Convolutional Neural Networks with Data Augmentation for Classifying Speakers' Native Language

Gil Keren[1], *Jun Deng*[1], *Jouni Pohjalainen*[1], *Björn Schuller*[1,2]

[1]Chair of Complex & Intelligent Systems, University of Passau, Germany
[2]Department of Computing, Imperial College London, UK

gil.keren@uni-passau.de

Abstract

We use a feedforward Convolutional Neural Network to classify speakers' native language for the INTERSPEECH 2016 Computational Paralinguistic Challenge Native Language Sub-Challenge, using no specialized features for computational paralinguistics tasks, but only MFCCs with their first and second order deltas. In addition, we augment the training data by replacing the original examples with shorter overlapping samples extracted from them, thus multiplying the number of training examples by almost 40. With the augmented training dataset and enhancements to neural network models such as Batch Normalization, Dropout, and Maxout activation function, we managed to improve upon the challenge baseline by a large margin, both for the development and the test set.

Index Terms: Computational Paralinguistics, Deep Learning, Convolutional Neural Networks.

1. Introduction

In recent years, neural networks have become empirically successful in a wide range of supervised learning applications, such as computer vision [1, 2], speech recognition [3, 4] and natural language processing [5]. In addition, in many applications, the traditionally used models that are based on hand-crafted features to represent the data, have been replaced by neural networks models with convolutional layers [6], that are learned directly from raw data or from simpler general-purpose features in an end-to-end manner. In these models, the convolutional layers are typically the lower layers in the network (closer to the input) and can be seen as extracting features from small patches of data, to create a new and possibly more useful representation of the data. Prominent examples of the end-to-end approach can be seen in [1], in which an object recognition model is learned from raw images without any prior knowledge, and [4], where a state-of-the-art speech recognition model is learned directly from speech spectrogram.

In many cases in the field of computational paralinguistics, including the ComParE Challenge 2016 [7], the pipeline of a machine learning model begins by extracting features to represent the data at hand: first, low-level descriptors (LLDs) are extracted from short time windows and include short term characteristics of the audio signal such as voicing probability, HNR, F0 and zero-crossing rate [8]. This first step is possibly followed by a number of functionals (e.g., mean, max, percentiles), computed over time on these LLDs. Recent studies have shown ([9, 10]), that the end-to-end approach can be adopted to applications in computational paralinguistics, by training a convolutional neural network model based on simpler and non-specialized features.

A necessary condition for the success of neural network models for classification, is in many cases the presence of large amounts of labeled training data [11]. In many classification datasets in the field of computational paralinguistics, the number of labeled training examples is at most a few thousands [12, 13, 14], which is rather small when compared to object detection or speech recognition datasets that typically contain at least a few tens of thousands of labeled examples, and in many cases much more than that [15, 16, 4]. On the other hand, in audio datasets, this problem might be alleviated if every labeled example is long enough.

In an audio classification task, there might be some redundancy in long examples, when smaller parts of the whole example contain enough information each for correctly classifying the whole example. This is indeed the case, when very long time dependencies (relations between two distant time frames) in the data are not needed for correctly classifying the whole example. In that situation, we can safely introduce data augmentation of the training set by replacing each labeled example with many small patches from it, retaining the same class label. This approach is different than other data augmentation approaches, such as changing the speed of an audio signal ([17]).

In this year's ComParE challenge, the *Native Language* dataset is comprised of 5,132 labeled examples, where the length of each example is approximately 45 seconds. The task in the nativeness subchallenge is to classify the mother tongue of the speaker in each example. We assume that less than the whole 45 seconds are required for correctly classifying an example, therefore posing the possibility for the necessary data augmentation that will allow successfully training a large neural network model with convolutional layers, using relatively simple and general-purpose input features.

As a step towards the end-to-end approach, we extract only MFCC features with first and second order delta regression coefficients, and we train feedforward neural network models with convolutional layers to classify the data for the nativeness subchallenge of the ComParE 2016 competition. In addition, we introduce data augmentation as described above, and we use two well-known enhancements for the neural network models, namely *Dropout* [18] and *Batch Normalization* [19]. For comparison, we train a few additional neural networks and Support Vector Machine (SVM) classifiers, that vary in the input features used and training data augmentation.

Therefore, the purpose of this work is two-fold: First, to explore the possibility that general-purpose features of relatively lower complexity might yield good classification results when coupled with a neural network model, compared to models using the provided challenge features. Second, to explore the benefits of the training data augmentation resulting from training

models on smaller patches from the original training examples.

In the rest of the paper, we present the models we use and the specifications of the training data augmentation we apply (Section 2), our experiments and results (Section 3), and conclude our work in Section 4.

2. Model and Data Augmentation

For classifying audio signals with a neural network, we use a feedforward neural network comprised of a few convolutional hidden layers followed by a few fully connected hidden layers (dense layers), with a softmax layer [20] on top for classification. For this task, a more natural choice of neural networks might be Recurrent Neural Networks (RNNs), which are neural networks that process the input sequentially and are able to model dependencies over time. However, sequential processing of even a few tens of time steps can introduce a significant computational overhead, as the network then correspondes to a very deep neural network, with many more steps of computation. This becomes even more significant, when using variants of RNNs that require more computation when processing each time step, such as Long Short Term Memory networks (LSTMs) [21].

We elaborate on the key components of our feedforward neural networks.

2.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are neural networks that contain one or more convolutional layers. A convolutional layer processes a two-dimensional input of size $m \times n$ with k channels $x \in \mathbb{R}^{m \times n \times k}$ in the following manner: The input x is split spatially across its first two dimensions into (potentially overlapping) patches of the same size $\{x_{ij}\}_{i \in I, j \in J}$, where $x_{ij} \in \mathbb{R}^{m_1 \times n_1 \times k}$. Then the n-dimensional output vector of a patch x_{ij} is:

$$c_{ij} = \sigma(x_{ij} \odot W_1 + b_1, \ldots, x_{ij} \odot W_n + b_n) \qquad (1)$$

Where $W_1, \ldots, W_n$ are weight matrices of size $m_1 \times n_1 \times k$, the bias terms are $b_1, \ldots, b_n \in \mathbb{R}$, $\odot$ is an element-wise matrix multiplication and σ is a non-linear activation function operating element-wise. The output of the convolutional layer is the spatial map $c = \{c_{ij}\}_{i \in I, j \in J}$ of dimension $|I| \times |J| \times n$. The number n is called the number of *feature maps* in the convolutional layer, and c can be viewed as a spatial two dimensional map with n channels in each spatial location.

Potentially, a convolutional layer can include a max-pooling mechanism after the above described process, that operated as follows: The output of the above procedure c is again split spatially across its first two dimensions into (potentially overlapping) patches $\{p_{ij}\}_{i \in I', j \in J'}$. Then, a *max* operation is performed across the first two dimensions of a given patch, for each patch separately, resulting in an n-dimensional representation for each path. The output of the max-pooling mechanism (and the whole convolutional layer) p is the spatial map of all the n-dimensional representations, $p \in \mathbb{R}^{|I'| \times |J'| \times n}$.

Other types of pooling exist, such as mean-pooling [22], but models in this work use only the max-pooling mechanism.

In the case of an audio signal, the dimension of x can be seen as $1 \times t \times k$, where t is the number of time steps in the audio signal and k in the number of features used to represent each time step. In this case, Figure 1 depicts the procedure described above (without pooling).

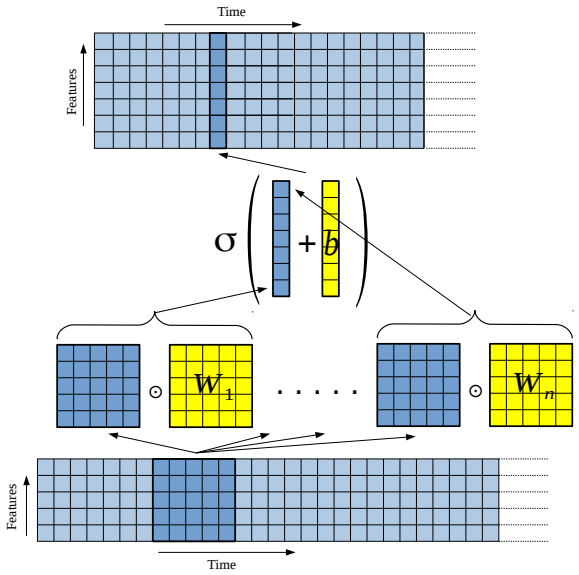

Figure 1: *A convolutional layer operating on an audio signal (without pooling)*

2.2. Batch Normalization

When training neural networks, the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. The authors of [19] name this phenomenon *internal covariance shift*, and state that it slows down training by requiring lower learning rates and careful parameter initialization. To alleviate this issue, they introduce *Batch Normalization*, which applies a linear transformation on the output of a layer (just before applying the activation function), to enforce values for its mean and standard deviation. The mean and standard deviation are calculated per mini-batch of examples, and for each element of the output vector separately, in the case of a dense layer, or for each feature map separately and across all spatial locations, in the case of a convolutional layer. The target mean and standard deviation are again decided for each element of the output vector / feature map separately in the case of a dense layer / convolutional layer, and are additional parameters of the model that are learned during training. At inference time, mini-batch mean and standard deviation are replaced by their equivalents calculated on the whole training set.

All neural network models in this work use Batch Normalization for all hidden layers, not including the the softmax layer.

2.3. Dropout

Dropout [18] is a technique for preventing overfitting in neural network models. When dropout is applied on some layer of a neural network, each element in the output (possibly multidimensional output, as in convolutional layers) is set to zero with probability p. This process is being done for each training example separately. At inference time, dropout is not used, therefore on average test examples might induce bigger outputs in absolute values, compared to training examples. To compensate for this effect, at inference time the output of the layer with dropout is multiplied by $\frac{1}{1-p}$. By using dropout, we might prevent different units in the neural network from co-adapting too much.

Table 1: *Architectures of the three main settings using neural networks.*

Data Augmentation	Features	Convolutional Layers	Dense Layers	Batch Normalization	Dropout
Yes	MFCC + deltas + delta-deltas	3	2	Yes	0.5
Yes	ComParE 2013	0	1	Yes	0.5
No	ComParE 2013	0	4	Yes	0.5

2.4. Data Augmentation

As was mentioned before, in some of our models we augment the training dataset by replacing each training example x with a few samples $x_1, \ldots, x_r$ from it. In this work, each training example is an audio signal, and each sample is a continuous patch of the original audio signal. It is important to note, that two samples, that are almost completely overlapping, are still very different in values when compared element-wise for each element in the input.

For classification at inference time, we need to predict one class label for each example x in a test/development set. Note that in the case of a neural network classifier with a softmax layer, for each sample x_i the network outputs the selected class label y_i, as well as a probability p_{ij} for each class j. We examine three different ways to combine the class predictions / probability distributions from the samples $x_1, \ldots, x_r$ into one class prediction y for the example x:

- Using the prediction of the sample the classifier is most sure about: $y = argmax_j \, max(\{p_{ij}\}_{1 \leq i \leq r})$
- Averaging all predictions: $y = argmax_j \, \frac{1}{r} \sum_{i=1}^{r} p_{ij}$
- Majority vote: y is the the most common value in $(y_1, \ldots, y_r)$. In case two classes labels are equally common, the one with the lower index is chosen.

Except for enlarging the training dataset, replacing examples with shorter samples extracted from them has two additional properties, that are crucial in order to use a feedforward neural network models on this data. First, using fixed size samples from the training data forces all inputs to be of the same size, allowing us to use feedforward neural networks for this problem. Second, since the original audio clips from the Native Language dataset are approximately 45 seconds long, extracting LLDs from short time windows can result in an infeasible size of input, and using shorter samples may alleviate this problem.

3. Experiments

For the main model used in this paper, we augment the training set by replacing each example with 10 seconds samples from it, with the starting point of samples shifted by one second. For example, for a 45 seconds audio signal, 36 samples of 10 seconds will replace the original audio signal in the training set. The augmented training set contains 122,980 examples, compared to 3,300 in the original set. For each example in the augmented dataset, we extract 12 MFCC features and the logarithmic energy for windows of 25 ms shifted by 10 ms. In addition, we extract first and second order delta regression coefficients, to result in a total 39 features to represent each 25 ms time window. We apply mean and standard deviation normalization for each feature separately.

We experiment with different network architectures, learning rate, momentum, dropout rate and training algorithm, and we report the setting that was used in our best performing model on the development set. The network contains three convolutional layers with 350 feature maps each, with a window size of five time steps, shifted by two time steps. In the first two layers, max-pooling is used, to pool over non-overlapping groups of two time steps. After each convolutional layer, a maxout activation function [23] is applied, grouping sets of two feature maps with the *max* operator, to result in 175 output feature maps for each convolutional layer. The output of the third convolutional layer is flattened to be one dimensional, and is fed into two consecutive dense layers with 1000 hidden units each, and a maxout activation function applied over groups of two units. The output of the second dense layer is fed into a softmax layer to output a probability distribution over the possible classes.

Batch Normalization is applied for each of the hidden layers (convolutional and dense) just before applying the activation function. Dropout is as well applied for all hidden layers, with dropout probability of 0.5. Initial values for all weights in the network were sampled from a Gaussian distribution with a standard deviation of 0.1. Optimization is performed using stochastic gradient descent with momentum [24], using a learning rate of 0.1, momentum value of 0.9 and a mini-batch size of 128 examples. Gradient-Clipping [25] is applied as well, to limit the L^2 norm of the gradient to 100.0. The selected model is the one performing best on the development set, and training is stopped after 25 training epochs with no improvement in Unweighted Average Recall (UAR) on the development set, using all three ways to combine predictions on samples to predictions on full-length examples, described in Section 2.4. The experiments were performed using the deep learning framework Blocks [26] based on Theano [27, 28].

Table 2: *Unweighted Average Recall on the development set for different values of the complexity parameter C, for an SVM classifier with a linear kernel.*

C	UAR [%]
5	50.40
2	50.60
1	50.61
10^{-1}	50.33
10^{-2}	52.18
10^{-3}	53.64
10^{-4}	55.11
10^{-5}	53.66

For comparison, two additional settings using neural networks are evaluated. First, a setting where we use the same data augmentation technique as the main model, but use the challenge's provided feature set (the ComParE 2013 feature set, with 6373 features), which includes functionals computed over LLDs. Second, a setting in which we use challenge's provided feature set and do not use data augmentation. We extract the ComParE 2013 features from each 10 seconds example, using openSMILE [29]. Note, that these features are not ordered by

Table 3: *Unweighted Average Recall on the development and test sets for best models of each of the four experiment settings and the challenge baseline. When data augmentations was used, we also report the UAR of samples from the development set, before combining the predictions on samples to predictions on full-length examples.*

Classifier	Features	Data Augment'	Dev' UAR [%]	Best Predictor	Samples UAR [%]	Test UAR [%]
SVM (baseline)	ComParE 2013	No	45.10	—	—	47.50
SVM	ComParE 2013	Yes	55.11	Majority Voting	41.94	—
NN	ComParE 2013	No	50.03	—	—	—
NN	ComParE 2013	Yes	55.78	Majority Voting	42.98	—
NN	MFCC + d + dd	Yes	59.67	Mean	50.99	55.95
NN (ensemble)	MFCC + d + dd	Yes	61.47	—	—	58.33

time or any other order, therefore it does not make sense to use convolutional layers in this case. Since all settings are different one from the other by the input features or the size of the training data, we had to configure each setting separately. We found that the same learning rate, momentum and dropout values, as well as the number of hidden units in the dense layers and the choice of activation function that yield best results on the development set, are the same in our main setting and in the two additional settings described above, leaving only the number of hidden layers to vary. An overview of the architectures used for the three settings we discussed so far appears in Table 1.

In the last additional setting we use for comparison, we still use the same training data augmentation method and the ComParE 2013 features, but we use a Support Vector Machine (SVM) with a linear kernel as the classifier, instead of a neural network. Since an SVM in its basic version does not output a probability distribution over the possible classes, at inference time we use only majority voting in order to combine the predictions on the 10 seconds samples to one prediction on the longer example from the development / test set. We use the implementation of an SVM from the Python package scikit-learn [30]. The complexity parameter C is optimized using the development set. The Unweighted Average Recall on the development set for the different values of C is found in Table 2.

Table 3 contains the UAR for the development and test sets for each of the total four experiment setting (three with Neural Networks, one with an SVM) and the challenge baseline. From the results in the table, it is evident that the best performing model on the development set was from the setting of a convolutional neural network with MFCC + deltas + delta-deltas input features, yielding a UAR of 59.67%, outperforming the challenge baseline (45.10%) by a large margin. A great improvement over the challenge baseline was observed also for the test set results, where our best model yielded a UAR of 55.95%, compared to 47.50% of the challenge baseline model. In particular, our best setting yielded superior results over a similar setting that differs only by the input features used. These results are positive evidence that simpler and more general-purpose features, paired with a convolutional neural network and a large enough dataset, can outperform a standard approach relying on a large number of specialized features, as we hypothesized in the introduction.

In addition, when data augmentation was used, Table 3 also reports the UAR on the samples themselves from the development set, and the best predictor, that is, the best method to combine the predictions on samples to one prediction on a full-length example. The results in the table allow us to evaluate the contribution of the training data augmentation, which seems improve the results for both SVM and neural network classifiers, using the ComParE features (a setting with a neural network classifier, no data augmentation and MFCC + deltas + delta-deltas input features was not evaluated for comparison, due to infeasible input size). For the SVM classifiers, combing the predictions on samples, that were classified with a UAR of only 41.94%, was enough to yield a UAR of 55.11% for full-length examples. A similar phenomenon regarding the gap between the UAR on samples and full-length examples is present in all experiment settings that include data augmentation.

To further improve our results, we evaluated the majority voting of an ensemble, comprised of our best model together with six other models from the same setting, that differ from the best model by the number of feature maps in the convolutional layers, dropout probability, learning rate and momentum value. The ensemble of models yielded a UAR of 61.47% on the development set, and 58.33% on the test set.

4. Conclusion

We compared models from a few settings of neural networks and Support Vector Machines (SVM), differing in data augmentation and input features used, for the classification task of the INTERSPEECH 2016 Computational Paralinguistics Native Language subchallenge. For training data augmentation, we replaced the original examples with shorter overlapping samples extracted from them, multiplying the number of training examples by more than 37. We found that the best performing model on the development set is a Convolutional Neural Network model, using the augmented data set and trained in an end-to-end manner from MFCCs and their first and second order delta regression coefficients as input features. In particular, our best model obtained an Unweighted Average Recall (UAR) of 55.95% on the challenge test set (and 58.33% with an ensemble of similar models), improving over the challenge baseline of 47.50% by a large margin. The results are a positive evidence that specialized features for tasks in computation paralinguistics can be replaced by general-purpose and simpler features like MFCCs. In addition, we evaluated the isolated effect of the training data augmentation, that yielded a 10.01% improvement in UAR using an SVM classifier with the provided challenge features, and a 5.75% improvement in UAR using a neural network model with the same features.

5. Acknowledgements

This work has been partially supported by the European Community's Seventh Framework Programme through the ERC Starting Grant No. 338164 (iHEARu) and the BMBF IKT2020-Grant under grant agreement No. 16SV7213 (EmotAsS). We further thank the NVIDIA Corporation for their support of this research by Tesla K40-type GPU donation.

6. References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proc. of Advances in neural information processing systems (NIPS)*, Lake Tahoe, NV, 2012, pp. 1097–1105.

[2] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proc. of The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Boston, MA, 2015, pp. 1–9.

[3] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[4] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. C. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: end-to-end speech recognition in English and Mandarin," *arXiv preprint arXiv:1512.02595*, 2015.

[5] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proc. of Advances in neural information processing systems (NIPS)*, Montreal, Canada, 2014, pp. 3104–3112.

[6] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[7] B. Schuller, S. Steidl, A. Batliner, J. Hirschberg, J. K. Burgoon, A. Baird, A. Elkins, Y. Zhang, E. Coutinho, and K. Evanini, "The INTERSPEECH 2016 Computational Paralinguistics Challenge: Deception & Sincerity," in *Proc. of INTERSPEECH 2016, 17th Annual Conference of the International Speech Communication Association*, San Francsico, CA, 2016, to appear.

[8] B. Schuller, S. Steidl, A. Batliner, A. Vinciarelli, K. Scherer, F. Ringeval, M. Chetouani, F. Weninger, F. Eyben, E. Marchi, M. Mortillaro, H. Salamin, A. Polychroniou, F. Valente, and S. Kim, "The INTERSPEECH 2013 Computational Paralinguistics Challenge: Social Signals, Conflict, Emotion, Autism," in *Proc. of INTERSPEECH 2013, 14th Annual Conference of the International Speech Communication Association*, Lyon, France, 2013, pp. 148–152.

[9] G. Keren and B. Schuller, "Convolutional RNN: an enhanced model for extracting features from sequential data," in *Proc. of 2016 International Joint Conference on Neural Networks (IJCNN) as part of the IEEE World Congress on Computational Intelligence (IEEE WCCI)*, Vancouver, Canada, 2016, to appear.

[10] G. Trigeorgis, F. Ringeval, R. Brückner, E. Marchi, M. Nicolaou, B. Schuller, and S. Zafeiriou, "Adieu features? End-to-end speech emotion recognition using a deep convolutional recurrent network," in *Proc. of the 41st IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Shanghai, P. R. China, 2016, to appear.

[11] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[12] B. Schuller, S. Steidl, A. Batliner, E. Nöth, A. Vinciarelli, F. Burkhardt, R. van Son, F. Weninger, F. Eyben, T. Bocklet, G. Mohammadi, and B. Weiss, "The INTERSPEECH 2012 Speaker Trait Challenge," in *Proc of. INTERSPEECH 2012, 13th Annual Conference of the International Speech Communication Association*, Portland, OR, 2012, pp. 254–257.

[13] B. Schuller, S. Steidl, A. Batliner, J. Epps, F. Eyben, F. Ringeval, E. Marchi, and Y. Zhang, "The INTERSPEECH 2014 Computational Paralinguistics Challenge: Cognitive & Physical Load," in *Proc. of INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association*, Singapore, Singapore, 2014, pp. 427–431.

[14] B. Schuller, S. Steidl, A. Batliner, S. Hantke, F. Hönig, J. R. Orozco-Arroyave, E. Nöth, Y. Zhang, and F. Weninger, "The INTERSPEECH 2015 Computational Paralinguistics Challenge: Degree of Nativeness, Parkinson's & Eating Condition," in *Proc. of INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association*, Dresden, Germany, 2015, pp. 478–482.

[15] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits," 1998.

[16] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[17] T. Ko, V. Peddinti, D. Povey, and S. Khudanpur, "Audio augmentation for speech recognition," in *Proc. of INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association*, Dresden, Germany, 2015, pp. 3586–3589.

[18] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[19] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proc. of of the 32nd International Conference on Machine Learning (ICML)*, Lille, France, 2015, pp. 448–456.

[20] J. S. Bridle, "Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition," in *Neurocomputing*. Springer, 1990, pp. 227–236.

[21] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[22] 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, 1998.

[23] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio, "Maxout networks," in *Proc. of the 30th International Conference on Machine Learning (ICML)*, Atlanta, GA, 2013, pp. 1319–1327.

[24] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proc. of the 30th international conference on machine learning (ICML)*, Atlanta, GA, 2013, pp. 1139–1147.

[25] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," in *Proc. of the 30th International Conference on Machine Learning (ICML)*, Atlanta, GA, 2013, pp. 1310–1318.

[26] B. van Merriënboer, D. Bahdanau, V. Dumoulin, D. Serdyuk, D. Warde-Farley, J. Chorowski, and Y. Bengio, "Blocks and fuel: Frameworks for deep learning," *arXiv preprint arXiv:1506.00619*, 2015.

[27] F. Bastien, P. Lamblin, R. Pascanu, J. Bergstra, I. J. Goodfellow, A. Bergeron, N. Bouchard, and Y. Bengio, "Theano: new features and speed improvements," Deep Learning and Unsupervised Feature Learning Workshop, Advances in neural information processing systems (NIPS), 2012.

[28] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proc. of the Python for Scientific Computing Conference (SciPy)*, vol. 4, 2010, p. 3.

[29] F. Eyben, F. Weninger, F. Groß, and B. Schuller, "Recent developments in openSMILE, the Munich open-source multimedia feature extractor," in *Proc. of the 21st ACM International Conference on Multimedia*, Barcelona, Spain, 2013, pp. 835–838.

[30] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.