PREPROCESSING-FREE SURFACE MATERIAL CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS PRETRAINED BY SPARSE AUTOENCODER

Mengqi Ji[1], *Lu Fang*[1,2], *Haitian Zheng*[2], *Matti Strese*[3], *Eckehard Steinbach*[3]

[1] Hong Kong University of Science and Technology, {*mji,eefang*}*@ust.hk*
[2] University of Science and Technology of China, *zhenght@mail.ustc.edu.cn*
[3] Technische Universität München, {*matti.strese, eckehard.steinbach*}*@tum.de*

ABSTRACT

Acceleration signals captured during the interaction of a rigid tool with an object surface carry relevant information for surface material classification. Existing methods mostly rely on carefully designed perception-related features or features adapted from audio processing motivated by the observed similarity between acceleration signals and audio signals. In contrast, our proposed method automatically learns features from RAW acceleration data without preprocessing. The approach is based on Convolutional Neural Networks (CNN) trained and tested on RAW data. For better performance and faster convergence of the CNN, we use the weights of a trained sparse Autoencoder (AE) to initialize the weights of the first convolution layers of the CNN. This strategy is named CNN pretrained by sparse AE (ACNN). Our classification results on a publically available Haptic Texture Database demonstrate that the proposed algorithm performs favorably against existing methods.

Index Terms— haptic texture classification, convolutional neural networks, CNN pretraining, sparse Autoencoder

1. INTRODUCTION

Acceleration signals captured while a rigid tool moves over an object surface reflect the characteristics of the surface material such as hardness, roughness, etc [1]. Utilizing acceleration signals to recognize the surface material and to eventually recreate the haptic feeling of the real surface recently attracted high interests [1, 2, 3, 4, 5]. Existing approaches typically require carefully designed perceptual features, such as roughness and hardness, and specific preprocessing operations. For example, the data-driven approach proposed in [2] works with specifically designed preprocessing procedures, such as thresholding and peak detection, resulting in a tight coupling of the deployed perceptual features to the specific classification task. Other approaches [1] take advantage of mature acoustic features widely used in audio processing

This work has been supported in part by a Research Fellowship for Lu Fang by the Alexander von Humboldt Foundation.

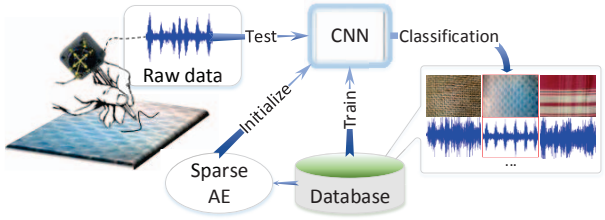


Fig. 1. Structure of the proposed ACNN.

[6, 7]. For the existing approaches, preprocessing the acceleration signals is an indispensable step. For example, the raw acceleration data is converted into Mel-frequency cepstral coefficients (MFCC) in [1] or in Spectrogram [6, 7] before being fed into the classifier. In spite of the satisfactory performance of the existing methods, the hand-tuning of their preprocessing phase tends to be tedious.

Examining the preprocessing procedure, the key purpose is to help the following classifier or detector to adapt to different input data. Consequently, to achieve high classification accuracy while reducing the burden of parameter tuning in preprocessing is of broad interest. In other words, classification algorithms that work without task-specific preprocessing lead to considerable labor liberation on parameter-tuning and can be easily utilized in other classification tasks, such as musical note detection and speech recognition. Standing on such prospect, we propose to use Convolutional Neural Networks (CNN) to automatically learn valid features for the task of surface material classification in contrast to previous methods that emphasize on designing task-specific, hand-crafted features. Specifically, as illustrated in Fig.1, the input of the proposed CNN classifier is the normalized RAW acceleration data without any carefully designed preprocessing operations. The CNN acts simultaneously as a feature extractor and a classifier in the proposed structure with the recognized real surface class being the output. The main contributions of our work are summarized as below:

1. We present a novel convolutional neural network for surface material classification with RAW acceleration data as input, which removes the need for laboriously hand-engineered features in the preprocessing phase.

978-1-4673-7454-5/15/$31.00

2. In particular, we use the weights of a trained sparse Autoencoder (AE) to partially initialize the weights of the CNN for faster convergence and higher classification accuracy. The proposed deep learning structure is named as CNN pretrained by sparse AE (ACNN) and is shown in Fig.1.

The paper is organized as follows. In Section 2, the structure of the proposed ACNN is discussed. We show how the CNN pretraining using the sparse AE's weights works and how the ACNN is trained. Experimental results showing the advantage of the ACNN and the necessity of the trained sparse AE are reported in Section 3. Finally, conclusions and future work are summarized in Section 4.

2. SURFACE CLASSIFICATION USING THE PROPOSED ACNN STRUCTURE

2.1. The ACNN Structure

The overall framework of the proposed ACNN is illustrated in Fig.1, which shows the three modules: CNN, Sparse AE and Haptic Texture Database. The RAW acceleration data recorded while scratching with a rigid tool on an object surface is directly fed into the proposed CNN which is pretrained using a sparse AE. The input signal is classified as a real surface class within the database.

• **CNN** is the trainable multi-stage feed-forward artificial neural network [8, 9], which has been extensively studied in the past years to extract good feature representations for supervised learning.

Our motivation is to avoid the necessity for sophisticated features. In our case, the input of the ACNN is one-dimensional acceleration data fragments which are obtained by converting the original three-dimensional acceleration data to maintain consistency in the dimensionality of the database (see [1, 2]). Therefore, the shapes of the kernels and feature maps are one-dimensional. The frequently used preprocessing method that stacks long input vectors into a rectangular 2D signal is not appropriate in our scenario, since the acceleration data is highly correlated in the time domain. Hence, the outputs of each layer, known as feature maps, are just meaningful along one dimension if 2D CNN is used.

• **Sparse AE** is the unsupervised learning algorithm used to automatically learn hidden features that in many cases work better than hand-crafted features in classification problems.

In the proposed algorithm, a sparse AE is trained using a set of low-dimensional fragments randomly chosen from the database. The sparse AE learns weights which serve for the CNN pretraining later. The motivation and analysis of this step is discussed in the next subsection.

• **The TUM Haptic Texture Database** [2] is composed of acceleration data traces collected for 69 different texture materials (such as marble, brick, wood, and etc.) at 10kHz sampling rate. The visual appearance of selected textures is shown in Fig.1. The acceleration data stored in the database has been normalized and converted from three dimensions to one dimension by using DFT321 [10]. Ten acceleration data traces with a length of 25 seconds have been recorded for each texture. These freehand recordings vary in scan force and scan velocity. As extreme force-velocity pairs will not occur in human user exploratory behavior this is the only constraint of the recorded data traces.

Note that in real surface material exploration scenarios, it is sufficient for a human to hold a tool scratching on an object surface for a few hundred milliseconds rather than to patiently keep scratching for seconds. Moreover, the shorter the data fragments are, the less information the fragments contain, which is to say that it is hard to handle the classification task using only tens of milliseconds long fragments of data. Consequently, one-second-long data fragments are used in this work. This leads to ten thousand dimensions at a 10kHz sampling rate. The ten thousand input dimensions of the proposed ACNN are definitely high and hundred times larger than the dimension of the perceptual features abstracted in [1, 2]. Even though the haptic signal is visually similar to acoustic signals, direct deployment of the mature acoustic signal processing methods on haptic signals leads to unsatisfactory performance [1], owing to the fact that haptic signals are spreading on the frequency domain from zero Hz to thousand Hz and contain more information in the low frequency must than acoustic signals.

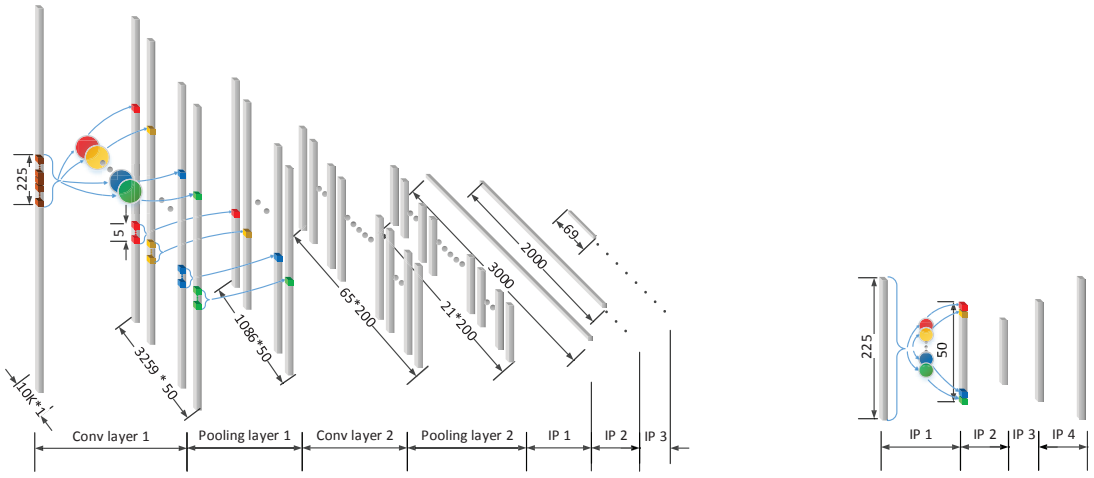
Given the overall ACNN configuration, with the purpose of learning hidden features from RAW acceleration data and then correctly recognizing the surface material category of the acceleration data, the remaining problems are: 1) how the sparse AE's weights initialize the CNN, and how the ACNN is trained; 2) how to deal with the extremely high dimensionality of the input; 3) how to prepare the dataset for training, which will be investigated in the following subsections respectively.

2.2. Details of the ACNN

Fig.2 depicts the structures of the CNN and the AE, where the trained Autoencoder's colored weights are used to partially initialize the CNN's corresponding colored weights. The input and output of each layer are named as feature maps. Both the CNN and the AE are commonly used neural network architectures, where CNNs usually contain Convolution layers, Pooling layers and Inner Product layers, while AEs are merely composed by Inner Product layers [12]. Without loss of generality, let $x_h^{(l)} \in \mathbf{R}^{m_1^{(l)} \times m_2^{(l)}}$ and $y_j^{(l)} \in \mathbf{R}^{n_1^{(l)} \times n_2^{(l)}}$ denote the h-th input and j-th output feature maps of layer l, respectively. In the following, we will elaborate each type of layer and the procedure of the proposed pretraining.

2.2.1. Neural Network Layers

The **convolution layer** convolves input feature maps with filter banks to generate output feature maps. Each filter is used to extract a specific type of local feature from patches of the input feature map [13]. We denote by $k_i^{(l)} \in \mathbf{R}^{s_1^{(l)} \times s_2^{(l)}}$ and



(a) (b)

Fig. 2. (a) Proposed CNN structure. Red, yellow, blue and green colored circles indicate the feature kernels in the first layer of the CNN. There are 50 kernels of size 225. (b) The structure of the Autoencoder. The colored circles represent the weights of the first inner product layer IP1, whose input and output sizes are 225 and 50, respectively. The colored weights of the trained Autoencoder are used to partially initialize the CNN's corresponding colored weights.

$b^{(l)} \in \mathbf{R}$ the i-th convolution kernel and the bias of the layer l respectively. The response of the convolution layer is given by:

$$y_j^{(l)} = \sigma\left(x_h^{(l)} * k_i^{(l)} + b^{(l)}\right), \tag{1}$$

where $*$ denotes the two dimensional convolution operator, $j = (h, i)$ shows that the j-th output feature map is produced based on the h-th input feature map and the i-th convolution kernel in layer l. In addition, there are $H^{(l)}$ input feature maps, $I^{(l)}$ convolution kernels, and $J^{(l)}$ output feature maps in layer l. For the CNN in Fig.2(a), the input feature map for the first convolution layer $x_1^{(1)}$ is a one-second-long fragment of the RAW acceleration data with a sampling rate of 10kHz. So that $x_h^{(1)} \in \mathbf{R}^{10k\times 1}$ and $H^{(1)} = 1$. Because there are $I^{(1)} = 50$ kernels partially colored in Fig.2(a), the number of output feature maps in the first layer is $J^{(1)} = H^{(1)} \times I^{(1)} = 50$. The kernel size of the first convolution layer is 225, i.e., $k_i^{(1)} \in \mathbf{R}^{225\times 1}$. Conventionally, a convolution layer has a stride $d_{conv}^{(l)} = 1$, indicating that the neighboring patches extracted from the input feature map to calculate the convolution is 1-pixel away from each other. In our method, greater-than-1 strides are used, which leads to the size of the first output feature map being 3259 with the stride $d_{conv}^{(1)} = 3$, i.e., $y_j^{(1)} \in \mathbf{R}^{3259\times 1}$. The constants in the 2nd convolution layer are $H^{(2)} = 50$, $I^{(2)} = 4$, $J^{(2)} = H^{(2)} \times I^{(2)} = 200$. The function $\sigma(\cdot)$ represents a nonlinear function, such as Sigmoid function, Hyperbolic Tangent function or Rectified Linear Unit (ReLU) [13]. Considering the computational efficiency, we use ReLU as the nonlinear function.

The **pooling layer** acts usually following a convolution layer to efficiently reduce the resolution of the input feature maps and the sensitivity of the results to distortion and shift. Natural images have the property of stationary, meaning that the whole image almost share the same statistics. Therefore, to describe a large image, one natural approach is to aggregate statistics of those features at various locations. Specifically, the pooling layer aims to extract patches from the input feature maps and to evaluate the corresponding output using the max value (max-pooling) or mean value (average pooling) of the patches. Basically, the patch size is larger than the stride size of the pooling layer, which is why the pooling layer can down-sample feature maps. The high dimensional signals are down-sampled after passing through the pooling layers. It is the pooling layers that make the high dimensional signal processing feasible.

The **inner product layer** refers to a fully connected layer, treats the input as a vector and produces a vector as the output feature map:

$$y^{(l)} = \sigma\left(W^{(l)} \times x^{(l)} + b^{(l)}\right), \tag{2}$$

where $W^{(l)} \in \mathbf{R}^{q^{(l)}\times p^{(l)}}$ is the transform matrix in layer l. $x^{(l)} \in \mathbf{R}^{p^{(l)}}$ and $y^{(l)} \in \mathbf{R}^{q^{(l)}}$ are vectors. As shown in Fig.2(b), $x^{(1)} \in \mathbf{R}^{225}$ and $y^{(1)} \in \mathbf{R}^{50}$. So that $W^{(1)} \in \mathbf{R}^{50\times 225}$.

2.2.2. *CNN Pretraining Using the Weights of the Sparse AE*

In our algorithm, the input vector of the sparse AE, $x^{(1)} \in \mathbf{R}^{p^{(1)}}$, is set to have the same size with $k_i^{(1)} \in \mathbf{R}^{s_1^{(1)}\times 1}$, i.e., let $p^{(1)} = s_1^{(1)}$. The output vector of the sparse AE's first layer, $y^{(1)} \in \mathbf{R}^{q^{(1)}}$, is set to be of size $I^{(1)}$, which is the total number of convolution kernels in the first layer of the CNN, i.e., $q^{(1)} = I^{(1)}$. So that $W^{(1)} \in \mathbf{R}^{I^{(1)}\times s_1^{(1)}}$. After the sparse AE is trained, the kernels of the first layer of the CNN are

initialized based on:

$$\left[\begin{array}{ccccccc} k_1^{(1)} & k_2^{(1)} & \ldots & k_i^{(1)} & \ldots & k_{I^{(1)}}^{(1)} \end{array}\right] = {W^1}^T, \quad (3)$$

where $(\cdot)^T$ denotes transpose.

To fully understand the weights of the sparse AE, we can simply say that the first layer weights $W^{(1)}$ can be treated as the transform matrix of common orthogonal transforms, such as Discrete Fourier Transform (DFT) or Discrete Cosine Transform (DCT). Those transforms have a fixed transform matrix, into which the orthogonal basis vectors are stacked. Some representative basis vectors of the sparse AE and the DCT are respectively illustrated in the lest and right part of Fig.3. While orthogonal transforms are popular in data compression, the sparsity of the transformed vectors is not optimal for all kinds of data. On the contrary, a sparse AE can learn a customized transform matrix (as known as weights of the AE) depending on different types of input data, and guarantees that sparsity and fidelity are competitive enough compared with commonly used transformation matrices.

As we can see in Fig.2(b), because the sparse AE is trying to learn an approximation to the identity function with a compression and a reconstruction procedure, the architecture of sparse AE is symmetric with respect to the middle layer. Having the minimum amount of neurons in the middle layer, the sparse AE will reduce the dimension of the input before reconstructing the input from the compressed feature map, which is the information in the middle layer of the AE. In general, the input information contained in the output will be more or less lost. To assure that layers with a reduced number of hidden neurons contain as much information as the input, the sparse AE learns the weights leading to sparse hidden neurons in the intermediate layers of the AE. The learned weights can be understood as a limited number of the projection directions in which the projection of the input has the largest sparsity and fidelity. Consequently, the trained transform matrix $W^{(1)}$ is nearly orthogonal, i.e.:

$$W^{(1)} \cdot {W^{(1)}}^T - I = \varepsilon, \quad (4)$$

where I is the identity matrix, and the elements in matrix ε approach zero. Example row vectors of $W^{(1)}$ are drawn in the left part of Fig.3, where we can see that, like other row vectors of commonly used transform matrices, like DCT shown in the right part of Fig.3. Both of them extract features from different frequencies as well as multiple patterns and are nearly orthogonal with each other.

Similarly, the convolution kernels of the first convolution layer in the CNN, as shown in Fig.2(a), have the same duty as the weights of a sparse AE. The reason why there is more than one kernel in the convolution layer of the CNN is that one kernel, in analogy to one projection direction of a transform matrix, can only represent information projected on one particular direction. In order to explore more features and possibilities, many uncorrelated kernels are essentially needed.

Inspired by the excellent feature extraction abilities of Sparse AEs and the great classification abilities of CNNs, we take advantage of the benefits of both the AE and CNN in our scheme, which have not been combined in previous works. Specifically, the weights of the trained sparse AE serve for initializing the kernels of the CNN's first convolution layer. Pretrained by the AE's weight, the CNN first layer's weight is almost fixed until the end of the learning process, because the CNN first layer's weight already has exclusive transformation-like patterns trained by sparse AE. In the training phase of the CNN, the first layer shows only tiny changes and the training of the following layers shares hardware resources, so that the ACNN is benefited from less training time. The training time of the sparse AE is negligible compared with that of the CNN due to the much shallow network which needs smaller dataset to train. It turns out that this attempt indeed results in better performance and 10% faster convergence, as demonstrated in our experiments (see Section 3).

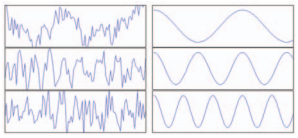


Fig. 3. Illustration of representative 1st layer weights of the sparse AE and the DCT respectively in the left and right part of the figure. Both of them can be treated as one dimensional feature extractors.

2.2.3. Training Part of the ACNN

For the sparse AE, illustrated in Fig.2(b), given a set of acceleration fragments $\{\mathbf{x}^{(1)}\}$, the Mean Squared Error (MSE) together with the KL sparsity penalty are used as the loss function:

$$\begin{aligned} L_{AE} = & \frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|{x^{(1)}}^{(i)} - {y^{(4)}}^{(i)}\right\|^{(2)}\right) \\ & + \frac{\alpha}{2}\sum_{l}^{4} Tr\left(W^{(l)} \cdot {W^{(l)}}^T\right) \\ & + \beta\sum_{l}^{2}\sum_{i}^{q^{(l)}} KL\left(y^{(l)}\{i\}\right), \end{aligned} \quad (5)$$

where the set size m equals $\left|\mathbf{x}^{(1)}\right|$, the ith fragment is represented by $x^{(i)} \in \left\{\mathbf{x}^{(1)}\right\}$, ${y^{(4)}}^{(i)}$ is the output feature map of the last IP layer when the input of the AE is ${x^{(1)}}^{(i)}$, and $KL(\cdot)$ is the Kullback-Leibler (KL) distance [18] commonly used for the sparsity penalty term, which describes how sparse the hidden neurons are. α is the weight for the regularization term, and β controls the weight of the sparcity penalty term.

For the pretrained CNN presented in Fig.2(a), a softmax loss layer is used and computes the multinomial logistic loss of the softmax of inputs. It is conceptually identical to a softmax layer followed by a multinomial logistic loss layer [11].

Both the AE and the CNN are trained using the stochastic gradient descent with 0.9 momentum, 0.01 initial learning

Table 1. Comparison of the classification accuracy on the TUM Haptic Texture Database. The results in *italics* are achieved using [2]. Models with an asterisk* means the sophisticated CNN.

Model \ Input Data Type	RAW data	Spectrogram [7]	other preprocessing [2]
LeNet with 100 fragments / data trace	2%	5.6%	—
LeNet with 1000 fragments / data trace	15%	55.7%	—
Naïve Bayes + spectrogram[15]	—	—	*70%*
Euclidean Distance + MFCC[16]	—	—	*74%*
CNN*	74.9%	72.2%	—
CNN* with data augmentation	77.8%	74.1%	—
CNN* with both the data augmentation and the drop-out layers (CNN**)	79.2%	75.2%	—
CNN** pretrained by AE (**ACNN**), **10%** faster convergence than CNN**	**81.8**%	—	—

rate, 0.0001 gamma and 0.75 power. For tricks about how to set the parameters refer to [13, 17].

3. EXPERIMENTS AND DISCUSSIONS

In this section, the proposed scheme is implemented using the widely used deep learning framework Caffe [11]. The performance is then evaluated by comparing it with method [2], which performs the classification using carefully designed perceptual features, and methods combining machine learning methods and commonly used preprocessing for audio signal processing [15, 16]. All the experiments are conducted on the TUM Haptic Texture Dataset [2] using a computer with Intel Core i7-4770 CUP (3.40GHz*8) and NVIDIA GPU (GeForce GTX 650).

3.1. Training Set Preparation

The training set is prepared by sampling the acceleration data from the Haptic Texture dataset. Specifically, 69 surface materials are collected in the dataset, each of which contains ten data traces, and each trace is 25-second-long free-hand-scratching acceleration data. There are several ways to prepare the training data: treat the whole data trace as a training fragment, or convert the one-dimensional acceleration signal into two dimensional feature maps using preprocessing operations, such as spectrogram [15] or MFCC [16], or sample a series of short fragments from each data trace. Different input types lead to completely different CNN structures.

Considering that in real applications, users may prefer to hold a tool scratching on a surface for around one second before the surface can be classified or retrieved. The CNN training samples we use in the following experiments are overlapping one-second-long RAW acceleration data fragments extracted from each 25-second-long data traces of each type of material in the dataset. Based on the analysis in Subsection 2.2.2, the input data size of the AE is as large as the kernel size of CNN's first convolution layer. So that one hundred 225-dimensional fragments are prepared from each training data trace for learning the AE's weights.

3.2. Results and Analysis

To the best of our knowledge, this is the first work that uses a deep learning architecture to automatically learn haptic texture features from RAW acceleration signals. In our first trial, LeNet is evaluated [14], which is popular for handwritten digit recognition. The observed 2% accuracy is only slightly greater than random guessing when there are hundred raw data fragments drawn out from each data trace, compared with an accuracy of 5.6% when using spectrograms [7] of data fragments as input, as shown in Table 1. In contrast, the accuracy is considerably improved after enlarging the training set using one thousand overlapping one-second-long fragments in each data trace. It declares that even though both 100 and 1000 random fragments can cover the whole data trace with the length of 25 seconds, the augmented 1000 fragments can considerably reduce the sensitivity of the results to shift. However, the LeNet still has limited performance for the enlarged taining set, because the architecture of LeNet is still too shallow to learn more sophisticated features for further improvement.

Afterwards, a more sophisticated CNN (referred as CNN* in Table 1) structure which is based on the CNN structure in [13] was tested. The structure of the proposed CNN structure is presented in Fig.2(a). Compared with the previous shallow network, the classification accuracy of the new network is significantly improved for both types of inputs, RAW data and spectrogram [7]. As we can see, the CNN* with RAW input has not only higher accuracy but also less data preprocessing than the CNN* for which the data fragments are converted into 2D spectrogram. At the same time, the preprocessing-free CNN* performs better than the methods Naïve Bayes + spectrogram[15] and Euclidean Distance + MFCC[16], both of which take task-specific features, such as hardness and roughness, and are implemented in [2]. Despite the CNN*'s excellent result the performance is heavily limited by severe overfitting.

To combat overfitting two primary ways, data augmentation and dropout layer [13], are deployed. The most trivial form of data augmentation is generating slightly shorter translated fragments within a training fragment, and another form

is adding Gaussian noise onto the data fragment. The data augmentation schemes capture a crucial property of the acceleration data, namely, that the identity of texture surfaces is invariant to acceleration data changes in the intensity and the translation. The augmented dataset let the CNN architecture become less sensitive to distortion and translation. Eventually, this scheme reduces the error rate by over 2%. On the other hand, dropout layers let the hidden neurons partially contribute to the forward pass and backward propagation, making sure only partial(50%) neural network is presented for each train fragment. So that a dropout layer can let the network learn more robust features [13]. After adding dropout layers, the CNN*'s accuracy is further improved over 1%, as shown in Table 1. Despite the acceptable performance, it takes more than 10 hours to train the CNN* with both the data augmentation and the drop-out layers (referred as CNN** in following).

The proposed ACNN is analysed in Subsection 2.2.2. Basically, pretraining the CNN** using a sparse AE's weights based on the methods stated in Section 2 achieves 10% faster convergence than CNN** which is randomly initialized. At the same time, the proposed pretraining scheme ACNN results in 2.6% higher accuracy than CNN** and the experiment possesses the most superior performance regarding the classification rate being **81.8%**, summarized in Table 1.

For other implementation details, the input dimension of our method is ten thousand in contrast to the dimensionality of those methods based on hand-crafted features which are less than hundreds. Due to limited GPU memory, the convolution layer with a stride of 1 is not feasible for a deep CNN. Consequently, convolution layers with greater-than-1 strides are chosen to down-sample the output feature map and to save memory for the more sophisticated network.

4. CONCLUSION AND FUTURE WORK

A preprocessing-free ACNN structure is investigated in this paper, which is pretrained by a sparse Autoencoder's weights. The ACNN is easy to implement and reproduce compared with the previous methods which usually require carefully designed feature engineering for specific type of data. Experiment results on the TUM Haptic Texture Database demonstrate the effectiveness of our ACNN structure, which can automatically learn even better feature representations than the labor-intensively hand-engineered ones. Our future work may include the exploitation of computational redundancy of fragment-by-fragment scanning, better CNN structure and training method to improve the result.

5. REFERENCES

[1] M. Strese, J.-Y. Lee, C. Schuwerk, Q. Han, H.-G. Kim and E. Steinbach, "A haptic texture database for tool-mediated texture recognition and classification," in *Proc. of IEEE HAVE*, Dallas, Texas, USA, October 2014.

[2] M. Strese, C. Schuwerk and E. Steinbach, "Surface Classification Using Acceleration Signals Recorded During Human Free-hand Movement," in *Proc. of IEEE World Haptics Conference*, Chicago, USA, June 2015.

[3] J. A. Fishel and G. E. Loeb, "Bayesian exploration for intelligent identification of textures," *Frontiers in neurorobotics*, vol. 6, no. 4, pp. 1-20, June 2012.

[4] J. M. Romano and K. J. Kuchenbecker, "Methods for Robotic Tool-Mediated Haptic Surface Recognition," in *IEEE Haptics Symposium (HAPTICS)*, Houston, Texas, USA, February 2014.

[5] J. M. Romano and K. J. Kuchenbecker, "Creating realistic virtual textures from contact acceleration data," *IEEE Transactions on Haptics*, vol. 5, no. 2, pp. 109-119, April 2012.

[6] L. Deng, M. Seltzer, D. Yu, A. Acero, A. Mohamed and G. Hinton, "Binary Coding of Speech Spectrograms Using a Deep Auto-Encoder," in *Proc. Interspeech*, Makuhari, Chiba, Japan, September 2010.

[7] H. Lee, Y. Largman, P. Pham and A. Ng, "Unsupervised feature learning for audio classification using convolutional deep belief networks," *Advances in Neural Information Processing Systems (NIPS)*, pp. 1096-1104, 2009.

[8] M. Matsugu, K. Mori, Y. Mitari and Y. Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," *Neural Networks*, vol. 16, no. 5, pp.555-559, 2003.

[9] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-Based Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, Nov. 1998.

[10] N. Landin, J. M. Romano, W. Mcmahan and K. J. Kuchenbecker, "Dimensional Reduction of High-Frequency Accelerations for Haptic Rendering," *Haptics: Generating and Perceiving Tangible Sensations*, Springer Berlin Heidelberg, pp. 79-86, 2010.

[11] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick and et. al., "Caffe: Convolutional architecture for fast feature embedding," *Proceedings of the ACM International Conference on Multimedia*, ACM, pp. 675-678, 2014.

[12] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp.504-507, 2006.

[13] A. Krizhevsky, I. Sutskever and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems (NIPS)*, pp. 1097-1105, 2012.

[14] Y. LeCun, L. Botou, L. Jackel, H. Drucker and et. al., "Learning Algorithms for Classification: A Comparison on Handwritten Digit Recognition," *Neural Networks*, pp. 261-276, 1995.

[15] J. Sinapov and V. Sukhoy, "Vibrotactile recognition and categorization of surfaces by a humanoid robot," *IEEE Transactions on Robotics*, vol. 27, no. 3, pp. 488-497, June 2011.

[16] D. Ellis. (2015, May 10). *Reproducing the feature outputs of common programs using matlab and melfcc.m* [Online]. Available: http://labrosa.ee.columbia.edu/matlab/rastamat/mfccs.html

[17] L. Bottou, "Stochastic Gradient Descent Tricks," *Neural Networks: Tricks of the Trade*, Springer Berlin Heidelberg, pp. 421-436, 2012.

[18] S. Kullback, "Letter to the Editor: The Kullback-Leibler Distance," *The American Statistician*, vol. 41, no. 4, pp. 340-341, 1987.