

Article

Animal Sound Classification Using Dissimilarity Spaces

Loris Nanni¹, **Sheryl Brahn**², **Alessandra Lumini**³, **Gianluca Maguolo**¹¹ DEI. Via Gradenigo 6, 35131 Padova, Italy; loris.nanni.nanni@dei.unipd.it (L.N.); gianluca.maguolo@phd.unipd.it (G.M.);² Department of Information Technology and Cybersecurity, Missouri State University, 901 S. National Street, Springfield, MO 65804, USA; sbrahn@missouristate.edu³ DISI, University of Bologna, Via dell'Università 50, 47521 Cesena, Italy; alessandra.lumini@unibo.it

* Correspondence: loris.nanni@unipd.it

Abstract: The classifier system proposed in this work combines the dissimilarity spaces produced by a set of Siamese neural networks (SNNs) designed using 4 different backbones, with different clustering techniques for training SVMs for automated animal audio classification. The system is evaluated on two animal audio datasets: one for cat and another for bird vocalizations. Different clustering methods reduce the spectrograms in the dataset to a set of centroids that generate (in both a supervised and unsupervised fashion) the dissimilarity space through the Siamese networks. In addition to feeding the SNNs with spectrograms, additional experiments process the spectrograms using the Heterogeneous Auto-Similarities of Characteristics. Once the similarity spaces are computed, a vector space representation of each pattern is generated that is then trained on a Support Vector Machine (SVM) to classify a spectrogram by its dissimilarity vector. Results demonstrate that the proposed approach performs competitively (without ad-hoc optimization of the clustering methods) on both animal vocalization datasets. To further demonstrate the power of the proposed system, the best stand-alone approach is also evaluated on the challenging Dataset for Environmental Sound Classification (ESC50) dataset. The MATLAB code used in this study is available at <https://github.com/LorisNanni>.

Keywords: audio classification; dissimilarity space; siamese network; ensemble of classifiers; pattern recognition; animal audio

1. Introduction

Over the last decade, research in sound classification and recognition has gained in popularity and rapidly broaden in its application from the more traditional focus on speech recognition [1] and music genre classification [2] to biometric identification [3], computer-aided heart sound detection [4], environmental audio scene and sound recognition [5, 6], biodiversity assessment [7], human voice classification and emotion recognition [8], English accent classification and gender identification [9], to list a few of a widerange of application areas. As with research in pattern recognition generally, the features fed into classifiers were initially engineered, which in the case of sound applications meant extracting from raw audio traces such descriptors as the Statistical Spectrum Descriptor and Rhythm Histogram [10].

In the last decade, researchers began exploring the possibility of visually representing audio signals to apply more powerful image classification descriptors and techniques. Initially, visual representations of audio traces centered around the display overtime of the frequency spectrum: examples of these representations include the spectrogram [11] and Mel-frequency Cepstral Coefficients spectrogram [12]. Although a spectrogram is typically a graph with two dimensions (time and frequency), additional dimensions can be included, such as pixel intensity [13], which includes at each time step the representation of an audio signal's amplitude in a specific frequency. Initially, popular texture descriptors, such as Haralick's Grey Level Co-occurrence Matrices (GLCMs) [14], Gabor filters [15] and Local Binary Patterns (LBP) [16] and its variants [17] were extracted from these spectrograms for the task of music genre classification [18-20]. Fusions of a large set of state-of-the-art texture descriptors were then experimentally applied to spectrograms, and certain combinations were shown to enhance further the accuracy of music genre classification [2].

The rise in popularity of deep learning due to affordable Graphic Processing Units (GPUs) changed the trajectory in machine learning research. Deep learners, such as the Convolutional Neural Network (CNN), produced far superior results to most other classifiers when it came to image classification. Consequently, more attention was placed on representing acoustic traces through visual representations so that deep learning approaches could be applied. Engineered features diminished in importance since deep classifiers learn which patterns perform best for a specific problem during the training process. However, engineered features have been shown to augment deep learning approaches when fused. Early work with CNNs applied to visual representations obtained state-of-the-art in chord detection and recognition [21, 22] and in music genre classification [23]. In [23], for example, spectrograms were converted into GCLM maps and trained on CNNs. In [24], canonical approaches, such as LMP-trained SVMs were fused with CNNs and shown to outperform previous systems.

Another development in sound classification involves the design of deep learners and feature sets specific to audio classification. For example, in [25], the authors explored variations in CNN architecture and parameters, discovering that the Rectified Linear Units (ReLU) instead of stochastic gradient descent combined with the Hessian Free optimization and sigmoid units reduced training time; in [26] a novel *sparse coding* CNN was developed and shown to perform well if not better than the state-of-the-art for sound event recognition and retrieval. Also of note is the hybrid multimodal deep learning approach proposed in [27] for multilabel music genre classification that combined album cover images, reviews, and audio tracks; this system was shown to outperform the single-modality approaches.

When it comes to animal sound classification, the focus of this study, fusions of CNNs with other methods to classify animals have been evaluated in [28] and [29] for fish identification using the Fish and MBARI benthic animal dataset and in [30] for bioacoustic bird species classification using a dataset of 5428 bird flight calls from 43 species. In [28], the authors combined engineered features with CNNs, and in [30], deep learning was combined with shallow learning. Both demonstrated that the fusions performed better than the standalone approaches.

Animal sound classification is a vibrant area of research. Many benchmark datasets are available for a variety of animals, such as birds [31, 32], whales [33], frogs [31], bats [32], and cats [34]. Animal audio classification can broadly be divided into two categories: the CNN approach discussed above and fingerprinting [35], which involves a compact audio representation for comparing audio segments in terms of similarity and dissimilarity [36]. Both approaches, however, suffer from limitations: fingerprinting can only find exact matches, and CNNs require large datasets for accurate training; many animal datasets are small because of the difficulty of collecting samples.

The superiority of combining the deep learning approach with fingerprinting is demonstrated in [37], where a Siamese Neural Network (SNN) produced semantic representations of audio signals. SNNs have been applied to sound classification in [37], [38], and [39] and have the advantage over the canonical CNN in their ability to generalize. In [40], a system was developed based on dissimilarity spaces, such as that proposed in [41] for brain image classification, where a distance model was learned by training a SNN [42] on dissimilarity values. This system combined different clustering approaches to generate a dissimilarity space that was then used to train an SVM. The clustering methods transformed the spectrograms in a bird [43] and cat [34, 44] dataset to a set of centroids that were used to generate a vector space representation for each pattern. This vector was then used to train a SVM. Results showed that this approach worked better than the standalone CNNs.

The system proposed in this work is similar to [40] in that it generates a dissimilarity space from the training set using an SNN to define a distance function from the input spectrograms. The objective at this point in the process is to maximize the distance separating the patterns of the different classes. Unlike [40], however, four different CNN architectures are selected for the twin classifiers, and both the original input spectrograms and the spectrograms processed by Heterogeneous Auto-Similarities of Characteristics [45] make up the inputs to the SNNs. In the testing phase, the different SNNs compare two samples to obtain a measure of their dissimilarity. Although it is that case that the entire training set can function as the centroids of the dissimilarity space, it is desirable to reduce dimensionality by selecting a smaller number of prototypes. In this work, both a supervised and unsupervised clustering algorithm is used to reduce dimensionality. The dissimilarity space represents each input (both the original and the processed spectrograms) by its distance from each of

the centroids, or prototypes, a distance that is learned by the SNNs. In other words, the SNNs compare a given spectrogram to each of the centroids to obtain a dissimilarity feature vector. A support vector machine (SVM) is then trained on these features.

The approach taken in this paper is evaluated on the same animal vocalization datasets as in [40], i.e., on cats [34] and birds [7]. In addition, the system is tested on the challenging benchmark dataset for Environmental Sound Classification (ESC-50) [46]. As in [40], an ensemble of SVMs trained on different dissimilarity spaces generated by changing the value of k in the clustering approaches and network topologies are combined by sum rule. Performance is compared with both the state-of-the-art as well as with fusions with the state-of-the-art. Results demonstrate the power of using dissimilarity spaces based on an ensemble of SNNs, in particular if coupled with a standard CNN approach.

The remainder of this paper is broken down into the following sections. In Section 2, a detailed overview of the proposed approach is provided; in Section 3, SNN is discussed along with the different CNN architectures that make up the subnetworks. In Section 4, the supervised and unsupervised clustering methods are outlined, and, in Section 5, experimental results are presented along with comparisons with the state-of-the-art. The paper concludes in Section 6 with an overview and some suggestions for future research.

2. Proposed Approach

The proposed system presented here for spectrogram classification extends that proposed in [40] and is schematized in Figure 1, which illustrates the approach using one SNN. The pseudocode in Algorithms 1 and 2 that correspond to Figure 1 are detailed in the remainder of this section.

The process begins by generating a similarity space during the training phase via a learning distance measure $d(x, y)$ from a set of prototypes $P = p_1, \dots, p_k$. The distance measure is learned by four SNNs trained 1) to maximize the similarity between pairs of spectrograms belonging to the same class and 2) to minimize the similarity for pairs of spectrograms belonging to different classes. The set of prototypes generated in this phase are the k centroids of the clusters produced by a clustering procedure that is both supervised and unsupervised. What results is a feature vector $f \in R^k$ that represents training sample x in the dissimilarity space, where a given f_i is the distance between x and the prototype p_i : $f_i = d(x, p_i)$. Once these features have been calculated, they are used to train a SVM classifier.

In the testing phase, each input spectrogram is represented in the dissimilarity space by simply calculating its distance to P , and the resulting feature vectors of the input images are then classified by SVM. In our experiment we test both spectrograms and HASC [45], a 2D descriptor extracted from the spectrogram, as input of the classification process. The method for extracting HASC is outlined in section 2.5

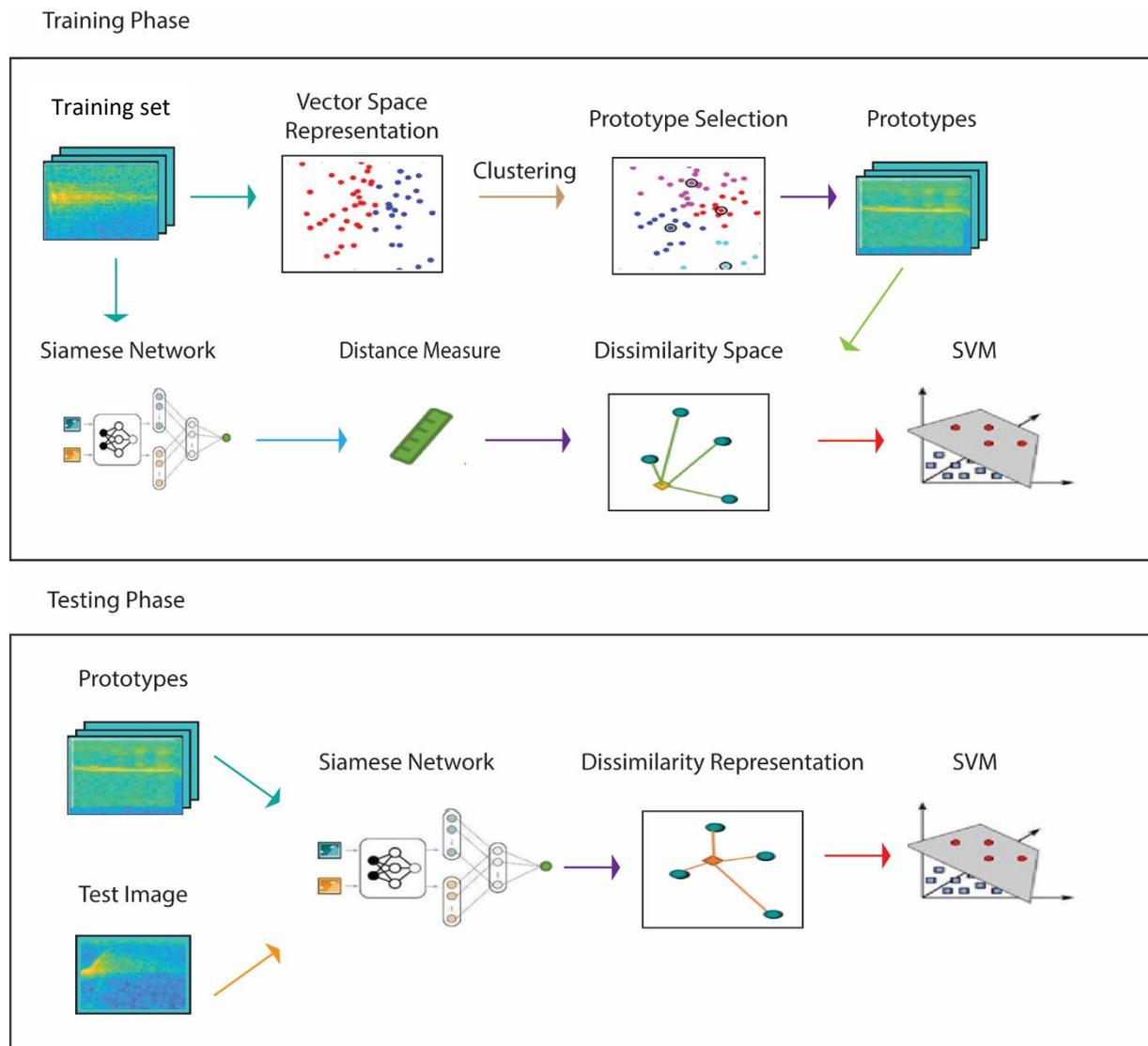


Figure 1. Proposed approach scheme.

Algorithm 1 Training phase

Input: Training images ($imgsTrain$), training labels ($labelTrain$), number of training iterations ($trainIterations$), batch size ($trainBatchSize$), number of centroids (k), and clustering technique ($type$).

Output: Trained SNN ($tSNN$), set of centroids (C), and trained SVM (svm).

- 1: $tSNN \leftarrow TRAINSIAMESE(imgsTrain, labelTrain, trainIterations, trainBatchSize)$
 - 2: $P \leftarrow CLUSTERING(imgsTrain, labelTrain, k, type)$
 - 3: $F \leftarrow GETDISSSPACEPROJECTION(imgsTrain, P, tSNN)$
 - 4: $tSVM \leftarrow TRAINSVM(labelTrain, F)$
-

Algorithm 2 Testing phase

Input: Test images ($imgsTest$), trained SNN ($tSNN$), Set of centroids (C), Trained SVM ($tSVM$).

Output: Actual test labels ($labelTest$).

- 1: $F \leftarrow GETDISSSPACEPROJECTION(imgsTest, P, tSNN)$
 - 2: $labelTest \leftarrow PREDICTSVM(F, tSVM)$
-

2.1. Siamese Neural Network Training

The SNN, which is described in more detail in Section 3, is trained to compare pairs of spectrograms by returning some measure of their similarity. Algorithm 3 presents the pseudocode for this phase of the proposed method, which corresponds to step 1 of algorithm 1. The SNN architecture is defined in steps 2 and 3 as defined in Algorithm 3. Steps 5–8 are repeated for each training iteration. Step 5 randomly extracts *batchSize* spectrogram pairs from the training set via the function GETSIAMESEBATCH. Step 6 inputs the pairs into the SNN and computes loss and gradients for gradient descent. Steps 7 and 8 take the gradients and loss and use them to update the weights of the fully connected (FN) layer of the Siamese subnetworks.

Algorithm 3 Siamese training pseudocode

Input: Training image (*trainImgs*), training labels (*trainLabels*), batch size (*batchSize*), and iterations (*numberOfIterations*).

Output: Trained SNN (*tSNN*).

```

1: function TRAINSIAMESE
2:   subnet ← NETWORK([inputLayer, ..., FullyConnectedLayer])
3:   fcWeights ← randomWeights
4:   for iteration ← from 1 to numberOfIterations do
5:     X1, X2, pairLabels ← GETSIAMESEBATCH(trainImgs, trainLabels, batchSize)
6:     gradients, loss ← EVALUATE(subnet, X1, X2, pairLabels)
7:     UPDATE(subnet, gradients)
8:     UPDATE(fcWeights, gradients)
9:   end for
10:  return tSNN ← subnet, fcWeights
11: end function

```

Note: if SNN fails to converge on the training set, the training phase is repeated.

2.2. Prototype Selection

Prototype selection involves extracting a total of k prototypes from the training set. The dimensionality would be too high to select every spectrogram in the training set as a prototype. Dimensionality can be reduced by employing clustering techniques to calculate k centroids. In this work we perform both prototype selection separately per each class (kc prototypes per class) and global (i.e. unsupervised) clustering (k prototypes). In both cases this would reduce the dimension of the resulting dissimilarity space. Algorithm 4 presents the pseudocode for prototype selection. As can be observed, a clustering technique is selected from a set of four possible clustering methods, each of which is employed separately to cluster the training samples belonging to each class. For the sake of space, only the pseudocode for the supervised clustering methods are included, though it should be noted that this work used both supervised and unsupervised clustering approaches.

Algorithm 4 Clustering pseudocode

Input: Training images (*imgsTrain*), training labels (*labelTrain*), number of clusters (k), and clustering technique (*type*).

Output: Centroids P .

```

1: function CLUSTERING
2:   numClasses ← number of classes from labelTrain
3:   kc ←  $k / \text{numClasses}$ 
4:   for  $i$  ← from 1 to numClasses do
5:     images ← images of the class  $i$  from imgsTrain
6:     switch type do
7:       case "k-means"  $P_i$  ← KMEANS(imgs, kc)
8:       case "k-medoids"  $P_i$  ← KMEDOIDS(imgs, kc)
9:       case "hierarchical"  $P_i$  ← HIERARCHICAL(imgs, kc)
10:      case "spectral"  $P_i$  ← SPECTRAL(imgs, kc)
11:      $P$  ←  $P \cup P_i$ 
12:   end for
13:   return  $P$ 
14: end function

```

2.3. Projection in the Dissimilarity Space

Classically, classifiers are trained to predict patterns within a feature space. It is also possible, as demonstrated here, for patterns to be represented in a dissimilarity space such that every pattern x is represented by its dissimilarity to a selected set of prototypes $P = p_1, \dots, p_k$ and by a dissimilarity vector defined as

$$F(x) = [d(x, p_1), \dots, d(x, p_i), \dots, d(x, p_k)], \quad (5)$$

where the similarity of pattern $d(x, y)$ is obtained using a trained SNN.

Projection in dissimilarity space \mathcal{R}^k is described in Algorithm 5 where each input image (stored in X in step 3) is compared with the k centroids (stored in P) using the trained SNN $tSNN$ with the PREDICTSIAMESE function (step 4). The output is the feature space F that includes the projected features of all the input images.

Algorithm 5 Projection in the Dissimilarity space pseudocode

Input: Images ($imgs$), Centroids (P), number of centroids (k), and trained SNN ($tSNN$).

Output: Feature vectors (F).

```

1: function GETDISSPACEPROJECTION
2:   for  $j \leftarrow$  from 1 to SIZE( $imgs$ ) do
3:      $X \leftarrow imgs[j]$ 
4:      $F[j] \leftarrow$  PREDICTSIAMESE( $tSNN, X, P$ )
5:   end for
6:   return  $F$ 
7: end function

```

2.4. Support Vector Machine (SVM)

SVM [47] is a well-known binary learner that represents training samples as points in space (see Figure 2). The goal of SVM training (function TRAINSVM) is to find at least one hyperplane such that it separates the data that belongs to each of two classes. Prediction (function PREDICTSVM) is accomplished by mapping an unseen pattern to the side of the hyperplane representing the class for a given data point.

A hyperplane $D(x)$ is defined as

$$D(x) = w \cdot x - b, \quad (6)$$

where x is the input vector, w is the normal vector of the hyperplane, and $b/\|w\|$ is the distance of the hyperplane from the origin. The optimal hyperplane is that which maximizes the distance (or margin) from the nearest data point of a class and is defined as $2/\|w\|$.

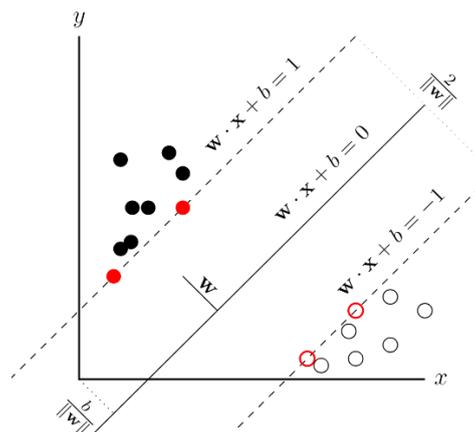


Figure 2. SVM.

As illustrated in Figure 2, class assignment is determined as follows: when $D(x_i) \geq +1$, then the i -th point x_i is assigned to the first class; when $D(x_i) \leq -1$, then it is assigned to the second class. Those points that lie on the margin line, the so-called *support vectors*, are defined as $D(x_i) = \pm 1$, and it is they that completely define the solution to the binary classification problem.

SVM, as defined above, does a poor job discriminating input that is not linearly separable in its original space. This difficulty can be overcome by selecting kernel functions that map the data into a higher dimensional space where separation is possible. A good kernel function, however, must also be computationally efficient.

Although SVM is binary, it can be applied to nonbinary or multilabel problems by training an ensemble of SVMs and combining their decisions. In the experiments reported here, the *One-Against-All* method is applied, where an SVM is trained systematically to discriminate each class against all the others combined. The pattern is then predicted to belong to that class that produces the highest confidence score.

2.5. Heterogeneous Auto-Similarities of Characteristics (HASC)

HASC [45] is applied to heterogeneous dense feature maps. It encodes linear relations by covariances (COV) and nonlinear associations with entropy combined with mutual information (EMI). Three reasons for considering covariance matrices as descriptors is that 1) they are low in dimensionality, 2) robust to noise (but with the exception that outlier pixels can render them more sensitive to noise), and 3) the covariance among two features is optimally able to encapsulate the features of the joint PDF (but with the caveat that they be linked by a linear relation). HASC obviates these limitations by combining COV with EMI.

The entropy (E) of a random variable measures the uncertainty of its value, and the mutual information (MI) of two random variables captures their generic linear and nonlinear dependencies. The way HASC utilizes these advantages is by dividing an image into patches from which it generates an EMI matrix ($d \times d$), such that the main diagonal entries encapsulate the amount of unpredictability of the d features, and the off-diagonal entries (element i, j) capture the mutual dependency between two features, that is, the i -th and j -th feature. HASC is the concatenation of vectorized EMI and COV. Specifically, the MI of a pair of random variables A, B is

$$MI(A, B) = \int_A \int_B p(a, b) \log \left(\frac{p(a, b)}{p(a)p(b)} \right) db da, \quad (1)$$

where $p(a)$, $p(b)$ and $p(a, b)$ are the PDF of A , the PDF of B , and their joint PDF, respectively. If $A = B$, then MI is the entropy of A :

$$E(A) = MI(A, A) = - \int_A p(a) \log(p(a)) da \quad (2)$$

If there exists a finite set K of realization pairs, MI can be estimated as a sample mean inside the logarithm, thus:

$$MI(A, B) \approx \frac{1}{K} \sum_{k=1}^K \log \left(\frac{p(a_k, b_k)}{p(a_k)p(b_k)} \right). \quad (3)$$

A fast and efficient method for calculating from the K realizations the probabilities inside the logarithm is to estimate them by building a joint 2D normalized histogram of values A and B , such that each $p(a_k, b_k)$ is estimated taking the value of the 2D histogram bin containing the pair a_k, b_k . In this way, $p(a_k)$ and $p(b_k)$ can be estimated by summing all the bins corresponding to a_k and b_k , respectively. Thus, the i, j -th component of the EMI matrix related to the patch P can be defined as

$$EMI_{p\{ij\}} = \frac{1}{K} \sum_{k=1}^K \log \left(\frac{\tilde{p}(z_{ki}, z_{kj})}{\tilde{p}(z_{ki})\tilde{p}(z_{kj})} \right), \quad (4)$$

where $\tilde{p}(\dots)$ and $\tilde{p}(\cdot)$ are the probabilities estimated with the histogram and z_{ki} is the i -th feature at pixel K .

In this study, HASC is extracted from the whole spectrogram. Given the function HASC, the output FEAT is a three-dimensional matrix ($w \times h \times d$) containing the features extracted from the image (d is the number of low-level features). The number of bins used to evaluate the histograms in the EMI computation is 28, and the number of low-level features is 6 (default parameters). FEAT is reshaped

to build the vector $img = [FEAT(:,1) FEAT(:,2); FEAT(:,3) FEAT(:,4); FEAT(:,5) FEAT(:,6)]$, and img is resized to the right dimension for the input into a CNN.

3. Siamese Neural Network (SNN)

SNN, initially proposed in [42], is a class of neural networks that contains two or more twin networks that share the same weights and parameters. As illustrated in Figure 3, an SNN has two inputs that compare two patterns and one output that corresponds to the similarity between the two inputs. In other words, an SNN identifies correlations between two different input patterns (see [48] for a fairly comprehensive overview of SNN). As shown in Figure 3, the SNNs used in this work are composed of five components, as described below.

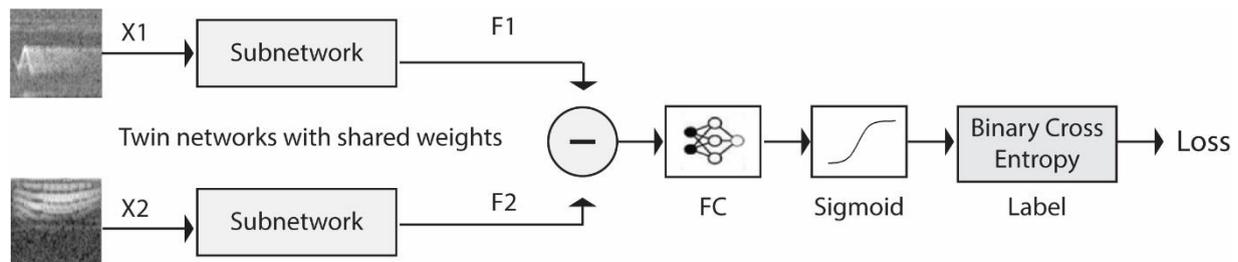


Figure 3. Siamese Neural Network architecture.

3.1. The two identical twin subnetworks

In this study, four twin CNN subnetworks are utilized. As illustrated in Figure 4, CNNs are constructed by assembling specialized layers composed of neurons. Some of the more common layers found in CNN architectures include convolutional, activation (ReLU), pooling, and fully connected (FC) layers. The convolutional layers extract features from the input volume and work by convolving a local region of the input volume (the receptive field) to filters of the same size. The output of the convolutional layers produces the input for the next layer, typically a nonlinear activation layer, such as ReLU. The activation layer improves the classification performance of the network. The pooling layers are frequently interspersed between the convolutional layers and perform nonlinear downsampling operations (e.g. max pooling) that reduce the dimension of the representation and the computational complexity of the CNN. FC layers usually make up the last hidden layers and have fully connected neurons to all the activations in the previous layer. The specific CNN architecture for the four twin subnetworks are outlined in Table 1.

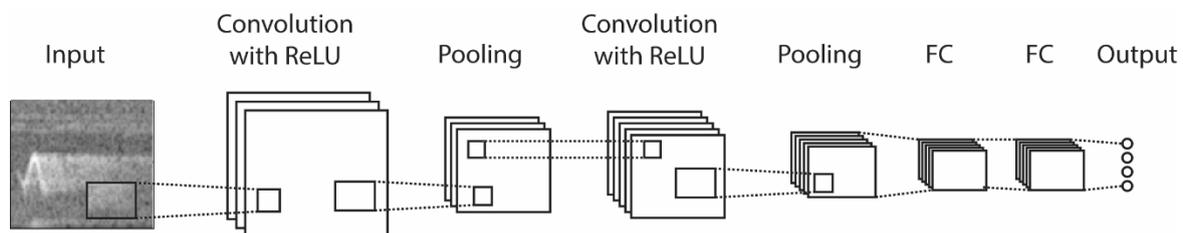


Figure 4. Basic CNN architecture.

Two activation functions are explored here: the ReLU activation function [49] and the Leaky ReLU [50]. The well-known ReLU activation function for the points (x_i, y_i) is defined as:

$$y_i = f(x_i) = \begin{cases} 0, & x_i < 0 \\ x_i, & x_i \geq 0 \end{cases} \quad (7)$$

The Leaky ReLU variation of ReLU is defined as

$$y_i = f(x_i) = \begin{cases} ax_i, & x_i < 0 \\ x_i, & x_i \geq 0 \end{cases} \quad (8)$$

Tables 1. CNN Siamese Networks (1-4) layers.

Siamese Network 1			
#	Layers	Filter Size	Number of Filters
1	Input Layer	224 × 224 images	
2	2D Convolution	10 × 10	64
3	ReLU		
4	Max Pooling	2 × 2	
5	2D Convolution	7 × 7	128
6	ReLU		
7	Max Pooling	2 × 2	
8	2D Convolution	4 × 4	128
9	ReLU		
10	Max Pooling	2 × 2	
11	2D Convolution	5 × 5	64
12	ReLU		
13	Fully Connected	Returning a 4096-Dimensional Vector	

Siamese Network 2			
#	Layers	Filter Size	Number of Filters
1	Input Layer	224 × 224 images	
2	2D Convolution	5 × 5	64
3	LeakyReLU		
4	2D Convolution	5 × 5	64
5	LeakyReLU		
6	Max Pooling	2 × 2	
7	2D Convolution	3 × 3	128
8	LeakyReLU		
9	2D Convolution	3 × 3	128
10	LeakyReLU		
11	Max Pooling	2 × 2	
12	2D Convolution	4 × 4	128
13	LeakyReLU		
14	Max Pooling	2 × 2	
15	2D Convolution	5 × 5	64
16	LeakyReLU		
17	Fully Connected	Returning a 2048-Dimensional Vector	

Siamese Network 3			
#	Layers	Filter Size	Number of Filters
1	Input Layer	224 × 224 images	
2	2D Convolution	7 × 7	128
3	Max Pooling	2 × 2	
4	ReLU		
5	2D Convolution	5 × 5	128
6	Max Pooling	2 × 2	
7	ReLU		
8	Max Pooling	2 × 2	
13	Fully Connected	Returning a 4096-Dimensional Vector	

Siamese Network 4			
#	Layers	Filter Size	Number of Filters
1	Input Layer	224 × 224 images	
2	2D Convolution	7 × 7	128
3	Max Pooling	2 × 2	
4	ReLU		
5	Max Pooling	2 × 2	
6	2D Convolution	5 × 5	256
7	ReLU		
8	2D Convolution	3 × 3	64
9	Max Pooling	2 × 2	
10	2D Convolution	3 × 3	128
11	ReLU		
12	2D Convolution	5 × 5	64
13	Fully Connected	Returning a 4096-Dimensional Vector	

The subnetworks of the Siamese CNNs 1-4 each learn the features that best represent the information in the spectrograms that are the input patterns to the two input nodes (X_1 and X_2) and return either a 2048 or 4096-dimensional feature vector (F_1 and F_2). The subnetworks share the same parameters and weights training.

3.2. Subtract Block

As illustrated in Figure 3, the output vectors of the subnetworks are subtracted, resulting in a feature vector Y that represents the features that differ between the two input spectrogram images, thus:

$$Y = |F_1 - F_2| \quad (9)$$

3.3. The FC Layer

In accordance with the method outlined in [41], the FC layer learns the distance model for calculating the dissimilarity. The output vector of the Subtract Block becomes the input to the FC block, which outputs the dissimilarity value for the pair of spectrogram patterns.

3.4. The Sigmoid Function

The sigmoid function is then applied to the dissimilarity value to convert it to a probability value in the range [0, 1] using the standard logistic function:

$$S(x) = 1 / (1 + e^{-x}) \quad (10)$$

3.5. The Binary Cross Entropy (BCE) Component

A popular loss function is the BCE. Given the prediction of the model and the correct observation binary label of 1 if the two spectrograms belong to the same class or 0 if they do not, BCE returns a measure the model's performance. Loss functions are computed to train a network by adjusting its weights. BCE takes the probability obtained from the sigmoid function and computes the gradients of the loss function by considering the weights of the network. In two-class problems, BCE is calculated as

$$BCE(y, p) = -(y \log(p) + (1 - y) \log(1 - p)), \quad (11)$$

where y is the binary value that indicates whether the class label c is correct for the observation o , p is the predicted probability that observation o is of class c , and \log is the natural logarithm.

4. Clustering

Clustering is a procedure that divides unlabeled patterns into groups that maximize the commonality between members within a group and their differences with members belonging to other groups. Clustering techniques often calculate the mean vector, or centroid, of all the patterns within a cluster when forming the clusters. Centroids encapsulate salient characteristics of patterns belonging to a cluster; for this reason, they can be used to reduce the dissimilarity space size without losing too much significant information. Even more information can be retained if the number of centroids representing each class is increased.

Both supervised and unsupervised clustering approaches are considered here. If kc clusters are extracted with a supervised approach; then, in each of NK classes, $kc \times NK$ clusters are extracted using unsupervised clustering on the training set. A description of the four clustering methods used in this study follows.

4.1. K-Means

K-means is one of the most popular clustering algorithms. It partitions patterns into k clusters by placing each observation into a cluster based on the nearest centroid. The MATLAB Statistics and Machine Learning Toolbox was used here with the Euclidean Distance measure.

The standard k-means algorithm involves four steps:

1. Randomly select a centroid from among the data points.
2. For each data point x remaining in the training set, compute the distance $d(x)$ between it and the nearest center that has already been selected.
3. Choose a new data point at random as a new centroid via a weighted probability distribution, where a point x is chosen with probability proportional to $d(x)^2$.
4. Repeat Steps 2 and 3 until k centers have been selected.

4.2. K-Medoids

K-medoids is a clustering technique that follows the same general logic behind k-means but differs in the specific way it partitions data points into clusters: K-medoids minimizes the sum of distances between a given pattern and the center of that pattern's cluster. In short, the center of a cluster in K-Means ends up being the centroid of the cluster, but the center in K-Medoids is a member (medoid) of the cluster. In other words, a medoid is that member in a cluster whose sum of distances from all other members is minimal.

The standard K-medoids algorithm involves three steps:

1. Step one is a build-step where each k cluster is associated with a potential medoid. There are many ways to select the first medoid; the standard MATLAB's implementation does this by means of the k-means++ heuristic.
2. Step two is a swap-step where within each point in a cluster is tested as a potential medoid by checking whether the sum of the within-cluster distances is smaller when using that point as the medoid. If it is smaller than the point is defined as the new medoid. Every point is then assigned to the cluster with the closest medoid.
3. The last step repeats steps 1–4 until medoids can no longer be swapped, in which case the algorithm converges, or until the maximum number n of iterations is reached.

4.3. Hierarchical Clustering

Hierarchical clustering is a clustering technique that partitions data by building a tree of clusters divided into n levels selected for the specific classification task. In general, hierarchical clustering is divided into two types:

1. *Agglomerative*, where each pattern starts in its own cluster. Then, by moving up the hierarchy, each cluster in the new level is obtained by merging two clusters in the previous level.
2. *Divisive*, where all patterns start in one cluster. Then, by moving down the hierarchy, each pair of

clusters is obtained by splitting one above.

In this work, the agglomerative hierarchical approach is employed as this is the default MATLAB implementation. The MATLAB algorithm involves three steps:

1. Using a distance metric, find the similarity or dissimilarity between every pair of data points in the dataset.
2. Aggregate data points into a binary hierarchical cluster tree by linking points in pairs according to their distance. As observations are paired into binary clusters, the newly formed clusters are grouped into larger clusters until a hierarchical tree is formed.
3. Determine where the hierarchical tree is cut into clusters. MATLAB's *cluster* function prunes branches off the bottom of the hierarchical tree and assigns all the observations below each cut to a single cluster. This produces k clusters.

Once the hierarchical tree is generated, the mean vectors of each cluster are computed.

4.4. Spectral

Spectral clustering partitions data into groups via the data's undirected similarity graph as represented by a similarity (or adjacency) matrix. Every node in the similarity graph is a data point. A pair of nodes are connected by an edge if their similarity is larger than a threshold (typically set to 0).

This clustering algorithm involves the following matrices:

- The similarity matrix, which is a square symmetrical matrix representing the similarity graph. If M is the similarity matrix, then the value of each cell m_{ij} is the similarity value of two connected nodes in the graph, which, in this application task, represent two spectrogram pairs s_i, s_j .
- The degree matrix, which is a diagonal matrix that is obtained by summing the rows of the similarity matrix rows and is defined as

$$D_g(i, i) = \sum_j m_{i,j}, \quad (12)$$

where D_g is the degree matrix, and m_{ij} is a similarity matrix cell value.

- The Laplacian Matrix, which is yet another matrix representation of the similarity graph. The Laplacian Matrix is defined as

$$L = D_g - M. \quad (13)$$

The algorithm for spectral clustering is a five-step process:

1. Define a local neighborhood for each data point in the dataset. There are many ways to define a neighborhood. The nearest-neighbor method is the default setting in the MATLAB implementation of spectral clustering. Then compute the pairwise similarities of each datapoint in the neighborhood using a distance metric.
2. Calculate the Laplacian matrix L .
3. Create a matrix V containing columns v_1, \dots, v_k , where the columns are the k eigenvectors, i.e., the *spectrums* (hence the name), corresponding to the k smallest eigenvalues of the Laplacian matrix.
4. Perform k-means or k-medoids clustering by treating each row of V as a datapoint.
5. Assign the original observations in the dataset to the same clusters as their corresponding rows.

5. Experimental Results

The proposed approach is tested and compared with canonical approaches with each using a stratified ten-fold cross-validation protocol. The performance indicator is the classification accuracy and methods were tested on the following animal vocalization datasets:

- *BIRDz*, which functioned as a control and a real-world audio dataset in [43]. The real-world tracks were collected from the Xeno-canto Archive (<http://www.xeno-canto.org/>). *BIRDz* includes samples of 11 North American bird species: (1) Blue Jay, (2) Song Sparrow, (3) Marsh Wren, (4) Common Yellowthroat, (5) Chipping Sparrow, (6) American Yellow Warbler, (7) Great Blue Heron, (8) American Crow, (9) Cedar Waxwing, (10) House Finch, and (11) Indigo Bunting. This dataset is composed of five different spectrograms: 1) constant frequency, 2) frequency modulated whistles, 3) broadband pulses, (4) broadband with varying frequency components, and 5) strong harmonics, making for a total of 2762 bird acoustic observations with 339 detected "unknown" events that include noise and unknown species' vocalizations. *BIRDz* has 3101 samples for 12 classes if the "unknown class" is included.
- *CAT*, [34, 44] is a dataset that contains ten balanced classes of approximately 300 samples per class for a total of 2962 samples. The ten classes represent the following cat vocalizations: (1) Resting, (2) Warning, (3) Angry, (4) Defense, (5) Fighting, (6) Happy, (7) Hunting mind, (8) Mating, (9) Mother call, and (10) Paining. The average duration of each sample is approximately 4s. Samples were garnered from such online resources as Kaggle, Youtube, and Flickr.

In this section we report experiments aimed at evaluating the proposed system by varying several components: i.e. the input images (spectrograms or HASC images), the topology of the Siamese Network (NN1, NN2, NN3, NN4), the clustering algorithm (K-Means, K-Medoids, Hierarchical, Spectral), the clustering modality (unsupervised or supervised, i.e. clustering on the whole training set or on each class), the number of prototypes ($kc = 15, 30, 45, 60$).

In the first experiment, reported in Table 2, only K-means clustering is explored. For each approach, the performance of fusion by sum rule of the four SVMs trained using the dissimilarity spaces built with all tested values for $kc = 15, 30, 45, 60$ is provided. Performance is reported for only NN1 and NN2 (the first two network topologies) to reduce computation time.

The ensemble in Table 2 are obtained by varying the input data (Sp= spectrograms, HASC= HASC images), the type of clustering (unsupervised or supervised) and the network topology. The clustering method is fixed to K-means for all the methods and number of prototypes belongs to the following set [15, 30, 45, 60]. The column #classifiers recaps the number of classifiers in the ensemble and the first column ("Name") assign a name to the ensemble.

The best average performance is obtained by the ensemble FA1_2 in the last row (which is the sum rule of the methods in the first 8 rows). On BIRD there is a boost in performance with NN1 and NN2 using the HASC images instead of the spectrograms, while on the CAT dataset, HASC images boost the performance of NN2 but not NN1.

Table 2. Performance obtained by k-means clustering.

Name	Input image	Network topology	Clustering method	Clustering type	#Prototypes	#classifiers	CAT	BIRD
Sup-1	Sp	NN1	K-means	S	15, 30, 45, 60	4	78.64	92.46
Sup-2	Sp	NN2	K-means	S	15, 30, 45, 60	4	76.95	92.74
UnS-1	Sp	NN1	K-means	U	15, 30, 45, 60	4	81.69	92.73
UnS-2	Sp	NN2	K-means	U	15, 30, 45, 60	4	75.25	92.80
HSUp-1	HASC	NN1	K-means	S	15, 30, 45, 60	4	78.64	94.52
HSUp-2	HASC	NN2	K-means	S	15, 30, 45, 60	4	81.69	93.22
HUnS-1	HASC	NN1	K-means	U	15, 30, 45, 60	4	79.32	94.53
HUnS-2	HASC	NN2	K-means	U	15, 30, 45, 60	4	81.36	92.97
FSp-1	Sp	NN1	K-means	S,U	15, 30, 45, 60	8	81.02	92.79
FSp-2	Sp	NN2	K-means	S,U	15, 30, 45, 60	8	76.95	92.77
FA-1	Sp,HASC	NN1	K-means	S,U	15, 30, 45, 60	16	82.37	94.50
FA2	Sp,HASC	NN2	K-means	S,U	15, 30, 45, 60	16	83.73	94.11
FA1_2	Sp,HASC	NN1+NN2	K-means	S,U	15, 30, 45, 60	32	84.41	94.37

The second experiment is aimed at comparing the clustering methods: to this aim in Table 3, the performance of different clustering approaches using HSup-2 approach (i.e. HASC images as input, NN2 as network topology and unsupervised version of the clustering). The last row reports the ensemble F_Clu obtained as the sum rule among the above 4 approaches. F_Clu obtains the highest performance, though this gain is only slighter higher than that of K-means. All the clustering algorithms are quite similar in performance, and since their fusion do not gain evident advantage against a single approach, in the next experiments we use only K-means strategy for clustering varying the number of prototypes kc .

Table 3. Performance obtained considering different clustering algorithms.

Name	Input image	Network topology	Clustering method	Clustering type	#Prototypes	#classifiers	CAT	BIRD
	HASC	NN2	K-means	S	15, 30, 45, 60	4	81.69	93.22
	HASC	NN2	K-Med	S	15, 30, 45, 60	4	81.02	92.85
	HASC	NN2	Hier	S	15, 30, 45, 60	4	81.69	93.01
	HASC	NN2	Spect	S	15, 30, 45, 60	4	80.00	93.13
F_Clu	HASC	NN2	All	S	15, 30, 45, 60	16	82.03	93.37

In Table 4, the four network topologies coupled with K-means clustering with HSup (i.e. HASC images as input and unsupervised version of the clustering) is evaluated. The last row, which reports the ensemble F_NN obtained as the sum rule among the above 4 approaches, produces the average best performance on this test.

Table 4. Performance obtained considering different network topologies.

Name	Input image	Network topology	Clustering method	Clustering type	#Prototypes	#classifiers	CAT	BIRD
	HASC	NN1	K-means	S	15, 30, 45, 60	4	78.64	94.52
	HASC	NN2	K-means	S	15, 30, 45, 60	4	81.69	93.22
	HASC	NN3	K-means	S	15, 30, 45, 60	4	78.64	94.91
	HASC	NN4	K-means	S	15, 30, 45, 60	4	82.37	93.33
F_NN	HASC	All	K-means	S	15, 30, 45, 60	16	84.07	94.99

Even better results are obtained by combining by sum rule all the approaches reported in the previous tables: the combined performance on CAT is 85.76 and on BIRD 95.08. It is clear that the ensemble strongly outperforms a simple Sup-1.

In the next table, the performance of an ensemble obtained by retraining Siamese HSup-1 is compared with ensembles obtained by varying the network topology. The results of Table 5 indicates that varying the network topology introduce diversity in the ensemble: while performance of 4, 8, and 16 networks HSup1 networks are increasing, but quite similar (with all three ensembles outperforming the single network), The ensembles named F_NN, and obtained varying the topology of the Siamese Network, show an evident performance gain. It is also interesting to observe the similar results of rows 2 and 3: both ensemble with have four networks, but the first is bade by simply retraining the same model, while the second has different numbers of prototypes: thus, varying the values of kc is not very important when building an ensemble; to obtain superior performance, it is necessary to vary the network topologies.

Table 5. Comparison between ensembles of reiterated Siamese Networks with NN1 and ensembles obtained considering different network topologies.

Name	Input image	Network topology	Clustering method	Clustering type	#Prototypes	#classifiers	CAT	BIRD
HSup1(1)	HASC	NN1	K-means	S	15	1	75.93	93.92
HSup1(4)	HASC	NN1	K-means	S	15	1×4	81.69	94.50
HSup-1	HASC	NN1	K-means	S	15, 30, 45, 60	4×1	78.64	94.52
HSup-1(8)	HASC	NN1	K-means	S	15, 30, 45, 60	4×2	80.68	94.56
HSup-1(16)	HASC	NN1	K-means	S	15, 30, 45, 60	4×4	81.02	94.63
F_NN(4)	HASC	All	K-means	S	15	4	83.39	94.73
F_NN(8)	HASC	All	K-means	S	15, 30	8	84.07	94.90
F_NN	HASC	All	K-means	S	15, 30, 45, 60	16	84.07	94.99

Finally, reported in Table 6 is a comparison between the Siamese networks and standard CNNs trained with spectrograms. The method labeled eCNN is the fusion among different CNNs (GoogleNet, VGG16, VGG19, and GoogleNetP365).

Table 6. Performance obtained considering different standard CNN.

Method	CAT	BIRD
OLD [40]	82.41	92.97
F_NN	84.07	94.99
GoogleNet	82.98	92.41
VGG16	84.07	95.30
VGG19	83.05	95.19
GoogleNetP365	85.15	92.94
eCNN	87.36	95.81
OLD+eCNN	87.76	95.95
FUS_n4c+eCNN	88.47	96.03

From the results reported in the previous tables, the following conclusions can be drawn:

- The best way for building an ensemble of Siamese networks is to combine different network topologies;
- The proposed F_NN ensemble clearly improves previous methods based on Siamese networks (cf. OLD in Table 6);
- F_NN obtains a performance that is similar to eCNN on BIRD but lower than eCNN on CAT;
- In both datasets, the best performance is obtained by sum rule between eCNN and FUS_NN (i.e. the fusion among CNNs and the Siamese networks).

To further validate our approach, we tested it on the ESC-50 benchmark audio classification dataset. To reduce computation time, only Sup-1 was tested. It obtained 52% accuracy but needed a high number of training iterations for network convergence. For the ESC-50 dataset, Sup-1 was trained for 25000 epochs rather than for 3000 epochs as was done for CAT and BIRD. For comparison, a simple 3-layer CNN with square filters trained on wideband mel-STFT [51] obtained an accuracy of 54%.

In Table 7, some additional state-of-the-art results in the literature are reported on the CAT and BIRD datasets. As can be observed, the performance of the ensembles described in this paper approaches those reported in the literature. Note that in Table 7, two results are reported from [34]; to distinguish these methods, they are labeled [34] and [34]-CNN.

Table 7. Literature results.

Descriptor	CAT	BIRD
[52]	–	96.3
[2]	–	95.1
[7]	–	93.6
[44]	87.7	–
[34]	91.1	–
[34]–CNN	90.8	–
[43]	–	96.7*

More papers in the field of acoustic animal classification need to present results across more than one dataset so that methods can be compared more accurately. The experiments presented in this paper speak to the robustness of the proposed approach: competitive classification accuracy, as compared to the state-of-the-art in the literature, has been obtained on two different problems without any ad-hoc dataset parameter optimization. These results were produced by following a clear and unambiguous testing protocol. The value of reporting methods across datasets means that the results reported here can reasonably serve as a baseline for later comparisons in research in this area.

6. Conclusion

This work presents a method for classifying animal vocalizations using four Siamese networks and dissimilarity spaces. Different clustering techniques taking both a supervised and unsupervised approach were used for dissimilarity space generation. A set of SVMs was trained on the dissimilarity spaces generated by the clustering methods using different numbers of centroids and the outputs of the four Siamese networks. The SVMs were combined by sum rule to obtain a highly competitive ensemble as tested on two datasets of animal vocalizations. In addition, experimental results demonstrated that the proposed approach presented in this work could be combined with other state-of-the-art approaches to improve classification accuracy. The fusions improved performance on both audio classification problems, outperforming the standalone approaches.

Future work in this area will focus on experimentally deriving ensembles using the same approach. The goal will be to assess the approach proposed here for generalizability across many sound classification problems, such as those cited in [33, 41]. This will involve testing the proposed method by adding more supervised and unsupervised clustering techniques and additional Siamese network architectures.

Author Contributions: L.N. conceived of the presented idea., G.M., L.N., A.L. performed the experiments. S.B. wrote the manuscript. S.B. provided some resources.

Funding: This research received no external funding.

Acknowledgments: The authors thank NVIDIA Corporation for supporting this work by donating a Titan Xp GPU.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Padmanabhan, J. and M.J.J. Premkumar, *Machine learning in automatic speech recognition: A survey*. IETE Technical Review, 2015. **32**: p. 240-251.
2. Nanni, L., et al., *Combining visual and acoustic features for audio classification tasks*. Pattern Recognition Letters, 2017. **88**(March): p. 49-56.
3. Sahoo, S.K., T. Choubisa, and S.R.M. Prasanna, *Multimodal Biometric Person Authentication : A Review*. IETE Technical Review, 2012. **29**(1): p. 54-75.
4. Li, S., et al., *A Review of Computer-Aided Heart Sound Detection Techniques*. BioMed research international, 2020. **2020**: p. 5846191-5846191.
5. Chandrakala, S. and S. Jayalakshmi, *Generative Model Driven Representation Learning in a Hybrid Framework for Environmental Audio Scene and Sound Event Recognition*. IEEE Transactions on

- Multimedia, 2019. **22**(1): p. 3-14.
6. Chachada, S. and C.J. Kuo. *Environmental sound recognition: A survey*. in 2013 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference. 2013.
 7. Zhao, Z., et al., *Automated bird acoustic event detection and robust species classification*. Ecological Informatics, 2017. **39**: p. 99-108.
 8. Badshah, A.M., et al., *Speech Emotion Recognition from Spectrograms with Deep Convolutional Neural Network*. 2017 International Conference on Platform Technology and Service (PlatCon), 2017: p. 1-5.
 9. Zeng, Y., et al., *Spectrogram based multi-task audio classification*. Multimedia Tools Appl., 2019. **78**(3): p. 3705-3722.
 10. Lidy, T. and A. Rauber, *Evaluation of feature extractors and psycho-acoustic transformations for music genre classification*. ISMIR, 2005: p. 34-41.
 11. Wyse, L. *Audio spectrogram representations for processing with convolutional neural networks*. ArXiv Prepr, 2017. **ArXiv1706.09559**.
 12. Rubin, J., et al., *Classifying heart sound recordings using deep convolutional neural networks and mel-frequency cepstral coefficient*, in Computing in Cardiology (CinC). 2016: Vancouver, Canada. p. 813-816.
 13. Nanni, L., Y.M.G. Costa, and S. Brahmam, *Set of texture descriptors for music genre classification*, in 22nd WSCG International Conference on Computer Graphics, Visualization and Computer Vision. 2014: Plzen, Czech Republic.
 14. Haralick, R.M., *Statistical and structural approaches to texture*. Proceedings of the IEEE, 1979. **67**(5): p. 786-804.
 15. Ojansivu, V. and J. Heikkila, *Blur insensitive texture classification using local phase quantization*, in ICISP. 2008. p. 236-243.
 16. Ojala, T., M. Pietikainen, and T. Maenpaa, *Multiresolution gray-scale and rotation invariant texture classification with local binary patterns*. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2002. **24**(7): p. 971-987.
 17. Brahmam, S., et al., eds. *Local Binary Patterns: New Variants and Applications*. 2014, Springer-Verlag: Berlin.
 18. Costa, Y.M.G., et al., *Music genre classification using LBP textural features*. Signal Processing, 2012. **92**: p. 2723-2737.
 19. Costa, Y.M.G., et al., *Music genre recognition using spectrograms*, in 18th International Conference on Systems, Signals and Image Processing. 2011. p. 151-154.
 20. Costa, Y.M.G., et al., *Music genre recognition using gabor filters and LPQ texture descriptors*, in 18th Iberoamerican Congress on Pattern Recognition. 2013. p. 67-74.
 21. Humphrey, E. and J.P. Bello, *Rethinking automatic chord recognition with convolutional neural networks*, in International Conference on Machine Learning and Applications. 2012. p. 357-362.
 22. Humphrey, E., J.P. Bello, and Y. LeCun, *Moving beyond feature design: deep architectures and automatic feature learning in music informatics*. International Conference on Music Information Retrieval, 2012: p. 403-408.
 23. Nakashika, T., C. Garcia, and T. Takiguchi, *Local-feature-map integration using convolutional neural networks for music genre classification*. Interspeech,, 2012: p. 1752-1755.
 24. Costa, Y.M.G., L.E.S. Oliveira, and C.N. Silla Jr., *An evaluation of Convolutional Neural Networks for music classification using spectrograms*. Applied Soft Computing 2017. **52**: p. 28-38.
 25. Sigtia, S. and S. Dixon, *Improved music feature learning with deep neural networks*, in IEEE International Conference on Acoustic, Speech and Signal Processing. 2014. p. 6959-6963.
 26. Wang, C.-Y., et al., *Recognition and retrieval of sound events using sparse coding convolutional neural network*, in IEEE International Conference On Multimedia & Expo (ICME). 2017. p. 589-594.
 27. Oramas, S., et al., *Multi-label music genre classification from audio, text and images using deep features*, in International Society for Music Information Retrieval (ISMIR) Conference. 2017. p. 23-30.
 28. Nanni, L., et al., *Ensemble of local phase quantization variants with ternary encoding*, in Local Binary Patterns: New Variants and Applications, S. Brahmam, et al., Editors. 2014, Springer-Verlag: Berlin. p. 177-188.
 29. Cao, Z., et al., *Marine animal classification using combined CNN and hand-designed image features*, in MTS/IEEE Oceans. 2015.
 30. Salamon, J., et al., *Fusing shallow and deep learning for bioacoustic bird species in IEEE International Conference on Acoustics, Speech and Signal Processings (ICASSP)*. 2017. p. 141-145.
 31. Acevedo, M.A., et al., *Automated classification of bird and amphibian calls using machine learning: a comparison of methods*. Ecological Informatics, 2009. **4**: p. 206-214.
 32. Cullinan, V.I., S. Matzner, and C.A. Duberstein, *Classification of birds and bats using flight tracks*. Ecological Informatics, 2015. **27**: p. 55-63.
 33. Fristrup, K.M. and W.A. Watkins, *Marine animal sound classification*. 1993, WHOI Technical Reports.

34. Pandeya, Y.R., D. Kim, and J. Lee, *Domestic cat sound classification using learned features from deep neural nets*. Applied Sciences, 2018. **8**(10): p. 1949.
35. Wang, A., *An industrial strength audio search algorithm*, in *ISMIR Proceedings*. 2003: Baltimore.
36. Haitisma, J. and T. Kalker. *A Highly Robust Audio Fingerprinting System*. in *ISMIR*. 2002.
37. Manocha, P., et al., *Content-Based Representations of Audio Using Siamese Neural Networks*. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018: p. 3136-3140.
38. Droghini, D., et al., *Few-Shot Siamese Neural Networks Employing Audio Features for Human-Fall Detection*, in *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence*. 2018, Association for Computing Machinery: Union, NJ, USA. p. 63–69.
39. Zhang, Y., B. Pardo, and Z. Duan, *Siamese Style Convolutional Neural Networks for Sound Search by Vocal Imitation*. IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2019. **27**(2): p. 429-441.
40. Nanni, L., et al., *Spectrogram classification using dissimilarity space*. Sensors, 2020. **10**(12): p. 4176.
41. Agrawal, A., *Dissimilarity learning via Siamese network predicts brain imaging data*. arXiv: Neurons and Cognition, 2019.
42. Bromley, J., et al., *Signature verification using a Siamese time delay neural network*. International Journal of Pattern Recognition and Artificial Intelligence, 1993. **7**(4).
43. Zhang, S.-h., et al., *Automatic Bird Vocalization Identification Based on Fusion of Spectral Pattern and Texture Features*. 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018: p. 271-275.
44. Pandeya, Y.R. and J. Lee, *Domestic Cat Sound Classification Using Transfer Learning*. Int. J. Fuzzy Logic and Intelligent Systems, 2018. **18**: p. 154-160.
45. San Biagio, M., et al., *Heterogeneous auto-similarities of characteristics (hasc): Exploiting relational information for classification*, in *IEEE Computer Vision (ICCV13)*. 2013: Sydney, Australia. p. 809-816.
46. Piczak, K.J., *ESC: Dataset for Environmental Sound Classification*, in *Proceedings of the 23rd ACM international conference on Multimedia*. 2015, Association for Computing Machinery: Brisbane, Australia. p. 1015–1018.
47. Vapnik, V., *The support vector method*, in *Artificial Neural Networks ICANN'97*. 1997, Springer, Lecture Notes in Computer Science. p. 261-71.
48. Chicco, D., *Siamese neural networks: An overview*, in *Artificial Neural Networks. Methods in Molecular Biology*, H. Cartwright, Editor. 2020, Springer Protocols: Humana, New York, NY. p. 73-94.
49. Glorot, X., A. Bordes, and Y. Bengio. *Deep Sparse Rectifier Neural Networks*. in *AISTATS*. 2011.
50. Maas, A.L. *Rectifier Nonlinearities Improve Neural Network Acoustic Models*. 2013.
51. Huzafah, M., *Comparison of Time-Frequency Representations for Environmental Sound Classification using Convolutional Neural Networks*. ArXiv, 2017. **abs/1706.07156**.
52. Nanni, L., et al., *Combining visual and acoustic features for music genre classification*. Expert Systems with Applications, 2016. **45**(45 C): p. 108-117.