NSCNet - A Nightingale Songs Clustering Network

Simone Marasi, Gaetano Signorelli and Daniele Sirocchi

Abstract. Deep Clustering (DC) combines together Deep Neural Networks (DNN) with classical clustering techniques, using the former for extracting the features, and the latter for creating groups of unlabeled data. DC makes it possible to cluster complex and high-dimensional data, pushing forward the boundaries of the previous state of the art models developed on the basis of classical clustering techniques. In our work, we implemented three different models: the BASENet, the VAENet and the NSCNet. The first is a very simple model, based on the Principal Component Analysis (PCA) followed by a clustering algorithm. The second uses a Variational Autoencoder (VAE) for creating the feature space, on which it is eventually applied a clustering technique. The third is an end-to-end model whose goal is to simultaneously learn a good feature representation while applying a clustering technique on the input data. Finally, the NSCNet results have been enhanced through the use of the ArcFace loss, and they have been compared with the Zipf's law, providing possible good insights about the dataset used for this work, which is made of several nightingale songs.

1. Introduction

1.1 Problem

Machine learning is endowed with several clustering techniques that have been used for decades, and that have been fundamental for working with unsupervised data. However, nowadays the data given as input to machine learning models is becoming increasingly complex, and this is the reason why classical clustering algorithms struggle to find good results. Deep Clustering allows to work with more complex data, such as images or raw audio files. The DC's goal is the same of the classical clustering techniques, namely to group together samples of the dataset that are characterized by similar features. This can be done by leveraging the feature extraction power of the Deep Neural Networks, which are capable of creating a feature space which is organized in a way manageable by classical clustering algorithms.

The presented work uses a private dataset that has been provided by Giacomo Costalunga, a Ph.D. student at the Max Plank institute for Ornithology, in Munich. The goal of the project is to cluster the elements that are in the dataset, where each sample represents the song of a nightingale.

1.2 Previous works

It is important to remark that one the most advanced branches of Artificial Intelligence is the one focused on Computer Vision tasks, and it has hundreds of available architectures in the literature. These architectures have greatly influenced models that work with totally different inputs, including the audio files used in this project. Thus, there are many models based on CNNs, such as [1], [2] and [4]. A huge number of architectures process audio files as images, typically through the use of Mel-spectrograms or chromagrams. Therefore, any Deep Clustering model that works with images could also work with audio files, and the literature contains a huge variety of techniques that could be used for this project, like autoencoders [3], transfer learning [4], transformers [5], and even more complicated ones, such as [6] that uses together a variational autoencoder and a generative adversarial network.

1.3 Selected choice

The proposed work uses Mel-spectrograms as inputs, and it consists of three different models. In this way it has been possible to work in an incremental way, starting from the simplest model, and finishing with the most complex one.

The first, called **BASENet**, performs a dimensionality reduction operation on the input through the Principal Component Analysis (PCA), and then it applies a clustering algorithm. The PCA operation is performed in order to decrease the dimensionality of the input, so as to reduce the effects of the curse of dimensionality that could badly influence the results of the clustering algorithm. Both k-means and OPTICS are used. This model is the simplest of the three, and its results represent a baseline for the more complex ones.

The second, called **VAENet**, is trained in two different stages. During the first stage, both the encoder and the decoder are trained together. Successively, only the encoder is kept, and it is used as the feature extractor of the input. These extracted features are eventually used for feeding the clustering algorithms (the same used in the BASENet).

The final, called **NSCNet,** is the most complex one. This is an **end-to-end** model whose goal is to solve the image classification task, where the labels of unsupervised data are created by the clustering algorithm applied while training the network. For this network, where the number of units of the last dense layer is fixed, only the k-means algorithm is used.

1.4 Clustering algorithms

Machine Learning has many alternative clustering algorithms, and for each of them there are strengths and weaknesses. Therefore, in this project has been made the choice of using two of them: k-means and OPTICS. The characteristics of these two algorithms are quite complementary, and therefore they have been considered as a good starting point for trying to solve the project's goals.

Due to its simplicity, speed and effectiveness, the **k-means** [8] is the most widely used clustering algorithm. It is a distance-based algorithm, and it is used in almost every architecture mentioned in the previous section, providing very often good results. However, it does not work well with non-convex clusters, it is badly influenced by noise and outliers, and knowing in advance the number of K is not an easy task, because it forces us to make assumptions that must be eventually validated by domain experts.

In the proposed work, the **metrics** for evaluating the k-means algorithm are the **inertia** (e.g., the elbow plot) and the **silhouette score**.

OPTICS [7] is a density-based algorithm, and it can be seen as a generalization of the DBSCAN algorithm by requiring one less parameter to tune (eps). Moreover, OPTICS does not require to know the number of clusters in advance, it works with any cluster shape, and it is robust to noise and outliers. However, its complexity is $O(N^2)$, it is very sensitive to hyper-parameters tuning and can have issues if there are clusters with different densities.

In the proposed work, the **metric** for evaluating the OPTICS algorithm is the **silhouette score**.

The clustering results have been plotted in a 2D space using the Linear Discriminative Analysis (LDA) reduction, and so exploiting the generated labels for a better representation compared to the Principal Component Analysis (PCA) reduction.

2. Input encoding

Applying deep learning techniques to digital recordings (e.g., audio files) requires the preliminary step of encoding: a representation of the data that could be passed as input of a neural network. The literature has proposed many different approaches, some more popular than others, that have been considered, trying to find the most suitable one for the intended goal. Particularly, the focus has fallen on three common possibilities, explained in the next three subsections.

2.1 Raw waveforms

The sound is represented as it is, a time series of values indicating the amplitude of its waveform in time. This approach is the most straightforward one, but it is rarely adopted in practice, due to a lack of many useful information that could be extracted beforehand by means of filters (see next points). In this way, the network will have to extract that information on its own, making the task more complicated. Furthermore, waveforms with typical values for sampling the frequency and the sound depth are usually memory-consuming. On the other hand, the raw data could be accompanied by other features such as the zero-crossing rate, the energy, the spectral centroid, and so on.

2.2 MFCC and Mel-Spectrograms

MFCC stands for Mel Frequency Cepstral Coefficients, and it is a way for processing the raw waveforms and transforming them into a signal expressed in the frequency domain; it is a specific type of spectrogram. The encoding consists of an RGB image. As shown in **Figure 1**, it is obtained by following these steps:

- Apply the Fourier Transform to the raw waveform and obtain a frequency spectrum
- Apply the log of the magnitude to the Fourier spectrum
- Perform DCT (discrete cosine transform)
- Convert each point into the Mel scale

Removing the third step (**Figure 2**) leads to the very widely used Mel-Spectrograms, which constitute the most adopted encoding strategy for audio signals in deep learning.

The Mel scale is an attempt to mimic how humans hear sounds. In particular, the human auditory system works on a logarithmic scale (e.g., low-frequency changes are much less noticeable than high-frequency changes). The formula used is:

$$Mel(f) = 2595log\left(1 + \frac{f}{700}\right) \quad (1)$$

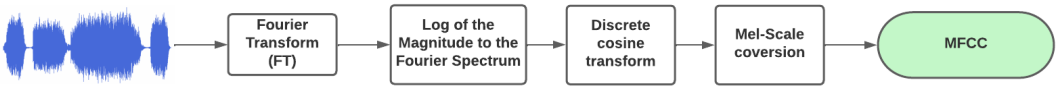

Fig. 1. MFCC

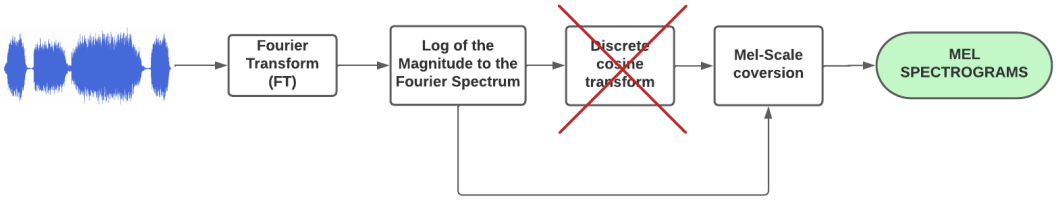

Fig. 2. Mel-spectrograms

2.3 Chromagrams

These are other spectrograms, less common than Mel-spectrograms, at least for general-purpose tasks. They are based on pitches, which represent relative highness or lowness of a sound. The higher the sound, the higher the pitch.

Pitches are specific frequencies that are distinguished by notes (indicated as letters) in music. Whenever a note is played, it falls in one of those pitches. And at the same time, each sound (considering a time-window of the original waveform) can be decomposed into a vector (chromo vector) containing the energy of each pitch. Additionally, the same note could be played with a multiple of its base frequency, denoted as octaves.

Pitches are 12, and they can give hints about which notes are most played in a song. For example, there could be some musical genres that change the notes very frequently, or others that play notes in a more stable way. With these considerations it could be possible, for example, to perform song genre classification, making chromagrams a reasonable choice in case of music-related signals (**Figure 3**).

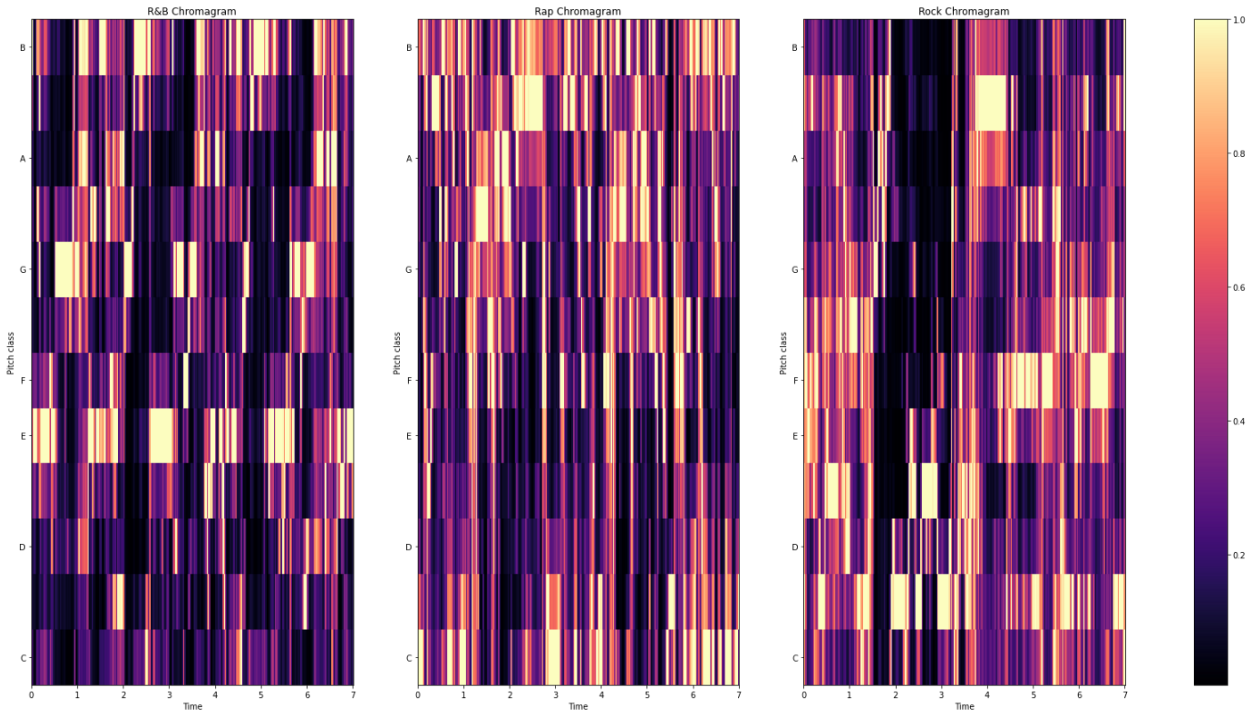

Fig. 3. For R&B (left), the artist decided to keep the song grounded around the E note, for Rap (center), pitches are highly scattered which makes sense given the genre leverages vocal cadence similar to that of percussion, and for Rock (right), it is centered around the note C# or D♭

2.4 Our choice

The choice between these three options has been the one based on **Mel-spectrograms**. In particular, raw waveforms have been discarded because of the mentioned negative considerations about them; while, between MFCCs, Mel-spectrograms and chromagrams, the second ones are absolutely the most popular in literature, making them quite reliable and allowing easier comparisons with state-of-the-art works. Even so, chromagrams seem to represent a valid alternative, and possibly a better choice, since nightingale songs can be approximately considered as pieces of music.

To prove this last assertion, an analysis has been accomplished on the songs: for each time-step the purity of notes has been checked (e.g., a chromo vector with 100% for a single pitch would be completely pure, while one with 0.08% for each of the 12 pitches would be completely noisy from a musical point of view). The purity measure adopted has been the entropy and it has been computed for all the time-windows of a song (for all the chromo vectors), taking the mean to get a general score for each recording. The entropy has been computed following the typical equation used in information theory:

$$H(X) = -\sum_{i=1}^{12} P(x_i) log_2 P(x_i) \quad (2)$$

Where X is the chromo vector and $P(x)$ represents the energy of a pitch.

Results show an average entropy of *2.53* (on a maximum of *3.58*) with a standard deviation of *0.36*; the general statistics are shown in **Figure 4**. To make some comparisons, the note-entropy for pure instrumental

recordings has also been tested, with an average value of *2.3-2.6*; while more noisy recordings presented an average around *3.2-3.5*. These observations point toward the fact that nightingale songs could be treated as music, thus chromagrams seem to be a very good encoding, considering also that they contain only selected useful frequencies, while other spectrograms do not. Therefore, this type of input encoding could be considered for a future work.

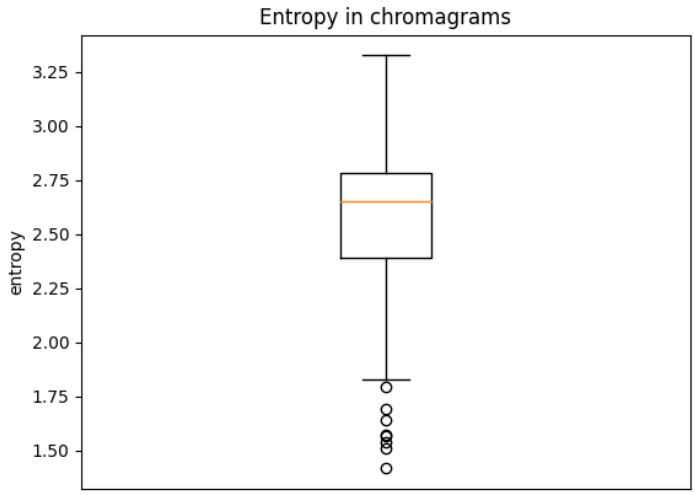

Fig. 4. Entropy computed on chromagrams obtained by processing nightingale songs

3. Dataset

3.1 Overview

The dataset is composed of a collection of **nightingale recordings**. The tracks are about recordings coming from all over the world, from Berlin to Ghana, obtained using high quality microphones. Such recordings are then grouped by birds and divided into two alternating channels, namely channel 1 and channel 2. The channel 2 is effectively the actual recording of the bird, while the channel 1 is a set of previously recorded songs used as stimuli to induce the following singing. The recordings provided are already split along channels and initially denoised. Moreover, for each bird, it is available a text file with the onsets and the offsets of each song so as to make the programmatic splitting of the songs easier. All the inputs have been then split through a Python script, obtaining little more than **11.000 songs** using both channels of the **36 birds** provided.

The songs extracted are then used to generate the actual inputs for the network architecture. As previously mentioned, the training and testing of the networks have been performed only with **Mel-spectrograms**, leaving the use of chromagrams as a possible future work. The handling of audio files has been possible through the use of Librosa [11], a Python package for music and audio analysis. Precisely, with this library the analysis of the audio files, and then the generation of the spectrograms, is quite straightforward thanks to a lot of helping built-in functions.

3.2 Analysis and preprocessing

Before the generation of the spectrograms, a **statistical analysis** of the audio files, regarding the five birds initially provided, has been performed for trying to highlight some possible similarities and insights from the plots generated. Some audio features (such as amplitude, duration, zero-crossing rate, frequency and energy) have been taken into account during this analysis, but the overall results were not so useful for the task itself except for the fact that they have been convincing about the requirement of a deeper approach to the problem.

The first attempt to generate spectrograms has not been so successful, since the obtained image was resulting quite dirty in the band below 1200-1500 Hz. Things got better with the application of a static high-pass filter with the threshold set at 1500 Hz. Another point to tackle before generating spectrograms for all the songs

extracted has been the choice of the **image dimensions**, and the consequent choice to pad or crop songs at a given duration threshold. In order to do this, it has been performed an accurate statistical analysis about the **duration of the songs**. From this analysis, and also looking at the boxplots, it has been turned out that a suitable threshold for the durations could be 11 seconds, since the average value computed was about 7 seconds and only the 3% of the songs last over the threshold, and then needs to be cropped. The last step before launching the spectrogram generation has been the decision to fix the image **file dimensions**, eventually chosen in 512x64 pixels.

Other types of input preprocessing, such as normalization and whitening, have been considered, but in a different fashion, namely letting the choice of the use of these techniques to the single network training. In this way, it has been possible to perform a sort of ablation study about these transformations. Both techniques are typically used only before the PCA application, and are aimed to improve the final representation, possibly with a much smaller loss of data. In particular, normalization, at least theoretically, could be something that is not really beneficial for audio data (spectrograms) since it makes little sense to perform variance normalization. PCA, in fact, is invariant to the scaling of the data, and will return the same eigenvectors regardless of the scaling of the input. For what concerns whitening (namely an attempt to make the spectrum of the signal "more uniform"), it has the goal to obtain a better PCA representation, trying to transform the covariance matrix to an identity one. Further details can be found in the results section of the various implemented architectures.

4. BASENet

4.1 Architecture overview

The BASENet is a very simple model, and its structure is depicted in the following picture:

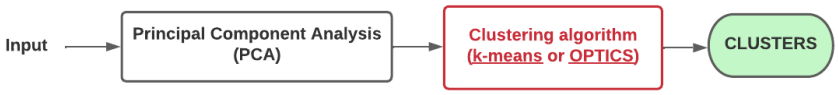

Fig. 5. BASENet architecture

The BASENet does not have any parameters to learn, and it does not have the expectation to obtain particularly good results. Nonetheless, these results can be used as a baseline for doing comparison with the outcomes of other models.

The **PCA** is responsible for decreasing the input dimensionality to a fixed dimension of 128. Before clustering, the PCA's output is whitened and l2-normalized [2].

Both **k-means** and **OPTICS** are tested, so as to be able to compare the results of the two different clustering algorithms.

4.2 Experiments, results and error analysis

The model is pretty simple, and the dataset is not very large. This allowed the exploration of a large portion of the parameter space.

PCA has been tested with 64, 128 and 256 dimensions. However, the results do not depict any relevant difference between the three different applied dimensions.

OPTICS has been applied by changing the *min_samples* parameter with the following values: 2, 4, 8, 16, 32, 64 and 128. When *min_samples* is very low, then the derived number of clusters is huge, and just a few of them dominate the whole dataset. On the opposite side, when *min_samples* is too high, then the number of

clusters is very small. Nevertheless, even with values in the middle, the silhouette score remains very low and there is not any valid hint on the correct number of clusters. Finally, **Figure 6** and **Figure 7** highlight that the silhouette score is very low, and the data is basically randomly scattered in the feature space.

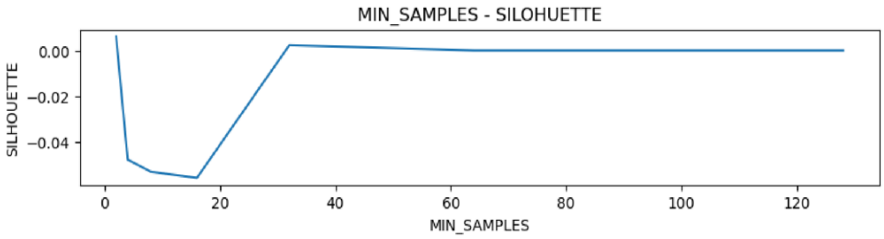

Fig. 6. Silhouette Score of the BASENet with OPTICS

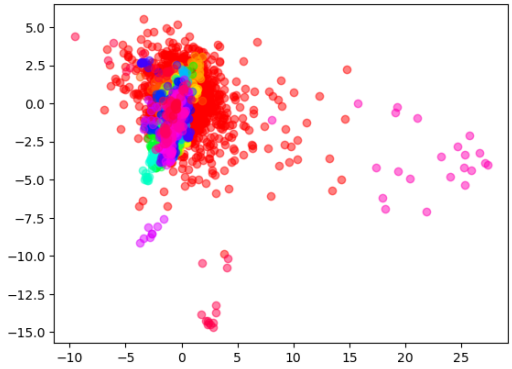

Fig. 7. Extracted features of the dataset plotted in 2d
with LDA compression and OPTICS with min_samples=16

The **k-means** has been applied by changing the parameter *K* with values 64, 128, 256, 1024 and 2048. Also in this case, the acquired outcome does not provide any good hint on how many different clusters there are in the dataset: the inertia and silhouette charts shown in **Figure 8** depict that the model seems to skew up the results. According to the silhouette score plot, the peak could be found with K=2048, but it is not very reasonable to think of having more than 2k clusters with a dataset of ~10k samples. Furthermore, the silhouette score remains very low for every K that has been tested. Finally, also the extracted features (**Figure 9**) portray a very bad sample distribution in the feature space.

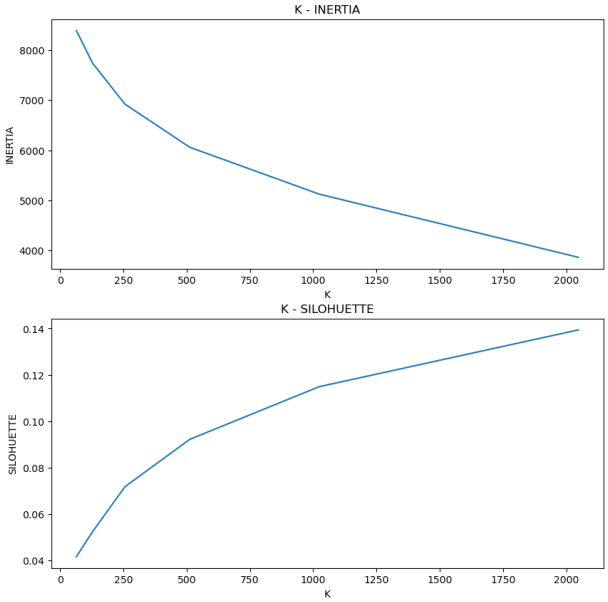

Fig. 8. Inertia and Silhouette Score of the BASENet with k-means

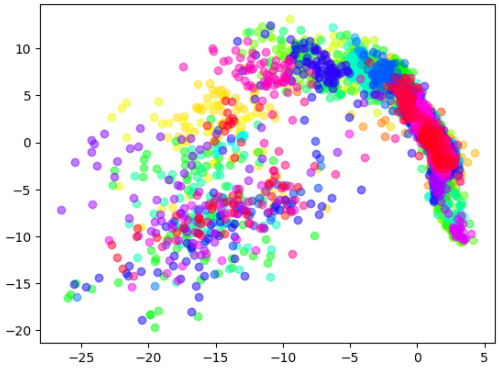

Fig. 9. Extracted features of the dataset plotted in 2d with LDA compression and k-means with K=256

4.3 Future work

As already mentioned before, it was not expected to obtain great results with the BASENet. Indeed, it was implemented with the goal of using a very basic model that is typically utilized for solving the clustering task on unsupervised data in Machine Learning.

The model is so simple that it is rather difficult to think how it could be improved, and it is much more reasonable to create different models that use different techniques, and just to use the BASENet's results as a baseline reference for the other models.

In conclusion, it is important to remark that images represent a very complex and high-dimensional input, and the capacity of the BASENet is not sufficient for extracting relevant information from them.

5. VAENet

5.1 Architecture Overview

The proposed architecture called VAENet is basically a trained Variational Autoencoder model, used to encode the input data (spectrograms), and then the sampled output representation is used for clustering, using both k-means and OPTICS algorithms. Below (**Figure 10**) it is possible to see a "bird's-eye" perspective of the network, explained better and deeper in the following paragraphs.

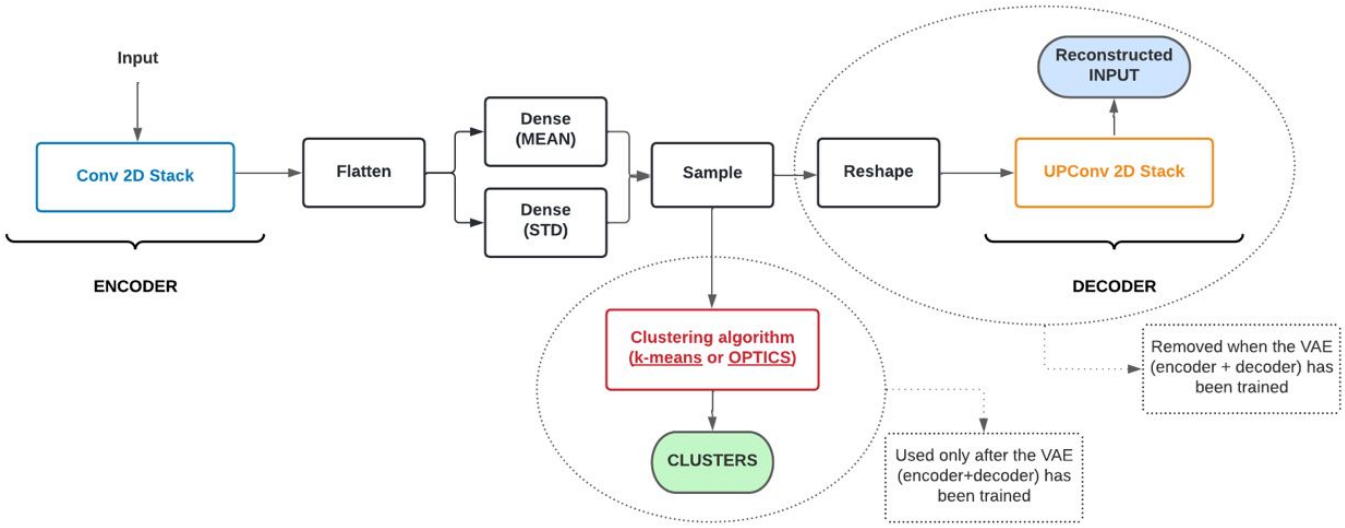

Fig. 10. VAENet overview

5.2 Experiments, results and error analysis

Input RGB images, with dimension 512x64 pixels, are fed into the network composed of five convolutional layers with stride equals to 2. At the end of the convolutional stack there is a strided convolution that performs downscaling. The function of the last convolutional layer is something similar to a max-pooling layer, with the difference that a pooling layer is a fixed operation, while an additional convolution introduces other trainable parameters, learnable by the network. In this way the initial input with dimensions 512x64x3 is reduced to 1x8x256, before being flattened by the following layer. The choice to adopt this solution has been taken considering pros and cons of both alternatives and looking at the existing literature [12] in order to correctly approach the problem.

The pipeline of this network is summarized as follows: first of all, a variational autoencoder is trained in a naive way, compressing the original image in the fixed latent space dimension. After the training, the decoder is discarded, and the encoder is then used to project all the data onto a space with lower dimension (e.g., the latent space). In the end, clustering techniques are applied to encoded data.

The choice of using a variational autoencoder instead of a classical one has been made because the autoencoder has the defect of spatially redistributing the compressed samples, and therefore it is certainly not suitable for clustering. This is because the model inevitably, where possible, tries to overfit so as to reduce the reconstruction loss. While the VAE introduces a sort of regularization during the training. Namely the VAE, in addition to reducing the loss for the reconstruction, it is also optimized so that the latent space has an organization that follows a normal distribution as much as possible. Specifically, there is a precise loss responsible for this regularization which is the Kullback-Leibler divergence, which is a measure that evaluates the distance between a given distribution and a reference one (the Gaussian in this case). The main advantage of VAE is that it does not give a direct representation as in AE, but it produces a latent space with a mean and a variance from which it is possible to sample and then do decoding.

Using this model has not produced performing results, and maybe it has been a bit below expectations. Even tuning the hyperparameters of the model (e.g., number of convolutional filters and the number of units of dense layers) have not sensibly improved the situation. During the training, in fact, the loss value has stopped decreasing quite early, attesting itself at high scores. This has prevented a decent reconstruction of the output with the model trained, and the trials done with input images have resulted in blurred reconstructed ones (see **Figure 15** and **Figure 16**).

Below it is possible to see the best obtained results, and they are clearly depicting a very poor clustering quality. The causes of such low-quality results may be various, including the possibility that maybe a vanilla VAE is not suitable enough to handle inputs like spectrograms, and it needs to be reviewed in the architecture by adding some statistical tricks. As this architecture has been thought only as a comparison, no further experiments and insights have been done about it and thinking about them could be a good starting point for future works.

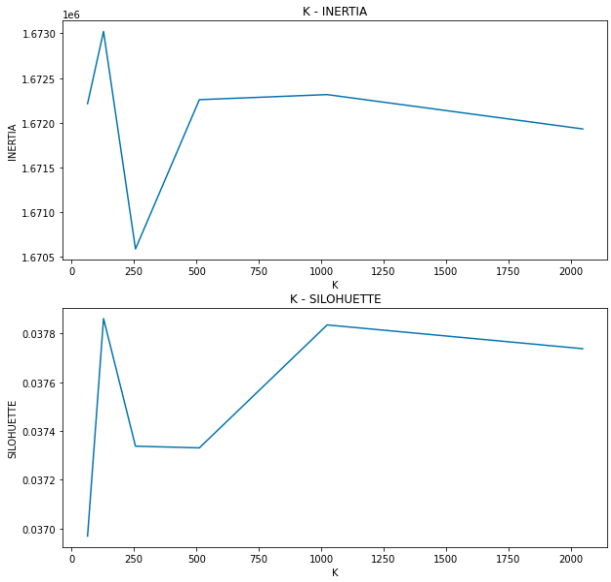

Fig. 11. Inertia and Silhouette Score of the VAENet with k-means

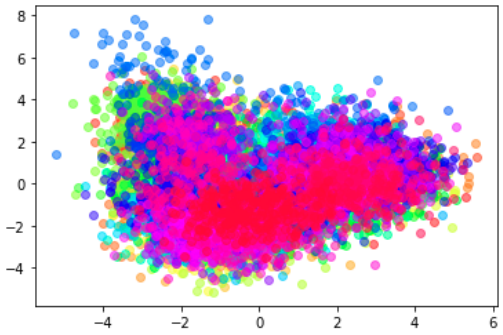

Fig. 12. Extracted features of the dataset plotted in 2d with LDA compression and k-means with K=256

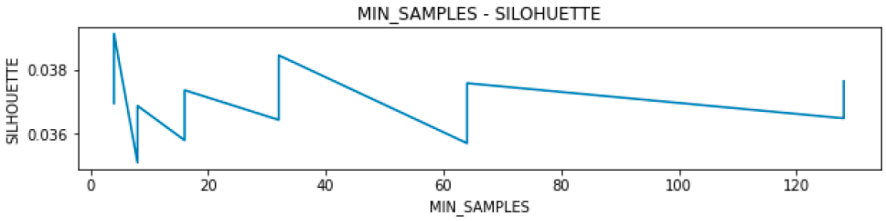

Fig. 13. Silhouette Score of the VAENet with OPTICS

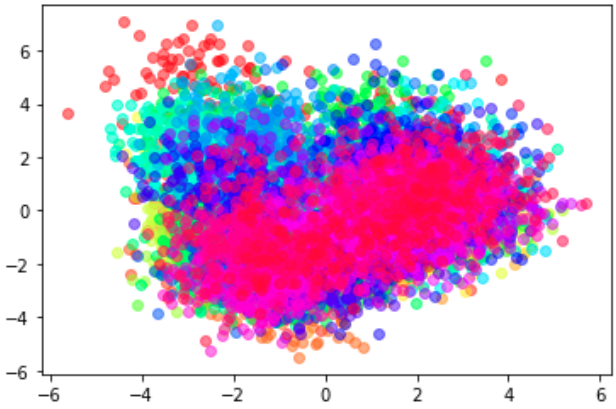

Fig. 14. Extracted features of the dataset plotted in 2d with LDA compression and OPTICS with min_samples=16

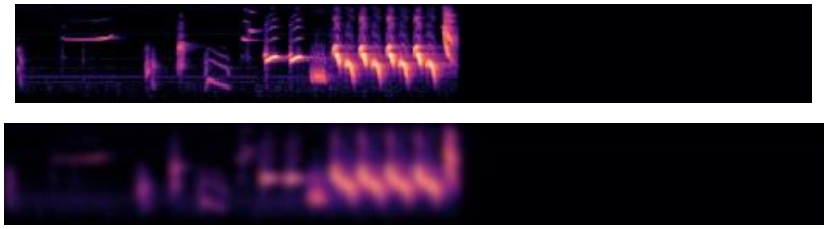

Fig. 15. 1st example of original (TOP) and reconstructed (BOTTOM) images

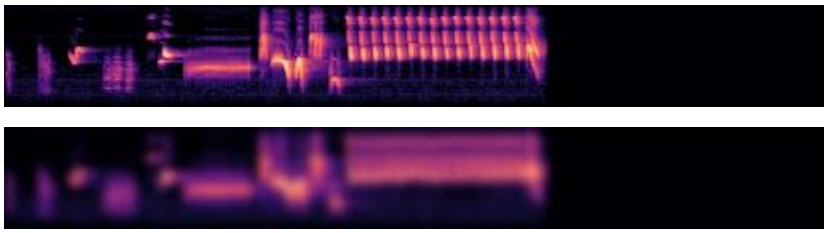

Fig. 16. 2nd example of original (TOP) and reconstructed (BOTTOM) images

5.3 Future work

As for the BASENet, it was not expected to obtain particularly good results with this architecture. Nevertheless, it was undoubtedly supposed to have better results. Therefore, there is surely space for improving this architecture.

For example, the architecture should solve the problem of the blurred reconstructed images. This is a clear symptom of a low model capacity, that should be increased. Additionally, other architectural tricks could be useful for improving the learned representation of the data, such as the pooling operations and residual links, which are very commonly used in models based on CNNs.

6. NSCNet

6.1 Architecture Overview

This architecture, inspired by [2], represents the core of the project. It is and **end-to-end** model that belongs to the family of **Closed-loop multistep Deep Clustering** [13], a group of strategies (among the most successful ones in recent years) characterized by an iterative loop where the clustering and the dimensionality reduction onto a compressed latent space are learnt together.

Diving into the details, the network, as shown in **Figure 17**, is made up of the following components:

1. **Convolutional network (feature extractor)**: this is the backbone of the entire network, and its aim is to compress the input (RGB images) to extract its features. The chosen network for this task has been the EfficienNet-B0 [10], a popular network known for its good performance, and for being light. This component is built with a series of convolutional layers that gradually decrease the dimension of the input until it reaches a dimension of *7x7x1280*; then a global max-pooling operator is applied and followed by flattening, eventually resulting in *1280* features for each single image in input; finally, a linear layer maps those features onto the latent space, whose dimension has been fixed to *512*.
2. **Clustering algorithm (pseudo-labels generator)**: this block takes the entire compressed dataset and applies a clustering algorithm, linking each sample with the corresponding cluster (pseudo-label). The adopted algorithm could be of any kind, but since the classification head requires the number of clusters from the beginning (see below), the common **k-means** algorithm has been implemented. This decision has also allowed a faster training, thanks to the not too high complexity of the chosen clustering algorithm.
3. **Data augmentation (spec augmentation)**: having a quite small dataset, counting a bit more than 10k samples, the data has been passed into the classification pipeline in an augmented form. To be more

precise, each image undergoes three random-based operations, belonging to the Spec augmentation approach:

1. Time warping: the image is warped along the time dimension in a random point
2. Time masking: a window of the time dimension is randomly chosen and masked (signal set to zero)
3. Frequency masking: a window of the frequency dimension is randomly chosen and masked (signal set to zero)

The process is also meant to train the network to compress and classify the images also when some information is missing.

4. **Cluster sampler**: this component is responsible for creating batches (used only at training time) that should be as balanced as possible in terms of labels. The risk is precisely to have batches where a label is dominating, leading to a trivial parametrization in the final classification of the network. Moreover, this risk is sufficiently big, because of a hypothesized strong class imbalance of the dataset (see results and Zipf's law). To avoid this issue, this block packs the data in batches with a uniform sampling between the clusters, allowing (if necessary) the repetition of some samples (that will never be exactly the same thanks to the Spec augmentation).

5. **Classifier**: it is the part of the network made up of a linear layer (with a number of outputs equal to the number of clusters) followed by a softmax layer and a cross-entropy loss, used to train the model to correctly classify each spectrogram with respect to the pseudo-labels generated by the clustering algorithm.

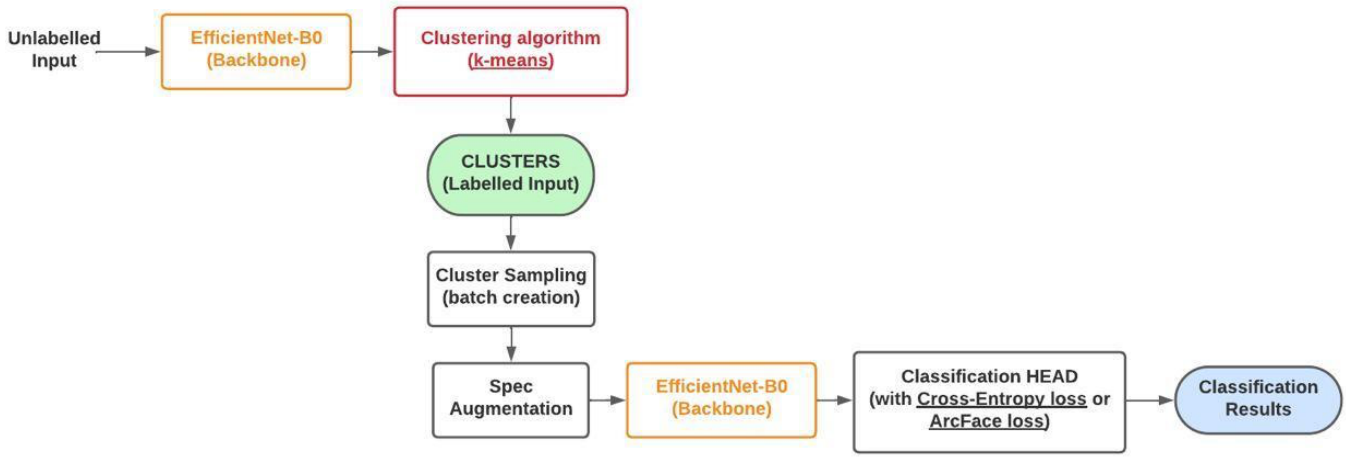

Fig. 17. NSCNet architecture

During the training phase, the model follows this pipeline: before each epoch, the features of the entire dataset are extracted using the EfficientNet-B0, subjected to optional post-processing operations (PCA dimensionality reduction, normalization, whitening, l2-normalization), and sent to the clustering algorithm to get the pseudo-labels (there is no training of the weights or backpropagation in this phase); then the epoch begins and an arbitrary number of batches (set to cover approximately all the samples n times, usually with $n=1$) are packed by the cluster sampler and augmented through the Spec augmentation process; subsequently, the network is trained on them (with backpropagation and weights update) using the EfficientNet-B0 and the Classifier as a classification model that uses the pseudo-labels as the ground-truth.

This mechanism relies on a loop where the model alternatively improves the quality of the clustering and the features extraction, with a theoretical convergence of both [2]. Additionally, the learned representation is already clustering-oriented, differently from the previous architectures.

To control the convergence of the discussed loop, a metric called Normalized Mutual Information (NMI) is monitored:

$$NMI(A;B) = \frac{I(A;B)}{\sqrt{H(A)H(B)}} \tag{3}$$

Where H is the entropy, and I denotes the mutual information:

$$I(A;B) = \sum_{b\in B}\sum_{b\in B} p(a,b)log\left(\frac{p(a,b)}{p(a)p(b)}\right) \tag{4}$$

The NMI score is used to evaluate the level of similarity between the labels assigned in two consecutive epochs: if the NMI remains stable on high values, then the difference between an epoch and the one before is low and the model is converging as expected. The NMI score is also used as a criterion for early stopping, with a patience of 10 epochs without improvement.

6.2 ArcFace Loss

One of the biggest clustering algorithms concerns is represented by the features that are used as its input. Each sample has its set of features, and each sample can be thought of as a point in the latent space, on which the clustering algorithm will be eventually computed.

NSCNet was initially trained using the standard cross-entropy loss, but it was unable to provide good results according to the considered metrics (*section 1.4)*. Therefore, this work tries to use a modification of the cross-entropy loss: the **ArcFace** loss [9].

The **ArcFace** loss' goal is to modify the latent space, so as to have the samples of the same class as near as possible (e.g., high intra-class similarity), while increasing the distance from the samples of the other classes (e.g., small inter-class similarity), as shown in the figure below.

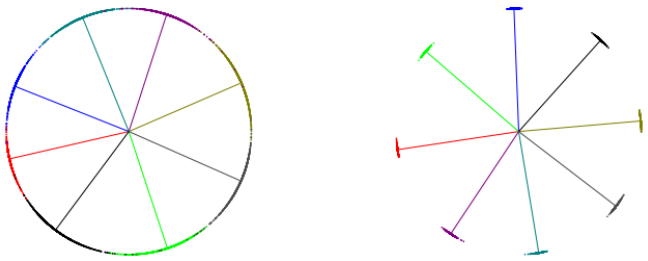

Fig. 18 2D feature examples with 8 possible classes (straight lines).
On the left, the 2D features distribution when the cross-entropy loss is applied.
On the right, the 2D features distribution when the ArcFace loss is used.

The ArcFace loss was initially born for solving the Face Recognition task. Nevertheless, the latent space organization that it provides could be very useful also for satisfying the aims of this project, thus it has been implemented and tested, and -as far as we know- it is probably the first time that it is applied to audio files.

The cross-entropy together with the softmax, a.k.a. *softmax loss* (1), represents the most widely used loss for solving classification tasks:

$$CE = -\frac{1}{N}\sum_{i}^{N} log\left(\frac{e^{W_{y_i}^T x_i}}{\sum_{j=1}^{n} e^{W_j^T x_i}}\right) \tag{5}$$

Where:

- x_i is the feature vector (e.g., the embeddings) of the i[th] sample, belonging to the y_i***-th*** class

- W_j is the ***j-th*** column of the weighting matrix of the last dense layer
- **N** is the number of samples in the dataset
- **n** is the number of classes
- **NOTE:** the bias has been omitted for simplicity

Geometrically, the dot product of two *normalized* vectors can be seen as the product of the magnitudes of the vectors and the cosine of the angle between them. By using equations (6) and (7), it is possible to rewrite the equation (5) and to transform it into the equation (8).

$$e^{W_j^T x_i} = \|W_j\|\|x_i\| \cos\cos\theta_j \tag{6}$$

$$s = \|W_j\|\|x_i\| \tag{7}$$

$$CE = -\frac{1}{N}\sum_{i}^{N} log\left(\frac{e^{scoscos\,\theta_{y_i}}}{e^{scoscos\,\theta_{y_i}} + \sum_{j=1,j\neq y_i}^{n} e^{scoscos\,\theta_j}}\right) \tag{8}$$

Where:

- The scale ***s*** is the first of the two hyper-parameters required by the ArcFace loss. It is a scale factor applied to the logits: the higher the scale, the peakier the logits vector becomes, and also larger gradients are computed during the training.
- W and x must be both L2 normalized, making the predictions basically only to depend on the angle θ between the two. Additionally, the normalization step tries to force the embeddings x and each column of the weighting matrix W (e.g., a class) to lay on a hypersphere, as shown in **Figure 18**.

Once the cross-entropy is written as (8), then can be easily added the second ArcFace loss hyper-parameter, called **margin *m***:

$$CE = -\frac{1}{N}\sum_{i}^{N} log\left(\frac{e^{scoscos\,(\theta_{y_i}+m)}}{e^{scoscos\,(\theta_{y_i}+m)} + \sum_{j=1,j\neq y_i}^{n} e^{scoscos\,\theta_j}}\right) \tag{9}$$

Adding ***m*** always penalizes the predictions that should be correct, and basically this forces the loss to be never satisfied of the results, because it always increases the angle θ that the model simultaneously tries to minimize during the training. The higher the margin, the higher should be the class separation.

The ArcFace paper [9] demonstrates how this process is helpful for creating a better latent space representation, as shown in **Figure 5**; the latent space become not only linearly separable (e.g., the goal of the cross-entropy), but also well-defined clusters are created.

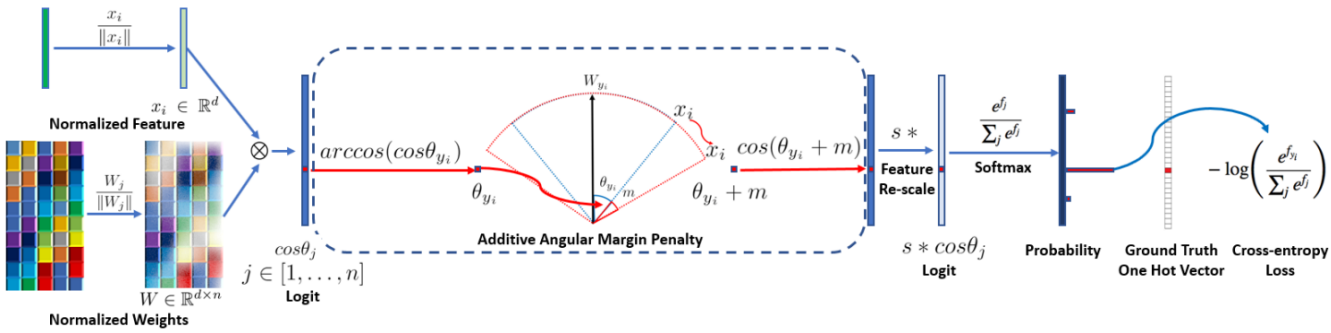

Fig. 19. A graphical view of how the ArcFace is applied

6.3 Experiments, results and error analysis

Experiments have been run using *Adam* optimizer with a learning rate of 10-4, batch size of *32* and *50* epochs. The activation function following the linear layer right after the EfficientNet-B0 has been set to "*Leaky*

ReLU", a variant of the standard "*ReLU*" that avoided the *"**dying ReLU**"* effect, encountered during the first tests.

All the trainings tested a different combination of the following possible parameters:

- **Normalization** before clustering: *True/False*
- **PCA** dimensionality reduction before clustering: *128, 256*
- **Whitening** before clustering: *True/False*
- **L2-normalization** before clustering: *True/False*
- **ArcFace loss**: *True/False*
- **Epoch's length:** *1, 3*
- **Number of clusters** (K): *64, 128, 256, 512, 1024, 2048.* Note that, after having determined the most promising values of K, other values have been searched in the interval, using a binary search strategy.

Experiments have led to the most performing configuration of the hyperparameters:

- Contrary to the expectations, the most performing setups had both normalization and whitening set to False
- Compressing the data with a PCA set to 256 dimensions has allowed to keep more information, without being victim of the curse of the dimensionality when applying the k-means
- The ArcFace loss has proved to be highly effective for this task, making the training much more stable and improving results sensibly (silhouette score around 50-60% better)
- Before assigning the new pseudo-labels, an attempt has been made by increasing the length of each epoch, with the goal of trying to reduce as much as possible the loss. This has been obtained by passing three times the number of samples before assigning the new-pseudo labels. Nevertheless, this has not brought any improvements and, oppositely, it has mined the convergence of the entire model, with a latent space changing too much between two consecutives epochs and destabilizing clustering labels. Consequently, the NMI had some troubles in stabilizing. In the end, the length of the epochs has been set to 1, as default.

The following images show the scores coming from the tuned model:

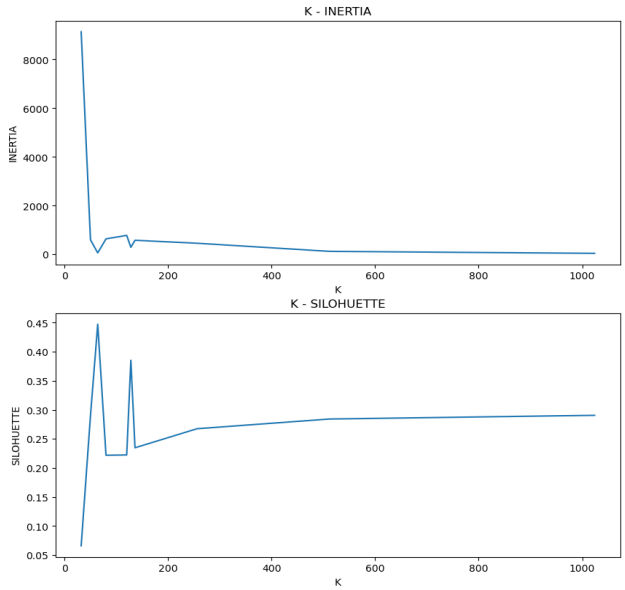

Fig. 20. Inertia and Silhouette Score of the NSCNet with k-means

Inertia and silhouette functions showed interesting local optima for *K=64* and *K=128*. As an example, the best results, obtained with *K=64*, are shown in **Figure 21**, **22** and **23**. The silhouette score is *0.45* and it seems to be quite good, also bearing in mind the limits of the dataset.

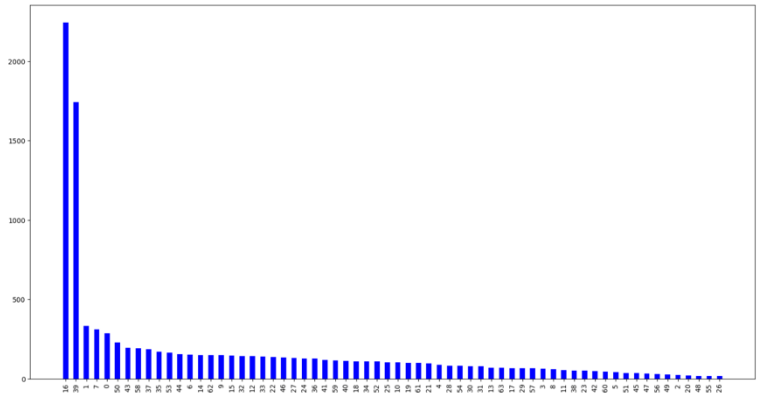

Fig. 21. Ordered distribution of the clusters; a power-law emerges (see Zipf's law below)

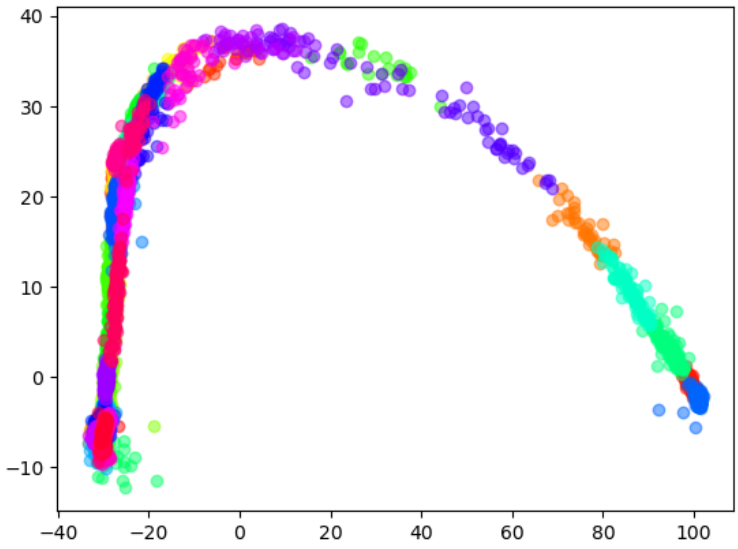

Fig. 22. Extracted features of the dataset plotted in 2D with LDA compression

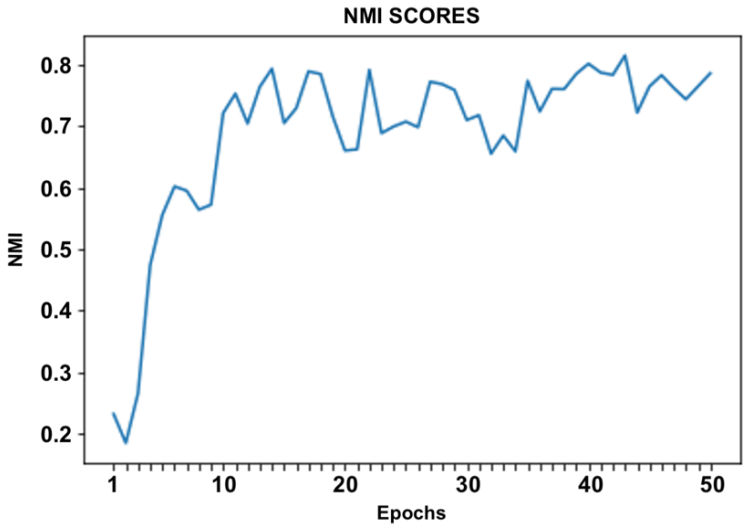

Fig. 23. The NMI score becomes more stable around high values, showing convergence

6.4 Future work

Even if this architecture outperforms the baseline based on PCA and the VAE model, there is room for improvement:

- The EfficientNet-B0 is probably too complex for the given small dataset and a more lightweight model could be tested
- ArcFace's margin and scale parameters should be fine-tuned. Scale has been set empirically, following [9]; while margin should depend on the number of classes (not known a priori), so it has also been set as indicated in [9]. Both deserve a deeper investigation
- Training could be pushed further beyond the 50 epochs (for example until early stopping for NMI convergence is reached)

In general, the architecture is limited by the nature of the problem (unknown number of clusters), the dataset (few samples for too many clusters and a strong class imbalance – see Zipf's law section) and the inner limits of the current state-of-the-art deep clustering techniques.

6.5 Zipf's law and the distribution of clusters

By observing the distributions of clusters ordered by frequency in the labelled dataset, it can be noticed, almost independently from the chosen number of clusters, that they follow a power-law. Apparently, there are certain types of songs that are used far more often than others.

An interesting phenomenon, hypothesized to be explanatory for this fact, is the empirical law called Zipf's law, a specific kind of power-law where the rank of a class (of the ordered distribution) has a direct inverse exponential relation with its frequency, following the formula:

$$f(k;s;N) = \frac{1}{k^s \sum_{n=1}^{N}(\frac{1}{n^s})} \tag{10}$$

Where *k* is the rank of the class, *N* is the number of classes and *s* is a parameter that controls how much rapidly the distribution falls down. This law affects several natural, biological and linguistic aspects of life. For examples, it appears in all the existing languages, where the frequency of each word in the vocabulary depends exactly on the formula above, with *s* close to 1.

A possible explanation for the obtained results is that nightingale songs act in the same way and, as for human languages and other natural phenomena, they follow a pattern similar to **Pareto's principle**.

Trying to prove the hypothesis of the Zipf's law, the ordered distributions produced by the most promising results (all coming from the NSCNet) have been analyzed, experimenting also with different numbers of clusters. The following steps have been proposed by [14] as a way to compute the probability of a distribution of coming from a generic power-law.

1. First, to find out the most reasonable value of *s,* to have the best possible approximation of a Zipfian distribution, the **MLE** (Maximum Likelihood Estimation) method has been used:
   Being $\sum_{n=1}^{N}(\frac{1}{n^s}) = H_s(n)$ the generalized harmonic number, the log-probability of a class i is:

   $$log\ log\left(\frac{i^{-s}}{H_s(n)}\right) = -s * log\ log\ (i) - log\ (H_s(n)) \tag{11}$$

   For independent data summarized by their frequencies, the global probability is the product of the individual probabilities, leading to:

   $$L(s) = -s * \sum_{i=1}^{n} f_i\ log\ log\ (i) - \left(\sum_{i=1}^{n} f_i\right) * log\ (H_s(n)) \tag{12}$$

   With f_i being the frequency of the i^{th}cluster. This is a function of *s* that represents the probability of the Zipfian with respect to the data, thus it is the log-likelihood. By maximizing this function, *s* has been estimated case by case.

2. Having identified the most probable value for *s*, the corresponding expected Zipf's distribution has been built.

3. To check the hypothesis of the distribution of clusters following the Zipf's law, three methods have been applied:

    a. Real frequencies and expected ones have been compared by means of a plot
    b. The distributions have been compared using the Kolmogorov-Smirnov test expecting, as suggested by [14], a p-value bigger than 0.1
    c. The distributions have been compared using the Chi-squared test.

By the end of this process, results have shown enough proofs in favor of the discussed hypothesis, with the distributions reasonably following Zipf's law in most of the cases. Remarkably, for *K=64* (the value with the highest silhouette score) it has been achieved the best approximation of a Zipfian, with a *p-value* of *0.30* resulting from the Kolmogorov-Smirnov test and a value of *s* very close to 1 (*0.98*), as for human languages (see **Figure 24** and **Figure 25**).

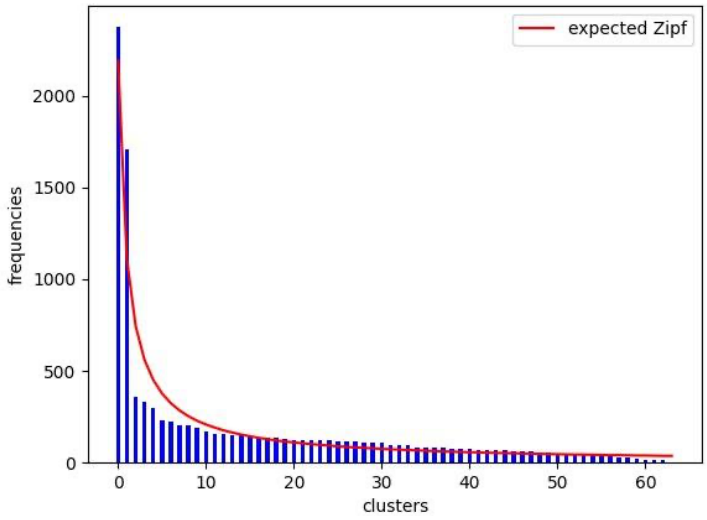

Fig. 24. Comparison with the expected Zipf's law and the distribution obtained with K=64

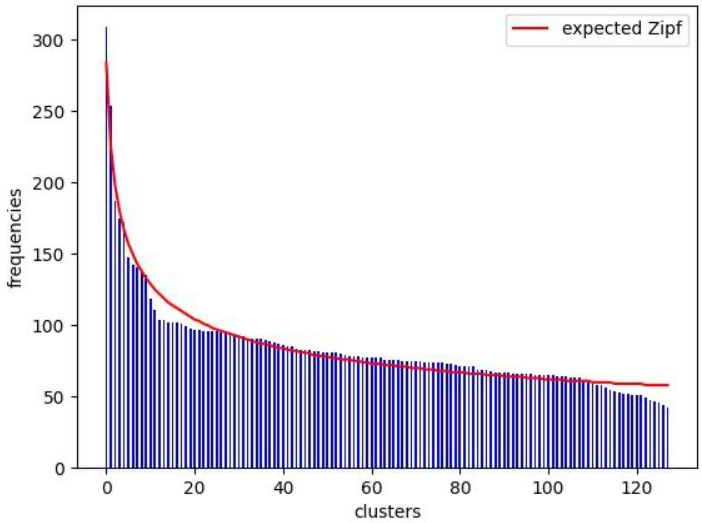

Fig. 25. Comparison with the expected Zipf's law and the distribution obtained with K=128

There are some hypotheses and conclusions that can be drawn from this point:

- The deep clustering approach looks to be strengthened, producing a pattern in the distribution that is very frequent in real life, hence more realistic and reliable. It is also worth noting that this does not hold for results produced by other models, such as the proposed baseline or the VAE network, showing that the law does not appear randomly.
- Nightingale songs may act as human languages, with a restricted number of them that are frequently repeated (possibly in response to common situations); and many of them that are rarely used or even known, a part of which may come from local dialects.
- The dataset has a high probably of being heavily affected by a class imbalance, with most of the clusters barely represented, making the task harder due to issues such as overfitting toward the most common clusters (partially handled by the custom sampler and the spec augmentation of the NSCNet), or a difficulty in learning a good representation for the rarest ones.
- If the real number of possible songs is high (e.g., around 3000, as estimated by some studies), the dataset in analysis may contain an insufficient number of samples to cover all of them: there is a considerable chance of many missing clusters.
- The influence of the Zip's law may also be the cause for the achieved results, where the relatively high silhouette score obtained for quite a low numbers of clusters contradicts the expectation of the presence of hundreds or even thousands of songs. Precisely, the silhouette score computed for each cluster depends on the size (number of elements contained) of the cluster itself: a good clustering job done on the (very few) biggest clusters leads to a high silhouette score. This means that training the network on a low number of clusters (K), makes it possible to focus more on the biggest ones (the smallest ones will be probably merged with them following the "preferential attachment" principle), with better results; while a large K forces the model to put the same effort in finding a good representation for samples belonging to both big and small clusters, with a higher error rate for the big ones, compared to the previous case, leading to worse results. Of course, this is true with some limitations, since a value such as 32 would be too small and many big clusters would be merged, with a poor score (as experimented). The natural implication would be that the silhouette score should not be taken as a metric for evaluating the best universal K, but as a way to estimate how well the architecture can correctly label the samples of the **top-K clusters**. In this sense, the best model has achieved good results with the top 64-128 types of songs, the most frequent ones.

Conclusions

This work represents the first research on the application of deep clustering techniques aiming at distinguishing between the different songs of the nightingales, while estimating the size of their repertoire. To achieve this goal, some of the most recent and effective strategies have been adopted, such as: Mel-spectrogram encoding, end-to-end model based on pseudo-labels, deep extraction of the features by means of the powerful but still lightweight EfficientNet-B0, Spec augmentation and many more.

Moreover, these techniques have been combined with some new ideas, taken from different domains. Particularly, the insertion of the ArcFace loss represents a novelty that has brought sensible improvements.

In order to be able to make some comparisons, for this and future works, the main architecture has been put side by side with two other models, a baseline and a more sophisticated approach from older literature, further confirming its strength. Thus, this work has also tried to set a good starting point for future research, fixing some interesting results as the first reference point.

In particular, tests with the fine-tuned model have shown some evidence that the repertoire may follow Zipf's law, meaning that most of the songs are very rare, while a few of them are extremely common, with a possible

analogy with human languages. This point has also highlighted a class imbalance of the dataset, a relevant difficulty in identifying all the possible different songs, and the importance of training the network to recognize the most frequent ones, with reasonably good results achieved on this last consideration.

References

1. Dai, W., et al.: Very Deep Convolutional Neural Networks for Raw Waveforms (2016)
2. Ghosh, S., et al.: DECAR: Deep Clustering for learning general-purpose Audio Representations (2021)
3. Guo, W., et al: Deep Embedded K-Means Clustering (2021)
4. Palanisami, K., et al.: Rethinking CNN Models for Audio Classification (2020)
5. Nagrani, A., et al.: Attention Bottlenecks for Multimodal Fusion (2021)
6. Chang, S.: Deep clustering with fusion autoencoder (2022)
7. Ankerst, M., et al.: OPTICS: Ordering Points To Identify the Clustering Structure (1999)
8. Arthur, D., et al: k-means++: The Advantages of Careful Seeding
9. Deng, J., et al.: ArcFace: Additive Angular Margin Loss for Deep Face Recognition (2019)
10. Tan, M., et al.: EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks (2020)
11. McFee, B., et al: librosa: Audio and Music Signal Analysis in Python (2015)
12. Springenberg, J. T., et al.: Striving for simplicity: The all convolutional net (2014)
13. Nutacchi, G.C., et al.: An Introduction to Deep Clustering (2019)
14. Clauset, A., et al.: POWER-LAW DISTRIBUTIONS IN EMPIRICAL DATA (2009)