Recurrent Support Vector Machines for Audio-Based Multimedia Event Detection

Yun Wang, Florian Metze
Language Technologies Institute
Carnegie Mellon University
Pittsburgh, PA 15213, USA
{yunwang, fmetze}@cs.cmu.edu

ABSTRACT

Multimedia event detection (MED) is the task of detecting given events (*e.g.* parade, birthday party) in a large collection of video clips. While the most useful information comes from visual features and speech recognition, a lot can also be inferred from the non-speech audio content, either alone or in conjunction with visual and speech cues. This paper studies MED with non-speech audio information only.

MED is usually performed in two stages. The first stage generates a representation for each clip in the form of either a single vector or a sequence of vectors, often by aggregating frame-level features; the second stage performs binary or multi-class classification to decide whether each target event occurs in each clip. Common classifiers used for the second stage include support vector machines (SVMs), feed-forward deep neural networks (DNNs), and recurrent neural networks (RNNs).

In this paper, we propose to classify clips for events using "recurrent SVMs". These models combine the kernel mapping and the large-margin optimization criterion of SVMs, and the ability to process sequences of variable lengths of RNNs. Reinforced with data augmentation, recurrent SVMs have achieved higher mean average precision (MAP) on the TRECVID 2011 MED task than both SVMs and RNNs.

Categories and Subject Descriptors

H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing;
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval;
I.5.1 [**Pattern Recognition**]: Models

General Terms

Algorithms, Experimentation, Performance

Keywords

Multimedia event detection (MED), noisemes, support vector machines (SVMs), recurrent neural networks (RNNs), kernel mapping, large margin, hinge loss, data augmentation

1. INTRODUCTION

Multimedia event detection (MED) is the task of detecting given events happening in a video clip, and can be used to index video clips in large collections such as YouTube for efficient retrieval. While it is obvious that the visual and speech information in the video clips is useful for event detection, non-speech audio information can also be important: some non-speech sounds may directly reveal the event happening, such as the distinct clatter of a bowling ball hitting the pins; other sounds indicate the environment in which events happen, such as the buzz of machines in a factory. Some of this audio information may not even be present in the visual content. As a result, a number of systems have been designed to perform event detection with non-speech audio information only (see below).

MED systems usually consist of two stages. In the first stage, a clip-level representation is generated for each video clip. Such representations usually come in the form of a vector or a sequence of vectors, and are often an aggregation of frame-level features. In the second stage, binary or multi-class classifiers are built for the target events, taking the clip-level representations as inputs.

The methods to generate clip-level representations may be classified into three types. The first type, inspired by speaker identification, models each video clip as a GMM supervector [1, 2] or an "i-vector" [3, 4, 5]. The second type, inspired by topic classification, first learns a "vocabulary" of frame-level acoustic features by vector quantization, and then represents each video clip with a bag-of-audio-words (BoAW) vector [6]. In both approaches above, the frame-level features are treated as mutually independent, and their order is ignored altogether. As a remedy to this loss of information, semantically meaningful short audio segments have been proposed as the units of the third type of clip-level representation. These units may be either learnt in an unsupervised fashion [7, 8] or defined by humans [9]. The semantic units can be detected with either hidden Markov models (HMMs) or recurrent neural networks (RNNs) [10, 11, 12, 13, 14]. In this paper, we adopt sequences of "noiseme confidence vectors" [15] generated with a bidirectional RNN [16] as the representation of video clips.

The clip-level representations may be classified in various ways. BoAW vectors can be modeled with latent Dirichlet allocation (LDA) [17], treating each event as a topic; alternatively, all types of clip-level representations can be classified with general-purpose classifiers such as support vector machines (SVMs) [2, 18], random forests [15], feed-forward DNNs [13, 14], and recurrent neural networks [16].

In [16], the authors report a mean average precision (MAP) of 4.6% on the TRECVID 2011 MED task [19]. This number is rather low considering the usually amazing ability of RNNs to process sequential data, and actually falls behind an SVM implemented as in [18] on the same clip-level representation (7.1%, see Sec. 3). In this paper, we propose "recurrent SVMs" to combine the merits of SVMs and RNNs. We start by grafting elements of SVMs on to a feed-forward neural network: first, we implement the implicit kernel mapping of SVMs explicitly as a frontend of the network; then, we replace the cross-entropy loss function of the neural network with a squared hinge loss as is common in SVMs. At this step, the model is essentially an SVM with a non-linear hidden layer. The recurrent SVM is finally constructed by making the hidden layer recurrent, and it achieves a higher MAP than both SVMs and RNNs.

2. THE TRECVID 2011 MULTIMEDIA EVENT DETECTION TASK

In this section, we introduce the data and the evaluation criterion of the TRECVID 2011 MED task [19], and the representation of video clips used in this paper.

The TRECVID 2011 MED corpus has been used in a number of papers [2, 17, 15, 20, 21]. It consists of 3,104 training video clips and 6,642 test video clips. These video clips are "user-generated content posted to Internet video hosting sites", and exhibit a diversity of quality and duration. The median duration of the clips is 82 seconds; the mode is around 30 seconds. Some of the clips are labeled with one out of 15 events[1]; others are background clips. The goal is to train one or multiple models that would produce a confidence score given a test clip and an event. The scores can be used to make binary predictions about whether an event occurs in a test clip, or to form a ranked list of the test clips for each event.

We adopt the mean average precision (MAP) as the evaluation metric, as is done in [15] and [20]. For each event, the *average precision* measures the quality of the ranked list produced by a model. Sort the test clips by their confidence scores in descending order, and suppose the i-th positive instance is ranked at the r_i-th position (we break ties by ranking negative instances higher). The average precision is defined as $\text{AP} = \frac{1}{n}\sum_{i=1}^{n} i/r_i$, where n is the total number of positive test instances. The mean of the AP across all events is called the *mean average precision* (MAP).

The representation of video clips used in this paper, as well as in [16], is sequences of "noiseme confidence vectors" (NCVs), proposed in [15]. A *noiseme* is an audio segment that has an interpretable and descriptive meaning, such as "music", "cheer", and "engine". 42 types of noisemes are defined in [9] (the corpus actually contains 48 distinct noiseme labels); we manually merged some rare noisemes into semantically close ones, ending up with 17 noisemes (plus a "background" class). We form the NCV sequence with the following steps:

1. Extract low-level acoustic features (*e.g.* MFCC, F_0) with the OpenSMILE toolkit [22], and compute various statistics of these features (*e.g.* mean, variance) using sliding windows of 2 s moving 100 ms at a time;
2. From the 6,669 dimensions of features produced above, select 983 dimensions using the information gain criterion, as done in [15];
3. Predict the probability of each noiseme occurring at each frame using a bidirectional recurrent neural network (BRNN) trained in [16], resulting in a 18-dimensional NCV sequence.

[1]See [16] for a complete list of the 15 events; note that the names of the events E004 and E005 were reversed by mistake in that paper.

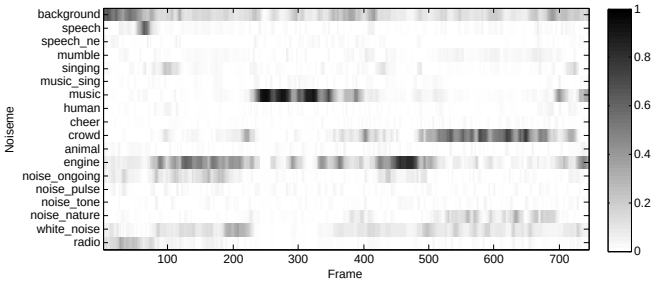

Figure 1: *An example noiseme confidence vector sequence.*

Fig. 1 shows an example of a clip represented as an NCV sequence.

For the non-recurrent SVM model and the feed-forward neural networks discussed in Section 3, the input to the model is the average of the NCVs across all the frames. For the recurrent SVM models discussed in Section 4, in theory the NCV sequences could be used directly as the input. However, the sequences are often very long. In order to save computation time, we first shorten all sequences whose length l is greater than 50 to $m = \sqrt{50l}$ frames, by dividing the original sequence evenly into m segments and taking the average within each segment. The resultant sequences, with a median length of 203 frames, are still too long to train a recurrent model reliably. So we further shorten each sequence of length m to m^β frames, with $\beta \in \{0.3, 0.5, 0.6, 0.7\}$.

3. MIMICKING AN SVM WITH A FEED-FORWARD NEURAL NETWORK

In [18], support vector machines (SVMs) were employed for event detection. One separate SVM was built for each event, using the χ^2-RBF kernel and a hinge loss function. By applying these SVM models to the NCV representation introduced in Sec. 2, we obtained a MAP of 7.1%. On the other hand, the authors of [16] report a MAP of 4.0% for a feed-forward DNN, and a MAP of 4.6% for a recurrent neural network using long short-term memory (LSTM) cells. In our experiments (see Sec. 3.4), we obtained a MAP of 3.6% with a feed-forward neural network with a single hidden layer of 1,000 rectified linear units (ReLUs). To bridge the gap between the neural network and the SVMs, we enhance the neural network by grafting elements of SVMs one at a time.

3.1 Explicit Kernel Mapping

By inspecting the predictions made by the feed-forward neural network, the first problem we realized was that the model did not have enough resolution in the low-dimensional input space. The neural network always assigned similar scores to instances whose features fell close in the input space, while the SVMs were able to single out many positive instances. Therefore we decided to map the 18-dimensional NCVs to high-dimensional vectors using the kernel mapping

of the SVMs, which is a composition of the χ^2 kernel and the RBF kernel.

The χ^2 kernel has been found to work well with histogram data (*e.g.* in [17]). NCVs, being L_1-normalized, are histograms, so the χ^2 kernel is suitable for them as well. For two D-dimensional histograms $\mathbf{x} = [x_1, \ldots, x_D]$ and $\mathbf{y} = [y_1, \ldots, y_D]$, the χ^2 kernel is defined as:

$$k_{\chi^2}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} \frac{2x_i y_i}{x_i + y_i} \tag{1}$$

and a distance measure can be derived from the kernel:

$$d^2_{\chi^2}(\mathbf{x}, \mathbf{y}) = k_{\chi^2}(\mathbf{x}, \mathbf{x}) + k_{\chi^2}(\mathbf{y}, \mathbf{y}) - 2k_{\chi^2}(\mathbf{x}, \mathbf{y}) = \sum_{i=1}^{D} \frac{(x_i - y_i)^2}{x_i + y_i} \tag{2}$$

The χ^2 kernel falls in the range of $[0, 1]$, and the χ^2 distance lies in $[0, \sqrt{2}]$.

The radial basis function (RBF) kernel is a kernel that depends only on the distance between the two arguments:

$$k_{\mathrm{RBF}}(\mathbf{x}, \mathbf{y}) = \exp[-\gamma d^2(\mathbf{x}, \mathbf{y})] \tag{3}$$

Plugging the χ^2 distance $d_{\chi^2}(\mathbf{x}, \mathbf{y})$ into the RBF kernel, we get the χ^2-RBF kernel. The SVMs in [18] use $\gamma = 4$.

A kernel $k(\mathbf{x}, \mathbf{y})$ can be understood as the inner product of the two input arguments mapped into a high-dimensional feature space via a feature mapping function $\Phi(\cdot)$:

$$k(\mathbf{x}, \mathbf{y}) = \langle \Phi(\mathbf{x}), \Phi(\mathbf{y}) \rangle \tag{4}$$

Because SVMs can be solved in the dual space, it is enough to know the formula of the kernel $k(\cdot, \cdot)$, and the mapping function $\Phi(\cdot)$ can be kept implicit. To use kernels in neural networks, however, $\Phi(\cdot)$ must be implemented explicitly.

Following [23], we construct an approximation of the χ^2 mapping function, which is a 306-dimensional real vector consisting of the following elements ($D = 18, \Delta = 2\pi/17$):

$$\begin{aligned} \Phi_{\chi^2}(\mathbf{x}) = [&\sqrt{\Delta \cdot x_i}, \sqrt{2\Delta \cdot x_i \mathrm{sech}(\pi\omega)} \cos(\omega \log x_i), \\ &\sqrt{2\Delta \cdot x_i \mathrm{sech}(\pi\omega)} \sin(\omega \log x_i)] \\ &(i = 1 \ldots D, \omega = \Delta \ldots 8\Delta) \end{aligned} \tag{5}$$

And using the method in [24], we approximate the RBF mapping function as follows:

$$\begin{aligned} \Phi_{\mathrm{RBF}}(\mathbf{x}) = [&\cos\langle \boldsymbol{\omega}_1, \mathbf{x} \rangle, \ldots, \cos\langle \boldsymbol{\omega}_N, \mathbf{x} \rangle, \\ &\sin\langle \boldsymbol{\omega}_1, \mathbf{x} \rangle, \ldots, \sin\langle \boldsymbol{\omega}_N, \mathbf{x} \rangle] \end{aligned} \tag{6}$$

where $\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_N$ are 306-dimensional real vectors sampled from the Gaussian distribution $\mathcal{N}(0, 2\gamma I)$. We choose $N = 5,000$, resulting in 10,000-dimensional feature vectors.

3.2 Squared Hinge Loss

The second problem we found with the feed-forward neural network lies with the loss function. The network had 15 sigmoid output nodes, one for each event, whose values stood for the probabilities of each event occurring in the clip. The traditional cross-entropy loss function was chosen as the objective of the optimization. In the face of a highly unbalanced training corpus (on average only 3% positive instances), the network spent most of its effort driving the scores of the negative instances very close to zero, instead of boosting the scores of the few positive instances. This is because the cross-entropy loss still has a non-zero gradient no matter how close the outputs are already to the target. On the contrary, the hinge loss in SVMs has a value of zero as long as an instance is correctly classified and far enough from the decision boundary. Even though the training data are highly unbalanced, the loss function only takes contribution from the few instances falling within the margin.

We replace the sigmoid output nodes of the neural network with linear output nodes, and apply the hinge loss function. An instance having class label $t \in \{-1, +1\}$ and an output $y \in \mathbb{R}$ will incur a hinge loss of

$$\mathrm{hinge}(yt) = \max(0, 1 - yt) \tag{7}$$

Some previous works [25, 26] investigating the hinge loss on neural networks used the squared hinge loss. The squared hinge loss is also used in the LibLINEAR toolkit [27]. In our experiments, we have also found the squared hinge loss to work better than the hinge loss.

3.3 Data Augmentation

To further deal with the problem of data imbalance, we performed augmentation on the training data. We perturb the NCVs of the positive class to produce new training instances. However, simply adding Gaussian noise is not feasible, because the NCVs must be kept non-negative and L_1-normalized. To comply with this constraint, we generate a perturbed NCV $\mathbf{y}$ from an existing NCV $\mathbf{x}$ by sampling from a Dirichlet distribution $D(\alpha x_1, \ldots, \alpha x_D)$ [28, p. 594]. The mean of this Dirichlet distribution is equal to $\mathbf{x}$, and the parameter α controls its degree of concentration. For each training clip in which at least one of the target events occurs, we generate 9 extra training instances with $\alpha = 10,000$.

3.4 Experiments

We implemented neural networks with the Theano toolkit [29]. Our initial feed-forward neural network had 18 input nodes, one hidden layer of 1,000 ReLU units, and 15 sigmoid output nodes. The network was optimized for the cross-entropy loss, using the gradient descent algorithm with a Nesterov momentum coefficient [30] of 0.9. We adopted the adaptive learning rate schedule as in [16]. The best initial learning rate was selected using 5-fold cross validation.

Starting from this initial model, we made the following incremental modifications one by one:

1. Transform the input into 306-dimensional real vectors using the χ^2 kernel mapping;
2. Transform the input into 10,000-dimensional real vectors using the χ^2-RBF kernel mapping;
3. Replace the sigmoid output nodes with linear units, and the cross-entropy loss with a squared hinge loss;
4. Perform data augmentation on the training instances.

For each setup, Table 1 shows the average MAP obtained with 4 neural networks trained with different random initializations. It can be seen that each modification made a little contribution, and finally the neural network matched the performance of the SVMs.

4. RECURRENT SUPPORT VECTOR MACHINES

The model we have constructed so far is essentially an SVM with an extra hidden layer. Although its performance doesn't yet exceed that of conventional SVMs, the extra hidden layer opens up the possibility of recurrence. By

Model	MAP (%)
Feed-forward NN	3.6
+ χ^2 kernel	4.2
+ RBF kernel	5.8
+ Squared hinge loss	6.9
+ Data augmentation	7.2
SVM	7.1

Table 1: *Mean average precision after every modification made to the feed-forward neural network inspired by SVMs.*

making the hidden layer recurrent, the model will be able to process NCV sequences of variable lengths. And we call such a recurrent model a *recurrent SVM*.

A similar model with the same name has been proposed in [31], where the output of an RNN is used as the input of an SVM. Our model differs from [31] in two points: first, the kernel mapping in our model is applied *before* the recurrent layer, while in [31] the kernel mapping is a part of the SVM, *i.e.* applied *after* the recurrent layer; second, we train our model with the stochastic gradient descent (SGD) algorithm, while a genetic algorithm is used in [31].

In this section, we explore four setups of recurrence: the hidden units may be either ReLU units or LSTM cells [32]; the hidden layer may be either unidirectional (one chain going forward) or bidirectional (one chain going forward and one chain going backward). In the two unidirectional setups, the 15 linear output nodes are only connected to the hidden units of the last frame; in the two bidirectional setups, the output nodes are connected to the last frame of both chains (note that the last frame of the backward chain actually corresponds to the first frame of the clip).

We trained all the four types of recurrent SVMs with 500 hidden units in each chain. The input sequences were shortened from m frames to m^β frames, where $\beta \in \{0.3, 0.5, 0.6, 0.7\}$. This is because training tended to diverge if the input sequences were too long. We used the stochastic gradient descent algorithm with a batch size of 15,000 frames, and the same adaptive learning rate schedule as in Sec. 3. Nesterov momentum was not applied. The initial learning rate was selected using 5-fold cross validation.

With the capacity brought by the recurrence, all the recurrent SVM models demonstrated overfitting to the training data. The models were able to produce scores larger than $+1$ for almost all the positive instances and scores smaller than -1 for almost all the negative instances, thereby achieving a MAP of nearly 100%. On the other hand, the MAP on the test data started to degenerate after reaching a peak. We resorted to early stopping by monitoring the 5-fold cross-validation MAP at each epoch; if the maximum cross-validation MAP didn't get updated for 5 successive epochs, we stopped the training and took the result of the epoch with the highest cross-validation MAP as the final result.

Table 2 compares the MAP obtained with different types of recurrence and input sequence lengths. For each combination, four recurrent SVMs were trained with different random initializations. The numbers in the table are the average MAP of the four runs (or three if one of the runs diverged). "Divergent" means at least two runs diverged.

We can see that all the recurrent SVMs are able to outperform non-recurrent models by a remarkable margin. Bidirectional models generally achieve higher MAPs than unidirectional models. LSTM cells are able to make use

β	0.3	0.5	0.6	0.7
Median sequence length	5	14	24	41
Unidirectional ReLU	8.4	5.7	2.8	Divergent
Bidirectional ReLU	8.8	6.4	6.1	Divergent
Unidirectional LSTM	7.5	7.8	6.0	Divergent
Bidirectional LSTM	8.0	8.2	6.4	6.5

Table 2: *Mean average precision (MAP) of recurrent SVMs with different setups and different input lengths.*

of slightly longer sequences ($\beta = 0.5$) than ReLU units ($\beta = 0.3$), but unfortunately their performance falls short.

We experimented with a number of other techniques in the hope of boosting the performance of recurrent SVMs with LSTM cells. We tried to make the state of the models at every time step contribute to the loss function (instead of only the last time step); we also tried to train the models for longer sequences using models for shorter sequences as a starting point, as done in [16]. Nevertheless, neither of these techniques made a significant difference.

An inspection at the average precision of each event individually reveals that the improvement from the feed-forward neural network to the recurrent SVM mainly comes from the events on which the baseline SVMs already perform well. That is, recurrent SVMs are good at detecting the "easier" events better. As for the "difficult" events where the baseline SVMs exhibit poor performance ($< 3\%$ AP), the recurrent SVM made a small improvement on some, and performed as poorly as the baseline SVM on others. This may indicate that the current feature set does not contain the necessary information for detecting these events.

5. CONCLUSION

In this paper, we studied the task of audio-based event detection using sequences of "noiseme confidence vectors" as input features. We contrasted a feed-forward neural network with SVMs, and pointed out the elements that made SVMs suitable for the task: the χ^2-RBF kernel mapping, and the squared hinge loss function. By grafting these elements onto the feed-forward neural network, and with the help of data augmentation, we were able to make it match the performance of SVMs; by further making the hidden layer recurrent, we constructed recurrent SVMs that outperformed the baseline SVMs. The excellent performance of recurrent SVMs is attributed to the improved ability of detecting the "easier" events.

The best recurrent SVM we built used bidirectional chains of ReLU units. LSTM cells, being able to keep a longer memory, should be able to outperform simple ReLU units in theory, but we did not observe this in our experiments. It remains a direction for future research to make recurrent SVMs with LSTM cells reach the performance they deserve.

6. ACKNOWLEDGMENTS

This work used the Extreme Science and Engineering Discovery Environment (XSEDE), which is supported by NSF grant number OCI-1053575.

This work was partially funded by Facebook, Inc. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Facebook, Inc.

7. REFERENCES

[1] W. M. Campbell, D. E. Sturim, and D. A. Reynolds, "Support vector machines using GMM supervectors for speaker verification", in *IEEE Signal Processing Letters*, vol. 13, no. 5, pp. 308-311, 2006.

[2] Q. Jin, P. F. Schulam, S. Rawat, S. Burger, D. Ding, and F. Metze, "Event-based video retrieval using audio", in *Proc. of Interspeech*, pp. 2085-2088, 2012.

[3] N. Dehak, P. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-end factor analysis for speaker verification", in *IEEE Trans. on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788-798, 2011.

[4] X. Zhuang, S. Tsakalidis, S. Wu, P. Natarajan, R. Prasad, and P. Natarajan, "Compact audio representation for event detection in consumer media", in *Proc. of Interspeech*, pp. 2089-2092, 2012.

[5] B. Elizalde, H. Lei, and G. Friedland, "An i-vector representation of acoustic environments for audio-based video event detection on user generated content", in *IEEE Int'l Symposium on Multimedia*, pp. 114-117, 2013.

[6] S. Pancoast and M. Akbacak, "Bag-of-audio-words approach for multimedia event classification", in *Proc. of Interspeech*, pp. 2105-2108, 2012.

[7] B. Byun, I. Kim, S. M. Siniscalchi, and C.-H. Lee, "Consumer-level multimedia event detection through unsupervised audio signal modeling", in *Proc. of Interspeech*, pp. 2081-2084, 2012.

[8] S. Chaudhuri, M. Harvilla, and B. Raj, "Unsupervised learning of acoustic unit descriptors for audio content representation and classification", in *Proc. of Interspeech*, pp. 2265-2268, 2011.

[9] S. Burger, Q. Jin, P. F. Schulam, and F. Metze, "Noisemes: manual annotation of environmental noise in audio streams", technical report CMU-LTI-12-07, Carnegie Mellon University, 2012.

[10] Z. Kons and O. Toledo-Ronen, "Audio event classification using deep neural networks", in *Proc. of InterSpeech*, pp. 1482-1486, 2013.

[11] O. Gencoglu, T. Virtanen, and H. Huttunen, "Recognition of acoustic events using deep neural networks", in *Proc. of the 22nd European Signal Processing Conference*, pp. 506-510, 2014.

[12] E. Cakir, T. Heittola, H. Huttunen, and T. Virtanen, "Polyphonic sound event detection using multi-label deep neural networks", in *IEEE Int'l Joint Conference on Neural Networks*, 2015.

[13] M. Ravanelli, B. Elizalde, K. Ni, and G. Friedland, "Audio concept classification with hierarchical deep neural networks", in *Proc. of the 22nd European Signal Processing Conference*, pp. 606-610, 2014.

[14] K. Ashraf, B. Elizalde, F. Iandola, M. Moskewicz, J. Bernd, G. Friedland, and K. Keutzer, "Audio-based multimedia event detection with DNNs and sparse sampling", in *Proc. of the 5th ACM Int'l Conference on Multimedia Retrieval*, pp. 611-614, 2015.

[15] Y. Wang, S. Rawat, and F. Metze, "Exploring audio semantic concepts for event-based video retrieval", in *Proc. of ICASSP*, pp. 1360-1364, 2014.

[16] Y. Wang, L. Neves, and F. Metze, "Audio-based multimedia event detection using deep recurrent neural networks", in *Proc. of ICASSP*, 2016.

[17] S. Rawat, P. F. Schulam, S. Burger, D. Ding, Y. Wang, and F. Metze, "Robust audio-codebooks for large-scale event detection in consumer videos", in *Proc. of Interspeech*, pp. 2929-2933, 2013.

[18] L. Bao, *et al.*, "Infomedia @ TRECVID 2011", in *Proc. of TREC Video Retrieval Evaluation*, 2011.

[19] NIST, "2011 TRECVID multimedia event detection evaluation plan", Online: `http://www.nist.gov/itl/iad/mig/upload/MED11-EvalPlan-V03-20110801a.pdf`

[20] F. Metze, S. Rawat, and Y. Wang, "Improved audio features for large-scale multimedia event detection", in *IEEE Int'l Conference on Multimedia and Expo*, pp. 1-6, 2014.

[21] Y. Yan, Y. Yang, D. Meng, G. Liu, W. Tong, A. G. Hauptmann, and N. Sebe, "Event oriented dictionary learning for complex event detection", in *IEEE Trans. on Image Processing*, vol. 24, no. 6, pp. 1867-1878, 2015.

[22] F. Eyben, F. Weninger, F. Gross, and B. Schuller, "Recent developments in OpenSMILE, the Munich open-source multimedia feature extractor", in *Proc. of ACM Multimedia*, pp. 835-838, 2013.

[23] A. Vedaldi and A. Zisserman, "Efficient additive kernels via explicit feature maps", in *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 34, no. 3, pp. 480-492, 2012.

[24] A. Rahimi and B. Recht, "Random features for large-scale kernel machines", in *Advances in Neural Information Processing Systems*, pp. 1177-1184, 2007.

[25] Y. Tang, "Deep learning with linear support vector machines", in *ICML Workshop: Challenges in Representation Learning*, 2013.

[26] S. Chen and Y. Wang, "Convolutional Neural Network and Convex Optimization", UCSD class project, 2014.

[27] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin, "LIBLINEAR: A library for large linear classification", in *The Journal of Machine Learning Research*, vol. 9, pp. 1871-1874, 2008.

[28] L. Devroye, *Non-Uniform Random Variate Generation*, Springer-Verlag, ISBN 0-387-96305-7.

[29] 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 9th Python for Scientific Computing Conference*, pp. 1-7, 2010.

[30] Y. Nesterov, "A method of solving a convex programming problem with convergence rate O(1/sqr(k))", in *Soviet Mathematics Doklady*, vol. 27, pp. 372-376, 1983.

[31] J. Schmidhuber, M. Gagliolo, D. Wierstra, and F. Gomez, "Evolino for recurrent support vector machines", technical report IDSIA-19-05, 2005.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory", in *Neural Computation*, vol. 9, pp. 1735-1780, 1997.