform which renders scale-invariance for image recognition. Also, we explain our method for processing data in real-time. Then we introduce a training set extracted from real data. We test the trained convolutional neural network on a portion of the training set to determine accuracy. Afterwards, we employ this trained convolutional neural network to identify events in microseismic data.

INTRODUCTION

Seismic imaging and interpretation depends on finding events in seismic data. Geophysical scientists employ computers to process data in order to find events. Teaching computers to locate events in seismic data through image-recognition seems like a natural next step. One method for teaching image-recognition to computers is machine learning.

Over the last century, the advancement of computers led to the question of how to go about teaching computers to perform tasks. Recently, image-recognition joined the ranks of such tasks to teach computers to perform. One aspect of machine learning that is particularly useful in image-recognition are neural networks. Given the nature of seismic data, convolutional neural networks are among the most effective.

While convolutional neural networks are very effective, they are not the sole answer to recognizing images. They do a good job of defining events, but they require some help with regards to scaling. One answer to this problem is tree-structured wavelet transforms. These transforms apply a wavelet transform across a tree where each node of the tree is the data convolved with a wavelet in the wavelet family. The wavelets correspond to a specific frequency band and scale the original data to smaller sizes. Therefore, pairing convolutional neural networks with tree-structured wavelet transforms provides a scale-invariant image-recognition method.

In the following section, we will introduce neural networks with a focus on convolutional neural networks. We will deal with the concern of image scaling in image-recognition with a study of tree-structured wavelet transforms. This study leads to the introduction of inverted tree-structured wavelet transforms. In the subsequent section, we will introduce the microseismic data we will be using as the outside-samples for testing the convolutional neural network (CNN) we built with the help of code from the UFLDL Deep Learning

CREWES, University of Calgary, Department of Mathematics
Fotech Solutions

Tutorial at Stanford University (Ng et al., 2013) and (Yang, 2014). In the next section, we discuss the architecture of the CNN created to detect events, or hyperbolas in this case, in microseismic data. From there, we consider the training set and test it on a portion of known events. We then apply both the inverted tree-structured wavelet transform and CNN to both sets of microseismic data and compare the results on each level of the tree. Finally, we conclude.

NEURAL NETWORKS

The natural world provides inspiration for mathematical and computational methods. Neural networks hold a strong resemblance to the human body's nervous system (Wu, 1992). Just as the body's nervous system interprets outside stimuli to determine how the body should react or decipher information, an artificial neural network utilizes 'neurons' to make decisions, or output answers, about information fed into the network.

For the most part, the architecture of a neural network can be generalized to a connected graph of layers with units in each layer. Several different types of layers exist for neural networks, each of which performs different tasks. The type of neural network is dependent on how the units in these layers are connected to units of other layers. For larger data sets, it is important to limit these connections between units, as this decreases computational cost. One way to address the need for limiting connections is a convolutional neural network (Ng et al., 2013).

In a convolutional neural network, there is at least one convolutional layer. A convolutional layer has a number of units equal to the number of filters set when training the network. The choice of filters determines the number of weights in the layer; it is also necessary to fix a dimension for the filters. For improved accuracy, the dimension of the filters should be the size of the event being trained to detect. In the example in the Applications section, we wish to detect hyperbolas extracted from a data set of someone walking parallel to a fibre-optic cable. The hyperbolas are approximately 91×91. As such, we set the dimension of the filters to be 91×91 when training the convolutional neural network. These filters are convolved through the data and produce a smaller set of convolved features. These convolved features save the computational cost of processing the entire data set in a fully-connected network.

After the convolutional layer, the results from each unit are directed to the next layer. This layer can be another convolutional layer or a pooling layer. Depending on the size of the data, it may be prudent to have another convolutional layer. A pooling layer takes the convolved features and pools them together using different methods to produce a pooled feature. This layer condenses the data further, cutting down on computational costs and allowing for much faster classification. Some methods employed for pooling include max-pooling or mean-pooling. The max-pooling involves taking the maximum of each convolved feature and the mean-pooling takes the average of each convolved feature. Not only does pooling reduce the dimension of the data, but it also allows for less overfitting which improves results.

Depending on the complexity of the convolutional neural network, the pooling layer can feed into another convolutional layer or pooling layer. Our convolutional neural network will feed into a densely-connected output layer. The output layer gives the results or predictions of the neural network. For example, in hand-written digit recognition, there would be 10 output units, one for each number between 0 and 9. In this paper, our CNN has two output units in the output layer; one for if an image has a hyperbola and another if it does not have a hyperbola.

An important note is how exactly these networks are trained. Networks are trained by inputing learned parameters. The weights involved in the convolutional layer are part of these learned parameters. In order to learn the parameters for a neural network, it is necessary to have a training set. It follows that if a network must learn to predict events it needs training objects containing samples of the events which should be predicted. In image recognition, the training set contains images of events the user wants to detect.

Parameters can be taught using gradient methods. These gradient methods must converge to a local optima which is why batch methods are often used. We utilized the stochastic gradient method to learn our parameters. The stochastic gradient method overcomes the slow calculation common to batch methods. The stochastic gradient method learns parameters by computing the gradient of the cost function $J(\theta)$ over a few batches of training images as opposed to the entire training set to update the parameters after each iteration. For this paper, the cost function J is the average of the least squares difference between the hypotheses of each sample x and the label y for that sample with a regularization term (Ng et al., 2013). A learning rate α is chosen small prior to computation and is updated after each iteration. For the stochastic gradient method, it is also necessary to pick a momentum. The momentum helps push the stochastic gradient descent objective down any shallow ravines present in the calculation in order to improve performance of the method. As with the learning rate, the momentum is updated after each iteration. The initial choice of the momentum is around 0.5 before being chosen between 0.9 and 1. The stochastic gradient descent outputs learned parameters after a user-determined number of loops through the training set. The user then employs the learned parameters in the neural network to classify events.

For a detailed tutorial on neural networks and other deep learning methods, please see (Ng et al., 2013).

TREE-STRUCTURED WAVELET TRANSFORMS

In neural networks, necessity requires training images to be small. If the images in a training set are too big, then the computational cost increases. The small nature of the training images poses a potential issue given that most data sets are much larger than a training image. It becomes necessary to window the data set in some way where the windows are the size of the training images. One answer to this windowing issue is tree-structured wavelet transforms.

Chang and Kuo introduce tree-structured wavelet transforms in (Chang and Kuo, 1993) for application in texture analysis. Tree-structured wavelet transforms are tree-structured algorithms which implement a wavelet transform. The top level of the tree is the data.

The tree branches into several nodes depending on the number of wavelets in the wavelet family. Each subsequent level applies the wavelet transform to each node and then produces new nodes on the next level of the tree. Fig. 1 is an example of a tree-structured wavelet transform, or a wavelet tree.

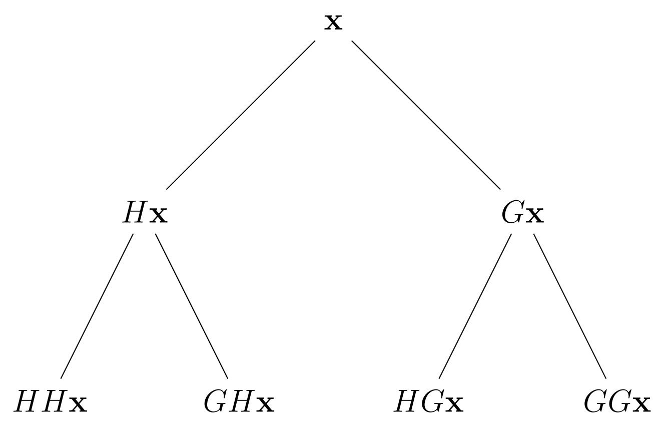


FIG. 1. An example of a wavelet tree using the Haar transform. The data $\mathbf{x}$ represents the data, $H\mathbf{x}$ represents the approximation coefficients, $G\mathbf{x}$ represents the detailed coefficients.

The results at each node are the convolution of specific wavelets, chosen from within a family of wavelets, with the data. Each wavelet relates to a specific frequency band of the data (Bruns, 2004). In the case of the Haar transform, there is only the approximation coefficients and detail coefficients. Thus, the size of the data is reduced by a power of two in the x-domain when the transform is applied. The results are two scaled processed versions of the original data set which we assign to a node in the subsequent level of the wavelet tree. Scaling the data in this manner provides us with an answer to dealing with scale-invariance in image recognition techniques.

Depending on the size of the data set, which generally can be quite large, the wavelet tree will need several levels in order to approach the size of a training image for the convolutional neural network. For example, if a training image is 128×128 and the original data $\mathbf{x}$ has $2^{14} = 16384$ bins in the x-domain, then the wavelet tree needs 7 levels in order to reduce the size of the data to 128 bins in the x-domain. The number of nodes N in a level is determined by the equation

$$N = Q^{m-1} \tag{1}$$

where Q is the number of wavelets in the wavelet family and m is the level of the tree the node resides. Thus, the seventh level of a wavelet tree using the Haar transform will have $2^6 = 64$ nodes. Also, recall that there are 6 other levels with 2^{m-1} nodes to take into account when considering the computational costs of this method. It is clear that this approach can become computationally expensive very quickly. As such, we introduce the inverted tree-structured wavelet transform.

An inverted tree-structured wavelet transform has the same basic principle as the tree-structured wavelet transform. The top level contains the original data and successive levels contain the results of applying a specific wavelet from the wavelet family to the data in the node. The main difference is that our top level has 2^{M-1} nodes, where M is the max number of levels in the tree. Each of the nodes in the top level holds a $N \times D$ strip of data where N is some power of two that divides the x domain of the data at least twice and D is the size of the data in the t-domain. The next level of the data is produced by pairing the first node with the second and then pairing each subsequent node to its successor if it is not already paired with the node that directly precedes it and applying the wavelet transform. The new node is the result of the wavelet transform applied to the concatenation of the data in the two nodes in the previous level. The next level of the tree contains 2^{M-m} nodes where m is the level of the tree the node resides. For example, if we have an inverted wavelet tree with a maximum of three levels, then the top level has $2^2 = 4$ nodes, the second level has $2^1 = 2$ nodes and the last level has $2^0 = 1$ nodes.

Mathematically, let $\mathbf{x}$ be a $P \times D$ matrix which represents the data set. The inverted tree-structured wavelet transform consists of creating a tree T with M levels and 2^{M-m} nodes in each level where m is the current level of the tree T. For each node $a_{1,k}$ in level 1 with $k \in \{1, ..., 2^{M-1}\}$,

$$\begin{aligned} a_{1,k} &= \mathbf{x}(1 + (k-1)\xi : k\xi, :) && (2) \\ &= \{x_{i,j} : 1 + (k-1)\xi \le i \le k\xi, 1 \le j \le D\} && (3) \end{aligned}$$

where $\xi = 2^\ell < P/2$ for $\ell \in \mathbb{N}$. In levels $1 < m \le M$, the nodes

$$a_{m,k} = H[a_{m-1,k}\ a_{m-1,k+1}] \tag{4}$$

where $a_{m-1,k}$ and $a_{m-1,k+1}$ are parent nodes from the previous level $m - 1$. The term H indicates which wavelet is used when applying the wavelet transform to the matrix $[a_{m-1,k}\ a_{m-1,k+1}]$. Fig. 2 provides an example of a 3-level inverted wavelet tree.

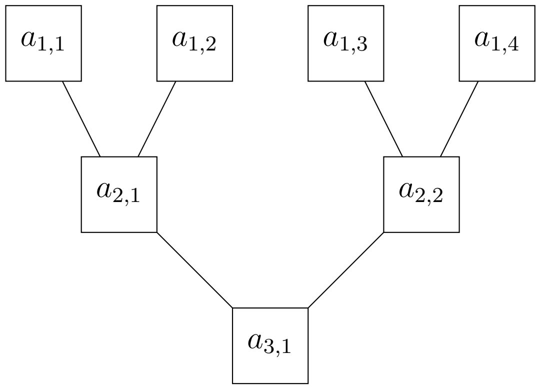


FIG. 2. An example of an inverted wavelet tree.

Applying the inverted tree-structured wavelet transform to convolutional neural networks dictates that ξ is equal to the dimension of the training images. Training images for convolutional neural networks are by necessity quite small in comparison to data sets. In order to limit the computational cost of applying inverted wavelet trees to the data in order to achieve scale invariance, we need to window the tree through the data. This idea entails redefining the nodes in the top level. The data in the first two nodes is thrown out and replaced with the data from their succeeding two nodes. After moving forward the final two

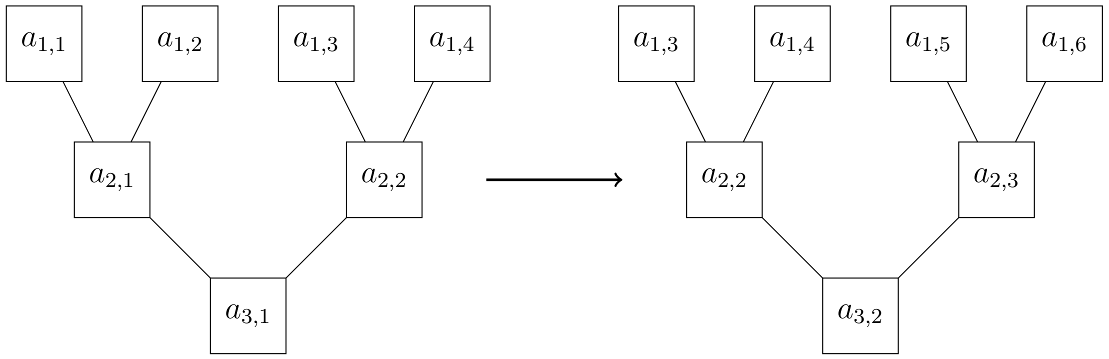


FIG. 3. The right inverted wavelet tree shows the position of each node before windowing occurs. The left inverted wavelet tree shows the position of each node after windowing occurs. After windowing, the information in nodes $a_{1,1}$ and $a_{1,2}$ is thrown out along with their children nodes and the grandchild node. Then, the information is replaced with the data from nodes $a_{1,3}$ and $a_{1,4}$. The node $a_{2,2}$ follows its parents to replace node $a_{2,1}$. The nodes $a_{1,5}$ and $a_{1,6}$ represent the new data which replaces the original information in these nodes. After applying a wavelet transform to the appropriate nodes, children nodes $a_{2,3}$ and $a_{3,2}$ are created.

nodes in the top level, then the next portions of the original data set are placed in the last two nodes. The children nodes for each pair of nodes in the top level follow their parents over in their respective level. As such, when we apply the wavelet transform, it need only be applied to the final two nodes of level 1 and the subsequent branch. Thus, a windowing of the data set is achieved. Fig. 3 depicts this windowing process.

Mathematically speaking, suppose we have a tree T with n levels. Therefore, Level 1 of T has $N = 2^{n-1}$ nodes. The windowing for the tree T in the first level occurs by letting nodes $a_{1,k} = a_{1,k-2}$ and $a_{1,k+1} = a_{1,k-1}$ where n is the level and $k \in \mathbb{N}$. If $k - 2 \leq 0$, then the data from nodes $a_{1,k}$ and $a_{1,k+1}$ is thrown away. We define the final nodes in the level as

$$\begin{aligned} a_{1,N-1} &= \mathbf{x}(1 + N\xi : (N+1)\xi, :) && (5) \\ &= \{x_{i,j} : 1 + N\xi \leq i \leq (N+1)\xi, 1 \leq j \leq D\} && (6) \end{aligned}$$

and

$$\begin{aligned} a_{1,N} &= \mathbf{x}(1 + (N+1)\xi : (N+2)\xi, :) && (7) \\ &= \{x_{i,j} : 1 + (N+1)\xi \leq (N+2)\xi, 1 \leq j \leq D\}. && (8) \end{aligned}$$

The children of nodes $a_{1,k}$ for $2 < k \leq N$ follow their parents in their respective level. Thus, the only computation necessary for updating the tree is the branch from the newly defined nodes $a_{1,N-1}$ and $a_{1,N}$. Applying the wavelet transform through this new branch produces the new inverted wavelet tree where the nodes

$$\begin{aligned} a_{2,N^{1/2}} &= H[a_{1,N-1}, a_{1,N}], \text{ and} && (9) \\ a_{3,N^{1/4}} &= H[a_{2,N^{1/2}-1}, a_{2,N^{1/2}}]. && (10) \end{aligned}$$

The windowing of data set through the inverted tree-structured wavelet transform limits the size of the wavelet trees and thus reduces the computational cost it requires to process the data. Furthermore, it allows for translation-invariance. It is important to note that the nodes in subsequent levels can be the results of applying any wavelet from a wavelet family to the node as long as the same wavelet is applied throughout the tree. We chose to use the approximation coefficients from the Haar transform in the experiments in the following sections.

DATA

We apply the convolutional neural network and wavelet tree analysis to two microseismic data sets acquired by Fotech Solutions using distributed acoustic sensing and fibre optics. Fig. 4 shows the first data set we consider later and Fig. 5 shows the second data set we consider further in the report.

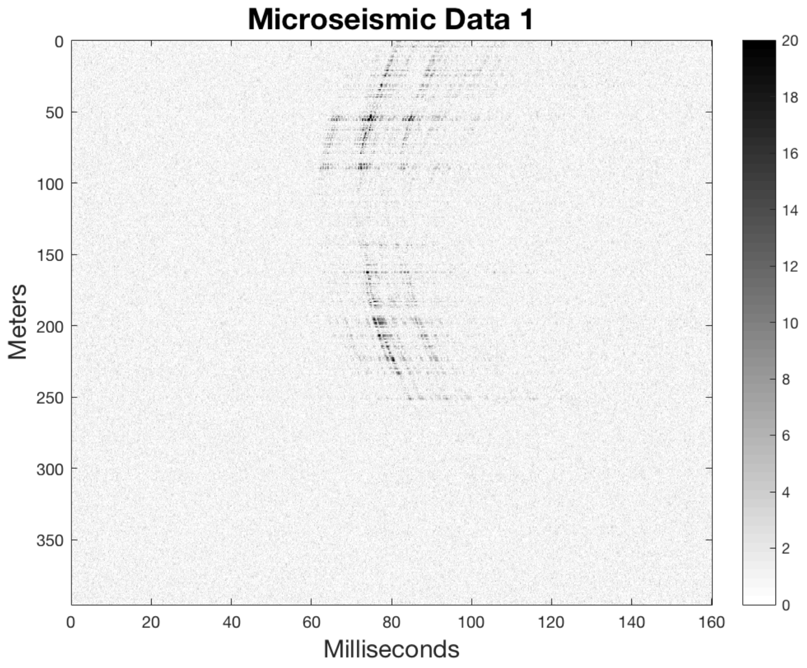


FIG. 4. The first microseismic data set to which we will apply the CNN.

APPLICATIONS

In the following sections, we will apply a CNN we developed over the last year to the microseismic data sets we discussed in the previous section. We developed our CNN using supplementary code from (Ng et al., 2013) and (Schmidt, 2005). In the following sections, we will analyze the structure of the convolutional neural network we built. We will consider the training set we created to train the convolutional neural network to detect hyperbolas and test the trained CNN on a portion of the training set. Finally, we apply the trained CNN to detect hyperbolas in microseismic data.

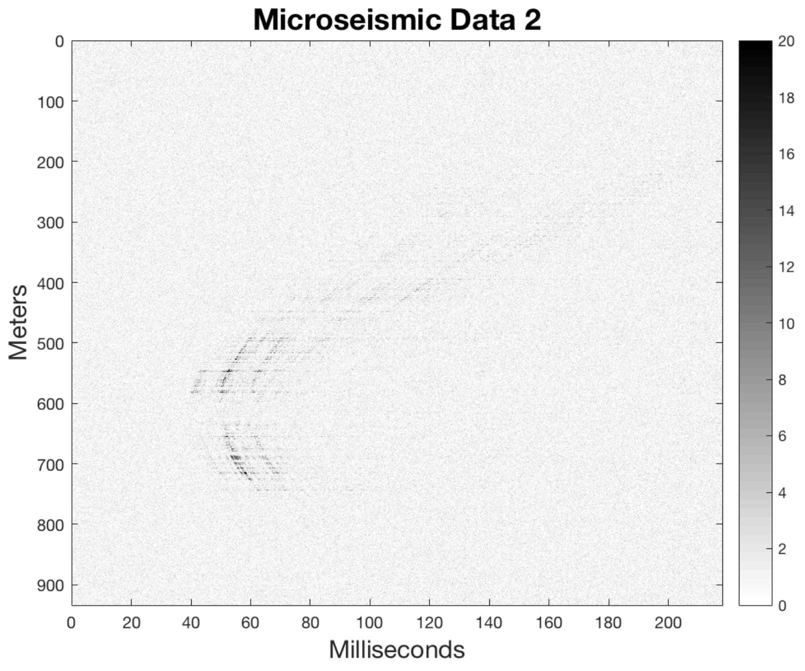


FIG. 5. The second microseismic data set to which we will apply the CNN.

Convolution Neural Network Architecture

We built a convolutional neural network using the UFLDL Tutorial from Stanford University (Ng et al., 2013). Due to time constraints, we used code for the cost function and stochastic gradient descent based on (Yang, 2014). The convolutional neural network has three layers, not including the input layer. The first layer was a convolutional layer which utilizes 20 filters of dimension 91×91. The second layer was a pooling layer that applied a mean-pooling. The outputs were propagated into a softmax regression with cross entropy objective. The dimension of the pooling features was 2×2. The final layer was the output layer which contained two units determining one of two cases: whether there was a hyperbola present or whether there was not a hyperbola present in the image.

We employed a stochastic gradient descent method using momentum in order to optimize the parameters of a given objective. This function involved using the minFunc function created by Mark Schmidt (Schmidt, 2005). The stochastic gradient descent method calculated the learned parameters to train the convolutional neural network to detect hyperbolas in data sets based on the cost function $J(\theta)$ and allowed for a faster convergence. We set the momentum at 0.99. To train the neural network, we used the learning rate $\alpha = 0.01$. The SGD subsampled over batches containing 32 training samples. The training ran through the training set three times in order to educate the convolutional neural network.

Training set

Training sets are an important component of neural networks. In image recognition, these sets are comprised of various images which are used to train the neural network to identify specific objects. In this project, we taught the convolutional neural network to identify hyperbolas given that geophysical anomalies in microseismic often take the shape of a hyperbola.

We created a training set by extracting images from real data. The real data training set was segemented from the data in Fig. 6. This image is of someone walking parallel to a length of fibre-optic cable and was acquired using distributed acoustic sensing. Upon close examination of each step individually, one finds a hyperbola. We removed each of these hyperbolas to create a training set. This training set contains 528 images of size 128×128. The set contains 371 hyperbolas and 157 non-hyperbolas. We defined an image as a non-hyperbola if it did not contain a complete hyperbola in the window. The top row of Fig. 7 shows an example of an image in the real data training set which is considered a hyperbola while the bottom row depicts examples of images which are considered non-hyperbolas.

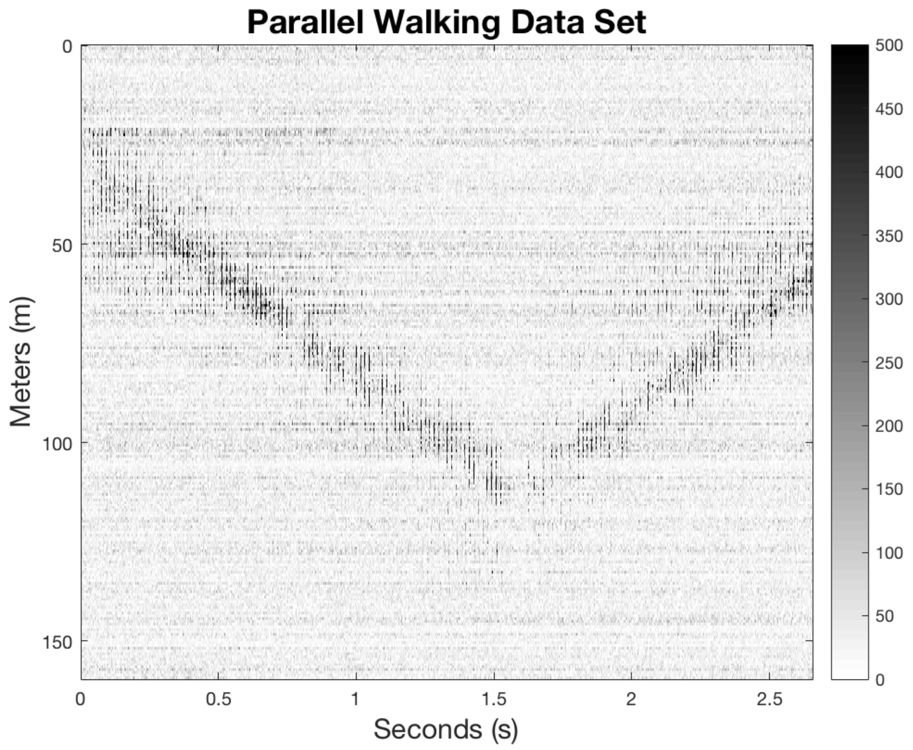
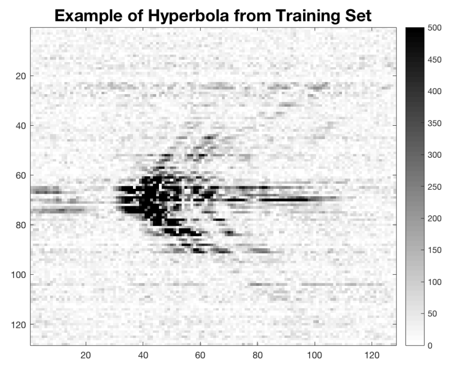


FIG. 6. An image of someone walking parallel to a fibre-optic cable acquired using distributed acoustic sensing. The parallel walking data set is employed to create the training set for the convolutional neural network.

Testing on Training Set

To help determine the accuracy of our trained convolutional neural network, we trained the CNN on 400 images in the training set. Afterwards, we tested it on the remaining 128 images. Fig. 8 depicts the probabilities that an image is a hyperbola or not a hyperbola for each image in the test set. As the reader can see, the CNN remained confident for determining hyperbolas giving a probability of approximately 1 for many of the images. For about 13 images, the CNN decided they were not hyperbolas. Comparing the predictions



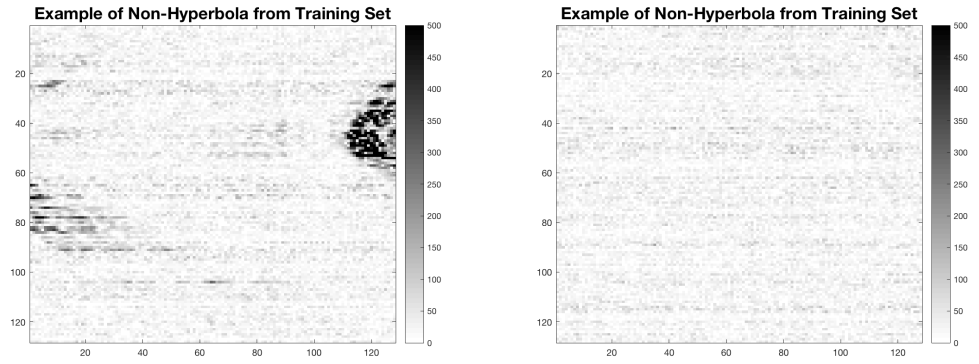


FIG. 7. (Top) An image of a hyperbola from the training set. (Bottom) Examples of non-hyperbolas from the training set.

of the CNN to the manually applied labels, we get an accuracy of 85.16%.

Fig. 9 (left) gives an example of a image the CNN determined was a hyperbola whereas Fig. 9 (right) gives an example of an image the CNN predicted was not a hyperbola. In both cases, the CNN predicted the image accurately.

Testing on Outside Sample

Now, we apply the convolutional neural network that we have trained and tested previously to the microseismic data we shared in the Data section. We begin by considering the microseismic data set from Fig. 4. We now need to implement the inverted wavelet tree in order to apply the CNN. The microseismic data set is a whole set as opposed to the test case in the previous section where we had a number of segemented images the size of the training images. We need to window the CNN through the microseismic data. Furthermore, given the size of the data set, it is necessary to downsample the data set in order to analyze it using the CNN.

Fig. 10 provides an image of the downsampled version of the data pictured in Fig. 4. In this case, we downsampled by 75%.

Given the size of the downsampled data, we built a 3-level inverted wavelet tree which applies the Haar transform producing the approximation coefficients at the nodes in the inverted wavelet tree. Fig. 11 gives the probabilities of whether or not a hyperbola is

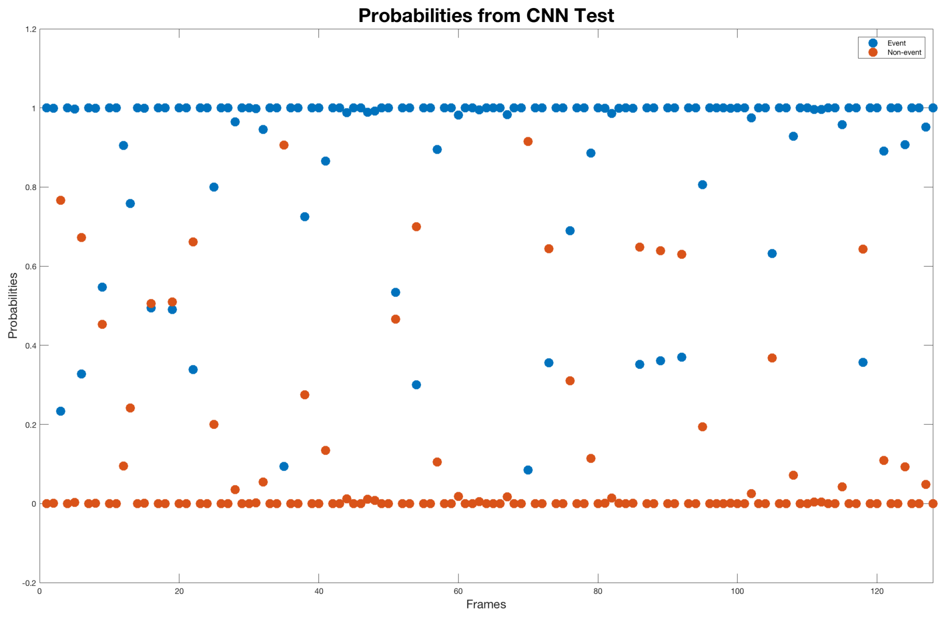


FIG. 8. The probability that each image in the remaining 128 images of the training set is a hyperbola (described by a blue dot) or a non-hyperbola (described by an red dot).

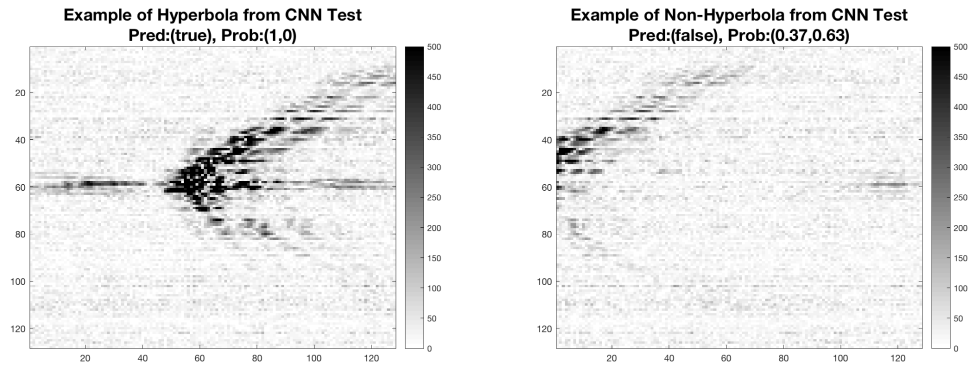


FIG. 9. (Left) An example of a hyperbola the CNN predicted from the remaining training set images. (Right) An example of a non-hyperbola the CNN predicted from the remaining training set images.

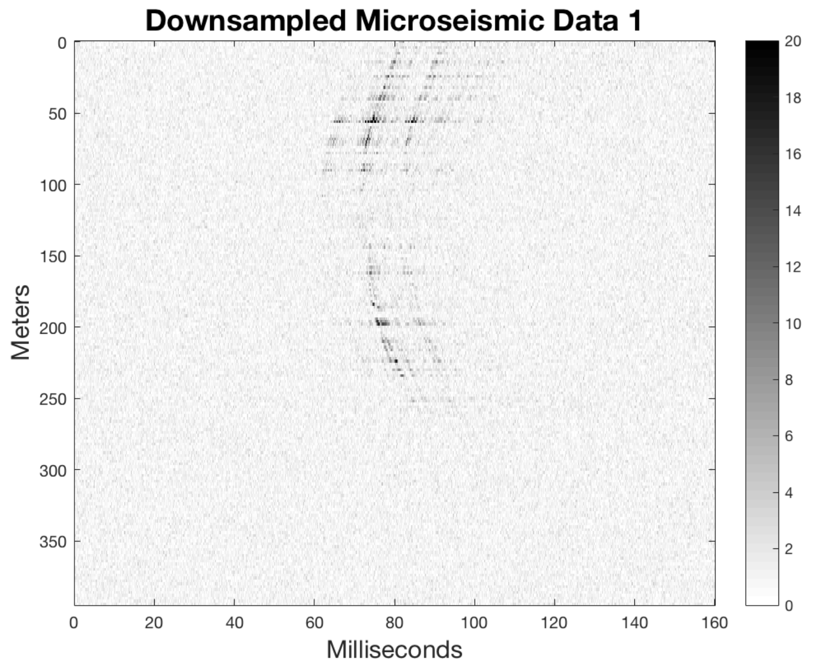


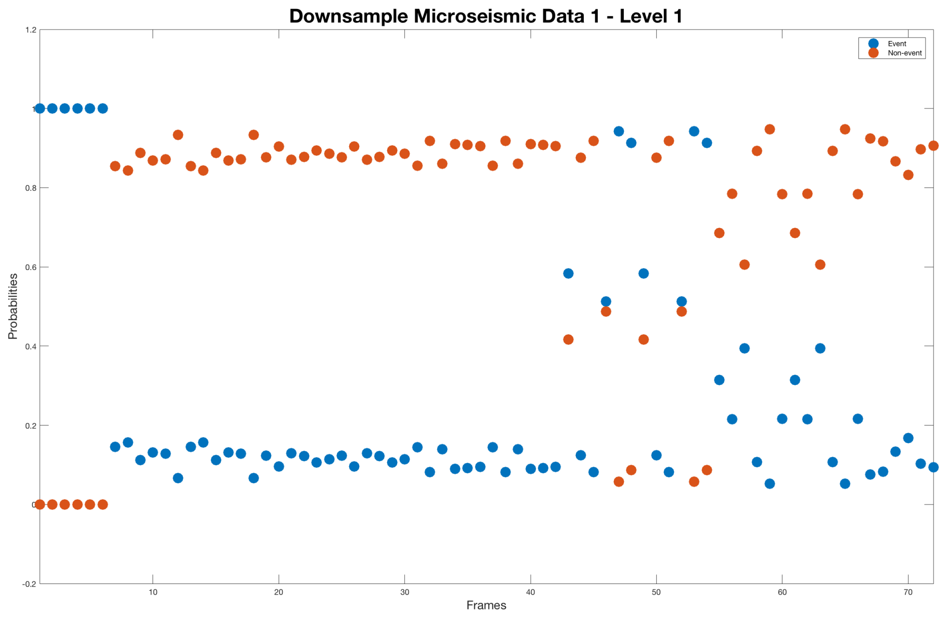
FIG. 10. The downsampled version of the microseismic data from Fig. 4.

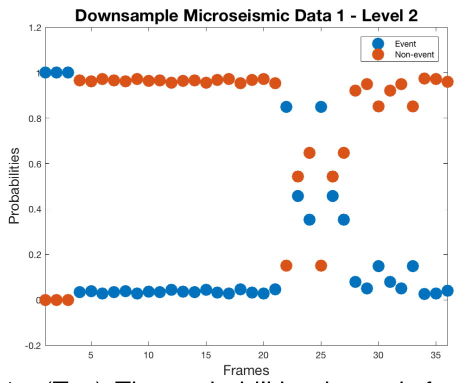
present in the frame at each level. The red dots represent the probability that a hyperbola is not present and the blue dots represent the probability that a hyperbola is present. There are only predictions for hyperbolas in Level 1 and Level 2 which gives some idea as to the size of the events in the data. The location of these predictions suggest that there are hyperbolas in the data correlating to Frames 40 to 55 in Level 1 and Frames 20 to 27 in Level 2. Moreover, the scale of the events is somewhere between the scale in Level 1 and the scale in Level 2. While Level 3 has no predictions for hyperbolas, some variation in prediction resides between Frames 9 and 16 which may suggest an event; however, the event does not fit the scale provided in Level 3.

Fig. 12 provides examples of what our trained CNN determined was a hyperbola versus a non-hyperbola. As the reader can see in the figure, we have programmed the code to say ‘true’ if a hyperbola is present and ‘false’ if a hyperbola is not present.

We also applied the CNN to the second microseismic data set (Fig. 5) from Data section. As with the first data set, we needed to utilize an inverted wavelet tree as well as downsample the data by 75% in order to apply the CNN; see Fig. 13.

As with the previous data set, we built a wavelet tree with three levels and applied the Haar transform, taking the approximation coefficients for the nodes of the tree. Fig. 14 shows the probabilities for identifying hyperbolas in the second downsampled microseismic data set at each level. None of the levels of the wavelet tree predict a hyperbola in the data set. There is significant activity in Level 1 between Frames 20 and 60. The main activity in Level 2 is between Frames 10 and 15; and in Level 3, the activity is isolated around Frame 6. This could suggest that there is some event in the corresponding locations of the data set; however, it is not the definitive answer that we need. Potentially, the lack of





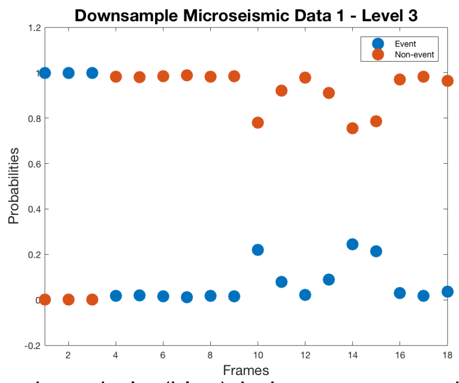


FIG. 11. (Top) The probabilities in each frame of a hyperbola (blue) being present and a non-hyperbola (red) being present on Level 1. (Bottom left) The probabilities in each frame of a hyperbola (blue) being present and a non-hyperbola (red) being present on Level 2. (Bottom right) The probabilities in each frame of a hyperbola (blue) being present and a non-hyperbola (red) being present on Level 3.

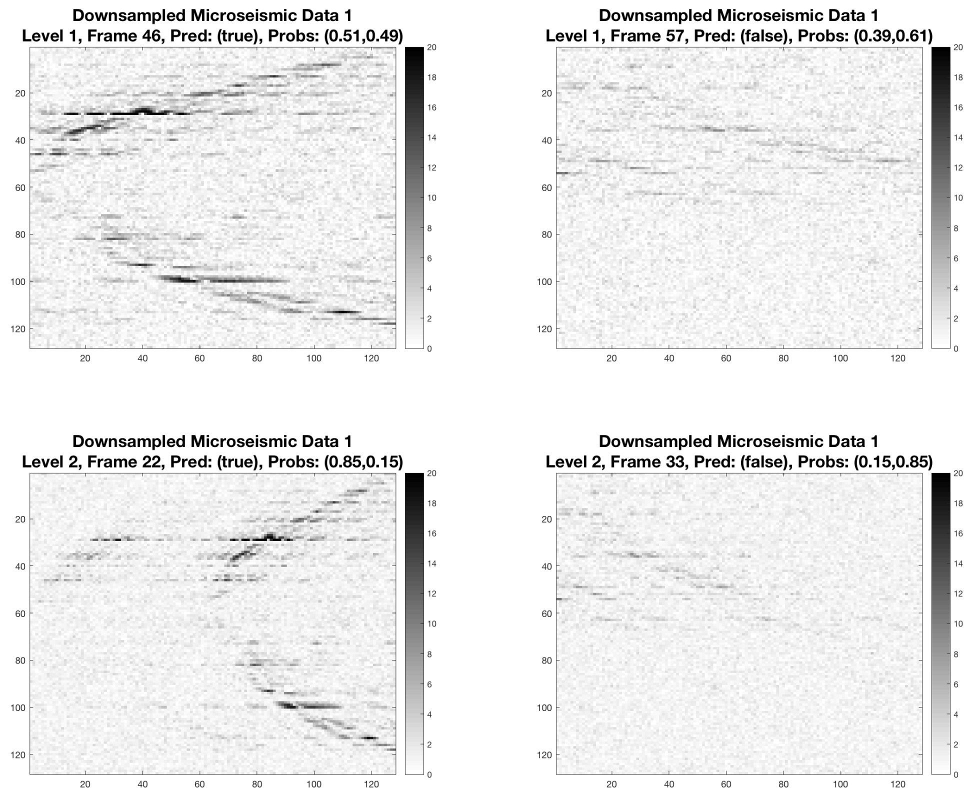


FIG. 12. Images of hyperbolas and non-hyperbolas from the first downsampled microseismic data set predicted by the trained CNN. (Top left) An example of a hyperbola on Level 1 of the inverted wavelet tree. (Top right) An example of a non-hyperbola on Level 1 of the inverted wavelet tree. (Bottom left) An example of a hyperbola on Level 2 of the inverted wavelet tree. (Bottom right) An example of a non-hyperbola on Level 2 of the inverted wavelet tree.

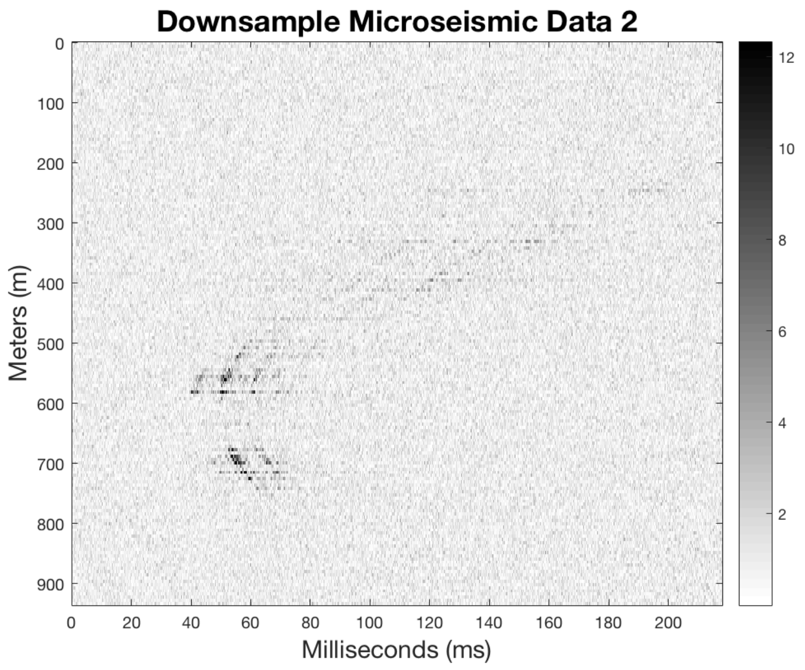
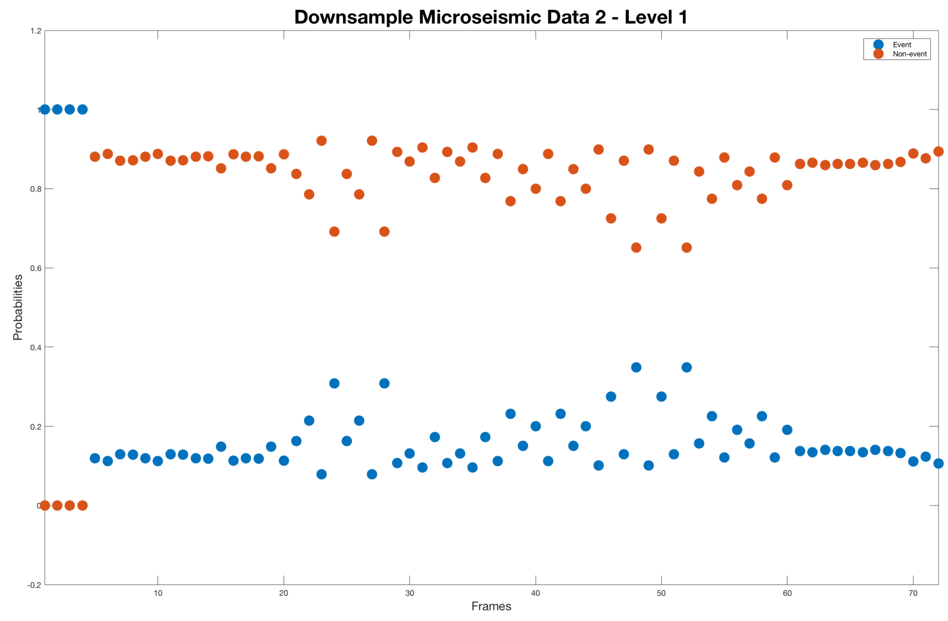
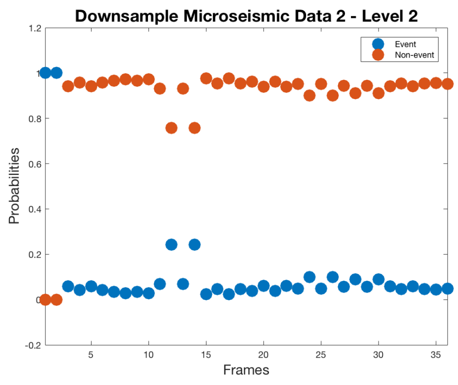


FIG. 13. The downsampled version of the microseismic data from Fig. 5.

hyperbolas could mean that the events did not fit in any level for the inverted wavelet tree's scaling. It may be necessary to downsample the microseismic data by more than 75% in this case unlike in the previous case. Otherwise, the events in this data set may not be as hyperbolic as the CNN has been taught that they should be.





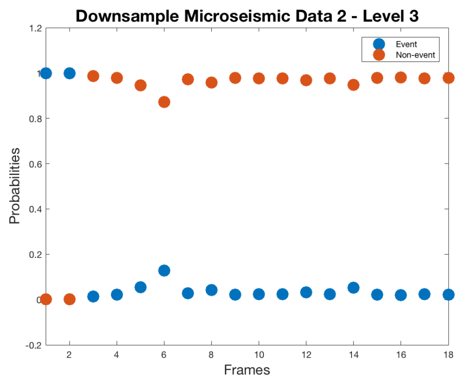


FIG. 14. Probabilities of hyperbolas and non-hyperbolas from applying the trained CNN to the second downsampled microseismic data set. (Top) The probabilities in each frame of a hyperbola (blue) being present and a non-hyperbola (red) being present on Level 1. (Bottom left) The probabilities in each frame of a hyperbola (blue) being present and a non-hyperbola (red) being present on Level 2. (Bottom right) The probabilities in each frame of a hyperbola (blue) being present and a non-hyperbola (red) being present on Level 3.

For both Microseismic Data 1 and Microseismic Data 2, the convolutional neural network predicted false positives for the first 3 to 4 frames of inverted wavelet tree. This error potentially results from the fact that the first few frames of the inverted wavelet tree are zero matrices. Given that we are using the stochastic gradient descent method, the false positives could stem from the application of the stochastic gradient descent to a zero matrix. Since all the values of a zero matrix are identical, then all the weights for the units in the hidden layer will be the same, thus pushing the same value through to the output layer (Ng et al., 2013). In the case of those first few frames on each level, it pushes through a 'true' prediction.

FUTURE WORK

For future work, it would be of interest to build a convolutional neural network containing more convolutional layers as well as pooling layers. A convolutional neural network with more layers may be able to provide better image recognition. Another way in which we could improve image recognition is to acquire more training samples of hyperbolas for our training set. Futhermore, we would consider training the neural network to identify patterns of hyperbolas instead of simply hyperbolas.

CONCLUSIONS

We began with a short study of neural networks with an emphasis on convolutional neural networks for the purposes of image-recognition. We discussed the tree-structured wavelet transform and recognized its potential to provide scale-invariance in image-recognition. We introduced the inverted tree-structured wavelet transform. We also discussed the application of convolutional neural networks to inverted wavelet trees. Furthermore, we considered the architecture of the convolutional neural network we used for experiments on microseismic data. We introduced the training set for the neural network and the parameters we chose to train the CNN. We saw that the CNN had approximately 85% accuracy for detecting hyperbolas in the testing set. Then, we applied the CNN and the inverted wavelet to two sets of microseismic data. In one case, we were able to detect events in the data set; in the other data set, we were unable to detect events. Finally, we discussed the false-positives for both microseismic data sets in the first few frames of the analysis and concluded that the stochastic gradient descent method failed for the zero matrices in the first few nodes of the inverted wavelet tree.

ACKNOWLEDGEMENTS

We thank the sponsors of CREWES for their support. We also thank our collegues at Fotech as well as CREWES for their support. We also gratefully acknowledge support from NSERC (Natural Science and Engineering Research Council of Canada) through the grant CRDPJ 461179-13 and through a Discovery Grant for the third author. Finally, we gratefully acknowledge the support from NSERC (National Science and Engineering Research Council of Canada) through the Engage Grant - EGP 494101 - 16.

REFERENCES

Bruns, A., 2004, Fourier-, hilbert- and wavelet-based signal analysis: are they really different approaches?: Journal of Neuroscience Methods, **137**, 321–332.

Chang, T., and Kuo, C.-C. J., 1993, Texture analysis and classification with tree-structured wavelet transforms: IEEE Transactions on Image Processing, **2**, 429–441.

Ng, A., Ngiam, J., Foo, C. Y., Mai, Y., Suen, C., Coates, A., Maas, A., Hannun, A., Huval, B., Wang, T., and

Tandon, S., 2013, Ufldl: Welcome to the deep learning tutorial.
URL `http://ufldl.stanford.edu/tutorial/`

Schmidt, M., 2005, minfunc.
URL `https://www.cs.ubc.ca/ schmidtm/Software/minFunc.html`

Wu, B., 1992, An introduction to neural networks and their applications in manufacturing: Journal of Intelligent Manufacturing, **3**, 391–403.

Yang, B., 2014, Bin yang.
URL `https://github.com/byangderek`